© SANS Institute 2024
https://linktr.ee/offsecexam

670.1

Windows Tool Development

Tamarisk OffsecExam Exam solution

© SANS Institute 2024
© 2024 Jonathan Reiter. All rights reserved to Jonathan Reiter and/or SANS Institute.
https://linktr.ee/offsecexam

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE (“SANS INSTITUTE”) FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this CLA. Courseware means all printed materials, including course books and lab workbooks, slides or notes, as well as any digital or other media, audio and video recordings, virtual machines, software, technology, or data sets distributed by SANS Institute to User for use in the SANS Institute course associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and User and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium, whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. User may not sell, rent, lease, trade, share, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute. Additionally, User may not upload, submit, or otherwise transmit Courseware to any artificial intelligence system, platform, or service for any purpose, regardless of whether the intended use is commercial, educational, or personal, without the express written consent of SANS Institute. User agrees that the failure to abide by this provision would cause irreparable harm to SANS Institute that is impossible to quantify. User therefore agrees to a base liquidated damages amount of $5000.00 USD per item of Courseware infringed upon or fraction thereof. In addition, the base liquidated damages amount shall be doubled for any Courseware less than a year old as a reasonable estimation of the anticipated or actual harm caused by User’s breach of the CLA. Both parties acknowledge and agree that the stipulated amount of liquidated damages is not intended as a penalty, but as a reasonable estimate of damages suffered by SANS Institute due to User’s breach of the CLA.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. A written amendment or addendum to this CLA that is executed by SANS Institute and User may accompany this Courseware.

SANS Institute may suspend and/or terminate User’s access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User, (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

The Apple® logo and any names of Apple products displayed or discussed in this book are registered trademarks of Apple, Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

Governing Law: This CLA shall be governed by the laws of the State of Maryland, USA.

Courseware licensed to User under this CLA may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department (“OFAC”), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce (“BIS”). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulation. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

All reference links are operational in the browser-based delivery of the electronic workbook.

© SANS Institute 2024

https://linktr.ee/offsecexam

SEC670.1 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control

SANS

Windows Tool Development

© 2024 Jonathan Reiter | All Rights Reserved | Version J01_05

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control: 670.1
Welcome to Section 1 of 670. In this section, we will explore what it's like to develop tools for the Windows platform.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (1)

	Page
Course Overview	4
Developing Offensive Tools	17
Developing Defensive Tools	24
Lab 1.1: PE-sieve	29
Lab 1.2: ProcMon	31
Setting Up Your Development Environment	34
Windows DLLs	50
Lab 1.3: HelloDLL	78
Windows Data Types	85
Call Me Maybe	110
Lab 1.4: Call Me Maybe	120
SAL Annotations	129

This page intentionally left blank.

Vanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (2)	Page
Lab 1.5: Safer with SAL	145
Windows API	155
Lab 1.6: CreateFile	187
Bootcamp	197
Lab 1.7: Can'tHandleIt	199
Lab 1.8: RegWalker	200
Lab 1.9: It's Me, WinDbg	201
Lab 1.10: ShadowCraft	202

Tamarisk OffSecExam Exam solution

This page intentionally left blank.

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

In this module, we will discuss at a high level what the course will be covering.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

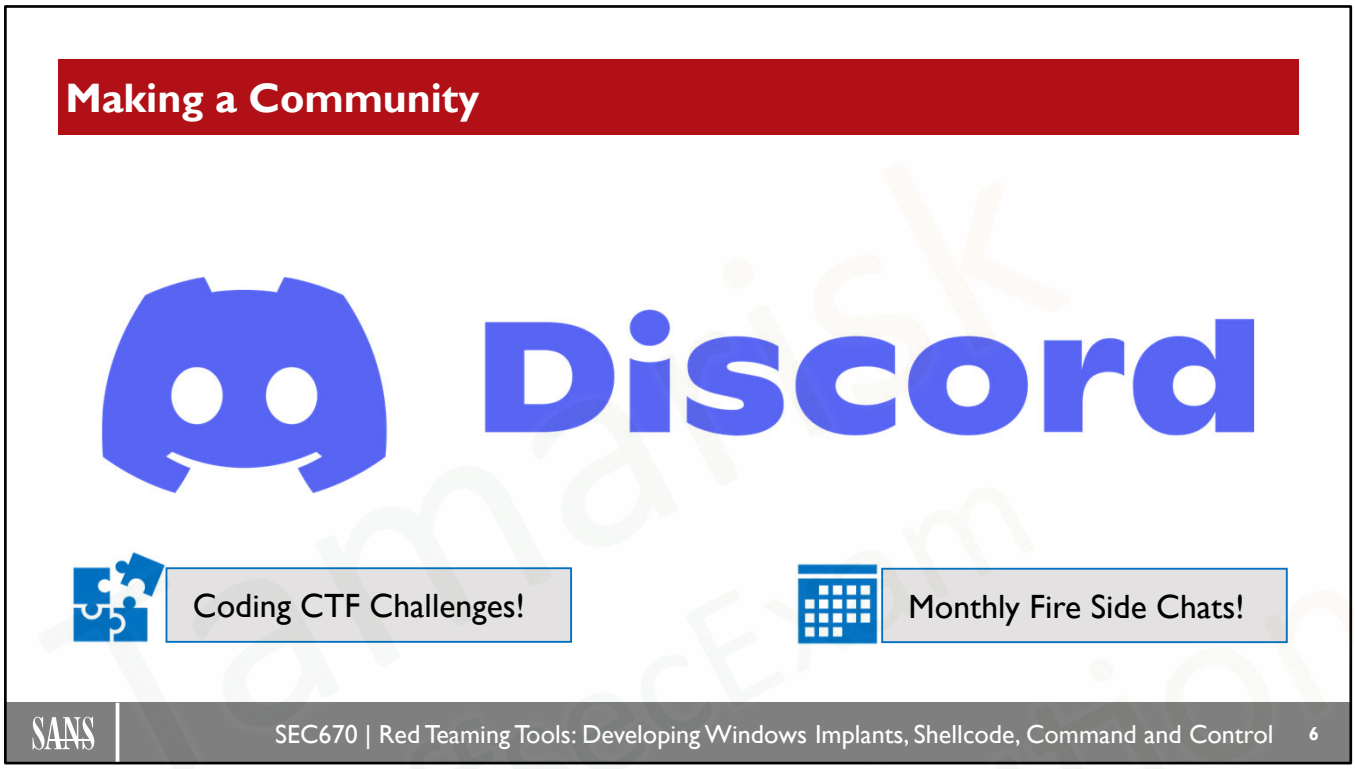
- Discuss what tool development is
- Determine what surveying the land means
- Figure out how to carry out operational actions
- Discover various persistence methods
- Enhance your implant

Objectives

The objectives for this course are straightforward. Each section will be talked about at a high level and then during that section's content, we will take a deep dive into what each one involves. We start off discussing tool development, what it is, how to do it, and different perspectives. Next, we will discuss surveying the land and how to develop custom recon tools. During operational actions, we will discuss what can be done after initial access is obtained: actions like ruining or corrupting data on the target, exfiltration, killing processes, etc. We will also go over several ways to keep your access to a target system. Finally, we will discuss how we can enhance the implant.

© SANS Institute 2024

https://linktr.ee/offsecexam



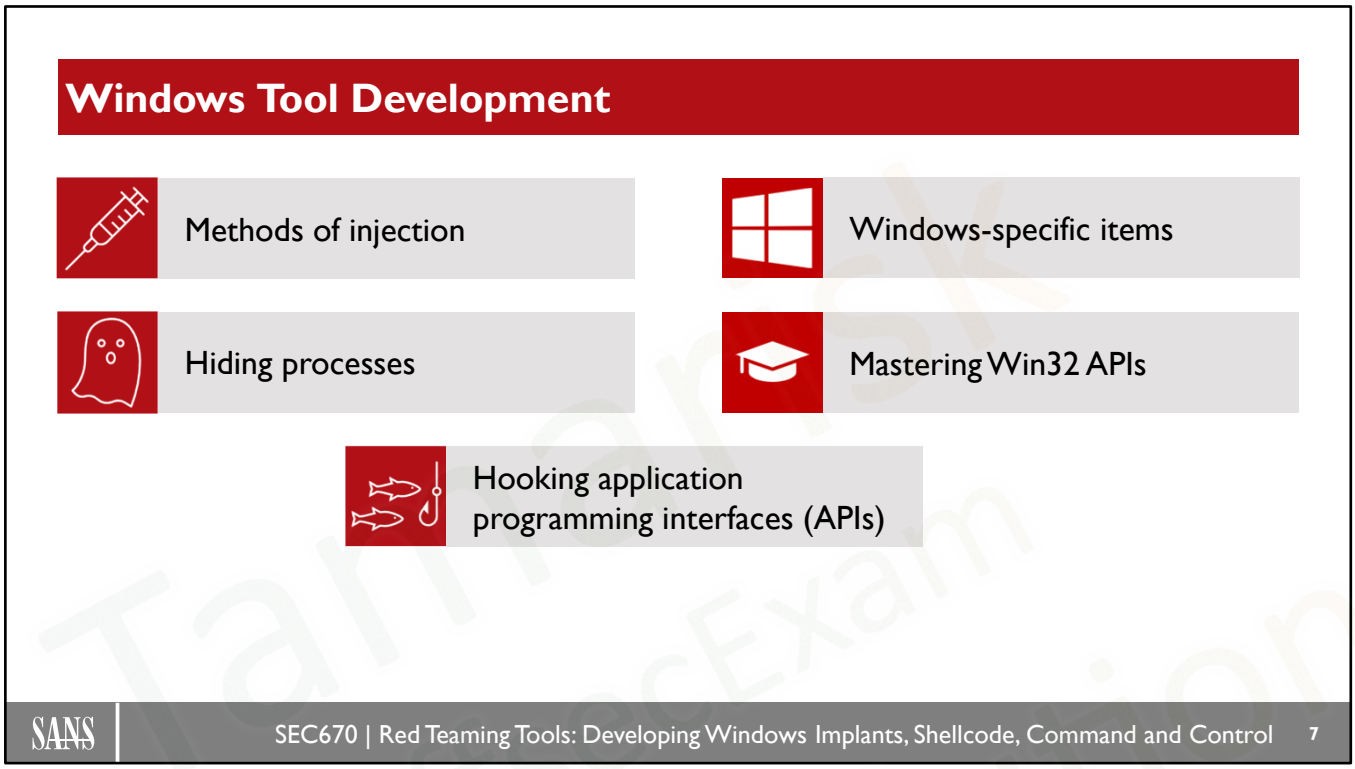
Objectives

This course has a custom Discord server, a community solely for Windows developers and those aspiring to become one. The server is open to anyone, but mainly created for students of this course. Private channels and special roles have been made only for you as a way for us to give back to you beyond the class. Please join the existing community of SEC670 Alum when you get a moment. See you there!

Reference:
https://discord.gg/offensivewindowsdev

© SANS Institute 2024

https://linktr.ee/offsecexam



Windows Tool Development

Windows tool development is more than just creating a console application that prints out "Hello, World!" Instead, we will focus on becoming familiar with Windows APIs that will leverage greater offensive capabilities for your tooling. In the world of offensive tools, it is hard to "see" the effectiveness of your tool. In fact, if you are really good at what you do, your tool will never be seen. Not really seeing a tool in action makes demos somewhat boring since there is no fancy GUI application that shows something taking place. Regardless, here is a short list of some items you will be able to take away from this course.

- Becoming familiar with Windows-specific items
- Mastering the Win32 APIs that leverage offensive ops
- Understanding various methods of injection
- Hooking APIs via various techniques
- Learning how to conduct research and create new techniques

© SANS Institute 2024

https://linktr.ee/offsecexam

Getting to Know Your Target



What happens after you gain initial access to the target?

Typically, you are not the one who is creating the exploit that gives initial access to the operator. You are the one creating the tool that gets dropped after initial access. You aid in assisting the operator in determining the purpose of the target. Arguably, one of the best methods for accomplishing such a task is developing a host survey tool.

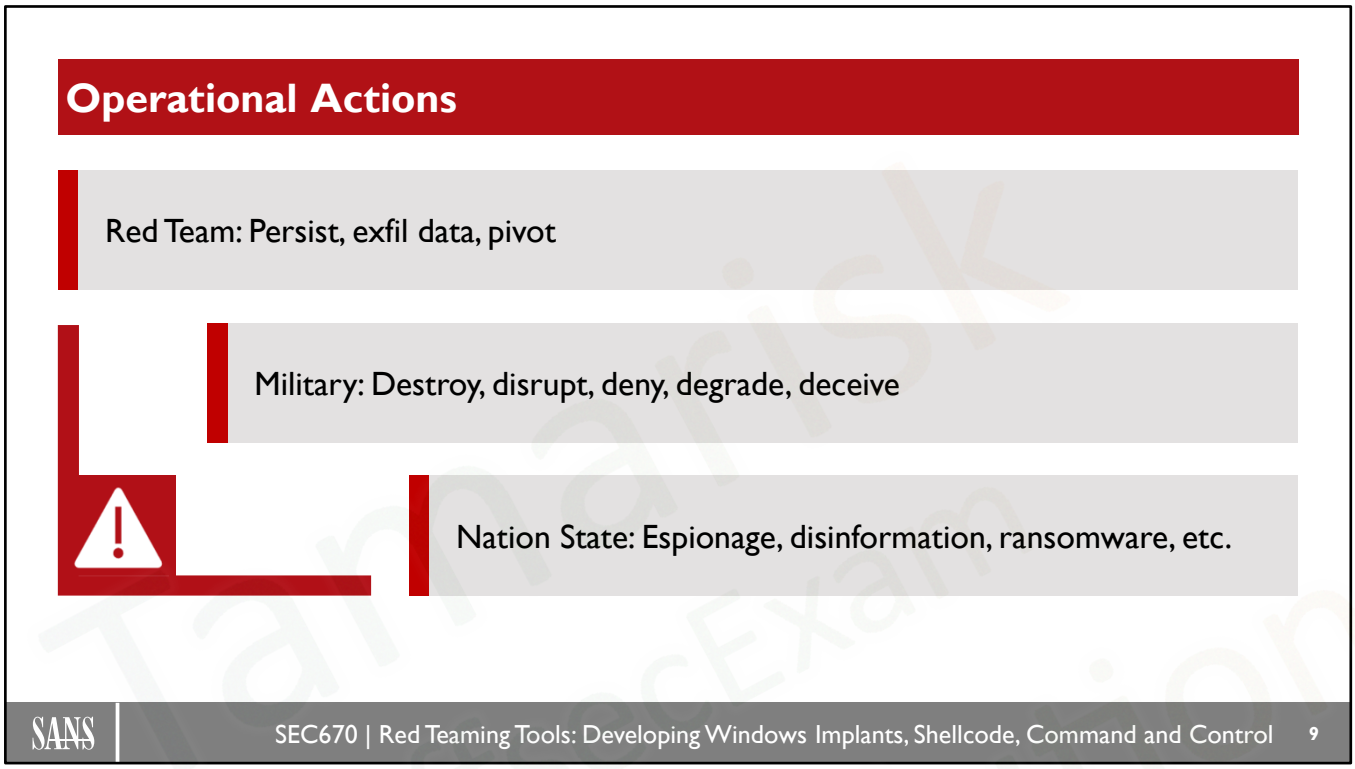
Getting to Know Your Target

This course is not aimed at gaining that initial access (that happens in SEC560), but rather what custom tools can be made to leverage that access. Since you are the developer, what actions are taken next are nearly limitless. You are no longer going to be bound by a current tool or C2 framework and you will not have to say, "I wish this tool allowed me to do this," or "I wish someone would create something that did this." Guess what—you are going to become that person who will create that missing thing. If you want a tool or a feature that enumerates specific registry keys, you can create that feature.

Perhaps one of the most important first tasks you perform with your initial access is determining the purpose of your target. Gathering as much information as possible will enable you to make a very accurate decision, and perhaps the most important decision to make from the gathered information is, "Do you continue?" Perhaps it would be better to clean off the target as best you can because you saw something in your gathered information that indicated the target was watched like a hawk or that the sysadmin was very tech savvy. Maybe you saw the installation of a new AV product you did not expect to see? The list can go on and on but in the end, creating and throwing down a host survey tool is a great first action.

© SANS Institute 2024

https://linktr.ee/offsecexam



Operational Actions

After you have gotten to know your target a bit more, your objectives can be very different depending on the work role you are currently filling. If you are developing tools for a red team, you are probably not creating and releasing ransomware seeking out a payday. Instead, you might be working with the blue team to create a custom tool that is designed to test specific detections, which leads into purple teaming. If you are developing tools for your nation's military, then you probably have the authority to create tools that can destroy data on your target. The military typically has certain authorities granted by the government that authorize what operations can and cannot be executed, like dropping a missile on an ammo depot or wiping the computer of an enemy nation-state operator. If you are some rogue group actor or enemy nation state, then the sky seems to be the limit with what can be done. Countries and companies have suffered the effects of ransomware, espionage, data theft, and more.

© SANS Institute 2024

https://linktr.ee/offsecexam

Persistence: Die Another Day



Problem: Persistence is not always necessary. When it is, what is the best way to establish persistence?

Simple or complex? The chosen method depends on your target and objective.

Persistence: Die Another Day

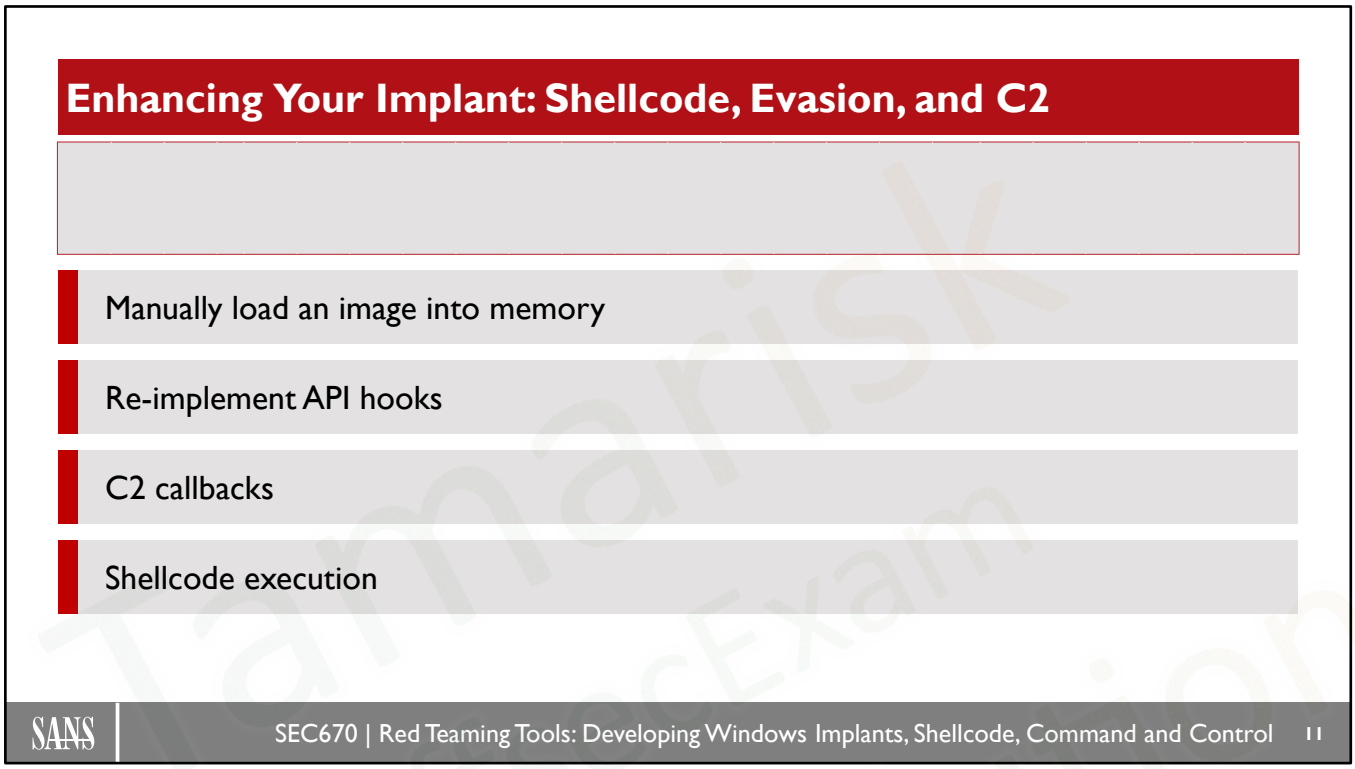
When a decision has been made to maintain access to a target, a red team operator or other operator position might want a few options for how to persist. An important question a red team operator should answer either before or during an op is whether persistence is required to successfully complete your objective(s). There is risk with establishing a foothold and that risk must be weighed against your target, your objective, and being caught. A lot of times, persistence can require pulling down another tool, your precious implant that your team of developers worked on for months on end. Are you okay with giving up that tool just to persist on that one target? If not, then complete your op and clean off. If you are, then determine what persistence method will work for your op and install your implant.

There is more than one method for persisting on your target and some methods might get you caught faster than others. The method chosen should match up with how important that target is to your op. Perhaps a simple modification to a Registry Run key is enough because nothing is in place to monitor it. Perhaps you need to be more creative and stealthier because the admin of that box is savvy and has tools in place to watch out for common locations like Run keys.

In the end, the lowest hanging fruit wins. Meaning, you should go after the easiest method that you know you can get away with on target.

© SANS Institute 2024

https://linktr.ee/offsecexam

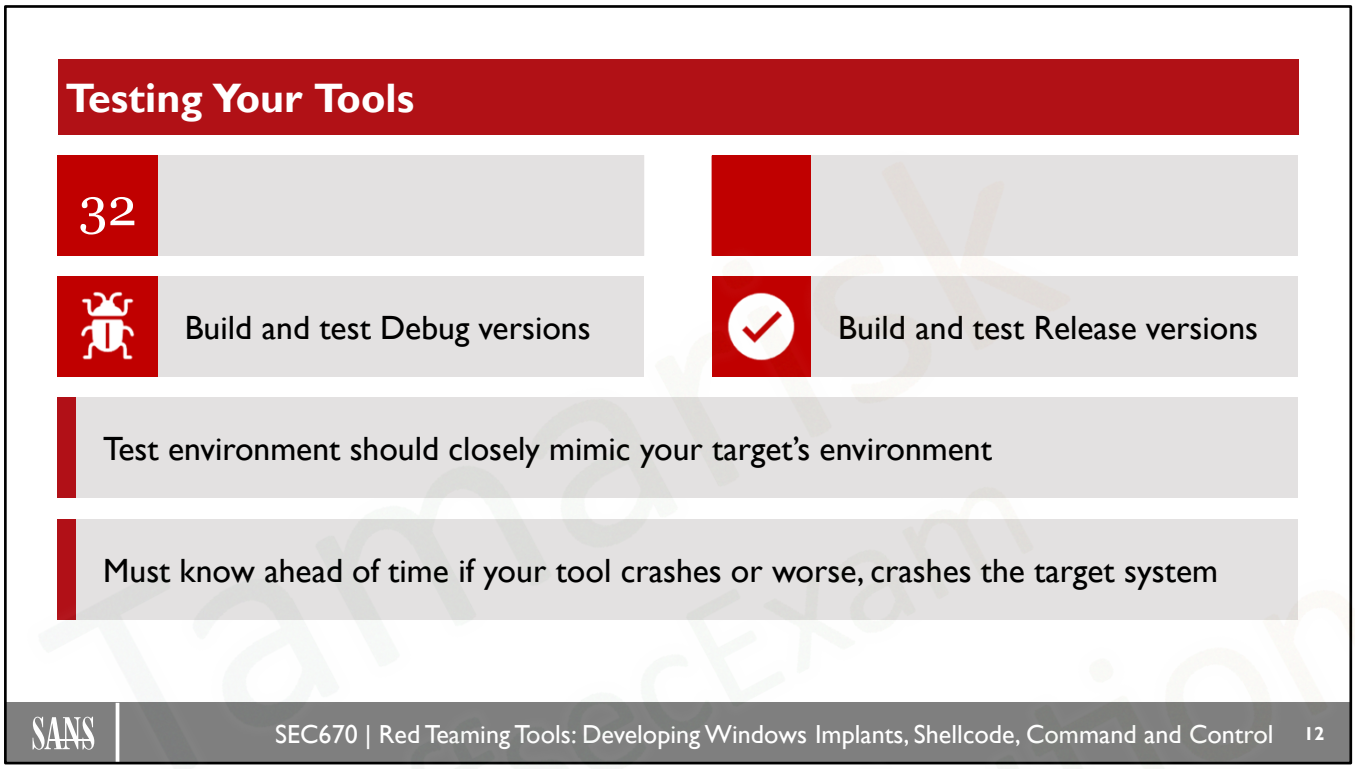


Enhancing Your Implant: Shellcode, Evasion, and C2
Sometimes you will be forced to produce a more advanced tool to get the job done. These should be a last resort option because, depending on what capability you are developing, it can take a while to develop and release. Plus, do you want to spend a few months developing an advanced capability for an easy to knock over target? If there isn't a single AV product installed and the sysadmins are lazy, then why bother pulling down your most prized possession?

Many of the advanced techniques will require intimate knowledge of the PE32 and PE32+ format, process virtual address space, the registry, OS internals, etc. Such mastery comes after years of tool development and debugging undocumented APIs. Also, not much is going to happen if you cannot control your implant and give it taskings. For tasking to happen, it must have the ability to communicate out of the target network and to your C2 infrastructure. Also, having the ability to execute custom shellcode payloads is a great feature to implement.

© SANS Institute 2024

https://linktr.ee/offsecexam



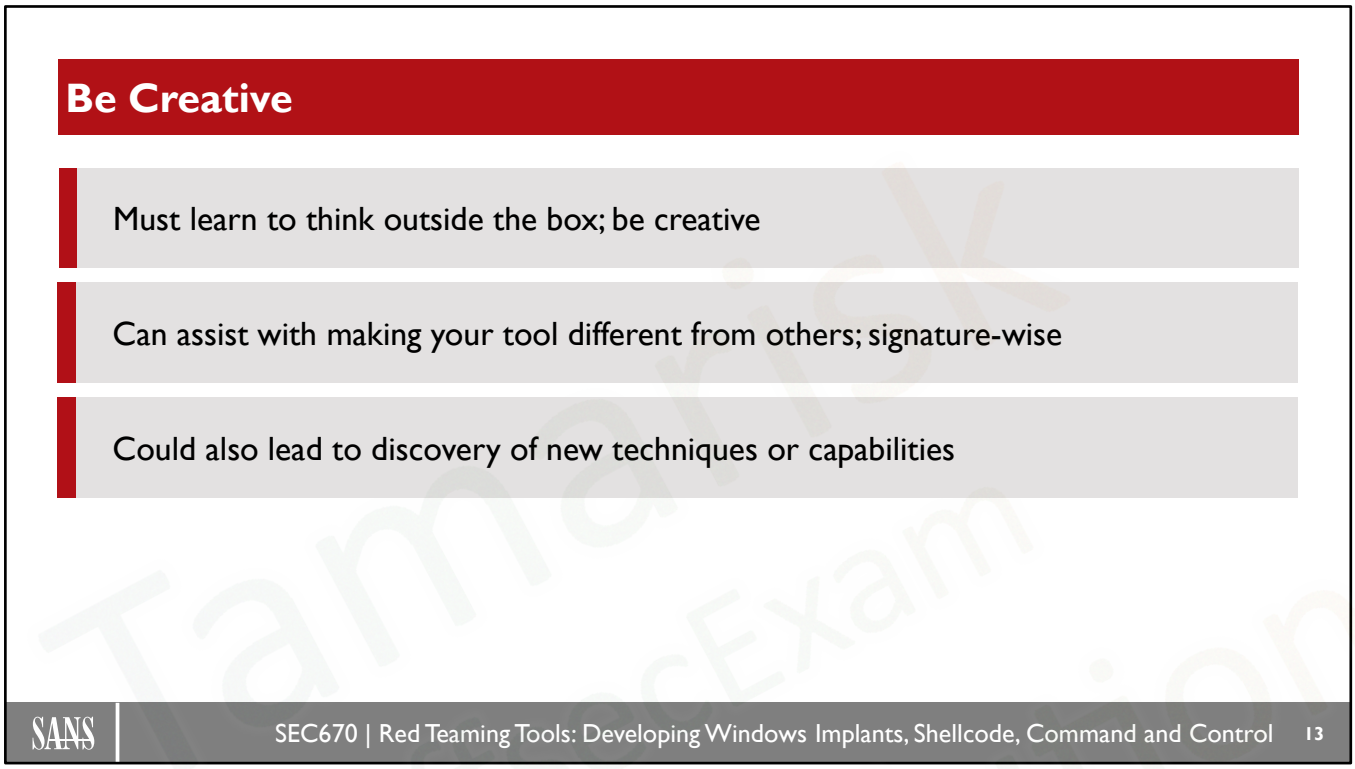
Testing Your Tools

Once your tool is finally developed and ready for operational use, you must resist the urge to use it anywhere and everywhere you can. One of the first things you should do before operational deployment is test it. Proper testing is extremely important if time permits because there are rare conditions that require modifications to already existing tools on the fly. You should have the skillset of being able to tear down your code too after compilation, meaning disassembly. Reversing your tool and stepping it through with a debugger will be not only valuable for fixing bugs but also for verifying your code. Did the compiler do what you were wanting it to do, or do you need to pass it some flag options to change things a bit? You can change certain compiler and linker options to shrink the overall size of the binary. You could also eliminate the dependency on the C runtime along with other dependencies. Eliminating external references, or XREFs as we call them, can aid in making your code position independent. Position independent code, or PIC, is something that we will dive into at the end of the course.

Another item you might want to take into consideration would be easily identifiable strings. Strings are always the easiest item to find thanks to tools like *strings.exe* from the Sysinternals Suite of tools, but they can also give away too much about what your tool is designed to do on target. If you can eliminate them altogether that would be best—the least you can do is obfuscate and/or encrypt your strings. Section 5 will lightly touch on how you can cut down on strings in your compiled binary.

© SANS Institute 2024

https://linktr.ee/offsecexam



Be Creative

Tool development can be greatly limited by your creativity and/or your knowledge of operating system internals. It is great to ask yourself questions like, “What happens if I implement this?” or “What if I trigger this event and do this?” Questions like that and many others are what pull out the creative side of your brain. This is how new capabilities are developed as well as how new vulnerabilities can be found. Creativeness, or newly found ideas, is not the best place for production code that is going to be used for a high value op. Explore and test what you found to explore and discover any edge cases.

One person online who does an amazing job at showing creative thinking is x86matthew. The URL for his blog post is down below, and in his posts, he will explore how to do basic, everyday tasks, like writing to process memory, in a different way. There are other, lesser-known APIs that can do the very same thing that the more commonly used APIs can do. When you can find a completely different API to accomplish the same thing, use it, because EDR solutions might not be looking out for those APIs.

Here is the URL for x86matthew: https://www.x86matthew.com.

© SANS Institute 2024

https://linktr.ee/offsecexam

Requirements



The Red Team is your customer

Most mature organizations with a dedicated developer shop directly support the company's internal Red Team. The Red Team will be a client of yours and should be producing a list of requirements that are needed for their engagements. Your objective will be to satisfy those requirements and release a tool to them.

Requirements

When it comes to determining how to make a capability that does X, Y, and Z things, you need to think of the end user of your tool and how they might want it deployed in an engagement. When the roles are split out properly, meaning a company does not require their Red Team operators to be developers as well, they will have created and properly funded a development shop whose sole purpose is to support the Red Team or the Penetration Testing Team. There should be some formal system in place for how the Red Team gives the development shop a list of requirements for a feature to add to an existing tool or to make a brand-new tool from scratch. This could be a Jira ticket, a Word doc, an email, etc., something that has a paper trail and is formal. The development shop should then discuss those requirements to determine how long it might take to meet them and go over if some of them are not even feasible. Once a potential date is determined, the tool should be released and formally accepted by the Red Team.

© SANS Institute 2024

https://linktr.ee/offsecexam

Group Exercise



Scenario: You have just received requirements from the Red Team lead

- Go over the requirements
- Determine which ones are not technically possible
- Determine which ones need more of an explanation
- Discuss a timeline for release

Group Exercise

Let us pause for a moment and go over a scenario where you are the team lead for the development shop. The Red Team lead has just created a ticket that has a fairly large number of requirements in it for a new capability for an upcoming engagement they have. The Red Team will be facing a new security product they have yet to test against but have now required that from you. Here is the list of requirements the Red Team lead has given along with a timeline:

- Compiled 32-bit and 64-bit binaries
- Main program is an EXE
- Additional payloads in DLL format
- Must be no larger than 400kb
- Must perform recon
- Must be able to do code injection
- Must be able to persist
- Must be able to elevate privileges
- Must be able to bypass security products user-mode hooks
- Must be able to load additional payloads, e.g., DLLs, shellcode blobs, etc.
- Must communicate over HTTP with Python C2
- Must support download capabilities from target
- Time to release: 6 days

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary

- Implants, survey tools, etc. are not trivial to develop
- Test and break your tools intentionally, see where they fail early versus on target
- Requires in-depth knowledge of Windows OS
- Requires in-depth knowledge of Windows APIs

Module Summary

It is not easy to develop well written and well performing code, but this is a necessity. The tools that you develop must never crash your target and should never run with unexpected results. Your testing should tell you everything your tool will do and will not do, like what breaks it. Unleash your creative side and explore the realm of new possibilities.

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

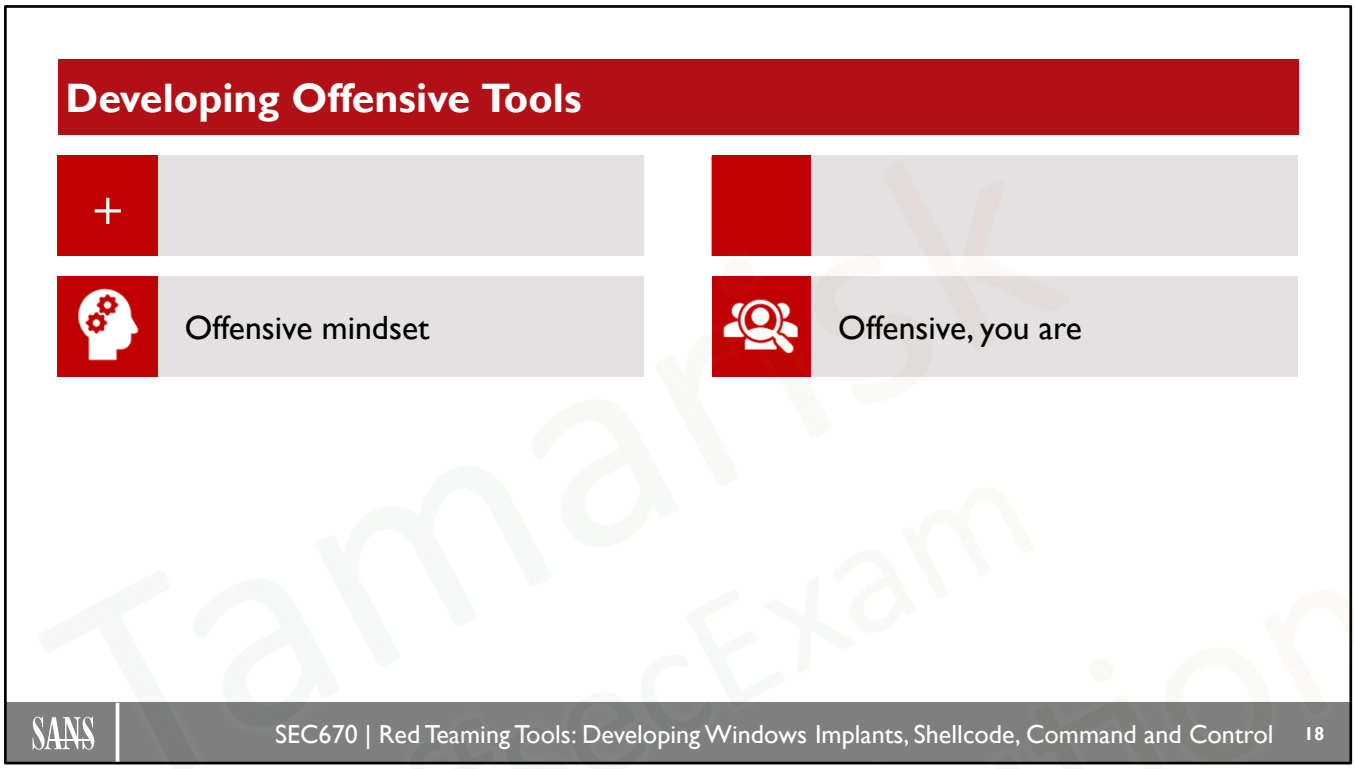
Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

In this module, we will discuss what it means to develop offensive tools.

© SANS Institute 2024

https://linktr.ee/offsecexam



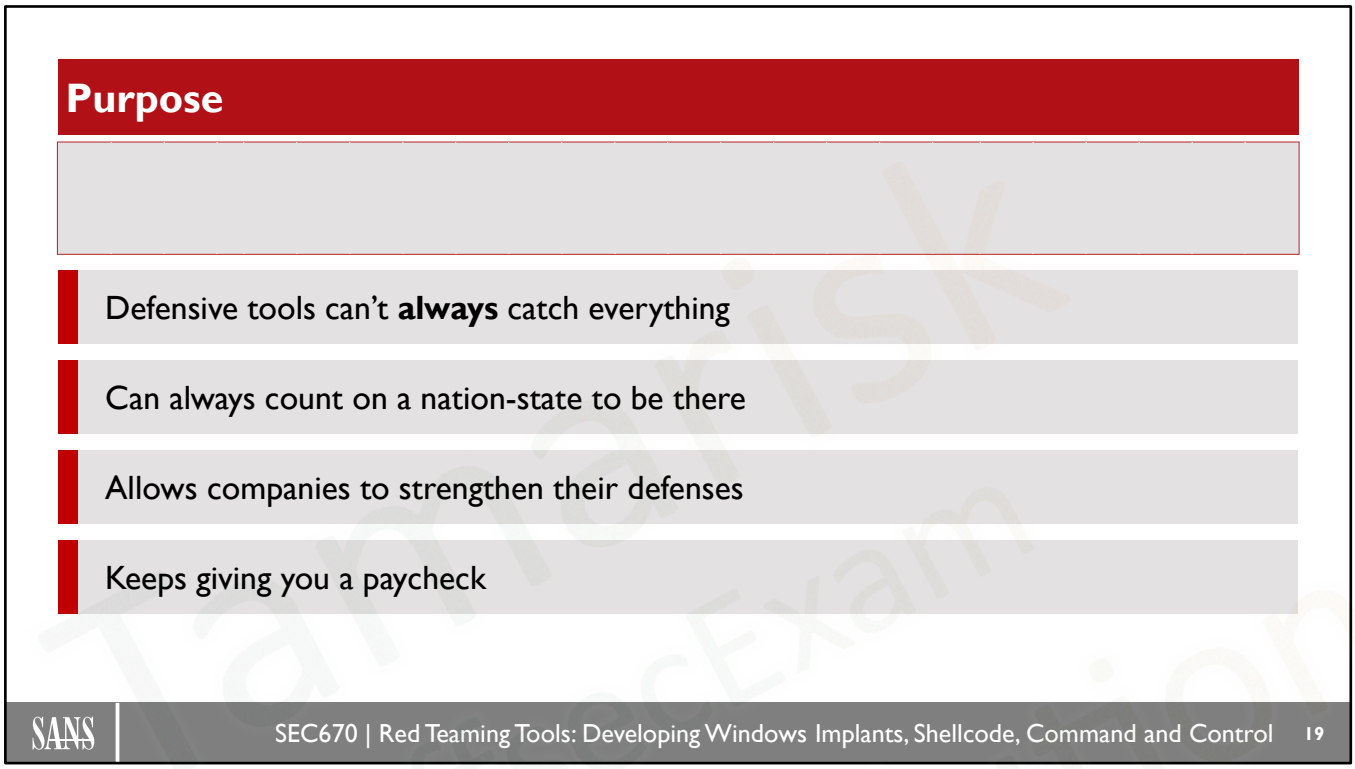
Developing Offensive Tools

Tim Medin, Founder and CEO of Red Siege, also a SANS Instructor/Author for SEC560, has a saying, "I am offensive." This saying is great because being offensive or having that offensive mindset is what will help you achieve your goals, and our main goal for this course is making you an offensive tool developer for Windows targets. The offensive mindset and action is all about taking advantage of weaknesses or vulnerabilities, exploiting built-in features to establish persistence, avoiding AV/EDR solutions, and out-maneuvering sysadmins.

You are going to be developing programs that do something nefarious. We will not be developing tools to detect malicious actions but to create the tool that puts those detection tools through the paces. If you are coming from the defensive side and have the defensive mindset, great! You can use that as an assist and eventually you can switch your mindset to more of an offensive one.

© SANS Institute 2024

https://linktr.ee/offsecexam



Purpose

Offensive tooling is necessary for several reasons, the first one is that you probably would not have a job without it. Several organizations might even require external assessments to be conducted every so often. Financial institutions have audit requirements that must take place at certain intervals, and an external red team might serve that requirement. Another reason would be for national security because you do not want enemies of your country to gain unauthorized access to your network all while you are trying to get access to theirs. How do you think you get access to their network aside from human operations or something similar? The answer: offensive tools.

Perhaps another reason is so we can better defend our networks from enemy nation states and non-nation-state actors by having effective red team ops using similar tools. The more you can practice your defense the stronger your defenses become, but you cannot practice alone. You need someone to practice against, so that you can identify possible weak points in your playbook. Until we can rid the world of bad actors, we will always need offensive tools or capabilities.

© SANS Institute 2024

https://linktr.ee/offsecexam

Current State of the Art Frameworks and Tools

There are many frameworks or offensive tools available for use today:


Metasploit Framework


Cobalt Strike


Empire


Mimikatz


PsExec


Eternal*

Current State of the Art Frameworks and Tools
Today, there are some amazing tools out there and many are free for anyone to grab and leverage. We see this happening very frequently and such actions have sparked intense discussions regarding open sourcing powerful tools and capabilities that nation state and non-nation-state actors can use against us. Regardless of where you stand in that debate, offensive tools will almost always be needed so we can sharpen up defenses.

In the open-source world, or the public community, the tooling is getting better as new techniques are discovered. Researchers and developers are getting more creative with their craft. Developers are discovering new methods to bypass AV/EDR solutions, or bypass UAC restrictions, or elevate privileges. Take #Zerologon, for example: at a high level, a researcher discovered that you could, with access to a domain controller, create an account in the domain with an empty password, which allowed for a complete takeover of the DC. Once a vulnerability is found then it is time to develop a tool to exploit it, which could then be added as a capability to an existing framework or a custom in-house tool.

There are several free tools that offer great capabilities that also might have a commercially licensed version that offers more advanced capabilities. This is where the major differences are. When tools come with a heavier price tag, you should expect it to do a lot more for you out of the box, or at least allow you to fully customize certain parts of it. What is being used at your organization: commercial, open-source, or your own internal framework?

© SANS Institute 2024

https://linktr.ee/offsecexam

Future

- How will time change how we develop tools?
- Will code repos eventually ban the storage of such tools?
- New tools are limited by your creative use of the APIs.

Tamarisk OffSecExam Exam Solution

Future

The future can be somewhat what we make it to be if you stop and think about it. If you close off all creativity, ban the posting on offensive security tools to GitHub and other similar sites then you could run the risk of not advancing by much. On the flip side, if your organization can invigorate and inspire creative thinking, then the future for developing new offensive tools and capabilities could be very bright.

© SANS Institute 2024

https://linktr.ee/offsecexam

How You Can Contribute

- Make your work available to the public; open-source your code
- Contribute to an already existing open-source code project
- Develop tools for your red team
- Don't compartmentalize your knowledge

How You Can Contribute

Before you start publishing your offensive tools online, be sure to check with your current employer as to what their intellectual property terms are. You do not want to be in a situation where whatever you create during your own free time with your own equipment belongs to your employer. It could land you in some hot water with a legal team. If you are good to go on that front then by all means, get your work out there. Feel free to contribute to an existing project or framework like Metasploit or Sliver C2. It can be a great way for you to practice and sharpen your development skillset.

If you are not already on a red team, then perhaps you could approach your company's red team lead or director about creating a tool development section within the team, if they do not have one already. Today, more and more red teams are creating development roles to create in-house tools. Whatever you choose to do, share your knowledge with those around you, especially those who are trying to get into this line of work. Compartmentalization just creates a single point of failure and is not good for the overall success of a team.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary

- Discussed the need for offensive security tools
- Discussed the current state of the art frameworks and offensive tools
- Learned to share your knowledge; be a mentor

Module Summary

In this module, we discussed several reasons why it is necessary to develop offensive security tools. We also discussed various state of the art frameworks and offensive tools, where we can take things from here, and not hoarding the knowledge and experience you might have gained over the years.

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview

Developing Offensive Tools

Developing Defensive Tools

Lab 1.1: PE-sieve, Lab 1.2: ProcMon

Setting Up Your Development Environment

Windows DLLs

Lab 1.3: HelloDLL

Windows Data Types

Call Me Maybe

Lab 1.4: Call Me Maybe

SAL Annotations

Lab 1.5: Safer with SAL

Windows API

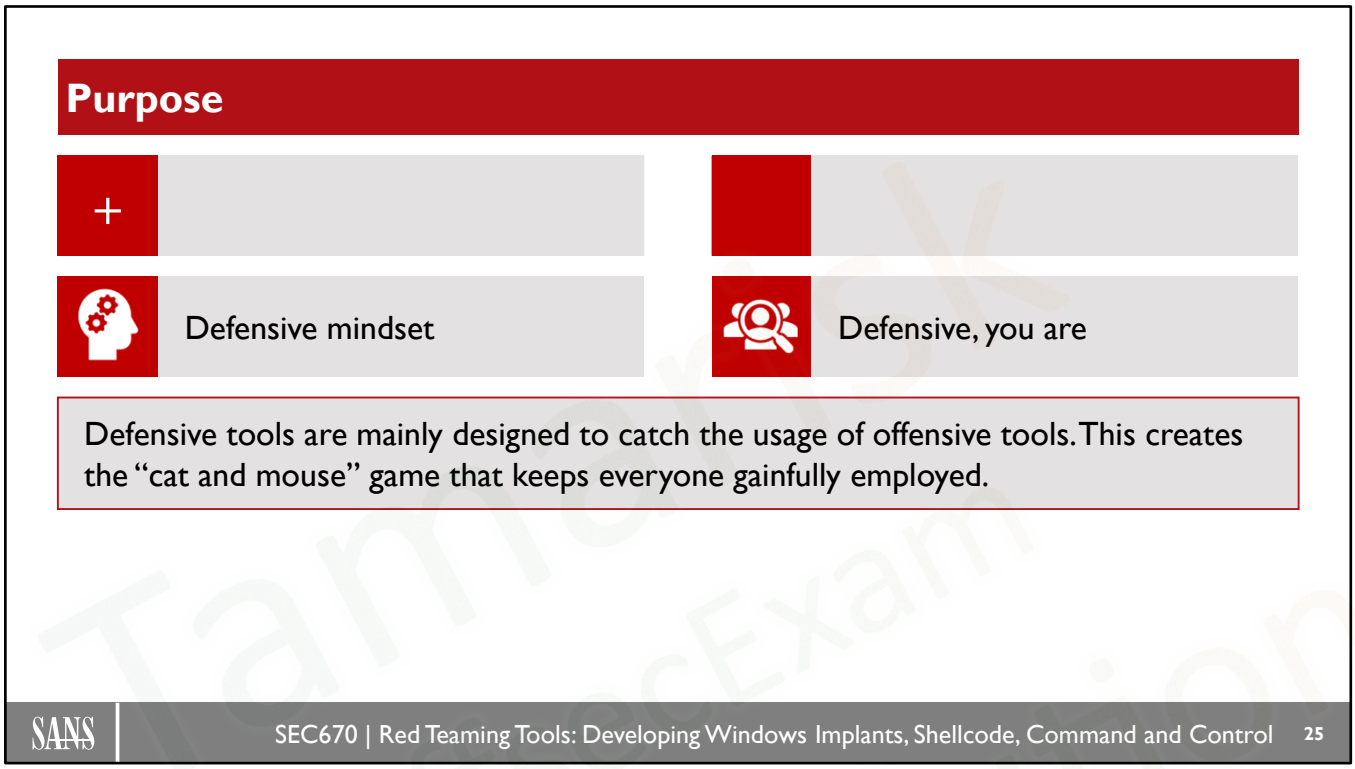
Lab 1.6: CreateFile

Bootcamp

In this module, we will discuss the development of defensive tools, their purpose, state of the art tools, their future, and how you can contribute.

© SANS Institute 2024

https://linktr.ee/offsecexam



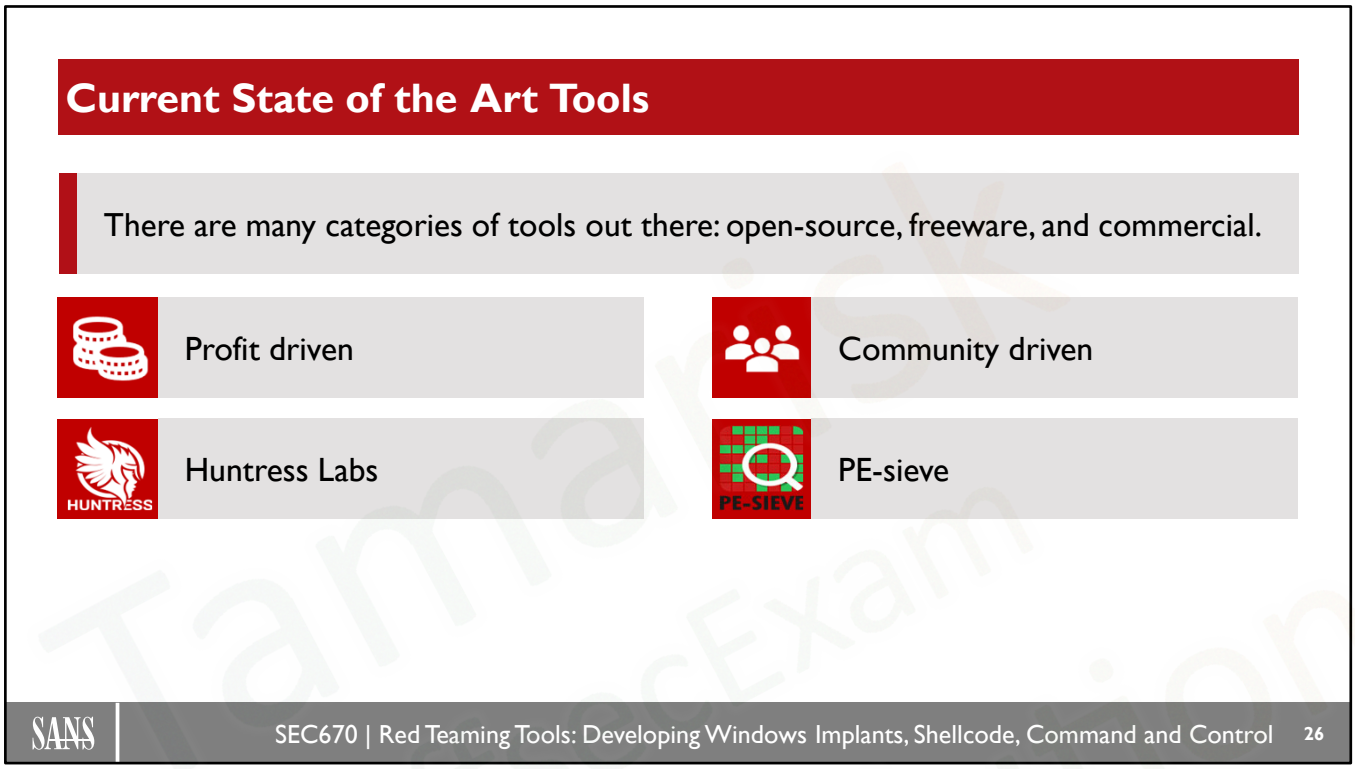
Purpose

Just like offensive tools, there is a necessity for defensive tools. Many companies profit from selling their defensive tools and there is nothing wrong with that. There have been many defensive tools that have been open sourced and pushed to GitHub for all the world to see. Some are truly open source while others are simply freeware, and there is nothing wrong with that, either. Defensive tools, when operating at their best, can be used to detect the execution of your offensive tools. As new offensive tools are created and used, defensive tools must be modified to check for those new methods. This back and forth is a cat and mouse game that will be played for many years to come.

Programming securely can be often viewed as a defensive programming technique. When done correctly it makes the job of vulnerability researchers and exploit developers that much more difficult. If you do not check for logic bugs, memory leaks, overflows, etc. before you release your product, then rest assured that someone else will find it for you; it is only a matter of time. Solid defensive programming is incredibly difficult, and you have to get it right every single time because that one time you do not, there is either a red teamer or a bad actor that is ready and willing to exploit the flaw.

© SANS Institute 2024

https://linktr.ee/offsecexam



Current State of the Art Tools

Open-source tools are great and sites like GitHub are full of them. There is nothing wrong with closed-source, commercial tools but when you can see the code like you can on GitHub, you can fork the project and modify it to fit your needs. Granted you will need to check the license the author put on it, but often the license is not that restrictive. Freeware tools are tools that are, well, free! One possible downside is that the code does not have to be made available to the public.

What about making a profit? There is nothing wrong with companies trying to make a profit, but commercial tools can sometimes come with a hefty price tag. The saying "You get what you pay for" can sometimes apply here and the more popular the company the more expensive the software. Amazing products can come from small to medium sized companies and one such company is Huntress Labs. Huntress Labs offers an amazing product that can complement your current security suite. If your organization is looking for something awesome, then check them out online and talk to Kyle Hanslovan.

PE-sieve is a great defensive tool aimed at finding malware. The author goes by the handle hasherezade and has published the tool on GitHub. The wiki describes the tool very well along with what the tool's capabilities are. For example, PE-sieve can easily dump implants for you that have been injected into process memory. It can also detect the various injection techniques that we will be learning about this week. Pro tip: If you want to see how good your tool is at evading detection, get it to bypass PE-sieve.

The Sysinternals Suite of tools are great too. For instance, ProcMon, with the right filters in place, can help identify if your software is looking for DLLs that are not on the system. Something like this can lead to DLL hijacking where an attacker drops a DLL that is being sought out by the target program. Sysmon is another powerful tool that operates as a system service that can monitor process creation, network connections, and several other items of interest.

© SANS Institute 2024

https://linktr.ee/offsecexam

Future

As newer technology comes out, newer programming languages could come out too. Languages that are natively more secure could replace lower-level languages like C.

- Secure hardware can limit attack vectors.
- AI/ML powered tools can detect threats more quickly and efficiently.
- Languages like Rust can change the game for defenders.

Future

Just like the future of developing offensive tools, the future of defensive tools and programming depends heavily on creativity and attention to detail. Some languages like Rust could really change the game for defensive tools and secure programming. Windows is rewriting some of the kernel in Rust and thus this would increase the level of difficulty for exploitation. Check out Rust if you ever get some free time to see the differences for yourself. Fully automated systems are increasingly becoming more efficient and accurate with the advancements of modern-day computing and technology. That technology can be folded into defensive tools to make them more robust and take some of the fatigue away from the analyst.

© SANS Institute 2024

https://linktr.ee/offsecexam

How You Can Contribute

- Make your work available to the public; open-source your code
- Contribute to an already existing open-source code project
- Spread the word; blog, live streams, tweet
- Double check with your employer's Intellectual Property agreement

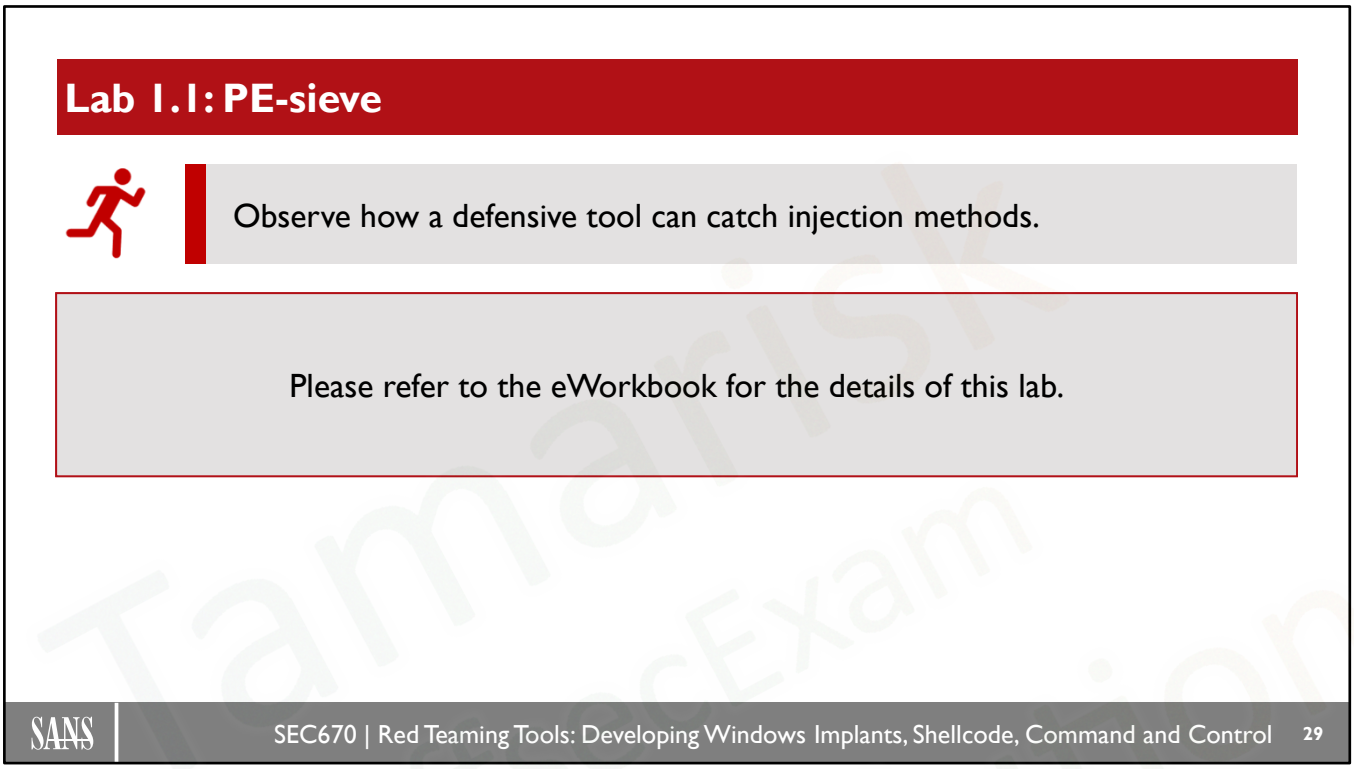
How You Can Contribute

There are many ways you can contribute, and GitHub is just one of the many ways for you to share your code with the world. Once you make a few projects you can start to spread the word about it and share the link to your repos. Many developers will write a short blog about their tool, how to use it, and what inspired them to develop it in the first place. If you do not want to release your own projects, then perhaps you could contribute to some existing ones. Many of the major projects out there love it when people contribute and submit a pull request. Look around and see if there are any projects that interest you.

As always, be sure to check with your employer before doing so.

© SANS Institute 2024

https://linktr.ee/offsecexam



Lab 1.1: PE-sieve
PE-sieve, according to hasherezade's GitHub repo, "is a tool that helps detect malware running on the system, as well as to collect the potentially malicious material for further analysis." The tool is designed to scan a single process, but it does a great job at detecting various items like injected PEs and hooks. It also has the ability to dump an implant should one be discovered injected into a process.

Please refer to the eWorkbook for the details of the lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.2: ProcMon



Observe how ProcMon can be used to spot flaws with a program.

Please refer to the eWorkbook for the details of this lab.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of that lab was to explore how defenders can use Process Monitor to see what their application is doing. Maybe they spotted something that needs to be fixed and they are able to do so before it ever is made available to the user.

Tamarisk OffSecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary

- Learned Rust is becoming more popular
- Learned AI/ML is getting better
- Discussed how defensive tools are getting more advanced
- Discussed how you should contribute however you can

TamarisK OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

Tanay Nigam offsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Setting Up Your Development Environment (1)

The virtual machines made available to you are already configured for this course. If you have your own development environment and would like to use it, that is fine.

- Please do not install any updates unless directed otherwise.
- Copy over everything from the media downloads to your host machine.

Tamang OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Setting Up Your Development Environment (2)



How to set up your own development environment at home:

- Must have the Windows Software Development Kit (SDK)
- Use Visual Studio Professional
- Install Windows C/C++ build tools
- Use some versions of Windows

Tanrik OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Choosing a Language

Does the language really matter when doing implant development or creating host survey tools for a target?

- An implant in pure C
- An implant in C++
- Will one be better than the other?

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Choosing an IDE

With all the various IDEs available, how can you choose one?


Visual Studio Code


Visual Studio Community


Visual Studio Professional


Visual Studio Enterprise


Xcode

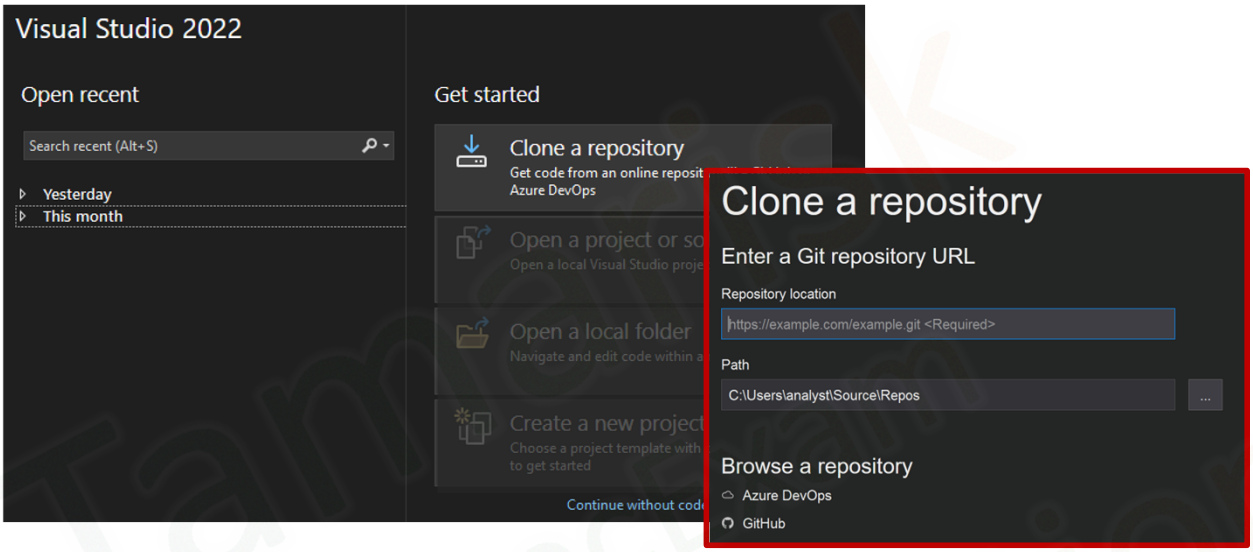

Code::Blocks

Yamato Offsec Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

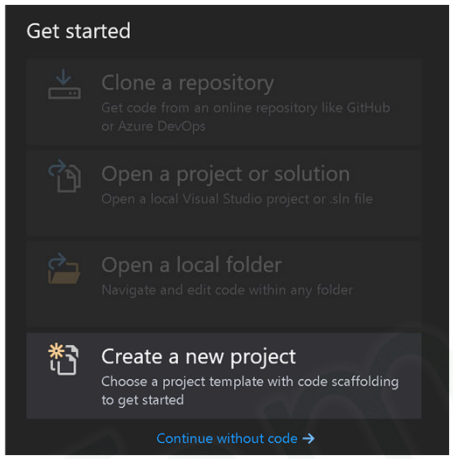
Cloning a Repository

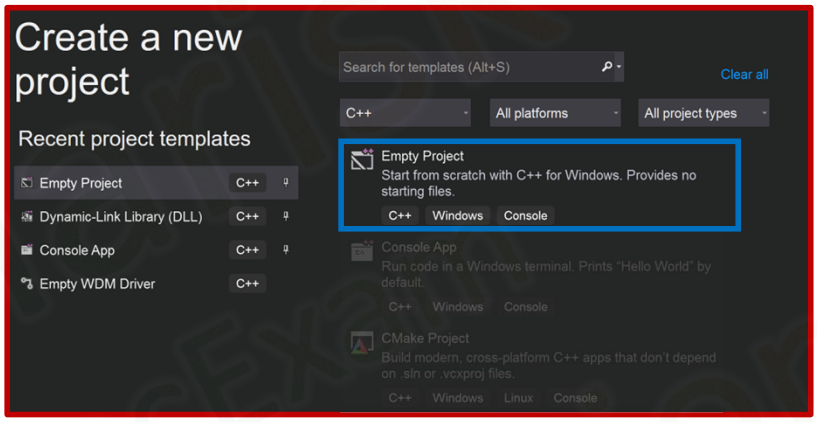


© SANS Institute 2024

https://linktr.ee/offsecexam

Creating a New Project

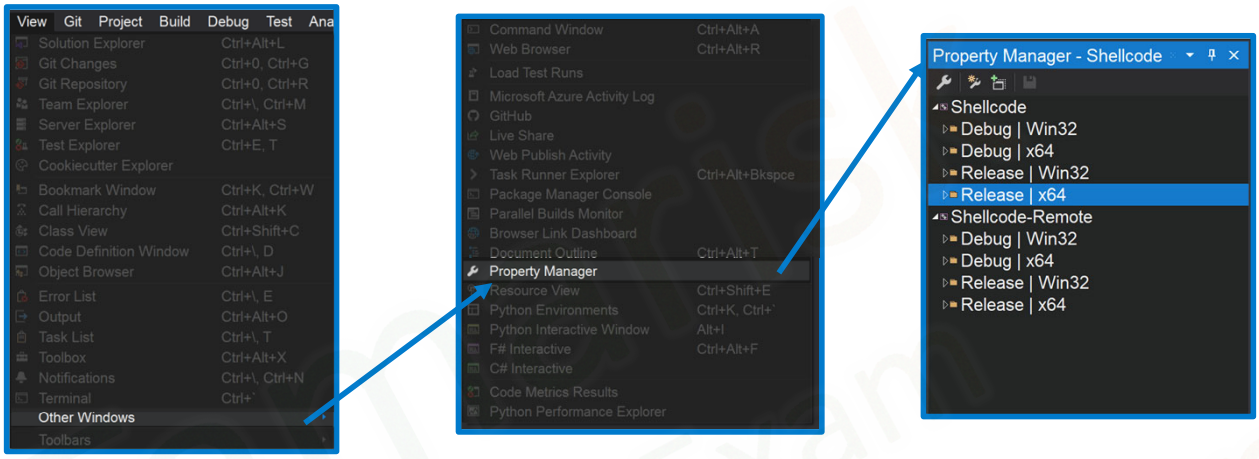




© SANS Institute 2024

https://linktr.ee/offsecexam

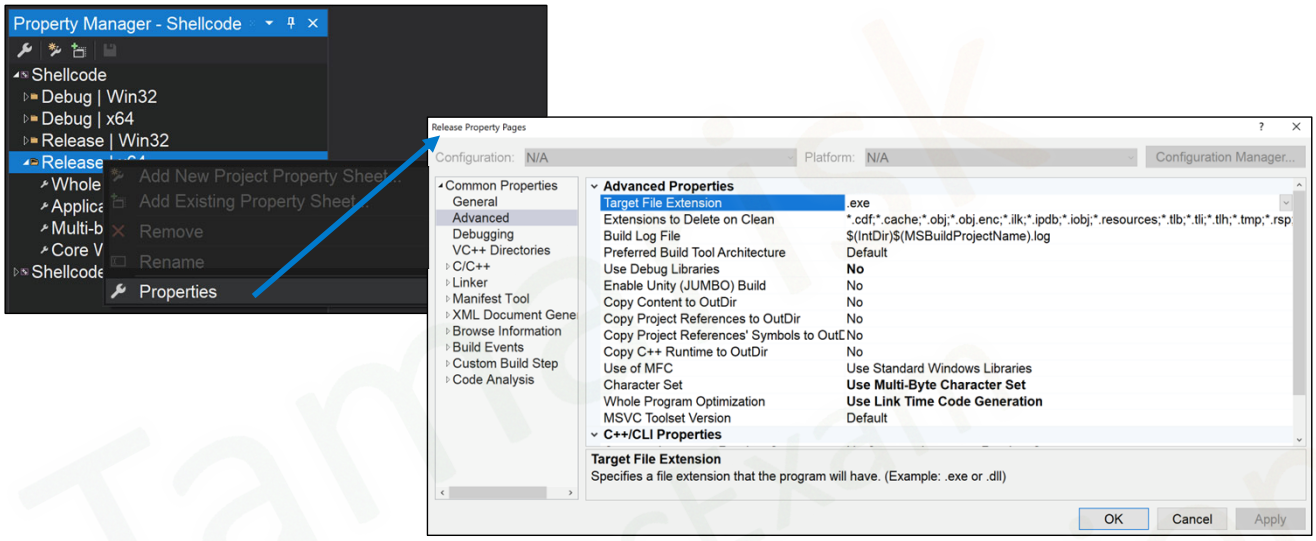
Visual Studio Property Sheets (1)



© SANS Institute 2024

https://linktr.ee/offsecexam

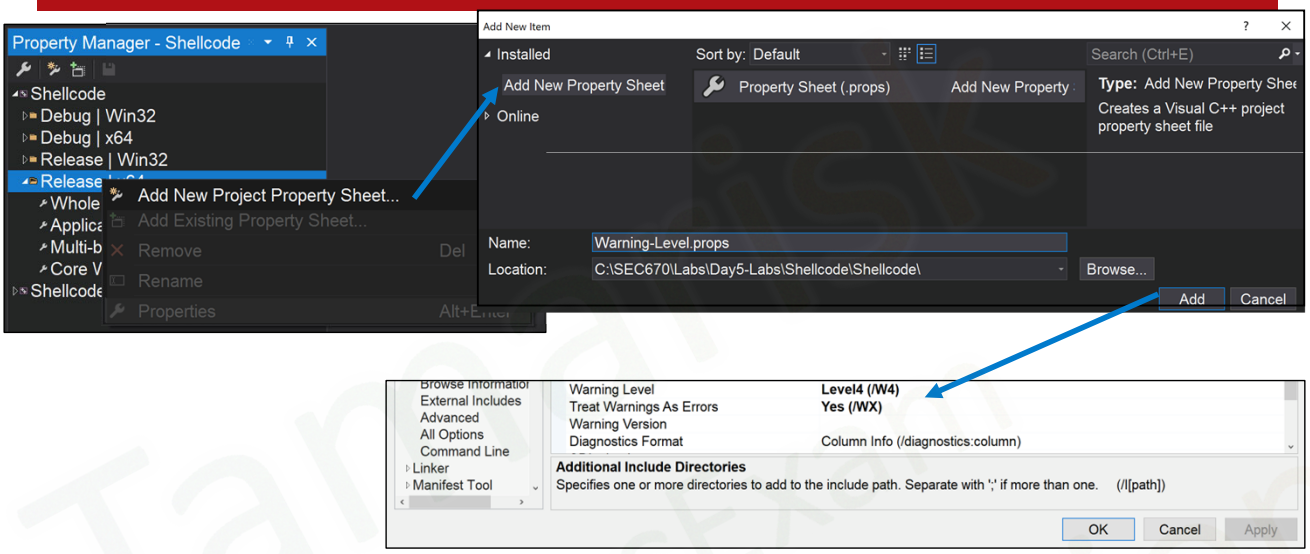
Visual Studio Property Sheets (2)



© SANS Institute 2024

https://linktr.ee/offsecexam

Visual Studio Property Sheets (3)



© SANS Institute 2024

https://linktr.ee/offsecexam

Your Testing Environment

Your testing environment is just for that, testing. It should be robust and capable of scalability to suit your needs.

- It might be a good idea to test your tool with various version of Windows.
- Don't perform only a single test case and assume all is well—perform hundreds.
- Validate your tool before you put it live on a target.

Tamir OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Build Targets

Even if you develop on the most recent version of Windows, it does not mean you cannot build a version that targets an older version like Windows XP.

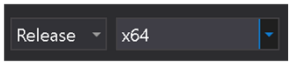
- Builds for a single target
- Builds for maximum effect and reach; multiple versions of Windows
- Your tool does not have to execute on every version of Windows

Tamarisk OffsecExam Exam solution

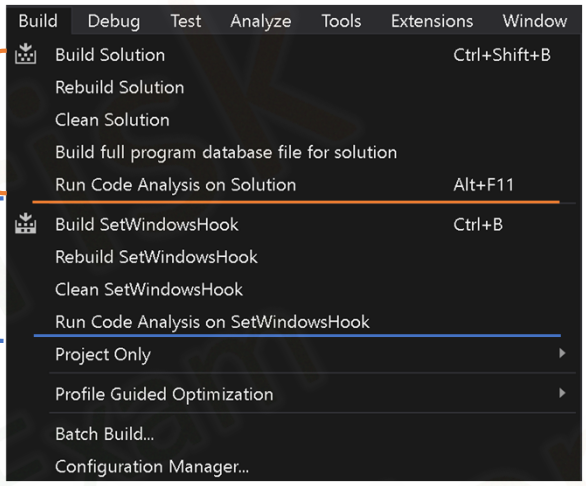
© SANS Institute 2024

https://linktr.ee/offsecexam

Build Visual Studio Solutions/Projects (1)



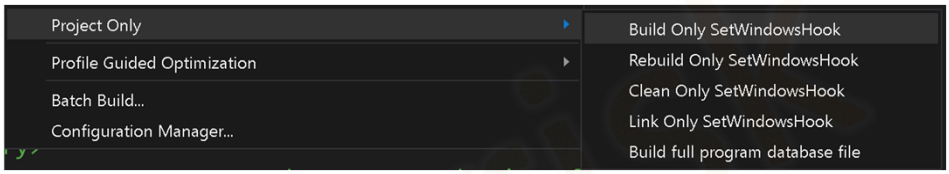
- Choose configuration and architecture: Debug/Release, x86/x64
- Build the entire solution?
- Build only the project?

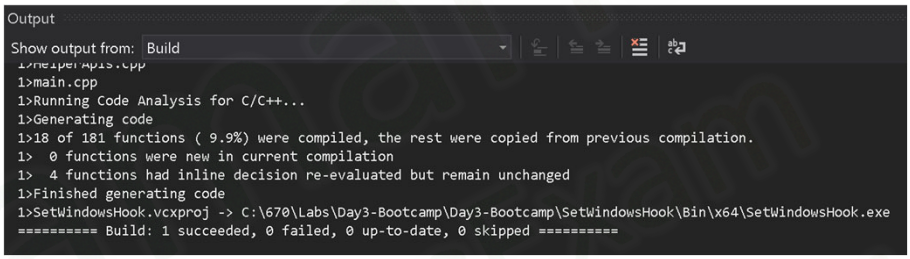


© SANS Institute 2024

https://linktr.ee/offsecexam

Build Visual Studio Solutions/Projects (2)

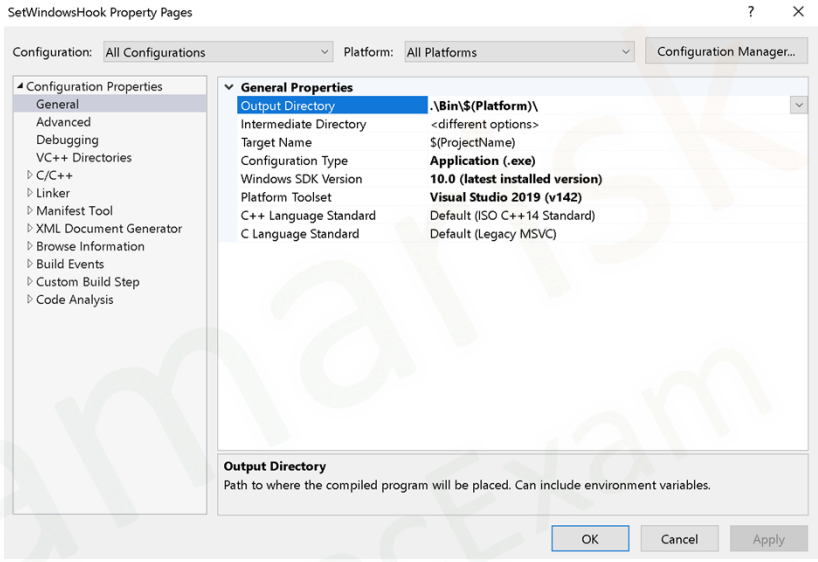




© SANS Institute 2024

https://linktr.ee/offsecexam

Visual Studio Project Settings



Visual Studio Project Settings

We will not be going over every single setting in the Project Properties window, but it is important to know some basic items and options here. At the topmost portion of the window, you have Configuration and Platform. The Configuration choices are nothing new: Debug, Release, or All. The Platform can be one of the following options: Win32, x64, or All. In the General Properties, the Configuration Type can be selected: EXE, DLL, or LIB. If your system has more than one SDK available, you can select a specific version, like one for Windows XP.

There are other sections in the Property Pages window, such as C/C++ where code optimization can be tweaked, and Precompiled headers can be modified and selected. There are also a few other options here, such as Warning Level and Calling Convention. The Linker section is where you can specify any additional dependencies—think of libraries like kernel32.lib, etc. You can also choose any module definition files that might be of use, say if you are building a DLL. In addition, there are linker optimizations that can be tweaked as well as characteristics like Data Execution Prevention (DEP) and Dynamic Base (ASLR). This is where you can tweak the size of your binary, and in the world of implants, the smaller your implant is the better. 40MB is not a good size for an implant.

The last section covered here is Code Analysis, which is where you can enable Microsoft Code Analysis to run as you develop and build your program. Be sure this feature is enabled as it can save you from some future headaches. Code analysis does what it sounds like. It will perform a static analysis of your code and provide some detailed feedback should anything pop up that needs your attention. To directly quote MSDN documentation about the feature: “Common coding errors reported by the tool include buffer overruns, uninitialized memory, null pointer dereferences, and memory and resource leaks.” Code analysis improves when your source code uses source-code annotation language (SAL). SAL is something that is used heavily in this course and it even has its own module and lab later in this section. Look at the references below for more information about the code analysis for C/C++.

Reference:
https://learn.microsoft.com/en-us/cpp/code-quality/code-analysis-for-c-cpp-overview?view=msvc-170

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary

Diversity with your tools is vital and perhaps more so depending on your current work role and employer. Dropping the same implant every time only works for so long.

- Discussed how to know your target platform with as much detail as possible
- Learned to create a Build for a single target
- Discussed choosing the IDE that best suits your needs

Tarnish OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section I

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

Tanay M
offsecEn
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (1)

Dynamic-linked libraries (DLL) look just like a standard Windows executable file. The purpose is what separates the two. DLLs primarily supply a common functionality to applications that need it.

- PE32 / PE32+ format
- Designed to be shared
- Various extensions: .lib and .dll

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (2)



What is inside of a DLL?

stub	DOS stub; useless today
PE	PE\0\0
COFF	Common object file format
optional	Image-specific file headers
section	Information about the sections
sections	Actual code, data, resources

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (3)



How do you see what is inside of a DLL?

- Dumpbin utility
- PEview
- PE-bear
- CFF Explorer

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (4)

```
// dumpbin /headers hello.dll
Dump of file hello.dll

PE signature found

File Type: DLL

FILE HEADER VALUES
       14C machine (x86)
         3 number of sections
  602A3E55 time date stamp Mon Feb 15 09:26:45 2021
         0 file pointer to symbol table
         0 number of symbols
        E0 size of optional header
      2102 characteristics
             Executable
             32 bit word machine
             DLL
```


TestOut OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (5)

```
OPTIONAL HEADER VALUES
        10B magic # (PE32)
      14.28 linker version
        200 size of code
        400 size of initialized data
          0 entry point
       1000 base of code
       2000 base of data
   10000000 image base (10000000 to 10003FFF)
       1000 section alignment
       4000 size of image
        400 size of headers
          0 checksum
          2 subsystem (Windows GUI)
        540 DLL characteristics (Dynamic base, NX compatible, No SEH)
     100000 size of stack reserve
       1000 size of stack commit
```


Tenarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (6)

```
10 number of directories
     2040 [    48] RVA [size] of Export Directory
        0 [     0] RVA [size] of Import Directory
        0 [     0] RVA [size] of Resource Directory
        0 [     0] RVA [size] of Exception Directory
        0 [     0] RVA [size] of Certificates Directory
     3000 [     C] RVA [size] of Base Relocation Directory
     2018 [    1C] RVA [size] of Debug Directory
        0 [     0] RVA [size] of COM Descriptor Directory
        0 [     0] RVA [size] of Reserved Directory
```

Virtual Address = Base Address + RVA

RVA = Virtual Address - Base Address

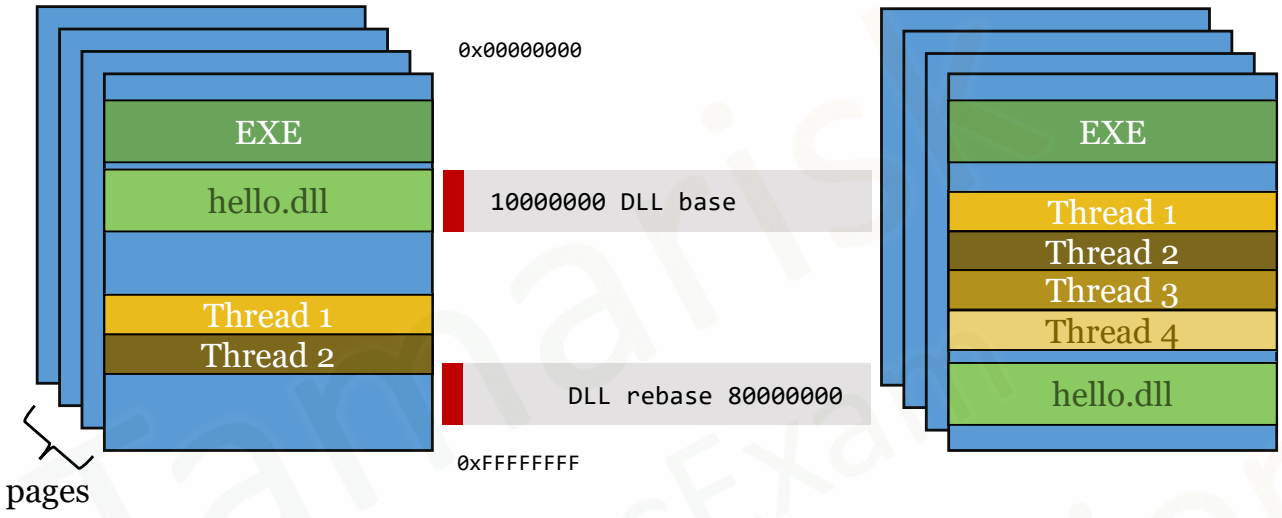
9F002040 = 9F000000 + 2040

FFC90 = 743AFC90 - 742B0000

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (7)



© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (8)

```
SECTION HEADER #1
  .text name
        A virtual size
     1000 virtual address (10001000 to 10001009)
      200 size of raw data
      400 file pointer to raw data (00000400 to 000005FF)
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
 60000020 flags
          Code
          Execute Read
```

.TEXT section

The executable code

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (9)

```
SECTION HEADER #2
 .rdata name
       D8 virtual size
     2000 virtual address (10002000 to 100020D7)
      200 size of raw data
      600 file pointer to raw data (00000600 to 000007FF)
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
 40000040 flags
          Initialized Data
          Read Only
```

.RDATA section

Read only, initialized data

OPTIONAL_HEADER Directories
2040 [48] RVA [size] of Export Directory ← within .rdata
2018 [1C] RVA [size] of Debug Directory ← within .rdata

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (10)

```
/exports hello.dll



00000000 characteristics
FFFFFFFF time date stamp
    0.00 version
       1 ordinal base
       1 number of functions
       1 number of names

ordinal hint RVA   name

   1  0 00001000 PrintHello
```

```
/disasm /section:.text



10001000: push ebp
10001001: mov ebp,esp
10001003: mov eax, 10002000h
10001008: pop ebp
10001009: ret

 Summary

    1000 .text
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Dynamic-linked Libraries (11)



System DLLs are forcefully mapped into nearly every single process.

NT	Ntdll
K32	Kernel32
KB	Kernelbase
U32	User32

Tamás OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Linking (1)



A DLL does nothing for you unless you link your program to it.

Implicit linking

The OS is going to load the DLL right when that executable uses it. The exported functions can be called as if the functions were linked statically and resided within the executable itself.

Explicit linking

The OS is going to load the DLL on demand right at runtime. The desired DLL must be **explicitly** *loaded* and *unloaded* by the client executable. Exported functions are called via pointers.

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Linking (2)



How to implicitly link a DLL.

- Must have header file; .h files
- Must have the actual DLL file
- Must have import library to link into the EXE
- Project Settings in Visual Studio

```
#pragma comment(lib, "hello")

//function from hello.lib
//PrintHello();

puts(PrintHello());
```

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Implicit Linking

```
// implicit_callhello.cpp
#include <stdio.h>

EXTERN_C PCHAR __cdecl PrintHello();

INT main(){
  puts(PrintHello());
}
```

Must be linked against the LIB file

```
link implicit_callhello.obj hello.lib
```

```
dumpbin /all hello.lib
```

```
5 public symbols
  [..snip..]
  62E _PrintHello
Archive member name at 62E:
  Type: code
  Symbol name: _PrintHello
  Name: PrintHello
```

© SANS Institute 2024

Image has the following dependencies:

hello.dll
KERNEL32.dll

dumpbin /exports hello.lib
File Type: LIBRARY

Exports

ordinal name

_PrintHello

dumpbin /all hello.lib
5 public symbols

18A __IMPORT_DESCRIPTOR_hello
3AC __NULL_IMPORT_DESCRIPTOR
4E0 hello_NULL_THUNK_DATA
62E _PrintHello
62E __imp__PrintHello

Archive member name at 62E: hello.dll/

DLL name : hello.dll
Symbol name : _PrintHello
Type : code
Name type : no prefix
Hint : 0
Name : PrintHello

Examrisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Explicit Linking (1)



How to explicitly link a DLL.

- Match the function signature
- LoadLibrary
- GetProcAddress
- FreeLibrary

```
typedef DWORD(CALLBACK* FPFUNC1)(DWORD);

HINSTANCE hMyDll;
FPFUNC1 fpFunc1;
hMyDll = LoadLibrary("MyDll");
fpFunc1 = (FPFUNC1)GetProcAddress(hMyDll,
                  "Func1");
fpFunc1(32);
FreeLibrary(hMyDll);
```

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Explicit Linking (2)

```
// callhello.cpp
#include <stdio.h>
#include <Windows.h>

INT main(){
  HMODULE helloDll = LoadLibraryExW(L"hello.dll", nullptr, 0);

  using t_PrintHello = PCSTR (__cdecl*)();
  t_PrintHello PrintHello = reinterpret_cast<t_PrintHello>(GetProcAddress(
                                                          helloDll,
                                                          "PrintHello"));


  puts(PrintHello());
  FreeLibrary(helloDll);

  return ERROR_SUCCESS;
}
```

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Explicit Linking (3)

```
dumpbin /dependents callhello.exe


Dump of file callhello.exe

File Type: EXECUTABLE IMAGE

 Image has the following dependencies:
    KERNEL32.dll

 Summary
    2000 .data
    6000 .rdata
    1000 .reloc
    D000 .text
```

```
dumpbin /imports callhello.exe


Section contains the following imports:

KERNEL32.dll
 40E000 Import Address Table
 4133EC Import Name Table
      0 time date stamp
      0 Index of first forwarder

    1AB FreeLibrary
    2AE GetProcAddress
    3C3 LoadLibraryExW
    [..snip..]
```

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Exports (1)

2 DLLs can create two kinds of functions.



Exported

- Available for other applications
- Functions are specified for export



Internal

- Not available for other applications

Tanglisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Exports (2)



How to specify functions to be exported



__declspec()

- __declspec(dllexport)
- #define DLLEXPORT



Module definition file, .def

- Contains a list of exported functions
- Requires a certain syntax

Tamaris OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Exports (3)



Example of a .def file

```
LIBRARY  Evil32
EXPORTS  ExecShellcode              @1       NONAME
         GetComputerName            @3       PRIVATE
         GetNetAdapter              @5
         GetSysInfo=another_dll.GetSystemInformation
VERSION 2.1
```

© SANS Institute 2024

https://linktr.ee/offsecexam

DLLs: Injection



DLL injection: Forcing a process to load a specific DLL

The code of the DLL will execute with the context of the target process

Unlimited access to everything in the target process

Injection does not always mean malicious activity; it is neither good nor bad

Tanish OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Shared Objects



- Linux shared objects have the ELF format
- Linux SOs do not have a specific export syntax
- Extensions are a **.so** or **.a**
- Can by dynamically loaded/unloaded with `dlopen/dlclose`
- Resolve symbols with `dlsym`

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Data Types / C Data Types



Windows data types do not natively exist for Linux.

Practically every data type that is used was defined using the typedef keyword. The traditional C data types can be used if so desired, but it is best to simply use the provided Windows data types.

```
int, long, unsigned long, double, float, etc.
```

```
INT, DWORD, BOOL, LPVOID, etc.
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Header Files



Also known as include files: `#include <stdio.h>`

Header files, or include files, are the same thing you might already be accustomed to with your Linux development.

Windows created their own header files.

Windows.h | Windef.h | WinNt.h | WinReg.h | WinSvc.h

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows APIs



Windows APIs and their calling conventions

This is a Windows-specific calling convention that behaves slightly differently from __cdecl convention.

API names can be very descriptive and lengthy.

Critical functionality is provided via these APIs.

Tanrisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Standard C Functions

- C standard library of functions are still available.
- memcpy, printf, fopen, etc.
- In the end, just use the Windows APIs.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.3: HelloDLL



Learn how to build a simple DLL.

Please refer to the eWorkbook for the details of this lab.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary

- Discussed how DLLs are just like SOs
- Covered how new header files bring new APIs for new functionality
- Discussed a best practice: Use Windows APIs whenever possible

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the preferred way to export functions in a DLL?

A. Using declspec()

B. Creating a definition file

C. Using EXTERN_C

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the preferred way to export functions in a DLL?

A Using declspec()

B Creating a definition file

C Using EXTERN_C

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is explicit linking?

A. Linking to DLLs at compile time

B. Linking to DLLs at runtime via Windows APIs

C. Linking to DLLs using preprocessor directives

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is explicit linking?

A. Linking to DLLs at compile time

B. Linking to DLLs at runtime via Windows APIs

C. Linking to DLLs using preprocessor directives

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Data Types



Taking an in-depth look behind the Windows data types

Windows data types are used to keep compilers from determining what they think the size of an *int* should be sized as.

Data type names are descriptive once you understand them.

```
BOOL, INT, DWORD, VOID, PVOID, LPVOID, HINSTANCE, HANDLE
```


Tanglik OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

BOOL/BOOLEAN



`BOOL` and `BOOLEAN` appear the same, but are quite different internally



Either on or off

Functions can return TRUE/FALSE

Variable can hold TRUE/FALSE values

```
// WinDef.h
typedef int BOOL;

// WinNT.h
typedef BYTE BOOLEAN;

BOOL IsRunning = TRUE;
BOOLEAN IsElevated = TRUE;
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: BOOL



BOOL example

```
BOOL EnableDebug(
  HANDLE Token,
  LPCTSTR Privilege,
  BOOL EnablePrivilege
)
{
  //adjust token privileges
  //attributes to
  //SE_PRIVILEGE_ENABLED

  if (!AdjustTokenPrivileges())
    return FALSE;

  return TRUE;
}
```

© SANS Institute 2024

https://linktr.ee/offsecexam

A Table of Data Types

CRT Type	Win32 Type	Description
int	INT/BOOL	32-bits signed integer
unsigned int	UINT	32-bits unsigned integer
unsigned short	WORD	16-bits unsigned integer
unsigned long	ULONG/DWORD	32-bits unsigned integer
unsigned char	UCHAR/BYTE/BOOLEAN	8-bits unsigned char
void	VOID	Nothing, can be any type
__int64	INT64	64-bits signed integer
unsigned __int64	QWORD	64-bits unsigned integer

A Table of Data Types

The table here is a nice snapshot of a small subset of the data types you will come across during your time doing Windows implant development. Many of the Windows types are typed to other Windows types, which are then typed to the standard c-runtime type. You can also see in the table that Windows can have several types that all boil down to the same CRT type. For example, UCHAR, BYTE and BOOLEAN all boil down to unsigned char, which is a single byte in memory.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: WORD, DWORD, LPDWORD, QWORD

```
DWORD ProcessId = GetCurrentProcessId();

PDWORD pProcessId = &ProcessId;

LPDWORD lpProcessId = &ProcessId;

DWORD pPeb32 = __readfsdword(0x30);

QWORD pPeb64 = __readgsqword(0x60);
```


Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

A Table of Pointer Data Types

CRT Type	Win32 Type	Description
int *	PINT/PBOOL	pointer to 32-bits signed integer
unsigned int *	PUINT	pointer to 32-bits unsigned integer
unsigned short *	PWORD	pointer to 16-bits unsigned integer
unsigned long *	PULONG/PDWORD/LPDWORD	pointer to 32-bits unsigned integer
unsigned char *	PUCHAR/PBYTE/PBOOLEAN	pointer to 8-bits unsigned char
void *	PVOID/LPVOID/LPCVOID	Nothing, can point to any type
__int64	INT_PTR	signed integer only for pointer precision when casting a pointer to an integer
unsigned __int64	UINT_PTR	unsigned int pointer

A Table of Pointer Data Types

The table here is a nice snapshot of a small subset of the pointer data types you will come across during your time doing Windows implant development. You will notice that some of the types have the letter L in front. This used to indicate the pointer is a long pointer to some data type. Long pointers are not relevant anymore and as such, they can be used interchangeably with traditional pointer types like so: LPVOID and PVOID. One thing of importance is with the PVOID data type, you cannot perform pointer arithmetic with it. Should you try to do so, Visual Studio should give it a red underline telling you it cannot be done. You should not even be able to build your project. If you are using a different compiler that lets you get away with that operation, the behavior will be unpredictable. The industry classifies that behavior as undefined behavior.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: VOID, PVOID, LPVOID, LPCVOID

```
VOID GiveGreeting()
{
 printf("I don't return anything. Muah ah ah!\n");
}

PVOID baseEntry = (PVOID)pimgOptionalHeader->AddressOfEntryPoint;

LPCVOID pConstant = &SomeConstant;

VOID OverflowMe(PCHAR Buffer)
{
  strcpy(Buffer, "Buffer overflows are in SEC660!");
}
```


Iamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

A Table of C++ Types

C++ Type	Description
std::vector<T>	like a C array; more powerful, safer, dynamic, self-cleanup
std::string and std::wstring	ANSI and Unicode string objects
std::string_view	a view into a std::string when ownership of the string isn't needed
std::bit_cast<T>()	safer way to cast data from one type to another
CStringA and CStringW	an ATL string class object that offers robust methods
std::make_shared<T>()	makes a smart, shared pointer; safer than raw pointers
std::shared_ptr<T>()	shouldn't use this with the new operator; stick with make_shared
std::make_unique<T>()	makes a smart, unique pointer; can only be one to same address
std::unique_ptr<T>()	a unique pointer; stick with make_unique instead

A Table of C++ Types

The here captures some of the more commonly used C++ types that you'll find in this course and outside this course. Many come from the Standard Template Library (STL) and others come from the standard library. Of course, each one will have their respective header files to include, unless you are using C++ 23, which utilizes header units. If this is your first time seeing some of these data types, take your time with them as they will come with a bit of a learning curve.

The std::vector<T> is by far one of the most common and the most versatile data type. You will see this being used in the course to act as a buffer. For example, it can be used to hold the contents of a file that you read into memory.

The std::string and std::wstring are a massive enhancement from a traditional C-style string, which are just CHAR arrays. They offer many methods for comparing strings, appending, updating, etc. These are much, much easier to use than the C-style strings too. One nice small feature of std::strings is the number of characters in it. Please note that this is a character count and not a byte count, which can be two different values.

The std::string_view is a reference to a constant string. It is the perfect type for when ownership is not needed, like when a function will not be modifying the contents of the string. This means that a string will not have to be copied each time it is being passed to another function. This would increase efficiency in larger applications. The string could be anything from a C-style string to a std::string. The string_view object could not care one bit what it views.

The std::bit_cast<T>() is a safe way to type cast bits from one object to another object of type T. It is best to use this when both types are of the same size and padding, otherwise you might introduce some undefined behavior. This method is extremely safer than the popular reinterpret_cast<T> you may have used or have seen. The reinterpret_cas<T> can also introduce undefined behavior. So, with C++ 20, use bit_cast<T>.

© SANS Institute 2024

https://linktr.ee/offsecexam

CStringA and CStringW are part of a class under the Active Template Library, or ATL. These objects are very powerful and feature rich. They give std::strings a run for their money because of their safety, features, and ease of use.

The std::make_shared<T>() and std::make_unique<T>(). These are both used to make smart pointers. Smart pointers are much preferred over raw pointers, or traditional C-pointers. Smart pointers imitate raw pointers in a sense because they contain an address to something, and you can generally use them the same way. The best thing about smart pointers is they free you of the requirement to call delete or delete[] operators when freeing memory. Smart pointers will be automatically released when they fall out of scope and are no longer needed. This eliminates memory leaks, dangling pointers, etc. Amazing isn't it?

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Example of C++ Data Types

```
std::vector<BYTE> fileContents{}; /* and empty vector */

std::string filename{}; /* an empty string */
std::string filename{ R"(c:\flag.txt)"}; /* raw literal string; escapes not needed */

std::string_view svFileName{ filename }; /* a view of filename */

/* cast the address of CloseHandle to the CloseHandle type */
/* this is a function pointer to CloseHandle */
auto pfnCloseHandle{ std::bit_cast<decltype(CloseHandle)*>(GetProcessAddress()) };

CStringA csMessage{ "" };

auto pSmart{ std::make_shared<DWORD>( 42UL ) };

auto pBestClass{ std::make_unique<DWORD>( 670UL ) }; /* unique DWORD pointer */
*pBestClass = 770UL; /* dereference the smart pointer */
auto pRawPtr = pBestClass.get(); /* obtain a raw pointer */
```


Malaris OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Handle Data Types



Handles are used to reference system objects.

A user application cannot directly access system objects, so they retrieve a handle to the object. The application can use the handle to modify the object. All handles are entered into an internal handle table that hold the addresses of the resources and the resource type.

- `LPHANDLE`
- `HRSRC`
- `HKEY`
- `HINSTANCE`

© SANS Institute 2024

https://linktr.ee/offsecexam

Handle Data Types Defined



Handles are defined in the WinNt.h header file.

```
typedef PVOID HANDLE;

typedef HANDLE* LPHANDLE;

typedef HANDLE HRSRC;

typedef HANDLE HKEY;

typedef HANDLE HINSTANCE;

typedef HINSTANCE HMODULE;
```


Tomarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

WINAPI, APIENTRY, CALLBACK



These data types are used for decorating functions that use `__stdcall`

WINAPI, APIENTRY, and CALLBACK are utilized when defining a function. Looking at some of the Windows header files, you will see these types scattered around right before a function body.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

WINAPI, APIENTRY, CALLBACK Defined



WINAPI, APIENTRY, and CALLBACK are defined in WinDef.h header file

```
#define WINAPI __stdcall

#define APIENTRY WINAPI

#define CALLBACK __stdcall
```


Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: WINAPI, APIENTRY, CALLBACK

```
BOOL WINAPI EnumProcesses(
             _Out_writes_bytes_(cb) DWORD* lpidProcess,
             _In_ DWORD cb,
             _Out_ LPDWORD lpcbNeeded );

WINADVAPI LSTATUS APIENTRY RegCreateKeyA (
                     _In_ HKEY hKey,
                     _In_opt_ LPCSTR lpSubKey,
                     _Out_ PHKEY phkResult );

typedef UINT (CALLBACK *LPFNPSPCALLBACKA)(
                     HWND hwnd,
                     UINT uMsg,
                     struct _PROPSHEETPAGEA *ppsp );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Learned you must become familiar with the Windows data types
- Discussed how Windows data types could prevent future headaches
- Learned some data types just stem from others; HINSTANCE and HMODULE
- Discussed how seeing the definitions can help understand something better
- Learned some data types are interchangeable; HINSTANCE and HMODULE

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What does the type WINAPI expand as?

A __stdcall

B __thiscall

C __cdecl

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What does the type WINAPI expand as?

A __stdcall

B __thiscall

C __cdecl

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What do the types HKEY, HINSTANCE, HRSRC have in common?

A. Nothing.

B. They are all of type HANDLE.

C. They all refer to GUI applications.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What do the types HKEY, HINSTANCE, HRSRC have in common?

A Nothing.

B They are all of type HANDLE.

C They all refer to GUI applications.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the root type for DWORD?

A. ULONG or unsigned long.

B. VOID or void.

C. WORD or double word.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the root type for DWORD?

A	ULONG or unsigned long.
B	VOID or void.
C	WORD or double word.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



On modern systems, what is the difference between LPVOID and PVOID?

A. There is no difference.

B. LPVOID is a long pointer, PVOID is not.

C. LPVOID is not a pointer.

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



On modern systems, what is the difference between LPVOID and PVOID?

A. There is no difference.

B. LPVOID is a long pointer, PVOID is not.

C. LPVOID is not a pointer.

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

© SANS Institute 2024

https://linktr.ee/offsecexam

__stdcall



The standard calling convention for Windows API functions



For functions with fixed number of arguments



Callee cleans the stack before return



Offers speed and efficiency



Arguments pushed right to left

Tamaris OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: __stdcall (x86)

```
CreateFileW( L"hello.txt", GENERIC_WRITE, 0, NULL, CREATE_NEW,
       FILE_ATTRIBUTE_NORMAL, NULL );

push  0                     ; hTemplateFile
push  80h                   ; dwFlagsAndAttributes
push  1                     ; dwCreationDisposition
push  0                     ; lpSecurityAttributes
push  0                     ; dwShareMode
push  40000000h                     ; dwDesiredAccess
push  offset FileName               ; "hello.txt"
call  ds:__imp__CreateFileW@28      ; CreateFileW(x,x,x,x,x,x,x)
mov   edi, eax                      ; caller does no clean up
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: (x64)

```
CreateFileW( L"hello.txt", GENERIC_WRITE, 0, NULL, CREATE_NEW,
       FILE_ATTRIBUTE_NORMAL, NULL );

mov      qword ptr ss:[rsp+30], 0            ; hTemplateFile on stack
lea      rcx, qword ptr ds:[7FF633682230]    ; ptr to L"hello.txt"
mov      dword ptr ss:[rsp+28], 80           ; FILE_ATTRIBUTE_NORMAL on stack
xor      r9d, r9d                            ; 0 lpSecurityAttributes
xor      r8d, r8d                            ; 0 dwSharedMode
mov      dword ptr ss:[rsp+20], 1            ; CREATE_NEW on stack
mov      edx, 40000000                       ; GENERIC_WRITE
call     qword ptr ds:[<&CreateFileW>]
```

© SANS Institute 2024

https://linktr.ee/offsecexam

__cdecl



The default calling convention for standard C functions



Still used on Windows



Caller cleans the stack after return



Callee has no responsibility



Arguments pushed right to left

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: __cdecl

```
printf( "Score: %d, Avg: %d, High: %d, Low: %d\n", 670, 92, 670, 660 );

push  294h            ; 660
push  29Eh            ; 670
push  5Ch             ; 92
push  29Eh            ; 670
push  offset _Format ; "Score: %d, Avg: %d, High: %d, Low: %d\n"
call  _printf
add   esp, 14h        ; caller cleans up the stack
```


Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

__fastcall



The faster calling convention, seriously.



First 2 arguments passed in registers



Remainder arguments on stack



Fastest when 2 arguments used



Arguments pushed right to left

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: __fastcall

```
INT FASTCALL SomeFastFunction( PDWORD, PDWORD, BOOL);

------Main------
push  1                     ; Passed
lea   edx, [ebp+dwAge]      ; Age
lea   ecx, [ebp+dwDollars]  ; Dollars
call  @SomeFastFunction@12  ; SomeFastFunction(x,x,x)
mov   [ebp+Age], eax        ; no clean up

-------SomeFastFunction-------
[..snip..]

retn  4           ; cleaning up the stack
```


Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

__thiscall



This is yet another Windows-specific calling convention for x86 C++.



Compiler creates a ***this*** pointer



All arguments are on the stack

ECX

Primary holder of ***this*** pointer



Arguments pushed right to left



For C++ class members

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: __thiscall

```
class MyClass { public: void PrintMsg( char* msg ); };

// this is really the same as above
struct TheClass {void TheMessage( char* msg ); };

// defined separately with the scope resolution operator ::
void MyClass::PrintMsg( char* msg ){ printf( msg ); };
```


Tanrisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.4: Call Me Maybe



Learn how to use the various calling conventions.

Please refer to the eWorkbook for the details of this lab.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Learned it's important to know what is happening to the stack/registers
- Learned every convention uses the stack for arguments; ***`fastcall`***
- Discussed that the default for Win32 APIs is ***`stdcall`***
- Discussed that the default for C/C++ is ***`cdecl`***
- Learned the C++ x86 class member functions will use ***`thiscall`***

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



For x86, how are arguments passed to functions?

A. In registers RCX, RDX, R8, R9

B. On the stack

C. In the heap

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



For x86, how are arguments passed to functions?

A. In registers RCX, RDX, R8, R9

B. On the stack

C. In the heap

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What calling convention primarily uses registers ECX/RCX and EDX/RDX?

A __fastcall

B __thiscall

C __stdcall

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What calling convention primarily uses registers ECX/RCX and EDX/RDX?

A __fastcall

B __thiscall

C __stdcall

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



For 64-bit, why do stack arguments start at RSP+20h and not RSP?

A. The first 20h bytes are reserved as a shadow stack enforcement

B. The first 20h bytes are reserved as a shadow store

C. They don't, this is a trick question

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



For 64-bit, why do stack arguments start at RSP+20h and not RSP?

A	The first 20h bytes are reserved as a shadow stack enforcement
B	The first 20h bytes are reserved as a shadow store
C	They don't, this is a trick question

Tanisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

© SANS Institute 2024

https://linktr.ee/offsecexam

What Are They?



Microsoft source-code annotation language (SAL) annotations

The source-code annotation language might seem like another language you will have to learn, but it's important to know its purpose. Microsoft uses it practically everywhere in its source code.

Definitions are found in the Sal.h header file.

© SANS Institute 2024

https://linktr.ee/offsecexam

Why Use Them?



Automatic static code analysis built into Visual Studio

SAL annotations make the intended use of function parameters clear

Very inexpensive compiler check for your code

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Basic Annotations (1)

Annotations could be categorized by level of complexity: basic, intermediate, advanced.

_In_	Read-only data provide to a function
Out	Output written to address space provided by the caller
Inout	Input to called function and output to caller; read-write data

Tanisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Basic Annotations (2)

_In_opt_	Inbound pointer that could also be NULL
_Out_opt_	Output to the caller that is optional
_Ret_z_	The function will return a null-terminated value, like a string

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Examples: Basic Annotations

```
WINBASEAPI
BOOL
WINAPI
CreateProcessA(
    _In_opt_ LPCSTR lpApplicationName,
    _Inout_opt_ LPSTR lpCommandLine,
    _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_ BOOL bInheritHandles,
    _In_ DWORD dwCreationFlags,
    _In_opt_ LPVOID lpEnvironment,
    _In_opt_ LPCSTR lpCurrentDirectory,
    _In_ LPSTARTUPINFOA lpStartupInfo,
    _Out_ LPPROCESS_INFORMATION lpProcessInformation
    );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Intermediate Annotations (1)

_In_reads_(s) _In_reads_bytes_(s)	When the parameter could have 1+ elements to read where (s) is the number of elements or bytes when _bytes_ is used
_In_reads_z_(s) _In_reads_or_z_(s)	The parameter is null-terminated and possibly 1+ elements should be read
_Out_writes_(s) _Out_writes_bytes_(s)	An out parameter that will write 1+ elements or bytes if _bytes_ is used

© SANS Institute 2024

https://linktr.ee/offsecexam

Intermediate Annotations (2)

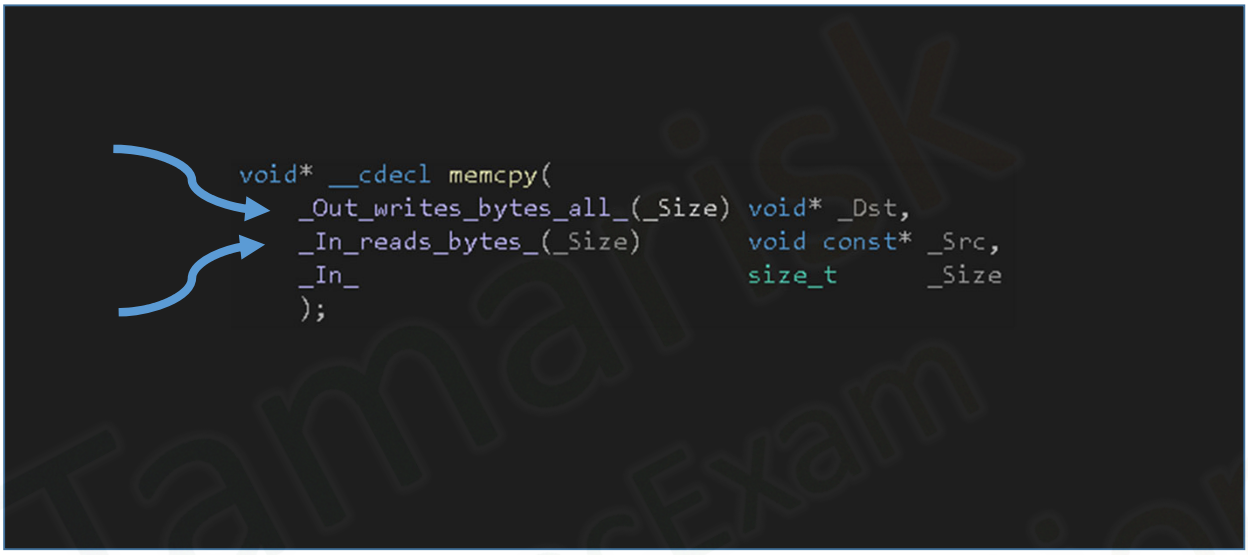
_Out_writes_bytes_all_(s)	All of 's' bytes will be written out
_Ret_maybenull_	Function's return value will, could, or won't be a nullptr

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

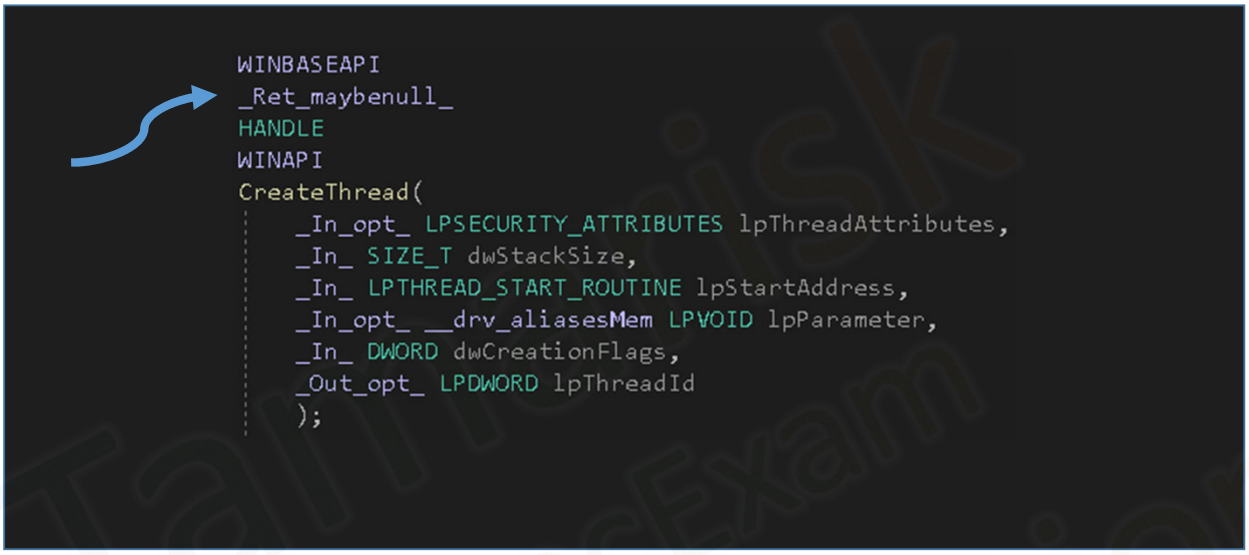
Example: Intermediate Annotations (1)



© SANS Institute 2024

https://linktr.ee/offsecexam

Example: Intermediate Annotations (2)



```
WINBASEAPI
_Ret_maybenull_
HANDLE
WINAPI
CreateThread(
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_ SIZE_T dwStackSize,
    _In_ LPTHREAD_START_ROUTINE lpStartAddress,
    _In_opt_ __drv_aliasesMem LPVOID lpParameter,
    _In_ DWORD dwCreationFlags,
    _Out_opt_ LPDWORD lpThreadId
    );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Advanced Annotations (1)

_Inout_updates_(s) _Inout_updates_bytes_(s)	Accepts a pointer to an array of elements read and written to by the function.
_Check_return_ _Must_inspect_result_	The caller must check the return value of the function.
_Use_decl_annotations_	Forces the reuse of a function's annotations for a function's declaration to avoid the duplication of them.

© SANS Institute 2024

https://linktr.ee/offsecexam

Advanced Annotations (2)

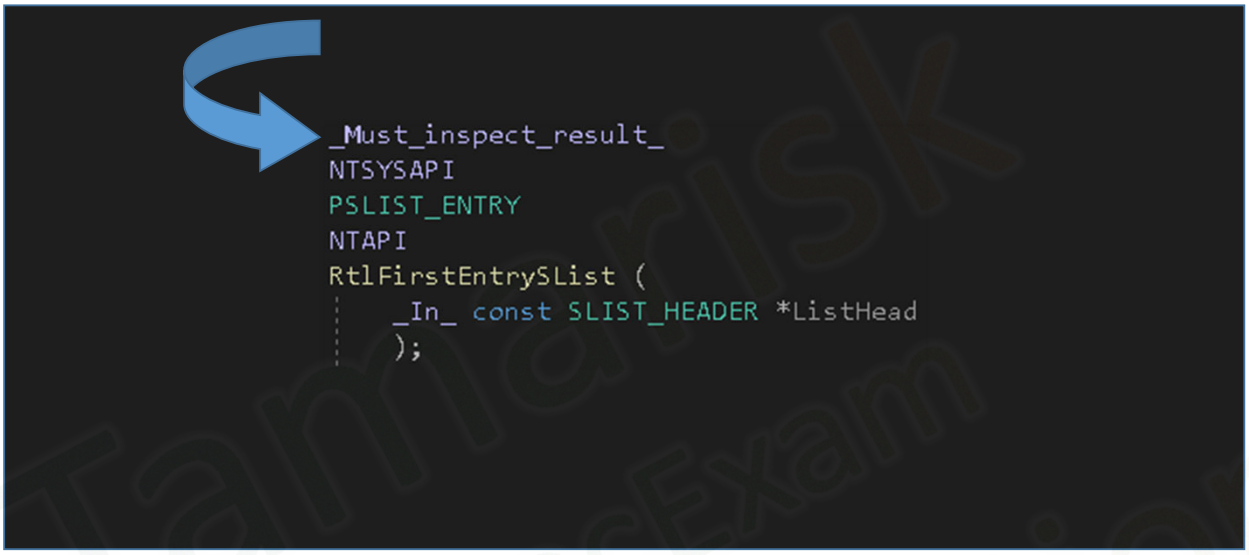
Success(*expr*)	Functions can fail or succeed, but success can be determined in the form of an expression.
When(*expr*, anno-list)	When the expression results TRUE the anno-list applies for the parameter, otherwise it's ignored.
_Ret_writes_to_(s, c)	Describes a return value that is a scalar, pointer to a struct, or pointer to a buffer. 's' is number of elements and 'c' is the number of elements written.

Tam Nguyen OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: Advanced Annotations (1)

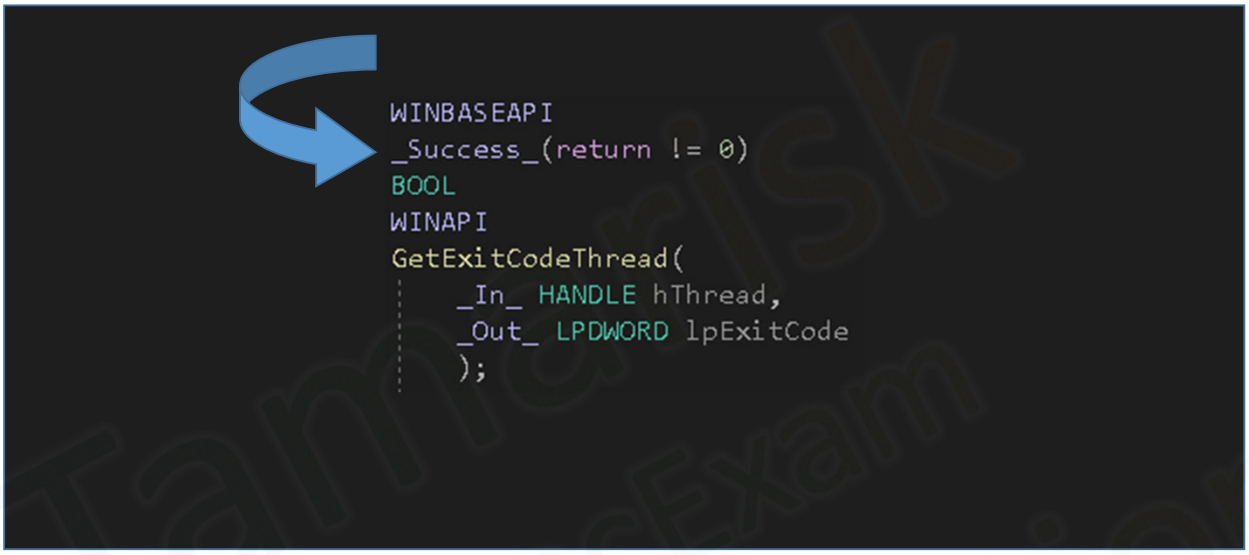


```
_Must_inspect_result_
NTSYSAPI
PSLIST_ENTRY
NTAPI
RtlFirstEntrySList (
    _In_ const SLIST_HEADER *ListHead
    );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: Advanced Annotations (2)



```
WINBASEAPI
_Success_(return != 0)
BOOL
WINAPI
GetExitCodeThread(
    _In_ HANDLE hThread,
    _Out_ LPDWORD lpExitCode
    );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: Advanced Annotations (3)

```
WINBASEAPI
_Success_(return != FALSE)
BOOL
WINAPI
InitializeProcThreadAttributeList(
    _Out_writes_bytes_to_opt_(*lpSize,*lpSize) LPPROC_THREAD_ATTRIBUTE_LIST lpAttributeList,
    _In_ DWORD dwAttributeCount,
    _Reserved_ DWORD dwFlags,
    _When_(lpAttributeList == nullptr,_Out_) _When_(lpAttributeList != nullptr,_Inout_) PSIZE_T lpSize
    );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: Advanced Annotations (4)

```
STDAPI
WldpQueryDynamicCodeTrust(
    _When_(baseImage != NULL, _In_opt_)
    _When_(baseImage == NULL, _In_)
    HANDLE fileHandle,
    _When_(fileHandle == NULL, _In_reads_bytes_opt_(imageSize))
    _When_(fileHandle != NULL, _In_reads_bytes_(imageSize))
    PVOID baseImage,
    _In_ ULONG imageSize
    );
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.5: Safer with SAL



Using SAL annotations makes your code more understandable.

Please refer to the eWorkbook for the details of this lab.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Additional Information



Microsoft is your best source for understanding SAL annotations.

Header files can serve to be a great example of how to use certain annotations.

You can use PowerShell to search for specific annotations.

```
Select-String -Pattern "_When_\(" -Path . \*.h -CaseSensitive
```


Tanisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed how SAL annotations can really assist with making code safer
- Learned some annotations can become complex very quickly
- Learned some annotations only apply for a function
- Covered how Visual Studio uses it for static code analysis

Tanish OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What does SAL stand for?

A. Source-code annotation language

B. Structured annotation language

C. Silent analysis language

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What does SAL stand for?

A. Source-code annotation language

B. Structured annotation language

C. Silent analysis language

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What annotation would you use for pass by reference parameters?

A _In_

B _Out_

C _Inout_

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What annotation would you use for pass by reference parameters?

A _In_

B _Out_

C _Inout_

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



Using the _When_ annotation, make a pointer parameter become _Out_.

A _When_(nullptr, _Out_)

B _When_(_Out_, lpPointer == LPVOID())

C _When_(lpPointer == nullptr, _Out_)

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



Using the _When_ annotation, make a pointer parameter become _Out_.

A	_When_(nullptr, _Out_)
B	_When_(_Out_, lpPointer == LPVOID())
C	_When_(lpPointer == nullptr, _Out_)

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

Tanay Offsec Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows API



The Windows Application Programming Interface

Provides nearly all functionality for the Windows OS to include userland and kernel land functionality. API names are very descriptive and can be very lengthy too. Heavily modified/updated APIs have "Ex" appended to the name.

APIs make creating shellcode more complex.

PrefixVerbTarget[Ex] / VerbTarget[Ex]

RegCreateKeyEx

© SANS Institute 2024

https://linktr.ee/offsecexam

Create APIs (1)



Objects and handles make the world go around.

There are at least 96 Windows APIs that start with "Create." Their main purpose is to help a user application create a system object. A Handle is typically returned that references the newly created object.

CreateProcess is one exception; its return value is of type BOOL

© SANS Institute 2024

https://linktr.ee/offsecexam

Create APIs (2)

- `CreateFile`
- `CreateFileMapping`
- `CreateDirectory`
- `CreateTimerQueue`
- `CreateMutex`
- `CreateEvent`
- `CreateThread`
- `CreateToolhelp32Snapshot`

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Create APIs (3)



`CreateProcessW`

Used to create a new process

Has a Boolean return type

```
BOOL CreateProcessW(
  _In_opt_ LPCWSTR pApplicationName,
  _Inout_opt_ LPWSTR pCommandLine,
  _In_opt_ LPSECURITY_ATTRIBUTES pProcAttrs,
  _In_opt_ LPSECURITY_ATTRIBUTES pThrdAttrs,
  _In_    BOOL   bInheritHandles,
  _In_    DWORD  dwCreationFlags,
  _In_opt_ LPVOID  pEnvironment,
  _In_opt_ LPCWSTR pCurrentDirectory,
  _In_    LPSTARTUPINFO         pStartupInfo,
  _Out_   LPPROCESS_INFORMATION lpProcInfo
);
```

© SANS Institute 2024

Just so you don't have to flip back several pages, here is the search order again in order.

1. Directory of the calling executable
2. Current directory of the process
3. System directory (this is returned by ***GetSystemDirectory***)
4. Windows directory (this is returned by ***GetWindowsDirectory***)
5. Directories found in %PATH%

A noticeable difference between the first two parameters is the type. *lpCommandLine* is of type LPWSTR, which is not a constant like for *lpApplicationName*, LPCWSTR. Constant are treated as read-only and because of this, the *CommandLine* must be available to modification by the function. In other words, the function will read and possibly write to the buffer that holds the image name. So, be sure to place the string in dynamic memory or on the stack, which should always be read/write.

lpProcessAttributes and *lpThreadAttributes* are the same type, which is a pointer to a SECURITY_ATTRIBUTES structure. NULL is typically passed here, but if you want the handle to have the inheritable attribute, then they cannot be NULL.

bInheritHandles is just a flag to indicate that inheritable handles will be inherited by a child process if one is created.

dwCreationFlags describe how the process is to be created. There are over a dozen flags that can be used in combination with others. One example is the CREATE_SUSPENDED flag that will place the primary thread in a suspended state that will not run until your code calls ***ResumeThread***. This technique will be explored later in the course.

lpEnvironment is a pointer to the environment block the new process should use. NULL is just fine here so the process can use the environment of the process that called it.

lpCurrentDirectory is the full path of the current directory and it can also be a UNC path. NULL is fine here too, and the new process will use the same drive and directory as the calling one.

lpStartupInfo is a pointer to one of two structures: STARTUPINFO or STARTUPINFOEX, which is just an extended version of the former.

lpProcessInformation is a pointer to a PROCESS_INFORMATION structure that will be filled out with information about the process once it's created. The handles that are in this structure need to be closed once no longer needed by your program, CloseHandle.

As mentioned previously, the function can return before the primary thread, or process, has had a chance to fully initialize itself. So, be sure to take this into account when you are developing your programs.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: CreateProcess

```
STARTUPINFO si = { sizeof si };
PROCESS_INFORMATION pi;
WCHAR commandLine[] = L"Notepad";

CreateProcess(NULL,
    commandLine,
    NULL,
    NULL,
    FALSE,
    0,
    NULL,
    NULL,
    &si,
    &pi);
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Create APIs (4)



PROCESS_INFORMATION

Holds information about the new process and its primary thread

```
typedef struct
_PROCESS_INFORMATION {
 HANDLE hProcess;
 HANDLE hThread;
 DWORD  dwProcessId;
 DWORD  dwThreadId;
} PROCESS_INFORMATION,
*PPROCESS_INFORMATION,
*LPPROCESS_INFORMATION;
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Create APIs (5)



`CreateToolhelp32Snapshot`

Used to create a snapshot of a process

Also, can be used for heaps, threads, and loaded modules

```
HANDLE CreateToolhelp32Snapshot(
 _In_ DWORD dwFlags,
 _In_ DWORD th32ProcessID
);

BOOL Process32First(
 _In_ HANDLE hSnapshot,
 _Out_ LPPROCESSENTRY32 lppe
);

BOOL Process32Next(
 _In_ HANDLE hSnapshot,
 _Out_ LPPROCESSENTRY32 lppe
);
```

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: CreateToolhelp32Snapshot

```
DWORD lastError = 0;
HANDLE snapShot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);

PROCESSENTRY32 pe32 = { 0 };
pe32.dwSize = sizeof PROCESSENTRY32;

printf("%20s %7s\n", "Process Name", "PID");
printf("..........................................\n");
do {
    printf("%20ws", pe32.szExeFile);
    printf("%8d\n", pe32.th32ProcessID);
} while (Process32Next(snapShot, &pe32));

CloseHandle(snapShot);
return ERROR_SUCCESS;
```


Example: CreateToolhelp32Snapshot

The example here intentionally omits error checking due to size limitations on the slide. Regardless, the main points are represented starting with the call to the ***CreateToolhelp32Snapshot*** API on the second line. We are only interested in capturing processes in the snapshot and we are not specifying a process ID as indicated by NULL (0). The function returns a handle value so that is saved off in the snapShot variable. In the full code, this would be checked against **INVALID_HANDLE_VALUE** to ensure the snapshot was successfully created. The next couple of lines create the pe32 variable of struct type **PROCESSENTRY32** and since it has a size field it should be set to the size of the struct. The next important part is the do while loop. What is not shown is a check for the function call ***Process32First***(snapShot, &pe32) to make sure that the first process in the snapshot can be retrieved. After that succeeds the loop can be entered. The meat of the loop is simply printing out the name of the executable and the corresponding process ID. The exit condition of the loop is met when the ***Process32Next*** API returns FALSE since it is a BOOL return type. To understand the process even more and what other information is available, research the **PROCESSENTRY32** definition in the TlHelp32.h header file.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Objects (1)



System resources represented as data structures in system address space

The Windows kernel does the hard work creating an object often at the request of a user application. There are over 4,000 object types that the Windows executive implements, some of which are not accessible via Windows APIs.

Files, images, threads, registry keys, and processes are several object types.

Windows Objects (1)

Objects in Windows are simply the system's way of representing system resources via a structured method. System resources should not be directly accessible by anyone, especially anyone from user space, thus secure access to them is maintained and enforced. To help the system with creating, deleting, and managing objects, there is the object manager, which is an executive module. When a user mode application calls a function like ***CreateFile*** or ***CreateThread***, a request is actually being made to create an object, or a system resource that represents that file or thread. The object manager gives the system a common interface for utilizing those created system resources. Objects should not be destroyed until all processes are done using it, and the object manager helps with that by providing rules that determine how long objects are retained.

There are officially three kinds of objects: kernel objects, executive objects, and user objects. Various executive components like the memory manger or the I/O manager will implement executive objects. The kernel objects are implemented by none other than the kernel itself, and as such, they are not available to user mode applications, only the executive.

Reference:

Mark Russinovich, David A. Solomon, and Alex Ionescu, *Windows® Internals, Part 1*, *6th Edition* (Washington, Microsoft Press, 2012), chap. 3.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Objects (2)

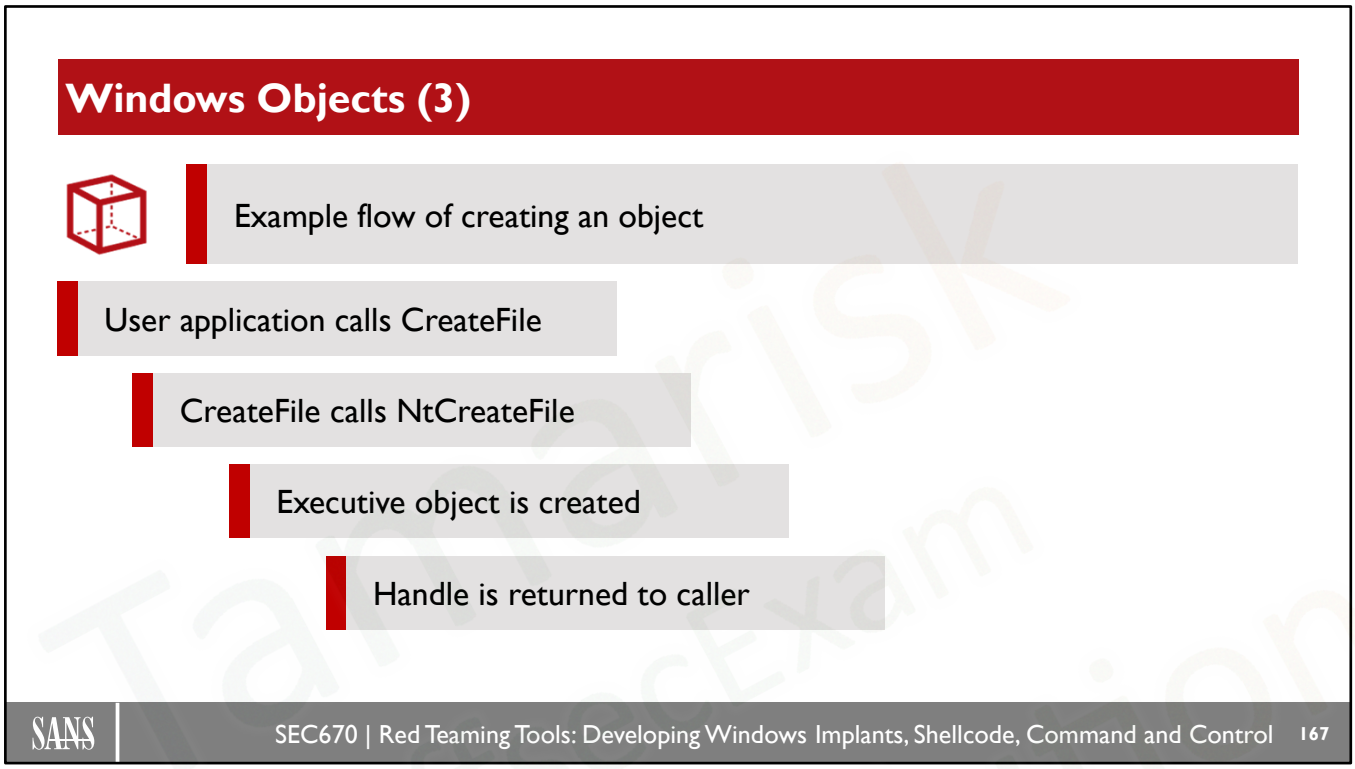
Object Type	Description
Process	Virtual address space that controls execution of thread object(s)
Thread	The executable portion of a process
Section	Shared memory, file-mapping object
Token	Security profiles for threads/processes
Mutex	Method of synchronization for serialized access
Key	Used to refer to data in the Registry
Desktop	An object within a window station

Windows Objects (2)
There are many types of objects on a Windows system, and the types listed on the slide are a small subset of the types of objects. The full listing can be found by running the WinObj utility from Sysinternals and browsing the ObjectTypes folder. For example, there are kernel objects, such as device and driver objects, that are utilized when developing kernel drivers. Also listed are process, job, and thread object types that are typically created at the request of user mode applications. Taking an example from the slide, the process object type represents a virtually contained address space for the execution of thread objects.

Reference:
Mark Russinovich, David A. Solomon, and Alex Ionescu, *Windows® Internals, Part 1*, *6th Edition* (Washington, Microsoft Press, 2012), chap. 3.

© SANS Institute 2024

https://linktr.ee/offsecexam



Windows Objects (3)

There are many reasons why a user mode application might want to create a file. Perhaps the program is creating an error log file to maintain a record of any errors that are encountered during its execution time frame. The application will need an object representing the file that it requested the kernel create for it. The first thing that happens is the application must make a call to the ***CreateFile*** API. ***CreateFile*** is implemented from Kernelbase.dll as one part of the Windows subsystem and once everything checks out, ***NtCreateFile*** will be called. The kernel will eventually be invoked via a system call, and it will create the file object, specifically an executive file object, and return a handle to that object. The handle should also be placed as an entry in the process' handle table.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Objects (4)



Every object has the same structure

This means that there can be one portion of the system that manages all objects. The appropriately named object manager has the role of maintaining all objects.

object header

- type
- name
- directory
- security descriptor
- handle count and list
- optional subheaders

object body

- unique to the object type

Windows Objects (4)

The object manager can perform several tasks, such as following:

- Create objects and validate that a process has the rights to use that object.
- Create the object handles and return them to the caller.
- Create duplicates of existing handles.
- Close handles to objects.

Each object is going to have a header and a body. The object header is controlled via the object manager and has a static size. The object body is controlled via the owning executive components and the size will vary based on the object type.

The structure of the object's header has more information than what fits in the slide, so the important portions are noted. Other items to note are the handle count (list) and type. The system needs to keep count of how many handles are currently open to the object, so it knows when to destroy it. The type is simply an index of object types found in a table named *ObTypeIndexTable*. Although each object header is structured the same way, they will undoubtedly contain different information with each instance of that object. The object name and security descriptors would be the most noticeable differences. The handle list is simply a list of open handles to that object. There could be several processes that are using that object via a handle.

Unlike the object header, the body of an object must be unique because there are several object types. However, objects that are of the same type will have the same object body format. If you wanted to peek into the internals of the object headers and type objects, they can be viewed using a kernel debugger.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Objects (5)



Objects have services that can operate on them.

The Windows subsystems makes these services available to Windows applications. All objects, regardless of type, support several generic services. In addition, each object will have its own services like create, open, and query.

Close, duplicate, query/set security, wait for single object, duplicate, etc.

Windows Objects (5)

With the standardization of object headers and sub headers, the object manager can provide applications with a small number of generic services. The generic services, being applied to each object regardless of type, can be executed with the corresponding APIs such as ***CloseHandle***. In addition, each object will have its own services like create, open, and query. We can go with one example for the creation of a file. We saw the overall flow of the process from a few slides back but diving into it a bit more here, the I/O system implements a service that allows an application to create a file object. The same thing applies for the creation of process objects. The process manager, another executive component, implements its create process service that allows an application to create a process object. Even though objects might be similar to each other, the routines used to create them are vastly different. This makes sense because the process of creating a file is vastly different from the process of creating a process. The object headers would be structured the same, but definitely not the object bodies. Furthermore, it would not make sense to suspend or terminate file objects the same way that you can terminate thread and process objects.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Objects (6)



Objects can leverage the security of Windows.

Objects that are exposed to user mode must be protected. Objects will have their own access control list (ACL) that dictates what actions can be performed on the object from a querying process. Securable objects have security descriptors, and the system acts as the gatekeeper to the objects.

"You shall not pass!"

Windows Objects (6)

Objects must be secured or protected from malicious abuse or unauthorized access. Whenever an object is exposed directly to the user, it must be protected. Objects that are not directly exposed do not needed protections like kernel objects (driver objects). In a similar fashion to how files are protected on the file system, objects are protected too, but via security descriptors. The object manager acts as the gatekeeper to objects and controls who accesses them. When a process wishes to obtain a handle to an object, say for read and write access, the object manager steps in to see what is happening. The object manager, with the help of the kernel, determines what access rights will be granted to that object. The desired accesses are requested by the user mode application via flags like GENERIC_READ or GENERIC_WRITE.

A process' handles to objects can be viewed using Process Explorer from Sysinternals. Once a process is selected, the handles can be observed, but if nothing is showing, be sure to unhide the lower pane from the toolbar. A handle, or object, can be selected and the properties can be observed after right-clicking on it.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Handles (1)



Handles act as the mechanism to interact with objects.

Named objects that are created will have handles to them. The handle is what the application needs in order to interact directly with the object.

- Always a multiple of 4
- Never handle value of 0
- First valid handle is always 4
- 32-bit / 64-bit handle values

Windows Handles (1)

Handles are the mechanism in place that allow a user mode application to interact with an object. When an object is created with a name, a named object, the object manager will provide the requester with a handle to it. All handles are the same regardless of the type of object they have a handle against. For example, a handle to a process is the same as a handle to a registry key or thread or file. This makes the job of reference counting easier for the object manager.

When it comes to handle values, they will always be some multiple of 4 and because of this, one thing you will never see is a handle value of 0. Therefore, the first valid handle value could only ever be 4. Handles will either be 32-bit values on 32-bit Windows and 64-bit values on 64-bit Windows. Throughout the remainder of this course, you will become extremely familiar with handles and how to pass them around to other functions that require them.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Handles (2)



Handle tables store handle table entries.

Each process will have its own handle table.

The handle is an index into the handle table.

Various APIs require a valid object handle to manipulate it.

Windows Handles (2)

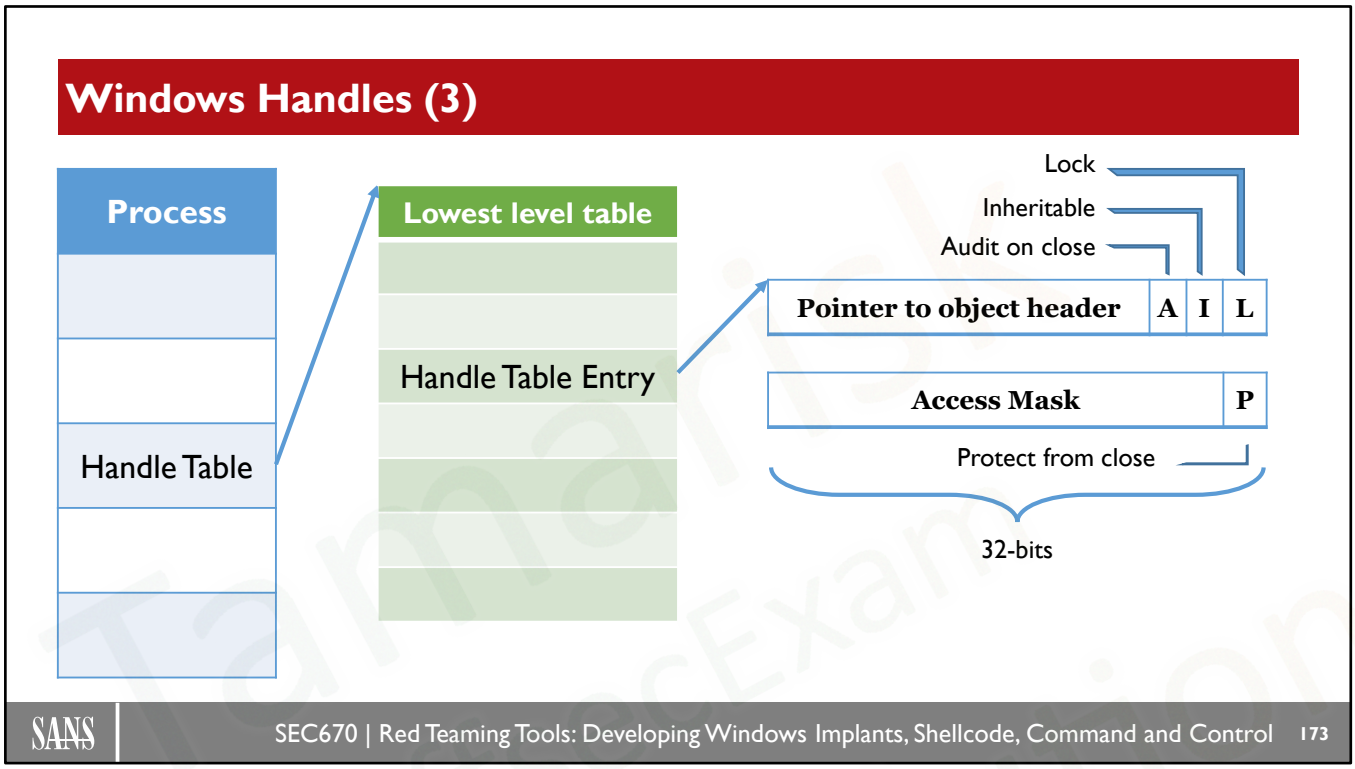
As mentioned previously, handles are the direct result of the creation of named objects. There are several APIs that can create objects and handles (they were also mentioned earlier), but one such API is ***CreateProcess***. When a process is created, the system will create a handle table unique to that process. The table stores handles that reference various objects like threads or other objects that threads could open handles against. As far as the application is concerned, the handle table is opaque.

Once the correct handle table entry is located, the pointer to the object can be followed. Along with the pointer to the object, the access is also stored to protect the object. If a process or thread attempted a write operation to an object, but the handle was opened with only read access, the operation would be happily denied by the object manager. It is best to open a handle with the minimum access required to do the job. As an example, calling ***`OpenProcess`***`(PROCESS_ALL_ACCESS, TRUE, `***`GetCurrentProcessId()`***`)` would return a handle with all access to the current process. Even worse, the handle can be inherited by any child process.

No matter how hard you might try, handles cannot be directly created with an API, but they can be closed (***CloseHandle***), compared (***CompareObjectHandles***), or duplicated (***DuplicateHandle***). Also, the information about a handle can be retrieved via ***GetHandleInformation***. Certain handle properties can also be modified via ***SetHandleInformation***.

© SANS Institute 2024

https://linktr.ee/offsecexam



Windows Handles (3)

Somewhere in the process' virtual address space will be a pointer to its handle table. Just like how there are multiple tables involved with the translation of virtual addresses to physical addresses, there are multiple tables for handles. This is how a process can have a large number of handles, upwards of 16,000,000. Not all of the three tables are shown on this slide due to space limitations, but the lowest level table is the only one created right when a new process is initialized. The other table levels are made when there is a need.

The number of entries is dependent on the architecture being used. As an example, x86 machines have a page size of 4096 bytes. The size of a handle table entry is 8 bytes (32-bit pointer). Dividing the size of a page by the size of the table entry would yield 512 - 1 = 511. Each handle table entry is a structure and on x86 systems, both members of the structure are 32-bits in size. For x64, the handle table entry is 12 bytes in size, and it will have a 64-bit pointer followed by a 32-bit access mask.

The structure has a pointer to the object's header along with four flags: lock, inheritable, audit on close, and protect from close. The lock bit indicates if the entry is currently in use. The inheritance flag indicates that child processes will inherit the handle and be stored in its process handle table. The third flag, audit on close, indicates if an audit message should be created when the object is closed. The last flag, protect from close, indicates if the caller should be allowed to close the handle. The protect from close flag can be set with the ***NtSetInformationObject*** call.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Handles (4)



Access rights for processes and threads

Access Flag (Constant)	Description
PROCESS_ALL_ACCESS	Give all access rights that are possible for a process object
THREAD_ALL_ACCESS	Give all access rights that are possible for a thread object
PROCESS_CREATE_PROCESS	Gives permissions to create a process
PROCESS_CREATE_THREAD	Gives permissions to create a thread
PROCESS_DUP_HANDLE	Gives permissions to duplicate a handle

Windows Handles (4)

Handles can have various permissions and some of them can be great for abuse when the right conditions are set. The table of access rights on the slide is not an exhaustive list by any means, but some were noteworthy to mention. Handles that have ***_ALL_ACCESS** are needed for a process or thread to act upon itself via a pseudo-handle to itself. For example, ***GetCurrentProcess*** does not return a real handle, but rather a pseudo-handle with a value of (HANDLE)-1. ***GetCurrentThread*** also returns a pseudo-handle to itself, (HANDLE)-2. The kernel does some checks and will grant the appropriate access, **PROCESS_ALL_ACCESS** or **THREAD_ALL_ACCESS**. The ***_CREATE_*** access flags allow you to create a process or a thread. The **PROCESS_DUP_HANDLE** will allow you to duplicate a handle. A common example of duplicating handles is duplicating STD handles into a child process. Say, the parent process is a service that cannot print anything to console handles like STDOUT so it can create a child process and duplicate those handles into it.

© SANS Institute 2024

Windows Handles (5)



Handles can be leaked.

Handles are considered to be *leaked* if they are not closed when the application is done with it. If you were to see a long list a handles to the same object that aren't being closed, chances are you might have leak in your program.

Process Explorer

SysInternals handle.exe

```
CHAR shellcode[] = "\xCC\xCC";
PVOID buff = VirtualAllocEx(process, NULL, 2, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(process, buff, shellcode, 2, NULL);
CreateRemoteThread(process, NULL, 0, buff, 0, 0, NULL);
```


Windows Handles (5)
When a developer is not careful to promptly close handles when they are no longer needed, they are considered to be leaked handles. There are various tools out there that let you view open handles and Process Explorer is one such tool. If you notice a large list of handles to the same object that have not been closed, then chances are pretty high that you are leaking handles. It is important to identify and correct your own leaked handles before someone like James Forshaw does and further proceeds to create a novel way to exploit it.

The highest access that could come with a handle would be specified with a flag like **PROCESS_ALL_ACCESS** or **THREAD_ALL_ACCESS**. If you created a SYSTEM level service or anything higher than a standard user application, and you called ***OpenProcess(*PROCESS_ALL_ACCESS, TRUE, *GetCurrentProcessId())***, you just created a possible vulnerability to be exploited. The all access will literally give you an all-access pass to do pretty much whatever you want in that process' address space; you could create your own thread and execute shellcode!

There is also a command-line tool from SysInternals called "handle.exe" that shows the same information. You can run the handle utility on a single process and whatever open handles it has will be printed to the terminal window. It is also not that difficult to create your own version of the Sysinternals handles.exe utility.

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (1)



Your code must handle errors properly.

You never know exactly when an API function might fail. There are many reasons for a failed call and how functions indicate success or failure is far from consistent.

- ***BOOL - FALSE, TRUE***
- ***LSTATUS OR LONG***
- ***HRESULT***
- ***HANDLE***

Handling Errors (1)
API functions can fail at any point in their execution and for a variety of reasons. Perhaps you messed up and passed in an incorrect parameter, or maybe a buffer you needed to have allocated to receive from an OUT parameter was not large enough. Something could also happen internally with the system that would cause your function to fail. The more common failures are either from access permissions or improper use of the function.

Here are some common return types for functions and how they could indicate failure. BOOL functions can indicate failure by having a zero (0) or FALSE be returned. The opposite, not zero, (TRUE (1)) would indicate that the function executed successfully. For BOOL type functions, you should call ***GetLastError*** for more details.

LSTATUS return type functions will return the error number itself back to the caller. This allows for easy lookup and avoids having to call ***GetLastError*** to determine what happened. We have seen LSTATUS types many times today. Anything other than ERROR_SUCCESS (0) is failure.

HRESULT return type functions will indicate success with something that is zero (0) or even something that is greater than zero. Sometimes you might see S_OK (0) in code. A failure is indicated with a negative value, so checking if the value is less than zero in your code should catch any kind of error. The number itself is also the error number.

HANDLE return type functions will indicate success with either not NULL (0) or not INVALID_HANDLE_VALUE (-1). ***GetLastError*** would be useful for this return type.

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (2)



GetLastError

Gets the last error for calling thread

```
// defined in errhandlingapi.h

WINBASEAPI
_Check_return_
_Post_equals_last_error_
DWORD
WINAPI
GetLastError(
    .....
);
```


Handling Errors (2)

The ***GetLastError*** API function does not take a single parameter, as you can see with VOID being specified inside the parentheses. Typically, the best and perhaps only times to call this function are when using functions that have a BOOL return type like ***CreateProcess***. If you are checking to see if a Boolean function failed or succeeded based solely on TRUE / FALSE, you might not get the information you were hoping.

An error will be set when your chosen Windows API function is executing. If you ever get bored and want to explore your reverse engineering side of things, you can reverse some Windows API functions and try to identify the code flow for when they fail. While doing so, you might see calls to ***SetLastError*** or similar. The last error being set is the one you need to get before any other API call is made. If your ***GetLastError*** call is after two consecutive calls, but you were hoping to check the status of the first call, well then too bad. If you want to get the true last error from a Win32 API, then ***GetLastError*** must be called immediately after your API function is called.

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (3)



Checking Boolean return types

```
// defined in minwindef.h
#define TRUE 1
#define FALSE 0

BOOL bStatus = SomeBoolApiFunction();

if ( bStatus == FALSE )
 printf( "Last Error: %d\n", GetLastError() );
```


Handling Errors (3)

Boolean return values could arguably be the easiest return value to check. A simple TRUE / FALSE check can let the operator of the tool know if a function has failed or errored out. In the end, almost every test condition is simply looking for NULL or non-NULL values. True could be indicated by any value that is not NULL (0). However, we will save that for other types because here, we are only concerned with 0 or 1.

If you do not know what your function should return, you could use BOOL instead of VOID. BOOL could be used to indicate that the function successfully executed and return TRUE. Any error that might happen during its execution can return FALSE where the "error" occurred.

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (4)



Checking HRESULT return types

```
// macros are from winerror.h
#define SUCCEEDED(hr) (((HRESULT)(hr)) >= 0)
#define FAILED(hr) (((HRESULT)(hr)) <0)

if ( SUCCEEDED( hResult ) )
 printf( "Success code: %d\n", hResult );

if ( FAILED( hResult ) )
 printf( "Failed with error: %d\n", hResult );
```


Handling Errors (4)

When checking the return values of functions that have a HRESULT type, you can use the SUCCEEDED and FAILED macros that are defined in the winerror.h header file. The main task these macros are doing is determining if the result is less than or equal to zero for success. If not, then the function failed with a negative, or less than zero value.

The interesting part about HRESULT types is that they are made up of four fields:

1. A 1-bit code will indicate the severity; zero is success and one is failure
2. A 4-bit value that is reserved for internal purposes only
3. A 11-bit code to indicate if it is a warning or error, which is also known as the facility code
4. A 16-bit code that describes the warning or error

Windows also provided macros that can interpret the one of the four fields of a HRESULT. For example, if one wanted to pull out the facility, they could use the HRESULT_FACILITY(hr) macro. To determine the severity, one could use the HRESULT_SEVERITY(hr) macro.

The following is a copy/paste from the winerror.h header file that explains the breakdown of the 32-bit value layout.

References:
Microsoft header file winerror.h
https://docs.microsoft.com/en-us/office/client-developer/outlook/mapi/hresult
https://docs.microsoft.com/en-us/windows/win32/seccrypto/common-hresult-values

© SANS Institute 2024

https://linktr.ee/offsecexam

```
// HRESULTs are 32-bit values laid out as follows:
//
//  3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
//  1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
// +-+-+-+-+-+-+---------------------+-------------------------------+
// |S|R|C|N|r|  Facility     |       Code      |
// +-+-+-+-+-+-+---------------------+-------------------------------+
//
// where
//
//   S - Severity - indicates success/fail
//
//     0 - Success
//     1 - Fail (COERROR)
//
//   R - reserved portion of the facility code, corresponds to NT's
//       second severity bit.
//
//   C - reserved portion of the facility code, corresponds to NT's
//       C field.
//
//   N - reserved portion of the facility code. Used to indicate a
//       mapped NT status value.
//
//   r - reserved portion of the facility code. Reserved for internal
//       use. Used to indicate HRESULT values that are not status
//       values but are instead message ids for display strings.
//
//   Facility - is the facility code
//
//   Code - is the facility's status code
```

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (5)



Checking LSTATUS return types

```
// definition from winreg.h
typedef _Return_type_success(return==ERROR_SUCCESS) LONG LSTATUS

// definition from winerror.h
#define ERROR_SUCCESS 0L

LSTATUS lStatus = RegOpenKeyExW(...);

if (lStatus != ERROR_SUCCESS )
 // handle error here
```


Handling Errors (5)

Most of the Reg* family of APIs are defined with LSTATUS return types, so it is important to know how to check them for errors and success. The winreg.h header file defines most of the Reg* APIs but also has a definition for LSTATUS. Here, you will find that it is simply a typedef as a LONG. While highlighted over LSTATUS, you can press F12, and the header file should open that contains its definition. Here is the full definition for LSTATUS. typedef _Return_type_success(return == ERROR_SUCCESS) LONG LSTATUS.

Knowing the return type and the ERROR_SUCCESS macro value, we can easily create a test condition to see if an error occurred. If success is indicated by a NULL value, then it does not matter if the returned value is negative or positive. ***GetLastError***, or similar, could be used to determine what the error code was and what it means. It would be best to have any errors logged in an error log file instead of to the terminal window. A simple description of the error could be given to the operator running to tool and could tell them that the log file would contain more details about the error.

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (6)



Sometimes you need more details about an error.

Many times, you will need to grab a string representation of the error number. To do that you need a way to format the error number into a message, or a string.

```
Error code 1999
```

```
Error code 5
```


Handling Errors (6)

Seeing your program or function fail with error number 5 might not mean anything at all to you. You would probably have to search online what the error code means, but you do not always have a means to do that lookup. Thankfully, this error lookup can be done programmatically with the help of the ***FormatMessage*** function.

The ***FormatMessage*** function is a very powerful function. Depending on a flag passed, it will allocate the buffer to hold the string on its own or simply let the caller give it a buffer large enough to hold the returning string. The latter option has a bit more manual effort and might be more error prone since it is up to you, the developer, to allocate the correct size buffer. It could be best to just let the function do the allocation for you.

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (7)

```
WINBASEAPI
_Success_(return != 0)
DWORD
WINAPI
FormatMessageW(
    _In_     DWORD dwFlags,
    _In_opt_ LPCVOID lpSource,
    _In_     DWORD dwMessageId,
    _In_     DWORD dwLanguageId,
    _When_((dwFlags & FORMAT_MESSAGE_ALLOCATE_BUFFER) != 0, _At_((LPWSTR*)lpBuffer, _Outptr_result_z_))
    _When_((dwFlags & FORMAT_MESSAGE_ALLOCATE_BUFFER) == 0, _Out_writes_z_(nSize))
             LPWSTR lpBuffer,
    _In_     DWORD nSize,
    _In_opt_ va_list *Arguments
    );
```


Handling Errors (7)

At first glance, this looks like a highly complicated function, but in the end, it is not that bad. The SAL annotations make it look more complex than it is, but now that you know how to interpret SAL annotations, the function and its parameters should be easier to figure out. To make it even more understandable, we can break down the parameters and take a quick look at an example of calling the function.

The return type is a DWORD, but that is not terribly important with this function. Also of note, you can see several annotations used to describe the function itself. Success for this function is when the return value does not equal zero (0). Now let's dive into the parameters.

dwFlags can be almost any combination of flags. You can pass it FORMAT_MESSAGE_ALLOCATE_BUFFER to indicate that you want the function to allocate the message buffer on your behalf. This ensures the correct size buffer will be allocated. Passing FORMAT_MESSAGE_FROM_SYSTEM indicates that you want the function to search its message-table resources for the message. The result from ***GetLastError*** can also be passed in when this flag is used. Passing FORMAT_MESSAGE_IGNORE_INSERTS indicates that insert sequences like %1 should be ignored. You will pass that flag if you want to format your message later.

lpSource is where the message is defined. It's best to simply pass NULL here.

dwMessageId is the error number that is being queried. Be sure to cast your values to the correct type.

dwLangaugeId is the language identifier for the message. Passing NULL here is fine too and the function will do an internal search for the message with the proper LANGID.

Now comes the more interesting part. All this boils down to whether the function is to allocate a buffer, or the user is passing one.

© SANS Institute 2024
https://linktr.ee/offsecexam

nSize is the minimum size of the buffer to allocate if the allocate buffer flag is passed. If the flag is not set, then this should be the size of the receiving buffer in TCHARS.

**Arguments* are for insert values for a formatted message. %1 is the first argument, %2 would be the second, and so on. Most of the time you can just pass NULL here.

Tamarisk
OffsecExam
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Handling Errors (8)



Additional flags for `FormatMessage` function

`FORMAT_MESSAGE_ARGUMENT_ARRAY`	A pointer to array of arguments
`FORMAT_MESSAGE_FROM_HMODULE`	Module handle with message-table
`FORMAT_MESSAGE_FROM_STRING`	Pointer to string message definition
`FORMAT_MESSAGE_FROM_SYSTEM`	Search system message-table

Handling Errors (8)

The *dwFlags* parameter can hold a single flag or even a combination of almost any flag. Some flags that cannot be used together are FORMAT_MESSAGE_FROM_STRING and FORMAT_MESSAGE_FROM_HMODULE. You can feel free to experiment with each of these to get a solid understanding of how they are used and what they provide. Here is a brief breakdown of the flags listed on the slide. FORMAT_MESSAGE_ARGUMENT_ARRAY indicates that the argument passed in for *Arguments* is a pointer to an array of arguments, similar to how arguments are passed into a formatted string message with ***printf()***.

The FORMAT_MESSAGE_FROM_HMODULE indicates that the *lpSource* parameter holds a handle to a module where the message-table can be searched. It is possible to create your own message-table and use it as a resource for making a resource only DLL. The Microsoft docs have a writeup of how to do that here: https://docs.microsoft.com/en-us/windows/win32/eventlog/message-text-files.

The FORMAT_MESSAGE_FROM_STRING flags indicates that the lpSource parameter is a string pointer that has the message definition in it. The string might also have insertions like %1.

The FORMAT_MESSAGE_FROM_SYSTEM flag has already been discussed but is provided here for easy reference. By passing this flag, it enables the caller to pass in the value from the ***GetLastError*** function as the dwMesageId.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: FormatMessage

```
LPSTR messageBuffer;

FormatMessageA(
         FORMAT_MESSAGE_ALLOCATE_BUFFER |
         FORMAT_MESSAGE_FROM_SYSTEM |
         FORMAT_MESSAGE_IGNORE_INSERTS,
         NULL, ErrorCode, 0, (LPSTR)&messageBuffer, 0, NULL);

printf("%s\n", messageBuffer);

LocalFree(messageBuffer);
```


Example: FormatMessge
When doing a system error lookup, say from some LRESULT function, you need to make sure you call this function correctly like in the example code above. Since you do not control a system message, you must pass in the FORMAT_MESSAGE_IGNORE_INSERTS flag. The reason for this is the fact that ***FormatMessage*** can expect insertions in the message and when one is found, it will take whatever the first argument in the argument list is; and attempt to insert it into the message. Since we are only dealing with error codes, there will not be one and the function would fail with error 87: ERROR_INVALID_PARAMETER. Also, the way this function is being called, we are asking the system to allocate a buffer on our behalf by passing in FORMAT_MESSAGE_ALLOCATE_BUFFER. When using the FORMAT_MESSAGE_ALLOCATE_BUFFER flag, you must not forget to free that allocated buffer. To do so, just call ***LocalFree*** when you are done with it. This should be done with any API that dynamically allocates chunks of memory on your behalf.

Bottom line: when using ***FormatMessage*** to perform system error code lookups, you must pass both flags FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.6: CreateFile



Using the CreateFile function create a file and write data to it.

Please refer to the eWorkbook for the details of this lab.

Lab 1.6: CreateFile
This exercise is to get you familiar with the ***CreateFile*** API.

Please refer to the eWorkbook for the details of the lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?

The point of this lab was to get you familiar with the various parameters of the ***CreateFile*** API. As you may have noticed, there are several parameters that can really change the behavior of the function. The other function that was used was ***WriteFile***, which was not discussed previously. This was done intentionally to force you to look at the parameters either online from MSDN or from the function's declaration in the header file.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed how Windows APIs provide robust capability
- Learned they can have lengthy but descriptive names
- Learned APIs can request to create kernel objects
- Learned how to handle errors

Module Summary

In this module, we discussed the importance of using the Windows API in your programs because they provide your program with such a robust capability. The API names can be quite lengthy, but they are very descriptive with what they are attempting to do, like ***CreateProcess***. There are many APIs that will make requests to the kernel to create objects or modify existing objects with handles returned from the create functions.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What does CreateFile return upon error?

A. A handle to the file

B. ERROR_INVALID_PARAMETER

C. INVALID_HANDLE_VALUE

Unit Review Questions
Q: What does CreateFile return upon error?

A: A handle to the file

B: ERROR_INVALID_PARAMETER

C: INVALID_HANDLE_VALUE

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What does CreateFile return upon error?

A. A handle to the file

B. ERROR_INVALID_PARAMETER

C. INVALID_HANDLE_VALUE

Unit Review Answers
Q: What does CreateFile return upon error?

A: A handle to the file

B: ERROR_INVALID_PARAMETER

C: INVALID_HANDLE_VALUE

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What macro(s) can be used to check HRESULT function return types?

A SUCCEEDED / FAILED

B GetLastError

C STATUS_OK / STATUS_FAILED

Unit Review Questions
Q: What macro(s) can be used to check HRESULT function return types?

A: SUCCEEDED / FAILED

B: GetLastError

C: STATUS_OK / STATUS_FAILED

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What macro(s) can be used to check HRESULT function return types?

A SUCCEEDED / FAILED

B GetLastError

C STATUS_OK / STATUS_FAILED

Unit Review Answers
Q: What macro(s) can be used to check HRESULT function return types?

A: SUCCEEDED / FAILED

B: GetLastError

C: STATUS_OK / STATUS_FAILED

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



How does a user-mode process organize handles?

A. By storing them in a system wide table shared with other processes

B. By storing them in the process handle table

C. By leaving it up to the developer to organize and maintain

Unit Review Questions
Q: How does a user-mode process organize handles?

A: By storing them in a system wide table shared with other processes

B: By storing them in the process handle table

C: By leaving it up to the developer to organize and maintain

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



How does a user-mode process organize handles?

A. By storing them in a system wide table shared with other processes

B. By storing them in the process handle table

C. By leaving it up to the developer to organize and maintain

Unit Review Answers
Q: How does a user-mode process organize handles?

A: By storing them in a system wide table shared with other processes

B: By storing them in the process handle table

C: By leaving it up to the developer to organize and maintain

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Course Overview
Developing Offensive Tools
Developing Defensive Tools
Lab 1.1: PE-sieve, Lab 1.2: ProcMon
Setting Up Your Development Environment
Windows DLLs
Lab 1.3: HelloDLL
Windows Data Types
Call Me Maybe
Lab 1.4: Call Me Maybe
SAL Annotations
Lab 1.5: Safer with SAL
Windows API
Lab 1.6: CreateFile
Bootcamp

The bootcamp portion of the class is really extended class hours but without the lecture. This will give you lecture-free time for you to go back and practice labs again or take on a few of the challenges listed on the next slide.

© SANS Institute 2024

https://linktr.ee/offsecexam

Bootcamp

- Develop your own custom error handling function
- Develop a Registry walker
- The Italian Debugger
- Windows Shells

Bootcamp

For this bootcamp, this is the time to put the concepts learned from this section into practice. There are several bootcamp challenges and all of them are optional for you to complete. The challenges for this bootcamp can be done in any order since they do not depend on each other. The first challenge is to create a custom error handling function that makes use of ***GetLastError*** and ***FormatMessage***. The second challenge is to create a program that can recursively walk a given Registry key and dump the key's values, if any. The debugger one is not really a challenge per se, but more of a guide to get deeply acquainted with WinDbg Preview and remote kernel debugging. The last one is about creating custom shells that can eventually be run remotely across systems.

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- **Windows Tool Development**
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 1

Lab 1.7: Can'tHandleIt
Lab 1.8: RegWalker
Lab 1.9: It's Me, WinDbg
Lab 1.10: ShadowCraft

The bootcamp portion of the class is really extended class hours but without the lecture. This will give you lecture-free time for you to go back and practice labs again or take on a few of the challenges listed on the next slide.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.7: Can’tHandleIt

- Develop custom function to perform system message lookups
- Only argument is the error number
- No need to return anything to the caller

Tamarisk OffSecExam Exam solution

Lab 1.7: Can’tHandleIt
Please refer to the eWorkbook for the details of this bootcamp challenge.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.8: RegWalker

- Recursively walk keys
- Feeling fancy, support command-line arguments
- Args: only keys, only values, recursive flag, key to walk

Lab 1.8: RegWalker
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.9: It's Me, WinDbg

- Become familiar with the WinDbg interface.
- Explore several Kernel and User mode structures.
- Break into a user mode process.

Lab 1.9: It's Me, WinDbg
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 1.10: ShadowCraft

- Create a basic shell.
- Implement features covered in this section.
- Implement thorough error checking.

Lab 1.10: ShadowCraft
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024
https://linktr.ee/offsecexam

670.2

Getting to Know Your Target

Tamarisk OffsecExam Exam solution

© SANS Institute 2024
© 2024 Jonathan Reiter. All rights reserved to Jonathan Reiter and/or SANS Institute.
https://linktr.ee/offsecexam

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE (“SANS INSTITUTE”) FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this CLA. Courseware means all printed materials, including course books and lab workbooks, slides or notes, as well as any digital or other media, audio and video recordings, virtual machines, software, technology, or data sets distributed by SANS Institute to User for use in the SANS Institute course associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and User and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium, whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. User may not sell, rent, lease, trade, share, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute. Additionally, User may not upload, submit, or otherwise transmit Courseware to any artificial intelligence system, platform, or service for any purpose, regardless of whether the intended use is commercial, educational, or personal, without the express written consent of SANS Institute. User agrees that the failure to abide by this provision would cause irreparable harm to SANS Institute that is impossible to quantify. User therefore agrees to a base liquidated damages amount of $5000.00 USD per item of Courseware infringed upon or fraction thereof. In addition, the base liquidated damages amount shall be doubled for any Courseware less than a year old as a reasonable estimation of the anticipated or actual harm caused by User’s breach of the CLA. Both parties acknowledge and agree that the stipulated amount of liquidated damages is not intended as a penalty, but as a reasonable estimate of damages suffered by SANS Institute due to User’s breach of the CLA.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. A written amendment or addendum to this CLA that is executed by SANS Institute and User may accompany this Courseware.

SANS Institute may suspend and/or terminate User’s access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User, (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

The Apple® logo and any names of Apple products displayed or discussed in this book are registered trademarks of Apple, Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

Governing Law: This CLA shall be governed by the laws of the State of Maryland, USA.

Courseware licensed to User under this CLA may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department (“OFAC”), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce (“BIS”). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulation. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

All reference links are operational in the browser-based delivery of the electronic workbook.

© SANS Institute 2024

https://linktr.ee/offsecexam

SEC670.2 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control

SANS

Getting to Know Your Target

© 2024 Jonathan Reiter | All Rights Reserved | Version J01_05

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control: 670.2
Welcome to Section 2 of SEC670. In this section, we will be getting to know the target very well by creating various tools to obtain detailed information.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (1)

	Page
Gathering Operating System Information	4
Lab 2.1: OS Info	14
Service Packs/Hotfixes/Patches	19
Process Enumeration	36
Lab 2.2: ProcEnum	45
Lab 2.3: CreateToolhelp	49
Lab 2.4: WTSEnum	53
Installed Software	65
Directory Walks	73
Lab 2.5: FileFinder	83
User Information	88
Services and Tasks	101

This page intentionally left blank.

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (2)

	Page
Network Information	115
Registry Information	130
Bootcamp	157
Lab 2.6: Ipconfig	161
Lab 2.7: Arp	162
Lab 2.8: Netstat	163
Lab 2.9: ShadowCraft	164

Tamarisk OffSecExam Exam solution

This page intentionally left blank.

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information

Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration

Lab 2.2: ProcEnum

Lab 2.3 CreateToolhelp

Lab 2.4 WTSEnum

Installed Software

Directory Walks

Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

In this module, we will dive into how to gather OS specific information and why it is important.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss the importance of determining OS information
- Explore public methods to retrieve information about the system
- Explore undocumented method to retrieve information about the system

Objectives

The objectives for this module are to understand how important it is to determine what OS version your target is running and/or what service pack your target has, explore a few public methods for retrieving system information, as well as explore some undocumented methods. There are some methods that are more reliable than others, as well as some methods that might return inaccurate information.

Tamaluk OffSecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Survey Script



Survey the host and determine where you are.

Knowing the system that you are on is vital to the success of your operation. A host survey tool can query various components and report back its findings that can then be used to determine the next action.



Survey Script

If you are a red teamer, or a penetration tester, or have taken SEC560, then you may already know that one of the first tasks that you would typically perform is some recon. Recon is such a broad term that can encompass many things, like users, networks, shares, etc. The focus at this point is to learn more about the system. The more detailed information you can gather, the more informed decisions can be made, like what the next action you would take next is. Privilege escalation could be an example of one such action to take after you have gained some insight into what that system is being used for and what value it might be for you. There are several tools that exist today that make this task trivial. There should be some questions coming to mind here. How do they get that information? What APIs are they using? What logs, if any, are generated from calling certain APIs? Since this is a developer-focused course, you will be creating a recon capability.

One of the tools you could create would be a tool that would execute on the target and query several pieces of information about the target. There can be a standard set of information that you might want to gather, like applications installed, especially any AV/EDR solutions. There can also be specific information that could directly relate to the primary goal of an operation, like looking for a specific folder or file. If that folder or file is not there, then the tool can clean itself off target and be done.

© SANS Institute 2024

https://linktr.ee/offsecexam

OS Information

Windows 7 x86 or Windows 10 x86_64?

Service Pack

A collection of updates to be applied as patches for bugs or vulnerabilities. Also provide features to the OS.

Kernel Version

The ntoskrnl.exe is the kernel file itself. The file is typically located under C:\Windows\System32.

OS Information

Perhaps one of the most important pieces of information to gather first would be the exact version of the operating system, if that is not known already. Typically, you would know at least some basic information about the target beforehand, like if it is a Windows 7 or Windows 10 target, but you might not know the exact service pack or what version of ntoskrnl.exe is on the target. The more details you can gather the better, because Windows 10 is not good enough, especially if someone would want to bring in additional payloads that would be specific to the target. If your tool incorrectly said the target was x64 when it was in fact x86, well, let us just hope a process crashes instead of the target system being bug checked, a.k.a. blue screen (BSOD). To aid us in gathering some of this information, there are a number of APIs that we can call. However, certain APIs might not be available on older versions of Windows. Just because an API is available to use on Windows 10 does not automatically mean that it will be available to use on an older version like Windows XP Service Pack 1. The Windows terminal services (WTS) API family is one such example and some of the WTS APIs will be used later for process enumeration.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Versions

Windows releases and their respective version numbers

Windows XP	5.1
Windows Server 2003	5.2
Windows Vista / Server 2008	6.0
Windows 7 / Server 2008 R2	6.1
Windows 8 / Server 2012	6.2
Windows 8.1 / Server 2012 R2	6.3
Windows 10 / Server 2016	10

Windows Versions

When you are querying the target to determine the specific version of the OS, you will not find something that tells you that the target is a Windows Vista system. Instead, you would be given back something like 6.1 to indicate Windows 7. The table is simply here for an easy reference when you are going about gathering information about the OS.

© SANS Institute 2024

https://linktr.ee/offsecexam

GetVersionEx



GetVersionExA/W

Gathers the OS version number

Has BOOL return type

```
BOOL GetVersionExA(
 _InOut_ LPOSVERSIONINFOA lpSystemInfo
);

BOOL GetVersionExW(
 _InOut_ LPOSVERSIONINFOW lpSystemInfo
);

typedef struct _OSVERSIONINFOA {
 DWORD dwOSVersionInfoSize;
 DWORD dwMajorVersion;
 DWORD dwMinorVersion;
 DWORD dwBuildNumber;
 DWORD dwPlatformId;
 CHAR szCSDVersion[128];
} OSVERSIONINFOA, *POSVERSIONINFOA,
*LPOSVERSIONINFOA;
```


GetVersionEx

The ***GetVersionEx*** API can be used in gathering the Operating System's version number, such as one of the ones listed on the previous slide. Here on this slide are two versions of this API, the ANSI and the Unicode versions. The function takes a single argument for a pointer to an OSVERSIONINFO struct. Before we look at the struct, there is some interesting behavior that we need to understand. By default, if you were to call this API on your Windows 10 Dev VM, you would not get back version number 10. The highest you might get back is version 6.2, which would indicate Windows 8.

The reason for this behavior is mainly compatibility. Back in the day, programs written for XP and Vista would check the version numbers and if they were higher than 5 and 1 (5.1), then the application knew they were on something beyond XP. The check would fail, though when version 6.0 was released as the minor version number of 0 is not >= 1. Windows came up with a solution to never increase the major version number, only the minor. This obviously ended when Windows 10 was released because they are not using version 6.4, they are indeed using version 10.0. The only way for this function to return the correct version number is to account for a possible higher operating system that would be declared in its manifest file. The manifest file is purely XML data and nothing more.

Please note that Windows recently deprecated this API and would really like for you to use the newer version helper functions like IsWindows7OrGreater, or IsWindows10OrGreater, etc.

dwOSVersionInfoSize; the size of the struct and should be set using size of (OSVERSIONINFO)
dwMajorVersion; the major version number
dwMinorVersion; the minor version number
dwBuildNumber; the build number
dwPlatformId; the OS platform
szCSDVersion[128]; if there is a service pack installed, then this would the value for it, like "Service Pack 1".

© SANS Institute 2024

GetNativeSystemInfo

GetNativeSystemInfo

Gathers current system information

Has VOID return type

```
VOID
GetNativeSystemInfo(
 _Out_ LPSYSTEM_INFO lpSystemInfo
);

typedef struct _SYSTEM_INFO {
[..SNIP..]
 DWORD   dwPageSize;
 LPVOID  lpMinimumApplicationAddress;
 LPVOID  lpMaximumApplicationAddress;
 DWORD_PTR dwActiveProcessorMask;
 DWORD   dwNumberOfProcessors;
 DWORD   dwProcessorType;
 DWORD   dwAllocationGranularity;
 WORD    wProcessorLevel;
 WORD    wProcessorRevision;
} SYSTEM_INFO, *LPSYSTEM_INFO;
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 10

GetNativeSystemInfo

The ***GetNativeSystemInfo*** API can be used in gathering some specific information about the target system for WoW64 applications. This will also work for x64 applications, but the results will be noticeably different. Since the function has a VOID return type, it will not return anything to the caller, so there would be no point in trying to assign its return value to a variable. What the API needs is a pointer to a structure that the API will fill out according to the information it queries. This is very common behavior for Windows APIs. In fact, most Windows APIs expect to be given a properly initialized structure to fill out for you. Here is a breakdown of the one and only argument to ***GetNativeSystemInfo***.

lpSystemInfo, as annotated by the SAL markup, it is an out parameter meaning the function requires write permissions to the variable. The variable, or pointer, must be of type LPSYSTEM_INFO, which is a structure that holds around 10 or so members.

Let us take a more detailed look at the SYSTEM_INFO struct.

dwOemId; is no longer being used but is simply maintained for compatibility. We need to use wProcessorArchitecture instead.
wProcessorArchitecture; the installed architecture of the process that is installed on the target. This field will have one of the values listed below:

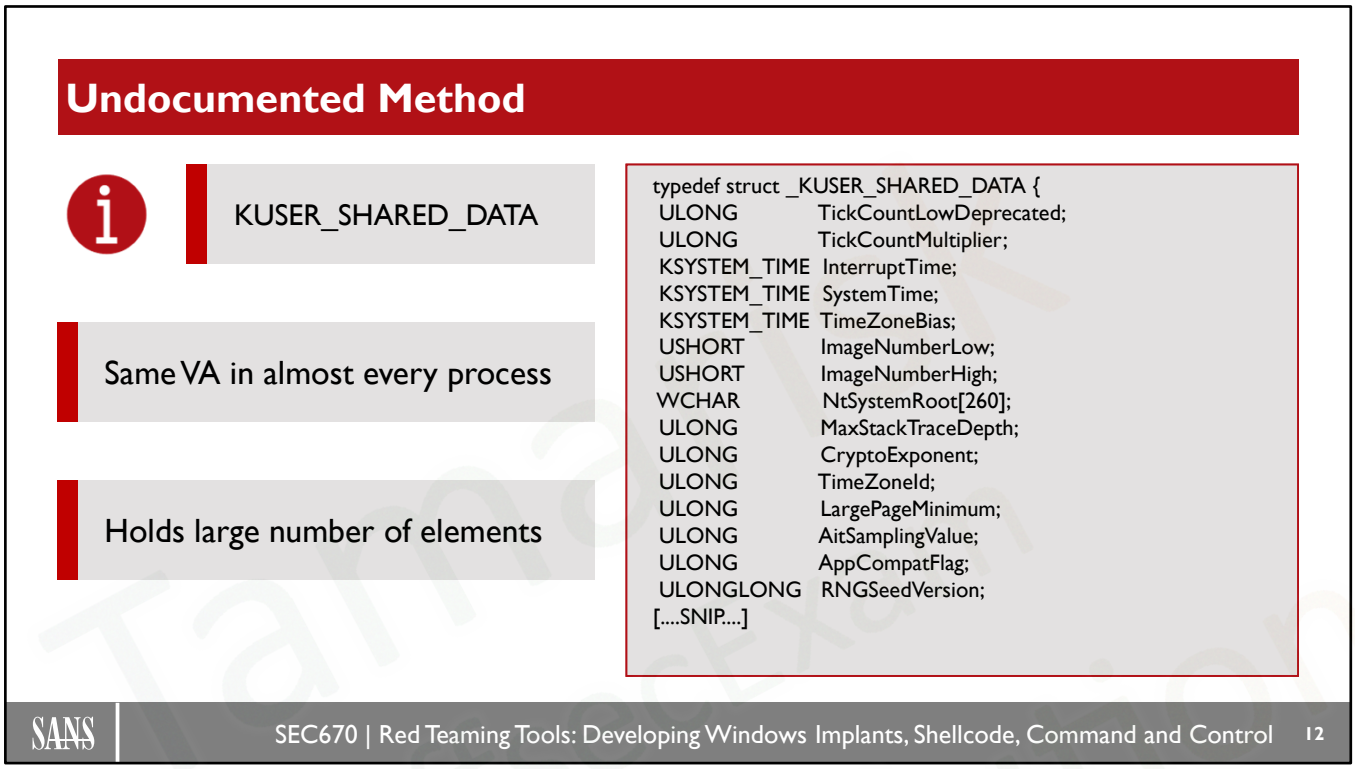
9 - PROCESSOR_ARCHITECTURE_AMD64 (x64 AMD or Intel)
5 - PROCESSOR_ARCHITECTURE_ARM (ARM)
12 - PROCESSOR_ARCHITECTURE_ARM64 (ARM64)
6 - PROCESSOR_ARCHITECTURE_IA64 (Intel Itanium-based)
0 - PROCESSOR_ARCHITECTURE_INTEL (x86)
0xffff - PROCESSOR_ARCHITECTURE_UNKNOWN

© SANS Institute 2024
https://linktr.ee/offsecexam

wReserved; reserved for supposedly something amazing in the future? Who knows?
dwPageSize; the page size along with the granularity of page protection and the commitment. *VirtualAlloc* relies on this value for its operations.
lpMinimumApplicationAddress; this is a pointer to the lowest memory address that will be made accessible to programs and their DLLs.
lpMaximumApplicationAddress; the exact opposite as the previous member.
dwActiveProcessorMask; the set of processors that are configured on the system in the form of a mask, 0-31 bits each one indicating the processor.
dwNumberOfProcessors; how many logical processors are in the current group.
GetLogicalProcessorInformation relies on this value.
dwProcessorType; this is obsolete so do not rely on it.
dwAllocationGranularity; for virtual memory allocations, this is the granularity for the starting address.
wProcessorLevel; the processor level that is dependent on the architecture.
wProcessorRevision; the processor revision that is dependent on the architecture.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024



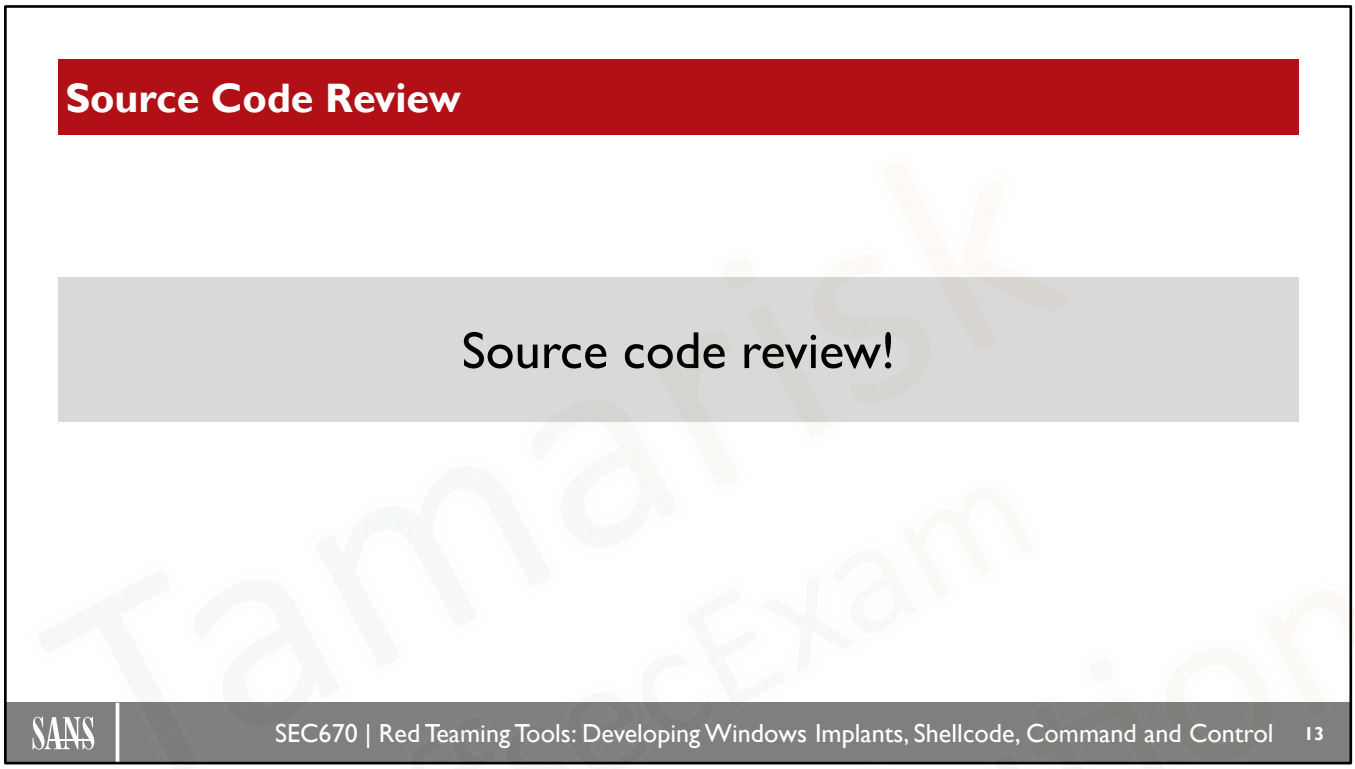
Undocumented Method

The undocumented method of retrieving system information is to query the KUSER_SHARED_DATA struct that is present in virtually every single process at the same Virtual Address: 0x7FFE0000. KUSER_SHARED_DATA is a massive structure that is defined in the ntddk.h header file. The structure stores an enormous amount of information that we can leverage for our needs of gathering system information and then some. However, this is not the recommended method of doing so, but we do not really care too much about that. Implant developers and malware authors tend to favor the undocumented methods more than anything else. Knowing the correct offsets for what information you need can be very useful instead of copying everything over from this structure. Currently, the offsets for the Major Version and Minor Version have been offsets 0x26C and 0x270, respectively. The Build Number can be found at offset 0x260. For more details about this structure, check out the header files and various online resources like Geoff Chappell's documentation.

References:
https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntexapi_x/kuser_shared_data/index.htm
https://www.vergiliusproject.com/kernels/x64/Windows%2010%20|%202016/2004%2020H1%20(May%202020%20Update)/_KUSER_SHARED_DATA

© SANS Institute 2024

https://linktr.ee/offsecexam



Source Code Review
Time to jump into the source code and understand it.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.1: OS Info



Gathering information about the OS and target

Please refer to the eWorkbook for the details of this lab.

Lab 2.1: OS Info
Please refer to the eWorkbook for the details of the lab.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffsecExam Exam solution

What's the Point?
The point of this lab was to understand how you can retrieve various information about the OS of your target.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Discussed how obtaining accurate system information is key

Covered documented and recommended methods to obtain the information

Covered undocumented methods to obtain the information

Module Summary

In this module, we discussed why you would want to know the exact details of your target's OS version and architecture, we also explored a few Windows APIs that enable us to do so, and finally, we took a look at an undocumented method by means of KUSER_SHARED_DATA.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What structure can be found at VA 0x7FFE0000?

A EPROCESS

B KPROCESS

C KUSER_SHARED_DATA

Unit Review Questions
Q: What structure can be found at VA 0x7FFE0000?

A: EPROCESS

B: KPROCESS

C: KUSER_SHARED_DATA

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What structure can be found at VA 0x7FFE0000?

A EPROCESS

B KPROCESS

C KUSER_SHARED_DATA

Unit Review Answers
Q: What structure can be found at VA 0x7FFE0000?

A: EPROCESS

B: KPROCESS

C: KUSER_SHARED_DATA

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

In this module, we will discuss how to gather information about service packs, hotfixes, and patches, as well as why the information might be important.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Determine what patches, hotfixes, etc. might be present
- Discuss the importance of patches

Objectives
The objectives are to determine what patches or hotfixes a system might have and how they might affect an operation.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Hotfixes



Used to fix critical issues in software

Also referred to as Quick Fix Engineering (QFE) updates, hotfixes are used to apply a vital fix to software applications. Users that have Windows updates set to automatic will have hotfixes downloaded without much user intervention. The only exception would be a reboot.

Windows Hotfixes

Windows updates bring with them any number of things, but the emphasis here would be hotfixes. The term "hot fix" traditionally would mean that a patch to a software program can be applied while the system was still running. You may have heard the phrases "hot swappable," "hot patching," etc. which all mean similar things, but the common component is "hot" where the system or device does not have to go through a shutdown procedure; hot swapping hard drives when one drive becomes full, or in our case, issuing a quick fix to a certain application while the system is still running, and the user is presumably performing work. Some Windows updates do, however, require a reboot for a change to come into effect, but there are times when a reboot is not required to be performed. It is important for us to know what hotfixes have been applied so that time is not wasted attempting to execute an exploit against something that has already been patched. Doing so could tip off the user/admin to our presence on the system along with generating unnecessary logs.

© SANS Institute 2024

https://linktr.ee/offsecexam

Service Packs



Bundled hotfixes

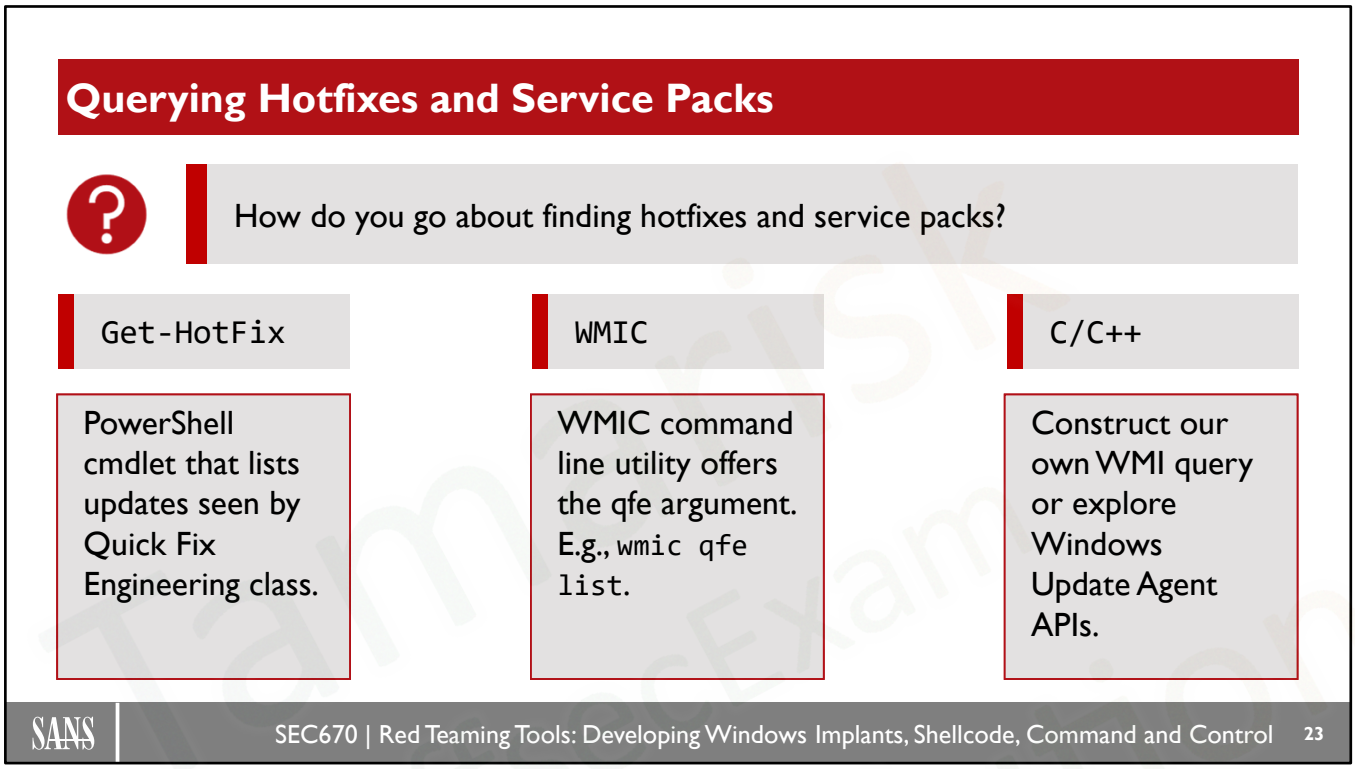
Each service pack brings with it a grouping of one or more hotfixes that will be applied to the OS. Each service pack that targets a particular OS version will have all previous hotfixes that former service packs brought with it so that a user can jump straight to the most recent service pack without installing each one sequentially.

Service Packs

It really would not make much sense for Windows to push down hotfixes by themselves one at a time, but to rather bundle them up in what is called a service pack. Bundling the hotfixes together is pretty great because it makes updating a lot more efficient. Each OS version can have any number of service packs issued, like Windows XP had three of them: SP1, SP2, and SP3. Each service pack might also mean that exploits might have to be adjusted depending on the target's SP level. Within the Metasploit Framework, the option to choose a specific target is made available. An exploit targeting a vulnerable FTP server might need to have the target be specified, say for Windows XP SP3 versus XP SP1. The same considerations must be taken if your implant is going to exploit anything from local privilege escalation to persistence, etc.

© SANS Institute 2024

https://linktr.ee/offsecexam



Querying Hotfixes and Service Packs

Windows provides users and admins with a number of options to go about querying patches, or hotfixes, which have been applied to a system. Perhaps the easiest method would be to drop into a PowerShell instance and run the Get-HotFix cmdlet. The downside is that the cmdlet does not return everything because on the back end it only queries the WMI Win32_QuickFixEngineering class. Knowing that, we can easily craft our own WMI (Windows Management Instrumentation) query and completely avoid using PowerShell. Another alternative is to use the *wmic qfe list* command. The QFE part of the command gives away what it is querying, and it also indicates that it will show the same results as Get-Hotfix. Lastly, it will have the same downside that the Get-HotFix cmdlet does. If you were to look at the Windows Programs and Features listing, you might notice a difference in the listing. This is because the QFE WMI class only sees certain types of updates. We want a more complete view and as such, we must utilize the Windows Update Agent APIs. We can use them as a standalone or in combination with what the QFE class returns.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Update Agent (WUA) APIs



Introduced with Windows XP, designed for system admins and developers

Scripts and/or programs can be developed to determine what updates are available to be installed on a system, what updates have been installed, or to remove any installed updates.

Windows Update

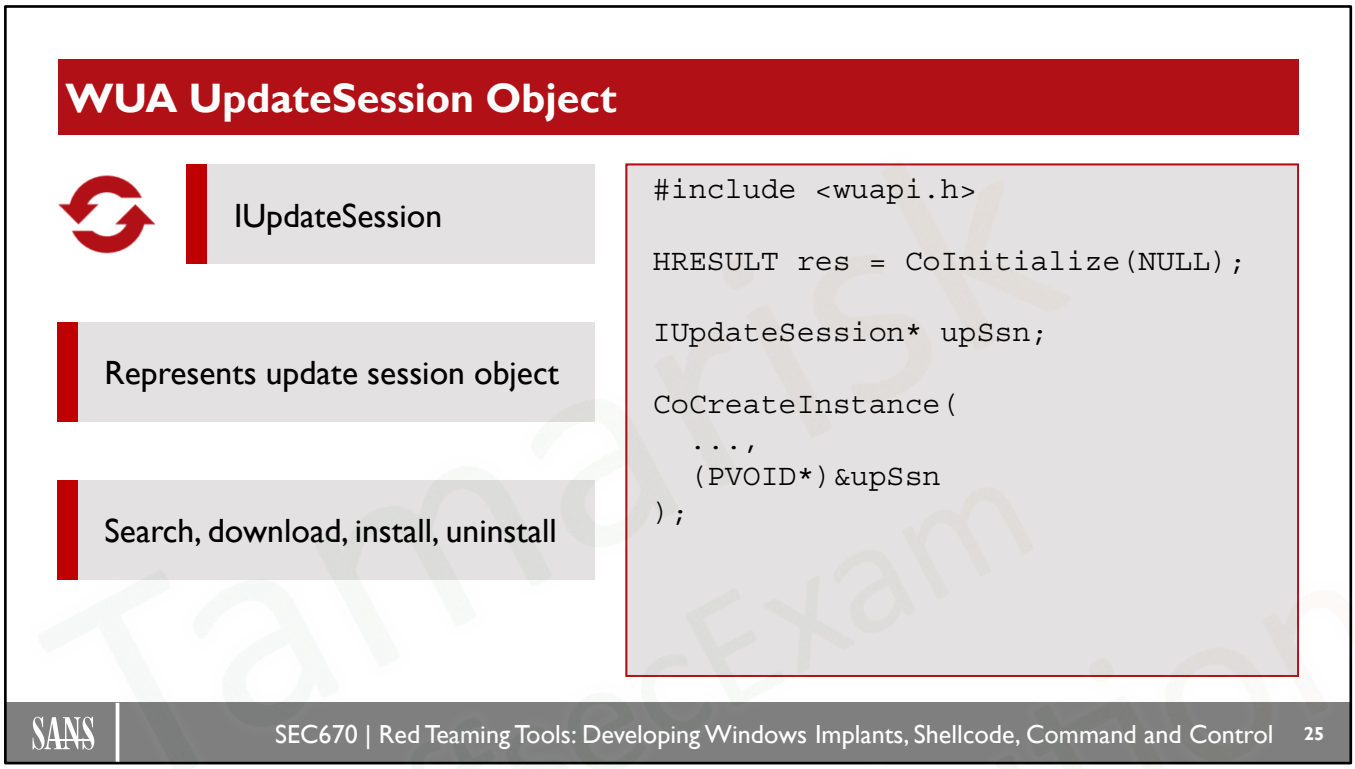
Windows Server Update Services (WSUS)

Windows Update Agent (WUA) APIs

Programmatically creating a solution is often more complicated than using pre-built programs or tools like the wmic tool. We turn to this option when existing tools fail or do not show us the entire picture we are hoping to see. Enter the WUA API family. Since its debut with Windows XP, it has been used by system administrators and developers alike, to determine what updates computers in their organization require. What we want to use it for is to determine what updates have already been applied because we like to go after the lowest hanging fruit. WUA is a set of COM interfaces, and we must create instances of whichever interface we need. Before the WUA APIs are made available to us, we must reference the proper header file and its respecting lib file; Wuapi.h and Wuguid.lib, respectively. To create the correct interface, we need to choose the COM object that is most suited for our needs, and that could be the *UpdateSession*, *UpdateSearcher*, and *SearchResult* WUA objects.

© SANS Institute 2024

https://linktr.ee/offsecexam



WUA UpdateSession Object

Because the WUA APIs are all COM-based, the calling thread must initialize the COM library by calling the ***CoInitialize*** API. The WUA *UpdateSession* object is a COM interface, hence the naming convention of adding the letter "I" to the API name. Before we can do anything useful, we must create an instance of the object by calling the ***CoCreateInstance*** API and passing in the address to our pointer variable *IUpdateSession*. The pointer variable is going to receive the pointer to the interface if the function call succeeds. We use the ***CoCreateInstance*** API because it will create and initialize the object of whatever class we are passing in for the CLSID parameter. It is from this newly created object that we will then be able to create other objects using methods provided by the *UpdateSession* object.

© SANS Institute 2024

https://linktr.ee/offsecexam

WUA UpdateSearcher Object



IUpdateSearcher

Created by the UpdateSearcher coclass

Used to search for updates on the target system

```
#include <wuapi.h>

...
IUpdateSession* upSsn;
IUpdateSearcher* upSearch;
ISearchResult* results;

upSsn->CreateUpdateSearcher(&upSearch);
...
upSearch->Search(criteria, &results);
```


WUA UpdateSearcher Object

The previous slide showed how to create the *UpdateSession* object and now that it is created, we can call its ***CreateUpdateSearcher*** method to make the *UpdateSearcher* object. We need to create the *UpdateSearcher* object because it provides a method that we need to start conducting a search of updates that have been applied to the system. The method of interest for us is the ***Search*** method. The ***Search*** method requires two arguments: the criteria and a pointer to the *UpdateSearcher* pointer variable. The criteria argument is where we can specify our search criteria—think of it like a filter of sorts. The criteria can be created as a string like the following: "IsInstalled=1". The criteria string can contain several filters using the "or" operator so you could expand the string to be "IsHidden=1 or IsInstalled=1". There are (sometimes) updates that are marked as hidden on the computer, but we want to see them anyway, so we set that variant to true. The collection of results that match the specified criteria will be stored in the results *ISearchResult* pointer variable. It is from this object that you can start to process the results from the search as it exposes several methods of interest to us.

© SANS Institute 2024

https://linktr.ee/offsecexam

WUA SearchResult Object



ISearchResult

Used to represent search results

Has methods that can query updates from a resulting search

```
// interface collection of updates from a
resulting search

ISearchResult* results;
IUpdateCollection* upList;
LONG upSize;

upSsn->CreateUpdateSearcher(&upSearch);
upSearch->Search(criteria, &results);

results->get_Updates(&upList);
upList->get_Count(&upSize);
```


WUA SearchResult Object

The *SearchResult* WUA object represents the collection of updates that matched the search criteria. To get the collection interface of those updates from the search, we will need to call ***get_Updates*** method from the *SearchResult* object. The argument we pass to the method is the address to the pointer variable of an *IUpdateCollection* pointer. The *UpdateCollection* object is used to represent the list of updates, which is an ordered list of updates. We can then use this ordered list and iterate over it based on the size of the list that can be gathered after calling the ***get_Count*** method. We pass the ***get_Count*** method the address to a LONG variable to store the size of the update list. Finally, we can gather the details of the updates we found on the system and either print them out to the terminal window or write the results to a log file somewhere on disk.

© SANS Institute 2024

https://linktr.ee/offsecexam

Sample Code

```
...
IUpdate* upItem;
BSTR upName;

results->get_Updates(&upList);
upList->get_Count(&upSize);

LONG index = 0;
for ( ; index < upSize; index++)
{
 upList->get_Item( index, &upItem );
 upItem->get_Title( &upName );
 ...
}
```


Sample Code

This slide just holds some pseudo code for how one could possibly iterate over a collection of results. Some new variables here are the pointer variable for the *IUpdate* collection object, which is used to obtain whatever properties and methods an update might have. The next variable is a basic string variable named *upName* that is short for the update name. As we iterate over the collection, the *upName* will be updated to the one at the proper index in the collection. The *IUpdate* collection object exposes other methods too, like ***get_Type***, to get the type of the update; ***get_Title***, to get the title of the update; or ***get_KBArticleIDs***, to get the collection of KB article IDs that are tied to the update.

Specifically, the code on the slide will get the current update item at the index specified, update the *upItem* variable so we can get the title of the current update, and move on with the iteration process after that. To be even more detailed and thorough, you could do more during this loop by calling some of the other *IUpdate* methods, but this is enough to get you started.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed the importance of updates and patches
- Learned how to obtain information about patches
- Used the WUA APIs

Module Summary
In this module, we discussed hotfixes, service packs, and how to get information about them using the WUA APIs.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions

What PowerShell cmdlet queries the Quick Fix Engineering class?

A Get-HotFix

B Get-Updates

C Get-ServicePack

Unit Review Questions
Q: What PowerShell cmdlet queries the Quick Fix Engineering class?

A: Get-HotFix

B: Get-Updates

C: Get-ServicePack

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What PowerShell cmdlet queries the Quick Fix Engineering class?

A Get-HotFix

B Get-Updates

C Get-ServicePack

Unit Review Answers
Q: What PowerShell cmdlet queries the Quick Fix Engineering class?

A: Get-HotFix

B: Get-Updates

C: Get-ServicePack

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the update family of APIs used to query hotfixes?

A WUA

B LUA

C FUA

Unit Review Questions
Q: What is the update family of APIs used to query hotfixes?

A: WUA

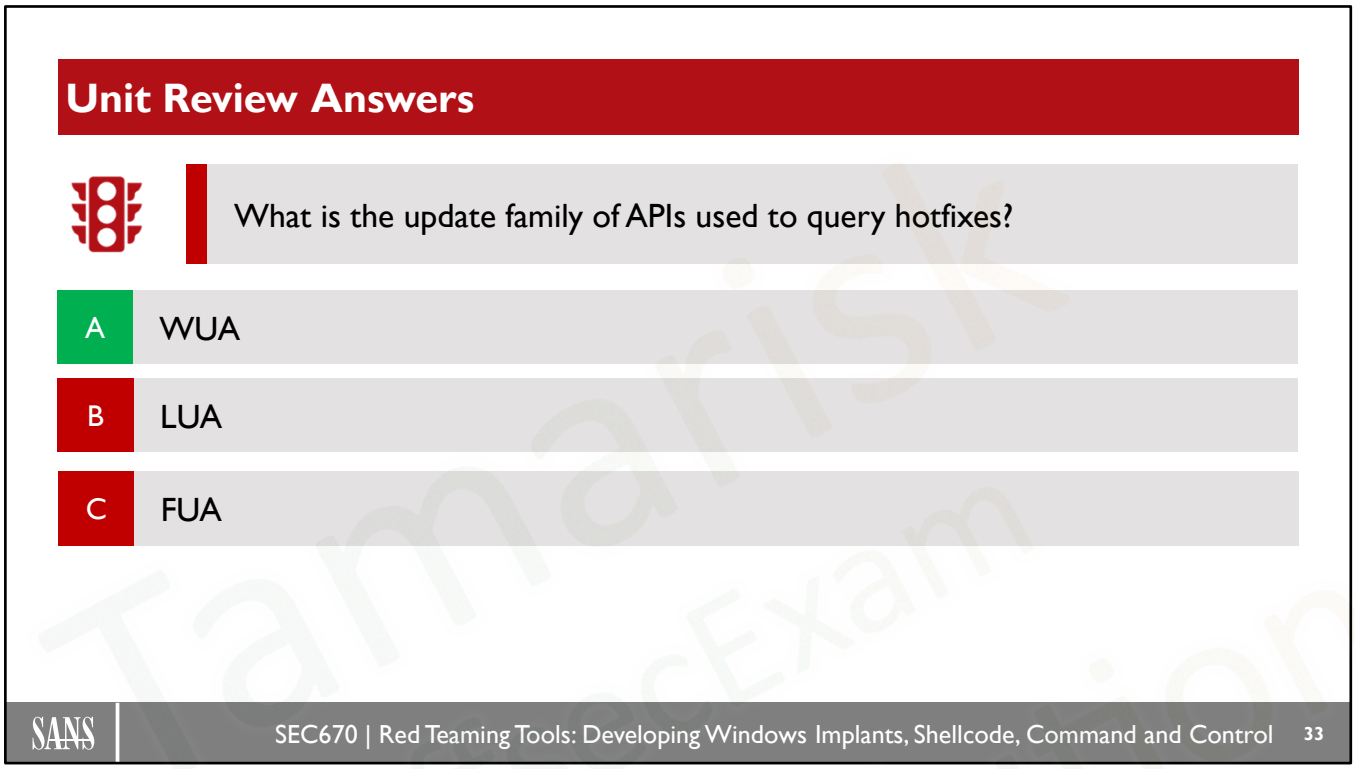
B: LUA

C: FUA

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam



Unit Review Answers
Q: What is the update family of APIs used to query hotfixes?

A: WUA (Windows Update Agent)

B: LUA

C: FUA

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What WUA object is used to find updates on a system?

A SearchResult

B UpdateSearcher

C UpdateSession

Unit Review Questions
Q: What WUA object is used to find updates on a system?

A: SearchResult

B: UpdateSearcher

C: UpdateSession

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What WUA object is used to find updates on a system?

A SearchResult

B UpdateSearcher

C UpdateSession

Unit Review Answers
Q: What WUA object is used to find updates on a system?

A: SearchResult

B: UpdateSearcher

C: UpdateSession

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

In this module, we will look at the how and why when it comes to process enumeration.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Understand the need for process enumeration
- Take a deeper look at processes
- Explore the various methods to enumerate processes

Tamarisk OffSecExam Exam Solution

Objectives

The objectives for this module are to understand the need for enumerating processes. Furthermore, to understand processes even more, we will look at what processes are, how they are created, different process states, and the several methods involved with enumeration. Let's get to it.

© SANS Institute 2024

https://linktr.ee/offsecexam

Why Enumerate Processes?



Must find out what applications are running

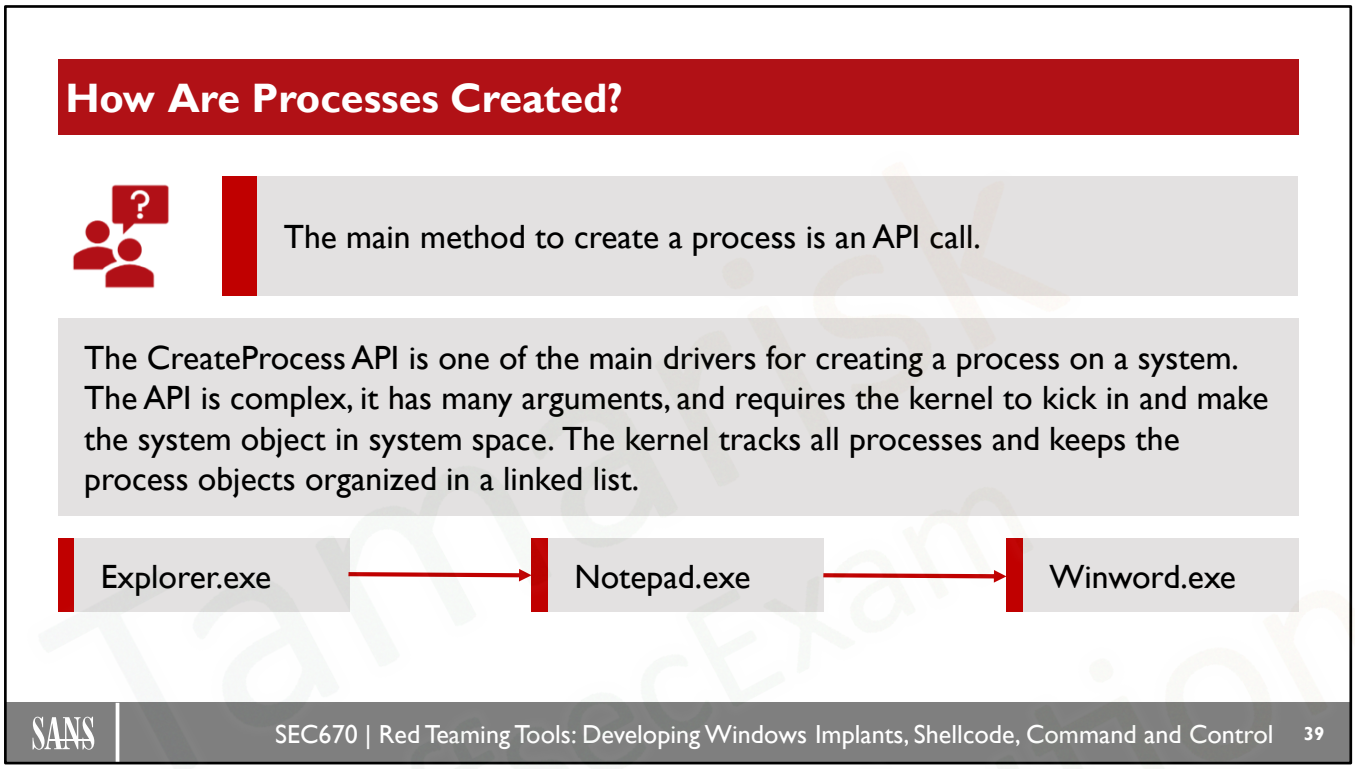
An important part of conducting a survey is gathering a list of running processes. Depending on what processes you find, your operation may come to a halt, or you might deem the target safe for further operations.

Why Enumerate Processes?

One of the features that implants often have is the ability to enumerate processes. If you are familiar with Metasploit's Meterpreter session, you might know that it can enumerate processes. There are several reasons why a red team operator might want to gather a list of running processes. One reason could be to find a suitable target process for shellcode injection. Another reason could be to find out if the target has a vulnerable application that could then be matched to an exploit to aid in escalation of privileges, should your initial access not be elevated already. Yet another reason to enumerate processes would be to determine if there is any security product present on the system. Depending on the security product installed on the system, you might decide to halt or suspend operating on that target. Many solutions have a cloud-based portion that will take your tooling and conduct analysis in its cloud engine. If you do not want your tooling to be siphoned off to their cloud, then perhaps you need to clean off the target.

© SANS Institute 2024

https://linktr.ee/offsecexam

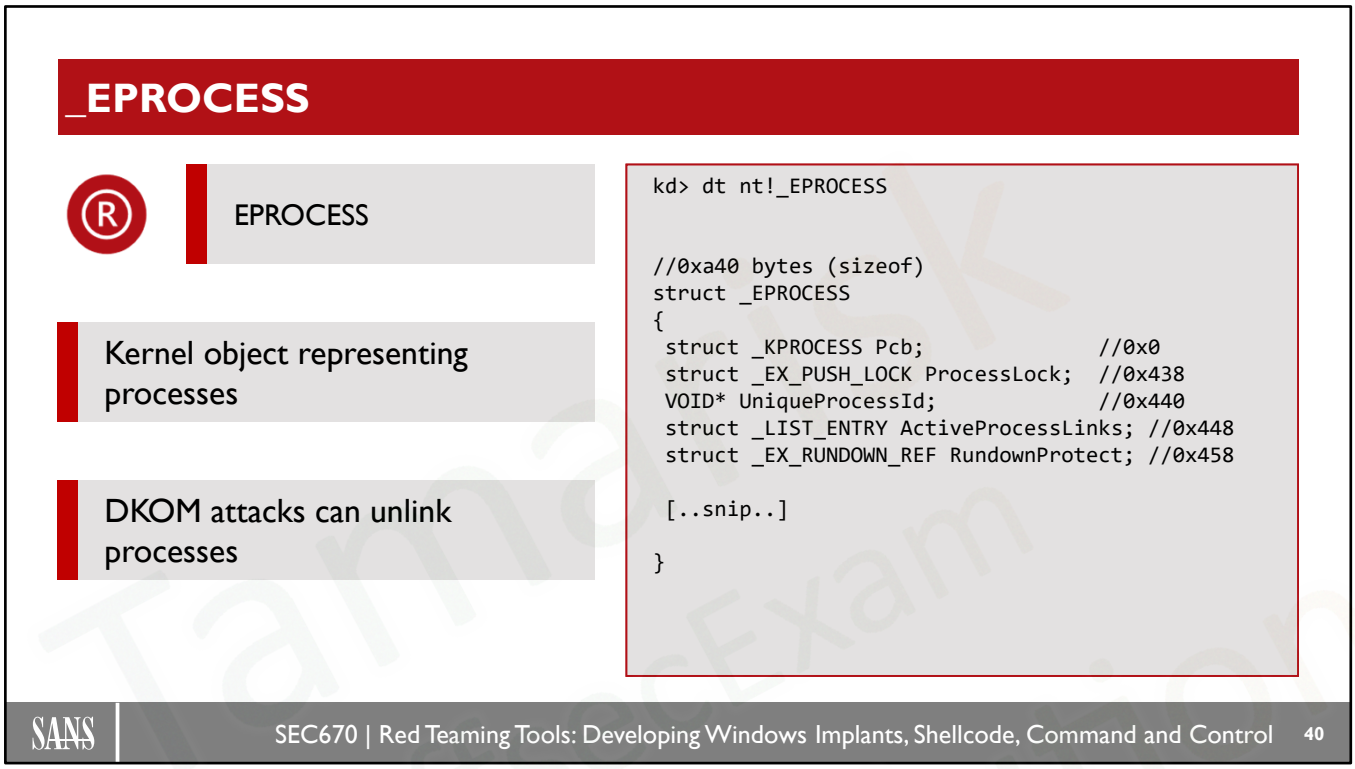


How Are Processes Created?

Process creation can be kicked off by calling the ***CreateProcess*** API. User programs can easily make calls to ***CreateProcess***, but they do not have to do any heavy lifting. The kernel will eventually take over and it is up to the kernel to create a process object in system space, inject the main thread into the process, and append the process object to the existing linked list of process objects. There is more that the kernel does for process creation, but those items hit the general responsibilities. There are two types of process objects that are linked together: the *_EPROCESS* object and the *_KPROCESS* object. Both objects, together, represent a single process and each one holds different information about a process that is important for various Windows subsystems. The *KPROCESS* object is the first member of the *EPROCESS* object and is intentionally designed this way. If you have the address of the *EPROCESS* object, you also have the *KPROCESS* as well since it is at offset 0x00.

© SANS Institute 2024

https://linktr.ee/offsecexam



_EPROCESS

The *_EPROCESS* object is one of the structures that Microsoft determines to be opaque to user mode apps and to developers. The *EPROCESS* object belongs to the Windows Executive subsystem since it needs to have access to all information to manage the process. From the small snippet on the slide, you can see some of the information stored in it. One of the best methods to peeking inside of opaque structures is to look at them during a kernel debugging session. KDNET makes getting kernel debugging working with minimal effort. Another way is to look at the Vergilius Project, which has done an amazing job at documenting various kernel structures from many Windows versions. Check out their site here: https://www.vergiliusproject.com.

The *ActiveProcessLinks* is an interesting one because its type is *_LIST_ENTRY*, indicating that it is used to link to other objects. Windows uses this type to make a doubly linked list. The *LIST_ENTRY*'s FLINK would take us to the next *EPROCESS* struct in the list. Many programs indirectly walk this list when enumerating processes like the PowerShell cmdlet *Get-Process* or executing *tasklist* at the command line. When an attacker can get into the kernel, these objects can be directly manipulated, and a process can be unlinked from the chain and therefore hidden from users.

© SANS Institute 2024

https://linktr.ee/offsecexam

_KPROCESS



KPROCESS

Kernel object representing processes

Used by the lower layer of the Kernel

```
//0x438 bytes (sizeof)
struct _KPROCESS
{
 struct _DISPATCHER_HEADER Header;   //0x0
 struct _LIST_ENTRY ProfileListHead; //0x18
 ULONGLONG DirectoryTableBase;       //0x28
 struct _LIST_ENTRY ThreadListHead;  //0x30
 ULONG ProcessLock;                  //0x40
 ULONG ProcessTimerDelay;            //0x44
 ULONGLONG DeepFreezeStartTime;      //0x48
 struct _KAFFINITY_EX Affinity;      //0x50
 ULONGLONG AffinityPadding[12];      //0xf8
 struct _LIST_ENTRY ReadyListHead;   //0x158
 [..snip..]

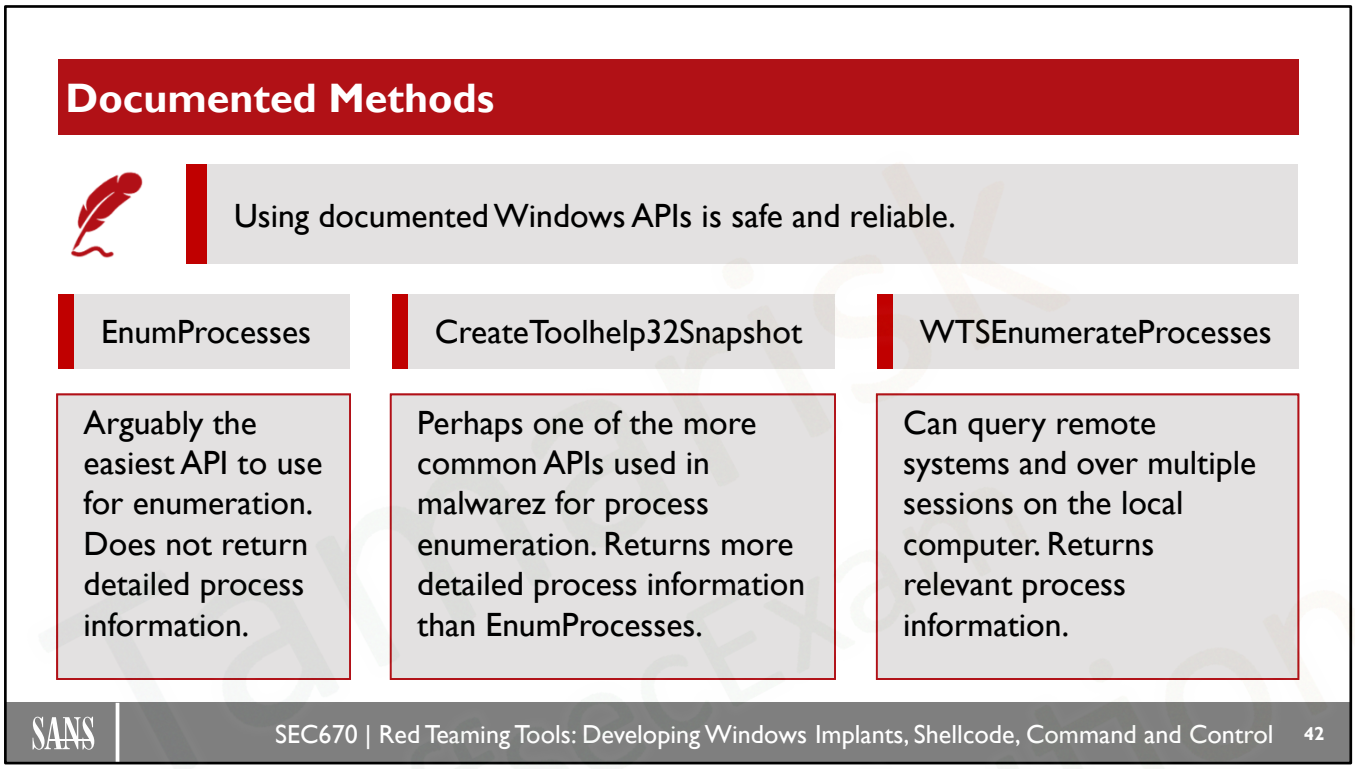
}
```


_KPROCESS

The *_KPROCESS* object is important to the core of the kernel and is not exposed via the Object Manager like the *EPROCESS* object is. Some of the members of the KPROCESS object are important for thread scheduling, like what threads are ready to run (*ReadyListHead*), tracking quantum, priorities, CPU affinity, linking of threads, etc. For example, the *ThreadListHead* is of type *_LIST_ENTRY* so we know it is part of a doubly linked list. The list is a chain of Threads that have been created in the process which is represented by the *KPROCESS* object. Another important member is the *DirectoryTableBase* of type *ULONGLONG*, which will hold the physical address of the process' Page Directory Table, an important item for Virtual Address Translation.

© SANS Institute 2024

https://linktr.ee/offsecexam

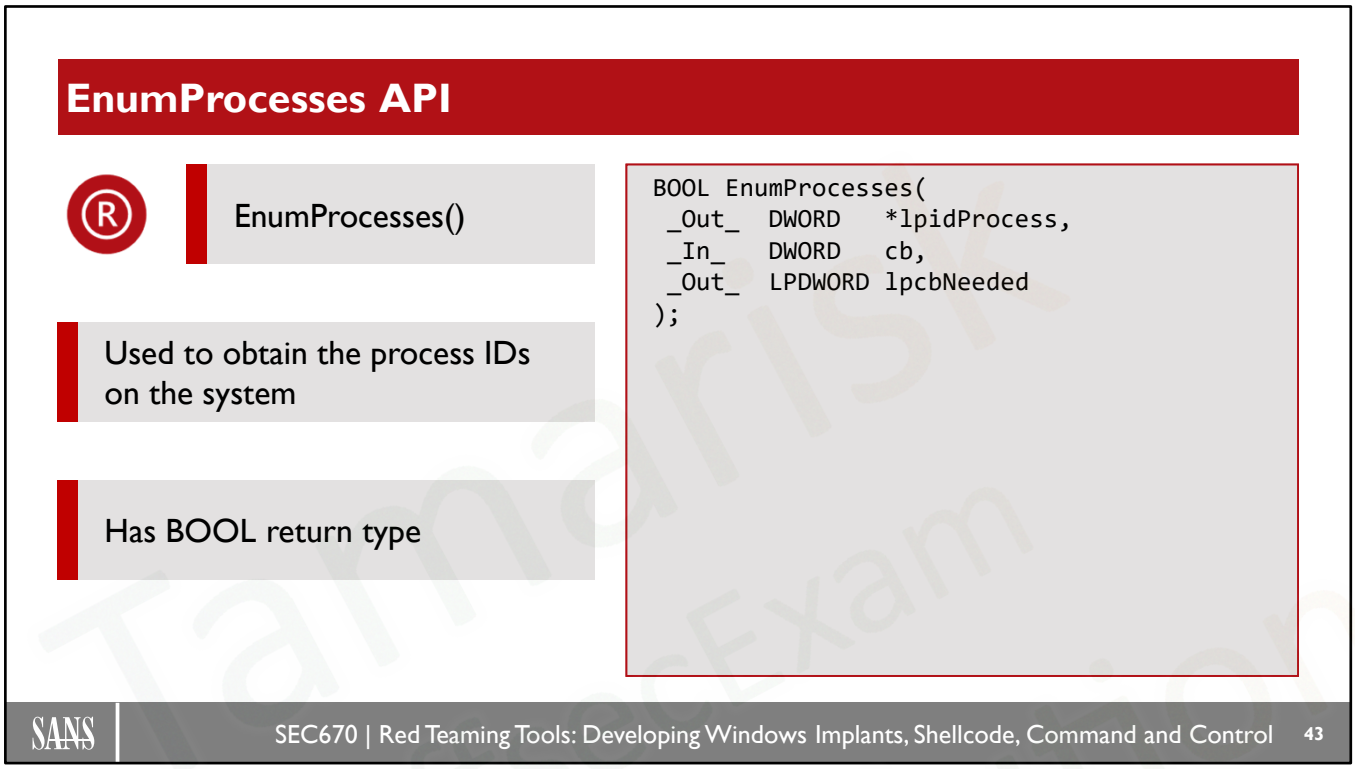


Documented Methods

When it comes to reliability and stability, using officially supported and documented APIs is great because you know the API should work as advertised. For the documented methods of enumerating processes, we have three options. The first option is the ***EnumProcesses*** API, which is a very simple API to understand and implement. One of the major drawbacks about ***EnumProcesses*** is the lack of detailed information about each enumerated process. If you wanted to get more information with this API, then you could attempt to open a handle to the process via its *ProcessId*. The second option is to use the ***CreateToolhelp32Snapshot*** API, which offers much more information about each process. The downside to this API is that because it only takes a snapshot of the currently mapped processes, you will miss any new processes after the snapshot is taken. The last one we will cover is ***WTSEnumerateProcesses***, which offers a nice feature of remote process enumeration. Some of the above APIs might not be available for every version of Windows, so you must be sure to find out and test.

© SANS Institute 2024

https://linktr.ee/offsecexam



EnumProcesses API

As mentioned on the previous slide, the ***EnumProcesses*** API is incredibly easy to use when it comes to process enumeration. The API can be chosen over the other options if you do not care about getting detailed information about the processes on the system. The API will only return the process IDs of each process object that the kernel has created at the time the API is called. There are three arguments required: two out parameters and a single in parameter. Let us break down some of these arguments.

lpidProcess, is an out DWORD pointer to the array that will hold the process IDs.

Cb, is an in DWORD that indicates the size, in bytes, of the array.

lpcbNeded, is an out LPDWORD that will indicate how many bytes were placed in the array.

Because the function requires a buffer, the array, it needs to be large enough to hold all the PIDs. The function will not indicate beforehand how large the buffer needs to be, so we can just make a large one to err on the side of caution and avoid the need of having to call the API twice.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: EnumProcesses

```
if ( !EnumProcesses(dwProcList, sizeof(dwProcList), &dwRealSize) )
{
    // fail and bail code here
    goto fail_and_bail;
}

// iterate over the results
for ( DWORD i = 0; i < dwCount; i++ )
{
    HANDLE hProc = OpenProcess( PROCESS_QUERY_LIMITED_INFORMATION, FALSE,
dwProcList[i] );
    // do something with the handle if OpenProcess succeeds
}
```


Example: EnumProcesses

This example of ***EnumProcesses*** is a small snippet for how to call this API. There is not a lot of code here to make this work because it is a very simple API to use, and it does not return a lot of information. The API will fill out a list that will hold the PIDs of the processes that are currently mapped into memory. The API does not tell us how many processes were found, so we must do that on our own. We can use some simple math and figure out how many entries there are in the array of PIDs. To determine the number of processes, divide the *dwRealSize* by the size of the DWORD data type. Now that the count has been calculated, we can use that value in a *for* loop so we can iterate over the array. Each iteration will execute the *wprintf* statement showing the user the PID that is currently being processed. Though, your implant would not be making *printf* type of function calls because there would be no terminal to see the results. Instead, it would be more beneficial to create a log file or place the results in a memory buffer to later be sent back to your C2 infrastructure. You can also gather more information about each PID by calling ***OpenProcess*** against each one. If you are successful with that operation, you could then get more details about that process via the handle.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.2: ProcEnum



Using EnumProcesses, enumerate the processes on the system.

Please refer to the eWorkbook for the details of the lab.

Lab 2.2: ProcEnum
Please refer to the eWorkbook for the details of the lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of the lab was to explore the ease of use for this API. It does have a few drawbacks, like the limited information, but that can be accounted for by opening a process handle to each PID returned by the API.

© SANS Institute 2024

https://linktr.ee/offsecexam

CreateToolhelp32Snapshot API

CreateToolhelp32Snapshot()

Creates a snapshot of a process

Can take snapshots of heaps and threads as well

```
HANDLE CreateToolhelp32Snapshot(
 _In_ DWORD dwFlags,
 _In_ DWORD th32ProcessID
);

BOOL Process32First(
 _In_  HANDLE hSnapshot,
 _Out_ LPPROCESSENTRY32 lppe
);

BOOL Process32Next(
 _In_  HANDLE hSnapshot,
 _Out_ LPPROCESSENTRY32 lppe
);
```


CreateToolhelp32Snapshot API

The ***CreateToolhelp32Snapshot*** API was discussed during the Create APIs module during Section 1, but it is being included again here as it specifically relates to enumerating processes. The API will create a snapshot of the specified processes, if any, and return a handle to that snapshot. Emphasis must be placed on the "snapshot" since you could easily miss a new process being created as this is not a dynamic view. You can use the handle to perform your queries and extract the information you are looking for like a specific process name or module name the process has loaded. The function is relatively easy to call since it only takes two parameters which are of the same type. The *dwFlags* parameter is the most important as it dictates what data should be collected in the snapshot. There are seven flags that can be passed here, but the most interesting flag for us in this use case is **TH32CS_SNAPPROCESS** because, as the name implies, will grab all processes that have been mapped into memory. There are other flags that can be used to capture other useful information, like modules and threads if you need to enumerate those items.

After the snapshot has been taken, you can perform our process enumeration using the ***Process32First*** function. The ***Process32First*** function will gather information about the first process in the snapshot. The function requires two parameters: the handle to the snapshot and a pointer to a variable of type *PROCESSENTRY32* structure. Typically, what you would see is a call to ***Process32First*** and then a loop that uses the ***Process32Next*** function to determine when to break out of the loop. ***Process32Next*** requires the same arguments as the ***Process32First*** function. The next slide will show a small snippet of code of what this could look like.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: CreateToolhelp32Snapshot

```
DWORD lastError = 0;
HANDLE snapShot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);

PROCESSENTRY32 pe32 = { 0 };
pe32.dwSize = sizeof PROCESSENTRY32;

printf("%20s %7s\n", "Process Name", "PID");
printf("..........................................\n");
do {
    printf("%20ws", pe32.szExeFile);
    printf("%8d\n", pe32.th32ProcessID);
} while (Process32Next(snapShot, &pe32));

CloseHandle(snapShot);
return ERROR_SUCCESS;
```


Example: CreateToolhelp32Snapshot

The example here intentionally omits error checking and the call to ***Process32First***, due to size limitations on the slide. Regardless, the main points are represented here, starting with the call to the ***CreateToolhelp32Snapshot*** function on the second line. We are only interested in capturing processes in this snapshot and we are not specifying a process ID as indicated by NULL. The function returns a handle value which is saved off in the *snapShot* variable. In the full code, this would be error checked against **INVALID_HANDLE_VALUE** to ensure the snapshot was created successfully. The next couple of lines create the *pe32* variable of struct type **PROCESSENTRY32** and since it has a size field, it should be set to the size of the struct. The next important part in this code snippet is the do while loop. The body of the loop is simply printing out the name of the executable and the corresponding process ID. The exit condition is met when the ***Process32Next*** function returns FALSE since it is a BOOL return type. To understand the process even more and what other information is available, research the **PROCESSENTRY32** definition in the TlHelp32.h header file.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.3: CreateToolhelp



Using CreateToolhelp32Snapshot, enumerate the processes on the system.

Please refer to the eWorkbook for the details of the lab.

Lab 2.3: CeateToolhelp
Please refer to the eWorkbook for the details of the lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

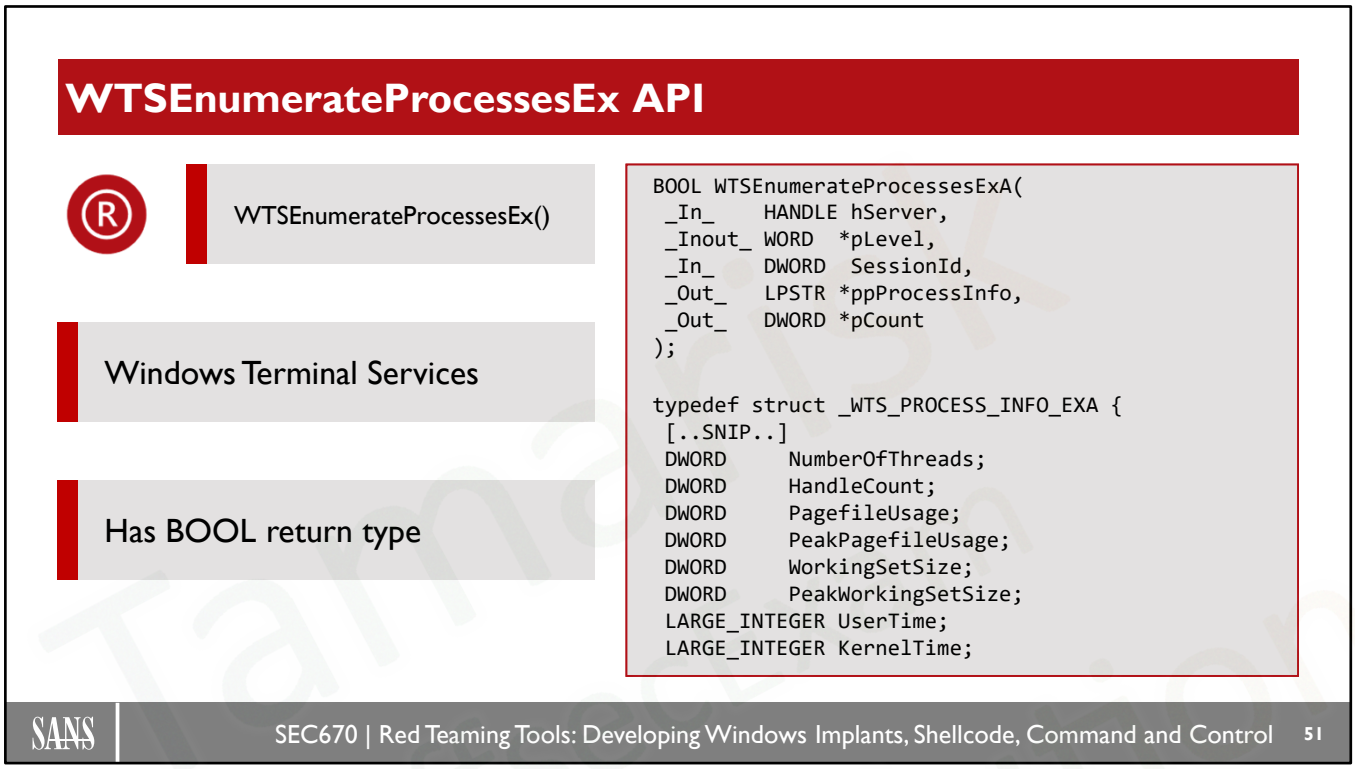
What's the point?

What's the Point?
The point of the lab was to explore one of the more popular methods of enumerating processes. The major downside to this method is you can miss newly created processes after the snapshot has been taken.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam



WTSEnumerateProcessesEx API

There is an entire family of WTS APIs that are defined in the wtsapi32.h header file. The WTS prefix is the abbreviation for Windows Terminal Services, so think Remote Desktop. With that being said, a big advantage of using the WTS APIs for enumerating processes is that we could do so against remote systems that are configured for it. Specifically, there is a registry key that must be configured to enable the remote interrogation, but if it is there then remote process queries are possible. The API will return very detailed information about each process that is running on a local or remote system. The information about each process is returned via the *ppProcInfo* parameter, which is a pointer to a *WTS_PROCESS_INFO* struct. This will be an array of *WTS_PROCESS_INFO* struct entries. Let us go ahead and break down the API's parameters.

hServer, a handle returned by the ***WTSOpenServer*** API. The handle can be for a local or remote system.

pLevel, used to determine what level of information you would like returned. Passing 1 would give the extended version of the *WTS_PROCESS_INFO* struct.

SessionId, used when you would want to query a different session on the system. You can, of course, query all sessions, just pass in WTS_ANY_SESSION.

ppProcInfo, the out parameter so the function can file out the *WTS_PROCESS_INFO(Ex)* structure with the detailed information for the process.

pCount, the out parameter that will indicate how many structures were created by the API, in other words, the number of processes.

When you are done processing the information in the returned buffer, we must free it using the ***WTSFreeMemoryEx*** API.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: WTSEnumerateProcessesEx

```
if (!WTSEnumerateProcessesEx(WTS_CURRENT_SERVER_HANDLE, &dwLevel, WTS_ANY_SESSION, (PWSTR*)&procInfoEx, &dwCount))
{
    wprintf(L"WTSEnumerateProcessesExW failed with error: %d\n", GetLastError());
    return FALSE;
}

if (NULL == procInfoEx) { ... }

wprintf(L"%-20s [%5s] [%6s] [%7s] [%6s] [%-20s]\n", L"ImageName", L"PID", L"Threads", L"Handles", L"Session", L"UserName");
wprintf(L"================================================================================\n");

// iterate over the results
DWORD dwIndex = 0;
for (; dwIndex < dwCount; dwIndex++)
{
    auto pProcInfo = procInfoEx + dwIndex;
    wprintf(L"%-20.19s [%5d] [%7d] [%7d] [%7d] [%20s]\n",
            pProcInfo->pProcessName,
            pProcInfo->ProcessId,
            pProcInfo->NumberOfThreads,
            pProcInfo->HandleCount,
            pProcInfo->SessionId,
            (PCWSTR)GetUserNameFromSid(pProcInfo->pUserSid) );
}
```


Example: WTSEnumerateProcessesEx

The example here shows how to use the API to make a local query. The amazing part about this API is that it provides detailed information about each process. We can even query the session to see what session is tied to a certain process. This can lead down a path of injecting into processes that are in a different session. This method is called cross-session process injection. Unlike the ***EnumProceses*** API, this one can return the count to you, which is a nice touch. Just like the other API, once the count is known, we can iterate over the array and process the structures that the API created to represent each process. Inside the body of the for loop, you can see some of the processing of this information. It is inside the loop body that you can decide to perform other operations, like writing the output to a log file or getting the data ready to send back to your C2 server.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.4: WTSEnum



Using WTSEnumerateProcesses, enumerate processes on the system.

Please refer to the eWorkbook for the details of the lab.

Lab 2.4: WTSEnum
Please refer to the eWorkbook for the details of the lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

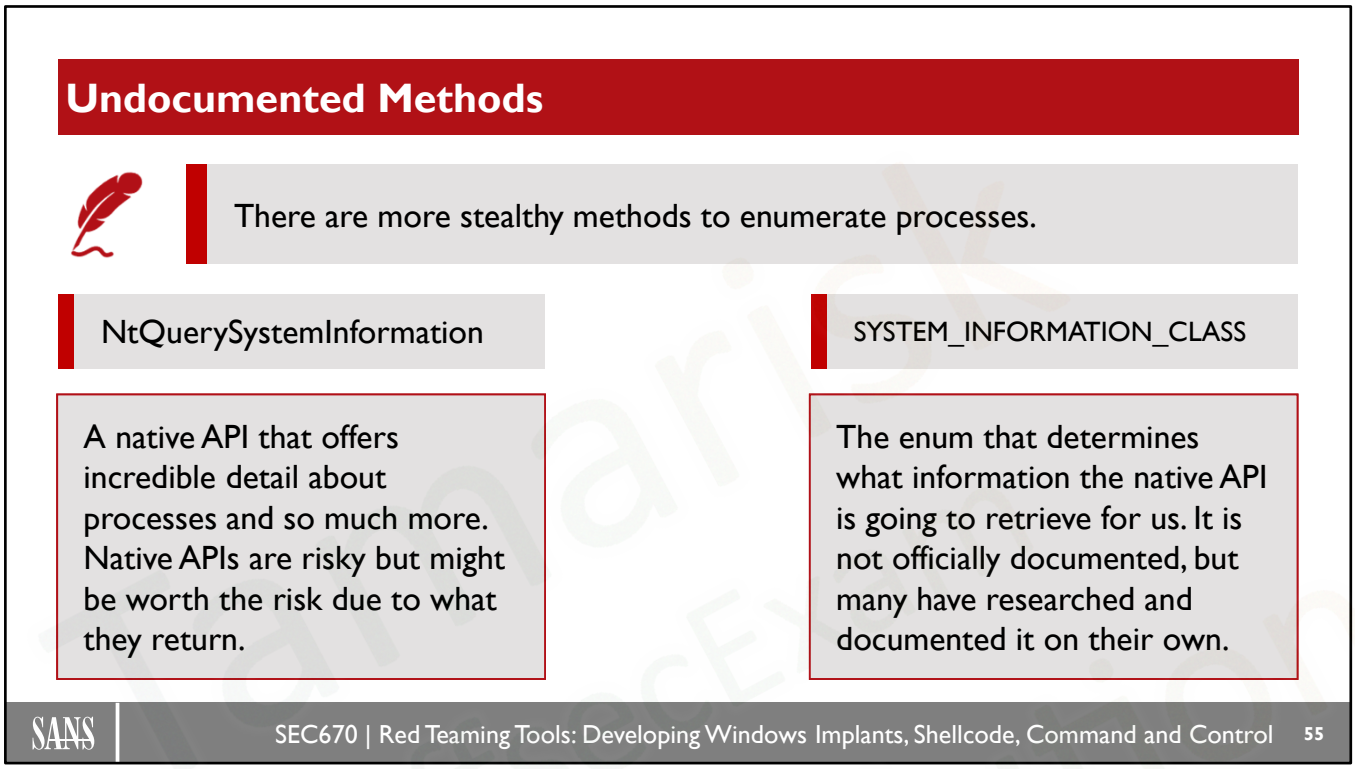
What's the Point?

What's the point?

Tamarisk OffSecExam Exam Solution

What's the Point?
The point of the lab was to explore another method to enumerate processes. Using the Windows Terminal Services is nice because you have the potential to query remote targets.

© SANS Institute 2024



Undocumented Methods

Native APIs tend to be very risky to use as they are not officially documented by Windows. The APIs could change without Microsoft making an announcement of any kind, so your tool could work just fine on one version of Windows and then come time for an update, the native API you were using breaks, rendering your tool useless. When it comes to process enumeration, there is an amazing native API that we can use called ***NtQuerySystemInformation***. MSDN does have some documentation on it and the main description for it says, "**NtQuerySystemInformation** may be altered or unavailable in future versions of Windows. Applications should use the alternate functions listed in this topic." Despite that warning, we are going to use it anyway because we do not have to heed to their warning; we are creating an implant after all.

The ***NtQuerySystemInformation*** API relies heavily on the *SYSTEM_INFORMATION_CLASS* enum, which is not formally documented. In this undocumented enum are entries that we can use to specify what type of system information we are interested in seeing. The enum entry that would be of interest for process enumeration would be *SystemProcessInformation*. MSDN does not list every entry for the enum, but several GitHub projects, like the one for the x64dbg debugger, have kindly posted their hard efforts for us.

References:
https://docs.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntquerysysteminformation
https://github.com/x64dbg/x64dbg/blob/development/src/dbg/ntdll/ntdll.h

© SANS Institute 2024

https://linktr.ee/offsecexam

NtQuerySystemInformation API

NtQuerySystemInformation

Grabs specific information about the system

Has NTSTATUS return type

```
NTSTATUS
NtQuerySystemInformation(
 _In_ SYSTEM_INFORMATION_CLASS InfoCls,
 _Inout_   PVOID SystemInformation,
 _In_      ULONG SystemInformationLength,
 _Out_opt_ PULONG ReturnLength
);

// enum entry SystemProcessInformation

// SYSTEM_PROCESS_INFORMATION struct
```


NtQuerySystemInformation API

As mentioned on the previous slide, the ***NtQuerySystemInformation*** function is a native function as annotated by the Nt prefix in the function name. Even though MSDN has documentation online for this function, they do not recommend we use it. Instead, alternate functions are listed as suggestions to use to retrieve information about the system. The reasoning behind this is that the function could "break" or be removed from any future version release. This is not likely to happen, so we use this function anyway because it can return some very rich information not only about processes but also about the system. The function is extremely useful, and its true power comes from what SYSTEM_INFORMATION_CLASS is passed into it. To better understand how to use this function, we can break down the parameters in detail.

InfoCls is an abbreviation for *SystemInformationClass*, of type SYSTEM_INFORMATION_CLASS, which is a massive enum. Since we are talking about process information, we will be using the *SystemProcessInformation* enum entry. Each enum entry has a corresponding structure and ours will be the SYSTEM_PROCESS_INFORMATION struct. What will be returned to use is an array of these structures where each entry represents a process that has been mapped into memory.

SystemInformation, of type PVOID, is the pointer to the buffer to hold the information to be returned from the function. The size of this buffer must be known ahead of time, and it can also vary based on the information that is being requested. To get this information, the function will need to be called twice with the first call used just to get the size of the buffer. Then you can allocate a buffer of the correct size for the second call.

SystemInformationLength, of type ULONG, is the size of the buffer that the *SystemInformation* parameter points.

ReturnLength, of type PULONG, is not required to be passed, but if used, it will hold the actual size of the requested information.

© SANS Institute 2024

https://linktr.ee/offsecexam

SYSTEM_PROCESS_INFORMATION Struct

```
typedef struct _SYSTEM_PROCESS_INFORMATION {
  ULONG NextEntryOffset;
  ULONG NumberOfThreads;
  LARGE_INTEGER WorkingSetPrivateSize;   // Since Vista
  ULONG HardFaultCount;                  // Since Windows 7
  ULONG NumberOfThreadsHighWatermark;    // Since Windows 7
  ULONGLONG      CycleTime;              // Since Windows 7
  LARGE_INTEGER  CreateTime;
  LARGE_INTEGER  UserTime;
  LARGE_INTEGER  KernelTime;
  UNICODE_STRING ImageName;
[..SNIP..]
  HANDLE UniqueProcessId;
  HANDLE InheritedFromUniqueProcessId;
  SYSTEM_THREAD_INFORMATION Threads[1];
[..SNIP..]
```


SYSTEM_PROCESS_INFORMATION Struct
Here is the struct as documented by the researchers and developers of x64dbg.

typedef struct _SYSTEM_PROCESS_INFORMATION
{
 ULONG NextEntryOffset;
 ULONG NumberOfThreads;
 LARGE_INTEGER WorkingSetPrivateSize; // Since Vista
 ULONG HardFaultCount; // Since Windows 7
 ULONG NumberOfThreadsHighWatermark; // Since Windows 7
 ULONGLONG CycleTime; // Since Windows 7
 LARGE_INTEGER CreateTime;
 LARGE_INTEGER UserTime;
 LARGE_INTEGER KernelTime;
 UNICODE_STRING ImageName;
 KPRIORITY BasePriority;
 HANDLE UniqueProcessId;
 HANDLE InheritedFromUniqueProcessId;
 ULONG HandleCount;
 ULONG SessionId;
 ULONG_PTR UniqueProcessKey; // Since Vista (requires SystemExtendedProcessInformation)
 SIZE_T PeakVirtualSize;
 SIZE_T VirtualSize;
 ULONG PageFaultCount;
 SIZE_T PeakWorkingSetSize;
 SIZE_T WorkingSetSize;
 SIZE_T QuotaPeakPagedPoolUsage;
 SIZE_T QuotaPagedPoolUsage;

© SANS Institute 2024


```
SIZE_T QuotaPeakNonPagedPoolUsage;
  SIZE_T QuotaNonPagedPoolUsage;
  SIZE_T PagefileUsage;
  SIZE_T PeakPagefileUsage;
  SIZE_T PrivatePageCount;
  LARGE_INTEGER ReadOperationCount;
  LARGE_INTEGER WriteOperationCount;
  LARGE_INTEGER OtherOperationCount;
  LARGE_INTEGER ReadTransferCount;
  LARGE_INTEGER WriteTransferCount;
  LARGE_INTEGER OtherTransferCount;
  SYSTEM_THREAD_INFORMATION Threads[1];
} SYSTEM_PROCESS_INFORMATION, *PSYSTEM_PROCESS_INFORMATION;
```

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: NtQuerySystemInformation

```
Status = NtQuerySystemInformation(
    SystemProcessInformation,
    SystemInformation,
    SystemInformationLength,
    &ReturnLength);
// error check again
if (!NT_SUCCESS(Status)) { ... }

auto SystemProcess = (PSYSTEM_PROCESS_INFORMATION)SystemInformation;
ULONG dwIndex = 0;
for (; ; dwIndex)
{
    // convert the handle to a ULONG
    auto Pid = HandleToULong(SystemProcess->UniqueProcessId);
    // if the PID is 0 it is for the Idle process, else we have another process we can grab the name for it
    auto Name = Pid == 0 ? L"[Idle]" : CString(SystemProcess->ImageName.Buffer, SystemProcess->ImageName.Length / sizeof(WCHAR));
    auto SessionId = SystemProcess->SessionId;
    auto HandleCount = SystemProcess->HandleCount;
    auto ThreadCount = SystemProcess->NumberOfThreads;
    auto NPP = SystemProcess->QuotaNonPagedPoolUsage;

    wprintf(L"%-5d  %-20.19s  [%7d]  [%7d]  [%7d]  [%10lld]\n", Pid, (LPCWSTR)Name, SessionId, HandleCount, ThreadCount, NPP);
```


Example: NtQuerySystemInformation

Whenever you decide to use native APIs, you are also deciding whether to accept the risk that comes with the usage. They do offer amazing benefits, like increasing your chances of bypassing AV/EDR detection and being able to obtain more detailed information like what this example shows. Hackers take risks all the time and developers must take risks too to increase the chances of a red team operator being successful for the operation at hand. Once the API call has been made, we have a linked list that we can iterate over. Each entry in the list is of type PSYSTEM_PROCESS_INFORMATION, and we can use it and some of its members to keep advancing to the next entry until there are no more entries. There is so much information that it will not fit on the slide, but the best way to see all this information is online using the MSDN documentation and some of the third party sources, like x64dbg's GitHub repo.

Bottom line: certain native functions are well worth the risk based solely on what they can offer.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Discussed the reason for enumerating processes

Explored the structures the kernel uses to represent processes

Explored various methods for process enumeration

Module Summary

In this module, we discussed why it is important to enumerate processes on a system, the structures the kernel uses to represent processes in system address space, and several methods to enumerate processes on local and remote systems.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What undocumented API can be used to enumerate processes?

A EnumProcesses()

B WTSEnumerateProcessesEx()

C NtQuerySystemInformation()

Unit Review Questions
Q: What undocumented API can be used to enumerate processes?

A: EnumProcesses()

B: WTSEnumerateProcessEx()

C: NtQuerySystemInformation()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What undocumented API can be used to enumerate processes?

A EnumProcesses()

B WTSEnumerateProcessesEx()

C NtQuerySystemInformation()

Unit Review Answers
Q: What undocumented API can be used to enumerate processes?

A: EnumProcesses()

B: WTSEnumerateProcessEx()

C: NtQuerySystemInformation()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What API could be used to obtain more information about a process given only its PID?

A OpenProcess()

B WTSEnumerateProcessesEx()

C NtQuerySystemInformation()

Unit Review Questions
Q: What API could be used to obtain more information about a process given only its PID?

A: OpenProcess()

B: WTSEnumerateProcessEx()

C: NtQuerySystemInformation()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What API could be used to obtain more information about a process given only its PID?

A OpenProcess()

B WTSEnumerateProcessesEx()

C NtQuerySystemInformation()

Unit Review Questions
Q: What API could be used to obtain more information about a process given only its PID?

A: OpenProcess()

B: WTSEnumerateProcessEx()

C: NtQuerySystemInformation()

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

Finding installed software can tell you a great deal about a target. Let us dive in, shall we?

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Look at where installed software is located
- Compile a listing of installed software
- Determine if an operation should continue

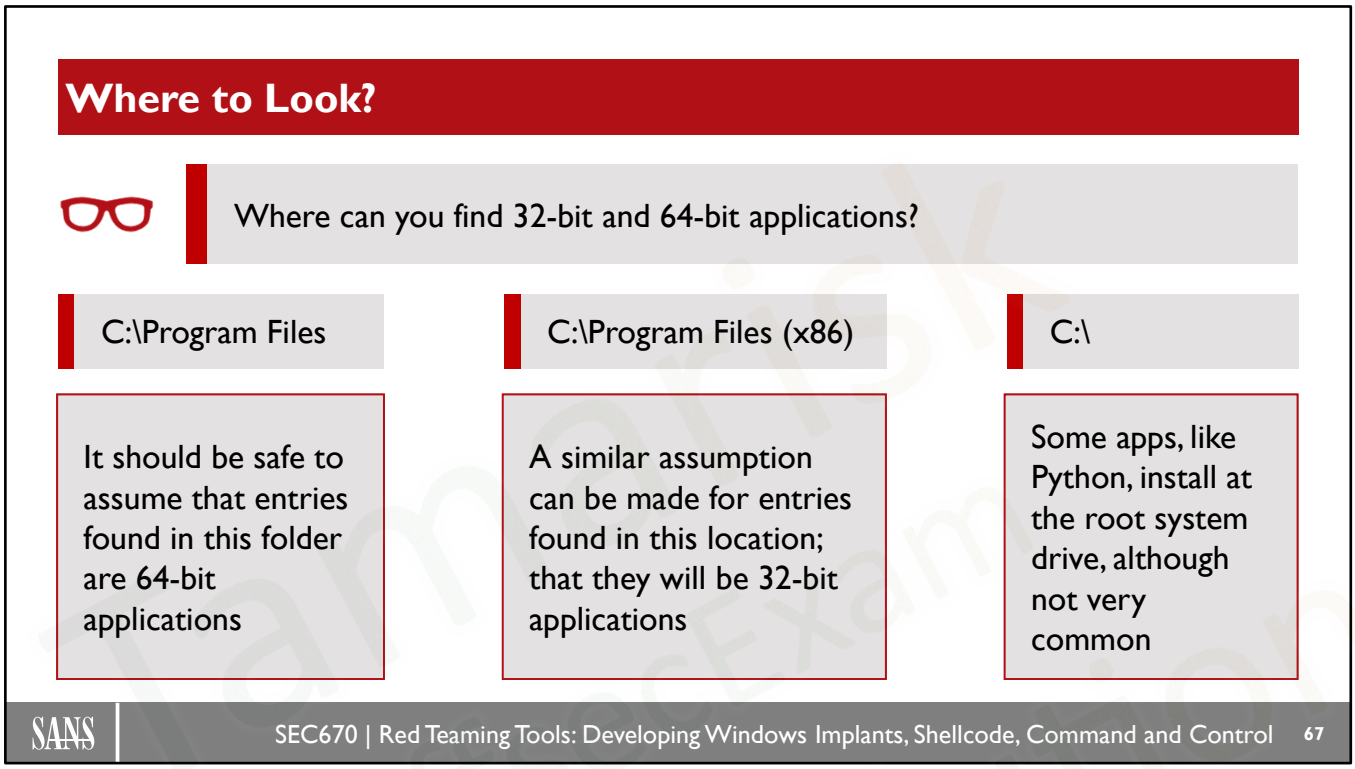
Objectives

The objectives for this module are to know where to look for installed software, compile a listing of all installed programs, and determine if an operation should continue given the presence, or absence, of software.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam



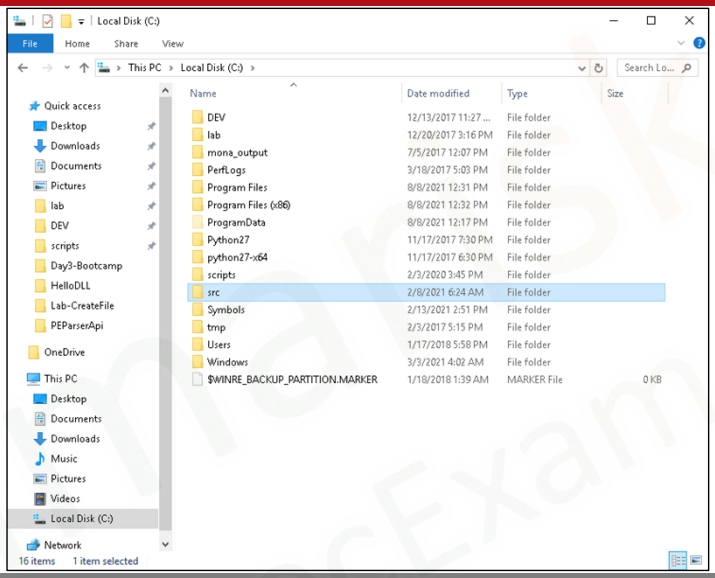
Where to Look?

One of the goals of recon is to determine what applications are installed on your target. Maybe you want to see if an already known vulnerable application is there, or perhaps you want to make sure a certain application is not installed. In either case, it is good to know where to find the installation folders. The assumption here is that the target is a 64-bit Windows installation. Based on that assumption, 64-bit applications will be located at the "C:\Program Files" directory or the C:\Progra~1 for using the 8.3 short name convention that NTFS supports. The next folder name really gives away what the purpose is: "C:\Program Files (x86)". The x86 portion in the folder name indicates that 32-bit applications are in this folder. The root of the system drive, typically represented by the letter C:\, does contain some entries of applications that have been installed there. Python 2.7 used to be one such example, among others. With all this being said, users that are going through an installation process typically can choose where to install the application. If your survey tool is only checking these three locations, you might miss one if a user decided to install an application in their Documents or Downloads folder.

© SANS Institute 2024

https://linktr.ee/offsecexam

C:\



C:

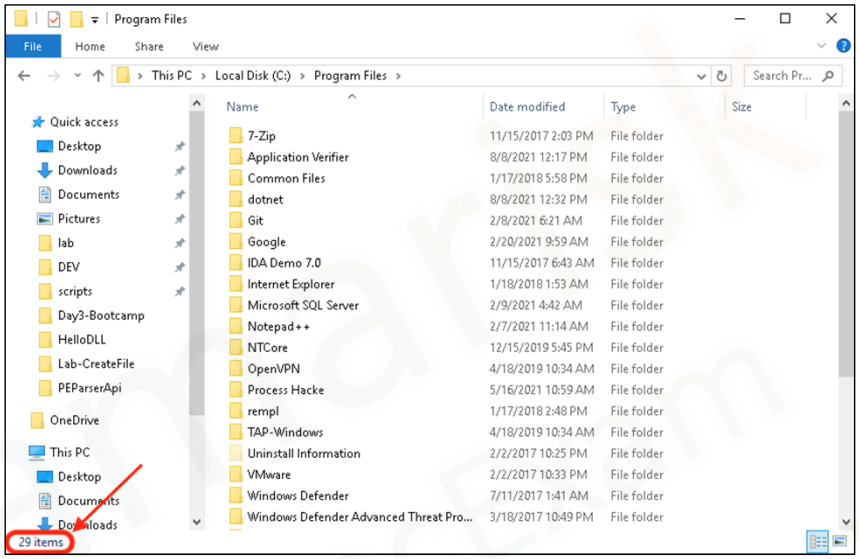
The system's root drive is typically annotated by the drive letter C, but it can be another letter. Because it can be any letter, it is best to not hard code file paths until you have determined the system root drive. Once done, you can append your file paths to that returned value. Back to the current topic, as you can see from the screenshot, there is not a lot being listed here as far as figuring out what software might be installed. We do have Python27 as a dead giveaway, but beyond that we would have to look elsewhere for our information.

One thing to note with folders like these would be permissions, because not everyone can write to these folders without the proper permissions.

© SANS Institute 2024

https://linktr.ee/offsecexam

C:\Program Files



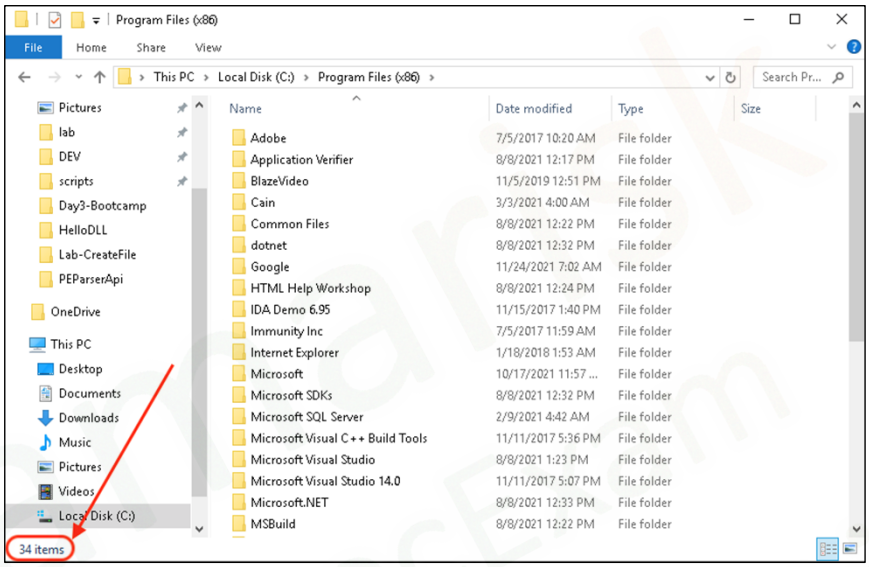
C:\Program Files

This screenshot shows the contents of the Program Files directory. The directory contains 29 entries, which could be a possible indicator that there are at least 29 applications that have been installed on this system; at *least* 29 items because some folders could easily hold other programs. If your tool was collecting this information then it would allow a red team operator to get a glimpse as to what 64-bit applications are here, and based on the applications, a guess could be made as to what the system's purpose is. Seeing applications like Notepad++, Process Hacker, VMware, etc. could indicate that this machine could be a research VM. This screenshot was taken from a Windows 10 Dev VM.

© SANS Institute 2024

https://linktr.ee/offsecexam

C:\Program Files (x86)



C:\Program Files (x86)

This screenshot shows the contents of the Program Files x86 directory. This directory contains 34 entries, which could indicate that there are at least 34 applications installed on this system that are 32-bit. An operator might be able to make a better educated guess as to what this system is being used for after seeing several entries for development software.

© SANS Institute 2024

https://linktr.ee/offsecexam

Should Operations Continue?

When should you abort an operation?

Aborting an operation based solely on a single application being installed is quite the decision to make. If you have no idea what the application does or what it would do if you drop more tools on the system, it could be a good decision to back off. This would allow you more time to conduct some research and hit the target later, possibly.

Should Operations Continue?

When deciding if you should continue with an operation, there is not always a clear yes or no answer. The more enumeration you do, the more informed your decision could be and the more confident you can feel about the decision. In a makeshift scenario, let us say you have gathered the following information: OS info/version, service pack/hotfixes, processes, and installed software. You might have enough information to make an informed decision. If you noticed that there was an application that detects your tool, then perhaps it would be a good idea to back off and find a different system to target. For the applications that you have never heard of before and have not had any time to research, you might have to decide if the risk of bringing more tools down is worth the reward. All of this should be done ahead of time, as much as possible. That way, when you come to a decision point, you already know what to do. The discussions could happen in an operational pre-brief where your team has a chance to brainstorm together. GO and NOGO criteria can be made during these pre-briefs where you know if it is okay to continue or if you have to back off right away.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Explored where applications can be installed

Learned to make a decision to continue or abort based on the listing of software

Module Summary

In this module, we explored where some applications might be installed, and we also discussed how GO/NO-GO decisions can be made based on gathered information.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

In this module, we will discuss a feature for enumerating directories. Many implants today already implement a directory listing feature, so we should put one in ours.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Understand how to enumerate files in a directory
- Understand how to implement a recursive directory walk; dirwalk
- Learn how to locate a specific file of interest

Objectives

The objectives for this module are to understand how to enumerate files in a directory, understand how to implement a recursive directory walk, and learn how to locate a specific file of interest.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Directory Enumeration



Directory listings is nothing new.

Many popular frameworks have implants that can perform directory listings. The famous Meterpreter session from the Metasploit Framework offers operators the ability for perform a directory listing. Native Windows binaries also perform directory listings, so it is not a behavior that should be categorized as malicious or suspicious.

Directory Enumeration

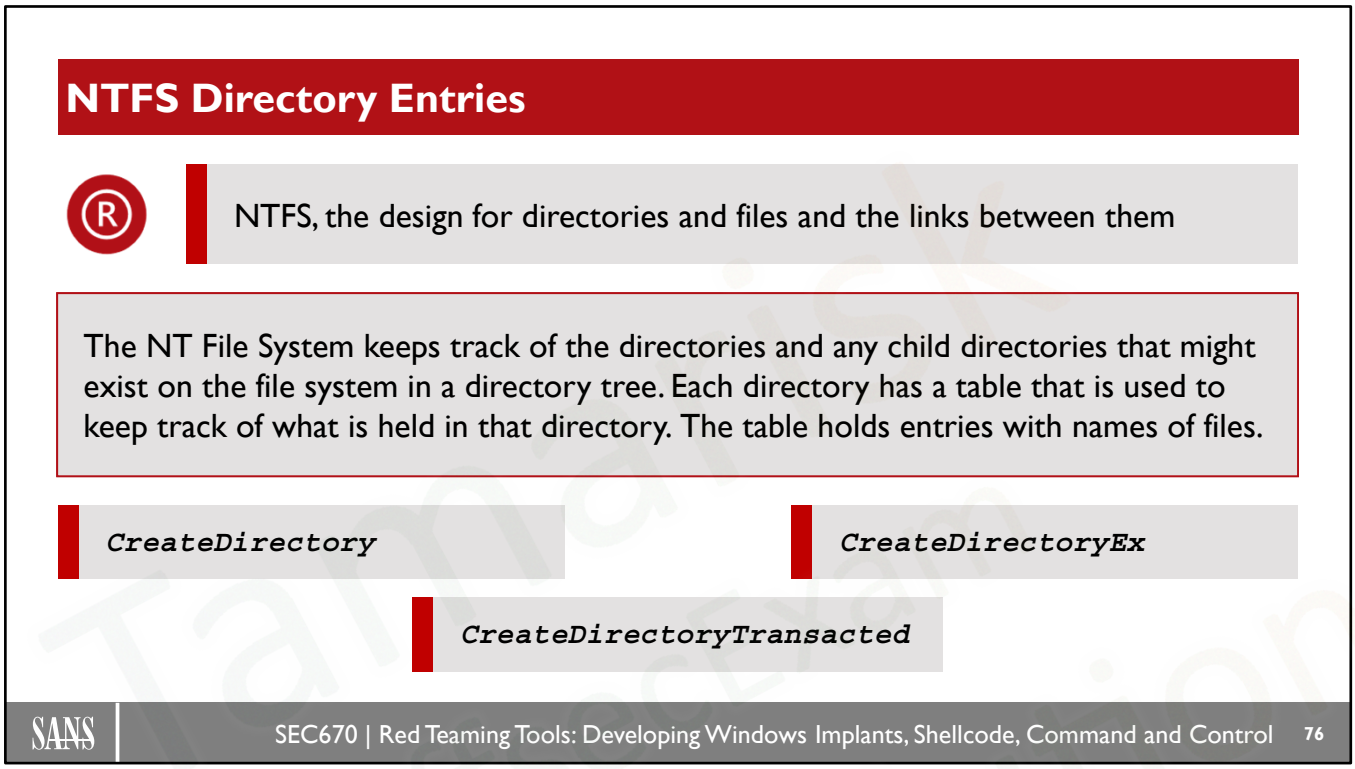
Directory enumeration is a very simple feature to implement programmatically, and it can be done in a number of different ways. MSDN provides a few simple examples of using the primary APIs involved and it is easy to build upon those examples. If you are familiar with the Metasploit Framework and its Meterpreter session, you might be familiar with its *ls* command to list the contents of a directory. Of course, *ls* is a native Linux command, but Meterpreter implements this as a platform agnostic function for Windows or Linux in their source code. The C source code for Meterpreter's *ls* command uses the exact same Windows APIs that we will be discussing in this module and for our lab. You can browse to their GitHub page to look at the code to get an idea of what is happening behind the scenes.

Enumerating directories is an important feature to implement because you would want to give your operators the ability to see what is in a certain directory. Perhaps the operator will want to download a file of interest or want to see what files are currently in a folder before making the decision to drop a file of their own in that same folder. The operator will do whatever in the end, but we must at least give an operator the ability to perform a directory listing.

Here is the URL to Raymond Chen's article for BFS: https://devblogs.microsoft.com/oldnewthing/20050203-00/?p=36533.

© SANS Institute 2024

https://linktr.ee/offsecexam

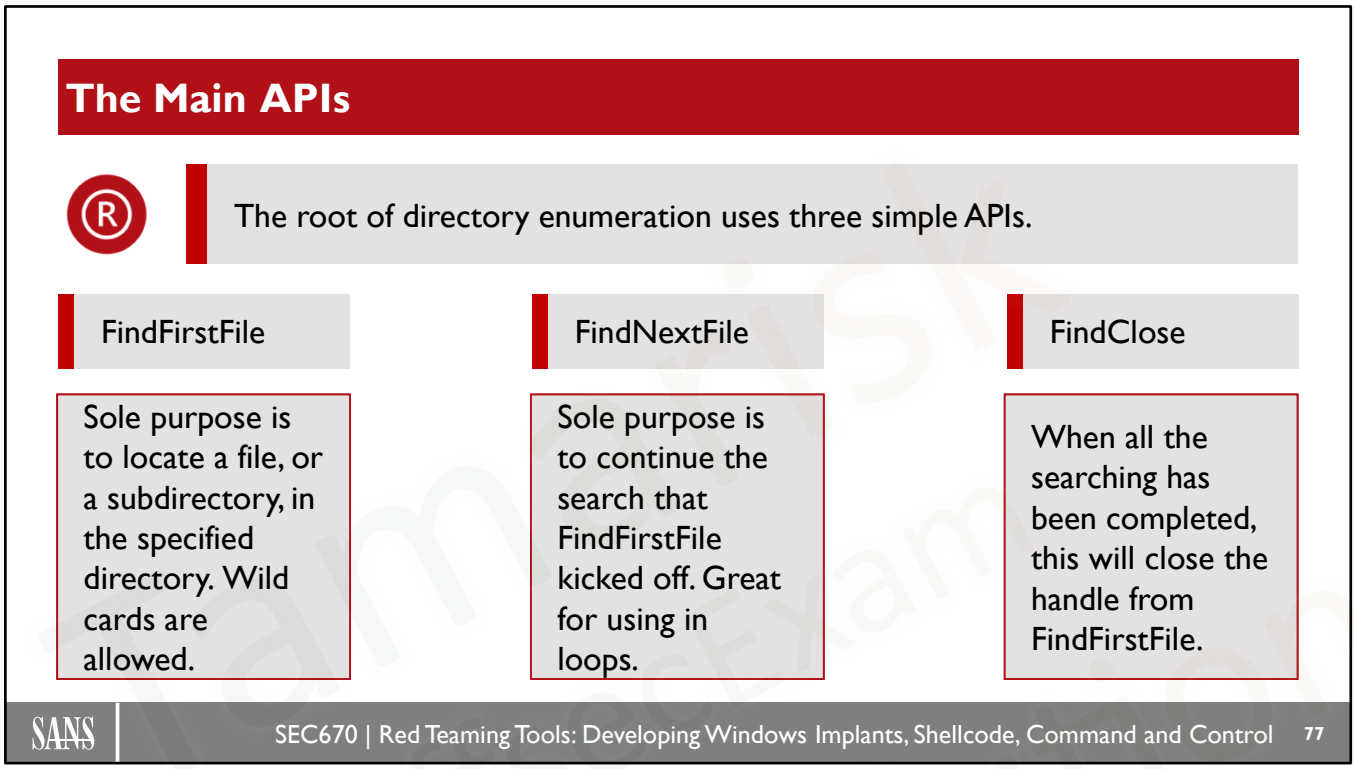


NTFS Directory Entries

You might already be familiar with how Explorer shows directories and the files in them, but what really makes that happen? When a directory is created via one of the three APIs listed on the slide—***CreateDirectory***, ***CreateDirectoryEx***, or ***CreateDirectoryTransacted***—it will have a corresponding directory table made for it. Whenever a file is created or moved into this directory, an entry will be placed into the table, and the entry will have the name of the file. There can be multiple entries for the same file in the directory table called links. A hard link is created when there is another entry in the table that is made for the same file. Trying to view the table itself is not important to us as that starts to dive down into the world of forensics. We will simply be using the APIs to do the querying of the tables for us.

© SANS Institute 2024

https://linktr.ee/offsecexam



The Main APIs

The act of enumerating directories is not malicious, nor should it even be deemed as suspicious. There are several native Windows applications that have this functionality built into it. Explorer must do this very same thing; in fact, we could search through many executables in the System32 folder to determine how many of them import these functions. CTF players could whip up a directory enumeration script to locate flag files or whatever their objective is. Nation State actors might enumerate directories to see if any files are worth downloading for further analysis. Actors that might be interested in espionage might conduct recursive directory walks looking for blueprints of engine designs for fighter jets, ships, stealth technology, etc. To get this done, there are two primary APIs that do the bulk of the work: ***FindFirstFile*** and ***FindNextFile***. ***FindClose*** is simply to clean things up when the search is done. The APIs also have extended versions like ***FindFirstFileEx***, so we could pass in extra attributes that the file should have. There is also a transacted API like ***FindFirstFileTransacted***, though it is not recommended to be used.

© SANS Institute 2024

https://linktr.ee/offsecexam

FindFirstFile API

FindFirstFileA()

Used to obtain a search handle

Has HANDLE return type

```
HANDLE FindFirstFileA(
 _In_ LPCSTR lpFileName,
 _In_ LPWIN32_FIND_DATAA pFindFData
);

// example
HANDLE hSearch = INVALID_HANDLE_VALUE;
WIN32_FIND_DATA FindData;

hSearch = FindFirstFileA(Dir, FindData);
```


FindFirstFile API

The first API that we need to use to kick of the directory walk is none other than the ***FindFirstFile*** API. Like most other APIs that have string arguments, it is just a macro that is expanded to support Unicode or ANSI depending on your project settings. The example on the slide is using the ANSI version. Should you have to deal with non-English characters, like Chinese, you better use the Unicode versions. The main purpose of the API is to return a search handle that the ***FindNextFile*** API would then use to continue searching for a file in a directory. The ***FindFirstFileA*** function has just two parameters, which are broken down below.

lpFileName, of type LPCSTR, is the directory and filename that should be searched. The filename can have wildcards in it too, which is nice when you do not know an extension for a file.

pFindFData, of type LPWIN32_FIND_DATAA, is the pointer to a WIN32_FIND_DATA structure that will be filled out by the API as it retrieves the information about a file or subdirectory.

If the function succeeds, a file search handle will be returned and the WIN32_FIND_DATA structure will start to be filled out as you progress. On failure, it will return INVALID_HANDLE_VALUE, so you would have to call ***GetLastError*** to determine what really was the cause.

© SANS Institute 2024

https://linktr.ee/offsecexam

WIN32_FIND_DATA Struct

```
typedef struct _WIN32_FIND_DATAA {
 DWORD    dwFileAttributes;
 FILETIME ftCreationTime;
 FILETIME ftLastAccessTime;
 FILETIME ftLastWriteTime;
 DWORD    nFileSizeHigh;
 DWORD    nFileSizeLow;
 DWORD    dwReserved0;
 DWORD    dwReserved1;
 CHAR     cFileName[MAX_PATH];
 CHAR     cAlternateFileName[14];
 DWORD    dwFileType;
 DWORD    dwCreatorType;
 WORD     wFinderFlags;
} WIN32_FIND_DATAA, *PWIN32_FIND_DATAA, *LPWIN32_FIND_DATAA;
```


WIN32_FIND_DATA Struct

The WIN32_FIND_DATA structure is filled with useful information. Many of the struct members do not need any explanation, like FileAttributes, CreationTime, LastAccessTime, LastWriteTime, etc. One that might need some explaining is the AlternateFileName with the fixed size of 14. This is for the short file naming conventions that NTFS supports. Specifically, it is an 8.3 convention with the filename containing 8 letters, then the dot, then the 3-letter extension. As an example, if you have a long file name, like backup-picture.jpg, it could look like this: BACKUP~1.JPG. The other members are not that interesting to us, so we can simply ignore them.

© SANS Institute 2024

https://linktr.ee/offsecexam

FindNextFile API

- FindNextFileA()
- Used to continue a search
- Has BOOL return type

```
BOOL FindNextFileA(
 _In_ HANDLE hFindFile,
 _In_ LPWIN32_FIND_DATAA pFindFileData
);

// example

do {

 // do stuff with the info

} while (
  FindNextFileA(hSearch, FindData) != 0
);
```


FindNextFile API

The next API we need to use to implement our directory walk is the ***FindNextFile*** API, which is really a macro that expands to ***FindNextFileA*** for ANSI, or ***FindNextFileW*** for Unicode. For this example, we will be using the ANSI version of the macro. The ***FindNextFileA*** function will not work on its own—it must have the file search handle that the ***FindFirstFileA*** function returns. Just like ***FindFirstFileA***, this function has two parameters: a handle to a valid file search handle, and a pointer to a WIN32_FIND_DATAA structure. Since handles are nothing new to us at this point and since the structure is the same one that is used in the ***FindFirstFileA*** function, there is no need to break down the parameters. Because the return type is BOOL, it is perfect to use in a loop like a while loop. Each iteration of the loop for each file found in a directory would have the structure filled out for the respective file. When the function returns false, the loop would break and be done.

If the function fails for whatever reason, the structure may not have been filled out properly, so it would be best to call ***GetLastError*** to see what really happened.

© SANS Institute 2024

https://linktr.ee/offsecexam

FindClose API

- FindClose()
- Used to close a file search handle
- Has BOOL return type

```
BOOL FindClose(
 _Inout_ HANDLE hFindFile
);

// example

FindClose( hsearch );
```


FindClose API

After you are done searching for files or performing your directory walk, the file search handle should be closed out. The ***FindClose*** API can do this for us, and it is a very simple API to understand and implement in code. ***FindClose*** only takes one argument and that is a valid file search handle. The return type is BOOL so you could check to see if the function was successful or not, and if it was not successful, you would call ***GetLastError*** to see the details.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: FindFirstFile, FindNextFile, FindClose

```
// attempt to enumerate the hard links to the file
hFileToFind = FindFirstFileW(fileName.GetBuffer(), &findData);
// error check
if (INVALID_HANDLE_VALUE == hFileToFind) { ... }
else
{
    do
    {
        // do work...
    } while (FindNextFileW(hFileToFind, &findData));
}
```


Example: FindFirstFile, FindNextFile, FindClose

The short example on the slide shows very basic usage for the two main APIs that are involved in enumerating a directory. The ***FindFirstFileW*** API is used to kick off the process of enumeration. The first argument being passed to the API is a CString type that has a method ***GetBuffer()*** to get a pointer to the buffer. This is done to satisfy the requirement of the API. The function can fail, so be sure to check for success for failure. The next part is the do/while loop that will continue as long as the ***FindNextFileW*** function keeps returning TRUE or 1. In the body of the do/while loop is where your processing of each entry would be done. You can do matching if you are looking for a certain file name, extension type, skip directories, etc. If you are keeping a list of everything being discovered, this is where you would be appending to that list.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.5: FileFinder



Enumerating directories is an important feature to create.

Please refer to the eWorkbook for the details of the lab.

Lab 2.5: FileFinder
Please refer to the eWorkbook for the details of the lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was the explore how you can programmatically enumerate a directory to find a file, and if you had time, enumerate any subdirectories.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed why we would perform a directory walk
- Learned how to perform a directory walk to find a file
- Discovered the main APIs involved

Module Summary

In this module, we discussed the how to perform a directory walk and why we would do one in the first place. A recursive walk from the system root could take a while, but at the same time it could yield some great information. There were only three APIs involved with this, but there were two that did the heavy lifting: ***FindFirstFile*** and ***FindNextFile***.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What user-mode structure holds the attributes of a file?

A WIN32_FIND_DATA

B KUSER_SHARED_DATA

C FILE_OBJECT

Unit Review Questions
Q: What user-mode structure holds the attributes of a file?

A: WIN32_FIND_DATA

B: KUSER_SHARED_DATA

C: FILE_OBJECT

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What user-mode structure holds the attributes of a file?

A WIN32_FIND_DATA

B KUSER_SHARED_DATA

C FILE_OBJECT

Unit Review Answers
Q: What user-mode structure holds the attributes of a file?

A: WIN32_FIND_DATA

B: KUSER_SHARED_DATA

C: FILE_OBJECT

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

This module will discuss the importance and benefits of gathering information about the users of a system. The module will of course discuss the Windows APIs that might be involved with retrieving user information.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss the importance of gathering user information
- Understand how to programmatically gather information about users

Objectives
The objectives for this module are to understand how to programmatically gather information about users and discuss the importance behind it.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

User Information

Who's who on the system

It is always a good idea to see what users are on the system. Limited privileged users are one thing, but finding out if a user is part of the Administrators group is great. You might even get lucky enough to see a Domain admin logged into a system!

User Information

Another part of conducting recon is gathering user account information. From the command line, there are some common tools that can be used to do this, like the *net* command line utility. The *net* command does more than gather user information but if you pass in the *net user* or the *net localgroup* options, it will gather user information and groups on the system, among other things. Using the command line is nice when you have a shell, but we can also do this programmatically. Why do we care about users in the first place? Well, if a user is part of the Administrators group, that could give us a solid option of escalating our privileges or attempting a UAC bypass. If the system is part of an Active Directory domain, something you would find in most mature Windows environments, you might see a domain admin logged into the system. The hash for that account would be great to grab and then further pivot around the domain. Dumping creds and hashes for accounts can help with our lateral movement, or as some say our East and West movement, something that SEC565 dives into.

© SANS Institute 2024

https://linktr.ee/offsecexam

GetUserName API



GetUserNameA()

Used to obtain the current username

Has BOOL return type

```
BOOL GetUserNameA(
 _Out_   LPSTR   lpBuffer,
 _Inout_ LPDWORD pcbBuffer
);

// example

PSTR userName;
DWORD cbSize = 32767;

GetUserNameA(userName, &cbSize);
```


GetUserName API

The ***GetUserName*** API is yet another macro that can expand to support ANSI or Unicode, depending on project settings. Keeping with the trend of showing ANSI versions, we will break down the ANSI version of this API. ***GetUserNameA*** requires two parameters: *lpBuffer*, and *pcbBuffer*. We can take a look at those in more detail below.

lpBuffer, of type LPSTR, is a pointer to a buffer that will end up storing the username.

pcbBuffer, of type LPDWORD, is a pointer to a variable that will indicate the size of the buffer pointed to by lpBuffer.

The function will indicate success by returning nonzero. Failures result in zero, which does not give us any real insight as to why it may have failed. For that, we would have to call our favorite error function ***GetLastError***. There are a few caveats with this function in that it will not return the username that one might expect. For example, threads have the ability to impersonate tokens/users and as such, the API would return the username of the client that is currently being impersonated.

© SANS Institute 2024

https://linktr.ee/offsecexam

GetUserProfileDirectory API

GetUserProfileDirectoryA()

Used to obtain the root directory of the user's profile

Has BOOL return type

```
USERENVAPI
BOOL
GetUserProfileDirectoryA(
 _In_      HANDLE  hToken,
 _Out_opt_ LPSTR   lpProfileDir,
 _In_opt_  LPDWORD lpcchSize
);
```


GetUserProfileDirectory API

The ***GetUserProfileDirectory*** API is useful when you would like to know the path of the root folder for the username that was passed into the function. The ***GetUserProfileDirectory*** API is another macro, so we should know what that means by now. The function only has one required parameter and two optional parameters. Even though a few of the parameters are optional, we can dive into the details of each one.

hToken, of type HANDLE, is a valid handle to a token for the user that can be gathered from calling a function like ***OpenProcessToken*** or ***OpenThreadToken***.

lpProfileDir, of type LPSTR, is a pointer to a variable that will hold the path to the user's profile directory.

lpcchSize, of type LPDWORD, is a pointer to the size of the buffer pointed to by lpProfileDir.

One nice thing about this function is that if your *lpProfileDir* buffer is not big enough, the needed size will be placed in *lpcchSize* by the function. This would allow you to make the call a second time using the correct size. On that note, your first call for functions like these should be with a zero-sized buffer. This would make the function fail, forcing it to give us the proper size. Once you have that, you can make the second call with the correct size. This is the preferred way and the most reliable method.

© SANS Institute 2024

https://linktr.ee/offsecexam

NetUserEnum API



NetUserEnum()

Used to obtain information about all user accounts

Has NET_API_STATUS return type

```
NET_API_STATUS
NET_API_FUNCTION
NetUserEnum(
 _In_    LPCWSTR servername,
 _In_    DWORD   level,
 _In_    DWORD   filter,
 _Out_   LPBYTE *bufptr,
 _In_    DWORD   prefmaxlen,
 _Out_   LPDWORD entriesread,
 _Out_   LPDWORD totalentries,
 _Inout_ PDWORD  resume_handle
);
```


NetUserEnum API

Getting account information about a single user account is fine, but it is also great to get account information about all user accounts on a system. The ***NetUserEnum*** API is perfect for this because it will return information from all user accounts on the local or remote system. The information returned from the API might seem similar to the output you may have seen after executing the *net user* command. The API has several parameters that need some explaining.

servername, of type LPCWSTR, is the pointer to a constant wide char string for the server. Passing in NULL here would indicate the local computer.

level, of type DWORD, is the level of information you intend to see. There are seven values that can be passed in here:
- 0 - is for user account names. There will be an array of USER_INFO_0 structures.
- 1 - is for user account details. There will be an array of USER_INFO_1 structures.
- 2 - is for account details and logon information. There will be an array of USER_INFO_2 structures.
- 3 - is for all of the above and now profile information. The array will have USER_INFO_3 structures.
- 10 - is for user account names and comments. The array will have USER_INFO_10 structures.
- 11 - is for more detailed account information. The array will have USER_INFO_11 structures.
- 20 - is for the user's names account attributes. The array will have USER_INFO_20 structures.

filter, of type DWORD, is used to specify what accounts should be included in the search.

bufptr, of type LPBYTE, is the pointer to the buffer that will end up holding the returned information. You are required to free this buffer using ***NetApiBufferFree*** when done, even if the API fails.

prefmaxlen, of type DWORD, is the maximum length of the data, in bytes. We do not need to specify a value here other than MAX_PREFERRED_LENGTH so that the function makes the proper space for the data.

© SANS Institute 2024

https://linktr.ee/offsecexam

entriesread, of type LPDWORD, is a pointer to the variable that will hold the number of entries the function queried.

totalentries, of type LPDWORD, is a pointer to the variable that will hold the number of entries that could have been queried from a position called the resume position.

resume_handle, of type PDWORD, is a pointer to a variable that is used as the resume handle. The resume handle can be used to continue searching user accounts and if this is what you want to do, then zero (0) should always be used for the first call. If you do not care about this, then passing NULL here is just fine.

Tamarisk
OffsecExam
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

NetLocalGroupEnum API

NetLocalGroupEnum()

Used to obtain local group information

Has NET_API_STATUS return type

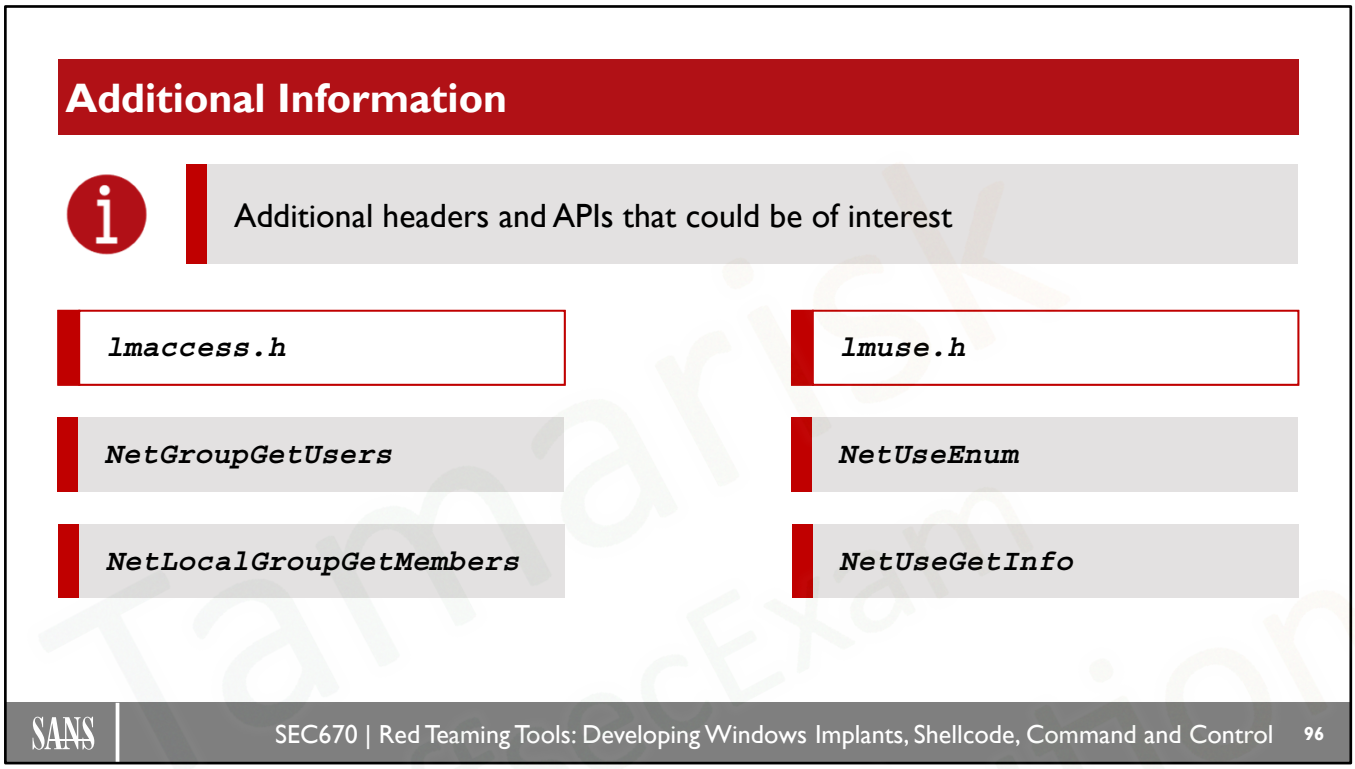
```
NET_API_STATUS
NET_API_FUNCTION
NetLocalGroupEnum(
 _In_    LPCWSTR  servername,
 _In_    DWORD    level,
 _Out_   LPBYTE  *bufptr,
 _In_    DWORD    prefmaxlen,
 _Out_   LPDWORD  entriesread,
 _Out_   LPDWORD  totalentries,
 _Inout_ PDWORD_PTR resumehandle
);
```


NetLocalGroupEnum API

In addition to gathering user information, we can gather information about the groups that might be present on a local or remote system. The ***NetLocalGroupEnum*** API is pretty much the equivalent to executing the *net localgroup* command from the command line. Perhaps the API should have been named NetLocalOrRemoteGroupEnum since it is not bound to local systems only. The API has almost the exact same parameters to the ***NetUserEnum*** API except for the level parameter. The level parameter only has two possible options: 0 and 1. Level 0 is for returning only the names of the local groups, which will be an array of LOCALGROUP_INFO_0 structures. Level 1 is for requesting the group names and the comments that are tied to them, if any. There will be an array of LOCALGROUP_INFO_1 structures for Level 1. Outside of the level parameter, the remaining ones are the same as the ***NetUserEnum*** API.

© SANS Institute 2024

https://linktr.ee/offsecexam

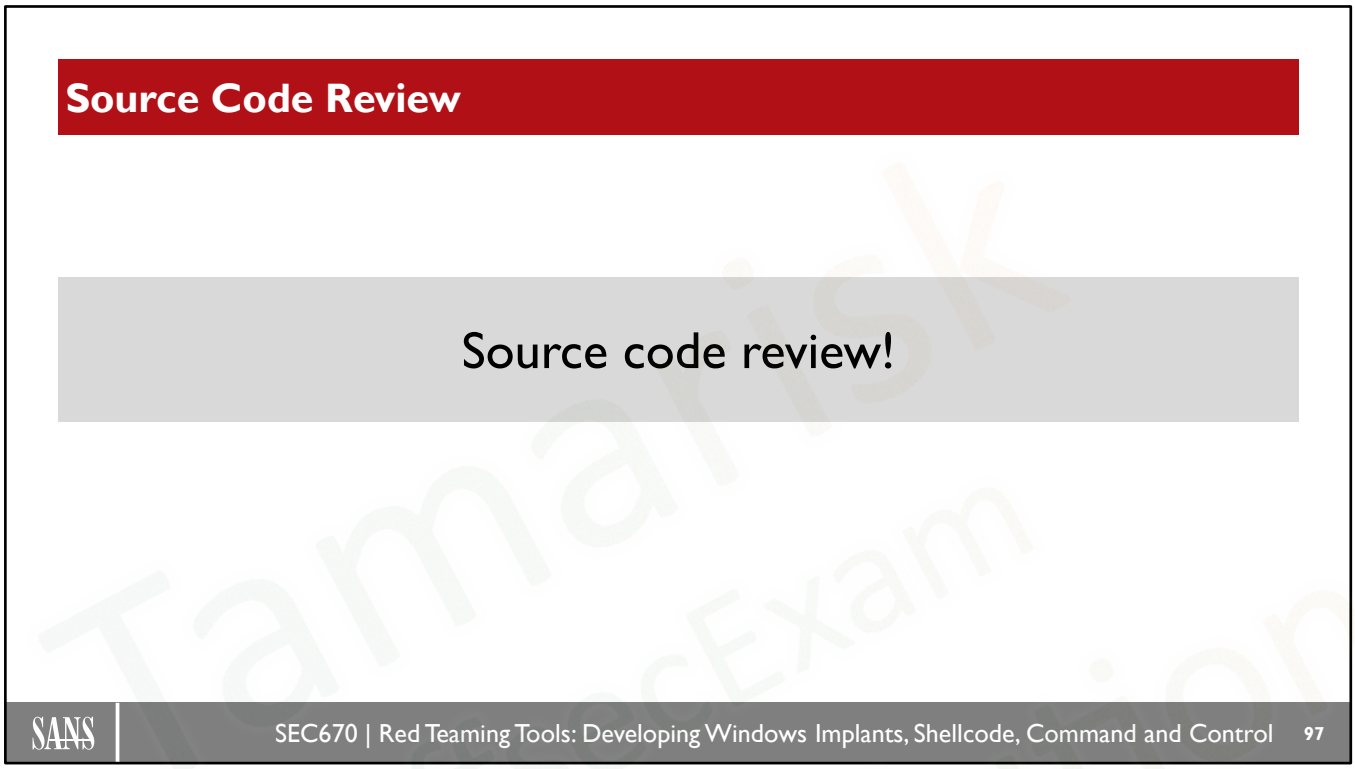


Additional Information

The lmaccess and the lmuse header files offer additional APIs that might be of interest when querying user and user group information. The listing on the slide is not an exhaustive list but merely a small sampling of what else is out there that can be used. Depending on the information that you are wanting to gather, you can implement the logic to get it. The more you add the more robust your survey tool will be for operators using it.

© SANS Institute 2024

https://linktr.ee/offsecexam



Source Code Review
Time to jump into the source code and understand it.

Tamarisk
OffsecExam
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Discussed reasons to gather user information

Explored several APIs that allow us to retrieve user and group information

Module Summary

In this module, we discussed why it might be important to gather information about the user and groups on systems. We also took a detailed look at several of the APIs that are available for us to use.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is one API to obtain a username?

A GetUserName()

B NetUserEnum()

C NetLocalGroupEnum()

Unit Review Questions
Q: What is one API to obtain a username?

A: GetUserName()

B: NetUserEnum()

C: NetLocalGroupEnum()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is one API to obtain a username?

A GetUserName()

B NetUserEnum()

C NetLocalGroupEnum()

Unit Review Answers
Q: What is one API to obtain a username?

A: GetUserName()

B: NetUserEnum()

C: NetLocalGroupEnum()

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

In this module, we will discuss how to enumerate services and tasks during the execution of your survey tool.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Understand Windows services
- Compare services and processes
- Understand Windows Tasks
- Discuss how to enumerate services and tasks

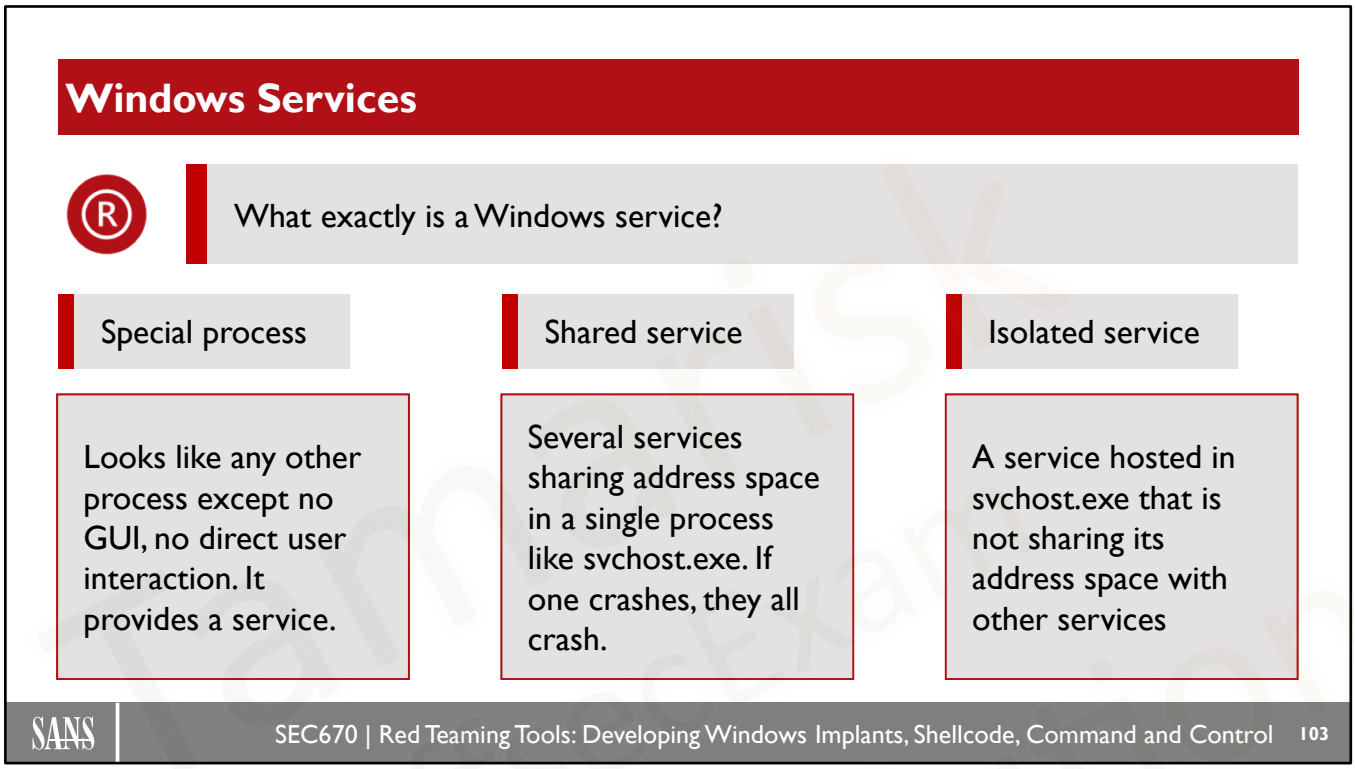
Objectives

The objectives for this module are to understand what a Windows service is, compare services and processes, understand what Windows Tasks are, and how to enumerate them all.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam



Windows Services

Even if you are not that familiar with Windows, you might be familiar with services in general. Linux has services too, typically called daemons, and they provide a certain set of services to users, like FTP access to some FTP server. Windows services are not that different as they still provide a service to users. Windows services are just special processes and they typically do not have a GUI. Most services do not have direct configuration or interaction with users on the system. The services operate in the background and mind their own business until a user needs whatever functionality they provide. The most interaction a user might have with a service is to perform some action against it, like Stop or Start actions. Another interesting item with Windows services is that they can share address spaces. The svchost.exe process acts like a container of sorts so that several services can be hosted in a single svchost.exe process. The downside of shared services is that if one service crashes for some abnormal reason, like exploit attempts, the other services go down with it. This is where isolated services come to play. They are services that can be hosted in a single svchost.exe process but they will not be sharing the address space with any other service. Later in the course, we will look at how we can make that choice programmatically when we create our own service.

© SANS Institute 2024

https://linktr.ee/offsecexam

Service Enumeration

Why enumerate services?

Awareness

The better awareness you have the more successful your operation. Detect services that could be vulnerable or ones that could belong to AV/EDR.

Purpose

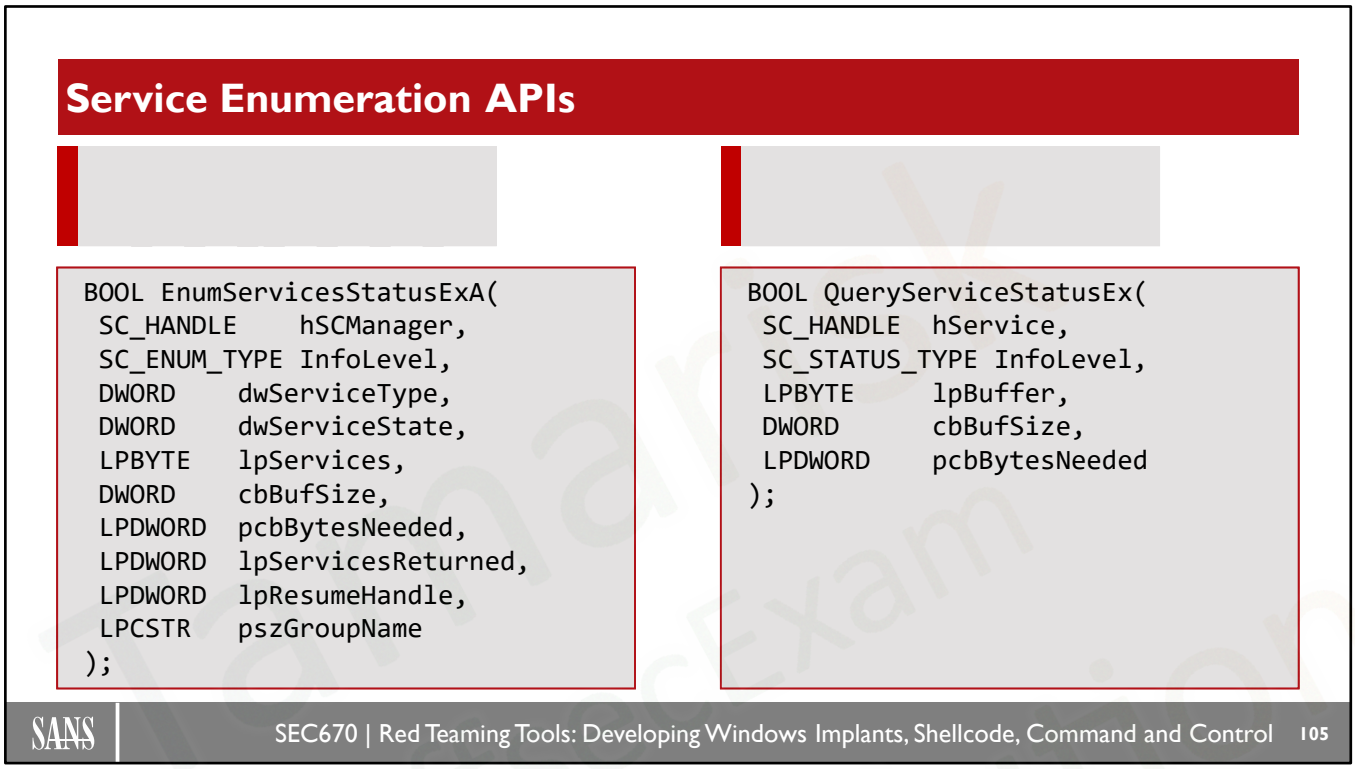
The purpose of a target will determine how it is most likely being used. It could also indicate if the target is high visibility or low.

Service Enumeration

We typically conduct service enumeration for similar reasons that we conduct process enumeration. Services, just like processes, can tell us what a target's purpose is or how it is being used. Certain services would be specific to servers like DHCP, DNS, FTP, to name a few, and you would not expect to find those services running on a client workstation. AV/EDR products can also have services running that handle certain portions of its functionality like scanning, file submission, watchdog services, etc. Finding those could be beneficial in making the decision to continue operating on that target. Again, if you did not expect that your target would have an AV solution installed, it might not be the best idea to continue and bring down additional tools. Another reason for service enumeration is to identify possible vulnerable services that could be exploited for privilege escalation and/or persistence, both of which will be talked about later in the course in detail.

© SANS Institute 2024

https://linktr.ee/offsecexam



Service Enumeration APIs

How exactly can we go about enumerating services? You might know how to do this via the command line using the sc.exe utility or with the PowerShell cmdlet Get-Service, but we are doing this programmatically in C. We have two options at our disposal for getting this done. We could create WMI queries to do this or we can use a few Win32 APIs to achieve similar results. For this instance, we will be using the service-specific Win32 APIs instead of using wmi-specific Win32 APIs. ***EnumServicesStatusEx()*** and ***QueryServiceStatusEx()*** are two great APIs that can help you enumerate services and gather more detailed information about them. The first function on the slide is ***EnumeServicesStatusExA()*** and it accepts a decent number of arguments. The function will query the Service Control Manager database of either a local target or a remote target for services and will return the name and status of each service, at a minimum. More information and data can possibly be gathered depending on the InfoLevel value passed in as an argument, which the only supported InfoLevel value documented at the time of this writing is SC_ENUM_PROCESS_INFO.

The other great service-specific API to use is ***QueryServiceStatusEx()***. It does not enumerate services but rather, it will obtain the status of a service that has already been enumerated. The two APIs used in conjunction with each other can provide valuable information about services on the target. The InfoLevel for this function, similar to ***EnumServicesStatusEx***, only supports one value: SC_STATUS_PROCESS_INFO.

In Section 3, we will dive much deeper into services when we discuss local damage that can be inflicted against a target.

© SANS Institute 2024

https://linktr.ee/offsecexam

Windows Tasks



What are Windows Tasks?

Automating routine tasks is great for sysadmins and users who do not want the headache of repeating an action over and over. Tasks are great for attackers to aid in persistence on a target.

Create new tasks

Hijack current tasks

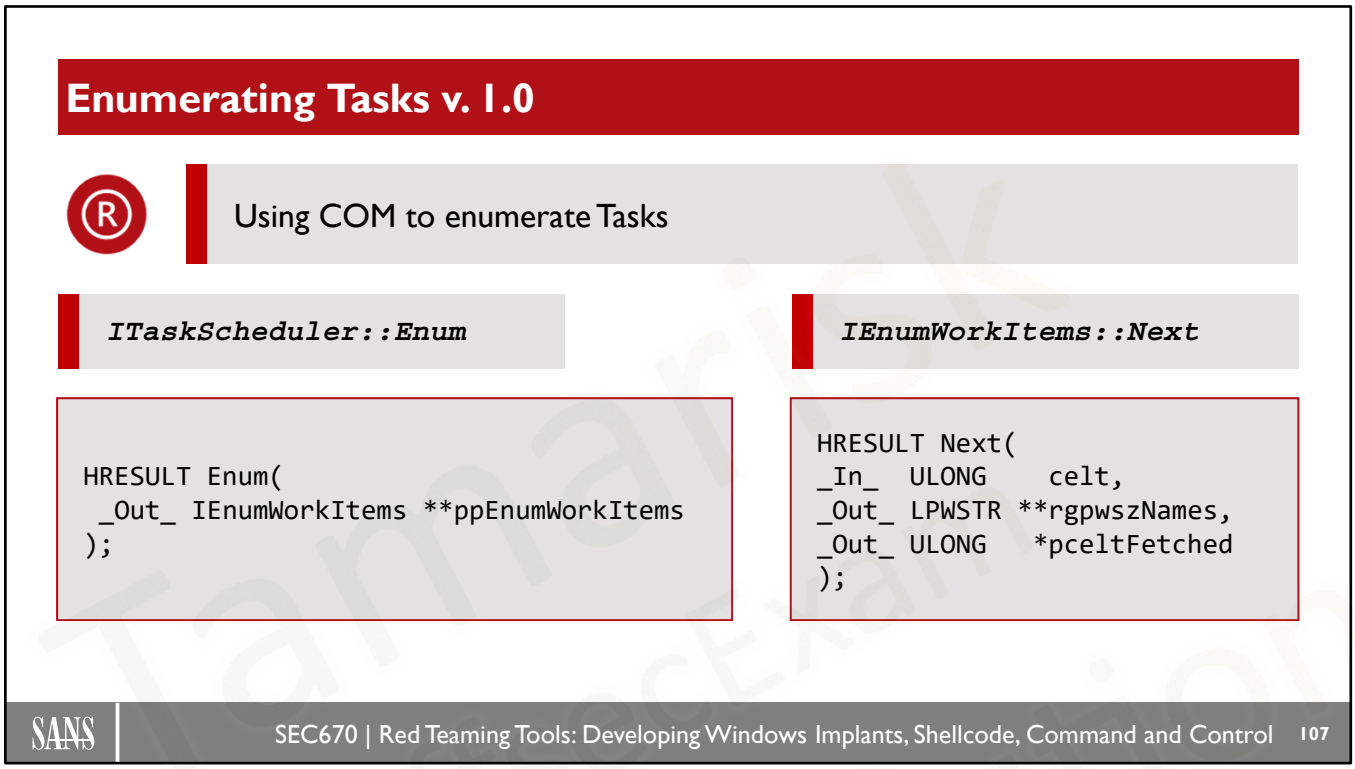
Windows Tasks

Admins, developers, hackers, etc. tend to be lazy in the sense that all would rather script something than do it manually. Tasks are a great way to help with this because they can perform certain actions when a specific trigger condition is met. Some of the triggers could be a user logging on to the system, the system itself booting up, certain event logs, etc. From an attacker's perspective, tasks would be a great option for maintaining access to the target. Attackers can either create a new task to achieve their persistence needs or hijack current tasks to be a bit less noisy. Hijacking currently scheduled tasks could be done by adding an additional action after the default action. For enumeration purposes, we are interested in seeing what tasks are currently registered and possibly already running. Perhaps we find a task that could be of interest to hijack, or we find evidence that we are not the first ones on the target, i.e., a current task for persistence made by another attacker. Tasks might not always be the most useful, but when conducting survey scripts, it is great to be thorough.

Please note: when it comes to keeping a low profile—something you should try to keep—creating new tasks could raise your profile a good amount. Create new tasks with caution.

© SANS Institute 2024

https://linktr.ee/offsecexam



Enumerating Tasks v. 1.0

For us to enumerate tasks on a system we must turn to COM. There is an exposed interface called ***TaskScheduler***, specifically ***ITaskScheduler***, that has a method called ***Enum***, which will allow us to create an enumeration object. Then, with that object in hand, we can create ***IEnumWorkItems*** interface that allows us to enumerate whatever tasks might be present at the time the code is ran. According to MSDN, here are the high-level steps to getting this done:

- Initialize the COM library using ***CoInitialize.***
- Use ***CoCreateInstance*** to make the Task Scheduler object.
- Call the Enum method of ***ITaskScheduler*** to create the enumeration object.
- Call the Next method of the ***IEnumWorkItems*** to enumerate tasks.
- Free the resources using ***CoTaskMemFree.***

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed what Windows services are
- Learned about services and processes
- Discussed what Windows Tasks are
- Discussed enumerating services and tasks

Module Summary

In this module, we discussed very briefly what services and tasks are. We also discussed why we would want to enumerate them, and the APIs involved for enumerating services and tasks.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What does SERVICE_WIN32_OWN_PROCESS indicate?

A The service shares its address space with other processes

B The service does not share its address space with other processes

C The service will be hidden from view

Unit Review Questions
Q: What does SERVICE_WIN32_OWN_PROCESS indicate?

A: The service shares its address space with other processes

B: The service does not share its address space with other processes

C: The service will be hidden from view

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What does SERVICE_WIN32_OWN_PROCESS indicate?

A The service shares its address space with other processes

B The service does not share its address space with other processes

C The service will be hidden from view

Unit Review Answers
Q: What does SERVICE_WIN32_OWN_PROCESS indicate?

A: The service shares its address space with other processes

B: The service does not share its address space with other processes

C: The service will be hidden from view

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



How do you get the COM library ready for use in your process?

A CoCreateInstance

B CoInitialize

C CoMemFree

Unit Review Questions
Q: How do you get the COM library ready for use in your process?

A: CoCreateInstance

B: CoInitialize

C: CoMemFree

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



How do you get the COM library ready for use in your process?

A CoCreateInstance

B CoInitialize

C CoMemFree

Unit Review Answers
Q: How do you get the COM library ready for use in your process?

A: CoCreateInstance

B: CoInitialize

C: CoMemFree

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions

What COM interface can be called to create an enumeration object?

A ITaskScheduler

B IUnknown

C IBelieve

Unit Review Questions
Q: What COM interface can be called to create an enumeration object?

A: ITaskScheduler

B: IUnknown

C: IBelieve

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What COM interface can be called to create an enumeration object?

A ITaskScheduler

B IUnknown

C IBelieve

Unit Review Questions
Q: What COM interface can be called to create an enumeration object?

A: ITaskScheduler

B: IUnknown

C: IBelieve

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

This module will look at how to gather information about the network and the target's network configurations.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Gather network information
- Gather NIC configurations

Objectives
The objectives for this module are to determine how to gather any network information we can, as well as the target's NIC configurations.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Network Information



What network is the target connected to?

Most enterprise computers will have a NIC configured with an IPv6 and an IPv4 address. IPv6 will be there even if the organization does not officially support it. Some computers could have dual NICs and be what can be called dual homed. A dual homed system is connected to two different networks and offers a great pivot point into a new environment.

Network Information

You might not be able to determine the purpose of a system just by gathering network information, but it definitely helps out. If your tool has network sniffing capabilities, you might get more information that way. Aside from sniffing packets, you can still gather some useful information. Perhaps one of the most interesting pieces of information to find out is coming across a system that has multiple NICs and is being dual homed. A dual homed system can be connected to two different networks at the same time. Some sysadmins have done this thinking it acts as a security boundary and that the NICs are separate, but Windows does its best to treat them as one. In a dual homed system, one NIC could be connected to the DMZ and the other could be connected to the trusted, internal network, or the intranet. Dual homed systems are great targets for us and other attackers because they could enable our movement throughout a network. Stuxnet would happily move through a dual homed system, as did the slammer worm way back in 2003.

We cannot forget about VLANs. These Virtual Local Area Networks virtually combine endpoints into a single broadcast domain, and it might be a good idea to look for that when you get on target. SEC660 discusses VLAN hopping and could be something of interest for developers to understand.

© SANS Institute 2024

https://linktr.ee/offsecexam

NIC Information/Configuration

IP Helper header file offers many great APIs for us to use.

GetIpStatistics

```
IPHLPAPI_DLL_LINKAGE
ULONG
GetIpStatistics(
 _Out_ PMIB_IPSTATS Statistics
);
```

GetAdapterAddresses

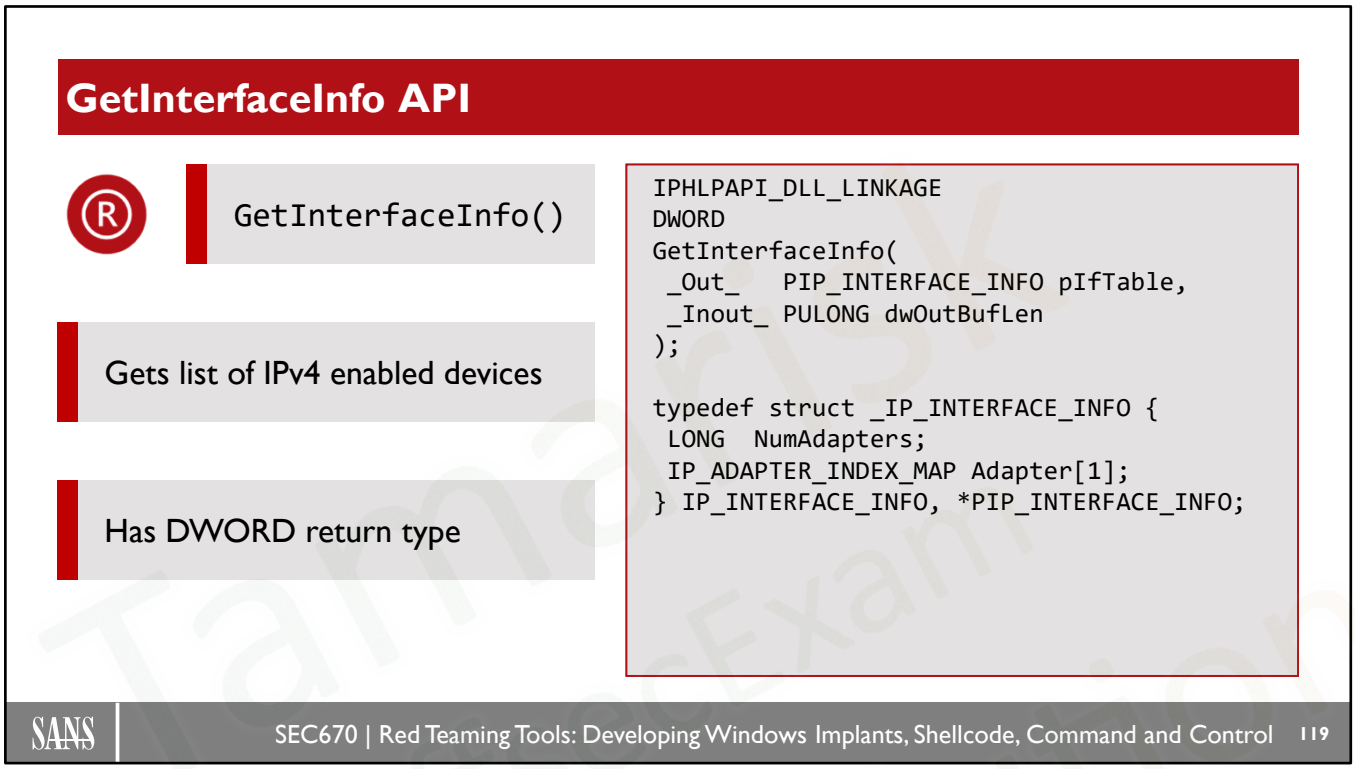
```
GetAdaptersAddresses(
 _In_ ULONG Family,
 _In_ ULONG Flags,
 _In_ PVOID Reserved,
 _Inout_ PIP_ADAPTER_ADDRESSES
AdapterAddresses,
 _Inout_ PULONG  SizePointer
);
```


NIC Information/Configuration

VMs are being used more and more these days in production and if you happen to get access to a VM in production, you might see several NICs on it. Your survey tool should gather each NIC's configuration. PowerShell offers some useful cmdlets for this, like Get-NetAdapter, Get-NetAdapterHardwareInfo, and more. The Windows command line offers utilities as well, like netstat, ipconfig, and others. Other interesting information that could be gathered would be the statistics, something that is gathered by running netstat -e. There are several ways to get all of this done, like using WMI or the IpHlpApi. While WMI can be useful, it is not always the easiest to work with when there are Win32 APIs like ***GetIpStatistics***. The IP Helper header file offers many useful APIs for us, and we will take a look at some of them, like ***GetIpStatistics***, ***GetInterfaceInfo***, ***GetAdapterAddresses***, and ***GetNumberOfInterfaces***.

© SANS Institute 2024

https://linktr.ee/offsecexam



GetInterfaceInfo API

The ***GetInterfaceInfo*** function can be used to gather a list of interfaces on the target that have IPv4 enabled. As you can see by the SAL annotations, the function has two parameters that are written to and one that is read. We can break down the parameters to better understand them.

pIfTable, of type PIP_INTERFACE_INFO, is used as a buffer to hold the list of interfaces that have been found. It is up to us as developers to allocate enough space in the buffer to hold this information. The structure holding the information is the IP_INTERFACE_INFO structure, which has two members. The first one, *NumAdapters*, holds the number of adapters that will be stored in the array. The second member is the Adapter, which is the array of IP_ADAPTER_INDEX_MAP entries. Every structure here will be tied to an index then tied to its corresponding name.

dwOutBufLen, of type PULONG, is a pointer to some DWORD variable that will be used to give the size of the buffer to the function. If you notice, this parameter is *inout*. The out portion is for when the size is not large enough to hold the data, the function will write to the variable the correct size needed. Then you can call this function again with the correct size and it should work just fine.

A successful call would return NO_ERROR. Failed calls can return any one of the following error codes:
- ERROR_INSUFFICIENT_BUFFER: The buffer is not large enough and the correct size has been stored in *dwOutBufLen*.
- ERROR_INVALID_PARAMETER: The *dwOutBufLen* is NULL or the function cannot write to the variable.
- ERROR_NO_DATA: Could not find any network adapters.
- ERROR_NOT_SUPPORTED: The function is not supported for this version of OS.

© SANS Institute 2024

https://linktr.ee/offsecexam

GetIpStatistics API

GetIpStatistics()

Grabs the IP statistics for the system

Has ULONG return type

```
IPHLPAPI_DLL_LINKAGE
ULONG
GetIpStatistics(
 _Out_ PMIB_IPSTATS Statistics
);
```


GetIpStatistics API

The ***GetIpStatistics*** function can help us recreate familiar Windows commands like netstat. The overall IP statistics are captured and returned to the caller of this function. The function is not very complex and accepts a single parameter: Statistics. The Statistics parameter is of type PMIB_IPSTATS, which is a struct that gets filled out by the function so that it has a place to store the information for any IP statistics that are gathered, hence the reason why the Statistics parameter is an out parameter. Upon success, the function will return NO_ERROR. When the function fails, it can return the following error code: ERROR_INVALID_PARAMTER for when the Statistics parameter is NULL or if the function is not able to write to the pointer. The structure the function fills out is very large, but the next slide holds a snippet of some of its members.

© SANS Institute 2024

https://linktr.ee/offsecexam

MIB_IPSTATS Struct

```
typedef struct _MIB_IPSTATS_LH {
[..SNIP..]
 DWORD dwDefaultTTL;
 DWORD dwInReceives;
 DWORD dwInHdrErrors;
 DWORD dwInAddrErrors;
 DWORD dwForwDatagrams;
 DWORD dwInUnknownProtos;
 DWORD dwInDiscards;
 DWORD dwInDelivers;
 DWORD dwOutRequests;
 DWORD dwRoutingDiscards;
 DWORD dwOutDiscards;
 DWORD dwOutNoRoutes;
[..SNIP..]
} MIB_IPSTATS_LH, *PMIB_IPSTATS_LH;
```


MIB_IPSTATS Struct

The MIB_IPSTATS structure is the structure that is filled out by the ***GetIpStatistics*** function. Some of the structure members are very useful and some you might not even care about other than just being extra detailed with information for the operator.

© SANS Institute 2024

https://linktr.ee/offsecexam

GetAdapterAddresses API

GetAdapterAddresses()

Grabs the addresses tied to the adapters

Has ULONG return type

```
IPHLPAPI_DLL_LINKAGE
ULONG
GetAdaptersAddresses(
 _In_ ULONG Family,
 _In_ ULONG Flags,
 _In_ PVOID Reserved,
 _Inout_ PIP_ADAPTER_ADDRESSES
AdapterAddresses,
 _Inout_ PULONG SizePointer
);
```


GetAdapterAddresses API

The ***GetAdapterAddresses*** can be used when you need to find out what adapters have what IP address. The function is great because not only can it do IPv4, but it can do IPv6 as well. ***GetAdapterAddresses*** has several **in** parameters and a few **inout** ones indicating that the function will be attempting to write to those variables. Let us break down each one of these parameters to get a better understanding of what they are.

Family, of type ULONG, is used to indicate what address family the function should get. This can be AF_INET (6), or AF_INET6 (23). The other interesting part is that you can specify both families at the same time with AF_UNSPEC. AF_UNSPEC indicates that any adapter that has IPv4 or IPv6 enabled, should have its information returned to the caller.

Flags, of type ULONG, is used to indicate what type of addresses to get. There is a long list of types that can be used for this parameter, but a few options are the following: GAA_FLAG_INCLUDE_GATEWAYS, GAA_FLAG_INCLUDE_ALL_INTERFACES, and quite a few more.

Reserved, of type PVOID, is reserved so we do not really care about it.

AdapterAddresses, of type PIP_ADAPTER_ADDRESSES, is a pointer to some variable that will act as the buffer filled with a linked list of IP_ADAPTER_ADDRESSES structures. The structure is a massive structure, and it is worth browsing to the MSDN documentation to completely understand it.

SizePointer, of type PULONG, is a pointer to the variable that stores the size of the buffer.

© SANS Institute 2024

Upon success, the function will return ERROR_SUCCESS. Should the function ever fail it will return one of the following error codes:

- ERROR_ADDRESS_NOT_ASSOCIATED: An address has yet to be associated with the device.
- ERROR_BUFFER_OVERFLOW: The buffer size indicated is not large enough to hold the requested information.
- ERROR_INVALID_PARAMETER: SizePointer is NULL, Family was not a valid family option.
- ERROR_NOT_ENOUGH_MEMORY: Literally not enough memory to complete the function.
- ERROR_NO_DATA: No addresses found.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

GetNumberOfInterfaces API

GetNumberOfInterfaces

Grabs the number of interfaces

Has DWORD return type

```
IPHLPAPI_DLL_LINKAGE
DWORD
GetNumberOfInterfaces(
 _Out_ PDWORD pdwNumIf
);

// example

DWORD dwCount = 0;
GetNumberOfInterfaces(&dwCount);

// error check
```


GetNumberOfInterfaces API

If you wanted to run something quick and easy, the ***GetNumberOfInterfaces*** function would be it. The only parameter you need to worry about is an **out** parameter. The function will write to it the number of interfaces that have been discovered on the local machine. The *pdwNumIf* parameter is of type PDWORD. All interfaces should be enumerated by the function, including the system's loopback adapter. If you do not care about the loopback interface, then you do not need to use this function. Other functions like ***GetAdaptersInfo*** and ***GetInterfaceInfo*** will not return information about the loopback interface. Also, the number returned might be higher than what you might be expecting as it does not directly relate to physical NICs on the target. Logical interfaces will be included in the count of interfaces, so if you were expecting to see 2 and got back something like 18, that would be the reason why.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Discussed how to gather information about the network

Discussed how to gather NIC information about the target

Module Summary

In this module, we discussed the why and how of gathering information about a target's NIC configuration, as well as any other information we can gather about the network overall. The information presented in this module can be the foundations for creating tools like arp, ipconfig, netstat, etc.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What API will give you an IP address for a network adapter?

A GetAdapterAddresses()

B GetNumberOfInterfaces()

C GetIpStatistics()

Unit Review Questions
Q: What API will give you an IP address for a network adapter?

A: GetAdapterAddresses()

B: GetNumberOfInterfaces()

C: GetIpStatistics()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What API will give you an IP address for a network adapter?

A GetAdapterAddresses()

B GetNumberOfInterfaces()

C GetIpStatistics()

Unit Review Answers
Q: What API will give you an IP address for a network adapter?

A: GetAdapterAddresses()

B: GetNumberOfInterfaces()

C: GetIpStatistics()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What API includes logical interfaces in its results?

A GetAdapterAddresses()

B GetNumberOfInterfaces()

C GetIpStatistics()

Unit Review Questions
Q: What API includes logical interfaces in its results?

A: GetAdapterAddresses()

B: GetNumberOfInterfaces()

C: GetIpStatistics()

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What API includes logical interfaces in its results?

A GetAdapterAddresses()

B GetNumberOfInterfaces()

C GetIpStatistics()

Unit Review Questions
Q: What API includes logical interfaces in its results?

A: GetAdapterAddresses()

B: GetNumberOfInterfaces()

C: GetIpStatistics()

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
Lab 2.1: OS Info
Service Packs/Hotfixes/Patches
Process Enumeration
Lab 2.2: ProcEnum
Lab 2.3 CreateToolhelp
Lab 2.4 WTSEnum
Installed Software
Directory Walks
Lab 2.5: FileFinder
User Information
Services and Tasks
Network Information
Registry Information
Bootcamp

This module will discuss how to enumerate the Windows Registry to find critical information about the system.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Gather registry information

Objectives
The objectives for this module are to understand what information can be found in the registry.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Registry Information



Troves of information

The Windows Registry contains troves of information that can arguably be deemed critical to your survey tool. The registry is so important that even the system itself relies on information found in the registry.

Registry Information

The registry was discussed in tremendous detail during Section 1, along with the APIs needed to enumerate practically everything in it. It is being included here during this day as a brief reminder that it should not be forgotten about when conducting your survey. The registry is an excellent source for collecting information from the target. Some sections you might not be able to query unless you have Administrator privileges or higher, but even still, you can collect useful information as a basic user.

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (1)



The registry API family provides most functionality for registry interaction.

The Registry is a collection of five hives where each one exposes information, some critical to the functionality of the OS. The hives have keys, which then have subkeys with values that applications or services might need to query.

It is has become the go-to location for developers for storing application information.

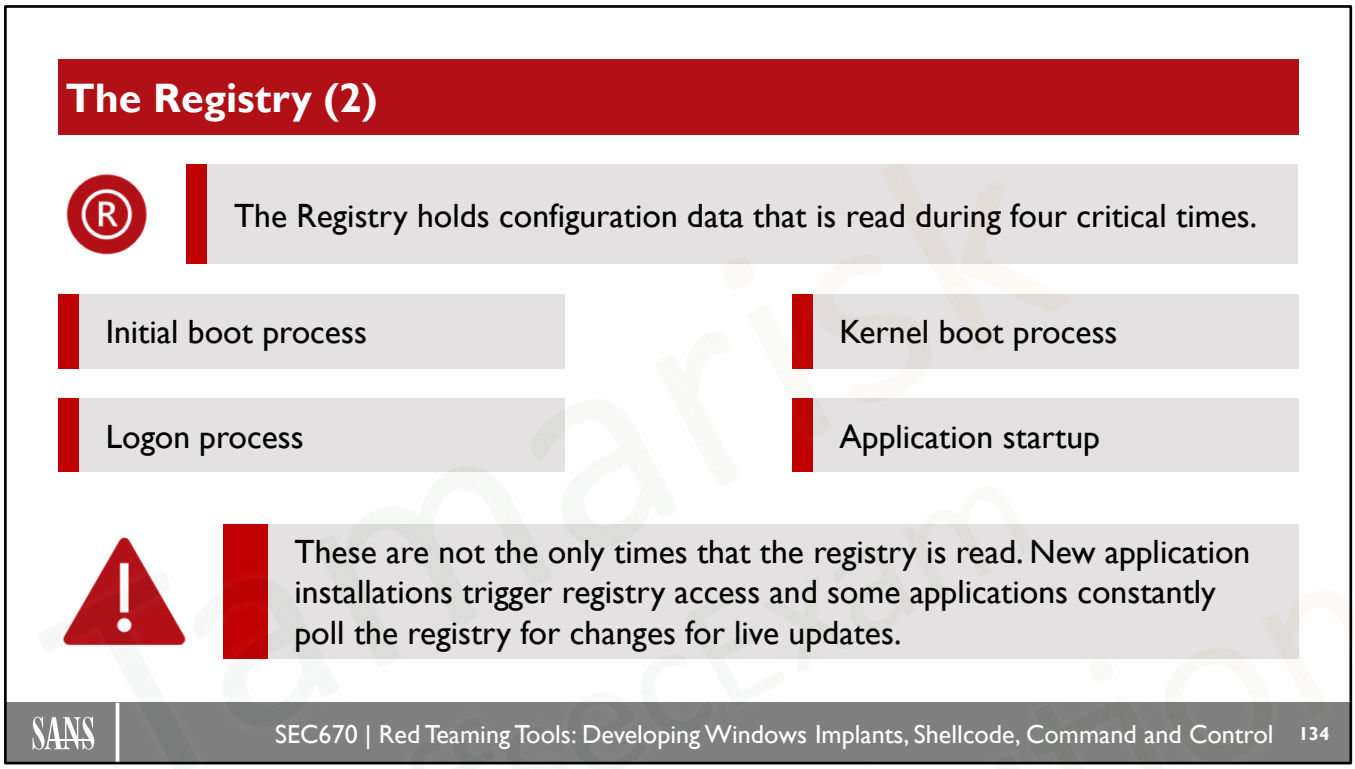
The Registry (1)

If you are coming from a pure Linux world, the Registry could be quite foreign to you. The Registry is simply a collection of five hives with each one exposing various pieces of information. In all reality, there are only two true hives, **HKEY_USERS** and **HKEY_LOCAL_MACHINE**. If you fire up regedit.exe you would see the five hives, but the other three hives are simply a combination of some of the data that can be found in the two main hives. The hives have keys and subkeys that hold settings and configurations that can be specific to users or to the machine. For 64-bit Windows, there will exist a 32-bit portion of the registry that will store information specific for 32-bit applications installed on the system. There will be a virtual redirection for 32-bit apps implemented by a mechanism called registry virtualization. The 64-bit apps see no such portion of the Registry as there is no need for virtual redirection.

The Registry used to be the go-to location for developers to store information for the applications they would develop. Now, Microsoft would like it if you left the Registry well enough alone so that processes internal to Windows are the only ones interacting with it. There are other ways to store information for your application, such as creating configuration files like INI, YAML, JSON, DAT, XML, etc. For this class and for our purposes, we will not be listening to that recommendation.

© SANS Institute 2024

https://linktr.ee/offsecexam



The Registry (2)
The configuration data held in the Registry is typically read from at four critical times.

1. The initial boot process
2. The kernel boot process
3. The logon process
4. Application startup

The Boot Configuration Database (BCD) is stored in a registry hive and the boot loader must read the configuration data to retrieve a list of boot device drivers to load into memory. Once that happens, the kernel can start its initialization process. The kernel will load the appropriate device drivers and determine how some managers like the process manager and memory manager configure themselves. All of this is a way to fine tune how the computer will behave.

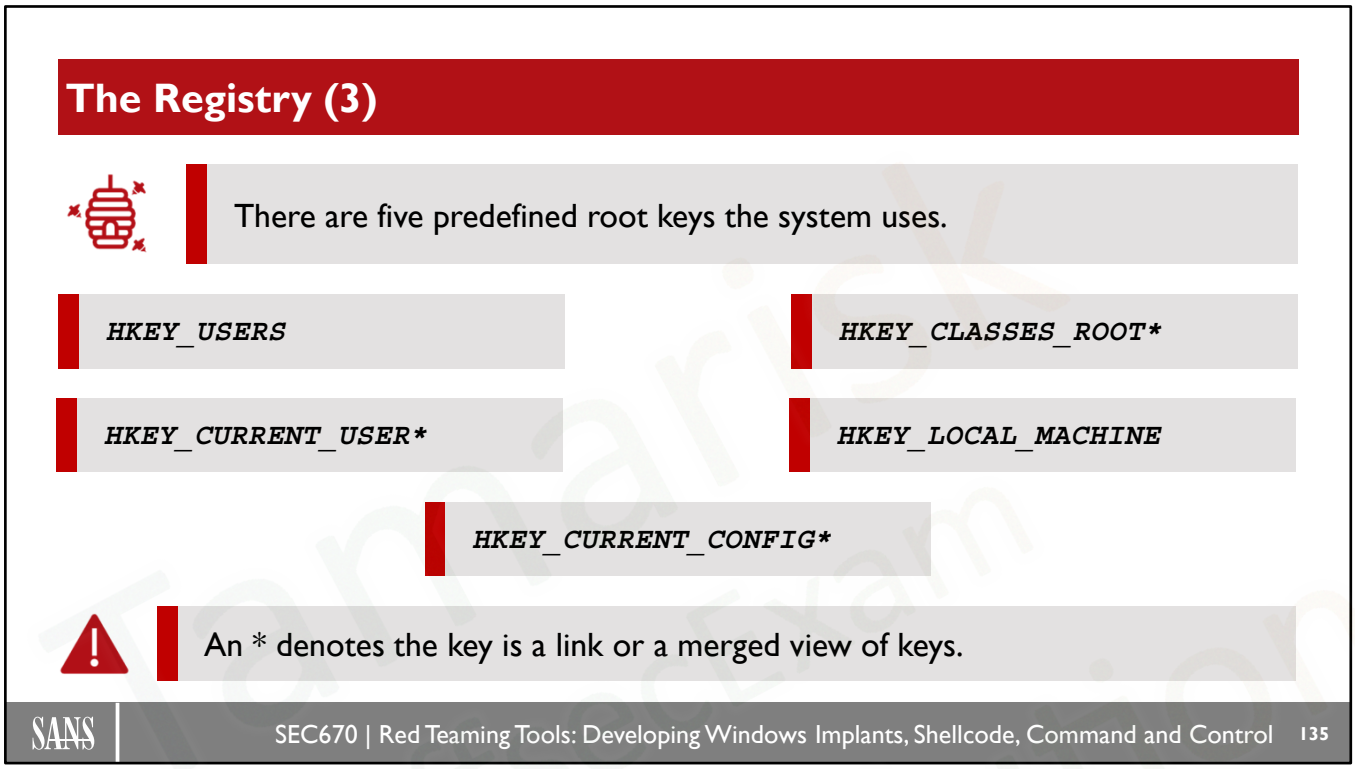
The logon process starts the per-user preferences. Explorer is one of many Windows components that must read the per-user profiles to set up various items like desktop wallpaper, screen saver, where desktop icons are placed, and what applications must be started. This is how each user on a local system can have different setups/preferences for their environment.

When the applications are being loaded by the system loader, they will read some systemwide settings as well as per-user preferences, like how the window layout is for an application like Word. If the user has a custom toolbar layout for an application, then that needs to be read and prepared when the application starts. Also, the most recently accessed documents will be retrieved and available to view.

New application installs are another moment when the registry is being read. The application will store various information in certain keys that are specific to it. Some of that data might be easily readable, but some might be obfuscated. Many AV products do this in an attempt to conceal some of their internal mechanisms, like how they communicate with their cloud engine when a suspicious file needs to be analyzed. Also, some applications can be overly aggressive with their registry access by polling for live changes made to its configuration data. This is not a best practice as idle systems should not have a lot of registry activity.

© SANS Institute 2024

https://linktr.ee/offsecexam



The Registry (3)
You might have noticed that each root key starts with an H. This is because the root key names are Windows handles (H) to keys (KEY); hence the name HKEY. The key names on the slide that are annotated with an asterisk "*" are links to other keys or a combination of two keys to provide a unique view of information. Here is a brief breakdown of the five root keys. For more detailed information, check out the *Windows Internals* books in addition to MSDN online documentation.

HKEY_USERS: Each time a new user logs in to the system, a new hive, called the user profile hive, will be created. Here, information about all system accounts is stored.

HKEY_CURRENT_USER: Data tied to the user who is currently logged on. This is a subkey under HKEY_USERS.

HKEY_CLASSES_ROOT: File association and Component Object Model (COM) object registrations.

HKEY_LOCAL_MACHINE: Systemwide settings.

HKEY_CURRENT_CONFIG: The current hardware profile.

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (4)



Deep dive: HKEY_USER (HKU)

The HKU key will hold a subkey (HKCU) for each user profile on the local system. There is also a profile for the system that has its own subkey, *HKU\.Default*. Winlogon uses the system profile to determine various settings like the desktop background.

ProfilesDirectory

ProfileList

The Registry (4)

The HKU key holds a wealth of user information. In fact, it holds user information for every user that has logged on to the local machine, as long as the user does not have a roaming profile with Active Directory or similar. Each user will have their own specific subkey—even the system has its own subkey located at *HKU\.Default*. The system profile is used by the Winlogon process to read information like the desktop background to show to the user when logging into their account. The user profiles are stored in a location determined by the registry value held in *HKLM\Software\Microsoft\Windows NT\CurrentVersion\ProfileList\ProfilesDirectory*. If the value has not been modified, the default is *%SystemDrive%\Users*.

If you wanted to develop some kind of user enumeration tool, then a good place to enumerate a list of profiles that exist on the system would be the ProfileList key. From there, the subkey name will be the Security Identifier, or SID, of the account that is tied to it.

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (5)



Deep dive: HKEY_CURRENT_USER (HKCU)*

This root key holds configuration information for the locally logged-on user regarding software configuration information and user preferences. They key points to the user profile, which is located at *\Users\<username>\Ntuser.dat*.

- Created for each new login
- Subkey under HKU
- Subkeys: console, software, control panel, identities, printers, keyboard layout, etc.

The Registry (5)

The HKCU key exists for each locally logged-on user. Each time a new user logs on, this key is created as a subkey under HKU. User-specific information is stored in this key located at \Users\<username>\Ntuser.dat. In addition, software configuration information is also stored in this key since different users might have different preferences for how their applications are laid out. In addition to this, different users could also have different user preferences that must be stored, and this is the perfect location to store that information. Also, any service processes that kick off under the context of a specific user will trigger their specific HKCU to be loaded.

There are several subkeys: AppEvents for sounds/events, Console for command window settings, Control Panel for screen saver information, Software for user-specific software preferences, Printers for printer connection settings, Keyboard layout for what country region layout should be (US or UK), or Identities for Windows Mail account information. This is not all the subkeys found under HKCU and to list them all, you can write an enumeration tool to gather that information.

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (6)



Deep dive: HKEY_CLASSES_ROOT (HKCR)*

This root key holds three types of information: file extension associations, COM class registrations, and virtualized registry root for the UAC. Every registered file extension will have its own key that is typically the REG_SZ value type. Sometimes they simply point to another key that holds the needed information.

HKCU\SOFTWARE\Classes

HKLM\SOFTWARE\Classes

The combination of the above Classes keys make this root key.

The Registry (6)

This root key is not a genuine root key like HKU or HKLM, nor is it a link or subkey of another key. It is the combination of two Classes keys: HKCU\SOFTWARE\Classes and HKLM\SOFTWARE\Classes. It might not make sense to have separate locations for user and system registration, but users having roaming profiles that are configured differently creates the need for that separation.

This key also holds information regarding file extensions and what applications they are associated with. Each registered file extension will have its own key that is simply a link to another location. Of course, the location and layout of the Registry greatly depends on what version of Windows you are running.

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (7)



Deep dive: HKEY_LOCAL_MACHINE (HKLM)

This root key holds vital information for the system. Some of the critical information like how the system boots is stored here. Other information is stored here, like systemwide software configurations, installed components, user passwords, and boot entries to name a few.

BCD: boot entries

SAM: account passwords

The Registry (7)
The HKLM key is the root key that holds all systemwide configuration subkeys.

- BCD00000000
- COMPONENTS
- HARDWARE
- SAM
- SECURITY
- SOFTWARE
- SYSTEM

BCD00000000 subkey is the Boot Configuration Data (BCD) information that is loaded as a registry hive. BCD is the replacements for the boot.ini file and it adds greater capabilities. There can be any number of entries in the BCD, like one for the Windows installation itself. The entries can be found in the subkey Objects.

COMPONENETS subkey stores information regarding the Component Based Servicing (CBS) stack. The Windows installation has various file and resources that make up the installation image. Depending on how many components are installed, this subkey can become very large and so it is loaded and unloaded dynamically, as necessary.

HARDWARE subkey holds information about the hardware in the system. This is read when the Device Manager user application is started.

SAM subkey holds user passwords, any group definitions, and associated domains. Because of the sensitive nature of the information stored in this subkey, the security descriptor does not even allow the Administrator account to access it.

SECURITY subkey holds policies that are systemwide; in addition, User-rights are also stored in this subkey.

© SANS Institute 2024

https://linktr.ee/offsecexam

SOFTWARE subkey holds systemwide configuration information that is not critical for booting the computer. Any third-party application installed on the system will also store their settings here.

SYSTEM subkey holds systemwide configuration information that is critical for booting the computer. Changing information in this subkey can render your system useless and for such occasions, Windows stores a last known good control set under this subkey. There, a lightweight copy of the subkey, called a maintenance copy, lets admins choose a working control set they know was good.

Tamarisk
OffsecExam
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (8)



Deep dive: HKEY_CURRENT_CONFIG (HKCC)*

This root key is nothing more than a link to the hardware profile that is stored in HKLM: *HKLM\SYSTEM\CurrentControlSet\Hardware Profiles\Current*. It's not prevalent today but Windows keeps it around in the name of backwards compatibility.

The Registry (8)

The HKEY_CURRENT_CONFIG, or HKCC for short, root key is one of the three links to other root keys. HKCC is formally linked to the HKLM root key and since this linked key points to whatever the current hardware profile is, the root key path is: HKLM\SYSTEM\CurrentControlSet\Hardware Profiles\Current. Windows does not support hardware profiles anymore, but the key is still created all in the name of backwards combability with legacy applications that depend on it being there.

© SANS Institute 2024

https://linktr.ee/offsecexam

The Registry (9)



Deep dive: HKEY_PERFORMANCE_DATA (HKPD)

This root key is unique because it cannot be accessed directly via the Registry Editor. It must be accessed programmatically via the Registry APIs. In it you would find performance counters either from system components or server applications.

RegQueryValueEx

Technically not stored here

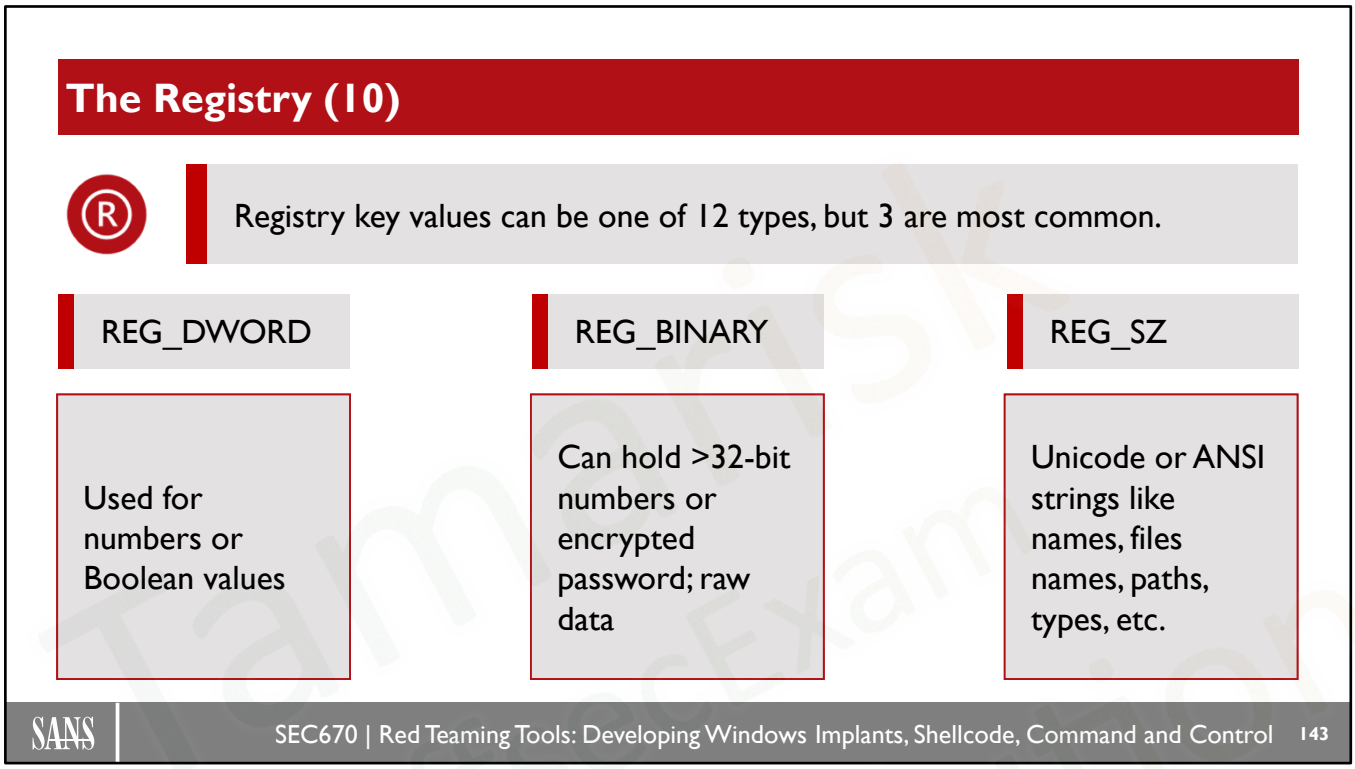
The Registry (9)

One might think that every registry key would be accessible for viewing via a tool like regedit.exe. Well, the HKEY_PERFORMANCE_DATA key is not one of those keys that regedit.exe can view. If you want to poll performance specific data, then you must do so using the appropriate Registry APIs like ***RegQueryValueEx***.

Another interesting tidbit about this key is that the performance counter information is not actually created/stored here but rather it comes from a provider. These performance data providers actually push the values to the key. One could, as another option for querying this information, use the Performance Data Helper API functions that are provided by the Pdh.dll, which would be the recommended choice to use.

© SANS Institute 2024

https://linktr.ee/offsecexam



The Registry (10)
A registry key's value can hold several different types, 12 types to be exact. Despite the number of value types, there are three that you are more likely to come across.

1. REG_DWORD
2. REG_BINARY
3. REG_SZ

If you remember from the Windows data types section earlier in this section, then the DWORD type should be familiar to you. REG_DWORD can hold 32-bit numbers or Boolean values to represent on/off or enabled/disabled flags.

REG_BINARY, as the name suggests, store raw binary information like encrypted passwords. It can also be used to store numbers that are larger than 32-bits.

The last of the common three types is REG_SZ. This is used to hold a string that has a null terminator. Perhaps the SZ means String Zero indicating String\0.

© SANS Institute 2024

https://linktr.ee/offsecexam

Registry Keys and Values (1)



RegOpenKeyEx

Return value is LSTATUS

Used to open a handle to a Registry key

```
//declared in Winreg.h

LSTATUS RegOpenKeyExW(
  _In_     HKEY  hKey,
  _In_opt_ LPWCSTR lpSubkey,
  _In_opt_ DWORD  ulOptions,
  _In_     REGSAM samDesired,
  _Out_    PHKEY  phkResult
);
```


Registry Keys and Values (1)
First and foremost, look at the function's return type: LSTATUS is a 32-bit signed integer, and any errors will be returned. The error code can be looked up using ***FormatMessage*** function passing in the **FORMAT_MESSAGE_FROM_SYSTEM** flag for a description of the error. No need to call ***GetLastError*** with NSTATUS return values. Let's break down the parameters in detail.

hKey requires an open handle to an open registry key. If there is no handle opened yet, then one of the predefined keys can be passed: HKEY_CLASSES_ROOT, HKEY_CURRENT_CONFIG, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS.

lpSubkey needs a pointer to the name of the registry subkey to open. Case sensitivity is ignored here. Because the SAL annotation contains the _opt_ suffix, the pointer is optional; nullptr. It is important to note that NULL can only be passed here if hKey is passed one of the predefined keys.

ulOptions should just pass NULL here. If not null, then certain options are to be applied when open the key. The only option noted by MSDN documentation is REG_OPTION_OPEN_LINK, meaning the key is simply a symbolic link.

samDesired is a mask indicating the desired access to the key.

phkResult, as annotated by the annotation _Out_, should be a pointer to a variable that will hold the handle to the opened key.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: RegOpenKeyEx

```
INT main(VOID)
{
    HKEY hHKCU = HKEY();
    RegOpenKeyExW(HKEY_CURRENT_USER, L"Console", NULL, KEY_READ, &hHKCU);
}
```


Example: RegOpenKeyEx
The example here initializes a variable of type HKEY that will be used to store the handle that the function gives upon success. Just like the function declares, the last parameter must be the address of the variable. Afterall, it is an _Out_ parameter so the user is responsible for making that available for the function to use.

The function doesn't "return" a handle because this function returns an LSTATUS value that could be used to determine why it may have failed.

Error handling is left out for brevity.

© SANS Institute 2024

https://linktr.ee/offsecexam

Registry Keys and Values (2)



RegQueryValueEx

Return value is LSTATUS

Used to read the type and data of a Registry key value

```
//declared in Winreg.h

LSTATUS RegQueryValueExW(
  _In_         HKEY    hKey,
  _In_opt_     LPCWSTR lpValueName,
  _Reserved_   LPDWORD lpReserved,
  _Out_opt_    LPDWORD lpType,
  _Out_        LPBYTE  lpData,
  _Inout_opt_ LPDWORD lpcbData
);
```


Registry Keys and Values (2)
Again, we have an LSTATUS return type, so the error code can be looked up using ***FormatMessage*** function passing in the **FORMAT_MESSAGE_FROM_SYSTEM** flag for a description of the error. Let's understand the function's parameters in detail.

hKey is a handle to a key, just like in the ***RegOpenKey*** function.

lpValueName is a pointer to the string holding the name of the registry value. Passing NULL or "" will make the function grab the type and data for the key's default or unnamed value. If the *lpValueName* cannot be found, or it does not exist, the function will return ERROR_FILE_NOT_FOUND.

lpReserved must be NULL since it is reserved for internal purposes only.

lpType is a pointer to a variable that will hold a code that indicates the type of data being stored in the mentioned value. MSDN documents the possible types that are acceptable to pass here, but some are REG_BINARY, REG_DWORD, and REG_SZ.

lpData is a pointer to a buffer that will hold the value's data. If there is no requirement for data, then simply pass NULL here. This buffer can be too small to accept the data and if that is the case, the function will return ERROR_MORE_DATA.

lpcbData is a pointer to a variable that indicates the buffer size in bytes pointed to by *lpData*. Upon return, the variable will hold the size of the copied data. *lpcbData* is allowed to be NULL but only when *lpData* is also NULL.

© SANS Institute 2024

Example: RegQueryValueEx

```
INT main(VOID)
{
    HKEY hHKCU = HKEY();
    DWORD dwType = 0;
    DWORD dwSize = 0;

    RegOpenKeyExW(HKEY_CURRENT_USER, L"Console", NULL, KEY_READ, &dwSize);
    auto Value = std::make_unique<BYTE[]>(dwSize);
    ReqQueryValueExW(hHKCU, L"FaceName", NULL, &dwType, Value.get(), &dwSize);

    // do something with the queried value ...

    RegCloseKey(hHKCU);

    return ERROR_SUCCESS;
}
```


Example: RegQueryValueEx

The example here builds from the first snippet on the previous slide showing ***RegOpenKeyExW*** where the handle to the key was obtained. That key handle is needed for future calls that are specific to that key. ***RegQueryValueExW*** accepts that key handle as its first parameter. You might question the need for consecutive calls to ***RegQueryValueExW*** and the reason for this is because at first, we do not know the size of the data being held in the FaceName key. To determine the proper size needed, the function allows us to pass in NULL for the lpData parameter and a non-NULL value for the lpcbData parameter. When the function does its parameter checking, it will see this and store the size of the data, in bytes, in the variable pointed to by lpcbData. In our case that would be ***size***. Before we get to the second call, we take the size of the data and use that to create a byte buffer using make_unique. Here, I am calling that ***value*** since it will hold the key's value when done.

Fully equipped with the information needed, the second call can be made. A buffer can now be passed into the lpData parameter, and the size of that buffer can be passed into the lpcbData parameter.

Error handling is left out for brevity.

© SANS Institute 2024

https://linktr.ee/offsecexam

Walking the Registry (1)



RegEnumKeyEx

Return value is LSTATUS

Used to enumerate subkeys under a specific key

```
//declared in Winreg.h

LSTATUS RegEnumKeyEx(
  _In_         HKEY    hKey,
  _In_         DWORD   dwIndex,
  _Out_        LPTSTR  lpName,
  _Inout_      LPDWORD lpcchName,
  _Reserved_   LPDWORD lpReserved,
  _Out_        LPTSTR  lpClass,
  _Inout_opt_  LPDWORD lpcchClass,
  _Out_opt_    PFILETIME lpftLastWriteTime
);
```


Walking the Registry (1)
Again, we have an LSTATUS return type, so the error code can be looked up using ***FormatMessage*** function passing in the **FORMAT_MESSAGE_FROM_SYSTEM** flag for a description of the error. This function is great for enumerating a particular key's subkeys. Let's break down the function's parameters.

hKey is a handle to a key, just like in the RegOpenKey function, but this time the handle must have been open with the KEY_ENUMERATE_SUB_KEYS access mask or you will be denied access when you make this call.

dwIndex will be used to keep track of loop iterations. Starting with a value of 0, we can increment this value each iteration until we get a return value of ERROR_NO_MORE_ITEMS.

lpName is just the simple key's name and not the root key name. If the buffer for this parameter is not large enough to hold the key's name, then the function will fail with ERROR_MORE_DATA. If this happens, don't expect lpName to have anything in it, it should be NULL.

lpcchName is the size of the lpName and should be set to the maximum number of characters that a buffer can hold. This includes the NULL byte. The function will modify this value to reflect the correct number of characters written, but not include the NULL byte. FYI, the max key name length is 255.

That is the bulk of the function. If a developer wished, they could grab the class name, hardly ever used or needed, along with the last time the key was modified. The latter part could be interesting if you are looking for something in particular.

© SANS Institute 2024
https://linktr.ee/offsecexam

Walking the Registry (2)



RegEnumValue

Return value is LSTATUS

Used to enumerate a key's value

```
//declared in Winreg.h

WINADVAPI
LSTATUS
APIENTRY
RegEnumValueW(
  _In_  HKEY  hKey,
  _In_  DWORD dwIndex,
  _Out_writes_to_opt_(*lpcchValueName,*lpcchValueName +
1) LPWSTR lpValueName,
  _Inout_   LPDWORD lpcchValueName,
  _Reserved_ LPDWORD lpReserved,
  _Out_opt_  LPDWORD lpType,
  _Out_writes_bytes_to_opt_(*lpcbData, *lpcbData)
__out_data_source(REGISTRY) LPBYTE lpData,
  _Inout_opt_ LPDWORD lpcbData
  );
```


Walking the Registry (2)
Again, we have an LSTATUS return type, so the error code can be looked up using ***FormatMessage*** function passing in the **FORMAT_MESSAGE_FROM_SYSTEM** flag for a description of the error.

Another observation you may have had is the addition of two SAL annotations that are describing the function. WINADVAPI is defined as DECLSPEC_IMPORT, which is defined as __declspec(dllimport). The APIENTRY type is one we have seen before earlier today. To refresh your memory, it's defined as WINAPI, which is defined as __stdcall. Now let's understand a few of the function's parameters in detail.

hKey is one we've talked about enough.

dwIndex is new and is used when doing a for loop while iterating over the values. When the function returns **ERROR_NO_MORE_ITEMS**, you can break out of the loop and continue.

Just like ***RegEnumKeyEx***, the buffers that you make for holding the ValueName and Data need to be large enough. If you want, you could initialize them to the values that are returned from the ***RegQueryInfoKey*** function described on the next slide.

lpType doesn't need to be described again. lpData will hold the values and lpcbData is the size of it—again, just like the other functions mentioned already.

If you run into any issues using this function, like getting **ERROR_ACCESS_DENIED**, then you should check the permissions you requested when you opened the handle to the root key. You might need **KEY_QUERY_VALUE** to have success.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: RegEnumValue

```
for (DWORD index = 0; ; index++)
{
    DWORD cbSize = cbMaxValueLen;
    DWORD cchName = cbMaxValueName + 1;

    auto status = RegEnumValueW(
        hKey,               // hKey
        index,              // dwIndex
        keyName.get(),      // lpValueName
        &cchName,           // lpcchValueName
        NULL,               // lpReserved
        &type,              // lpType
        keyValue.get(),     // lpData
        &cbSize             // lpcbData
    );

    if (status == ERROR_NO_MORE_ITEMS)
        break;
```


Example: RegEnumValue

The example here shows the function being used in a loop with the *index* variable being used as the loop iterator value, or counter. The key handle is passed followed by the dwIndex value, which is an index of the value that is to be retrieved. It is a good idea to have this value be NULL on the first iteration. Incrementing the value is fine for subsequent calls. The name of the key will be stored in the *keyName* buffer for the lpValueName parameter.

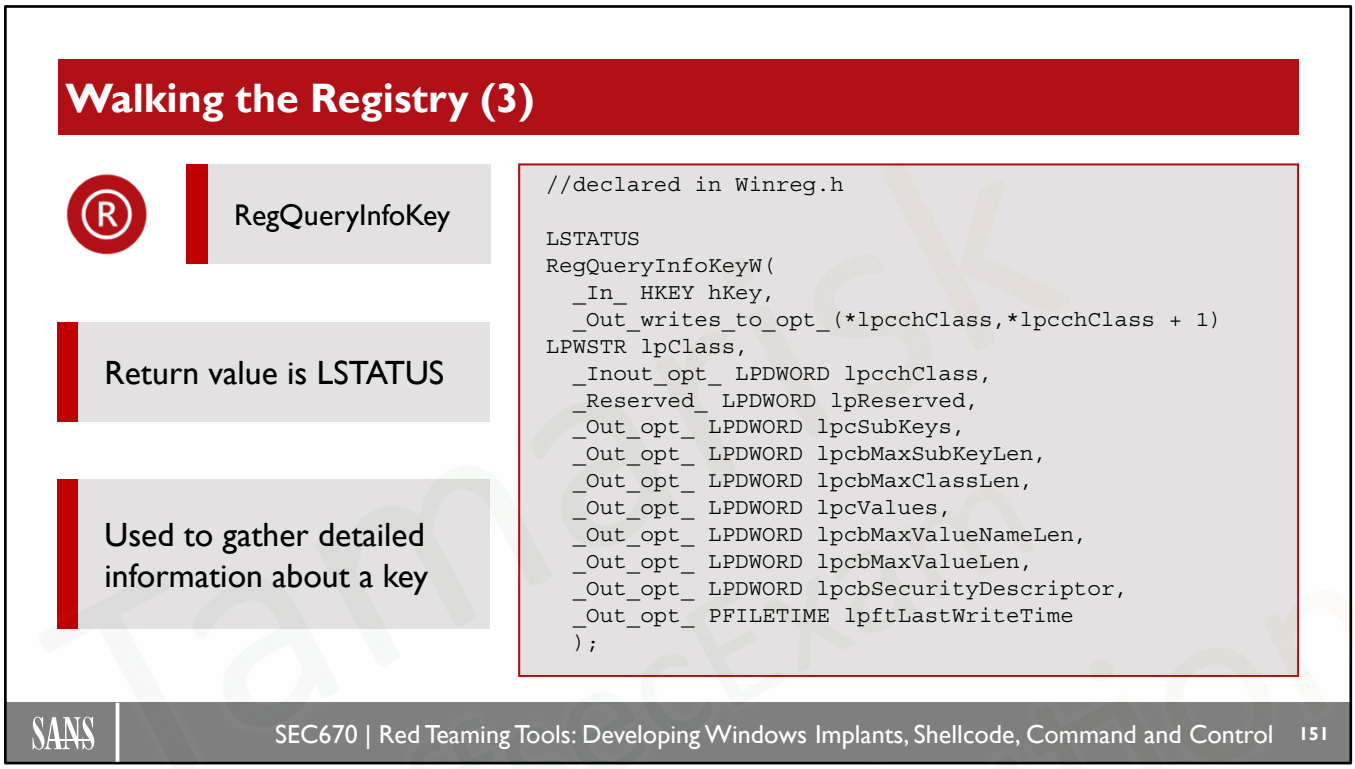
The lpcchValueName parameter, upon return, will hold the character count stored in the buffer. Note, the count does not take into account the NULL terminating character, so you need to do that in your program. lpReserved is NULL followed by the address of the variable to hold the type of data being stored. The lpData parameter was passed the address to the keyValue buffer that will be used to store the key's value. The lpcbData parameter is the size of the buffer but then after the function returns, will indicate the number of bytes written to the buffer.

The loop will continue until the function indicates there are no more items by returned the **ERROR_NO_MORE_ITEMS** error code.

The comments are provided for easy reference with the parameter names.

Error handling is left out for brevity.

© SANS Institute 2024
https://linktr.ee/offsecexam



Walking the Registry (3)

This RegQueryInfoKey API has many parameters to understand. It might look like a complex function, but it's not that difficult to use compared to some of the others we have discussed today. Looking at the data types for the parameters you can see that most of them are LPDWORD, which are simply DWORD*. Almost all of them are optional too, so that makes it nice to use. Depending on your needs, you could make the API call like this:

RegQueryInfoKeyW(hKey, NULL, NULL, NULL, &nKeys, NULL, NULL, &nValues, &nMaxValNameLen, &nMaxValSize, NULL, NULL)

That small example has seven NULL parameters, leaving just five parameters for you to prep for use. Now we can look at some of the parameters.

lpcSubKeys will hold the number of subkeys, if any, that are found to be under the queried key.

lpcValues will hold the number of—you guessed it—values that are found under the subkey. If you do not care about this, then just pass NULL here. It is optional after all.

lpcbMaxValueNameLen will hold the size of the key's longest value name, but it does not consider the NULL terminating byte, so your code will have to account for that one.

lpcbMaxValueLen will hold the size of the longest data component in bytes.

For failures, unlike previous functions, the function will return a system error code that is defined in WinError.h header file. You could also browse to the following URL for easy lookups: https://docs.microsoft.com/en-us/windows/win32/debug/system-error-codes.

© SANS Institute 2024

https://linktr.ee/offsecexam

Example: RegQueryInfoKey

```
DWORD retCode;
retCode = RegQueryInfoKeyW(
    hKey,                   //
    NULL, NULL, NULL,       //
    &cSubKeys,              //
    &cbMaxSubKey,           //
    &cbMaxClass,            //
    &cValues,               //
    &cbMaxValueName,        //
    &cbMaxValueLen,         //
    &cbSecurityDescriptor,  //
    &modified               //
);
```


Example: RegQueryInfoKey

The example here shows how ***RegQueryInfoKey*** could be called. A standard handle to a key is given followed by three NULL values: lpClass, lpcClass, and lpReserved. Next, the address of the *cSubKeys* variable is passed so the function can write the number of subkeys contained by that key. Next, the address of the *cbMaxSubKey* variable is passed so the function can write the size of the key with the longest name, minus the NULL terminating character of course. Next, the address of the *cbMaxClass* variable is passed to receive the size of the longest string for a subkey class. The address of the *cValues* variable will hold the count of values a key might have. The address of the *cbMaxValueName* variable will hold the size of the key's longest value name. The address of the *cbMaxValueLen* variable will hold the byte count of the size of the longest data component. The address of the *cbSecurityDescriptor* variable will hold the size of the key's security descriptor. The last one is the address of the ***modified*** variable that is a FILETIME structure to hold the last write time.

The comments indicating the parameter names are left out for you to fill them in later.

Error handling is left out for brevity.

© SANS Institute 2024

https://linktr.ee/offsecexam

Registry Watch Dogs (1)



You can be notified about changes in the Registry.

Perhaps it might be necessary for your code to be notified as soon as a change in the Registry happens. Maybe you want to know if an antivirus product was just installed after your implant was dropped. There could be several reasons you determine.

Don't poll too often

Choose your trigger

Registry Watch Dogs (1)

At times, it might serve your program well to be notified when certain changes happen in the Registry. As mentioned earlier, third party developers are being recommended to move away from storing items in the Registry, but it still happens. Malware is still using the Registry too, and your implants might as well as you develop them.

From an offensive perspective, you might want to watch certain existing keys for modification. Perhaps your own program creates a few keys and then sets up a watch dog to ensure keys aren't deleted. You also might want to check a certain key's value to see if your tool needs to delete itself or take some other action. It's up to you since you are the developer.

Taking a defensive and different angle, perhaps you drop your tool on a target that doesn't have any AV products just yet, but later the user installs one that creates a few registry keys. You might want to know this, so it doesn't alert the user to your implant. Seeing that a new key was created gives you the situational awareness you might need for your survival.

Typically, applications don't need to poll the Registry too often. The ***RegNotifyChangeKey*** is perfect for notifications. As mentioned earlier, an idle system shouldn't have any registry interaction. You can be too aggressive with polling for changes and there's just no need for that. Repetitive access to the same keys and or values could tip your hand to the user or Admin.

All of this is great and all, but one major drawback to this function is that it doesn't show you exactly what changed, only that a change has occurred. It is up to you to determine the finer details. Thankfully, there is alternative that will be discussed later in the course.

© SANS Institute 2024

https://linktr.ee/offsecexam

Registry Watch Dogs (2)



RegNotifyChangeKeyValue

Return value is LSTATUS

Used to be notified when specific changes happen

```
// defined in winreg.h

WINADVAPI
LSTATUS
APIENTRY
RegNotifyChangeKeyValue(
  _In_    HKEY hKey,
  _In_    BOOL bWatchSubtree,
  _In_    DWORD dwNotifyFilter,
  _In_opt_ HANDLE hEvent,
  _In_    BOOL fAsynchronous
  );
```


Registry Watch Dogs (2)
hKey is the key to be watched and the handle should have the REG_NOTIFY access mask when the handle is being obtained.

bWatchSubtree is a simple Boolean flag to indicate if just the one specified key should be watched (FALSE) or if the entire tree of keys under it need to be watched (TRUE).

dwNotifyFilter specifies the filter that triggers the notification to the caller. There are several to choose from, like watching for name changes. You can also combine flags to suit your needs using the pipe '|' or the OR operation. The filters are discussed in greater detail on the next slide.

hEvent is just a handle to an event. Depending on if the fAsynchronous flag is set, the function will return straight away, and the change is reported by the event being signaled. If it is not set, then the function will not return until some change takes place.

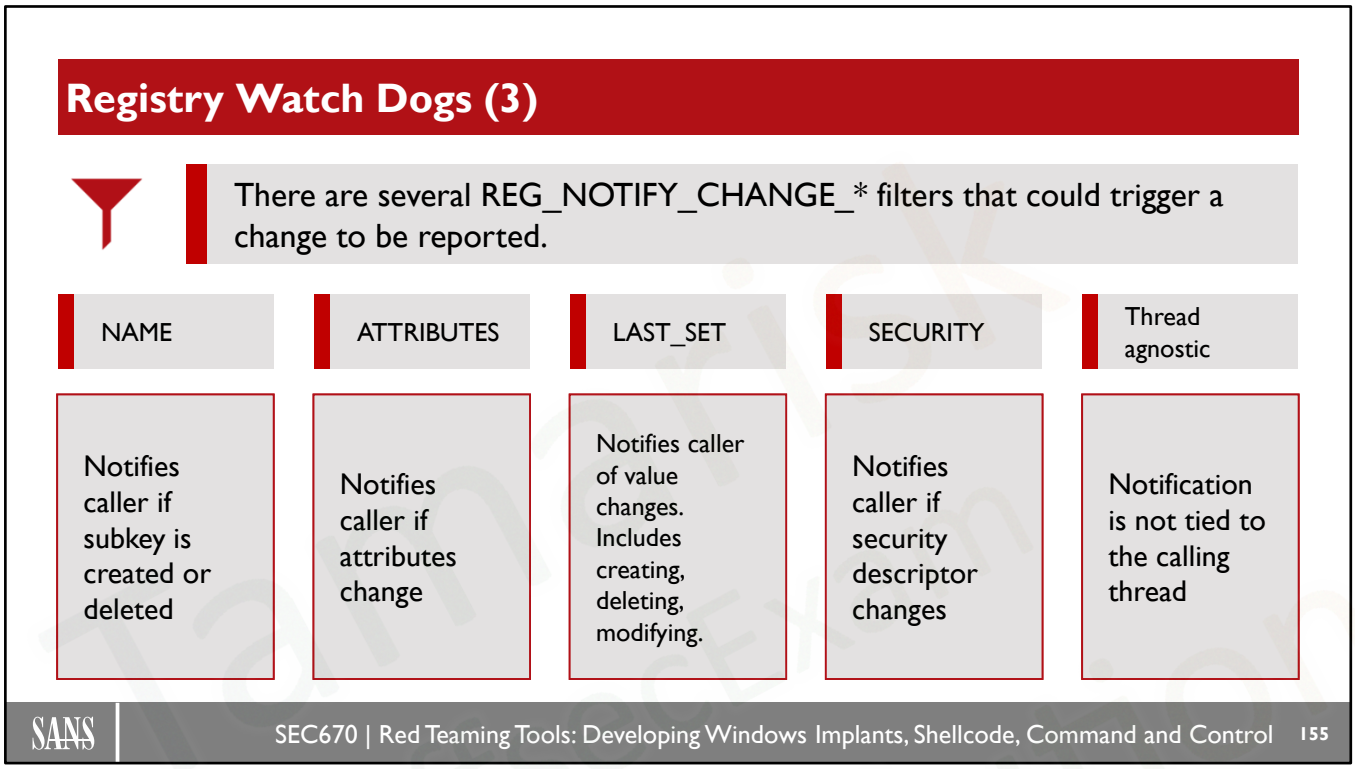
fAsynchronous is a flag to indicate when the function should return and signal an event. If it is set, TRUE, the function returns immediately and signals the specified event. If FALSE, the function waits until a change is made.

The function will return ERROR_SUCCESS upon success and will return a nonzero value upon failure. The code can be queried using ***FormatMessage*** if desired.

One last note about this function: it will only wait and watch for a single change. When that change happens, it has done its job, so if you want to keep being notified of changes then you must call the function again. Perhaps infinite loops in a new thread could serve well in your program.

© SANS Institute 2024

https://linktr.ee/offsecexam



Registry Watch Dogs (3)

REG_NOTIFY_CHANGE_NAME if specified, will alert the caller that a subkey has been added or deleted.

REG_NOTIFY_CHANGE_ATTRIBUTES filter will notify the caller if the key's attributes have changed. One of the attributes that could change is the security descriptor information.

REG_NOTIFY_CHANGE_LAST_SET filter will notify the caller if the value of the key has changed. Adding, deleting, and even modifying an existing key's value will trigger the alert.

REG_NOTIFY_CHANGE_SECURITY filter will notify the caller if the security descriptor of the key changes.

REG_NOTIFY_THREAD_AGNOSTIC removes the notification tie to the thread that called the function. This is great for when you want to spin up a new thread for the registration. Also, if that thread terminates, the registration won't die with it because the lifetime of the registration is not directly tied to the lifetime of the calling thread.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Discussed the registry and information found within it.

Module Summary

In this module, we discussed what the registry is, many of the keys, and some of the information that can be found within the registry.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 2

Gathering Operating System Information
- Lab 2.1: OS Info

Service Packs/Hotfixes/Patches

Process Enumeration
- Lab 2.2: ProcEnum
- Lab 2.3 CreateToolhelp
- Lab 2.4 WTSEnum

Installed Software

Directory Walks
- Lab 2.5: FileFinder

User Information

Services and Tasks

Network Information

Registry Information

Bootcamp

This is your time to go back and complete previous labs or move forward and complete the bootcamp challenges.

© SANS Institute 2024

https://linktr.ee/offsecexam

Bootcamp

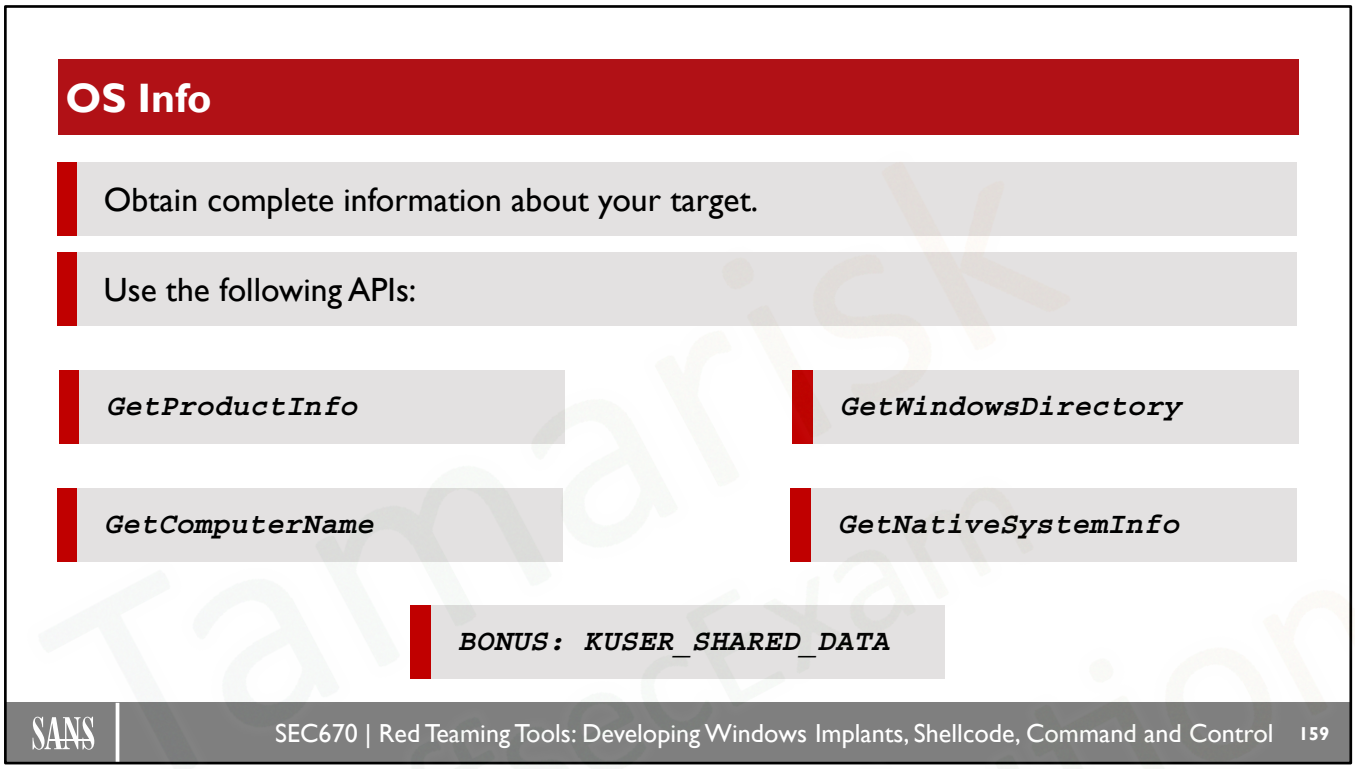
- OS Info
- Make your own ipconfig, arp, or netstat, and a custom shell
- Complete survey tool

Bootcamp
The bootcamp challenges will have varying degrees of difficulty. OS Info brings back something you already learned but also tosses in some new items to see how well you can look at MSDN documentation to learn how to implement new APIs you have not seen before.

The second challenge is to recreate one of the following utilities: ipconfig, arp, or netstat. If you have time, then you can complete all three.

The last challenge is to complete a thorough host survey tool (CustomShell) that would enumerate all of the information that was discussed today. In addition, combine what was learned during Section 1 and create a log file that stores the enumerated system information.

© SANS Institute 2024



OS Info

This bootcamp challenge is about leveraging a familiar API that you learned about earlier to gather some information about your target. The real challenge comes in with the introduction of several new APIs that can be called to help gather system information. You will have to teach yourself how to use these ones. As a bonus, if you finish the challenge and want to take on another challenge, get as much information as you can from the KUSER_SHARED_DATA struct that is available in almost every single user mode process. You can have all of the output print out to the terminal window to make it easier for testing and debugging. When it is all said and done, you can create a log file with all of the information in it.

Have fun!

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 2

Course Roadmap

- Windows Tool Development
- **Getting to Know Your Target**
- Operational Actions
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Lab 2.6: Ipconfig
Lab 2.7: Arp
Lab 2.8: Netstat
Lab 2.9: ShadowCraft

This is your time to go back and complete previous labs or move forward and complete the bootcamp challenges.

Tanarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.6: Ipconfig

- Create your own version of ipconfig
- Can add optional arguments
- Can make it fancy with colored output

Tamarisk OffsecExam Exam solution

Lab 2.6: Ipconfig
Please refer to the eWorkbook for the details of this bootcamp challenge.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.7: Arp

Create your own version of arp.

Implement arguments like –a and –n.

Lab 2.7: Arp
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.8: Netstat

Create your own version of netstat.

Implement arguments like –a, –n, and –t.

Lab 2.8: Netstat
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 2.9: ShadowCraft

- Create a basic shell.
- Implement features covered in this section.
- Implement thorough error checking.

Lab 2.9: ShadowCraft
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024
https://linktr.ee/offsecexam

670.3

Operational Actions

Tamarisk OffsecExam Exam solution

© SANS Institute 2024
© 2024 Jonathan Reiter. All rights reserved to Jonathan Reiter and/or SANS Institute.
https://linktr.ee/offsecexam

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE (“SANS INSTITUTE”) FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this CLA. Courseware means all printed materials, including course books and lab workbooks, slides or notes, as well as any digital or other media, audio and video recordings, virtual machines, software, technology, or data sets distributed by SANS Institute to User for use in the SANS Institute course associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and User and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium, whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. User may not sell, rent, lease, trade, share, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute. Additionally, User may not upload, submit, or otherwise transmit Courseware to any artificial intelligence system, platform, or service for any purpose, regardless of whether the intended use is commercial, educational, or personal, without the express written consent of SANS Institute. User agrees that the failure to abide by this provision would cause irreparable harm to SANS Institute that is impossible to quantify. User therefore agrees to a base liquidated damages amount of $5000.00 USD per item of Courseware infringed upon or fraction thereof. In addition, the base liquidated damages amount shall be doubled for any Courseware less than a year old as a reasonable estimation of the anticipated or actual harm caused by User’s breach of the CLA. Both parties acknowledge and agree that the stipulated amount of liquidated damages is not intended as a penalty, but as a reasonable estimate of damages suffered by SANS Institute due to User’s breach of the CLA.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. A written amendment or addendum to this CLA that is executed by SANS Institute and User may accompany this Courseware.

SANS Institute may suspend and/or terminate User’s access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User, (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

The Apple® logo and any names of Apple products displayed or discussed in this book are registered trademarks of Apple, Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

Governing Law: This CLA shall be governed by the laws of the State of Maryland, USA.

Courseware licensed to User under this CLA may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department (“OFAC”), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce (“BIS”). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulation. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

All reference links are operational in the browser-based delivery of the electronic workbook.

© SANS Institute 2024

https://linktr.ee/offsecexam

SEC670.3 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control

SANS

Operational Actions

© 2024 Jonathan Reiter | All Rights Reserved | Version J01_03

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control: 670.3
Welcome to Section 3 of SEC670. This section is all about what you could do locally against the target. Let's get started.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (1)	Page
PE Format	4
Lab 3.1: GetFunctionAddress	26
Threads	33
Injections	49
Lab 3.2: ClassicDLLInjection	55
Lab 3.3: APC Inception	60
Lab 3.4: ThreadHijacker	66
Escalations	81
Lab 3.5: TokenThief	100
Bootcamp	138
Lab 3.6: So, You Think You Can Type	140

This page intentionally left blank.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (2)	Page
Lab 3.7: UACBypass-Research	141
Lab 3.8: ShadowCraft	142

Tamarisk OffSecExam Exam solution

This page intentionally left blank.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- **Operational Actions**
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 3

PE Format
Lab 3.1: GetFunctionAddress
Threads
Injections
Lab 3.2: ClassicDLLInjection
Lab 3.3: APCInjection
Lab 3.4: ThreadHijacker
Escalations
Lab 3.5: TokenThief
Bootcamp
Lab 3.6: So, You Think You Can Type
Lab 3.7: UACBypass-Research
Lab 3.8: ShadowCraft

PE Format
In this module, we will discuss in detail the format of PE files. Knowing the structure of various headers is vital for carrying out a few injection methods, as well as retrieving other information to leverage like walking exported functions.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Tear into the PE format
- Use winnt.h as a guide
- Build a lightweight PE parser

Objectives
The objectives for this module are to tear into the anatomy of a PE file using the winnt header file as a guide. At the end, you will be armed with enough knowledge that you should be able to build a lightweight PE file parser.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

PE Format

Portable executable: the format is architecture agnostic

PE files, or executables, are complex in design due to the many structures that are involved under the hood.

Executable images (PE)

Object files (COFF)

PE Format

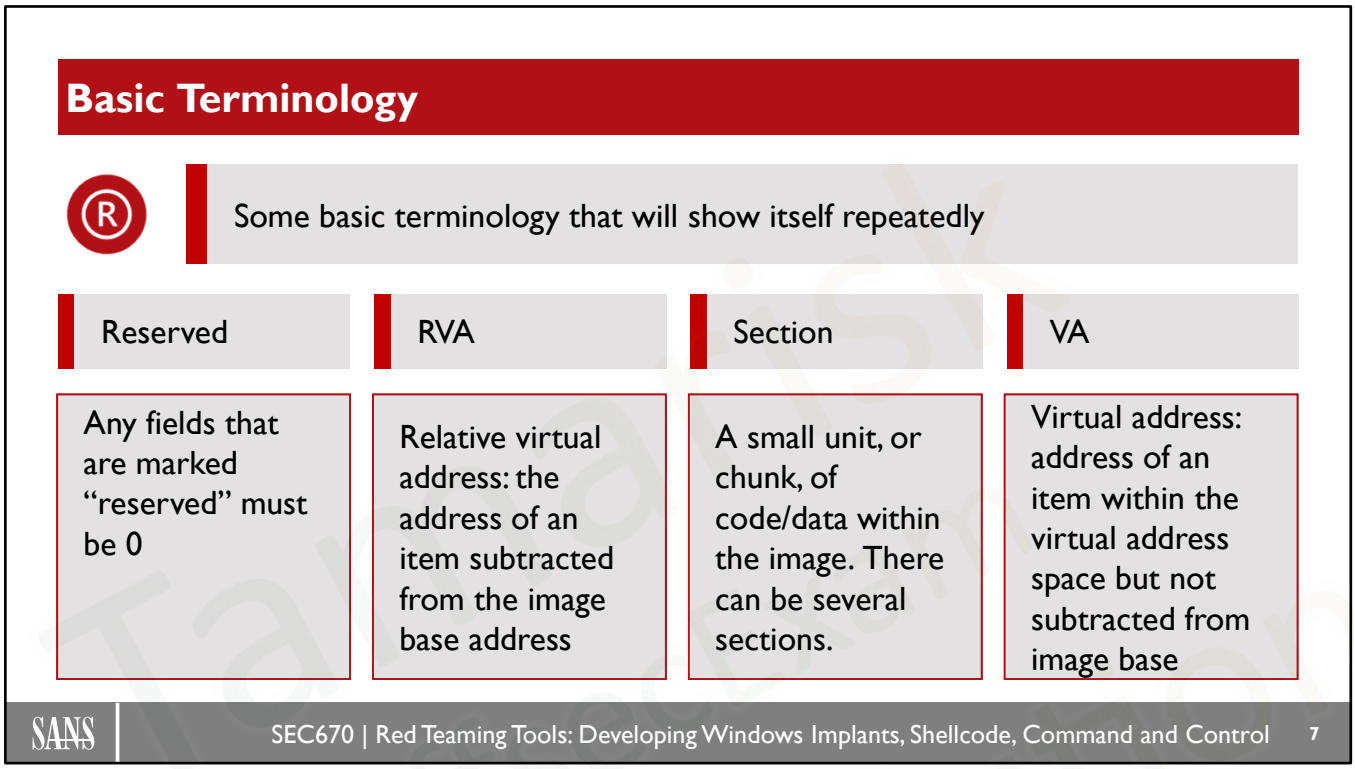
The internal structure of a PE file is very complex, even if you are using the MSDN documentation as a guide to go through it. There are several structures that you will need to understand and most, if not all of them are defined in the winnt.h header file. There is some very important information that is held in some of the struct fields, like what type of application it is (GUI or console), what architecture is it supposed to execute on, what functions are being imported, if any, and what functions are being exported, if any. When an image is in the process of being loaded into memory by the system loader, the system loader will parse every inch of the PE format to ensure the image is properly loaded into memory.

According to the MSDN page for PE Format, it will take you 127 minutes to read everything they have documented. To truly grasp everything documented and implement a highly detailed parser, it could perhaps take a week if it were your full-time job. For this reason, we will only be creating a lightweight parser for the lab at the end of this module.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



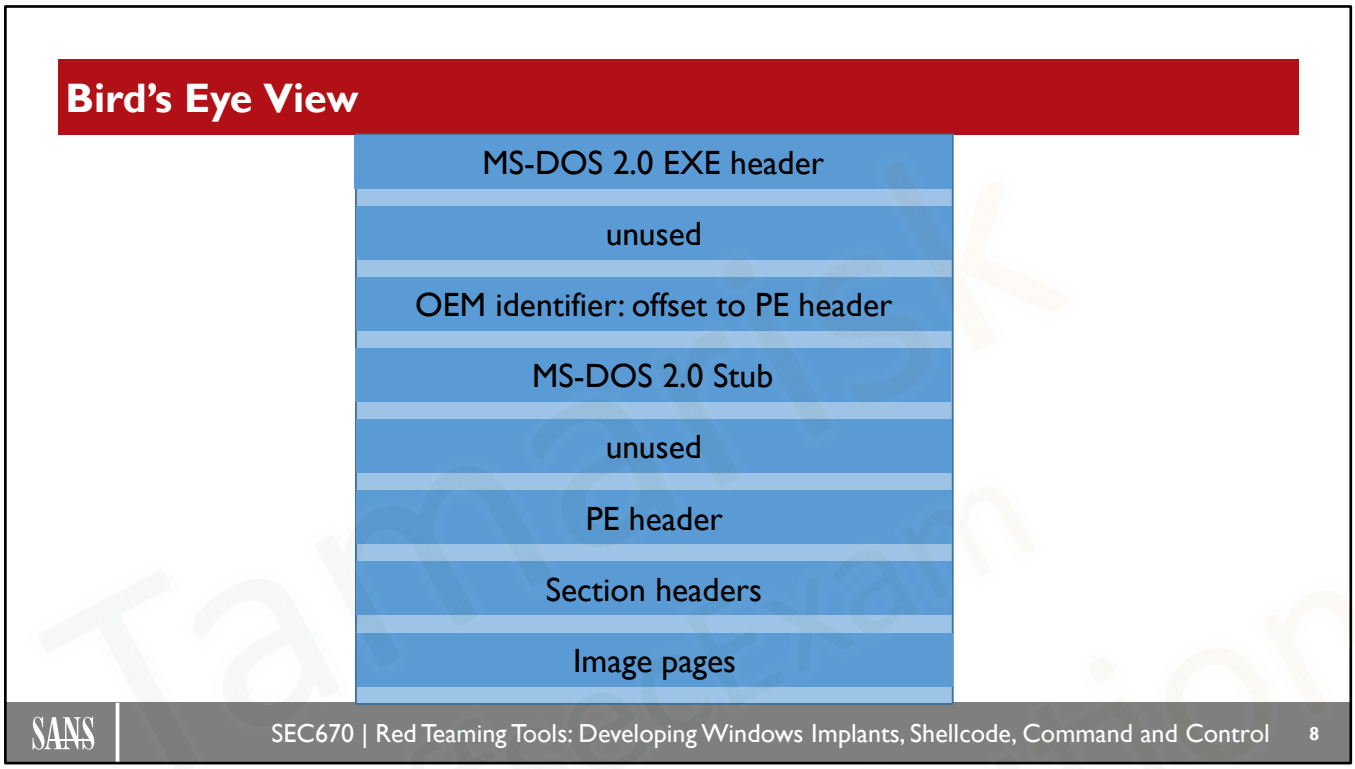
Basic Terminology

Whether you are browsing MSDN pages or blog posts related to the PE format, there are several terms that will show up repeatedly, so it is good to know them by heart. First up is Reserved. There are many structures that Windows will refer to as opaque, meaning they will not provide the full definition of a structure. Because of this, there might be several fields in a structure that are labeled Reserved. It is best to leave those alone unless you are absolutely certain you know what you are doing. Next, is RVA, which is the acronym for relative virtual address. This one should not be new to you because it was discussed in detail during Section 1 when we covered DLLs. However, it is relevant here as well, so it is provided for completeness. Next is Section, or a small segment of code or data within the image. Some of you might already be familiar with some of the section names like TEXT, DATA, BSS, PDATA, etc. In theory, there is no limit to the number of sections that a PE can have. The sections can be named anything a developer would like as the system loader could care less about section names. If you are in the field for long enough, you will come across this at some point. The last one worth mentioning is VA, or virtual address. The Virtual address is the address as it looks in memory, contained within the confines of the image's (now process) virtual address space.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Bird's Eye View

If you were to look at the PE format from a bird's eye view, this is what it might look like. This is straight from the MSDN page for the PE format but represented in a graphic to make it more digestible. The first item is the infamous EXE header that is marked with "MZ", which are the initials for the person who was a primary developer for MS-DOS, Mark Zbikowski. The more important item in that header is how to get to the PE header. The other major players here are the DOS stub, and PE header, and the next several slides will go into each one of these sections and break them down in more detail.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

MS-DOS 2.0 EXE Header

```
typedef struct _IMAGE_DOS_HEADER {
WORD e_magic;           // 00: MZ Header signature
WORD e_cblp;            // 02: Bytes on last page of file
WORD e_cp;              // 04: Pages in file
WORD e_crlc;            // 06: Relocations
WORD e_cparhdr;         // 08: Size of header in paragraphs
WORD e_minalloc;        // 0a: Minimum extra paragraphs needed
WORD e_maxalloc;        // 0c: Maximum extra paragraphs needed
WORD e_ss;              // 0e: Initial (relative) Stack Segment value
WORD e_sp;              // 10: Initial Stack Pointer value
WORD e_csum;            // 12: Checksum
WORD e_ip;              // 14: Initial Instruction Pointer value
WORD e_cs;              // 16: Initial (relative) Code Segment value
WORD e_lfarlc;          // 18: File address for the relocation table
WORD e_ovno;            // 1a: Overlay number
WORD e_res[4];          // 1c: Reserved
WORD e_oemid;           // 24: OEM identifier
WORD e_oeminfo;         // 26: OEM information
WORD e_res2[10];        // 28: Reserved
DWORD e_lfanew;         // 3c: Offset to extended header AKA PE Signature
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```


MS-DOS 2.0 EXE Header

This is the structure of the DOS header as defined in the winnt.h header file. The comments include the field offsets for the various members and a short description of what each field is for. The main areas of interest for our purposes are the first and last fields: e_magic and e_lfanew. The e_magic is a 2-byte field that holds the magic signature of PE files: MZ. Typically, when you would see this in memory, you could assume that it is part of an executable. The other field of importance found at offset 0x3C is e_lfanew, which is rumored to stand for the Long File Address for the New Executable header. This new executable header is what we are after because it sends us to the NT headers.

The winnt.h header file has a nice definition for the DOS signature: *#define IMAGE_DOS_SIGNATURE 0x5A4D /* MZ */.*

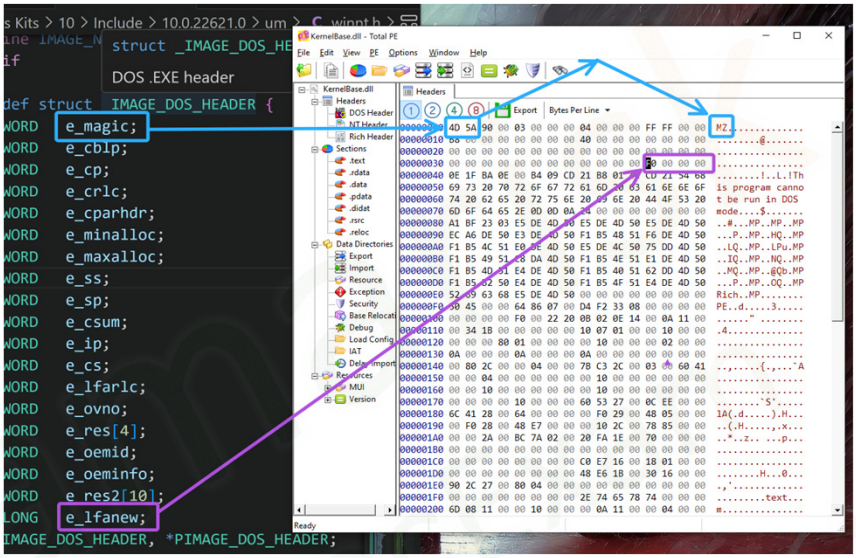
If you were developing a PE parsing tool, the other fields could be shown to the user just to give them a detailed view of the file.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

MS-DOS 2.0 EXE Header: kernelbase.dll



SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 10

MS-DOS 2.0 EXE Header: kernelbase.dll

Now that you know the structure of the DOS header, we can start to make sense of the hexdump of kernelbase.dll. The screenshot is from Visual Studio Code to see the structures and Total PE, a tool written by Pavel Yosifovich. A side-by-side layout like this can help make your way through the various PE headers and the fields inside each structure. For this IMAGE_DOS_HEADER structure, almost each field is a WORD size. Now that you know this, it is easier to identify each field and move on to the next field, you just jump two (2) bytes at a time to the end. The main hexdump inside Total PE has offsets to the left of the hexdump to easily locate the various fields.

Starting with the first field, *e_magic*, we can see the famous magic value of 4D 5A for MZ. One thing to note here is the NOP (0x90) immediately following the magic header. You will see several tools that do signature scanning while only looking for a match for MZ, but this can generate some false positives. To reduce a number of false positives, check to make sure that the 3rd byte is 0x90 and the 4th byte is 00 since they rarely change. There is no 100% guarantee the bytes following MZ will always be \x90\x00, so just keep that in the back of your mind. The second field that is important to locate and understand is the *e_lfanew* field, which is traditionally found at offset 0x3C. Whatever value found there will lead you to the next header.

© 2024 Jonathan Reiter

© SANS Institute 2024

NT Headers

```
typedef struct _IMAGE_NT_HEADERS {
 DWORD                    Signature;        // 0x00 "PE"\0\0
 IMAGE_FILE_HEADER        FileHeader;       // 0x04
 IMAGE_OPTIONAL_HEADER32  OptionalHeader;                  // 0x18
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

typedef struct _IMAGE_NT_HEADERS64 {
 DWORD            Signature;
 IMAGE_FILE_HEADER        FileHeader;
 IMAGE_OPTIONAL_HEADER64  OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;

#ifdef _WIN64
typedef IMAGE_NT_HEADERS64  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS64 PIMAGE_NT_HEADERS;
#else
typedef IMAGE_NT_HEADERS32  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS32 PIMAGE_NT_HEADERS;
#endif
```


NT Headers

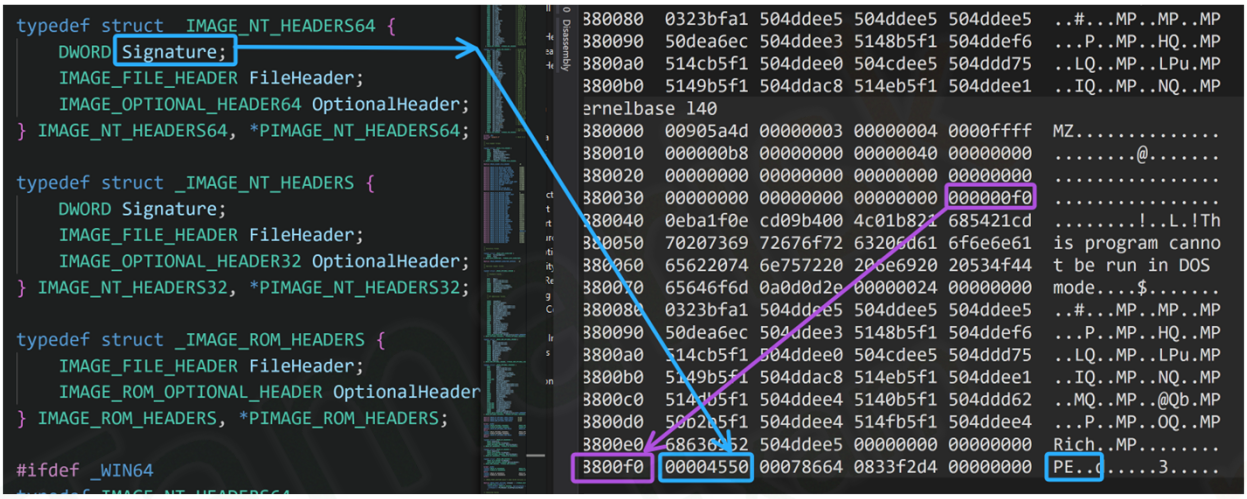
The next important structure to know is the IMAGE_NT_HEADERS struct. You will note that there is one for 32-bit applications and one for 64-bit applications. You do not need to specify one struct over the other because there is a check that happens to see if WIN64 is defined. If it is, then the structure names for 64-bit applications are typdef'd as IMAGE_NT_HEADERS. The same goes for 32-bit applications. For the moment, we will disregard that and just focus on the contents. The first field is the 4-byte signature, which is PE\0\0. The next field is the *FileHeader*, which is of struct type IMAGE_FILE_HEADER. The next slide will break down that structure, but while we are on it here there are some important fields, like the *SizeOfOptionalHeader* and *NumberOfSections*. The *NumberOfSections* is literally just that, the number of sections the executable holds: .text, .data, .bss, etc.

Following the *FileHeader* field is the *OptionalHeader* field, which is of struct type IMAGE_OPTIONAL_HEADER. This struct has a decent number of fields and several are important to us, like the DataDirectory. More on that one later.

© 2024 Jonathan Reiter

© SANS Institute 2024

NT Headers: kernelbase.dll



NT Headers: kernelbase.dll

For this side-by-side screenshot, the purple is coming from the IMAGE_DOS_HEADER->e_lfanew. That value there is used as an RVA that is then added to the base address of kernelbase.dll to give us the location of the first field in the IMAGE_NT_HEADERS struct, the *Signature*. The ASCII on the right hand side of the screenshot shows the signature being PE. Also, take note the size of the *Signature* is not two (2) bytes like it is for the IMAGE_DOS_HEADER->e_magic. This one is a DWORD, or 4 (4) bytes, so the next two (2) NULL bytes are part of it. Do not forget your endianness! Now, let us take a look at the next field, *FileHeader*.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

File Header

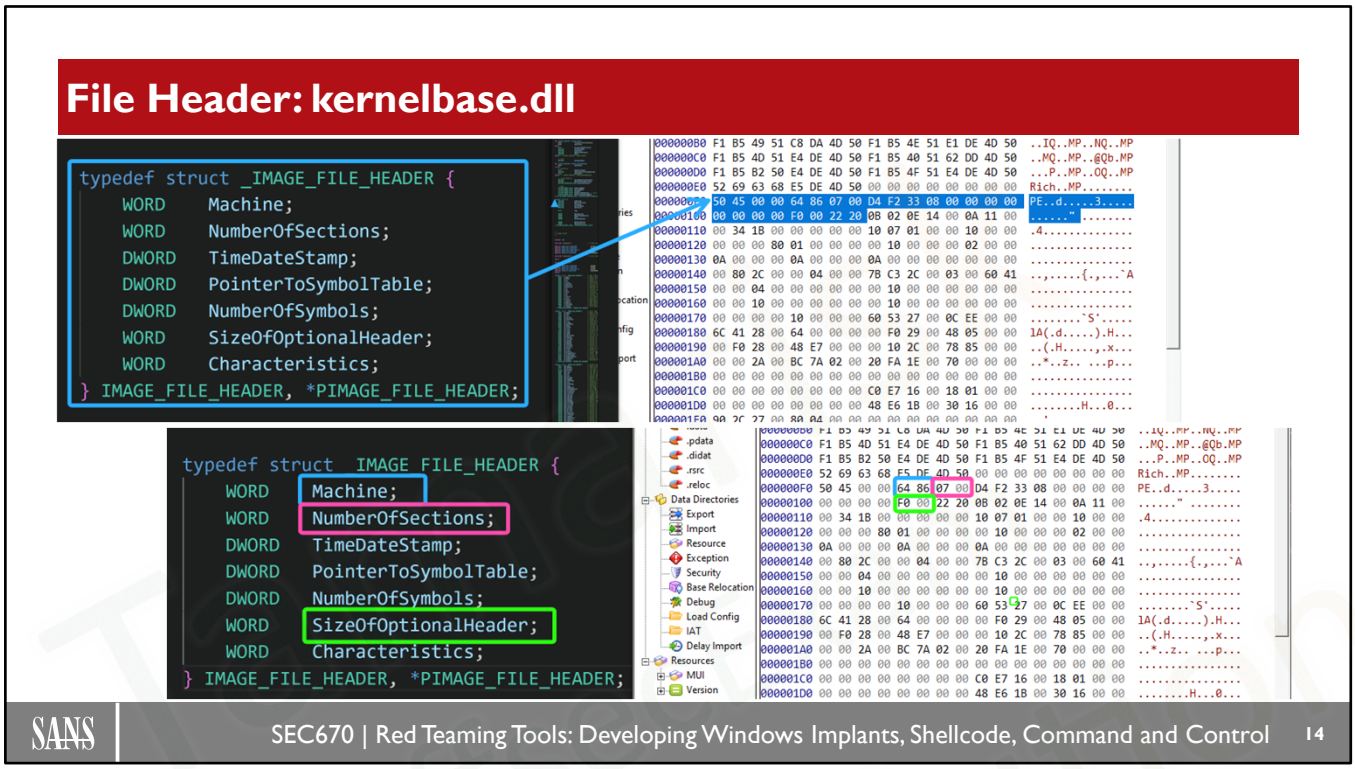
```
typedef struct _IMAGE_FILE_HEADER {
 WORD  Machine;
 WORD  NumberOfSections;
 DWORD TimeDateStamp;
 DWORD PointerToSymbolTable;
 DWORD NumberOfSymbols;
 WORD  SizeOfOptionalHeader;
 WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```


File Header

The file header holds just a few fields, but the ones that are of interest when building a PE parsing tool would be the *NumberOfSections* and the *SizeOfOptionalHeader*. The number of sections can be used in a loop to iterate over the sections, grabbing data from each one. These values also come into play when creating your own loader since each section will have to be parsed and loaded into an allocated region of memory. The header size is important because there is no direct pointer to the section headers. Therefore, once you determine the size of the optional header, you can jump directly to the section headers and start parsing those. There will be a SECTION_HEADER struct for every section in the file. The *Characteristics* field is simply some flags that are used to indicate if the file is a system file, user program, or DLL.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024



File Header: kernelbase.dll

The file header is one of the two structs inside of the NT headers struct. The upper screenshot is another side-by-side of the structure itself and the raw hexdump with Total PE. The entire IMAGE_FILE_HEADER struct is highlight in blue in Total PE so you can better see visually what it looks like. The lower side-by-side screenshot is simply highlighting the more important fields to know in the IMAGE_FILE_HEADER struct. The three (3) highlighted fields are important for us because they can be further used with our PE parsing efforts. Let us take the *NumberOfSections* field, this one indicates how many sections a binary has in it and there can be one (1) or more sections in a binary. This value can then be used to make a loop that iterates over each section until you have enumerated them all. The *SizeOfOptionalHeader* is useful because with it, we can effectively jump over the optional header and land at the first section of the binary. Typically, this first section is the .TEXT section, but that is not guaranteed. Once we have those 2 (values) we can continue with PE parsing.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Optional Header

```
typedef struct _IMAGE_OPTIONAL_HEADER64 {
 WORD      Magic;   // 0x20b
 BYTE      MajorLinkerVersion;
 BYTE      MinorLinkerVersion;
 DWORD     SizeOfCode; SizeOfInitializedData; SizeOfUninitializedData;
 DWORD     AddressOfEntryPoint;
 ULONGLONG ImageBase;
 WORD      DllCharacteristics;
 ULONGLONG SizeOfStackReserve; SizeOfStackCommit;
 IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64;

typedef struct _IMAGE_DATA_DIRECTORY {
 DWORD VirtualAddress;
 DWORD Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```


Optional Header
First off, there are several fields that are not being shown here simply because they would not all fit on the slide. Let us break down the optional header.

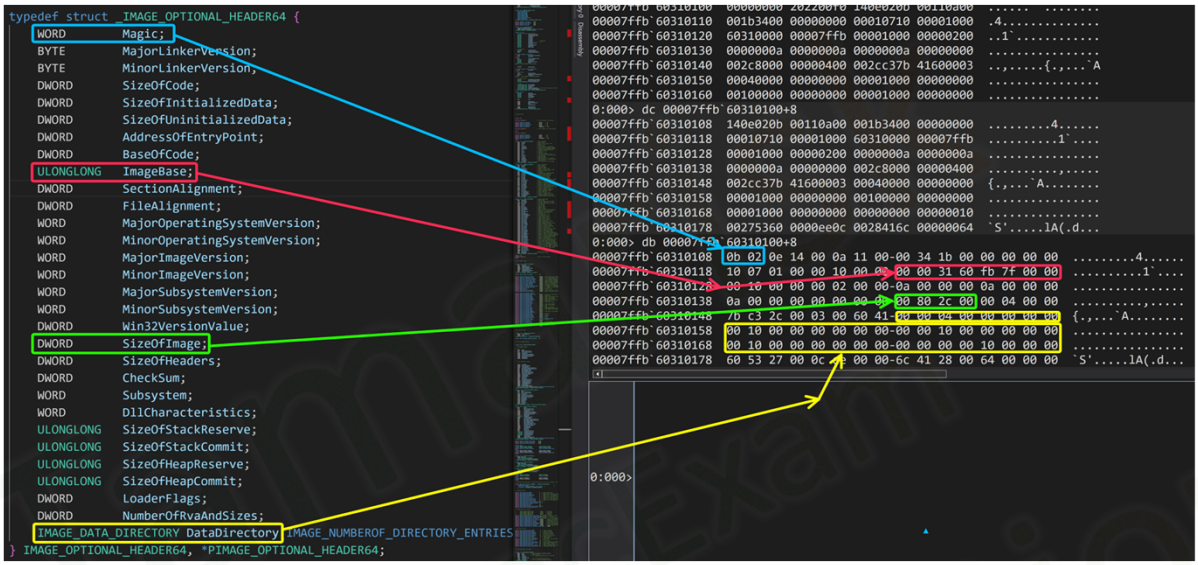
The optional header, despite its name, is not optional at all for executables; it is required. First up is the *Magic* field, which is used to indicate the state of the image file. The comment annotated on the slide (0x20b) indicates a 64-bit executable image (PE32+). On the other hand, if this were a 32-bit executable image (PE32), you would see 0x10b. The next several fields on the slide (*SizeOfCode*, *SizeOfInitializedData*, *SizeOfUnitializedData*) were placed on a single line because they are all DWORD types. These indicate the sizes for the various sections, like the .text (code), .data (initialized), and .bss (uninitialized) sections. There is also the entry point address for the program followed by the *ImageBase* field, which is the preferred address. For the *ImageBase* address, you would typically see 0x400000 for EXEs and 0x10000 for DLLs. For *DllCharacteristics*, these indicate various attributes for the image like if the image can be relocated when it is loaded, if it will be compatible with DEP, or if it will use SEH. The next two fields, *SizeOfStackReserve* and *SizeOfStackCommit*, deal with the stack. The reserve size will be set aside and only made available one page at a time when it is needed. Not all of the memory set aside for a thread's stack is committed right up front as this would just be wasteful. Rather, as stack consumption increases, more pages of reserved memory would then be committed.

The last field in the struct, and perhaps the most important one to us, is the *DataDirectory* field, which is of type IMAGE_DATA_DIRECTORY. This is a pointer to an array of data directory entries and as the comment on the slide indicates, there are 16 entries. The array entries that we care about the most are the first two, which are for the import and export entries. Inside the IMAGE_DATA_DIRECTORY struct, the field we care most about is *VirtualAdddress* as it will be key for parsing the table.

© 2024 Jonathan Reiter

© SANS Institute 2024

Optional Header: kernelbase.dll



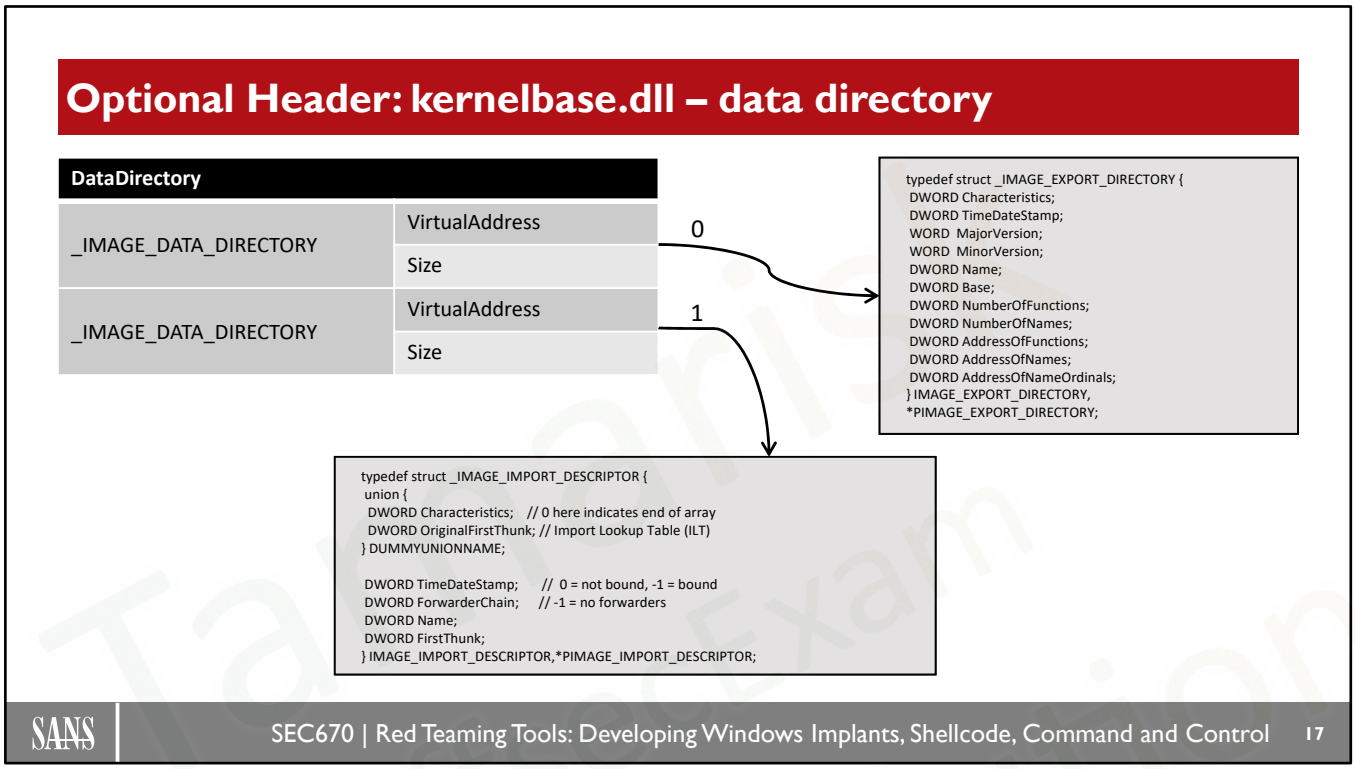
Optional Header: kernelbase.dll

This is the optional header from kernelbase.dll. The optional header is not as easy to parse because not all fields are the same size. As mentioned on the previous slide, there are a few places you can check to make sure you might be in the right place. The magic field can hold several values, but typically it will either be 0x10B or 0x20B for 32-bit (PE32) or 64-bit (PE32+), respectively. It would be very uncommon these days to see a different magic value for Windows binaries, but it could happen. The section and file alignment fields are interesting because they dictate how the sections will be aligned on disk or in memory. The section alignment value of 0x1000 indicates to the loader that the sections should be aligned on page boundaries. This is important because not every section will have the same permissions and page permissions can only be applied at single page granularity. For example, the .TEXT section of a PE32+ binary should have the characteristics of R/X. Naturally it would make sense for those same permissions be applied when mapped into pages of memory. The file alignment would be much closer together and seeing sections right next to other sections is common. Let us jump into the *DataDirectory* with a bit more detail.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Optional Header: kernelbase.dll – data directory

The last field of the optional header is arguably the most important because this is where you find the imports and exports for the image. For the most EXE binaries, there are no exports, so the value might be 0. For almost every single Windows DLL, there should be something that is being exported. You can absolutely make a DLL that does not export anything at all. Remember, the *DataDirectory* field can hold 16 arrays, or tables, which must be indexed before enumerating over them. Each entry in this table is another table of information. Inside the *DataDirectory*, you can pick apart each entry and dive into those tables to access what is need. At index 0 in the *DataDirectory* table is the entry for all exports. Each entry in the export table must be treated with the type of _IMAGE_EXPORT_DIRECTORY. Then, at index 1 in the *DataDirectory* table is the entry for all imports where each entry here must be treated with the type of _IMAGE_IMPORT_DESCRIPTOR.

Also, as you are using winnt.h as a guide for your PE parsing, you may have noticed several *#define* entries that tie into specific indexes for the *DataDirectory* table. The *#define* for exports is *IMAGE_DIRECTORY_ENTRY_EXPORT* with a value of 0. Then, there is *#define IMAGE_DIRECTORY_ENTRY_IMPORT* with a value of 1. This is a nice way to avoid the usage of what is typically called magic numbers. Magic numbers in code are just random values that can be confusing as to their meaning. Simply making a #define for them would make the code so much easier to read. Pro tip: avoid using magic numbers in your own code; just make a few #defines for them.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Exports: kernelbase.dll

```
typedef struct _IMAGE_DATA_DIRECTORY {
  DWORD  VirtualAddress;
  DWORD  Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

0:000> dx -r1 (*((combase!_IMAGE_DATA_DIRECTORY (*)[16])0x7ffb60310178))
    [0]          [Type: _IMAGE_DATA_DIRECTORY]  // exports
    [1]          [Type: _IMAGE_DATA_DIRECTORY]
    [2]          [Type: _IMAGE_DATA_DIRECTORY]
    [3]          [Type: _IMAGE_DATA_DIRECTORY]
    [4]          [Type: _IMAGE_DATA_DIRECTORY]
    [5]          [Type: _IMAGE_DATA_DIRECTORY]
    [6]          [Type: _IMAGE_DATA_DIRECTORY]
    [7]          [Type: _IMAGE_DATA_DIRECTORY]
    [8]          [Type: _IMAGE_DATA_DIRECTORY]
    [9]          [Type: _IMAGE_DATA_DIRECTORY]
    [10]         [Type: _IMAGE_DATA_DIRECTORY]
    [11]         [Type: _IMAGE_DATA_DIRECTORY]
    [12]         [Type: _IMAGE_DATA_DIRECTORY]
    [13]         [Type: _IMAGE_DATA_DIRECTORY]
    [14]         [Type: _IMAGE_DATA_DIRECTORY]
    [15]         [Type: _IMAGE_DATA_DIRECTORY]
```


Exports: kernelbase.dll

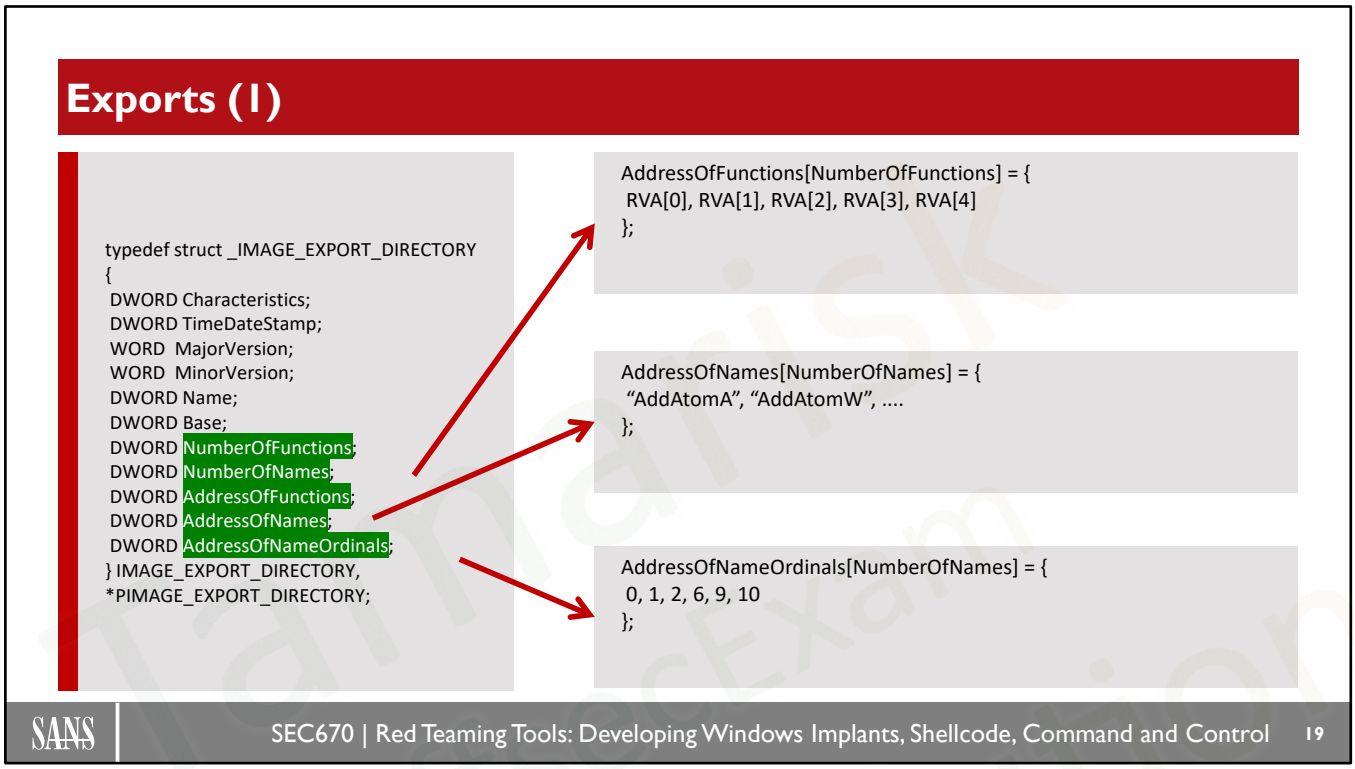
The *DataDirectory* field was located at offset F8 and since we know that it holds entries in the format of IMAGE_DATA_DIRECTORY structs, we can parse it pretty easily without using a tool.

The *VirtualAddress* of the imports entry is highlighted green, and the size of that same entry is highlighted black. When building out parsing logic, you need to remember what the data types are because they indicate how far to advance.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Exports (1)

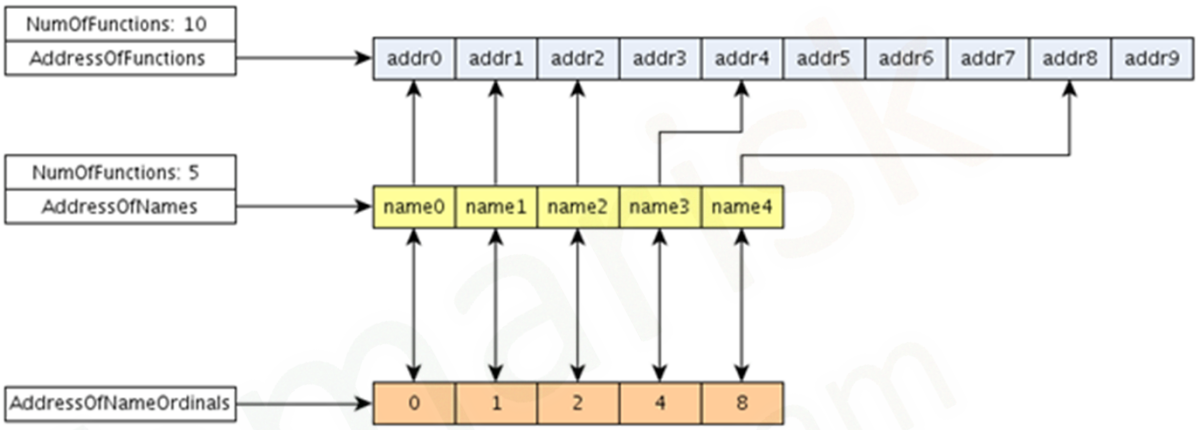
Following the exports is probably the most difficult one of all because of the added layers of complexity. The IMAGE_EXPORT_DIRECTORY struct has several important fields to us, which have been highlighted for you on the slide. The first field is the *NumberOfFunctions* and as you might have guessed, this is the number of functions that are being exported by that module (DLL). The next field, *NumberOfNames*, is the number of entries in the *AddressOfNames* and *AddressOfNameOrdinals* arrays. The last three fields are all pointers to arrays. The *AddressOfFunctions* points to an array of addresses, the *AddressOfNames* points to an array of addresses to function names, and the *AddressOfNameOrdinals* points to an array of ordinal numbers that are used as indexes into the *AddressOfFunctions* array. Multiple tools and APIs rely on these arrays for populating lists of modules and their exports, and one such API is ***GetProcAddress***.

The ***GetProcAddress***, at a high level, works by looping over the *AddressOfNames* and the *AddressOfNameOrdinals* arrays at the same time looking for a match for whatever function name you specified. Once it finds a match it can use the ordinal value as an index into the *AddressOfFunctions* array and grab the address. Later, you will be implementing your own version of ***GetProcAddress***, so be sure to know the overall logic.

© SANS Institute 2024

https://linktr.ee/offsecexam

Exports (2)



Exports (2)
This slide shows another way to visualize the flow and structure of the exports. At the top is the array holding the addresses for all of the imported functions. There happens to be 10 of them for this example, so the *NumberOfFunctions* reflects 10. If you were to only traverse the address of functions array, you would have no idea what function did what. For example, you would not know if the first entry is the address for ***GetProcAddress*** or ***TerminateProcess***. Not much can be done with that array just yet since we have not tied it to anything yet. In the middle of the graphic is the *AddressOfNames* array, which holds the addresses to the names of the imported functions. This is what is needed when creating your own version of ***GetProcAddress***. Lastly, there is the *AddressOfNameOrdinals*, which is the ordinal number that aligns with an entry in the address of names array.

The logic here is to loop over the address of names array while at the same time keeping track of your index in the array. Once you find a string match, the index you happen to be on at that time is used as an index into the address of ordinals array to obtain the ordinal number for that function name. Armed with the ordinal number, you can go right into the address of functions array and pull out that index and assign it to your function. At this point, you now have a function name, its address, and its ordinal value.

If a function is only exported by ordinal, you can simply take that ordinal and index directly into the address of functions array to grab the address. For OPSEC considerations, it is not common for a DLL to only export functions by ordinal, so keep that in mind when crafting a malicious DLL.

Reference:
resources.infosecinstitute.com/topic/the-export-directory

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Exports (3)

```
for (DWORD x = 0UL; x < NumberOfNames; x++) {
  // magical code goes here
        x=0          x=1          x=2        x=n

  strcmp(   AccessCheck | AccessCheckAndAuditAlarmW | AccessCheckByType | ...

  // found a match? Take x as an index into AddressOfNameOrdinals
        x=0          x=1          x=2        x=n

  ord =     0 | 1 | 5 | ...

  // take the ordinal value and use it as an index into AddressOfFunctions
        ord=0        ord=1        ord=2      ord=n

  rva =     1420 | 1430 | 1440 | ...
}
```


Exports (3)

For the programmatic part of it all, here is what you will have to keep in mind. Let us say you make a for loop like the pseudo code above. As you loop over the names of each function, you will be looking for a match and once you find the function of interest, the current index value of the for loop will be used as a lookup into the *AddressOfNameOrdinals* table. The ordinal value that is found at that current index is what is needed to serve as a lookup into the *AddressOfFunctions* table. Each entry in that table is an RVA so once you use the ordinal value as an index into this table, the RVA must be added to the module's base address. It might look like there is a lot going on with this slide, but there really is nothing too complicated about this. Once you start diving into the code of it all, you will see how relatively straightforward it is.

© 2024 Jonathan Reiter

© SANS Institute 2024

Exports: kernel32.dll

```
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
8B310 00 00 00 00 DF 43 FA F5 00 00 00 00 58 00 09 00
8B320 01 00 00 00 50 06 00 00 50 06 00 00 38 C1 08 00
8B330 78 DA 08 00 B8 F3 08 00 7D 00 09 00 B3 00 09 00

typedef struct _IMAGE_EXPORT_DIRECTORY {/*sizeof: 40 bytes*/
 DWORD Characteristics;
 DWORD TimeDateStamp;
 WORD  MajorVersion;
 WORD  MinorVersion;
 DWORD Name;                    // 090058 -> KERNEL32.DLL
 DWORD Base;                    // 1
 DWORD NumberOfFunctions;       // 650
 DWORD NumberOfNames;           // 650
 DWORD AddressOfFunctions;      // 08C138 -> 9007D, 900B3, 1E310, ETC.
 DWORD AddressOfNames;          // 08DA78 -> 90065 -> AcquireSRWLockExclusive
 DWORD AddressOfNameOrdinals;   // 08F3B8 -> 00, 01, 02, ETC.
} IMAGE_EXPORT_DIRECTORY,
*PIMAGE_EXPORT_DIRECTORY;
```


Exports: kernel32.dll

The example here is from a module that is a bit more mature, Kernel32.dll, which exports a larger number of functions than that of a traditional EXE. This library seemed like a perfect choice since it is loaded into almost every process. Kernel32.dll is also one that is relied upon very heavily by implant developers like us and by malware authors for the use of functions like ***GetProcAddress*** and ***LoadLibrary***. You might often see position independent shellcode rely on these two APIs and that is to give the malware the ability to load modules and find the addresses of other important APIs.

Let us break down this hexdump and jump straight to the *NumberOfFunctions* field and obtain that value; 0x650. Immediately following that is the *NumberOfNames*; 0x650. Next, in red, is the *AddressOfFunctions* with RVA 0x08C138. *AddressOfNames* follows with RVA 0x08DA78, and then last field is *AddressOfNameOrdinals* with RVA 0x08F3B8. Armed with the RVAs, you can then follow them to start diving into each of the arrays like the *AddressOfNames* array. Comments on the slide have already begun following the RVAs to their values. Feel free to follow along on your Windows 10 Dev VM using a tool like PE-bear or PE Explorer.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Imports: kernelbase.dll

```
typedef struct _IMAGE_DATA_DIRECTORY {
  DWORD  VirtualAddress;
  DWORD  Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

0:000> dx -r1 (*((combase!_IMAGE_DATA_DIRECTORY (*)[16])0x7ffb60310178))
    [0]          [Type: _IMAGE_DATA_DIRECTORY]
    [1]          [Type: _IMAGE_DATA_DIRECTORY]  // imports
    [2]          [Type: _IMAGE_DATA_DIRECTORY]
[.. SNIP ..]
    [15]         [Type: _IMAGE_DATA_DIRECTORY]

0:000> dx -r1 (*((combase!_IMAGE_DATA_DIRECTORY *)(0x7ffb60310178 + 8)))
    [+0x000] VirtualAddress   : 0x28416c
    [+0x004] Size             : 0x64
```


Imports: kernelbase.dll

The *DataDirectory* field of the _IMAGE_OPTIONAL_HEADER64 was located at offset 0x70 and since we know that it holds entries in the format of IMAGE_DATA_DIRECTORY structs, we can parse it pretty easily without using a tool, or we can just use WinDbg! The address used in the WinDbg command on the side was calculated after adding the base address to the RVA found for the *DataDirectory* field. We simply cast it to the proper type needed to see the results, but we also must do something different here because we are dealing with an array. We use the [16] in the command because we want the 16 entries to be displayed. Since we also know the size of each IMAGE_DATA_DIRECTORY entry is 8 bytes, we can manually calculate the next entry just like you see in the command on the slide. Please note, just because it says *VirtualAddress*, does not mean the actual virtual address because it is still very much an RVA that must be added to the base address of the module. Before going any further into the imports part of the PE header, let us take a look at the definition of the _IMAGE_IMPORT_DESCRIPTOR.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Imports Structure

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
 union {
  DWORD Characteristics;    // 0 here indicates end of array
  DWORD OriginalFirstThunk; // Import Lookup Table (ILT)
 } DUMMYUNIONNAME;

 DWORD TimeDateStamp;      //  0 = not bound, -1 = bound
 DWORD ForwarderChain;     // -1 = no forwarders
 DWORD Name;
 DWORD FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR,*PIMAGE_IMPORT_DESCRIPTOR;
```


Imports Structure

The imported libraries (DLLs) and their functions are stored in this array of IMAGE_IMPORT_DESCRIPTORs, which is index 1 of the *DataDirectory* array. Remember, index 0 is for the exports, which we will cover later. Highlighted on the slide are some important fields, the first one being *OriginalFirstThunk*. As the comment indicates, this is simply the RVA to get to the ILT; Import Lookup Table. Keep in mind that this is only temporary until the loader is done processing the imports and locating addresses for the imported functions. On disk, it is the RVA to the IAT; Import Address Table. Sometimes, imported functions are simply jump entries in a table that jumps program code to another DLL where the function is defined. These are called forwarded functions and there is a field called *ForwarderChain* to indicate it. If there are no functions that are forwarded, then this field will be set to -1. The next field is *Name*, which is the RVA to the name of the DLL. After that is the *FirstThunk* field, which is the RVA of the IAT. On disk, before the EXE is loaded, the *OriginalFirstThunk* and the *FirstThunk* should be pointing to the same location or have the same RVA and what they are both pointing to is another array. This array holds IMAGE_THUNK_DATA entries that represent each imported DLL. To indicate to the system loader that there are no more entries in the list, one last entry of type IMAGE_THUNK_DATA will be made but all fields will be NULL. So, if a program has 18 imported DLLs, then there will be 19 IMAGE_THUNK_DATA entries and the last one will be all NULLs to indicate the end of the array.

 © 2024 Jonathan Reiter

© SANS Institute 2024

Imports: kernelbase.dll

```
0:000> dx -r1 (*((combase!_IMAGE_DATA_DIRECTORY *)0x7ffb60310180))
  [+0x000] VirtualAddress   : 0x28416c
  [+0x004] Size             : 0x64

0:000> ? kernelbase + 0x28416c
  Evaluate expression: 140717629981036 = 00007ffb`6059416c

0:000> dt combase!IMAGE_IMPORT_DESCRIPTOR 00007ffb`6059416c
  +0x000 Characteristics  : 0x284220
  +0x000 OriginalFirstThunk : 0x284220  // import lookup table
  +0x004 TimeDateStamp    : 0         //  0 = not bound, -1 = bound
  +0x008 ForwarderChain   : 0         // -1 = no forwarders
  +0x00c Name             : 0x286a0a  // name of the imported DLL
  +0x010 FirstThunk       : 0x1be698

0:000> ? kernelbase + 0x286a0a
  Evaluate expression: 140717629991434 = 00007ffb`60596a0a
0:000> dc 00007ffb`60596a0a
  00007ffb`60596a0a  6c64746e 6c642e6c 0547006c 516c7452  ntdll.dll.G.RtlQ
```


Imports: kernelbase.dll

There is a lot happening on this slide, but do not worry, we will break it down a step at a time. The output here is from the *DataDirectory* array focusing on the imports entry. In code, it could look something like the following: *OptionalHeader.DataDirectory*[IMAGE_DIRECTORY_ENTRY_IMPORT]. Even though we went over the IMAGE_IMPORT_DESCRIPTOR on the previous slide, it is included here for easy reference while making sense of the hexdump. The size of the struct is 0x10 bytes with each field being a DWORD (4 bytes), which makes parsing it pretty easy. Remember, the first field is really a union so that is why the output in the debugger shows both at offset 0x00. The first field is the *OriginalFirstThunk*, which holds the value 0x284220. If you would follow that RVA, it would lead you to another RVA. Skipping down to the *FirstThunk* field, you can see the RVA of 0x1be698, which when followed shows you yet another RVA to find your data. The *Name* field is the name of the imported module, and it also is an RVA that can be followed. In WinDbg it is easy to follow RVAs. You can see on the slide that the *Name* RVA, when added to the base address of the module, is: ntdll.dll. The next 20 bytes would be specific for the next entry and so on and so on until you find a NULL entry. Also, you should start to see the pattern of finding your data when RVAs are given to you. Simply take the RVA value and add it to the module's base address and there you have your data. Here is a WinDbg way of getting to the next import entry.

```
0:000> dt combase!IMAGE_IMPORT_DESCRIPTOR 00007ffb`6059416c + 0x14
   +0x00c Name             : 0x286da4
0:000> ? kernelbase + 0x286da4
   Evaluate expression: 140717629992356 = 00007ffb`60596da4
0:000> dc 00007ffb`60596da4
   00007ffb`60596da4  2d697061 772d736d 652d6e69 746e6576  api-ms-win-event
   00007ffb`60596db4  2d676e69 766f7270 72656469 2d316c2d  ing-provider-l1-
   00007ffb`60596dc4  2e302d31 006c6c64 7369091e 68706c61  1-0.dll
```

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.1: GetFunctionAddress



Parse a PE file to obtain the address of a given function

Please refer to the eWorkbook for the details of the lab.

Lab 3.1: GetFunctionAddress
Please refer to the eWorkbook for the details of the lab.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to help you become intimately familiar with the format that PE files use. You should have been able to see the important information that the loader requires to properly load a file on disk and into memory. This will not be the last time you do some PE parsing. Keep this tool handy for later.

Tamarisk OffSecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Learned the PE structure is complicated at first

Covered that understanding the key components aids in future techniques

Discussed how messing with the PE structure can yield interesting results

Module Summary

In this module, we covered the PE header in depth and many of the important structures that make up the PE header. Armed with this knowledge, you can proceed to experiment with adding additional headers to see what the loader does when it comes across, let us say, two valid PE headers. The results can be interesting, to say the least.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is typically the next byte that comes after the MS-DOS header?

A 0x00

B 0x90

C 0x5A

Unit Review Questions
Q: What is typically the next byte that comes after the MS-DOS header?

A: 0x00

B: 0x90

C: 0x5A

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is typically the next byte that comes after the MS-DOS header?

A 0x00

B 0x90

C 0x5A

Unit Review Answers
Q: What is typically the next byte that comes after the MS-DOS header?

A: 0x00

B: 0x90

C: 0x5A

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



In the optional header, what magic value indicates a PE32+ binary?

A 0x20B

B 0x10B

C 0x00B

Unit Review Questions
Q: In the optional header, what magic value indicates a PE32+ binary?

A: 0x20B

B: 0x10B

C: 0x00B

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



In the optional header, what magic value indicates a PE32+ binary?

A 0x20B

B 0x10B

C 0x00B

Unit Review Answers
Q: In the optional header, what magic value indicates a PE32+ binary?

A: 0x20B

B: 0x10B

C: 0x00B

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- **Operational Actions**
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 3

PE Format
- Lab 3.1: GetFunctionAddress

Threads

Injections
- Lab 3.2: ClassicDLLInjection
- Lab 3.3: APCInjection
- Lab 3.4: ThreadHijacker

Escalations
- Lab 3.5: TokenThief

Bootcamp
- Lab 3.6: So, You Think You Can Type
- Lab 3.7: UACBypass-Research
- Lab 3.8: ShadowCraft

Threads

Before we move into injections, we need to continue learning some of the internals of the system, particularly threads.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Define a thread
- Understand various thread states
- Understand thread contexts
- Explore the structure of a thread
- Create a thread

Objectives

The objectives for this module are to define what a thread is. We will explore the various states a thread can be in and what state we need a thread to be in for a certain injection method. Each thread will have its own context that becomes relevant when a thread enters its quantum. We will also look at how a thread is structured in the system and the components of them. Lastly, we will get some hands-on practice with creating threads.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Definition



What is a thread?

According to MSDN: "A thread is an entity within a process that can be scheduled for execution."

Definition

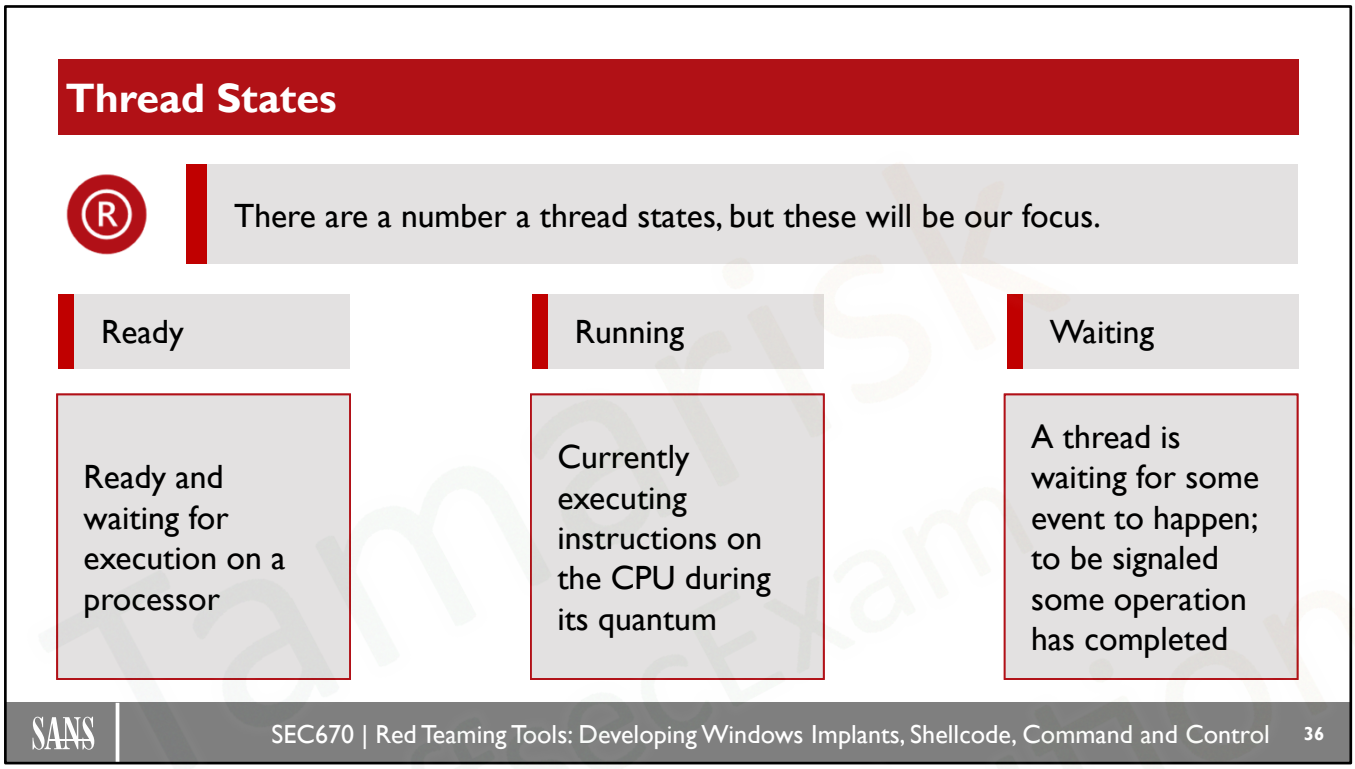
MSDN provides us with a definition of a thread: "A thread is an entity within a process that can be scheduled for execution." What does that really mean? Well, first things first, threads are what execute the instructions of a program and they are to be eventually executed by the CPU. Think of a thread as the smallest unit of execution that is tied to a process. Each process will have at least one thread that kicks off the image's code at its *AddressOfEntryPoint*. This is done after the system loader finished mapping an executable image into memory and that first is injected into the process by the kernel. This routine is how a program's main function is eventually called and inside of main's function body, more threads can be created to accomplish various tasks.

Remember, threads run and execute on a system, and processes are just mapped sections of an executable image in memory.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Thread States

There are at nine states a thread can be in at any given moment, but the focus here will only be on three states as noted on the slide: Ready, Running, and Waiting. A thread is said to be in the Ready state when it is waiting for execution and the Windows dispatcher will only schedule threads to run when they are in the Ready state. Running is the state for when a thread has officially entered its quantum on the processor and the processor has completed its switch to it. Waiting is a state where a thread is hanging around waiting for a specific event to happen. Most of the time, a thread enters the Waiting state on a voluntary basis, meaning the thread called a function like ***WaitForSingleObject***, which will make the thread become alertable. The thread then waits for that object to sync up with its execution, and a great tool to use when debugging a multithreaded program is Performance Monitor. Of course, you cannot go wrong with the best tool of all time, WinDbg Preview.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Thread Scheduling



The Windows dispatcher

Because Windows is a preemptive, and priority-based system, threads can be selected for execution but never execute because it gets preempted by a thread with a higher priority. Threads run for a certain number of clock cycles during their quantum.

Thread Scheduling

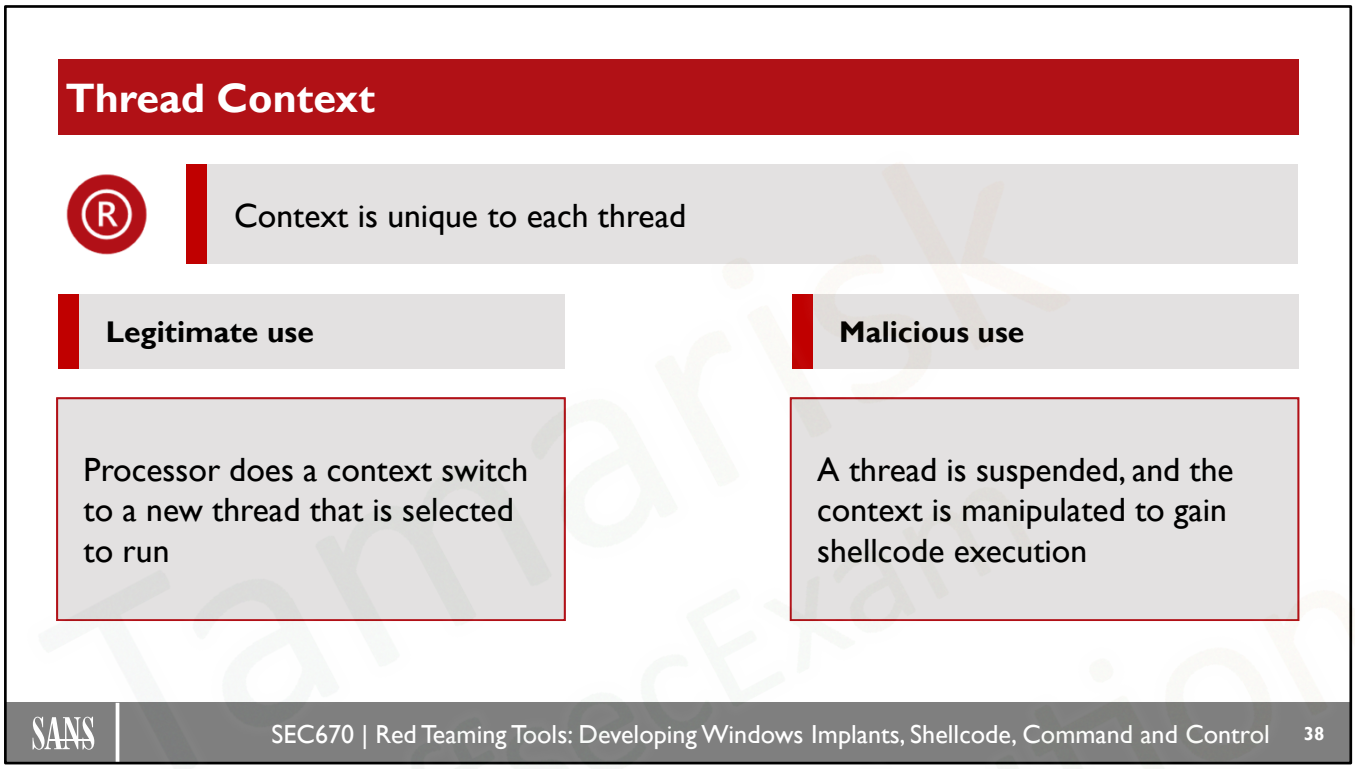
Threads can be assigned higher levels of priority that enable them to run before threads with a lower priority. When a thread with a high priority leaves its waiting state and becomes ready to run, it will preempt any other thread that is currently in its quantum if it has a lower priority. There are caveats with this, but this is the typical scheduling operation. There are a few events that might trigger the dispatcher to wake up and do its job, so let us discuss a few. One such event is the one previously mentioned: a thread leaving its waiting state. Another event is a newly created thread that is ready to run and must be scheduled. Thread priorities can be changed by system services or when it is about to hit a service call. With all the scheduling happening with threads, context switching becomes vital.

For completeness, a quantum is the time that is allotted for a thread to run on a CPU. Servers typically have a default quantum of 12 clock intervals while traditional endpoints have a quantum of 2 clock intervals. Of course, the quantum values can be changed but again, those are typically the defaults.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Thread Context

Before a different thread executes, the current state of the registers must be saved. Each thread has a context that is saved when its quantum is over or is preempted by a thread with a higher priority. These saved context states are swapped in and out each time a thread is entering its quantum. This is called context switching. The CR3 CPU control register plays a very important role when it comes to context switching because it holds the physical address of the PML4 table for x86_64 and the Page Directory Table for x86. Whatever is in the CR3 register represents the current context for the CPU. When it comes to contexts, Windows has a few APIs to deal with them: ***GetThreadContext*** and ***SetThreadContext***. We can leverage these two APIs to redirect execution to our shellcode by grabbing the context of a thread, manipulating it, then later resuming the thread. We will be doing some thread context manipulation later in this section.

For more detailed information about threads, the *Windows Internals* books are a great resources as well as the book *What Makes It Page?* for more of a memory manager perspective.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Thread Structure



ETHREAD/KTHREAD/TEB

All reside in system address space except the TEB

```
struct _ETHREAD {
 _KTHREAD       Tcb; // thread control block
 _LARGE_INTEGER CreateTime;
 PVOID          SartAddress;
 [... SNIP ...]
};

struct _TEB {
 _NT_TIB     NtTib;
 _CLIENT_ID  ClientId;
 ProcessEnvironmentBlock;    // PEB
};

struct _NT_TIB {
 ExceptionList; // EXCEPTION_REGISTRATION_RECORD
 StackBase;
 StackLimit;
};
```


Thread Structure

The kernel holds the ETHREAD and KTHREAD objects in system space, but not the TEB. The structure of a thread and its environment block are important to know, but because the ETHREAD and KTHREAD objects reside in system space, we will only be concerned with the TEB. Each thread will have its own environment block called the Thread Environment Block, or the TEB. The process and the loader must be able to read information in this structure, so that is why it resides in the process address space. You can see that the first field of the TEB is the TIB, which was originally used for Win9X applications and has remained for backwards compatibility. The TIB holds information relevant to the list of exceptions, and the size limitations of a thread's stack. For those who have taken SEC660 or FOR610, SEH should be familiar to you. For those who did not, there is a list comprised of addresses that will be called in an attempt to handle an exception. There is an entire chain of these that are linked together and the mechanism driving this is structured exception handling.

Back to the TEB, another important field is the pointer to the Process Environment Block of the parent process. The PEB, as mentioned earlier, is unique per process, and it holds valuable information we need to leverage to accomplish certain tasks, like finding loaded modules and function addresses.

In a kernel debugging session, *dt nt!_ethread* or *_kthread* will show you the structures and *dt nt!_teb* will should you the TEB struct.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

CreateThread / CreateRemoteThread



CreateThread
CreateRemoteThread

Used to create a local/remote thread

Has a HANDLE return type

```
HANDLE CreateThread(
 LPSECURITY_ATTRIBUTES lpThreadAttributes,
 SIZE_T   dwStackSize,
 LPTHREAD_START_ROUTINE lpStartAddress,
 LPVOID   lpParameter,
 DWORD    dwCreationFlags,
 LPDWORD  lpThreadId
);

HANDLE CreateRemoteThread(
 HANDLE  hProcess
 LPSECURITY_ATTRIBUTES lpThreadAttributes,
 SIZE_T  dwStackSize,
 LPTHREAD_START_ROUTINE lpStartAddress,
 LPVOID  lpParameter,
 DWORD   dwCreationFlags,
 LPDWORD lpThreadId
);
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 40

CreateThread / CreateRemoteThread
Arguably the easiest way to create a local thread is to call the ***CreateThread*** API. When the API successfully returns, it will provide a valid handle to the newly created thread. Let us break down the arguments to better understand what this function needs.

lpThreadAttributes, is an optional pointer to a SECURITY_ATTRIBUTES struct that is used to determine what security attributes will be applied to the returned handle. Leaving this NULL will not allow the handle to be inherited. For our purposes, we really do not care if our thread handles are inheritable, yet.

dwStackSize, lets you determine the size of the stack in bytes. The system will automatically round up your number to the nearest page since it must be on a page boundary. You can pass 0 here and the default stack size will be used, which is typically 1MB. The stack size is also stored in the executable image's header.

lpStartAddress, is a pointer to a function the new thread should start executing. This can also be shellcode, which is what we care about the most.

lpParameter, is a pointer to some variable that you want to be passed along to the newly created thread.

dwCreationFlags, is a thread can be created using a couple different flags, like CREATE_SUSPENDED. For the most part, 0 will be passed here since we would want the thread to execute the function right when it is created.

lpThreadId, is an optional pointer to some variable to store the thread's ID. Most of the time, we do not always care about the ID of our newly created thread, so we can just pass NULL here.

There is another variant of this function that will create a thread in a remote process: ***CreateRemoteThread***. Everything is the same except the first argument, which is a valid handle to the process.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Creating Threads



What happens behind the scenes?

- Parameters converted to flags; Client ID and TEB address added to an attribute list
- Determine if the thread should be created in local or remote process
- Call NtCreateThreadEx, initialize user-mode thread context, call PspCreateThread
- Thread is initially suspended and then later resumed so it can be scheduled

Creating Threads

For this scenario of thread creation, the main thread of a process has already been created and is running.

- The main thread creates a new thread using one of the Create Thread APIs.
- The function ends up calling an extended version called ***CreateRemoteThreadEx*** even if your code only calls ***CreateThread***. The extended version will take the arguments that were passed in and convert them to flags. It will also create the necessary native structures to describe the object parameters for the system to utilize.
- Before the thread can be made, the system must know if it should make the thread in the local calling process or another process. This determination can be done by checking the value of the handle.
- Once that is determined, it can call ***NtCreateThreadEx*** from Ntdll to make the jump into kernel mode with the exact same arguments.
- The user mode thread context will be made, and the executive thread object will be suspended by calling ***PspCreateThread***. After the system is done initializing everything, the thread is eventually resumed and is now ready to be scheduled for its quantum.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Defined threads
- Covered various threads states
- Discussed thread contexts and structures

Module Summary

In this module, we discussed what threads are from an internal perspective, we explored how to create threads using two different APIs, we looked at contexts, quantums, etc. We now have enough knowledge to understands what happens when we inject into a thread or suspend a thread.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



When a new thread is created, where does the object reside?

A User space

B System space

C Process handle table

Unit Review Questions
Q: When a new thread is created, where does the object reside?

A: User space

B: System space

C: Process handle table

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



When a new thread is created, where does the object reside?

A User space

B System space

C Process handle table

Unit Review Answers
Q: When a new thread is created, where does the object reside?

A: User space

B: System space

C: Process handle table

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What state is a thread in during its quantum slice?

A Running

B Waiting

C Ready

Unit Review Questions
Q: What state is a thread in during its quantum slice?

A: Running

B: Waiting

C: Ready

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What state is a thread in during its quantum slice?

A Running

B Waiting

C Ready

Unit Review Answers
Q: What state is a thread in during its quantum slice?

A: Running

B: Waiting

C: Ready

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the default quantum for servers?

A 2 clock cycles

B 8 clock cycles

C 12 clock cycles

Unit Review Questions
Q: What is the default quantum for servers?

A: 2 clock cycles

B: 8 clock cycles

C: 12 clock cycles

Tamarisk ObscExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the default quantum for servers?

A 2 clock cycles

B 8 clock cycles

C 12 clock cycles

Unit Review Answers
Q: What is the default quantum for servers?

A: 2 clock cycles

B: 8 clock cycles

C: 12 clock cycles

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- **Operational Actions**
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 3

PE Format
- Lab 3.1: GetFunctionAddress

Threads

Injections
- Lab 3.2: ClassicDLLInjection
- Lab 3.3: APCInjection
- Lab 3.4: ThreadHijacker

Escalations
- Lab 3.5: TokenThief

Bootcamp
- Lab 3.6: So, You Think You Can Type
- Lab 3.7: UACBypass-Research
- Lab 3.8: ShadowCraft

In this module, we will discuss several techniques centered around injection. There are a large number of injection methods, and as you can see, we have our hands full of exercises in this section. To kick things off, this module will start with the classic DLL injection.

Tanay OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Define what injection is and reasons for it
- Explore several injection techniques and their mechanisms
- Understand more about the PE structure
- Rehash threads
- Discuss some mitigations

Objectives

The objectives for this module might seem light on the surface, but they are not. We are going to be diving deep into various methods of injection followed by a brief mention of possible mitigations.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Process Injection

What exactly is process injection and what are some reasons for injecting into a process?

Forcefully making a process execute arbitrary code

Avoid detection by having a legit process execute your shellcode

Process Injection

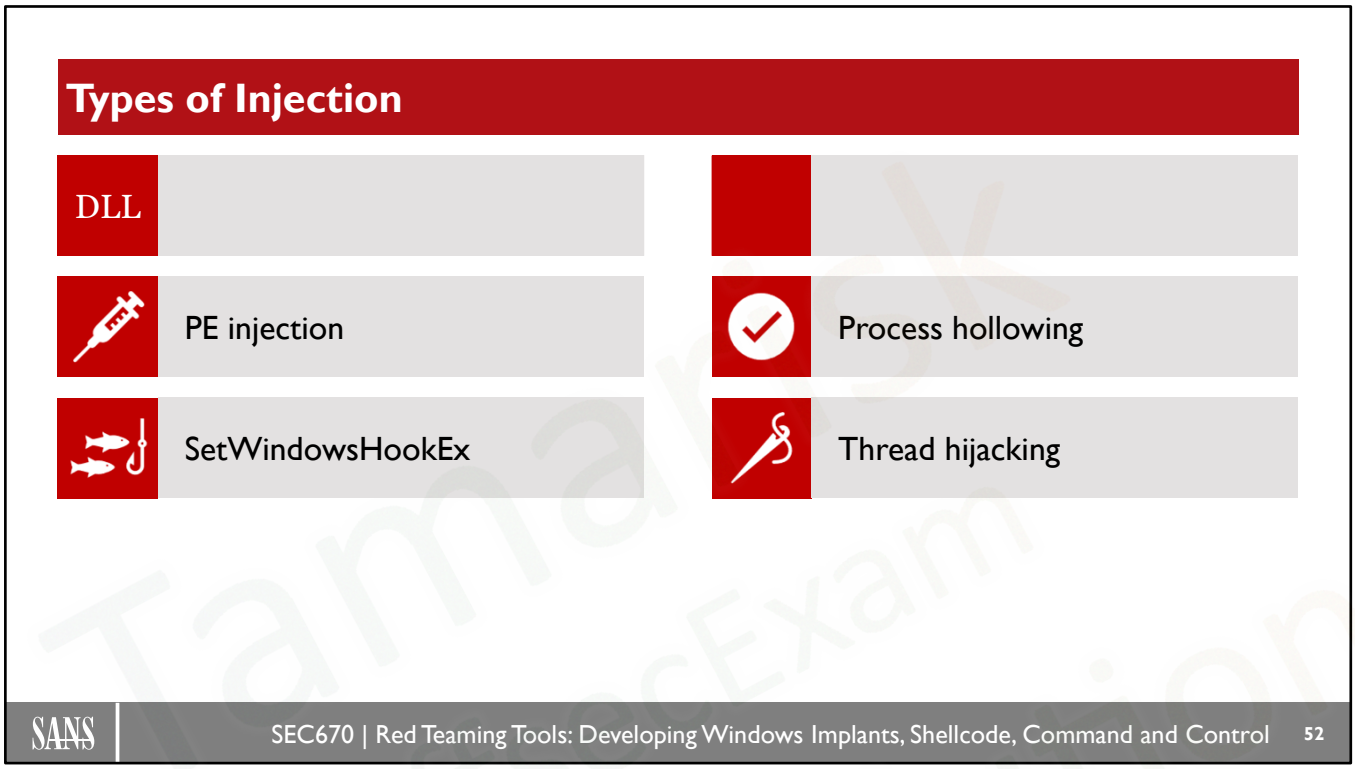
Depending on what blog post you read or what YouTube video you watch, you might get a different definition of what process injection is. For this course, we define process injection as a method of forcing code from one userland process, say malware, into another userland process to execute arbitrary code. We will discuss other techniques that are not true to that definition, like AppInit, Image File Execution Options, DLL hijacking, or other pre-execution styles. It is good to know many techniques and what some of the cons might be for a certain technique.

There could be several reasons for injecting into a process, but what we are concerned with is trying to avoid various levels of detection. You could easily develop a program that executes shellcode locally, meaning inside your own process' virtual address space. Detecting local execution could be rather trivial, but if you could have another process, say a legitimate Windows process or third party application execute your malicious deeds, why not have it do so. Depending on what technique you implement, this could help avoid detection for a longer period of time.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Types of Injection

There are several process injection methods, and this slide is not a comprehensive list of every single process injection technique that is out there, but it is enough to get us going. At this point, a few injection methods will be discussed at a high level and later on will be the deep dive into each one. First up is the famous DLL injection. There are two different versions of DLL injection: classic and reflective. In this section, we will just focus on classic and save reflective injection for later in the course. At a high level, this technique forces another process to load and execute a DLL of your choosing.

Next is APC injection. The asynchronous procedure call is a Windows mechanism that allows functions to execute, as the name implies, asynchronously in some thread. With this technique, we look for certain threads, or all of them, that we can queue an APC to and have it execute our code.

PE injection is what I call PE inception because we are hiding a PE inside of another PE, a dream within a dream. Once the attack is executed, the target process will be holding two PE images, the legitimate one and ours.

Process hollowing creates a new process, but it does so in the suspended state so no code can execute from it. Then it proceeds to hollow out the original image with a new one and once done, the main thread can be resumed. In the end, it still holds the same process name, but on the inside, it is completely different.

SetWindowsHookEx is an API that can be used to inject a DLL into a GUI process. It could also be one of many methods used to act as a key logger. This technique depends on the process having the User32 module loaded.

Thread hijacking is the manipulation of a thread's context so that it will wind up executing shellcode we give it.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Classic DLL Injection



The injector and the injectee

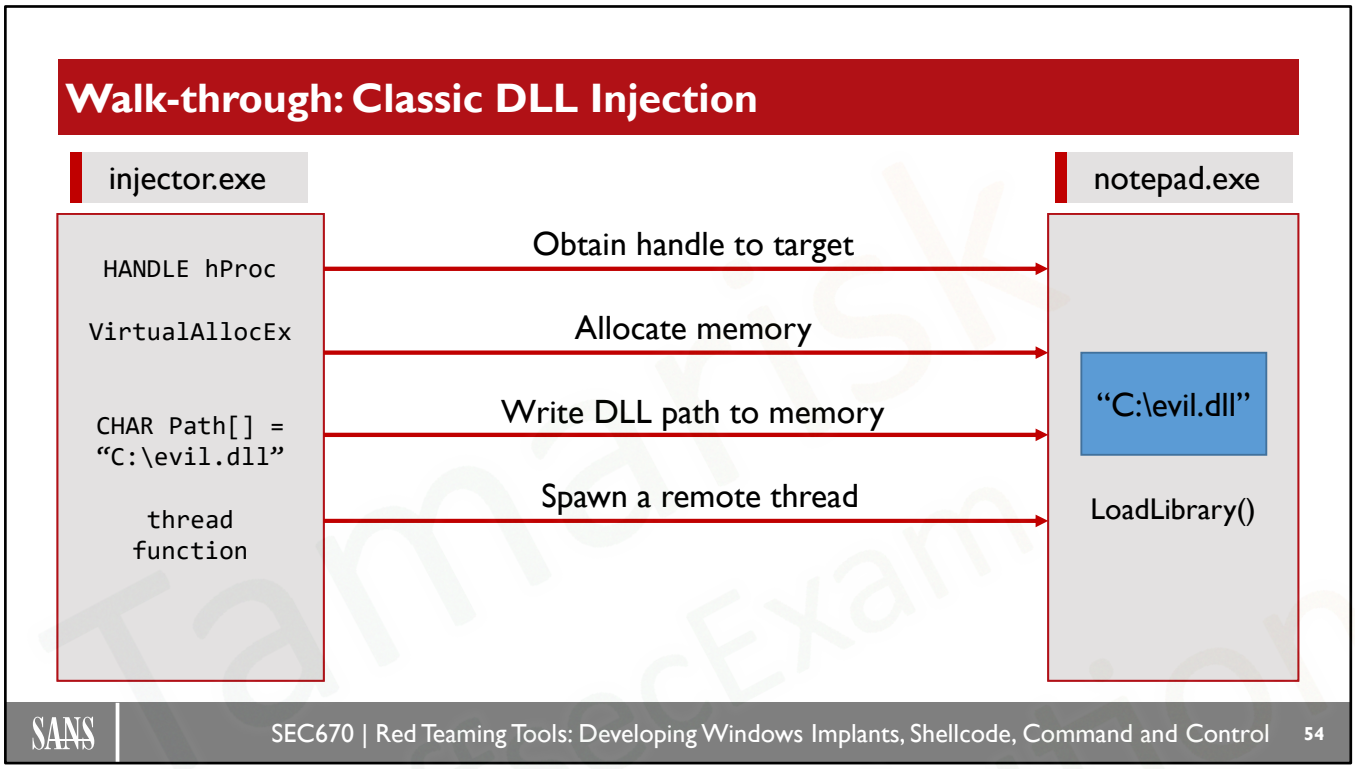
For DLL injection to work properly, you must obtain a handle to the target processes to allocate memory pages, write to them, and create a new thread. This logic will be implemented in the DLL injector. The DLL is the malicious portion of this method.

Classic DLL Injection

Classic DLL injection is fairly common as it is not too complex of a method to understand. The main functions involved with this method are ***OpenProcess***, ***VirtualAllocEx***, ***WriteProcessMemory***, and ***CreateRemoteThread***. The main purpose of this method is to forcefully load a DLL into the address space of the target process and then execute it. The items you must provide as the tool developer would be the injector portion of the tool. The DLL to be injected can be provided by the operator of the tool, or even by you. You will be creating your own at first because you need to test your tool to work out any bugs.

© 2024 Jonathan Reiter

© SANS Institute 2024



Walk-through: Classic DLL Injection

Classic DLL injection is arguably the easiest to perform. If you were to look at this from a 10,000-foot view of the overall process, this is what it would look like. There are a few things that are needed for this to be successful: the injector and the DLL. The injector has the responsibility of executing the logic for injecting the DLL into the target process. One of the first steps of the injector is to obtain a valid process handle to the target process via ***OpenProcess***. The handle must have the necessary permissions to get the job done; permissions will be discussed during the source code review. Once the handle is obtained, a chunk of memory must be allocated in the target process via ***VirtualAllocEx***. This newly allocated space is used to store the absolute path to the DLL via ***WriteProcessMemory***. The final step is to call the ***CreateRemoteThread*** API and hope everything works as desired.

With these steps in mind, let us jump over to the source code.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.2: ClassicDLLInjection



Injecting your own DLL into a target process

Please refer to the eWorkbook for the details of the lab.

Lab 3.2: ClassicDLLInjection
Please refer to the eWorkbook for the details of the lab.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to get your feet wet with injecting a DLL into a target process. This also puts down the foundation so it can be built upon later in the course when we go over reflective DLL injection.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

APC Injection



GUI applications have threads too!

Synchronous and asynchronous execution: what is the difference and how does each come into play? Asynchronous allows an operation to execute in the background while the user application is doing something else instead of being frozen.

APC Injection

Asynchronous procedure call, or APC, is a mechanism that allows an I/O operation to be completed later. When an I/O operation is set to complete immediately, the application will wait and ends up blocking the user from being able to do anything else. This is said to be a synchronous operation. On the other hand, asynchronous operations should allow the user to continue working while the operation is carried out behind the scenes. As an example, there could be a thread running in some GUI process like Notepad and when the user attempts to load a large file, that read operation could happen asynchronously. Should it be synchronous instead, the GUI application might appear to be frozen or not responding. For asynchronous I/O, the thread that kicked off the operation will be notified, or alerted, when the operation is complete via an APC object. The APC objects are placed into a queue properly named the APC queue. Each thread has its own APC queue and when a thread gets its quantum time, a check is done to see if there are any items in the queue, and if there are, they get executed. This means that APC functions, or callbacks, will execute in the context of the thread from which they were requested, and will only be sent when that thread is in an alertable waiting state.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Queueing an APC



QueueUserApc

Return value is DWORD

Used to queue an APC to user thread

```
DWORD
QueueUserAPC(
 PAPCFUNC  pfnAPC,
 HANDLE    hThread,
 ULONG_PTR dwData
);
```


Queueing an APC

Brought to you by the processthreadsapi.h header file, the ***QueueUserApc*** API will add an APC object to the APC queue of the specified thread. Now, let us look at the function declaration. The function has a return type of DWORD, or unsigned long, and has three parameters: pfnAPC, hThread, and dwData.

pfnAPC, is of type PAPCFUNC and should be a pointer to the APC-typed function, or shellcode, that you want to be called when the thread becomes alertable or does some alertable action.

hThread, is of type HANDLE and is a valid handle to the thread of which you are attempting to queue an APC object. The handle should, at a minimum, have the THREAD_SET_CONTEXT access right.

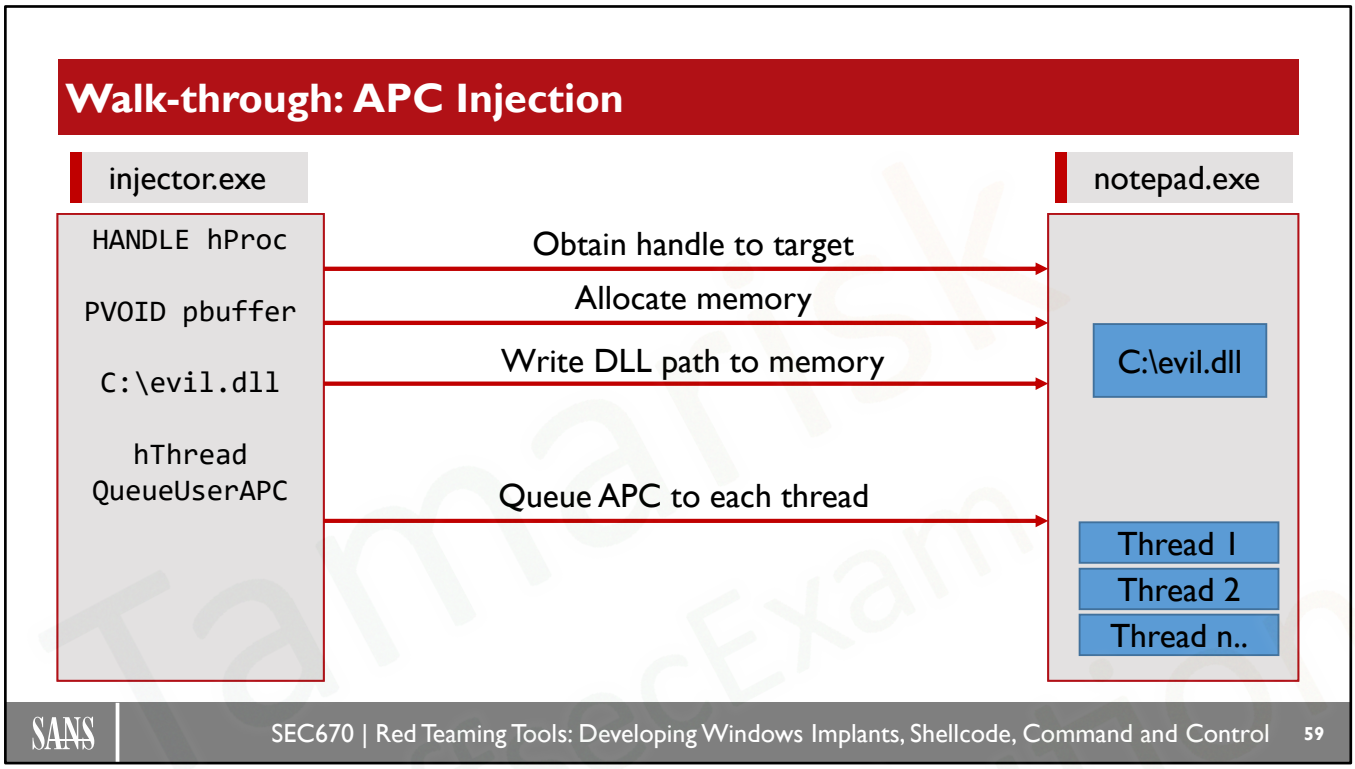
dwData, is of type ULONG_PTR and for our purposes, this can just be NULL. Other use cases, this could be one value that gets passed to the APC function being called.

If the function ever fails, be sure to call ***GetLastError()*** to find out why.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Walk-through: APC Injection

The first step for APC injection is to obtain a handle with the correct permissions to the target process. Once that is obtained, allocate a chunk of memory in the target process so the path to the DLL can be stored. Remember, this must be the full path of the DLL on disk. Before we can queue the APC to each thread, we need to figure out how many threads the process has running. This is when the snapshot comes into play. We can make a list of all threads that belong to the target process and enumerate over that list later in code. If any new threads kick off after we did our enumeration, they would not be discovered. Since the list is now populated with thread IDs, we can iterate over it and obtain a thread handle to each thread. Using that temporary thread handle, we can call the ***QueueUserApc*** function each iteration to queue an APC. What exactly are we going to queue? The APC function will be the address of ***LoadLibraryA*** and the pBuffer will be the argument passed to ***LoadLibraryA***.

With these steps in mind, let's jump over to the source code.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.3: APCInjection



Queue an APC to a target thread.

Please refer to the eWorkbook for the details of the lab.

Lab 3.3: APCInjection
Please refer to the eWorkbook for the details of the lab.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to explore the details of the APC injection method, the APIs involved, manipulating threads, and gaining execution for your code.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Thread Hijacking



Taking over a process' thread

Hijacking a thread can be useful when you do not want to target every thread or create a new thread. Hijack the first thread of the target process and redirect its execution to your shellcode.

Thread Hijacking

Hijacking a thread means that we are going to take over and control what it is executing. Threads have a context that the system must keep track of and that is what we are going to modify; the context. The context of a thread cannot be modified until the thread is in a suspended state, so we will have to suspend the thread by calling ***SuspendThread***. Once suspended, we can obtain the context by calling ***GetThreadContext***. When our shellcode, or other payload, has been copied into the target process' address space, we can redirect execution to it and resume the thread. The CONTEXT structure is very complete in the sense that it will capture all registers of the CPU at the state they were in right before the context switch.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Obtaining Context



GetThreadContext

Return value is BOOL

Used to obtain a thread's context

```
BOOL
GetThreadContext(
 HANDLE    hThread,
 LPCONTEXT lpContext
);



typedef struct _CONTEXT {
 [..SNIP..]
 DWORD64 Rip;
};
```


Obtaining Context

Each thread will have a context structure that is unique to each thread. The context struct is fairly large and it must be so that the system can keep track of the context each time a different thread enters its quantum. A prime example for obtaining a thread's context is a debugger. When a thread is being debugged, its context must be made known. For our purposes, we will grab the context and modify the instruction pointer so that when we resume the thread, it will be pointing to our shellcode. Below is a larger snippet of the CONTEXT structure for a 64-bit OS.

```
typedef struct _CONTEXT {
 [...SNIP...]
 DWORD   EFlags;
 DWORD64 Dr0;
 DWORD64 Dr1;
 DWORD64 Dr2;
 DWORD64 Dr3;
 DWORD64 Dr6;
 DWORD64 Dr7;
 DWORD64 Rax;
 DWORD64 Rcx;
 DWORD64 Rdx;
 DWORD64 Rbx;
 DWORD64 Rsp;
 DWORD64 Rbp;
 DWORD64 Rsi;
 DWORD64 Rdi;
 DWORD64 R8;
 DWORD64 R9;
 DWORD64 R10;
```

© 2024 Jonathan Reiter

© SANS Institute 2024

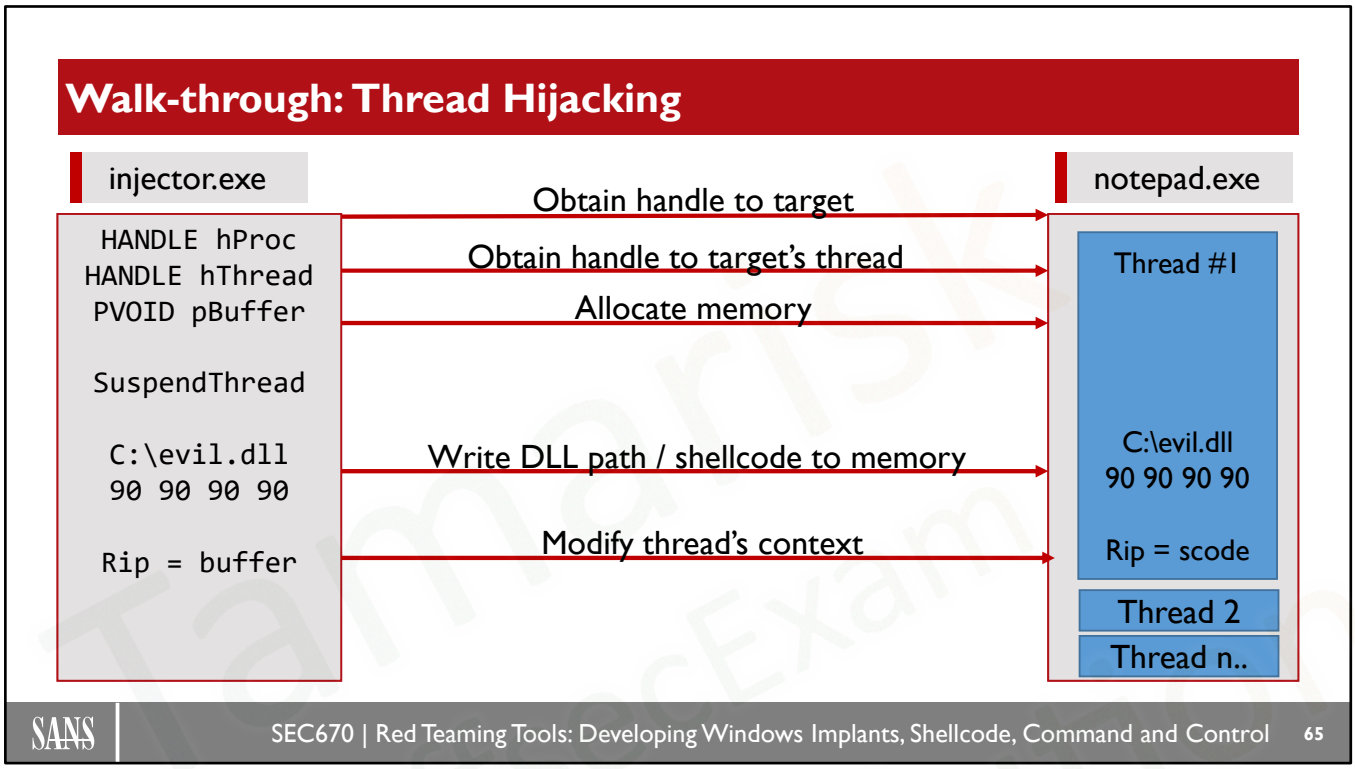

```
DWORD64 R11;
  DWORD64 R12;
  DWORD64 R13;
  DWORD64 R14;
  DWORD64 R15;
  DWORD64 Rip;
  [...SNIP...]
  DWORD64 LastExceptionToRip;
  DWORD64 LastExceptionFromRip;
} CONTEXT, *PCONTEXT;
```

Tamarisk
OffsecExam
Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Walk-through: Thread Hijacking

As you perform the various injection methods, you might notice that some portions repeat or carry over from one method to the next. You would be right because they do have very similar portions. The slide here shows a high-level walk-through for hijacking a thread's context. There are two handles that must be obtained: the handle to the target process (hProc) and a handle to a thread (hThread). Both handles must have the minimum permissions necessary to achieve our objectives. For the process, we have virtual memory operations to carry out and we will need to write to the virtual memory. For the thread, we need to suspend it, get its context, update the context, and set its context. The only modification we need to make is changing the instruction pointer, which needs to point to the recently allocated memory (pBuffer). After everything has been completed, we can resume the thread. If we did everything correctly, the shellcode will be executed, and it will load our library.

With these steps in mind, let us jump over to the source code.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.4: ThreadHijacker



Hijack execution of a thread

Please refer to the eWorkbook for the details of the lab.

Lab 3.4: ThreadHijacker
Please refer to the eWorkbook for the details of the lab.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffSecExam Exam solution

What's the Point?
The point of this lab was to explore the process of hijacking a thread's context.

© SANS Institute 2024

https://linktr.ee/offsecexam

Process Hollowing



Hollow out the current process' image.

Instead of targeting threads, we can target the executable image itself by replacing it with an image of our own. The current image is removed, or hollowed out, and an image of our choosing is copied over.

Process Hollowing

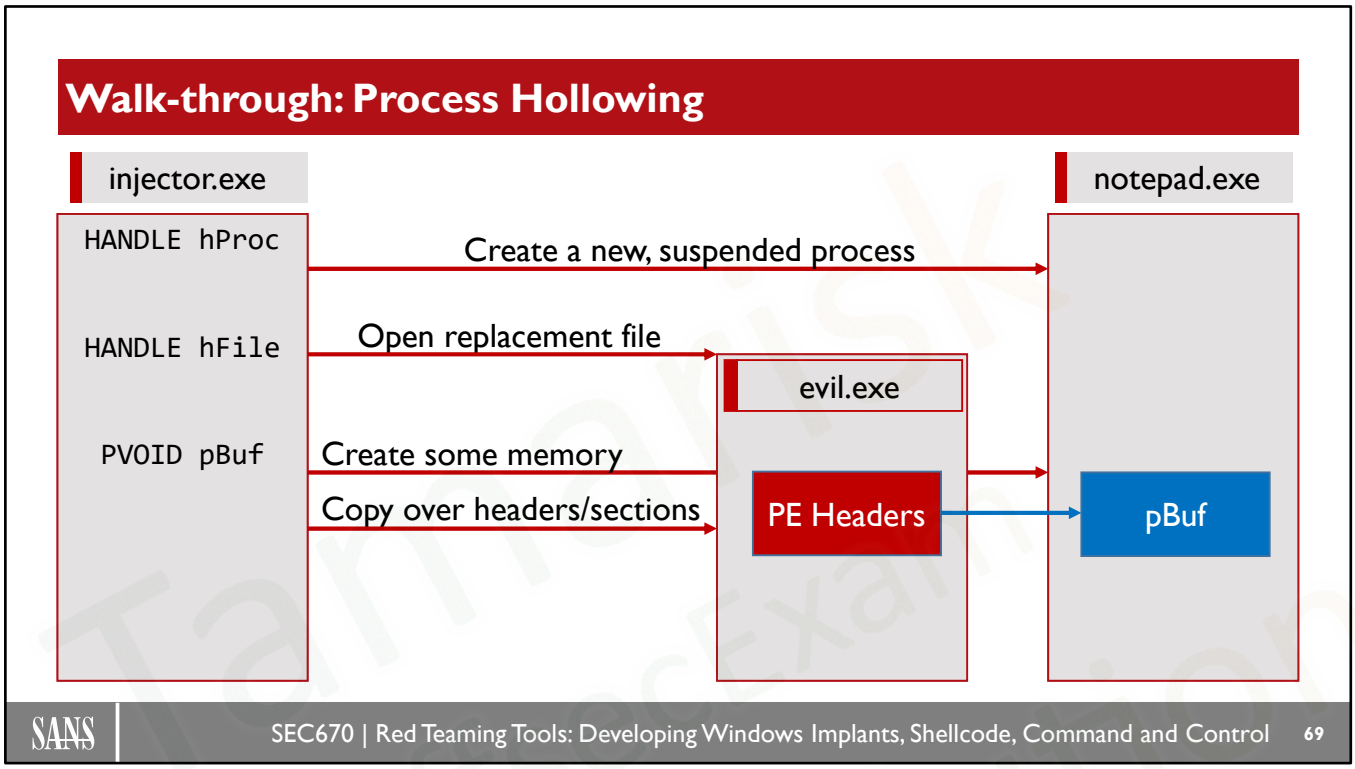
You do not always have to target a process' thread(s) to achieve your injection needs. If you recall, an EXE is mapped into memory where we then call it an image. Once the image is fully mapped into memory, certain portions of the image can be removed without causing any visible changes. Process hollowing does this removal and it removes the current process' image directly from memory. The replacement image will be written to the target process' address space, including all sections, and the main thread will be resumed after modifying its context. We have covered a few methods so far and you might already have some idea as to what APIs might be involved with this method.

Something to think about while we discuss this method: what would happen if you were to create crss.exe, rundll32.exe, or another seemingly legit process in a suspended state and hollow it out?

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Walk-through: Process Hollowing

On the slide is a very high-level overview of the steps involved for hollowing out a process. The first step is to create a new process. This new process is going to be "hollowed out" and will soon contain the PE headers and sections of a different image. The important note here when creating the process is to pass the CREATE_SUSPENDED flag so that the process does not continue its operations. Once the new process is loaded and we have our handle to it, we can allocate a buffer in that process. We do not know how large the buffer should be until we open the replacement image and determine its size. Next, we need to copy over the replacement's PE headers as well as its sections.

Now, even though we have done a lot of work so far, nothing has really changed within the suspended process. If it were to resume execution, our replacement image would not be executed. We can change that by modifying the suspended process' Process Environment Block (PEB). We give it an updated ImageBase address with that of our replacement image. Finally, after updating the thread's context, we can call ***ResumeThread*** and our replacement image should be executed.

With these steps in mind, let us jump over to the source code.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

PE Injection



A PE injected inside another PE

Instead of replacing the PE image of the target process, or hollowing it out, this method simply adds an additional PE image inside. The target process will literally be holding two PE images.

PE Injection

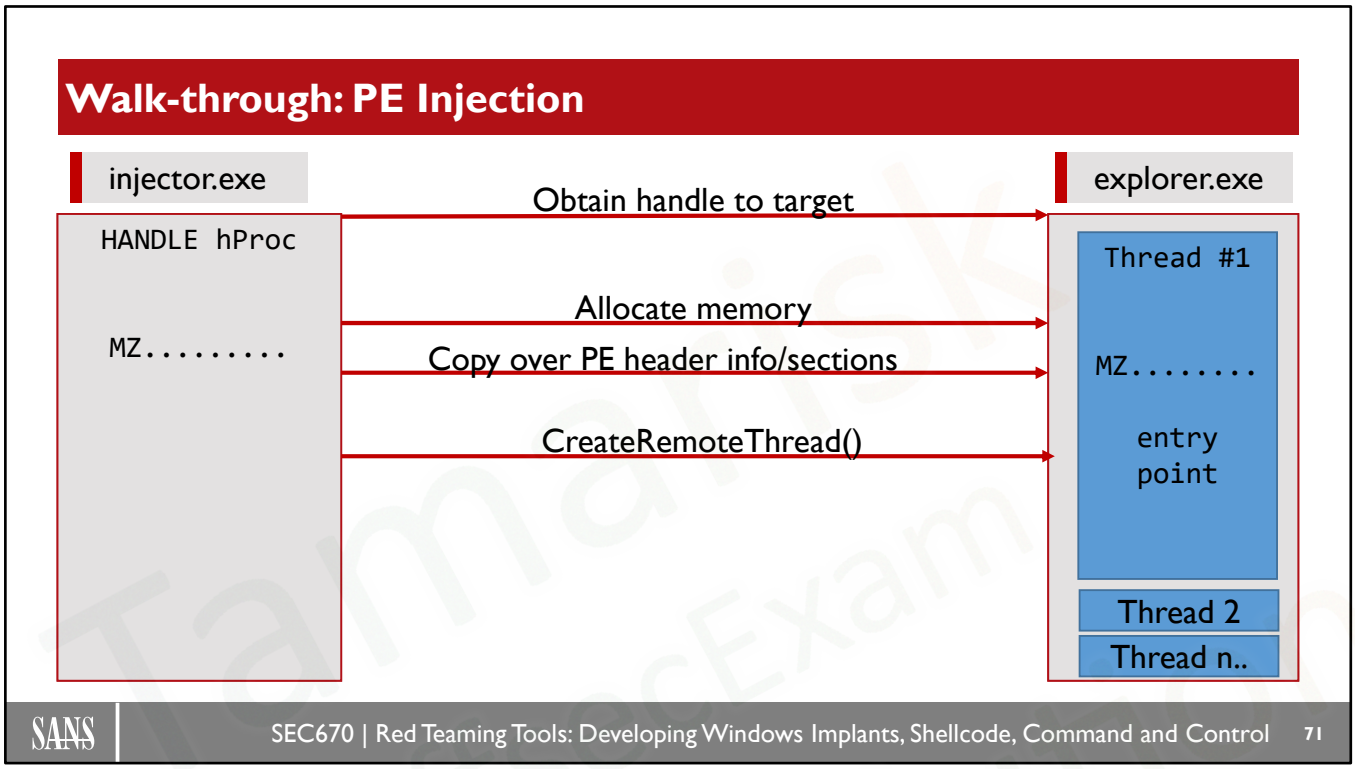
PE injection can sometimes be confused with the process hollowing method because we are injecting a PE file in another process just like with process hollowing. However, this method does not "hollow" out the image from the target process. So, instead of the "hollowing", we are adding another PE image in the target process. The main idea here is that a handle will be obtained to the target process, some virtual memory will be allocated in the target process, our PE data will be copied into the newly allocated memory space, and finally a remote thread will be kicked off to execute our program. The APIs that this method uses are not new, but here they are: ***CreateToolhelp32Snapshot*** to get the PID of the target process, unless manually entering it; ***OpenProcess*** to obtain a handle to the target process; ***GetModuleHandle*** to obtain a handle to own process; ***VirtualAlloc*** and ***VirtualAllocEx*** to handle the allocation of memory in our own process and remote process; ***WriteProcessMemory*** to copy the data over, and ***CreateRemoteThread*** to create a thread in the target process.

Some new ones that you have not seen yet are to enable the debug privileges: ***OpenProcessToken*** to obtain a token handle, ***LookupPrivilegeValue***, and ***AdjustTokenPrivileges***. There is also some PE header parsing that will take place.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Walk-through: PE Injection

We can try to visualize PE injection just like what we have done with the other injection methods thus far. Again, this method is extremely similar to process hollowing with a few differences. With this method, there is no need to have any shellcoding knowledge or experience since we can do what we need to do purely in C/C++ and the Windows APIs. Also, unlike process hollowing, we do not have to create a new process and place it in a suspended state.

- Breaking it down, we have a few of the same steps we have seen already, minus getting a handle to a thread. Assuming we have the target Process ID in hand, we must obtain a handle to the target process using an API like ***OpenProcess***.
- With the newly acquired process handle, we must then allocate a chunk of memory in the target's virtual address space. To determine how much memory to allocate, we must obtain the size of the PE image we are injecting. If you recall, this value is stored in the Size of Image member of the Optional Header structure.
- After the memory has been allocated, we can then proceed with copying over our PE image to include all sections. It would be a good idea to apply fixups to the .reloc section because we are not being loaded by the system loader who normally does that for us.
- We must find the delta of where the base of our image is—basically the address returned to us from the call to the ***VirtualAllocEx*** API—and the preferred base address.
- The image base address is under the Optional Header, so we use that to determine the delta offset. After all fixups or relocations have been applied, we are finally ready to kick it off with the execution of our injected image.
- We take care of that using the ***CreateRemoteThread*** API by passing it the address of our entry point. If everything has gone perfectly smooth, then you will have successfully executed a PE image inside the address space of another PE image.

© SANS Institute 2024

https://linktr.ee/offsecexam

SetWindowsHookEx



GUI applications can be targets as well.

GUI applications might need to hook events so they can properly respond to certain events like mouse clicks or keyboard events. This method requires the target process have the User32.dll module already loaded. This API is very robust and using it for DLL injection is barely scratching the surface for its capabilities.

SetWindowsHookEx

The ***SetWindowsHookEx*** API is a very comprehensive API that can do more than a simple DLL injection. Windows uses it primarily for hooking events like mouse clicks and keyboard events for GUI applications that have focus. There can be an entire chain of events that have been hooked and each one will be iterated over to see who needs to respond to a specific event like someone typing in a Notepad++ window. The API has both an ANSI and Wide char (Unicode) version so you would see ***SetWindowsHookExA*** or ***SetWindowsHookExW*** when looking at online documentation. The hook chain mentioned is what we will be injecting a hook into. More specifically, we will be adding a hook event into an already existing hook chain. The events are system level events, and it is up to Windows to determine what application the hook event procedure is destined. It could be meant for a specific thread or all threads running on the system, or rather the same Desktop session as us—the calling thread.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

SetWindowsHookExA



SetWindowsHookExA

Adds a hook procedure to hook chain

Has a HHOOK return type

```
HHOOK SetWindowsHookExA(
 int       idHook,
 HOOKPROC  lpfn,
 HINSTANCE hMod,
 DWORD     dwThreadId
);

// ... EXAMPLE ...

SetWindowsHookExA(WH_GETMESSAGE,
 EvilHookFunction,
 hEvilDll,
 ThreadId );
```


SetWindowsHookExA

Despite the ***SetWindowsHookEx*** API only having four parameters, they are powerful, as you will see during the lab and via your own reading/experimentation. Let us go ahead and break down these parameters.

idHook, is an *int* type and is used to indicate what hook procedure type will be utilized. The possible values start with WH_* and perhaps the WH stands for Windows Hook. Some options are CALLWNDPROC, which is used to grab messages before the OS pushes them to the appropriate window. GETMESSAGE is used to literally get messages from a message queue. KEYBOARD and MOUSE are the interesting ones because they monitor messages from the keyboard and mouse. This option would give you the chance to intercept and change those messages.

lpfn, of type HOOKPROC, is used to point to the function, or the procedure that is to be called when the message event is generated. This should be pointing to an exported function in a DLL that you make. You could have the hook procedure in the code tied to the current process, but only if the Thread ID is not 0.

hMod, of type HINSTANCE, is used to serve as the handle to the DLL that implements the hook procedure. Again, this procedure would be the one that is exported by a DLL that you make and is pointed to by the previous parameter, *lpfn*.

dwThreadId, of type DWORD, is used to associate the hook to a certain thread. We want to pass 0 here to be associated with all threads that are executing in the same desktop session as us—the calling thread. Because we are using 0 here, we must have the *lpfn* hook procedure in a DLL.

Upon successful execution, the API will return a handle to the hook (HHOOK) that was added to the system's chain of hooks.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Learned process injection comes in many forms; results are really the same

Discussed how some methods have similar techniques

Discussed how some methods serve as an injection method but also offer bonus features

Module Summary

In this module, we covered several methods of injection and even tossed in something extra with the **SetWindowsHookEx** API. If you are a defender taking this class, then it is great to understand how injection is implemented programmatically. It might give you some ideas as how to create detection signatures for this if your organization does not already have some implemented. For the offensive students, it is great to explore the various ways to carry out injection, and as you saw, there is no single way to do anything. Perhaps there are even more methods that are not even known by the public.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What API allows you to inject a DLL into GUI applications?

A CreateRemoteThread()

B WriteProcessMemory()

C SetWindowsHookEx()

Unit Review Questions
Q: What API allows you to inject a DLL into GUI applications?

A: CreateRemoteThread()

B: WriteProcessMemory()

C: SetWindowsHookEx()

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What API allows you to inject a DLL into GUI applications?

A CreateRemoteThread()

B WriteProcessMemory()

C SetWindowsHookEx()

Unit Review Answers
Q: What API allows you to inject a DLL into GUI applications?

A: CreateRemoteThread()

B: WriteProcessMemory()

C: SetWindowsHookEx()

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



When hijacking a thread, what construct must be modified?

A Thread state

B Thread context

C Thread priority

Unit Review Questions
Q: When hijacking a thread, what construct must be modified?

A: Thread state

B: Thread context

C: Thread priority

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



When hijacking a thread, what construct must be modified?

A Thread state

B Thread context

C Thread priority

Unit Review Answers
Q: When hijacking a thread, what construct must be modified?

A: Thread state

B: Thread context

C: Thread priority

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What mechanism allows threads to process routines when it enters its quantum?

A APC queue

B Contexts

C Event objects

Tamarisk OffSecExam Exam solution

Unit Review Questions
Q: What mechanism allows threads to process routines when it enters its quantum?

A: APC queue

B: Context

C: Event objects

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What mechanism allows threads to process routines when it enters its quantum?

A APC queue

B Contexts

C Event objects

Unit Review Answers
Q: What mechanism allows threads to process routines when it enters its quantum?

A: APC queue

B: Context

C: Event objects

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- **Operational Actions**
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 3

PE Format
- Lab 3.1: GetFunctionAddress

Threads

Injections
- Lab 3.2: ClassicDLLInjection
- Lab 3.3: APCInjection
- Lab 3.4: ThreadHijacker

Escalations
- Lab 3.5: TokenThief

Bootcamp
- Lab 3.6: So, You Think You Can Type
- Lab 3.7: UACBypass-Research
- Lab 3.8: ShadowCraft

In this module, we will discuss and implement various ways to escalate your local privileges.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss the reasoning for escalating privileges
- Explore several methods
- Implement a few methods in code

Objectives

The objectives for this module are to talk about the reasoning for escalating your privileges. There could be times when you do not have to operate with higher privileges than what you have already. We will also discuss and explore several methods to programmatically elevate your current privileges.

Tamarisk OffSecExam Exam Solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Why Escalate?

? Is there always a requirement to escalate privileges?

Knowing what you can or cannot do with your current level of privileges is important for your tool and the operator using it.

non-admin

admin

Why Escalate?

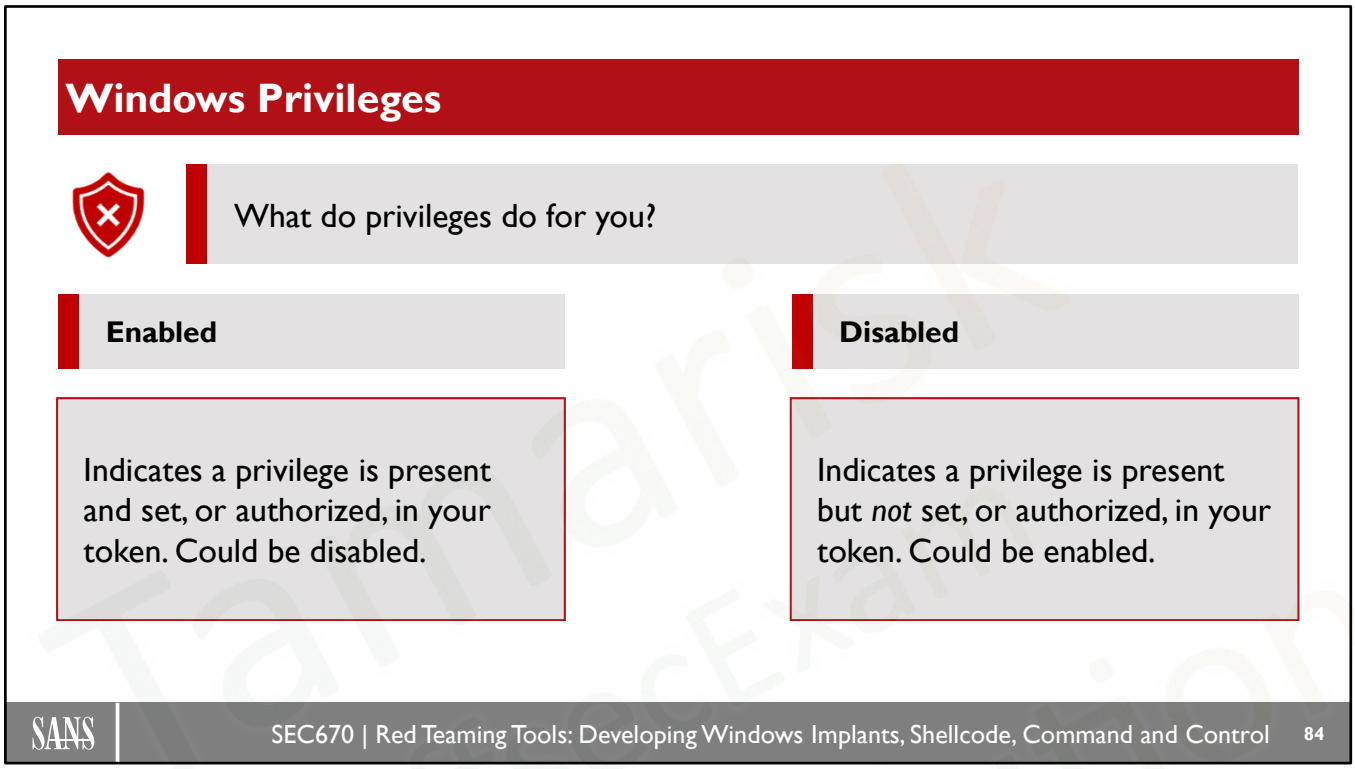
There is only so much that you can do with standard user permissions and accesses. Depending on what the end goal is or what you need to accomplish on the target, you might not even need to escalate your privileges. That might contradict from what others have said and what you might know and that is fine. All I am saying is, why waste your time coding something to be done as Admin when it all could have been done as a regular user. A lot of enumeration and survey tools can run without needing to be Admin and can still pull back enough information about your target to decide if that target is worth bringing down additional tools or capabilities.

Anyway, say you finally made the choice that you must be Admin to accomplish its needs. Now you need to find a way to escalate or gain more privileges. In my opinion, this could be something small like enabling the debug privilege. The debug privilege could come in handy when enumerating processes or opening process handles to them. Some processes will not allow you to open a handle to them because they might belong to a different user, but if you have the debug privilege (*SeDebugPrivilege*, SE_DEBUG_NAME), then you should not have much of a problem. What are privileges anyway?

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Windows Privileges

What is a privilege and what does it do for you? According to Microsoft, "A privilege is the right of an account, such as a user or group account, to perform various system-related operations on the local computer, such as shutting down the system, loading device drivers, or changing the system time." Now that we know what a privilege is, we can discuss what can be done or cannot be done. When a target is first compromised, you would typically find yourself in a standard user account. Very rarely are you exploiting a system and are greeted with domain admin right out of the gate. So, as a standard user with basic privileges, what can be done? What privileges are typically enabled for that account's token? As mentioned previously, when performing process enumeration, you will not be able to obtain a process handle to certain processes if you do not have the *SeDebugPrivilege* enabled. The process manager is the gatekeeper for that action and will happily reject your attempt if the access checks do not pan out. Typically, when Windows creates your token for your user account, you cannot change the privileges that are present, in other words, the privileges that are displayed after issuing the *whoami /priv* command. However, if you make the jump into the kernel, you can do whatever you want and give yourself new privileges. What about stealing privileges in the form of tokens? Let us get ready to dive into that.

Reference:
https://docs.microsoft.com/en-us/windows/win32/secauthz/privileges

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Securable Objects



Objects that have a corresponding security descriptor

Most objects are created at the request of the user: *CreateProcess*, *CreateThread*, *CreateFile*, etc. The functions typically accept a pointer to a SECURITY_ATTRIBUTES structure. There are several object types that can be secured.

files | **processes** | **reg keys** | **threads**

Securable Objects

Remember the section where we were talking about the ***Create**** family of Win32 APIs? It might seem like a long time ago, so here is a refresher. The Create* API family is a way for a user-mode application to ask the kernel to create on object. The kernel does so, after checking a few items, and gives the application a handle to the newly created object. The handle is safe and sound, stored away in the process' handle table. The *SECURITY_ATTRIBUTES* parameter for some of the Create* functions, like ***CreateProcess*** or ***CreateFile***, allow objects to be secured, or hardened, depending on what is passed in as an argument. If you pass in NULL for the SECURITY_ATTRIBUTES parameter, you are saying to the system that the default security descriptor is good enough, and typically the default is good enough. So, when someone attempts to call ***OpenProcess*** on that process, the process manager is going to check to see what the security descriptor is for that process. If everything checks out, then your handle will be returned. If not, you will be greeted with the infamous 0xC0000005 (*ERROR_ACCESS_VIOLATION*) error.

There are several types of objects that you can specify security descriptors for to control access to them. The types noted on the slide are not an all-inclusive list, but rather some of the more common objects you might create and/or come across.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Can I Have a Token?



Access tokens are given after successful authentication.

Each process that you create after logging in will have a primary token, which is tied to you, the user. The security descriptor of the token can and will be checked when attempting to access a securable object.

- `SID/logon SID`
- `Privileges`
- `Default DACL`
- `Primary or impersonation`

Can I Have a Token?

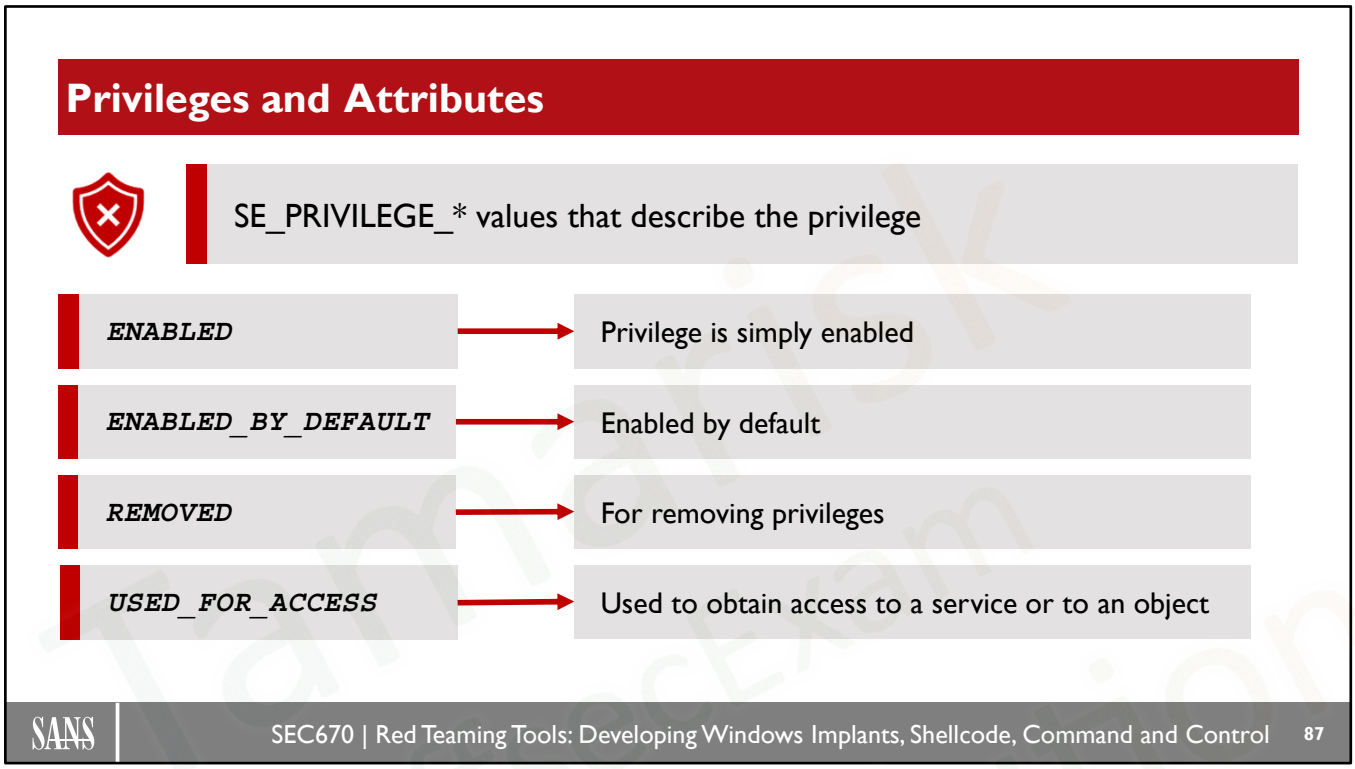
Right when you log on to a system, your user account will be given a token, which holds information like what operations you are or are not allowed to carry out. This is called your access token as it describes your level of access. When it comes to processes that you create, whether being done programmatically or by double-clicking an executable, it should have a primary access token tied to it, which is then tied back to you. There is also an impersonation token, which is created when a process, like a server process, impersonates a client thread to interact with the object using the client's permissions. Impersonation is a way to protect access to sensitive objects since services typically run at a higher integrity level than that of a client process connecting to it.

What is in a token and what is inside a token? There are several items annotated on the slide that can be inside of a token, but of course there are more items that a token holds. For starters, we have the security identifiers (SIDs), which represent the user or group, but also in the token is where the access token originated, and the SID for the primary group. Our favorite header file, winnt.h, defines many *TOKEN_** structures, like *TOKEN_USER*, *TOKEN_ORIGIN*, *TOKEN_PRIVILEGES*, *TOKEN_GROUPS*, and many more. The *TOKEN_PRIVILEGES* structure is an interesting one because it contains an array of privileges for a specific access token. The array entries are of type *LUID_AND_ATTRIBUTES* structures, which, by no surprise, holds the *LUID* and attributes of a privilege. As the name eludes, privileges do have attributes.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Privileges and Attributes

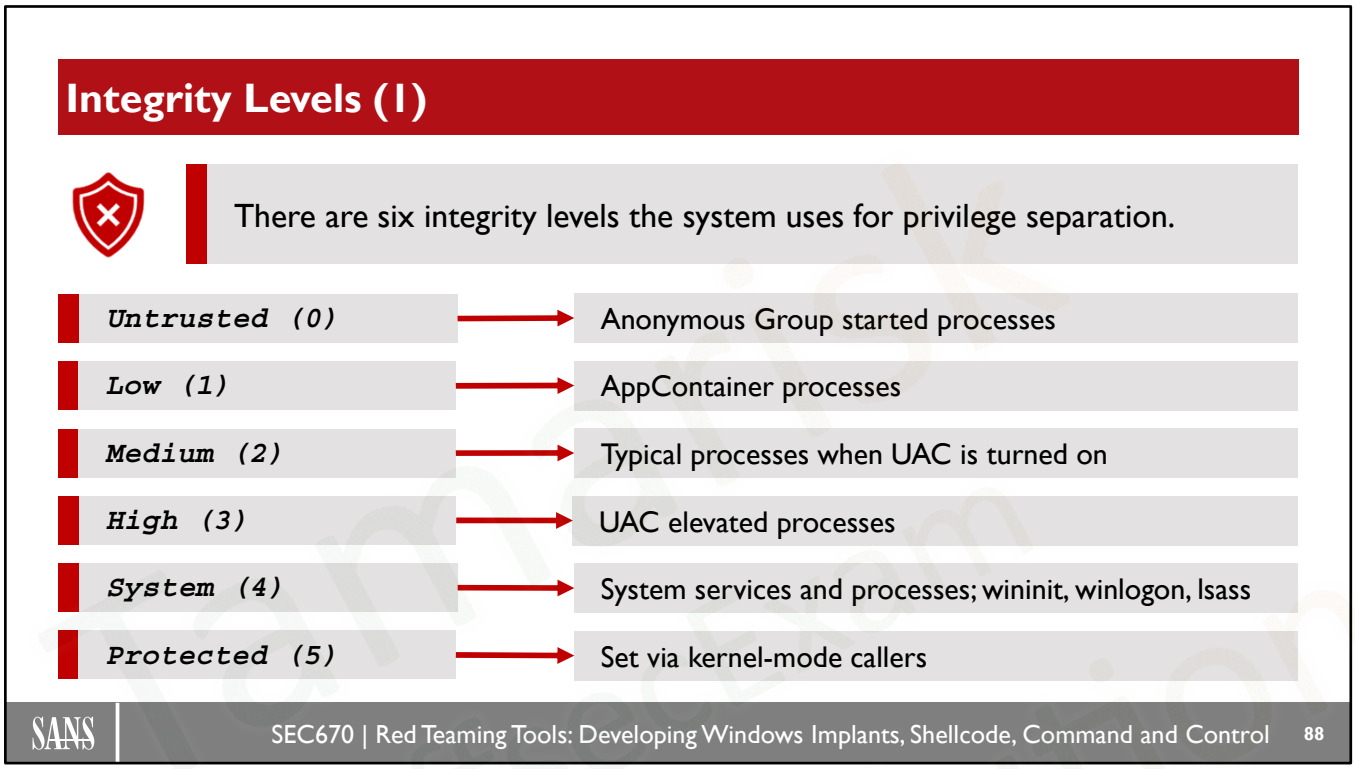
Privileges are what determine if a user, or process, is allowed to carry out an operation within the system. Privileges are not necessarily tied directly to an object, but rather are tied to what can be done. Debuggers like WinDbg Preview that attach to processes must have the *SeDebugPrivilege* privilege, and programs that wish to load a kernel mode driver must have the *SeLoadDriverPrivilege* privilege. All privileges have attributes, and they can be any one of the attributes listed on the slide, as well as a combination of attributes. As mentioned previously, the attributes for a privilege are stored in the *TOKEN_PRIVILEGES* structure; more specifically, the privileges member of the structure, which is a *LUID_AND_ATTRIBUTES* struct. Here is how that struct is defined.

```
typedef struct _LUID_AND_ATTRIBUTES {
  LUID Luid;
  DWORD Attributes;
} LUID_AND_ATTRIBUTES, *PLUID_AND_ATTRIBUTES;
```

The Luid is a locally unique identifier that is used for identifying privileges unique to the current boot and system. Furthermore, they are 64-bit values, which aids in their uniqueness for that current boot. The Luid has attributes that are comprised of bit flags; 32 of them. The *LUID* structure has two members: *DWORD LowPart* and *LONG HighPart*. The members hold the unsigned low portion of the ID and the signed high portion of the ID, respectively.

© SANS Institute 2024

https://linktr.ee/offsecexam



Integrity Levels (1)

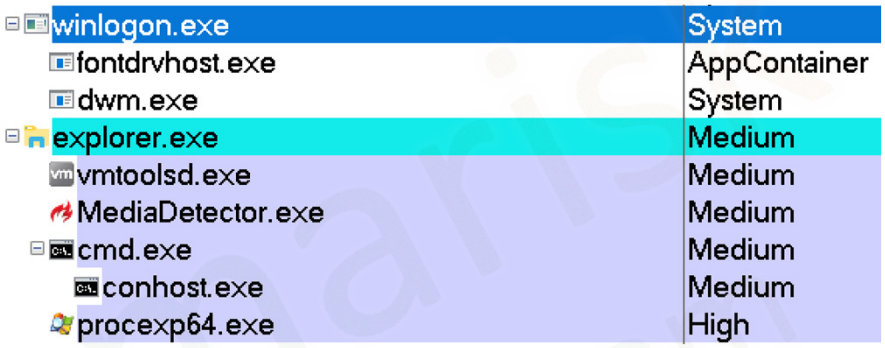
Tokens can have an integrity level (IL) tied to them, which can be queried using the ***GetTokenInformation*** function. There are six levels implemented by the OS and they aid in the separation of privileges, with the lowest level being 0, the Untrusted level. It has the SID S-1-16-0x0 tied to it and this level blocks almost all write access. Processes that are executed from the Anonymous Group typically fall into this level. Levels 1-4 are the levels you mostly read about online, even with MSDN online documentation. The Low-IL is what the AppContainer processes will use when they start. These are the applications built for UWP, or the Universal Windows Platform. Basically, applications that can run universally, like on Xbox and your PC are part of the Universal Windows Platform. You could think of AppContainer as its own IL, but it really boils down to Low-IL. Most objects cannot be written to at Low-IL, like Registry Keys. Now, when you have UAC enabled, the default should be set to Medium-IL, which gives you a little bit more freedom. As a matter of fact, most processes will be launched at this level. At the High-IL is where a process with Administrative rights will show the prompt for the UAC consent box. This could be thought of as an elevated process. The highest integrity level is System-IL where the system services like lsass operate. Many antivirus products have components that run at this IL as well. Level 5, the Protected level, is not enabled by default and is rarely used. The Protected level can only be set from callers in kernel-mode.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Integrity Levels (2)



Integrity Levels (2)

The screenshot on the slide is from Process Explorer, which, as you can see, is running with High-IL. The reason you see this is because I right-clicked on the executable image icon, selected “Run as Administrator” and gave consent to the UAC prompt. You can see other processes and what their corresponding integrity levels are, like explorer.exe and winlogon.exe.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

whoami /priv: non-admin (1)

```
Privilege Name                Description                          State
============================= ==================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled
```


whoami /priv: non-admin (1)

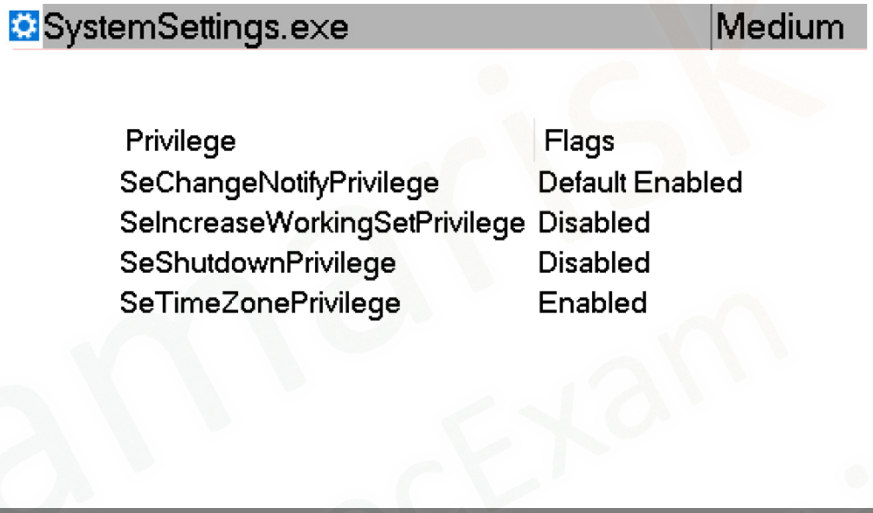
As mentioned previously, privileges are tied to your primary token. The privileges shown on the slide are from the standard user account on the Windows Dev VM. After issuing the *whoami /priv* command, any privileges that are marked as Enabled will be listed in the command's output. The only privilege that is enabled at the moment is *SeChangeNotify*, which, as the name suggests, allows you to traverse different directories to get to files or subdirectories that you can access. The privilege does not give you the access to list contents of every directory you navigate. This privilege should be present and enabled by default for every account. The remainder of the privileges are disabled, but that does not mean they cannot be enabled when you need to perform some operation that the privilege allows. Say, for example, you needed to shut down the system and you already have the privilege to do that action, so it would become enabled. Let us look at another example for changing the system time or the time zone.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

whoami /priv: non-admin (2)



whoami /priv: non-admin (2)

While going through something as simple as changing the time zone, the *SeTimeZonePrivilege* has changed from Disabled to Enabled, as shown on the slide. This was done without any prompts showing up before, during, or after the modification. Another item to note is that the SystemSettings.exe is running with a Medium-IL. There was no need for it to prompt the user for elevation to change the time zone.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

whoami /priv: High Integrity

```
Privilege Name                            Description                                                          State
========================================= ==================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                   Disabled
SeSecurityPrivilege                       Manage auditing and security log                                     Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                             Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                       Disabled
SeSystemProfilePrivilege                  Profile system performance                                           Disabled
SeSystemtimePrivilege                     Change the system time                                               Disabled
SeProfileSingleProcessPrivilege           Profile single process                                               Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                         Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                    Disabled
SeBackupPrivilege                         Back up files and directories                                        Disabled
SeRestorePrivilege                        Restore files and directories                                        Disabled
SeShutdownPrivilege                       Shut down the system                                                 Disabled
SeDebugPrivilege                          Debug programs                                                       Disabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                   Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                             Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                  Disabled
SeUndockPrivilege                         Remove computer from docking station                                 Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                     Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                            Enabled
SeCreateGlobalPrivilege                   Create global objects                                                Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                       Disabled
SeTimeZonePrivilege                       Change the time zone                                                 Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                                Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled
```

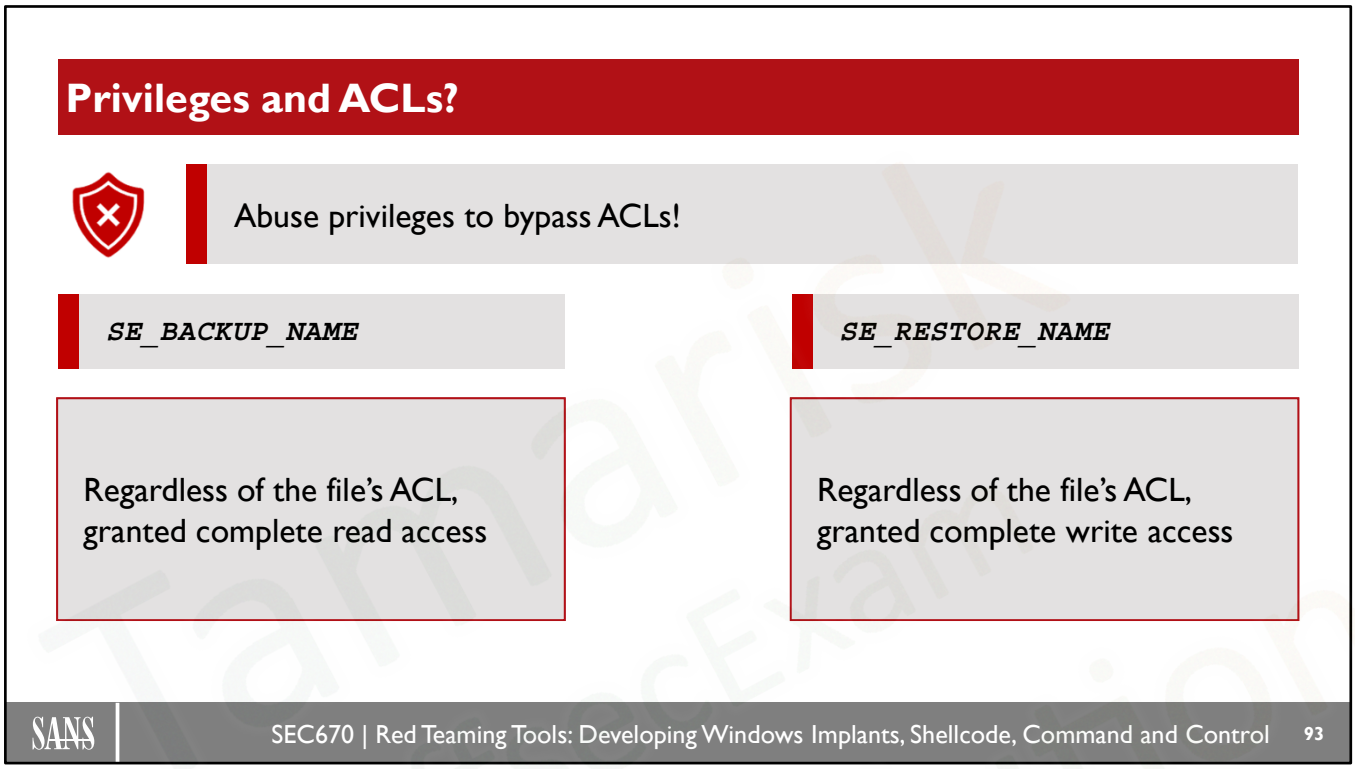

whoami /priv: High Integrity

Let us look at a process with the High-IL. As can be seen on the slide, a High-IL process, command prompt in this instance, has a large increase in the number of privileges that are present. As we have seen already, even though most of the privileges are disabled, they can be enabled on the fly on an as needed basis.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Privileges and ACLs?

There is an interesting tidbit when it comes to privileges. Most privileges allow you to perform some operation, but still only after the system does a privilege check. Well, there are two privileges that bypass that check: *SeBackupPrivilege* and *SeRestorePrivilege*. MSDN describes these two privileges as ones that are used to back up files and directories, and restore files and directories, respectfully. Also, they clearly indicate that with just the *SeBackupPrivilege*, you will be given all the read access you could ever want to the file system without a single check being done. With this privilege, you will have several access rights, including FILE_GENERIC_READ. You might be thinking that while it is nice to read any file, what about being able to write? Well, the *SeRestorePrivilege* has you covered. The restore privilege allows system-wide access to write to any file you would like. Again, it does not matter what the ACL says because this trumps it. As with the backup privilege, this one comes with several access rights, including FILE_GENERIC_WRITE.

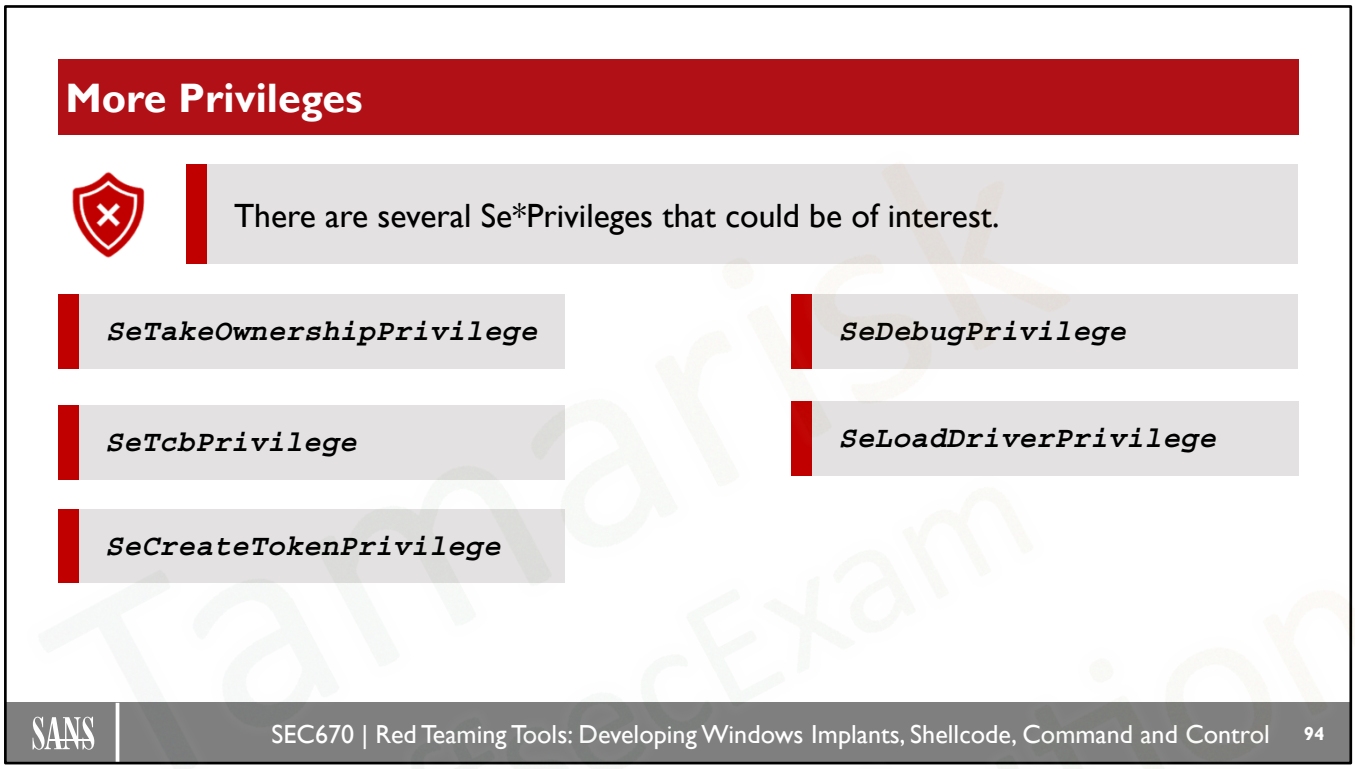
If you would like to explore more privilege constants, you can find them in the winnt.h header file where they are defined.

Reference:
https://docs.microsoft.com/en-us/windows/win32/secauthz/privilege-constants

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



More Privileges

We talked about several privileges up to this point and what they can enable you do, but there are more that would be of interest. The first thing you might be wondering is why none of the privileges noted on the slide are even present for most standard user accounts, and that observation is accurate. The privileges will not be present until you have been able to escalate your privileges. The privileges on the slide can be abused to help you escalate from Admin to SYSTEM.

The first privilege on the slide is *SeTakeOwnershipPrivilege*. This is a powerful privilege because it allows you to take ownership of any securable object you desire. The privilege even allows you to take ownership of securable objects tied to protective processes!

SeTcbPrivilege is described as the Trusted Computing Base (TCB). With this privilege, you become a trusted part of the computer just like how some of the Windows subsystems are.

The *SeCreateTokenPrivilege* allows you to do just what it sounds like it would do—create tokens. You could abuse this to create tokens for arbitrary users on the local system.

We talked about *SeDebugPrivilege* previously, but one item to note is that even with this privilege enabled, you cannot mess around with protected processes.

SeLoadDriverPrivilege does just what it sounds like it would do—load drivers. In the kernel, pretty much everything is trusted, so bringing your own vulnerable driver to the game would greatly boost your efforts.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Privileges: Programmatically



Privileges can be enabled/disabled programmatically.

When you have a set of privileges that are present, but listed as disabled, you can programmatically adjust those privileges to be enabled. The opposite is also true, but why limit yourself?

- `LookupPrivilegeValue`
- `OpenProcessToken`
- `AdjustTokenPrivileges`

Privileges: Programmatically

With the help of the Win32 API, we can create programs that can enable or even disable privileges that are present in our access token. Present means they are listed in the output of the *whoami /priv* command. If you were able to find a UAC bypass to gain an elevated CMD prompt, you would then have many more privileges that would be present, but just not enabled, yet. There are three main APIs involved with this process: ***LookupPrivilegeValue***, ***OpenProcesToken***, and ***AdjustTokenPrivileges***. One common privilege that can be enabled is the *SeDebugPrivilege*. Of course, there are many others that you can select to use, but we will use this one for an example. Let us take a look at the APIs involved and what arguments they will need.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

LookupPrivilegeValue



LookupPrivilegeValue

Gets the current LUID

Has a Boolean return type

```
BOOL LookupPrivilegeValueA(
 _In_opt_ LPCSTR lpSystemName,
 _In_     LPCSTR lpName,
 _Out_    PLUID  lpLuid
);

// EXAMPLE

if ( !LookupPrivilegeValue(...) )
{
  // code here
}
```


LookupPrivilegeValue
Whenever you need to retrieve the locally unique identifier for a privilege constant or privilege name like *SeDebugPrivilege*, this is the function to use. It has a BOOL return type, so it is simple to error check. Simply wrap the function call inside the condition of an if statement and add the code you want to execute inside the body like shown on the slide. Let us take a look at the function's parameters.

lpSystemName, indicates what system to retrieve the Luid for, which indicates that the function should use the local system instead of a remote system. This parameter can optionally be NULL.

lpName, is a pointer to the privilege name as it is defined in the Winnt.h header file. So, you could give it a constant like SE_DEBUG_PRIVILEGE or *SeDebugPrivilege*.

lpLuid, will be a pointer to a variable you use to store what the function finds.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

OpenProcessToken

OpenProcessToken

Obtains a handle to a process' access token

Has a Boolean return type

```
BOOL OpenProcessToken(
 _In_  HANDLE  ProcessHandle,
 _In_  DWORD   DesiredAccess,
 _Out_ PHANDLE TokenHandle
);

// EXAMPLE

if ( !OpenProcessToken(...) )
{
  return FALSE;
}
```


OpenProcessToken

You cannot change any privileges in a token without having a handle to it. The ***OpenProcessToken*** gets you that token handle, when successful, of course. As with the ***LookupPrivilegeValue*** function, it has the same BOOL return type. One example for how to call this function has been noted on the slide. The function arguments have been omitted for space constraints. Speaking of arguments, let us take a look at the parameters.

ProcessHandle, is the handle to a process for which you want a token handle. Be sure that the PROCESS_QUERY_INFORMATION permission is present.

DesiredAccess, is the access mask that you will pass to dictate, or rather ask, what type of access you wish to have for the access token. This is what gets compared with the DACL, the discretionary access control list of the token. This will ultimately be used to approve or deny the requested access type.

TokenHandle, is the pointer to some variable that will hold the handle to this token.

The *TokenHandle* can then be used for the call to ***AdjustTokenPrivileges***.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

AdjustTokenPrivileges



AdjustTokenPrivileges

Enables or disables privileges

Has a Boolean return type

```
BOOL AdjustTokenPrivileges(
 _In_      HANDLE  TokenHandle
 _In_      BOOL    DisableAllPrivileges
 _In_opt_  PTOKEN_PRIVILEGES NewState
 _In_      DWORD   BufferLength
 _Out_opt_ PTOKEN_PRIVILEGES PreviousState
 _Out_opt_ PDWORD  ReturnLength
);

// EXAMPLE

if ( !AdjustTokenPrivileges(...) )
{
  return FALSE;
}
```


AdjustTokenPrivileges
This is the function you can use to enable or disable privileges and is really the last and final step you would have to take if you wanted to enable or even disable a token's privileges. Compared to the previous two functions, ***LookupPrivilegeValue*** and ***OpenProcessToken***, there is a lot more going on with this function. It has a BOOL return type, so we can continue to plug this into an if condition just like the previous two functions. Depending on what you would like to do, it will require a different setup before calling the function. We only care about enabling privileges, so that is the setup of interest.

Let us take a closer look at the function's parameters and types.

TokenHandle, HANDLE, is used for the handle to the access token you obtained by calling ***OpenProcessToken***.

DisableAllPrivileges, BOOL, if set, will disable *all* privileges for the access token. We do not want that, so FALSE it is.

NewState, PTOKEN_PRIVILEGES, is a pointer to a filled-out structure that indicates the Luid and its attributes for the privilege being enabled.

BufferLength, DWORD, should be the size of the TOKEN_PRIVILEGES struct that you made somewhere before this call; sizeof(TOKEN_PRIVILEGES); .

PreviousState, PTOKEN_PRIVILEGES, is a parameter we do not really care about because we are making a new state; (PTOKEN_PRIVILEGES)NULL; .

ReturnLength, PDWORD, is also a parameter we do not care about; (PDWORD)NULL; .

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Stealing Tokens

Source code review!

Stealing Tokens
Time to jump into the source code and explain it before you implement it on your own.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.5: TokenThief



Nothing more fun than being a token thief

Please refer to the eWorkbook for the details of the lab.

Lab 3.5: TokenThief
Please refer to the eWorkbook for the details of the lab.

Tamarisk OffsecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to explore the steps and APIs involved with stealing a token for escalating privileges.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Put Our Service to the Test!



Services and what they can do for your escalation needs

Services are a special kind of process that interact with the SCM. Services do not need a user to login to start as they can be started at boot and run without any user logged on to the system.

- `schedule`
- `EventLog`
- `gupdate`
- `iphlpsvc`
- `BITS`

Put Our Service to the Test!
There are several services that execute behind the scenes, even when there is no user logged on to the system. Services can be configured to start with the system at boot, shortly after boot, when a user logs on, manual start, etc. Regardless of the start trigger, all services must answer to the Service Control Manager (SCM). The installed drivers and services are in a database that is maintained by the SCM. The SCM database can be found under the *HKLM\SYSTEM\CurrentControlSet\Services* registry key with each subkey being the name of the service. Because the SCM acts as a Remote Procedure Call (RPC) server, services can be configured and managed remotely. Depending on what applications your system has installed, you could see some of the services listed on the slide: schedule for the Task Scheduler, EventLog for the Windows Event Log, gupdate for Google Chrome's update service, iphlpsvc for the IP Helper net service, and BITS for the Background Intelligent Transfer Service.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Attributes



Each service will come with some attributes.

Start Type	Service Type	Error Level
Indicates if the service is auto-start, on-demand, or disabled	Indicates if the service runs in its own process or share one with other services	Indicates the severity level if the service or driver fails to start, and what action to take

Services: Attributes

There are certain pieces of information that are good to know about services that I just call service attributes. The service's type is used to indicate to the system, and SCM, if the service is going to execute in its own process, or if it will share a process with some other services. The service's start type indicates how the service, or driver, will begin its execution. Auto-start means the service, or driver, will kick off automatically when the system boots. On-demand means that a user must manually start it; however, if an auto-start service depends on some on-demand service, then the SCM will happily start the on-demand service. Disabled means the service, or driver, cannot start at all. The error control level is used to indicate how bad things will be if a service, or driver, fails to start when it should. Perhaps the system will not boot properly if Driver A does not start before Driver B starts. With this, a predefined action can be specified, which can simply be to attempt to start the service again. Another attribute is the absolute or fully qualified path to the service/driver. Services should have the EXE extension and drivers should have the SYS extension.

The SCM does a few housekeeping steps when it starts a service:

1. Search the SCM database for the account information.
2. Make sure the service account is logged on. If not, log on.
3. Load the user profile.
4. Like process hollowing, the service is started, but in a suspended state.
5. Give the process its logon token and resume execution.

The service should now be ready to go on the system.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Handles



Interacting with the SCM requires handles

There are several handles to objects that are required to be obtained when you want to interact with, modify, delete, or create a new service.

`SCManager` `Service` `Database lock`

Services: Handles

The SCM database can be queried, modified, etc. once you obtain a handle to it. What you really have is a handle to the SCManager object and within that container object are the service objects. To request such a handle, you must call the ***OpenSCManager*** API. With the returned handle, you can then use that to enumerate services, delete services, install services, open services, or lock the services database lock. For those coming from red team or penetration testing team background, enumerating services might be something you do when looking for some low hanging fruit for privilege escalation. Searching for services with unquoted paths could be one such method that boosts you up to SYSTEM as long as the service is running as SYSTEM, which some of them do. Maybe you ran a command like the one below before to find those unquoted service paths. While it is nice to be able to run a Windows command to do this, doing this programmatically would be better.

wmic service get name,pathname,displayname,startmode | findstr /i auto | findstr /i /v "C:\Windows\\" | findstr /i /v ""”

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Enumeration



You cannot find what you do not look for.

Enumerating services is just another part of on target recon with hopes of finding something to exploit. Most tools that deal with Windows services enumerate and query them to show the operator some potential LPE vectors.

EnumServicesStatus

QueryServiceStatus

Services: Enumeration

LPE via services has been a great success for red teamers. Regardless of the tool you may have used in the past, it probably did some form of enumeration of services. To do this, you could manually parse the subkeys from the services registry key and then manually query each one, but a great way is to use the service-specific Win32 APIs that are available, like ***EnumServicesStatus*** or ***EnumServicesStatusEx***. One of the items to look for after enumerating the services would be an unquoted path.

As a refresher, services with an unquoted service path means you are looking for spaces to be present in the absolute path along with missing quotation marks around that path, so something like this: *C:\Users\student\SEC 670\Labs\Day 3\Unquoted Service Paths*. That example path has spaces in it and there are no quotations marks around it. When the system goes to find the EXE, it will stop at the first space that it comes across and use that path to start its search. Sticking with that example path it would become *C:\Users\student\SEC.exe*. Most likely, that EXE would not exist on the system, so it would have to keep searching: *C:\Users\student\SEC 670\Labs\Day.exe.* The system would keep searching until the EXE is found. The idea is to give it something to find early in the search process to force your EXE to run.

You could also take advantage of services with weak permissions. As a reminder, regular users should not be able to modify the configuration information of services. Local administrators should be the only ones with those permissions enabled. Say for an example there was a service with a weak configuration or incorrect permissions—the service might be pointing to some directory where you have write access. The write access could let you drop a DLL to that location and gain execution there. One real world example was the CVE-2019-1322 notice with the UsoSvc service. The service's binary path could be modified to point to your malicious executable, like a msfvenom payload or a custom one you developed.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

EnumServicesStatusEx



EnumServicesStatus

Enumerates services in SCM database

Has a Boolean return type

```
BOOL EnumServicesStatusExA(
 _In_  SC_HANDLE      hSCManager,
 _In_  SC_ENUM_TYPE   InfoLevel,
 _In_  DWORD          dwServiceType,
 _In_  DWORD          dwServiceState,
 _Out_opt_ LPBYTE     lpServices,
 _In_  DWORD          cbBufSize,
 _Out_ LPDWORD        pcbBytesNeeded,
 _Out_ LPDWORD        ServicesReturned,
 _Inout_opt_ LPDWORD  ResumeHandle,
 _In_opt_    LPCSTR   pszGroupName
);
```


EnumServicesStatusEx

Before you can query the status of a service, you must first find a service to query, which can be done using the ***EnumServicesStatusEx*** API. All the services in the SCM database will be enumerated with this API. Let us take a look at the parameters for it.

hSCManager is the SC_HANDLE returned from the ***OpenService*** or ***CreateService*** APIs. The handle should at least have the SC_MANAGER_ENUMERATE_SERVICE access mask.

InfoLevel has only one option, SC_ENUM_PROCESS_INFO.

dwServiceType is a filter for a certain service type. Typically, you would pass in SERVICE_WIN32 here to indicate that you want all service types.

dwServiceState gives you the option to enable finer filtering by not only specifying the service type but also the current state of the service. Typically, you would want to pass in SERVICE_STATE_ALL.

lpServices is the buffer that will wind up holding this information.

cbBufSize is the size of the buffer in bytes. Most times you might not know how many bytes to pass here, but we can find that out. If you really wanted to save another function call, just use 256KB as the size since the buffer cannot be larger than that.

pcbBytesNeeded will let you know if the buffer you passed in is too small to hold everything. If so, the API will fail. Then you can call the API again with the adjust value.

lpServicesReturned is a pointer to a variable that will be used to store how many services were returned.

lpResumeHandle is optional, but it could be used if there is a need to make multiple calls since not all of the information might be given after the first call.

© 2024 Jonathan Reiter

© SANS Institute 2024
https://linktr.ee/offsecexam

pszGroupName could be used to filter the services according to their group name. NULL here means ignore any group a service is a part of and enumerate all groups.

Tamarisk
OffsecExam
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

QueryServiceStatusEx



QueryServiceStatus

Obtains the status of a service

Has a Boolean return type

```
BOOL QueryServiceStatusEx(
 _In_   SC_HANDLE      hService,
 _In_   SC_STATUS_TYPE InfoLevel,
 _Out_opt_ LPBYTE      lpBuffer,
 _In_   DWORD          cbBufSize,
 _Out_  LPDWORD        pcbBytesNeeded
);
```


QueryServiceStatusEx

After you have obtained a list of the services installed on your target, you would most likely want to query them for more detailed information. To do that, you would call the ***QueryServiceStatusEx*** API. Let us take a look at the five parameters.

hService is the SC_HANDLE from ***OpenService*** or ***CreateService*** APIs. Regardless of the chosen API, the handle must have SERVICE_QUERY_STATUS at a minimum.

InfoLevel indicates the service attributes that you would like to know. The buffer you pass in for *lpBuffer* will be filled out with this information. The only value supported here is SC_STATUS_PROCESS_INFO. Maybe someday in the future this will change.

lpBuffer is a pointer to a buffer that will store the status information of the service.

cbBufSize is the size of the buffer that you are pointing to via *lpBuffer*. This is in bytes too, by the way.

pcbBytesNeeded is a pointer to a variable that will store the number of bytes needed to hold the status information of the service. This is useful when you do not know how many bytes you need to allocate for your *lpBuffer*.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

QueryServiceConfig



QueryServiceConfig

Obtains configuration of a service

Has a Boolean return type

```
BOOL QueryServiceConfigA(
 _In_   SC_HANDLE hService,
 _Out_opt_ LPQUERY_SERVICE_CONFIGA pSvcCfg,
 _In_   DWORD     cbBufSize,
 _Out_  LPDWORD   pcbBytesNeeded
);
```


QueryServiceConfig

The ***QueryServiceConfig*** API will obtain the configuration parameters for whatever service you give it. There is also another version of this API named ***QueryServiceConfig2*** that will obtain the configuration parameters that are optional. This API only has four parameters, so let us dive into them.

hService is the SC_HANDLE from ***OpenService*** or ***CreateService*** APIs. Regardless of the chosen API, the handle must have SERVICE_QUERY_STATUS at a minimum.

lpServiceConfig, *pSvcCfg* on the slide for brevity, is a pointer to the buffer that will be used to store the configuration information about the service. As can be seen by the type, this will be in the format of the QUERY_SERVICE_CONFIG structure, which is made up of members like *dwServiceType*, *dwStartType*, *dwErrorControl*, *lpBinaryPathName*, etc. The binary path name could be of interest if the path has spaces and is missing quotes around it.

cbBufSize, just like for the other APIs mentioned thus far, is the size of the buffer that is pointed to by the *lpServiceConfig* parameter, in bytes, of course. The buffer need not be larger than 8KB as this is the max size.

pcbBytesNeeded: If the API would ever fail with ERROR_INSUFFICIENT_BUFFER, this pointer to a variable will end up holding the number of bytes needed so that the buffer pointed to by the *lpServiceConfig* parameter could be sized accordingly for a secondary call.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

ChangeServiceConfig



ChangeServiceConfig

Modifies a service's configuration

Has a Boolean return type

```
BOOL ChangeServiceConfigA(
 _In_     SC_HANDLE hService,
 _In_     DWORD     dwServiceType,
 _In_     DWORD     dwStartType,
 _In_     DWORD     dwErrorControl,
 _In_opt_  LPCSTR   lpBinaryPathName,
 _In_opt_  LPCSTR   lpLoadOrderGroup,
 _Out_opt_ LPDWORD  lpdwTagId,
 _In_opt_  LPCSTR   lpDependencies,
 _In_opt_  LPCSTR   lpServiceStartName,
 _In_opt_  LPCSTR   lpPassword,
 _In_opt_  LPCSTR   lpDisplayName
);
```


ChangeServiceConfig

The ***ChangeServiceConfig*** API is very similar to the ***CreateService*** API that will be introduced later. The purpose of this API is to make a change to the configuration of an already existing service. This function can be used to change almost any configuration except the service's name. After you have enumerated all the services and decided what service you would like to modify, this is the function you use to get it done. Perhaps one example could be that you found a service that points to a writable location, and you change the binary path for it; a simple way to gain execution. With this API you can keep everything the same and only change the *lpBinaryPathName*. The parameters are, for the most part, the same as the ***CreateService*** API. The major differences here are that if you are not going to make a change to one of the parameters, you either pass in SERVICE_NO_CHANGE or NULL. The SERVICE_NO_CHANGE can only be used for the first three parameters and then NULL must be used for the remainder.

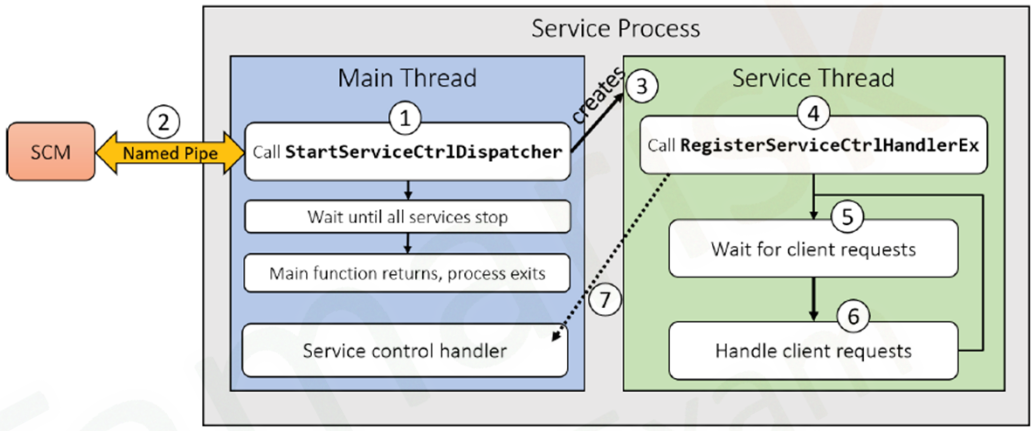
There is also the ***ChangeServiceConfig2*** API, which allows you to change a service's trigger. Triggers are an interesting concept because they can be used to start or stop your service. An example trigger could be when an IP address is first pulled to a NIC for availability or when the NIC loses its availability. Another trigger for a service could be when the computer joins an active directory domain. Your malicious service could also have a trigger, which is something that is up to you since you are the developer. You just have to do what makes sense and what would meet the requirements for your end customer—say, a red team operator.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Creation (1)



Services: Creation (1)
For us to programmatically create and eventually install a service, there are a few requirements your code must meet.

At a high level, it must have a service entry point, a service ***ServiceMain*** function, and a service control handler function. Our service will not be sharing its address space with other services, so it will be of type SERVICE_WIN32_OWN_PROCESS. The ***ServiceMain*** function is the main function you might already know. If not, the ***ServiceMain*** function has several important items to carry out when it is called by the service control dispatcher. One item is that it must call ***StartServiceControlDispatcher*** so that we can be connected to the SCM via a named pipe. The next item we must take care of is call the ***RegisterServiceCtrlHandler*** function so that it can register a Handler function that will be called when control requests come to the service. Some of the control requests that could come into the service are start, stop, pause, etc. Finally, we must call the ***SetServiceStatus*** function to report to the SCM that we are running, SERVICE_RUNNING. The Handler function is required because it is the function that is called by the control dispatcher when the service receives a control request. The function given is executed in the context of the control dispatcher.

Reference:
Pavel Yosifovich. *Windows 10 System Programming, Part 2*. Pavel Yosifovich, 2022.

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Creation (2)



Small code snippet for a service's main function

```
VOID WINAPI EvilMain(...);

INT main() {
 CHAR ServiceName[] = "notEvil";
 SERVICE_TABLE_ENTRY table[] = {
  { ServiceName, EvilMain },
  { NULL, NULL }                          // the end of the array
 };
 if (!StartServiceCtrDispatcher(table)) { return 1;}
}
```


Services: Creation (2)

As mentioned previously, the ***main*** function has the responsibility of calling the ***StartServiceCtrlDispatcher*** function. If other services are going to share this process' address space, the table is passed as an argument to the function, so SCM knows about them. The table array holds SERVICE_TABLE_ENTRY structs that have two members: the service name (LPSTR) and a pointer to the service's main function (LPSERVICE_MAIN_FUNCTION). The *lpServiceName* struct member is the name of the service that you would see listed in Task Manager or in the registry. The *lpServiceProc* is the pointer to the ***ServiceMain*** function that has a very specific signature, or prototype.

VOID LpServiceNameFunction(
DWORD dwNumServicesArgs, // the number of arguments that are in the lpServiceArgVectors array
LPSTR lpServiceArgVectors // null-terminated strings that are passed into the service via the **StartService** function call when the service was started.*
);

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Creation (3)



Code snippet for ServiceMain()

```
SERVICE_STATUS g_ServiceStatus;
SERVICE_STATUS_HANDLE g_ServiceStatusHandle;
HANDLE g_ServiceStopEventHandle = NULL;

EvilMain(...){
 g_ServiceStatusHandle = RegisterServiceCtrlHandlerEx(...);
 g_ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
 g_ServiceStatus.dwServiceSpecificExitCode = 0;
 g_ServiceStatus.dwWaitHint = 500;
}
```


Services: Creation (3)
The ***ServiceMain*** function, ***EvilMain*** in our case, is where most of the service initialization takes place. There are some global variables near the top of the slide that are filled out and set like the g_ServiceStatusHandle and the g_ServiceStatus. Below is the function prototype for the ***RegisterServiceCtrlHandlerExA*** function.

SERVICE_STATUS_HANDLE
WINAPI
RegisterServiceCtrlHandlerExA(
  LPCSTR lpServiceName, // the name of the service that is executed by the calling thread and the name passed to ***CreateService***
  LPHANDLER_FUNCTION_EX lpHandlerProc, // the handler function to be registered with the SCM. Below is the signature for the handler.
  LPVOID lpContext // simply a pointer to some user-defined data that could be used to identify the service if it is running in a shared process with other services
);

This is the signature for a handler function:

DWORD
WINAPI
HandlerEx(
  DWORD dwControl,
  DWORD dwEventType,
  LPVOID lpEventData,
  LPVOID lpContext
);

© SANS Institute 2024

https://linktr.ee/offsecexam

If you recall, the service type of SERVICE_WIN32_OWN_PROCESS indicates that we do not intend to be part of a shared process. This is particularly important because if your malicious service is going to be shared in a process with other services, what happens when another service crashes? Your service will crash right along with it, thus it is best to be in your own process. As a fail-safe, you could configure your service failure action to simply restart the service and be just fine.

Tamarisk
OffsecExam
Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services: Installation



sc.exe is a common tool for service installation.

It has been common for users to use the service-related PowerShell cmdlets or the sc.exe built-in utility for installing and querying services. Regardless of the tool being used, the underlying API for installing a service is the *CreateService()* API.

Services: Installation

A very common tool for installing services is the sc.exe (service create) built-in utility. The name passed to this should also be the same name passed in your code's call to ***RegsiterServiceCtrlHandlerEx***. For example, if your service is going to be named *notevil*, then at the command line you would use *sc screate notevil binPath= "c:\path\to\your\notevil.exe"*. Later, after successful installation, you can query your service using *sc query notevil*. The sc.exe utility, along with PowerShell cmdlets, uses Windows APIs under the hood to perform their operations. The installing a service capability of those tools utilize APIs like the ***CreateService*** API, which has whopping 13 arguments! It is a powerful function and a complicated one behind the scenes. Thankfully, there are no structures that must be set up and passed into it. We will go into more details about that function next. In the meantime, it is important to know the difference between the code for your actual service, and the service control program, or the service installer program. Typically, your program will not install itself as a service, although it can. It would be best for your implant to install a separate binary as a service; one that could be used for persistence, even. Metasploit's Meterpreter's persistent service is a good example and is freely available for you to see on GitHub.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

CreateService



CreateService

Creates service object

Has a SC_HANDLE return type

```
SC_HANDLE CreateServiceA(
  _In_      SC_HANDLE hSCManager,
  _In_      LPCSTR    lpServiceName,
  _In_opt_  LPCSTR    lpDisplayName,
  _In_      DWORD     dwDesiredAccess,
  _In_      DWORD     dwServiceType,
  _In_      DWORD     dwStartType,
  _In_      DWORD     dwErrorControl,
  _In_opt_  LPCSTR    lpBinaryPathName,
  _In_opt_  LPCSTR    lpLoadOrderGroup,
  _Out_opt_ LPDWORD   lpdwTagId,
  _In_opt_  LPCSTR    lpDependencies,
  _In_opt_  LPCSTR    lpServiceStartName,
  _In_opt_  LPCSTR    lpPassword
);
```


CreateService

MSDN describes the ***CreateService*** API as one that will create a service object when called. It will also add the service to the SCM database, which is most likely the default database for the local system. Let us tackle some of these parameters and describe what they do.

First up is *hSCManager* of type SC_HANDLE, which indicates that it needs a handle—one you would have obtained from calling the ***OpenSCManager*** API. The ***OpenSCManager*** API accepts three arguments, the name of the machine to connect to, the name of the SCM database to connect, and finally, the desired access to the database.

lpServiceName, is the unique name you would see in the registry. Here, you are limited to 256 characters.

lpDisplayName is what is known as the friendly name and is what you would normally see in the Services tab in the Task Manager. This string is also limited to 256 characters.

dwDesiredAccess is the access the returned service handle will require. So, to start a service by calling the ***StartService*** API, you would need to specify the SERVICE_START mask.

dwServiceType is the type of service that will be created. For our purposes this will be WIN32_OWN_PROCESS (0x10).

dwStartType indicates when the service should be started. Services can have AUTO_START (2), DEMAND_START (3), or DISABLED (4).

dwErrorControl dictates what is to be done when service initialization fails. It can be IGNORE (0), NORMAL (1), SEVERE (2), or CRITICAL (3).

© 2024 Jonathan Reiter

© SANS Institute 2024

lpBinaryPathName must be the full path where the executable is located. Command-line arguments can also be passed in here after the executable's name.

lpLoadOrderGroup is optional, so NULL is just fine here.

lpdwTagId is only for kernel drivers and as this is not a kernel class, we do not need to worry about this one.

lpDependencies is an optional list of strings naming other services that this service depends on for successful initialization.

lpServiceStartName is the account that this service should execute under.

lpPassword would be for the password to the given user account.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Pipes!



One of many method of interprocess communications (IPC)

Application data sharing and communications, together, are known as IPC. Pipes are one such mechanism of IPC. They allow for two-way communications either with parent-child processes, or processes with no relationship.

Anonymous pipes

Named pipes

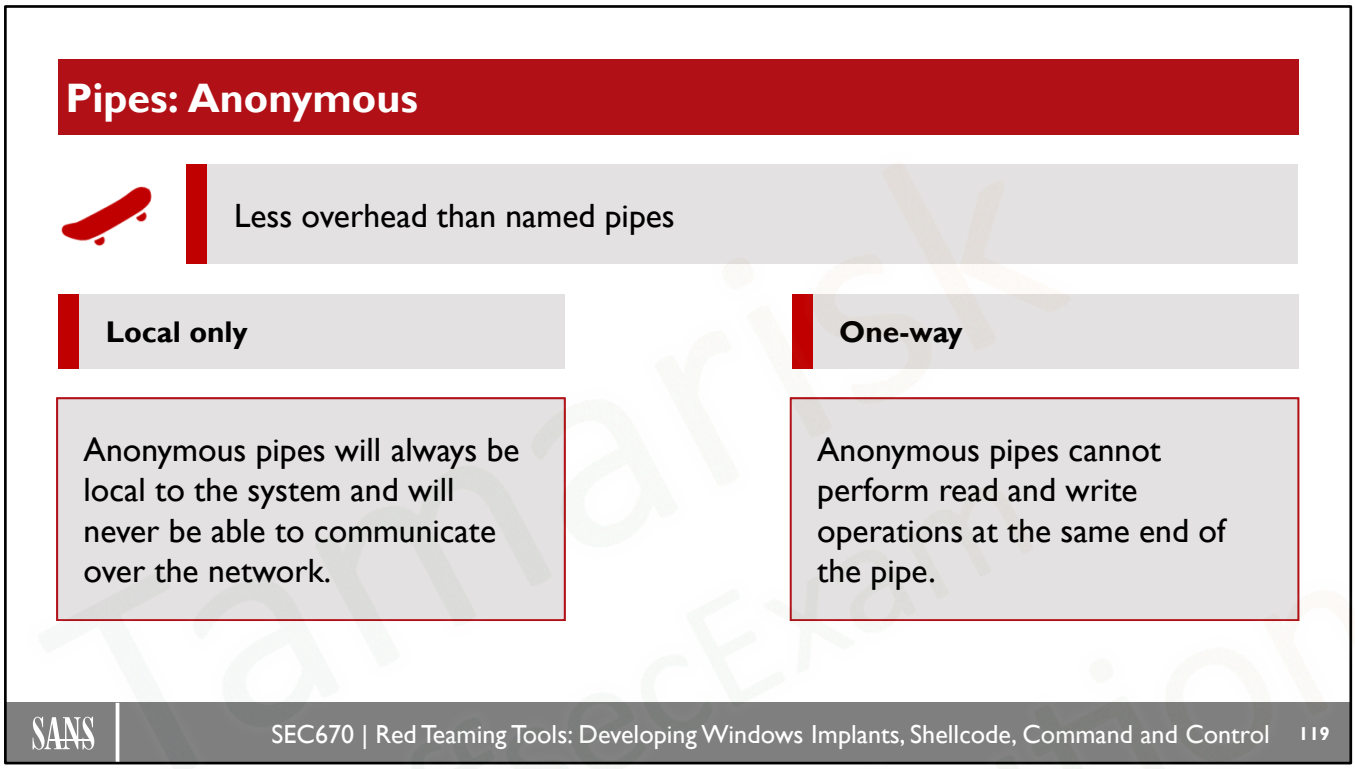
Pipes!

There are times when processes must communicate with each other or share some data. Windows provides a mechanism that allows the communication and sharing to take place. Such a mechanism is called interprocess communications, or IPC for short. Even if you have not heard of IPC before now, you have definitely used it at least once if you have ever cut, copy, or pasted anything. The clipboard could arguably be one of the most commonly used IPC mechanisms. Some of the other mechanisms are COM, the Component Object Model, which is the backbone to how Excel data can be embedded in Word documents. Another mechanism is file mapping, which has many uses! Windows Sockets are yet another example of IPC. The IPC mechanism we are focusing on at the moment is pipes. Pipes can allow communication to flow in both directions called duplex operations. Think of a conduit—a pipe that an electrician might use to run wires through. Windows pipes are very similar: they have two ends, and data can flow through them. The end points of the pipes are where the reading and writing of data takes place. A duplex pipe allows the read and write operations to take place at the same end of the pipe. A one-way pipe is where a write operation takes place at one end and the read operation takes place at the other end. Pipes come in two flavors: anonymous and named.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Pipes: Anonymous

Anonymous pipes are pipes without names, which have certain restrictions that named pipes do not have. For instance, they cannot be used to communicate over the network, and they cannot be used to communicate with processes that are not related to them (parent/child). Anonymous pipes come to play when you need a quick IPC mechanism between your process and a child process. As mentioned previously, your child process could take commands and also send the output of commands back to the parent process. This flow can happen over anonymous pipes with the redirection of standard in and standard out. When a process creates a pipe, that process is the pipe server. Pipe clients would then connect to it and reading and writing to the pipes can take place.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Creating Anonymous Pipes



`CreatePipe`

Used to create anonymous pipe

Has a Boolean return type

```
BOOL
CreatePipe(
 PHANDLE hReadPipe,
 PHANDLE hWritePipe,
 LPSECURITY_ATTRIBUTES lpSecAttr
 DWORD nSize
);
```

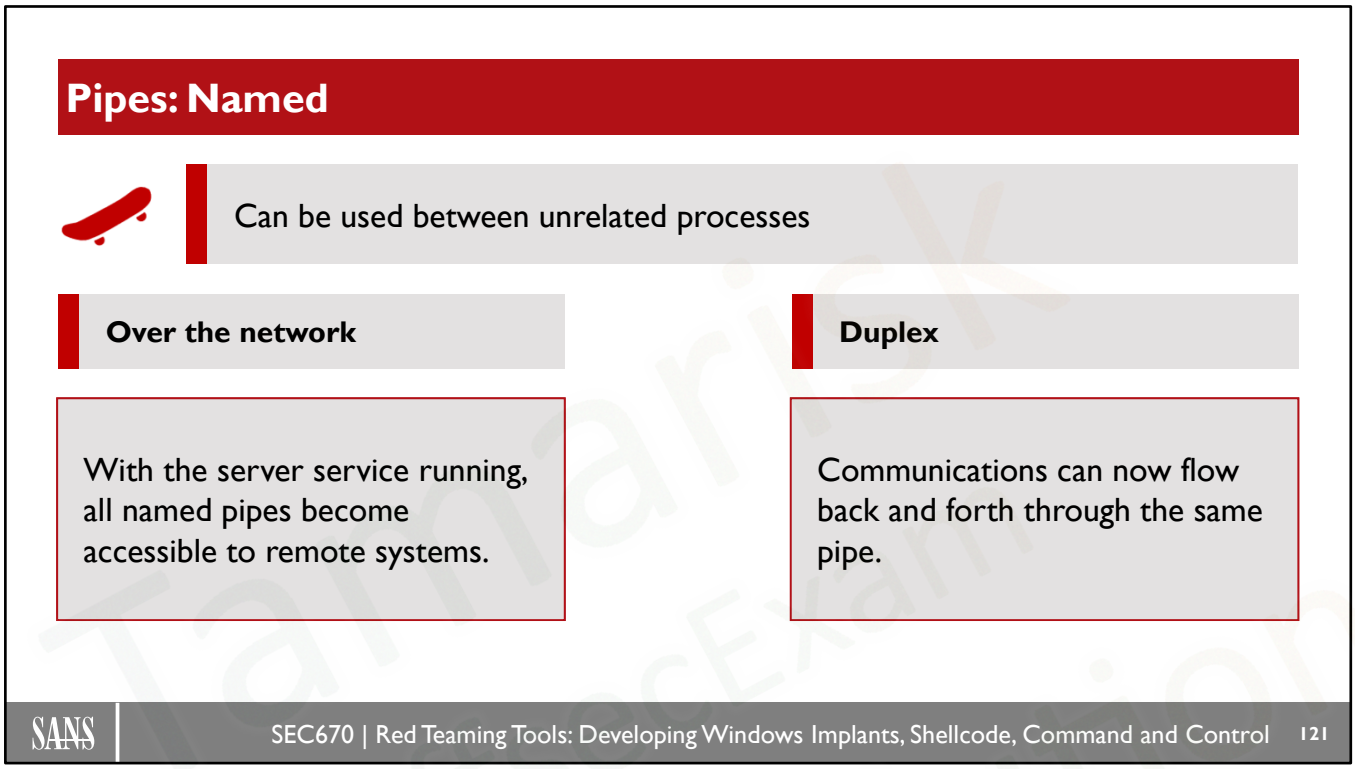

Creating Anonymous Pipes

Anonymous pipes are created using the ***CreatePipe*** function. The function will return handles to the read end of the pipe as well as the write end of the pipe. The handle for the read end of the pipe will naturally only have read access. The same thing goes for the write end of the pipe—it will only have write access. The process calling this function is the pipe server and clients will connect to it. To do so, they will need to be allowed to access the proper handle, which is easily done using handle inheritance. The child process, or the client, can either be sent the handle for the read or write ends of the pipe. If the pipe server sends the read handle, the client would have to use the ***ReadFile*** function. The ***ReadFile*** function will return as soon as data has been written to the pipe using the ***WriteFile*** function. Inheriting handles is just one method a process can be given pipe handles. The other method is using a function named ***DuplicateHandle***, which basically makes a copy of a handle that could be shared using shared memory.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Pipes: Named

Named pipes are pipes that have a name associated with them, and there are several advantages that named pipes have over anonymous pipes. Named pipes can be one directional like anonymous pipes, but they can also be duplex pipes. Duplex pipes are great for pipe servers that will communicate with one or many pipe clients. This also means that any process could become the pipe server and the pipe client. Depending on the security attributes you choose for your pipe, any process could be allowed to interact with your named pipe. This is so much more convenient than having to worry about sharing handles to a pipe you created. Perhaps one of the bigger advantages is that named pipes can be used to communicate with remote processes. Should you have the server service running, then your named pipes will be remotely accessible.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Creating Named Pipes



`CreateNamedPipe`

Used to create named pipes

Has a HANDLE return type

```
HANDLE
CreateNamedPipe(
 LPCSTR lpName,
 DWORD  dwOpenMode,
 DWORD  dwPipeMode,
 DWORD  nMaxInstances,
 DWORD  nOutBufferSize,
 DWORD  nInBufferSize,
 DWORD  nDefaultTimeOut,
 LPSECURITY_ATTRIBUTES lpSecAttr
);
```


Creating Named Pipes

Named pipes are created using the ***CreateNamedPipe*** function. Instead of returning handles to the ends of pipes, it will create an instance of the pipe. The process that calls ***CreateNamedPipe*** is the pipe server and can use the function to create the first instance of the pipe or, with an existing pipe name, it can create another instance. The pipe server can later call the ***ConnectNamedPipe*** function to wait for a pipe client to connect to the pipe instance. The client will use the ***CreateFile*** function or the ***CallNamedPipe*** function to connect. Clients can use the following format when connecting to a named pipe: *ComputerName*\pipe*PipeName*. The *ComputerName* is going to be the name of the computer you wish to connect. Of course, this can also be the local computer. The *PipeName* is the unique name for the pipe and the name can contain almost any character except a backslash.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Creating Pipes

Source code review!

Creating Pipes
Time to jump into the source code and explain it before you implement it on your own.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

UAC Me, Now You Don't



UAC is not a security boundary.

Processes that could lead to a system compromise typically will run with a Low-IL. Browsers often do this in case you browse to a malicious site.

Processes that typically run with High-IL are ones that have system-wide configurations or operations like Lsass.exe and Winlogon.exe.

UAC Me, Now You Don't

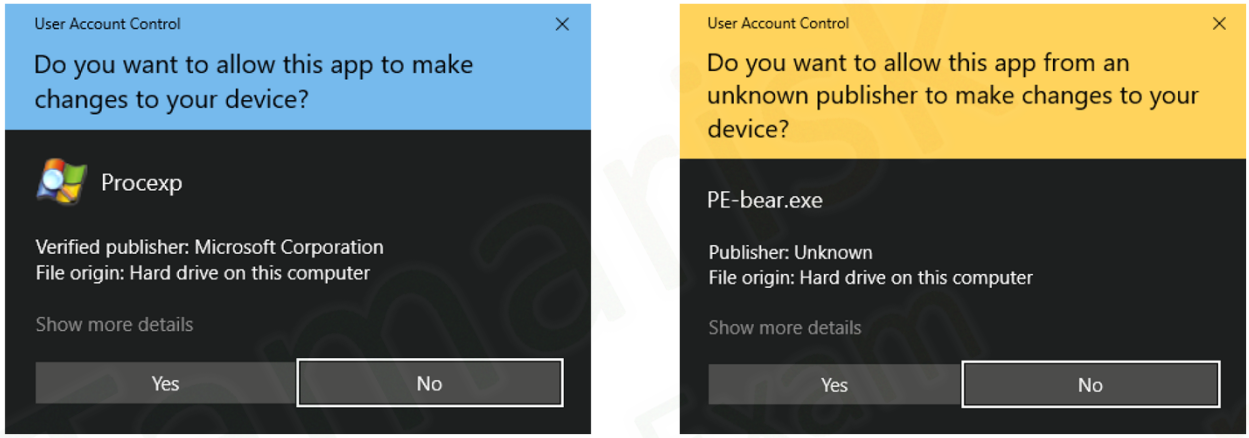
Many people interpret UAC as a security boundary, thinking that UAC is a mechanism that is protecting them. UAC, to me, is really just an annoying method of protecting you from yourself but definitely not a true security boundary. There are a couple of positive items about UAC, so it is not all that bad. UAC gives you the convenience of running an application as Admin without having to switch user accounts. Another convenience is that you no longer have to issue the RunAs command if you want to execute a command as a different user. Perhaps one final positive note about UAC is that it can possibly mitigate how effective malware is on a system. There are a number of UAC bypass projects out there and it is possible to find the resources to bypass UAC, as we will see later.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

UAC: Elevation Prompts



UAC: Elevation Prompts

It was mentioned before that when you are performing operations as a standard user, you typically run with Medium-IL. When you need to elevate a process because you want it to have more privileges, then you most likely will see a UAC prompt or consent pop-up. The example on this slide is from when I right-clicked on the Process Explorer icon and selected ”Run as Administrator.” The UAC prompt with the blue title bar is an indication that the application is trusted and signed by Microsoft. The yellow UAC title bar prompt indicates that a process’ publisher could not be verified or was not signed. The user should proceed with caution when a yellow UAC prompt presents itself. Since this was PE-bear.exe, I trust the application to do no harm to the system. One other title bar color you might see is red. A red UAC elevation prompt indicates that the application you are trying to run has a publisher that has been blocked. A block in Group Policy could also trigger a red UAC prompt. The user should be able to click through with confidence knowing that nothing malicious is about to happen to their system.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

UAC: Fusion



Applications and their manifests

Many applications have a manifest file tied to it that is used to describe the application itself. This XML file contains detailed information about the application's security context. There are several elements in the manifest.

- `supportedOS`
- `autoElevate`
- `heapType`

UAC: Fusion

When a process is being created, there are several checks that the ***CreateProcess*** API performs. One of them is calling into the system's Fusion database where information from an application's manifest file is stored. .NET applications make heavy use of manifests, but traditional Win32 applications can also use them. The manifest file is simply an XML file that can be embedded in the EXE as a resource. Including manifests as a resource is the Microsoft recommended method. The manifests have many case-sensitive elements in them and there are a few that can directly apply to us when it comes to elevating privileges or simply gathering information.

When developing applications, there is a choice to make as to what OS your program will support. The *supportedOS* element has an Id attribute that can indicate what OS is supported, and each OS version has a GUID tied to it. The *heapType* element is used to override whatever the default heap implementation is. For example, specifying *SegmentHeap* for the program will make it use the segmented heap, which tends to lower the amount of memory your application uses. The *autoElevate* is the most interesting to us as it indicates that the program can automatically elevate its privileges without prompting the user for consent via the UAC consent prompt. When the *autoElevate* element is set to TRUE, it means it is enabled and there will be no elevation prompt. Finding applications with *autoElevate* set to TRUE could give you a means to bypass UAC and escalate your privileges.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

UACMe Project



GitHub repo hosts many UAC bypass methods

Inside the repo are a few files that handle parsing the manifest files, and fusion.c is one such file. The main idea is to find files that have embedded manifests to parse and checking to see what the autoElevate element value is. There are roughly three key functions.

`FusionScanDirectory` `FusionScanFiles` `FusionCheckFile`

UACMe Project

One part of the process of finding UAC bypasses is to search an application's embedded manifest for the *autoElevate* field. The applications that have this value set to TRUE would be of interest because they could be part of a step in elevating privileges. Instead of recreating the wheel and making your own parser, UACMe by hfiref0x has done this already. The project is hosted on GitHub and the entire project demonstrates several bypass methods. Of particular interest, though, are the code portions that deal with parsing the embedded manifests. If you look under Source\Yuubari\fusion.c, you will find the logic for that parsing. The header file fusion.h holds the definitions for several custom structures, as well as function signatures. In the fusion.c source file, it appears there are three functions that directly relate to the parsing: *FusionScanDirectory*, *FusionScanFiles*, and *FusionCheckFile*. To better understand what each function is doing, we should take a deeper look at the code being used.

The UACME project can be found here: https://github.com/hfiref0x/UACME.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

UACMe Project: FusionScanDirectory



Responsible for scanning current directory

This function takes in a directory path and kicks off the file scanning process. Once that is done, it does a directory walk looking for any other items in that folder, like subfolders and application files. There are three functions that help get this done.

- `RtlSecureZeroMemory`
- `FindFirstFile`
- `FindNextFile`

UACMe Project: FusionScanDirectory

This function pretty much does what it sounds like—scans a directory. It will scan a directory for files, like applications, as well as any subfolders. With any subfolders that are found, it will recursively call itself to process any files in the subfolder. This logic is implemented by using functions like ***FindFirstFile*** and ***FindNextFile***. Both functions, when used in combination, enable an application to programmatically enumerate a directory. The ***RtlSecureZeroMemory*** function is used to securely zero out items like arrays and structures. The system guarantees that the block of memory will be zeroed out. For any files that are found, the function ***FusionScanFiles*** is called.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

UACMe Project: FusionScanFiles



Responsible for scanning EXE files

This function takes in a directory path and will look for EXE files in the directory. Any EXE that is found, the function will call FusionCheckFile. The file's information is stored in the process heap.

`HeapAlloc/HeapFree` `FindFirstFile` `FindNextFile`

UACMe Project: FusionScanFiles

The ***FusionScanFiles*** function will scan for any EXE files in the directory path that was passed in as a function argument. Since ***FindFirstFile*** and ***FindNextFile*** return information about a file in the form of the WIN32_FIND_DATA structure, the function allocates some space in the process heap to hold that information. When using the ***HeapAlloc*** function to allocate your block of memory, you should also use the ***HeapFree*** function to indicate the block is no longer being used by your program. The stored WIN32_FIND_DATA structure is passed as an argument to ***FusionCheckFile***.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

UACMe Project: FusionCheckFile



Responsible for parsing embedded manifests

The function takes in the path to the EXE, opens a file handle to it, maps it into memory, and searches the image to see if there is an embedded manifest.

`LdrResSearchResource` `NtCreateFile` `NtCreateSection`

UACMe Project: FusionCheckFile

The ***FusionCheckFile*** function accepts three arguments, but the ones of focus are the path of the EXE and the WIN32_FIND_DATA structure that describes the file. The function uses this information to open the file with read access (***NtCreateFile***) so that it can then map the entire image into memory (***NtCreateSection*** and ***NtMapViewOfSection***). Once mapped into memory, the function can then proceed to find the embedded manifest, if there is one. Applications can have embedded resources like the application manifest, and each one has a type associated with it. The resource type for our manifest is RT_MANIFEST. The ***LdrResSearchResource*** function will look over the mapped image and indicate if the desired resource type was located. If one is found, a flag will be set in the *IsFusion* member of their custom structure, *FusionCommonData*.

After locating the resource, an activation context is created. The context allows certain elements to be queried, like the *autoElevate* element. The *autoElevate* element will have a value of TRUE or FALSE with TRUE indicating that indeed, the application can elevate automatically without an elevate prompt. This information is queried with the assistance of the **RtlQueryActivationContextApplicationSettings** function. Now, with the foundation established, we can look at some source code.

If you're interested in other resource types, check out this URL: https://docs.microsoft.com/en-us/windows/win32/menurc/resource-types.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Covered many ways to escalate your privileges
- Discussed pipes, services, tokens
- Discussed finding bypasses for UAC

Module Summary

In this module, we covered several methods for how you could programmatically escalate your privileges. Learning how Meterpreter's *getsytem* command works on the back end can lend a hand when creating your own version of it or making a modification to it so that it would work in your target environment. The more you understand basic services, privileges, tokens, etc., the better equipped you will be for creating your own novel local privilege escalation (LPE) technique. Perhaps you might even discover and create a remote method as well. In the end, persistence pays off.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What type of pipe can operate over a network?

A Half pipe

B Named pipe

C Anonymous pipe

Unit Review Questions
Q: What type of pipe can operate over a network?

A: Half pipe

B: Named pipe

C: Anonymous pipe

Tamarisk OffsecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What type of pipe can operate over a network?

A Half pipe

B Named pipe

C Anonymous pipe

Unit Review Answers
Q: What type of pipe can operate over a network?

A: Half pipe

B: Named pipe

C: Anonymous pipe

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What API gives you a handle to a process' token?

A OpenProcessToken()

B OpenToken()

C OpenProcess()

Unit Review Questions
Q: What API gives you a handle to a process' token?

A: OpenProcessToken()

B: OpenToken()

C: OpenProcess()

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What API gives you a handle to a process' token?

A OpenProcessToken()

B OpenToken()

C OpenProcess()

Unit Review Answers
Q: What API gives you a handle to a process' token?

A: OpenProcessToken()

B: OpenToken()

C: OpenProcess()

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What privilege gives complete write access regardless of the ACL?

A SE_BACKUP_NAME

B SE_RESTORE_NAME

C SE_WRITE_NAME

Unit Review Questions
Q: What privilege gives complete write access regardless of the ACL?

A: SE_BACKUP_NAME

B: SE_RESTORE_NAME

C: SE_WRITE_NAME

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What privilege gives complete write access regardless of the ACL?

A SE_BACKUP_NAME

B SE_RESTORE_NAME

C SE_WRITE_NAME

Unit Review Answers
Q: What privilege gives complete write access regardless of the ACL?

A: SE_BACKUP_NAME

B: SE_RESTORE_NAME

C: SE_WRITE_NAME

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- **Operational Actions**
- Persistence: Die Another Day
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 3

PE Format
Lab 3.1: PEParser
Threads
Injections
Lab 3.2: ClassicDLLInjection
Lab 3.3: APCInjection
Lab 3.4: ThreadHijacker
Escalations
Lab 3.5: TokenThief
Bootcamp
Lab 3.6: So, You Think You Can Type
Lab 3.7: UACBypass-Research
Lab 3.8: ShadowCraft

In this module, we will discuss several techniques centered around injection. There are a large number of injection methods and as you can see, we have our hands full of exercises today. Let's kick this off with the classic DLL Injection.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Bootcamp

- So, You Think You Can Type
- UAC Bypass-Research
- CustomShell

Bootcamp
For this bootcamp, this is the time to put the concepts learned from this section into practice. There are several challenges and all of them are optional. The challenges can also be done in any order since they do not depend on each other. The first challenge is to develop a program that implements similar functionality as ***GetProcAddress***.

Another challenge is to figure out how to intercept keystrokes that a user types into the notepad process.

The UAC Bypass challenge gives you the skillset to find a novel UAC bypass. The lab will only take you so far and then you must continue on your own. There is an excellent blog that heavily inspired this lab found here: https://github.com/Yet-Zio/WusaBypassUAC.

The final challenge is to continue creating your first baby implant. It should combine all features that have been covered so far.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.6: So, You Think You Can Type

- Must handle Windows messages
- Must intercept user key presses
- Must create a DLL to inject into target

Lab 3.6: So, You Think You Can Type
This challenge offers a great opportunity to explore building a keylogger. Since most keystrokes are commonly meant for GUI applications like Notepad, we will make that our focus. The idea for this challenge is to create a DLL that will install a Windows hook that will then be used to monitor keystrokes. Once you have the main logic completed, you can spice it up a bit by rejecting certain keystrokes. As an example, you can prevent the user from sending any vowels to the Notepad process. More robust keyloggers will just log everything and write it all out to a log file that will eventually be sent out to the C2 back end for analysis.

Please refer to the eWorkbook for the details of this bootcamp challenge.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.7: UACBypass-Research

- Find system binaries that have autoElevate set to true.
- Explore the process behavior using Process Monitor.
- Find a vulnerability and weaponize it to bypass UAC.

Lab 3.7: UACBypass-Research
This challenge is not for the faint of heart. There are two paths you could take with this one: you can follow along with the lab guide that only takes you so far, or you can dive off and attempt to find a brand-new UAC bypass! If you have time, do both. If you find a new method, please share it with the class and consider contributing it to the UACme project on GitHub.

Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 3.8: ShadowCraft

- Create a basic shell.
- Implement features covered in this section.
- Implement thorough error checking.

Lab 3.8: ShadowCraft
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024
https://linktr.ee/offsecexam

670.4

Persistence: Die Another Day

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

© 2023 Jonathan Reiter. All rights reserved to Jonathan Reiter and/or SANS Institute.

https://linktr.ee/offsecexam

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE (“SANS INSTITUTE”) FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware means all printed materials, including course books and lab workbooks, slides or notes, as well as any digital or other media, audio and video recordings, virtual machines, software, technology, or data sets distributed by SANS Institute to User for use in the SANS Institute course associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS Institute may suspend and/or terminate User’s access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User; (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There’s an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

REMnux® is a registered trademark of Zeltser Security Crop. Used with permission.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

Courseware licensed to User under this Agreement may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department (“OFAC”), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce (“BIS”). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulations. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

All reference links are operational in the browser-based delivery of the electronic workbook.

© SANS Institute 2024

https://linktr.ee/offsecexam

SEC670.4 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control

SANS

Persistence: Die Another Day

© 2024 Jonathan Reiter | All Rights Reserved | Version J01_03

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control: 670.4
Welcome to Section 4 of SEC670. In this section, we will discuss a number of ways that you could persist on target.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (1)	Page
In Memory Execution	4
Dropping to Disk	13
Binary Patching	20
Registry Keys	26
Services Revisited	39
Lab 4.1: PersistentService	47
Port Monitors	67
Lab 4.2: Sauron	74
IFEO	79
Lab 4.3: IFEOPersisto	90
WMI Event Subscriptions	97
Bootcamp	114

This page intentionally left blank.

Examrisk OffsecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents (2)	Page
Lab 4.4: NotInService	116
Lab 4.5: InitToWinit	117
Lab 4.6: OhMyWMI	118
Lab 4.7: CustomShell	119

Tamarisk OffSecExam Exam solution

This page intentionally left blank.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In Memory Execution
In this module, we will discuss what in memory execution is and how it can benefit us. We will also look at some disadvantages with only living in memory. Hint: it is volatile.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Describe in-memory execution
- Discuss advantages and disadvantages
- Discuss memory forensics

Objectives
The objectives for this module are to describe what in-memory execution is, discuss some advantages of it, as well as a few disadvantages, before wrapping it all up with a brief note on memory forensics.

© 2024 Jonathan Reiter

© SANS Institute 2024

In Memory Execution



Code execution not backed by a file on disk

In memory means exactly that—code is running in memory. For our purposes, however, we are going to define it as having an image in memory that is not backed by a file residing on the target's disk. Perhaps you have heard of fileless malware?

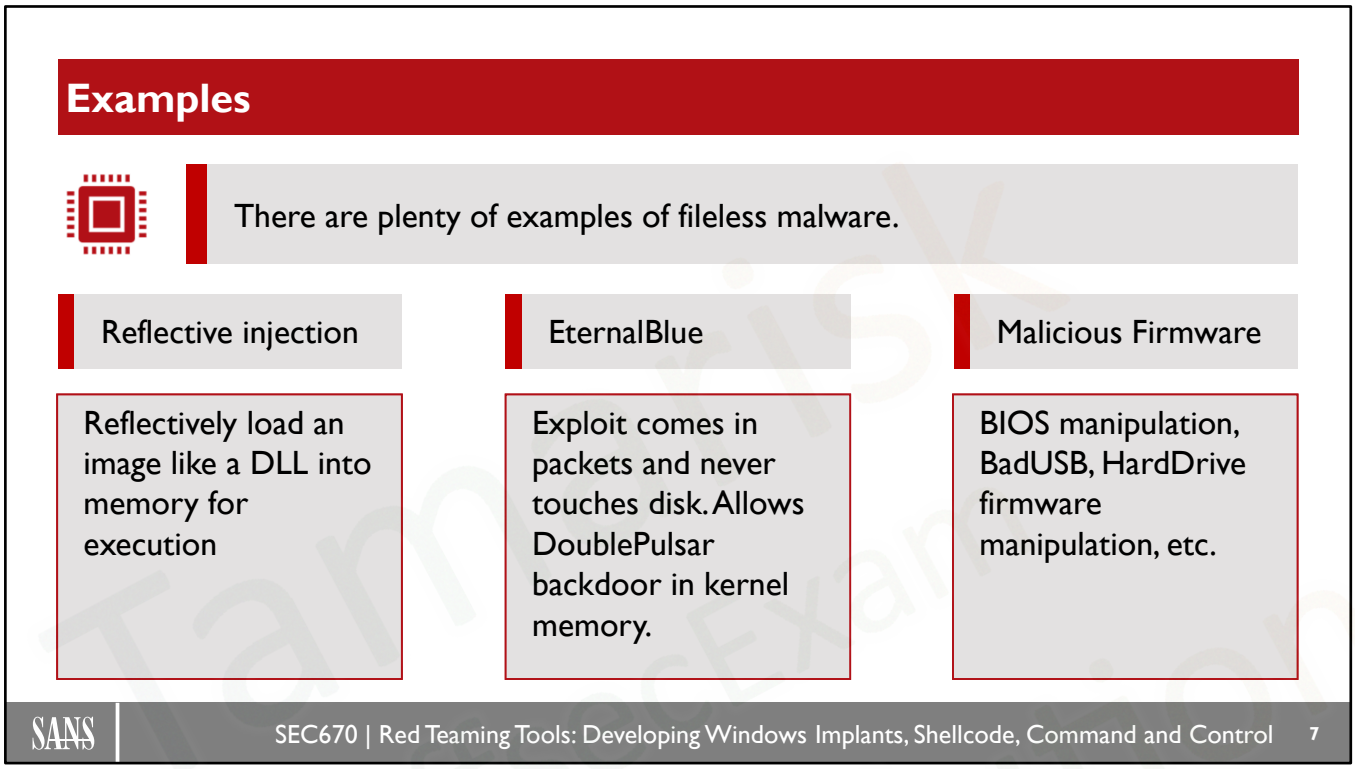
In Memory Execution

We could say that in-memory execution is when an executable image is mapped into memory and then a main thread is created to kick off the instructions located at the address of entry point, but we are not going to say that. Instead, we are going to say that in-memory execution is the execution of code, like shellcode or an executable image, executing in memory that is not backed by anything on the disk. Effectively, this is fileless malware. Fileless malware is malware that only exists in memory without ever leaving a single file on disk. Fileless code execution is very beneficial for the attacker. It is also more difficult to detect, which is another reason why fileless malware has been and is so popular. There have been many vulnerabilities in the past that attackers have been able to take advantage of for fileless execution. A prime situation for fileless code execution comes in the form of specially crafted packets that hit the NIC first and foremost. Kernel drivers, such as tcpip.sys, http.sys, netio.sys and others, are responsible for making sense of the data the NIC is holding in its buffers. That process presents an opportunity for kernel-level execution. Let us look a few examples.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Examples

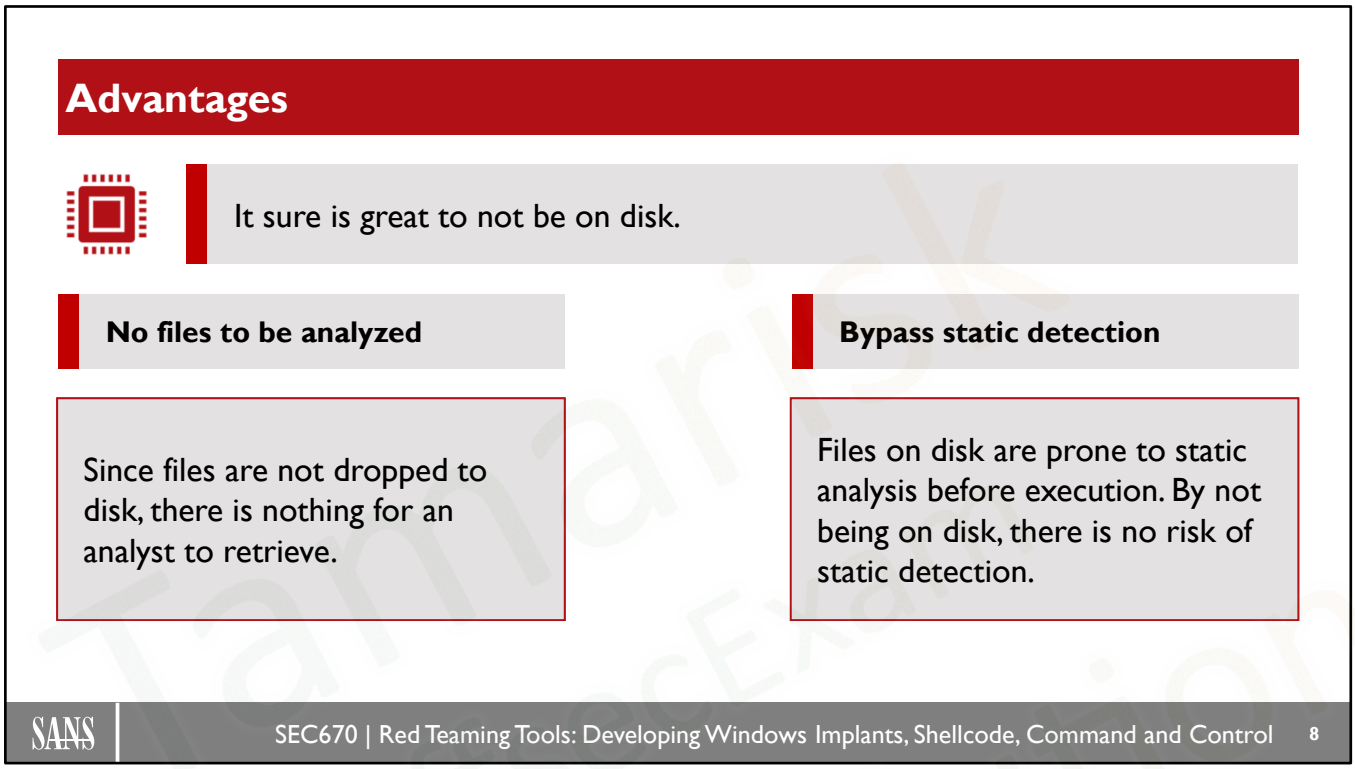
When it comes to examples of fileless malware and methods to achieve fileless execution, there is no shortage. Perhaps one of the more famous examples is WannaCry with its usage of EternalBlue. If you recall, EternalBlue came in the form of specially crafted network packets taking advantage of a 2017 SMB vulnerability. The nice thing about packets is they do not touch disk at all unless someone is saving them off using a packet capture tool like Wireshark. The backdoor that followed up with the exploit was dubbed DoublePulsar, which resided in the kernel. Another example of fileless malware or implants could come in the form of manipulating the BIOS. Not many EDR/AV solutions are built to check the BIOS. Same goes for performing a malicious firmware update to the controller of a hard drive. Going that low is incredibly stealthy and can provide amazing persistence and hiding capabilities. Red Balloon Security offers a unique hard drive challenge for its applicants, and it is quite the challenge! Also, you can reflectively inject an executable image into another process to achieve fileless execution. We will take a deeper look at Reflective DLL Injection during Section 5.

References:
https://awakesecurity.com/glossary/fileless-malware/
https://gbhackers.com/fileless-malware-wmi-eternalblue/

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Advantages

The biggest advantage for fileless malware is the fact that nothing is dropped to disk. The fact that certain programs, system files, tools, etc. cannot be blocked by IT staff without hindering support keeps enabling fileless attacks. Have you seen an organization that has completely blocked and/or removed PowerShell? Even without PowerShell you can still execute PowerShell; PowerShell without PowerShell. A forensic analyst or a reverse engineer would have nothing to do without files or without a memory dump. Another advantage is that by not having anything on disk, you will not be the victim of static analysis by some AV/EDR solution. There is nothing to hash, no bytes on disk to scan, nothing. The fact that fileless techniques are so powerful has forced EDRs and AV solutions to step up their game with behavior detection, memory scanning, and leveraging machine learning for their next-gen products. Even still, there is no single product out there that will always catch everything 100% of the time. It would be too resource intensive for products to constantly scan all memory regions of every process all the time.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Disadvantages



It sure is terrible to not be on disk.

Reboots flush RAM

Living in RAM is only good until a reboot flushes you out! What kicks off your execution now?

Initial access is lost

When the system powers on again, how is your access to the target regained?

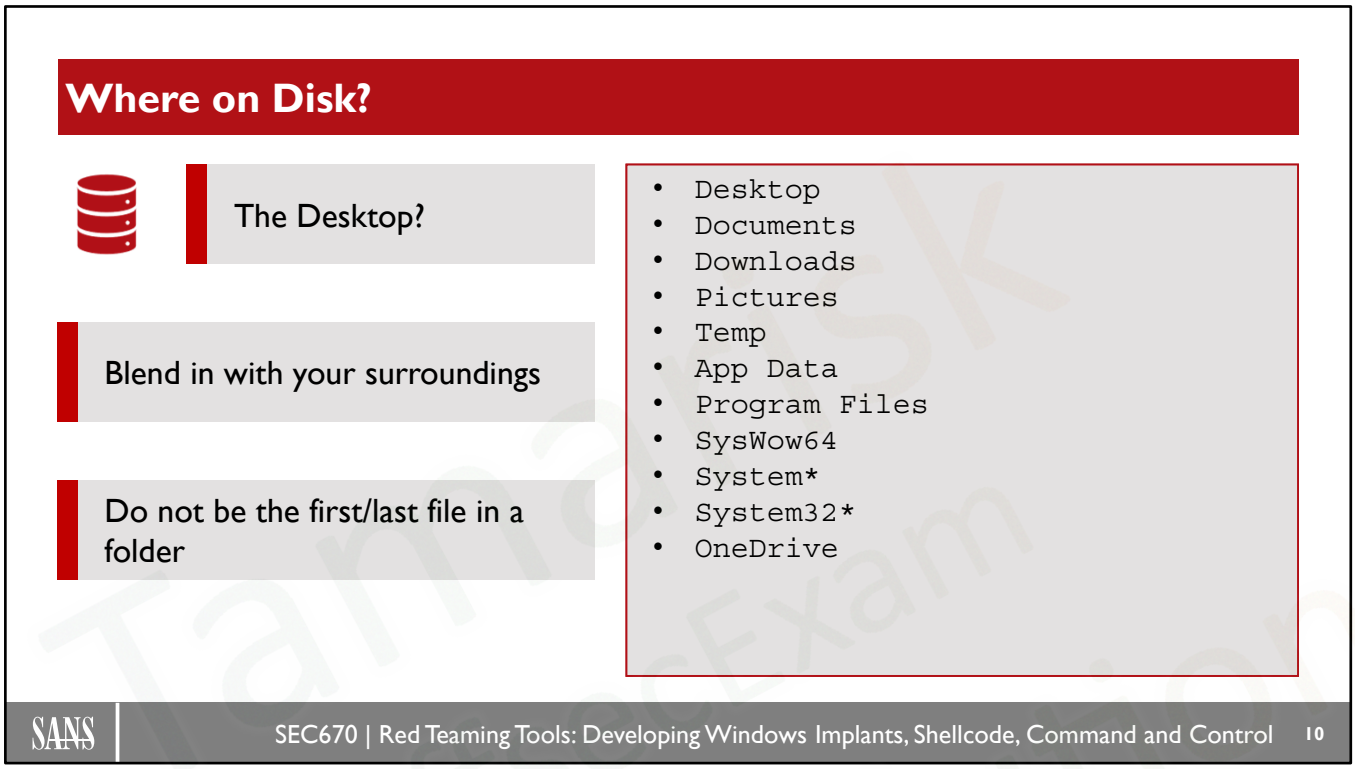
Disadvantages

How do you persist if you are only living in memory? Spoiler alert: you really cannot. This is arguably the biggest disadvantage when it comes to fileless malware. Machines are rebooted from time to time for various reasons: installing updates, power failure, hardware upgrades, scheduled downtime, etc. If you are someone who really enjoys bragging about system uptime, then fileless malware thanks you. Bottom line, though—a system will be rebooted at some point, so you must be prepared for this event. If you are not prepared, then your initial access is gone in a cloud of smoke. How do you get back on the target? Do you attempt to throw your exploit again to get you back on the system? The risks of getting caught go up much more significantly by throwing another exploit at the target. If the target is of no real importance to you, then you just move on and chalk it up as a temporary setback. However, if that system is of high value, then you need to maintain access to it, and that could mean dropping something to disk. It is often said that having two methods of persistence is just one method. Having just one method of persistence is really not having any at all.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Where on Disk?

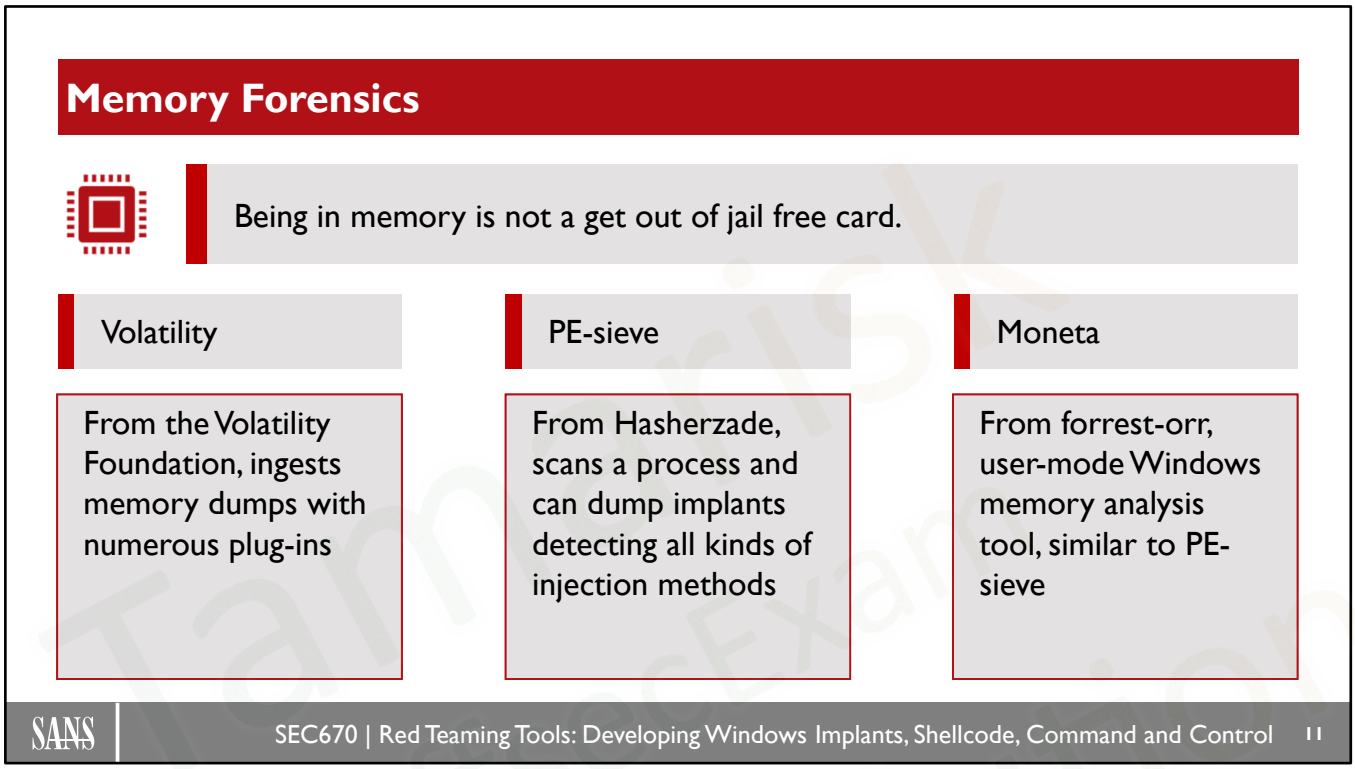
Imagine this scenario: you are running only in memory at the moment and your implant is monitoring for system shutdown events or changes to the power status, like if a laptop is no longer being charged. The system notifies you that a shutdown is pending, and you must maintain your access to this system because there is still intel on it that you have yet to collect. You make the decision to drop to disk so when the system powers back on, your file is triggered, and your access is renewed. The question you need to ask yourself is: Where do I drop to disk? This is a vulnerable moment for your implant, but it is also a necessity, so we must choose a smart location. The location could be entirely dependent on the system's environment, like what third party applications are present, what folders do you have write access to, and if there is cloud storage attached that you can drop into?

There are some general tactics and criteria you could use when looking for a possible location. Find a folder with a decent number of files in it. This would allow you to hide in plain sight somewhere past the halfway point in the listing. You do not necessarily want to be first, nor do you want to be last. You should avoid locations that are frequently browsed by the user like the Desktop, Documents, or Downloads locations. There is one caveat to the Desktop: some users like to store everything on the Desktop. They can be so cluttered that you can hardly see the user's background wallpaper. If that is the case, you could consider that as a possible location.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Memory Forensics

Almost everything you do on a Windows system can be logged by something. Even if a certain Windows tool does not catch your activity, highly motivated memory analysts with time on their hands will. There are several tools that can run live on a system, like PE-sieve and Moneta. They are both amazing tools and hiding from them has been seen as a challenge by researchers and developers alike. Only recently was Moneta bypassed by the awesome work of the MDSecLabs crew with their C2 Framework Nighthawk. Both PE-sieve and Moneta can scan processes and report suspicious findings, if any. Volatility is a Python framework geared at finding practically anything that was still resident in memory at the time of capture. They also host plug-in contests and some truly amazing plug-ins come from that. There are many more tools out there that offer similar capabilities to the ones noted on the slide. Bottom line, it is only a matter of time before you get caught. One great advantage here is that security products that are performing memory scanning simply cannot scan all regions of memory of every process every second of the day. It must be targeted and done in such a way so that it does not cause a performance hit for the user. There is nothing users hate more than a slow system. If a security product is causing high levels of degradation on system performance, odds are the user will remove it or ask that it be removed.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Defined and described in memory execution fileless malware
- Discussed fileless malware and methods
- Discussed advantages and disadvantages of being fileless
- Discussed memory forensics

Module Summary

In this module we covered what in memory execution is as it relates to implant development. We also discussed fileless malware and some popular examples, like WannaCry and EternalBlue SMB vulnerability. We also took a brief look at some advantages and disadvantages of only being in RAM. Are you ready for a shutdown/reboot?

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In this module, we will discuss why you would need to drop something to disk, where to drop, cleaning up, and more.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss the need to drop to disk
- Discuss the risks of being on disk
- Cover how to protect yourself and your data

Tamarisk OffSecExam Exam solution

Objectives

The objectives for this module are to discuss the need to drop to disk and when to do it. Being on disk brings some risk and we will discuss those risks. If we must be on disk, then we should try to protect ourselves as best as possible if our files become discovered.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Dropping to Disk



There is almost no escaping the need to be on disk.

Once your implant is on disk, it will be scanned and/or captured. It is only a matter of time. What is more important for your operation: access or your implant? If you get burned once, you are arguably burned everywhere if using the same tool.

Dropping to Disk

There will come a moment during the lifetime of your implant that something will have to touch disk. Dropping to disk is practically unavoidable these days when you are trying to survive across reboots. There are several risk factors that must be taken into consideration when dropping to disk, and the following is a non-exhaustive list of questions that should be answered before dropping to disk:

- Where should I drop to disk?
- Where do I have read/write permissions?
- What binary should I drop?
- How should I blend in with other items in the same folder?
- Is it acceptable to be scanned by AV solution?

The listing of considerations/questions should be answered with the data retrieved during the host survey phase. It would not be wise to blindly drop to some random folder and hope for the best. A location should be chosen strategically, one that would not indicate your presence to the user or sysadmin too quickly. If you are not elevated to at least a local Admin, you will not be able to drop your binary to some locations like System32, for example. System32 has many entries in it, and it would be easier to blend in there versus the current user's Downloads folder. Also, being on disk might mean that the installed AV solution might scan your binary. Is that okay? Perhaps the binary you drop only has basic features like your persistence/callback feature and that is it. This way, your entire capability is not discovered, and you do not have to spend months retooling your entire implant.

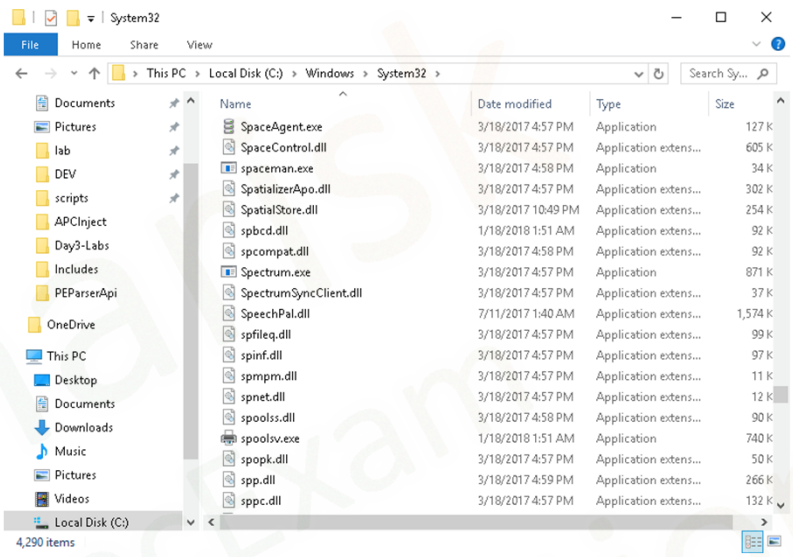
© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Blending In

On this Windows 10 VM, there are over 4,200 items in the System32 folder. Plenty of options to blend in with files around you.



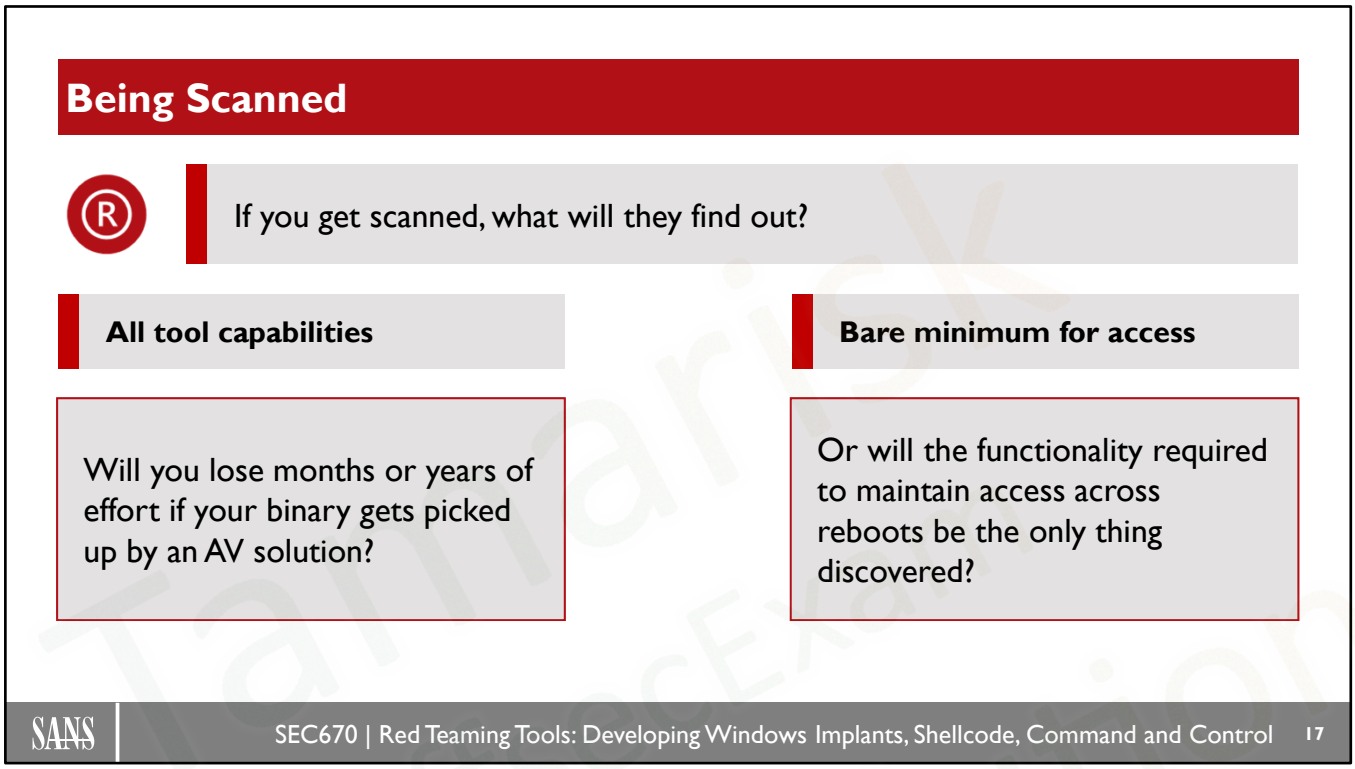
Blending In

When it comes to blending in, it can be somewhat easy with the proper level of permissions. As one possible example, a prime spot could be the System32 folder where there are well over 4,200 items to surround yourself. You would not necessarily want to be the first or the last entry in the folder but pick a spot that would require the user to scroll down for quite some time. Users are notorious for scrolling right past something unless they are specifically looking for it. Even then, most users still go past what they are looking for at least once or twice. Once your location has been determined, a filename would have to be chosen. The screenshot above shows several files that start with the letter "s." If there are more s-named files, then perhaps naming your binary something that starts with "s" might be a good idea. The screenshot also shows dates for the folder's files. The dates should be taken into consideration and your binary should reflect something close to them. If none of the files in the folder are beyond a certain year, like 2019, then you would not want to be the only binary reflecting the current year. Regardless of location, the bottom line is this, the more care you take with trying to blend in with your surroundings, the longer you might maintain a presence on that target.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Being Scanned

Another risk you must consider is should there be some AV solution installed like Defender, will that deter you from dropping to disk? If not, are you okay with it possibly being scanned? Some AV solutions require the submission to be sent to its cloud-based analysis engine if it has not seen it before. Most Windows system binaries and files are common, and their hashes/signatures are known. As such, cloud AV solutions do not need to upload them for analysis. The one-off binary, though, like yours, hopefully has never been seen or even scanned before. All of that could change when the solution detects that a new file has been created on the file system. What will the vendor find out if your implant is scanned? Will all your trade secrets be swept up and sent to the cloud or perhaps only the required features to maintain your access to the target? I cannot answer these questions for you as they are questions that can be ever changing even between operations.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Protecting Yourself



What can you do to protect your implant?

You could get extremely creative and DRM your implant like the PoC Skrull did. According to the author, the malware launchers are anti-copy and are thus broken if, and when, they are submitted for analysis.

Protecting Yourself

There are several public techniques and tools that are out there today that aid your efforts to protect yourself. There are commercial tools like packers and encryptors that do a tremendous job annoying reverse engineers. One such tool is Themida, which is made by the company Oreans. Themida is labeled as Advanced Windows software protection system and does so by protecting each code block, as well as adding an overall level of protection. You can read more on their website when you have a moment. Another interesting technique and perhaps the most interesting one that has recently surfaced, as of this writing, is Skrull. According to the author, Sheng-Hao Ma, “Skrull is a malware DRM, that prevents Automatic Sample Submission by AV/EDR and Signature Scanning from Kernel. It generates launchers that can run malware on the victim using the Process Ghosting technique. Also, launchers are totally anti-copy and naturally broken when got submitted.” Seems very interesting and clever. The source code for Skrull is hosted on GitHub for your viewing pleasure.

References:
https://www.oreans.com/themida.php
https://github.com/aaaddress1/Skrull

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed why we drop to disk to persist
- Discussed where to drop
- Learned the risks of being on disk
- Covered the various methods for protecting your work

Module Summary

In this module, we discussed why we would even drop to disk, where to drop, the risk(s) of being on disk, and how to protect what we have on disk.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In this module, we will discuss what binary patching is and how we can leverage it for persistence on target.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Define binary patching
- Discuss benefits of binary patching

Objectives
The objectives for this module are to define what binary patching is and discuss some of the benefits that binary patching has to offer.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What Is Binary Patching?



Modifying binaries to achieve results

What would happen if you patch a system file like NTDLL where it sits in System32? Your hooks would be implemented all over the place and it could draw way too much attention to you. Instead, you could patch a secondary or tertiary DLL that NTDLL loads.

What Is Binary Patching?

Binary patching is often referred to as modifying a binary as it resides on disk or in memory with the intention of changing how it executes. In memory patching is often done by AV/EDR solutions to change how functions of interest behave. What they implement is function hooking, which will be covered in detail in Section 5. Regardless, they are still modifying bytes of a file in memory. Patching files on disk can be challenging and could have damaging effects. Take for example an attempt to patch ntdll.dll: the file resides in the System32 folder and as such, you must have at least local Admin privileges. Because ntdll.dll is so heavily utilized, you could cause the system to become unstable and crash. Another common tactic is to patch the AV/EDR solution itself. Identifying a weakness in a solution can yield incredible results for not only persistence but also for protecting your toolset.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

In-Memory Patching



The patch is not permanent; will not survive reboots

Once you are injected into a process, how do you find what it is you are trying to patch? How about walking the Import Address Table? How about enumerating processes and obtaining handles to a process of concern? These are valid questions that could be answered either during your op or beforehand.

In-Memory Patching

Patching an image as it sits in memory is often referred to as in-memory patching. This is relatively safe since no changes are made to the binary on disk, so if the computer were to ever reboot, the changes would be flushed out of memory as if nothing ever happened. The big question you might have is, how do we find images in memory if we were just injected into a process? This is where process enumeration would come into play. If there is a process that you know about ahead of time based on gathered intel, then you can look for that process specifically. Otherwise, enumerate all processes and pick one from that newly collected information. Once you have found a process you want to patch, you need to obtain a process handle to it. If you recall, the handle is really the base address of the executable's image in memory, so if you were to read the first three bytes of memory you should find the famous MZ (\x4d\x5a) signature followed by a NOP, \x90. When conducting memory scanning and looking for the start of an image, it is always best to be as specific as possible so looking for \x4d\x5a\x90\x00 would possibly lower false positives. Now that you found the start of the PE header, it is simply a matter of parsing the rest of the PE headers to get to what you want. Often, you would be looking to patch out calls to certain functions like ***AmsiScanBuffer*** or ***AmsiScanString*** APIs from the Amsi.dll if you are trying to bypass AMSI. You could also decide to patch specific functions to change how they behave. You will see later how we can change how ***AmsiScanBuffer*** behaves.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

On-Disk Patching



Should survive reboots; cascading effect

Are files on disk better protected from patching than their memory mapped image? Is there a way to undo your changes if you accidentally break something with your patch? Could you render a system unstable if you patch system files? Could you get caught faster by patching files on disk?

On-Disk Patching

There might come a time when you would have to patch the file as it resides on the file system. If you have the proper permissions, you might be able to patch the binaries that offer signature scanning for the AV that might be installed on the system. Perhaps you found a way to get your tool added to some whitelist and you implement a patch to make sure you remain on the whitelist. System files might be able to be patched too, with the right permissions, of course. Take for an example patching Ntdll.dll. How many processes rely on Ntdll.dll? Practically all of them because the system loader is implemented in Ntdll.dll and it is the first module that is present in the process. Whatever patch you do on disk, each process is going to be affected by that change at some point in time, like after a reboot. This could make a system unstable and would require rigorous testing ahead of time. My recommendation is to avoid patching critical system DLLs unless there is a dire need that requires it. Until that time, stick to patching third party binaries.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed how patching can be incredibly useful
- Learned that in-memory patching is only temporary
- Learned that in-memory patching only effects one process
- Learned that on-disk patching is more permanent and could have cascading effects

Module Summary
In this module, we discussed two types of patching: in-memory and on-disk. Both have certain tradeoffs and which one you choose would be dependent on what your end goal is for your operation.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 4

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

Registry Keys
In this module, we will discuss how to persist using the Windows Registry.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss most used persistence key
- MITRE ATT&CK autostart locations
- Implement a few techniques

Objectives

The objectives for this module are to discuss what the most used key was for persistence. We will also explore several possible keys that can be leveraged for autostart entries. Lastly, we will reinforce the topics with source code review and a lab.

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Most Used Key



Run key is the most used by several APTs

The Run key exists for the current user (HKCU) and for all users (HKLM). Naturally, for HKLM, you would need elevated access, but not for HKCU. There is nothing novel about this method and yet many APTs and their malware families use it. The method is probably the easiest to implement.

`HKCU\...\CurrentVersion\Run`

`HKLM\...\CurrentVersion\Run`

Most Used Key

If you were to study all of the publicly available malware samples that have a persistence mechanism, you might find that perhaps the most used method is the Registry. More specifically, the Registry Key chosen for persistence is the Run key. Red Canary, Cynet, and a few other companies that specialize in malware analysis have noted that the Run key is most commonly used by malware authors. The interpretation is that they do not get caught right away, so why waste valuable time and effort trying to discover a novel method? APT28 (SOFACY), APT30, DarkComet, Emotet, FIN7, Gazer, Lazarus Group and more have all used this key as one of their persistence methods.

One could operate based on the concept of ”one is none, and two is one” where having only one method of persistence as your only lifeline is really not having any persistence at all. If you have two methods, then should one of the methods get burned or simply fail to trigger, you still maintain your foothold.

References:
https://attack.mitre.org/techniques/T1547/001/
https://www.cynet.com/attack-techniques-hands-on/the-art-of-persistence/

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Run vs. RunOnce

What is in a name?

Run key	*RunOnce/RunOnceEx key*
HKLM version runs as Admin. The HKCU version is great to be your downloader and initial method until you can find a better one.	Run the process once and delete the entry from the key. Good if trying to minimize artifacts. Bad if you need to stay on long term.

Run vs. RunOnce

There are several Registry keys that a person could use for persistence, but for this slide we will just focus on two of them. The keys under HKLM require Admin privileges and without them we would be forced to use a key under HKCU, which could expose our presence on the system quickly. The HKCU versions would require the user to log on to the workstation, whereas the HKLM versions kick off when the system boots up regardless of anyone logging on to the workstation. Sticking with the most used key among APTs, we have Run and RunOnce. The two keys do offer a means to accomplish your persistence, but they are definitely not equal. Despite the Run key being the most used method of persistence, the RunOnce key gets used as well, but is perhaps not always fully understood by everyone. The RunOnce key will do exactly what is sounds like it will do: it will run your application just once and be done. The difference is that after the one-time execution, your persistence mechanism has gone up in a cloud of smoke because your RunOnce entry will be deleted. Of course, it is not hard to make another entry after it has been deleted, just annoying is all.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

AppInit DLLs



Forcing a user process to load certain DLLs

When enabled, each newly created user mode process that is linked against User32.dll will load the DLLs annotated a list stored in the AppInit_DLLs Registry key. The list can be comma separated should there be a need to load more than one DLL.

LoadAppInit_DLLs (REG_DWORD)

AppInit_DLLs (REG_SZ)

AppInit DLLs
Windows provides users the flexibility to allow a pre-determined list of customized DLLs to be loaded into practically each user process. The catch is that the process must be linked against the User32.dll, and thankfully most, if not all GUI applications will be linked against it. The AppInit_DLLs Registry key is the key that dictates if the feature should be enabled or disabled for a system. The key is located under the HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows key for x64-bit processes and under HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\windows key for 32-bit processes. Under this key, a few items and values are looked for by User32.dll. User32.dll imports a function called ***LoadAppInitDlls***, which is implemented in Kernel32.dll. The module will perform a check to see if the LoadAppInit_DLLs key is set (1) or cleared (0). If it is set, then it is enabled system wide, and the list of DLLs specified in the AppInit_DLLs key will be loaded into the processes. Since this is under the HKLM hive, we would of course need to have Administrator privileges.

After our privilege escalation is done, we can create our list of DLLs with just one entry; the absolute path to our malicious DLL. After we do that, we enable the loading of AppInit DLLs by setting the LoadAppInit_DLLs key. A question that you may have had is what kind of DLLs can we use for this. Practically any DLL can be used with this technique so as long as it does not create a new process via the ***CreateProcess*** API. The resulting effect would cause an infinite loop of sorts because each time your DLL is loaded, it creates a new process, which will load your DLL, which creates a new process. One method to solve the infinite loop issue is to register a global namespace mutex that could be checked, and if it already exists, you know you have been triggered before so no need to execute again. The action of having our DLL creating a new process might not always be desired, so perhaps you could create some sort of check to see what process should be the one to load your malicious DLL.

This technique has been used by APTs such as APT39, CherryPicker, and T9000.

Reference:
https://attack.mitre.org/techniques/T1546/010/

© 2024 Jonathan Reiter

© SANS Institute 2024

AppCert DLLs



Certain Create* API calls look for AppCert.

Like AppInit, Windows will investigate the Registry for DLLs that must be loaded into a process. The AppCert key will be queried when the CreateProcess, CreateProcessAsUser, CreateProcessWithLogin, CreateProcessWithToken, or WinExec functions are called.

CreateProcess

WinExec

AppCert DLLs

Almost as a compliment to AppInit, there is AppCert, which is extremely similar in the sense that a Registry key will be queried to find a list of DLLs a process must load. The AppCert key would be under the HKLM\SYSTEM\CurrentControlSet\Current\Session Manager\AppCertDlls key, which means Administrator privileges are required. The key is queried whenever a process makes its first call to one of the following APIs: ***CreateProcess***, ***CreateProcessAsUser***, ***CreateProcessWithLogin***, ***CreateProcessWithToken***, or ***WinExec***. The major difference with AppCert is the format of the DLL being used. The DLL must export a specific function called ***CreateProcessNotify*** that will be called when the system is loading your DLL. One major drawback to this method is that the target computer must be rebooted before any of the changes you made are implemented. As such, AppCert is not stealthy and is easily detected by tools such as AutoRuns from SysInternals. We would never force a reboot live on a target even more so when a user is actively logged on to it. The best thing could be to wait for a corporate reboot policy to kick in and reboot the target for you.

This technique has been used by APTs such as Honeybee and PUNCHBUGGY.

Reference:
https://attack.mitre.org/techniques/T1546/009/

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Covered the most used persistence key
- Discussed MITRE ATT&CK autostart locations
- Discussed programmatic persistence

Module Summary

In this module, we covered current trends for registry persistence and what the most used key was. We also discussed the differences between the HKLM and the HKCU versions of keys like the Run key. We also hit on a few other keys, like AppInit and AppCert, along with the differences between them. All of these methods should be easily detected using tools like AutoRuns from SysInternals.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the most commonly used key for persistence?

A run

B runonce

C runtwice

Unit Review Questions
Q: What is the most commonly used key for persistence?

A: run

B: runonce

C: runtwice

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the most commonly used key for persistence?

A run

B runonce

C runtwice

Unit Review Answers
Q: What is the most commonly used key for persistence?

A: run

B: runonce

C: runtwice

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions

What permissions are needed to modify keys in the HKLM hive?

A User

B Admin

C Guest

Unit Review Questions
Q: What permissions are needed to modify keys in the HKLM hive?

A: User

B: Admin

C: Guest

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers

What permissions are needed to modify keys in the HKLM hive?

A User

B Admin

C Guest

Unit Review Answers
Q: What permissions are needed to modify keys in the HKLM hive?

A: User

B: Admin

C: Guest

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What technique should be used for processes linked against User32.dll?

A AppCert

B AppInit

C RunOnce

Unit Review Questions
Q: What technique should be used for processes linked against User32.dll?

A: AppCert

B: AppInit

C: RunOnce

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What technique should be used for processes linked against User32.dll?

A AppCert

B AppInit

C RunOnce

Unit Review Answers
Q: What technique should be used for processes linked against User32.dll?

A: AppCert

B: AppInit

C: RunOnce

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In this module, we will discuss how to persist using services. This includes services that we take advantage of or services that we create ourselves.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss how services can aid in persistence
- Understand the service-related Win32 APIs
- Create new services and/or modify existing ones
- Discuss methods for how to hide a service

Objectives

The objectives for this module are to revisit services, but this time around the focus is on persistence. We will spend some more time understanding how services are created by first understanding the Win32 APIs specific to Windows services. We will also explore how we could modify existing services to give us our persistence. Lastly, we will cover how services can be hidden from view and from tools that query services.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Services

To be, or not to be, a service?

Windows services are ripe for attackers for several reasons: commonly misconfigured, incorrect permissions, automatic starts, run as SYSTEM, etc. Most users do not even pay attention to new services or ones that have been modified.

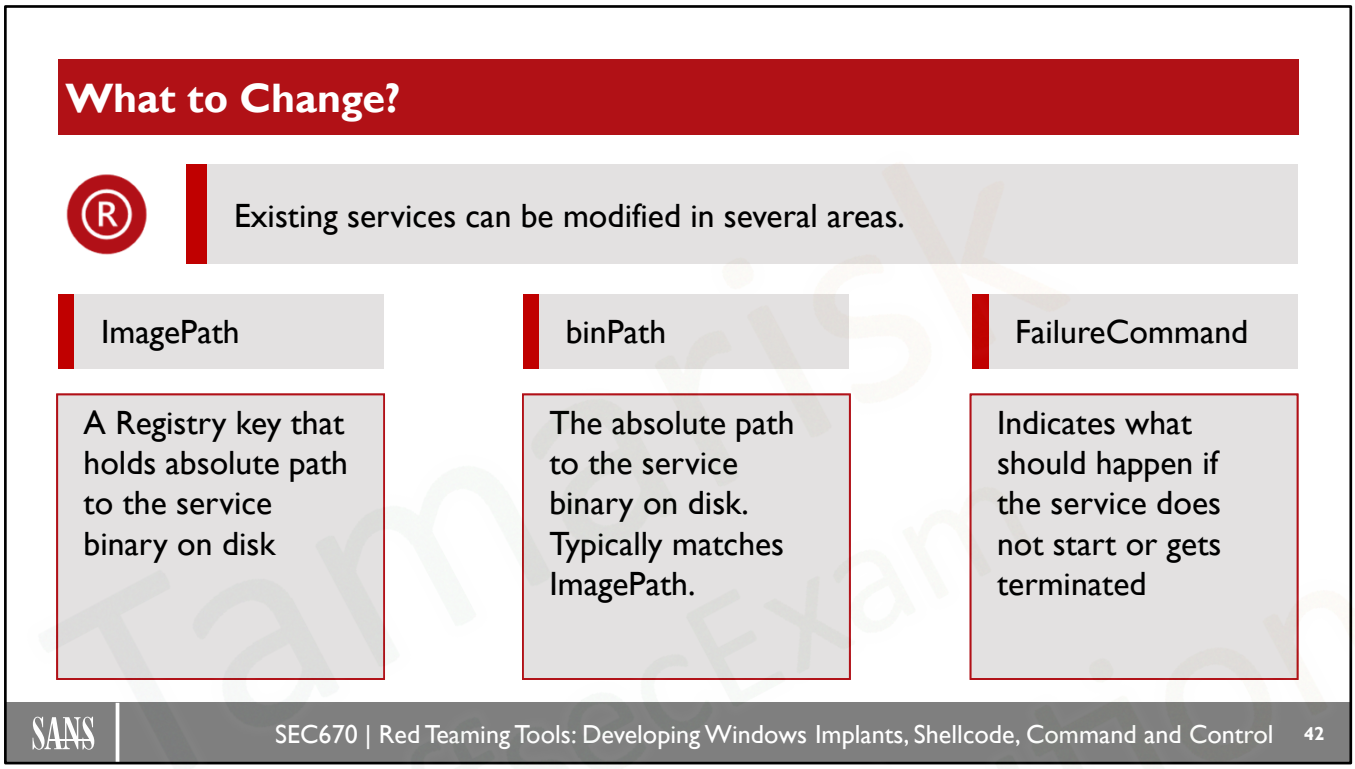
Services

This is not the first time that Windows services have been mentioned, so they must be important, and they are. Services are not just important to the operating system, but they are also critical for red teamers. Earlier in the course we saw how they play a role for our enumeration and for privilege escalation. Now we can change our perspective to that of persistence. Red teamers, penetration testers, hackers, malware authors, etc. all look for services to use for their advantage. Services, especially third-party ones, might have been created with misconfigurations. A service's binary path, if it has spaces in it, must have quotes around it. The Registry key *ImagePath* can be targeted along with the service's Failure Command. We can either fire off our persistence binary when the target service starts, when it terminates improperly, or even if it fails to start properly. Of course, we can also create our own service for persistence instead of hijacking an existing one.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



What to Change?

The *ImagePath* is a Registry key for the service. Most of the time, the value held is the path to the service executable on disk. The value, or arguments, in the key are passed to the service's main function—the service's entry point. When a program calls the ***CreateService*** API, the *lpBinaryPathName* argument indicates what the value will be for this key.

The *binPath* is the path of the service binary and the value typically will match that of the *ImagePath* Registry key. The system does not provide a default value for you, so this is a mandatory item. Many penetration testers or red teamers will forget to fulfill this requirement when making a service for their persistence only to see it fail within a few seconds. If we change the *binPath* to point to an executable that we dropped to disk, then we could achieve persistence. The *FailureCommand* is a command-line command that should be executed if the service fails for whatever reason. This change can also help us maintain our access to a system; but the service needs to fail, or our command will never kick off.

All of these could be manipulated via the command-line utility sc.exe.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Service Failure



Failure is an option.

The Service Control Manager considers a service to have failed when a service terminates but does not report the SERVICE_STOPPED status to it. Furthermore, if the Win32ExitCode member of the SERVICE_STATUS structure does not indicate success (ERROR_SUCCESS), then it is also considered to have failed.

Service Failure

When does a service really fail and by whose standard? In the world of Windows services, the Service Control Manager (SCM) is the end-all-be-all. Services must answer to the SCM and if they do not, they are promptly terminated. If that terminated service is yours—well, there goes your persistence. Services are deemed as a failure when it does not start up properly, and starting up properly means sending the correct status to the SCM in time. When it comes to service termination, services must also send a status to the SCM. The SERVICE_STOPPED status must be sent before its function returns or else the SCM will follow any failure actions that have been registered for the service. Even more so, when a service exits but fails to set an ERROR_SUCCESS, or NO_ERROR (0) inside the SERVICE_STATUS structure, it is deemed a failure. It is very critical that services inform the SCM of what is happening and in a timely fashion because the SCM is not very forgiving. The failure actions the SCM looks for are defined in the SERVICE_FAILURE_ACTIONSA structure.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

SERVICE_FAILURE_ACTIONS



SERVICE_FAILURE_ACTIONS

Dictates what takes place after a failure

Requires SCM to determine a service failed

```
typedef struct
_SERVICE_FAILURE_ACTIONSA {
 DWORD   dwResetPeriod;
 LPSTR   lpRebootMsg;
 LPSTR   lpCommand;
 DWORD   cActions;
 SC_ACTION *lpsaActions;
} SERVICE_FAILURE_ACTIONSA,
*LPSERVICE_FAILURE_ACTIONSA;
```


SERVICE_FAILURE_ACTIONS
The SERVICE_FAILURE_ACTIONS structure is utilized by the SCM as a representation of what must happen when the SCM fails a service. The actions defined in the structure are executed by the SCM each time the service enters a failure state. Existing services can have these optional configuration parameters changed using the ***ChangeServiceConfig2*** API and passing the SERVICE_CONFIG_FAILURE_OPTIONS flag for the *dwInfoLevel* parameter. The members of the structure are described below.

dwResetPeriod, is the period that must expire before the failure count is decremented to 0. The SCM maintains a count of failures so that the correct failure actions can be executed after repeat failures. A common example would be for critical services that may delay their restart time with each successive failure.

lpRebootMsg, is the message to be sent to the server to indicate to users that the system will be rebooted due to the service's failure. NULL here indicates to use the system's default message, but if you pass empty quotes (""), then the default message is overridden, and no message will be displayed to the users.

lpCommand, is the command line that is passed to the ***CreateProcess*** API. The command will be executed with the same permissions as the service.

cActions, is the number of elements that are in the *lpsaActions* array.

lpsaActions, is an array of SC_ACTION structures that indicate what action the SCM should take in the event the service fails for the nth time. For example, if there are four items in the array and the service failed for the fourth time, then that entry will be executed.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation

- SERVICE_FAILURE_ACTIONS
- Create and zero out the struct
- Call the change service API

```
SERVICE_FAILURE_ACTIONSA sfa;
SecureZeroMemory(&sfa, ...);

sfa.dwResetPeriod = INFINITE;
sfa.lpRebootMsg = "";
sfa.lpCommand = "ping C2";
sfa.cActions = 0;
sfa.lpsaActions = NULL;

ChangeServiceConfig2( hService,
  SERVICE_CONFIG_FAILURE_ACTIONS,
  &sfa );
```


Implementation

The pseudo code one the slide shows just one of several ways you could go about implementing a service's failure actions. It is best to do this after you have installed and created the service. The overall idea for this small example is to ping the configured C2 server if the service fails. The beginning of the code starts off with the SERVICE_FAILURE_ACTIONS struct. We make a variable called *sfa* of that type and securely zero out the memory of that struct by the size of the struct. The struct members are modified in the order they appear in the struct's definition, to keep things easier to read. The reset period has an INFINITE value because we do not care if the failure count is reset or not—for this example, anyway. For the reboot message, we pass in an empty string because we do not want the default message or really any message at all. The empty string overrides that and NULL does not change the reboot message. The command is the heart and soul for this method. Our example simply pings our C2 server to demonstrate what could be done. There are no actions in the SC_ACTION array, so 0 and NULL are used. After you are done modifying the structure, the *ChangeServiceConfig2* API can be called.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Persistent Service

Source code review!

Persistent Service

Time to jump into the source code and explain it before you implement it on your own.

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.1: PersistentService



Creating your own service for persistence

Please refer to the eWorkbook for the details of the lab.

Lab 4.1: PersistentService
Please refer to the eWorkbook for the details of this lab.

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to learn how to either create, install, or persist with your own service. When you have administrative and/or SYSTEM privileges, creating a service is a perfect action to take.

Tamarisk OffSecExam Exam Solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What Else?



Hiding a service

When the opportunity is there to hide ourselves from view, we tend to take advantage of it. We can hide a service similar to how we can hide a process from a user and other tools. The beauty about this is there is no need for a kernel driver of function hooking.

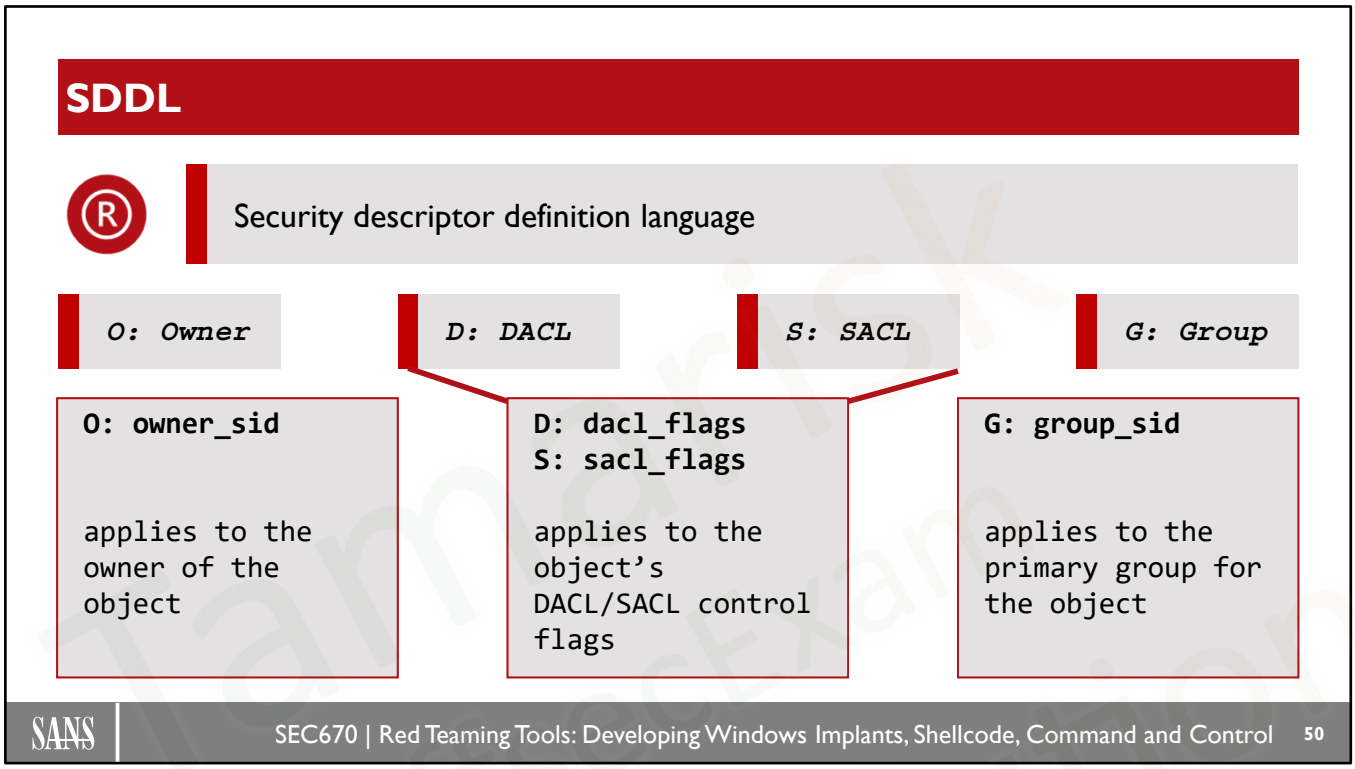
What Else?

What else can we do with services? Well, one thing we can do is we can hide a service. A service that we created for our own persistence needs can be hidden from the user and from several tools like the sc.exe command-line utility. We hide a service for similar reasons that we hide a process; stay hidden, stay safe. When you make it harder for users and tools to detect you, it helps you avoid detection longer. Hiding a service that implements your persistence portion could really help you stay on a system longer. With this technique, we do not need to hook any functions, nor do we need to install a kernel driver. Each service has a security descriptor tied to it and we can manipulate that object. The way we do this is through a Microsoft-specific language called SDDL, the security descriptor definition language.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



SDDL

The security descriptor definition language, or SDDL, is not really a language, per se, but something that is specific to the definition of a string format for two APIs: ***ConvertSecurityDescriptorToStringSecurityDescriptor*** and ***ConvertStringSecurityDescriptorToSecurityDescriptor***. The APIs are used to describe the security descriptor as a string. With SDDL, we can also hit specific components of a security descriptor just using strings. There is a specific format that must be followed for our hiding technique to work. At first, it might seem like it does not make any sense, but after a bit of practice crafting some strings, you will get the hang of it.

A security descriptor will have four main components, such as the Owner (O:), the SACL (S:) and DACL (D:), and the Primary Group (G:). The Owner is the owner of the object as identified by an *owner_sid*. The SACL has control flags as represented by *sacl_flags*. The DACL also has control flags as represented by *dacl_flags*. The Primary Group is the object's primary group as identified by the *group_sid*. There is also a *string_ace* that is enclosed in parenthesis that is used to describe the ACE for a SACL or DACL security descriptor. You must use an *ace_string* when changing the DACL or SACL of the object.

The control flags that can be used for the DACL or SACL are the following list of items:

- **"P"** - *SDDL_PROTECTED* - SE_DACL_PROTECTED flag is set
- **"AR"** - *SDDL_AUTO_INHERIT_REQ* - SE_DACL_AUTO_INHERIT_REQ flag is set
- **"AI"** - *SDDL_AUTO_INHERITED* - SE_DACL_AUTO_INHERITED flag is set
- **"NO_ACCESS_CONTROL"** - *SDDL_NULL_ACL* - ACL is NULL

For reference, a DACL is a discretionary access control list and SACL is a system access control list.

Reference:
https://docs.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-string-format

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Ace String Layout

ace_type; ace_flags; rights; object_guid; inherit_object_guid; account_sid; (resource_attribute)

ace_type

A: access allowed
D: access denied
OA: object allowed
OD: object denied
AU: audit
AL: alarm

ace_flags

CI: container inherit
OI: object inherit
NP: no propagate
IO: inherit only
ID: inherited
SA: audit success

generic rights

GA: generic all
GR: generic read
GW: generic write
GX: generic execute

registry rights

KA: all
KR: read
KW: write
KX: execute

standard rights

RC: read control
SD: standard delete
WD: write dac
WO: write owner

directory rights

RP: read property
WP: write property
CC: create child
DC: delete child
LC: list children
SW: self write

label rights

NR: no read up
NW: no write up
NX: no execute up

file rights

FA: all
FR: read
FW: write
FX: execute

Ace String Layout

As mentioned on the previous slide, *ace_strings* are used whenever we want to change the object's DACL or SACL. This slide details the syntax used for creating an *ace_string* and a short description of several of the values. There is a lot on the slide, but there is even more documentation to be found on the MSDN site. At the top of the slide is the syntax for how an *ace_string* should be formatted. The *ace_type* is the first field in the string and can be any number of types that belong to that group. The simplest is to allow or deny access to an object. The ace_flags field is typically skipped since it does not directly apply to hiding a service. We indicate an empty field value or a skipped field like so, ";;". Generic access rights, standard access rights, directory service object access rights, file access rights, registry access rights should sound somewhat familiar to you with the APIs that have been used so far in the course. Not mentioned on the slide due to lack of room are the *account_sid* values. The *account_sid* values indicate the owner or primary group of the object. The famous SID strings (S-1-1-0) can be used here, or you can use the equivalent constant with some of them listed below.

- AU: Authenticated users
- BA: Built-in administrators
- DU: Domain users
- LA: Local admin
- SY: Local system

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Viewing Security Descriptors

```
qusерservice----Queries for a local instance of a user service template.
delete----------Deletes a service (from the registry).
create----------Creates a service. (adds it to the registry).
control---------Sends a control to a service.
sdshow----------Displays a service's security descriptor.
sdset-----------Sets a service's security descriptor.
showsid---------Displays the service SID string corresponding to an arbitrary name.


C:\>sc sdshow bits

D:(A;CI;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
(A;;CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)S:(AU;SAFA;WDWO;;;BA)
```


Viewing Security Descriptors

Using the sc.exe command-line utility, we can view a service's security descriptor. Running the program with the /? argument shows the help menu. From the help menu we can see the argument *sdshow* and its description: Displays a service's security descriptor. This is what we want. From here we can choose a service, like BITS, and see what its security descriptor currently is. With the information we now know about SDDL and *ace_strings*, we can interpret the output without too much headache.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

SDDL Example #1

- MSDN Example
- Several field types skipped
- Uses the NULL well-known SID

```
"O:AOG:DAD:(A;;RPWPCCDCLCSWRCWDWOGA;;;S-1-0-0)"


Revision: 0x00000001,
Control:  0x0004, SE_DACL_PRESENT
  Owner: (S-1-5-32-548)
  PrimaryGroup: (S-1-5-21-397955417-626881126-
188441444-512)
DACL - Revision: 0x02, Size: 0x1c, AceCount: 0x01
  Ace[00]
    AceType: 0x00 (ACCESS_ALLOWED_ACE_TYPE)
    AceSize: 0x0014
    InheritFlags: 0x00
    Access Mask: 0x100e003f
    READ_CONTROL | WRITE_DAC | WRITE_OWNER |
    GENERIC_ALL, Others(0x0000003f)

   Ace Sid    : (S-1-0-0)
```


SDDL Example #1

This first example is taken directly from MSDN documentation. The *ace_string* is the most interesting piece of the string and here is the breakdown. Since the first field of the *ace_string* is the *ace_type*, this is allowing access and more specifically the ACCESS_ALLOWED_ACE_TYPE. The next field is *ace_flags*, which is empty. The *rights* field is next and is where our focus is for truly understanding what is happening. We break this down in pairs starting with "RP", which indicates the read property (ADS_RIGHT_DS_READ_PROP), "WP" indicates write property (ADS_RIGHT_DS_WRITE_PROP), "CC" indicates create child (ADS_RIGHT_DS_CREATE_CHILD), "DC" indicates delete child (ADS_RIGHT_DS_DELETE_CHILD), "LC" indicates list children (ADS_RIGHT_ACTRL_DS_LIST), SW" indicates (ADS_RIGHT_DS_SELF), "RC" indicates read control (READ_CONTROL) and is a standard access right, "WD" indicates write DAC (WRITE_DAC), "WO" indicates write owner (WRITE_OWNER), and "GA" indicates generic all (GENERIC_ALL). The *object_gui* is skipped and the *inherit_object_guid* is also skipped. For the *account_sid* we see the well-known NULL SID is used because the actual SID might not have been known at the time. To our benefit, the interpretation of the string has been given for a full explanation of the effects.

Reference:
https://docs.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-string-format

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Exercise: SDDL



Hiding the service

From Joshua Wright

Applies to several SIDs

```
"D:
(D;;DCLCWPDTSD;;;IU)
(D;;DCLCWPDTSD;;;SU)
(D;;DCLCWPDTSD;;;BA)
(A;;CCLCSWLOCRRC;;;IU)
(A;;CCLCSWLOCRRC;;;SU)
(A;;CCLCSWRPWPDTLOCRRC;;;SY)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
S:
(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)"
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 54

Exercise: SDDL
SANS instructor and author Joshua Wright crafted the SDDL string shown on the slide during an engagement and used it to hide a service he created. Take 10-15 minutes to break down this real-world example piece by piece just like we did together for the previous example from MSDN.

Reference:
https://www.sans.org/blog/red-team-tactics-hiding-windows-services/

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Exercise: SDDL: The Solution

```
"D:        DACL
(D;;DCLCWPDTSD;;;IU)
  interactive user, deny: delete, list, write, delete tree, standard delete
(D;;DCLCWPDTSD;;;SU)
  service user, deny: delete, list, write, delete tree, standard delete
(D;;DCLCWPDTSD;;;BA)
  built-in admins, deny: delete, list, write, delete tree, standard delete
(A;;CCLCSWLOCRRC;;;IU)
  interactive user, allow: create, list, selfwrite, list obj, control access, read control
(A;;CCLCSWLOCRRC;;;SU)
  service user, allow: create, list, selfwrite, list obj, control access, read control
(A;;CCLCSWRPWPDTLOCRRC;;;SY)
  local system, allow: create, list, selfwrite, read/write property, delete tree, list obj...
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
  built-in admins, allow: create, delete, list, selfwrite, read/write property, delete tree...
S:         SACL
(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)”
  everyone,
```


Exercise: SDDL: The Solution

Here is the breakdown of the string applied to their service. To make this a bit easier, we can first look at the SIDs for which the rules will apply. Starting with the DACL portion (D:), we see “IU”, “SU”, ”BA”, and “SY” being used for the *account_sid,* which apply to the logged-on interactive user, the service logon user, the built-in administrators, and local system, respectively. The logged-on user is denied delete and list accesses to the object, cannot write or delete the directory recursively, and cannot even use standard delete. The service logon user is denied the same items as the logged-on user as well as the built-in admins. The SACL has an *ace_string* that applies everyone’s access like so: audited, audited failure, create, delete, list, self-write, read/write property, delete tree, list object, control read, standard delete, read control, write dac, and write owner. If you were to apply this to a service that you created, try to query it with various tools like PowerShell’s Get-Service cmdlet, the sc.exe utility, etc. and see what you are shown.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Exercise: SDDL: Unhiding the Service

```
sc.exe sdset SWCUEngine
D:
(A;;CCLCSWRPWPDTLOCRRC;;;SY)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
(A;;CCLCSWLOCRRC;;;IU)
(A;;CCLCSWLOCRRC;;;SU)
S:
(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)


The command on a single line
sc.exe sdset SWCUEngine \
D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)(A;;CCLCSWLOCRRC
;;;IU)(A;;CCLCSWLOCRRC;;;SU)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)
```


Exercise: SDDL: Unhiding the Service
As professional red teamers, we need to clean up and one of the ways we can do that is to unhide the service. The command on the slide will restore the accesses back so that users and tools can "see" the service again. As can be seen on the slide, the command is broken out for readability. When running the command on target, it would all be on a single line as shown on the bottom of the slide.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Exercise Complete: STOP

You have successfully completed the exercise.
Congratulations!

Exercise Complete: STOP
This marks the completion of the exercise. Congratulations on successfully completing all the exercise steps!

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Programmatically Hide a Service



Manual versus programmatically

As with almost everything you do manually in a shell like the cmd prompt, there are Windows APIs on the back end that enable that effort. SDDL strings and ACE strings can be daunting to hand jam in an interactive session, but perhaps using the APIs is easier.

Programmatically Hide a Service

Crafting your SDDL string to change the permissions of an object is not intuitive by any means and rather annoying. This is largely due to the archaic SDDL syntax, but it can come in handy for those one-offs where you do not have tool that can implement this feature, nor do you have the time to create one. Thankfully, you are not stuck having to produce an ugly looking SDDL string. Windows has structures and APIs that can be configured and called to effectively hide a service. Knowing the various methods of how this is done via the various Windows APIs is important for developers—that way this feature can be added or created in our toolset. The securitybaseapi.h header file declares a very large number of APIs that are specific to changing the control bits of a security descriptor and then some. This is what we were doing manually with the SDDL strings and the *sc.exe sdset* command-line utility. APIs such as ***SetSecurityDescriptorControl***, ***SetSecurityControlDacl***, ***SetSecurityDescriptorSacl***, ***SetSecurityDescriptorOwner***, etc. are where you would want to look first for this implementation in code. For the most part, they are all Boolean type functions and are very easy to implement. Another important header file to look at would be aclapi.h and, as the name implies, there will be APIs, such as ***GetNamedSecurityInfo*** and ***SetNamedSecurityInfo*** that directly tie to obtaining and or modifying ACLs.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

GetNamedSecurityInfoA



GetNamedSecurityInfoA

Copies the security descriptor of the specified object by name

NTFS objects, services, keys, shares, file-mapping objects

```
DWORD GetNamedSecurityInfoA(
 LPCSTR pObjectName,
 SE_OBJECT_TYPE ObjectType,
 SECURITY_INFORMATION SecInfo,
 PSID *ppsidOwner,
 PSID *ppsidGroup,
 PACL *ppDacl,
 PACL *ppSacl,
 PSECURITY_DESCRIPTOR *pSecDscrptr
);
```


GetNamedSecurityInfoA

The **GetNamedSecurityInfoA** API is used for when you would want to obtain the security descriptor of an object of interest. The great thing about this API is that is can be used for objects that are local to the system as well as objects that might be on a remote system. Because Windows has many object types, it can be daunting to fully understand each object type and how they can be secured. Thankfully, we can peek into them with this API. The SAL annotations have been omitted on the slide for brevity but have been included down below in the slide notes. There are three in parameters that are required and five out parameters that are optional. To get the most details about an object, you can pass in pointers for each of the out parameters. Passing in pointers for those extra parameters will tell the API that you want information about the Owner and Group SID, along with the DACL and SACL.

Here is a detailed breakdown of the API's parameters.

_In_ LPCSTR pObjectName, is the pointer to the name of the object that you are querying. Remember, this string must be a NULL-terminated string.

_In_ SE_OBJECT_TYPE ObjectType, is an enum entry from the enum SE_OBJECT_TYPE. This lets the API know the type of the object that pObjectName should be in the system.

_In_ SECURITY_INFORMATION SecurityInfo, is how you indicate to the API the security information set you are querying. Many of these can be combined using a bitwise OR like ATTRIBUTE_SECURITY_INFOMRATION | DACL_SECURITY_INFORMATION.

*_Out_opt_ PSID *ppsidOwner*, if desired, will hold the pointer to the owning SID if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PSID *ppsidGroup*, if desired, will hold the pointer to the primary group SID if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PACL *ppDacl*, if desired, will hold the pointer to the DACL if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PACL *ppSacl*, if desired, will hold the pointer to the SACL if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PSECURITY_DESCRIPTOR *ppSecurityDescriptor*, will point to the security descriptor for the requested object. The buffer the API made for you must be freed by calling ***LocalFree***.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

SetNamedSecurityInfoA



SetNamedSecurityInfoA

Applies what is in the security descriptor for a chosen object

Objects will be given by their name

```
DWORD SetNamedSecurityInfoA(
 LPSTR pObjectName,
 SE_OBJECT_TYPE ObjectType,
 SECURITY_INFORMATION SecInfo,
 PSID psidOwner,
 PSID psidGroup,
 PACL pDacl,
 PACL pSacl
);
```


SetNamedSecurityInfoA

The ***SetNamedSecurityInfoA*** API is used for when you want to confirm or apply a change to security information in an object's security descriptor that you previously obtained. As with the ***GetNamedSecurityInfo*** API, this can deal with many types of objects. The objects can be local to the system or on a remote system given proper permissions or accesses to it. Let us break down the API's parameters. If you are modifying objects that do have existing ACEs, then you must call ***SetEntriesInAcl*** before you can call ***SetNamedSecurityInfo***. ACE entries will then be merged in the object's DACL.

_In_ LPSTR pObjectName, is the pointer to the name of the object that you are querying. Remember, this string must be a NULL-terminated string.

_In_ SE_OBJECT_TYPE ObjectType, is an enum entry from the enum SE_OBJECT_TYPE. This lets the API know the type of the object that pObjectName should be in the system.

_In_ SECURITY_INFORMATION SecurityInfo, is the combined flags using a bitwise OR for the information that is to be set.

_In_opt_ PSID psidOwner, is a pointer to the SID of who the owner of the object should be. For this to take effect, you must first have WRITE_OWNER access or more like SE_TAKE_OWNERSHIP_NAME. Must use OWNER_SECURITY_INFORMATION for this to work.

_In_opt_ PSID psidGroup, if passed, can be used to indicate who the primary group of the object will be from here on out. Must use GROUP_SECURITY_INFORMATION for this to work.

_In_opt_ PACL pDacl, if passed, can be used to point to the new DACL of the object. Must use DACL_SECURITY_INFORMATION for this to work.

_In_opt_ PACL pSacl, if passed, can be used to point to the new SACL of the object. Must use SACL_SECURITY_INFORMATION, as well as a few others, for this to work.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

EXPLICIT_ACCESS_A



EXPLICIT_ACCESS_A

Defines access control information for a trustee

The user, group, program to apply it against

```
typedef struct _EXPLICIT_ACCESS_A
{
  DWORD       grfAccessPerms;
  ACCESS_MODE grfAccessMode;
  DWORD       grfInheritance;
  TRUSTEE_A   Trustee;
} EXPLICIT_ACCESS_A,
  *PEXPLICIT_ACCESS_A,
  EXPLICIT_ACCESSA,
  *PEXPLICIT_ACCESSA;
```


EXPLICIT_ACCESS_A
The ***EXPLICIT_ACCESS_A*** structure is heavily used whenever modifications are being made to the ACL of an object. The structure is used to describe to the system the information in an ACE that resides in an ACL. Let's jump right in and start to better understand the structure members and how you might want to set them to change an object.

DWORD grfAccessPermissions, is for a combination of flags that will be used to dictate the permissions granted or denied to the object; permissions like EVENT_ALL_ACCESS, SYNCHRONIZE, GENERIC_ALL, READ_CONTROL, and so many more.

ACCESS_MODE grfAccessMode, will be for one of the enum values in the ACCESS_MODE enum and depending on if you are messing with a DACL or SACL, this member will mean different things. For DACLS, the flag will indicate if there will be deny or allow access rights in the ACL. For SACLs, this will indicate if audit messages for access attempts should be generated.

DWORD grfInheritance, is used to indicate if the ACE of the primary object can be inherited by other containers or objects. Setting something like NO_INHERITANCE will indicate that the ACE cannot be inherited.

TRUSTEE_A Trustee, is a structure that is used to indicate how the ACE should apply. ACEs can be applied to a user, group, or a Windows service.

Below are some enums and structs that are used within the EXPLICIT_ACCESS struct discussed above.

```
typedef enum _ACCESS_MODE {
 NOT_USED_ACCESS,
 GRANT_ACCESS,
 SET_ACCESS,
 DENY_ACCESS,
 REVOKE_ACCESS,
 SET_AUDIT_SUCCESS,
 SET_AUDIT_FAILURE
} ACCESS_MODE;
```

```
typedef struct _TRUSTEE_A {
  struct _TRUSTEE_A   *pMultipleTrustee;
  MULTIPLE_TRUSTEE_OPERATION  MultipleTrusteeOperation;
  TRUSTEE_FORM  TrusteeForm;
  TRUSTEE_TYPE  TrusteeType;
  union {
    LPSTR  ptstrName;
    SID  *pSid;
    OBJECTS_AND_SID    *pObjectsAndSid;
    OBJECTS_AND_NAME_A  *pObjectsAndName;
  };
  LPCH  ptstrName;
} TRUSTEE_A, *PTRUSTEE_A, TRUSTEEA, *PTRUSTEEA;
```

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed persistence via services
- Learned what APIs are related to services
- Explored how to create new services and modify existing ones
- Performed manual service hiding using SDDL

Module Summary

In this module, spent a little bit of time discussing how services can be used for our persistence needs. Throughout that discussion, we discovered what APIs are related to services. We saw that actions can be taken based on the service starting successfully, as well as failing. One of the final items covered was manually changing the permissions of a service to hide it from the user.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What language can be used to describe the security of a descriptor?

A SDDL

B ACL

C DACL

Unit Review Questions
Q: What language can be used to describe the security of a descriptor?

A: SDDL

B: ACL

C: DACL

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What language can be used to describe the security of a descriptor?

A SDDL

B ACL

C DACL

Tamarisk OffSecExam Exam solution

Unit Review Answers
Q: What language can be used to describe the security of a descriptor?

A: SDDL

B: ACL

C: DACL

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions

What command-line utility lets you view an object's security descriptor?

A cmd.exe

B sc.exe

C tasklist.exe

Unit Review Questions
Q: What command-line utility lets you view an object's security descriptor?

A: cmd.exe

B: sc.exe

C: tasklist.exe

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What command-line utility lets you view an object's security descriptor?

A cmd.exe

B sc.exe

C tasklist.exe

Unit Review Answers
Q: What command-line utility lets you view an object's security descriptor?

A: cmd.exe

B: sc.exe

C: tasklist.exe

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In this module, we will discuss how to persist using port monitors.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Define and discuss port monitors
- Understand APIs involved
- Explore its usefulness for persistence

Objectives

The objectives for this module are to define and discuss port monitors; what they are and what they are used for. Next, we will look at the APIs involved with the implementation of this method before we move into exploring how useful it might be for our persistence needs.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Port Monitors



What are port monitors?

Windows has two type of print monitors: language monitor and port monitor. Port monitors do what they say by monitoring a printer port and bridging the physical connection to the printer queue, which we see as a user.

Port Monitors

What exactly are port monitors? According to Microsoft, a port monitor acts like a bridge of sorts from user-mode to kernel-mode. The user-mode side comes from the spoolsv.exe image and it communicates with a port driver that resides in the kernel. The spoolsv.exe is a Windows service known as the Windows Print Spooler. The kernel port driver is what accesses the I/O port hardware. Port monitors are utilized to configure printer ports, and also manage them. As a user, you see the queue for a printer. The port is the physical connection that bridges that gap between the queue and the physical printer. Ports are assigned to port monitors using a Win32 API named *AddPrinter*. What we are doing, though, has nothing to do with printers or adding print job to a printer queue.

Reference:
https://docs.microsoft.com/en-us/windows-hardware/drivers/print/port-monitors

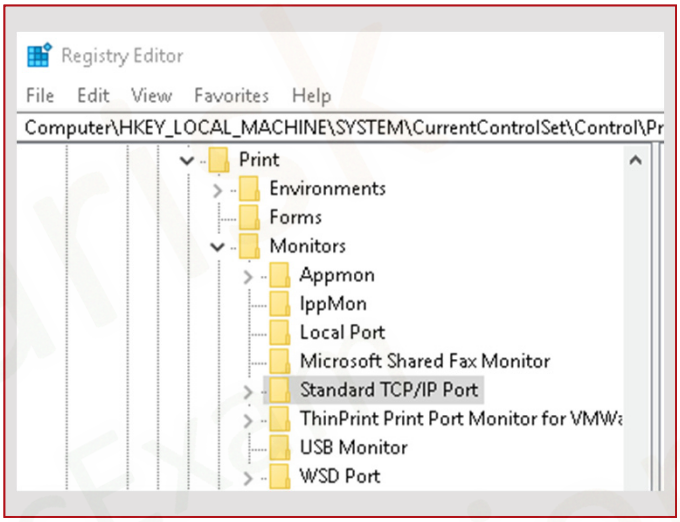
© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Abusing Port Monitors: The Registry

- Method one: Registry
- Key holds the port monitors in place
- Need local admin



Abusing Port Monitors: The Registry

There are two ways that we can leverage and abuse port monitors. The first method we are going to look at is using the Registry to modify the Print>Monitors hive. The full path to the key is **HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors** and the subkeys under it should be port monitors. We can manually view this using the regedit.exe GUI. Browsing to the key, we can select each subkey and view the values. Each one should have a DLL that it is using to monitor the port. Depending on your computer's configuration, you might see several subkeys like *Local Port*, or *USB Monitor*. Via the GUI, we can manually add a new subkey here and create our new port monitor. The new key that we create will have a new string value that will be the name of the DLL used to "monitor" the port. The catch is, the DLL being used for the "Driver" key value must reside in the C:\Windows\System32 folder. For this reason, we must have local admin privileges. The beauty behind this, though, is when the DLL gets kicked off, what comes back is SYSTEM. Not a bad deal at all. However, another downside to this method is that the system must be rebooted for this change to take effect. Probably not something you want to initiate on your own unless you know the user is gone for the day and will not notice. If you can determine that there will be a scheduled reboot to meet some corporate reboot policy, then that could work too.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

AddMonitor

AddMonitor

Used to install a local port monitor

Has a BOOL return type

```
BOOL
AddMonitor(
 _In_ LPTSTR pName,
 _In_ DWORD  Level,
 _In_ LPBYTE pMonitors
);


typedef struct _MONITOR_INFO_2 {
 LPTSTR pName;
 LPTSTR pEnvironment;
 LPTSTR pDLLName;
} MONITOR_INFO_2,*PMONITOR_INFO_2;
```


AddMonitor
The *AddMonitor* API is used when you need to create and install a port monitor on the local machine. The API will fail if your monitor does not match the architecture of the system you are targeting. This means that for a 64-bit system, your environment would be "*Windows x64*". The parameters are described below.

pName, is a pointer to a NULL-terminated string for the server that we are installing the port monitor.

Level, is for the version, but the only option the API will accept is 2, so 2 it is.

pMonitors, is a pointer to the MONITOR_INFO_2 struct that we have set up before the API call.

The MONITOR_INFO_2 struct is used by the API to identify the monitor that we are hoping to install. Here are the structure members.

pName is a pointer to a NULL-terminated string that will be the name of the monitor being installed.

pEnvironment is a pointer to a NULL-terminated string that indicates the environment that the monitor is being installed against. "Windows x64" and "Windows NT x86" are some
examples of strings acceptable for the environment.

pDLLName is a pointer to a NULL-terminated string that is used to indicate the name of the monitor DLL.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Abusing Port Monitors: AddMonitor



Method two: The API

Create our own DLL

Make it the monitor DLL

```
BOOL
CreatePortMonitor(void)
{
 MONITOR_INFO_2 mInfo2;
 // configure the struct members
 CHAR dllName[12]="NotEvil.dll";
 CHAR envName[12]="Windows x64";
 CHAR name[7] = "Sauron";
 mInfo2.pName = name;
 mInfo2.pEnvironment = envName;
 mInfo2.pDLLName = dllName;
 // call the function
 AddMonitor(...(LPBYTE)&mInfo2);
}
```


Abusing Port Monitors: AddMonitor

Let us talk about the second method for abusing port monitors, using the *AddMonitor* API. The API was discussed in detail on the previous slide so that we can cover a snippet of pseudo code here showing a possible implementation of it. The overall goal with the code is to bring our own DLL to the game and use it to tell Windows that it needs to use our DLL to monitor a port. We can create a small function that can execute this for us and just call it from *main()* or another function. What is happening in the code is that we prepare the MONITOR_INFO_2 struct for use and give it a variable name. Before we start initializing each struct member, we make a few variables that will be assigned to the appropriate struct member. The name of our DLL will be "NotEvil.dll" because we will not be doing anything evil here. The environment this will be installed on will be "Windows x64" because it is a 64-bit OS. Finally, the name of the port monitor itself will be "Sauron" because it will act as the all-seeing eye for the port. With that out of the way, we can assign them to the proper members of the struct and make our API call. Done!

A nice effect with this method is there is no reboot required since the changes made here are immediate. A downside to this, though, is that you will still need to be a local admin to pull this off, but good thing we know what to look for when trying to elevate our privileges.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Port Monitor Source Code

Source code review!

Port Monitor Source Code
Time to jump into the source code and explain it before you implement it on your own.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.2: Sauron



Implement a port monitor for persistence

Please refer to the eWorkbook for the details of the lab.

Lab 4.2: Sauron
Please refer to the eWorkbook for the details of this lab.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to introduce you to another persistence technique that can be done programmatically. There was only one API that we needed to learn, and it was not a complicated one at all. We just had to set up a few things inside a structure, and everything was ready to execute.

Tamarisk OffSecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Defined and described port monitors
- Looked at two ways to implement the method
- Discussed the permissions required

Module Summary
In this module, we defined port monitors and discussed how Microsoft intended for them to be used. Two ways of leveraging port monitors for persistence were explored from a manual method to a programmatic method. Lastly, we touched on a minor limiting factor for this method; the requirement to be a local admin.

Tamarisk OffSecExam Exam Solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What API is called to create a new port monitor?

A CreateNewMonitor

B AddMonitor

C AddNewMonitor

Unit Review Questions
Q: What API is called to create a new port monitor?

A: CreateNewMonitor

B: AddMonitor

C: AddNewMonitor

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What API is called to create a new port monitor?

A CreateNewMonitor

B AddMonitor

C AddNewMonitor

Unit Review Answers
Q: What API is called to create a new port monitor?

A: CreateNewMonitor

B: AddMonitor

C: AddNewMonitor

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In this module, we will discuss how to persist using Image File Execution Options (IFEO).

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Define what IFEO is
- Abuse the feature to gain persistence
- Discuss what permissions are required

Objectives
The objectives for this module are to define Image File Execution Options and abuse the feature to service our persistence needs. We will also talk about what permissions are required for pulling this off.

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What Is IFEO?



Image File Execution Options

IFEO is a Windows Registry key that enables the debugging or tracing of a process when it is started. The IFEO key is a great for developers so their application can be debugged, but it is also great for malware authors looking to persist on the target.

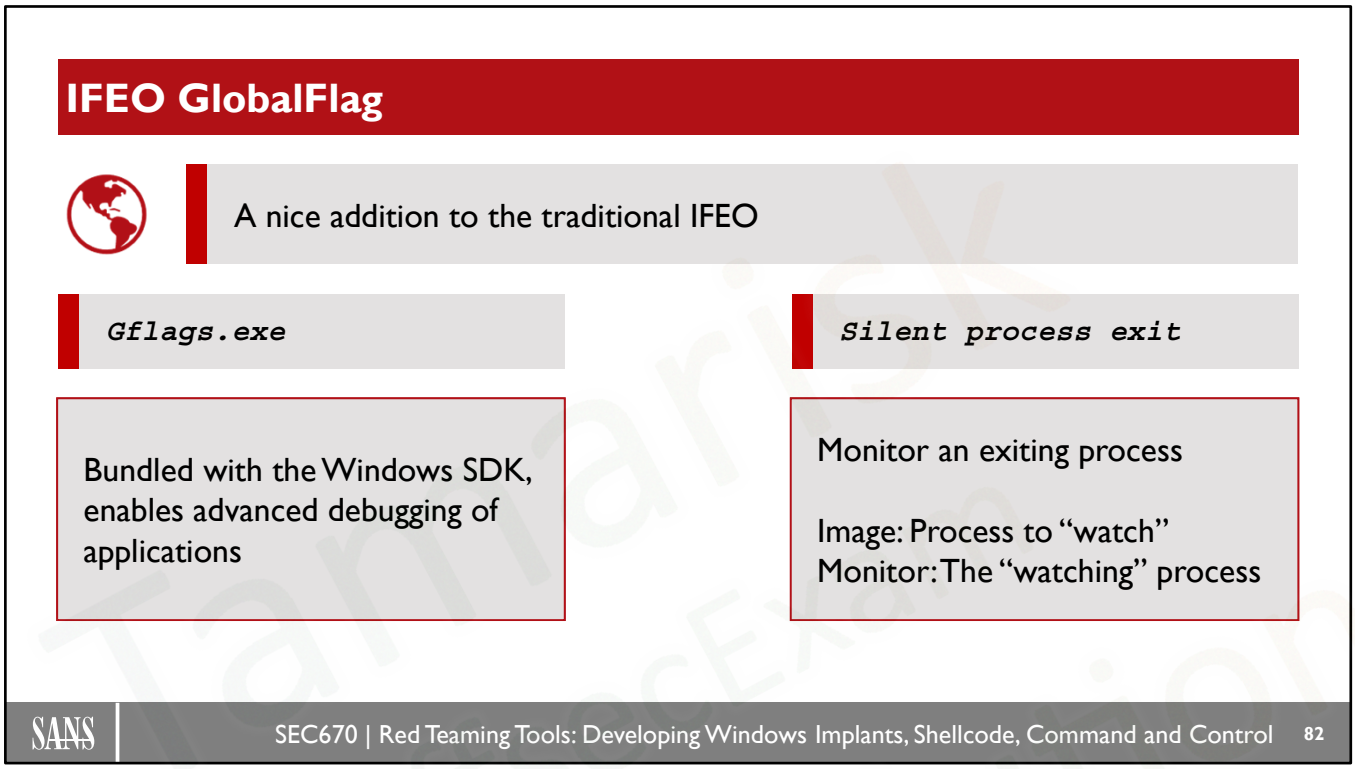
What Is IFEO?

IFEO stands for Image File Execution Options and comes in the form of a Windows Registry key. The idea behind it is to give some more options for when a process begins execution. The options can be any number of actions, like having a debugger launch when the process does, or having your implant run when the process does. If you are a Sysinternals fan, you can easily have Process Explorer or Process Monitor launch when the Windows Task Manager is launched. In fact, Process Explorer gives you this option natively by choosing Options->Replace Task Manager. As a developer, we can take advantage of this key as a method to monitor our application. As an implant developer, we can take advantage of this key as a persistence mechanism same as malware authors have been doing. This method really works great for EXE files, and not so much for DLLs.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



IFEO GlobalFlag

To use GlobalFlags, the SDK must be installed. After installation is completed, the gflags binary should be located at: **C:\Program Files (x86)\Windows Kits\10\Debuggers\x64**. MSDN describes the binary as one that can enable more advanced debugging and is used to turn on other indicators that other tools track. Gflags has the power to set system-wide debugging settings under the following Registry key: **HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\GlobalFlag**. However, for this to take effect you must reboot the system. With it, we can also affect settings for a specific application. To make this setting stick, we would change the **HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Image File Execution Options*ImageFileName*\GlobalFlag**.

But wait, there's more! If you were to scroll down toward the bottom of the MSDN page, you would see something called Silent Process Exit. This feature allows you to "monitor" the silent exit of a process. This means we get to specify what actions we want to take when a process exits. The gflags.exe utility will make the necessary changes to this registry key: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options*ProcessName*\GlobalFlag** and set the FLG_MONITOR_SILENT_PROCESS_EXIT flag. Simply replace "ProcessName" with the one that you want to "monitor" and then you have your persistence kicking off when the process exits.

To summarize the relationship between the setting and the Registry Key, see the below list.

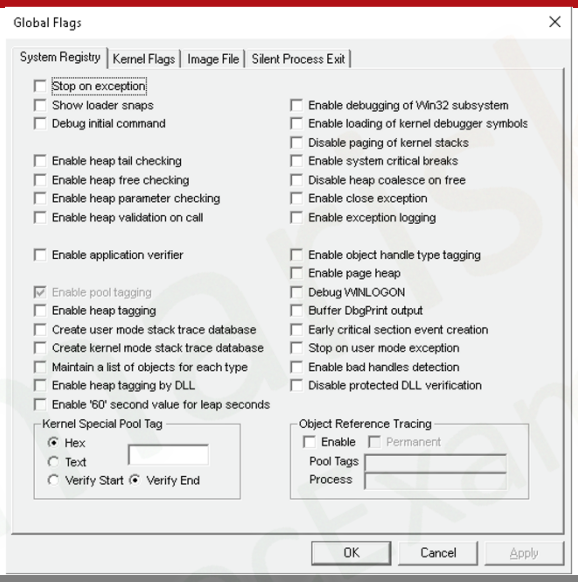
- Global settings: **HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\GlobalFlag**
- App-specific for all users: **HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Image File Execution Options*ImageFileName*\GlobalFlag**
- Silent exit, app-specific, for all users: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit*ImageFileName***
- App debugger: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options*ImageFileName*\Debugger**

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Running Gflags.exe



Running Gflags.exe

The GUI version of the gflags program looks similar to the screenshot on the slide. You can see the various tabs at the top of the window that are specific to categories like the Kernel, Image File, and Silent Process Exit. Feel free to explore the tool and the features that it provides. Once you have an understanding you can programmatically implement many of these items on your own.

Tanner Osk Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

GflagsX



A modern take on the original gflags.exe utility

Pavel Yosifovich created a new version of gflags that offers a great new look to the tool. Check out his repo for this and other awesome tools: https://github.com/zodiacon.

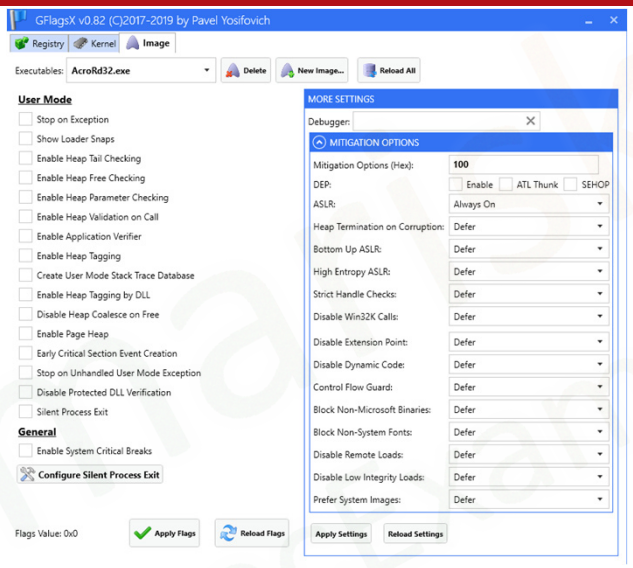
GflagsX
Pavel Yosifovich has been recreating several tools with better features and a nicer looking GUI. Pavel has been posting them publicly on his repo https://github.com/zodiacon. If you get a chance, check them out for yourself.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Running GflagsX



Running GflagsX

The GUI for Pavel's tool looks much nicer and more modern than the legacy tool. There are not as many tabs at the top of the window, but many of those features are consolidated. For example, the Silent Process Exit tab and its options are located under the Image tab for GflagsX.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Manual Implementation



reg.exe

Done with interactive shell

Could be done with APIs

```
reg add HKLM\...\Image File Execution
Options\ProcessName /v GlobalFlag /t
REG_DWORD /d 512

reg add
HKLM\...\SilentProcessExit\ProcessNam
e /v ReportingMode /t REG_DWORD /d 1

reg add
HKLM\...\SilentProcessExit\ProcessNam
e /v MonitorProcess /d
"C:\Path\To\implant.exe"
```


Manual Implementation

It is possible to manually make the Registry keys using the reg.exe command-line utility. The examples on the slide demonstrate how we could implement this if we were using an interactive shell on the target. The other option is to do this programmatically with the appropriate Win32 Registry APIs we covered earlier in the course, like ***RegOpenKeyExA***. After obtaining a handle to a key, we can then call ***RegCreateKeyExA*** so we can create or modify a key, and then call ***RegSetValueEx*** so we can write the values to the proper keys, etc. Needless to say, there is more than one way that we can go about getting this done.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Abusing IFEO

1010
1010

Take advantage of what is given.

We can take advantage of IFEO to achieve our own persistence goals. To do this, it would be beneficial to choose an application that starts early in the boot process, or one that we are certain will execute either by the user or the system.

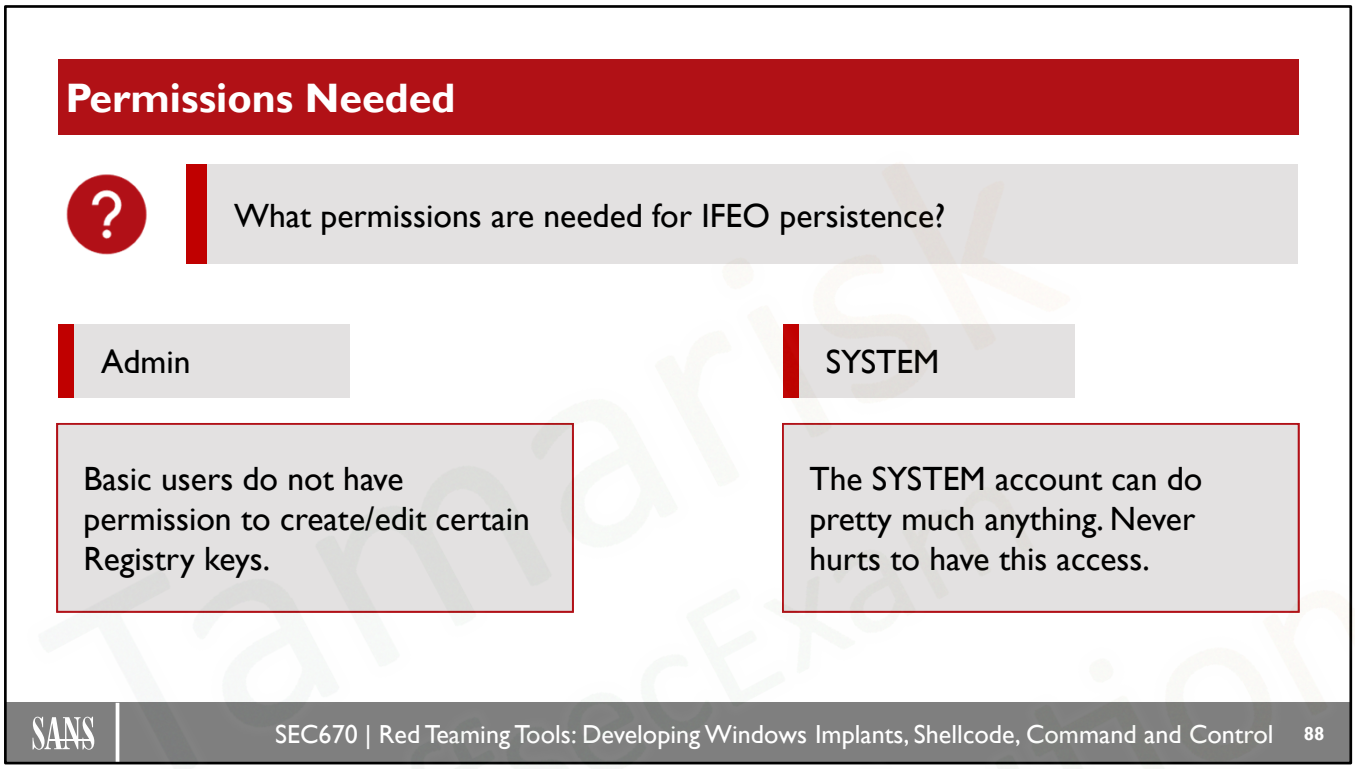
Abusing IFEO

Since we are interested in persisting on the target, we are going to abuse this option. The trick is choosing the best target process to do this against. We should choose one that kicks off when the system is starting up. A good possibility that fits into that role would be userinit.exe as it starts when the system does. You can leverage your knowledge of Windows internals for other options. There could also be third party applications that start when the system does, so they can serve as good options too. Perhaps you know that a user opens a particular process as part of their daily work routine; we can leverage that as well. In the end, pick something that you do not have to trigger manually.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



Permissions Needed

Sadly, if we only have permissions as a basic user, then we will be denied access when trying to modify the HKLM Registry keys needed for the IFEO persistence method. Since we have already discussed a few ways that we can escalate our privileges, there is no need to bring that discussion here, but we can use any technique we can to become Admin. We might be lucky and find the path to Admin and perhaps even SYSTEM is trivial. Regardless of the path to gain higher privileges, now that we have them, we can modify the Registry as we see fit. One thing to remember is to properly clean up should you need to get off the target. We would want to put things back where they belong and leave the system how it was before we came on the box, so disabling IFEO for an image would need to be done. The logic could be built into an uninstall command, or similar command, and sent to the implant so that when sent, it will go through and reverse many of the actions taken, like Registry key modifications.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

IFEO Persistence

Source code review!

IFEO Persistence
Time to jump into the source code and explain it before you implement it on your own.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.3: IFEOPersisto



Demonstrating persistence via IFEO keys

Please refer to the eWorkbook for the details of the lab.

Lab 4.3: IFEO Persistence
Please refer to the eWorkbook for the details of this lab.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to become familiar with the persistence method of Image File Execution Options and the two variants: process start and silent exits.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Defined IFEO
- Abused IFEO manually and programmatically
- Observed limiting factors such as permissions for IFEO

Module Summary

In this module, we defined IFEO and discussed how Microsoft intended for it to be used. We also moved into discussing another variant of IFEO, which was the SilentProcessExit option. From there we looked at how we can abuse both variants while at the same time observing permissions issues if not being done with elevated permissions like Admin or SYSTEM.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What registry key could be used to watch for process termination?

A SilentProcessExit

B Debugger

C DebuggerProcessExit

Unit Review Questions
Q: What registry key could be used to watch for process termination?

A: SilentProcessExit

B: Debugger

C: DebuggerProcessExit

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What registry key could be used to watch for process termination?

A SilentProcessExit

B Debugger

C DebuggerProcessExit

Unit Review Answers
Q: What registry key could be used to watch for process termination?

A: SilentProcessExit

B: Debugger

C: DebuggerProcessExit

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What registry key could be used to watch for process creation?

A SilentProcessExit

B IFEO

C DebuggerProcessExit

Unit Review Questions
Q: What registry key could be used to watch for process creation?

A: SilentProcessExit

B: IFEO

C: DebuggerProcessExit

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What registry key could be used to watch for process creation?

A SilentProcessExit

B IFEO

C DebuggerProcessExit

Unit Review Answers
Q: What registry key could be used to watch for process creation?

A: SilentProcessExit

B: IFEO

C: DebuggerProcessExit

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 4

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

In this module, we will discuss why you would need to drop something to disk, where to drop, cleaning up, and more.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Introduce WMI and its purpose
- Discuss WMI events and subscriptions
- Explore triggers for specified actions

Objectives

The objectives for this module are to start off with an introduction to the Windows Management Instrumentation (WMI) and its intended usage. We will also get into WMI events, filters, and subscribing to receiving events of interest. Lastly, we will look at the triggers to kick off an event, like the creation of a new process, a new logical drive being loaded, or a failed logon attempt. WMI abuse is an active technique still being used by nation state actors today. WMI attacks are not new at all and yet they are still being overlooked by cybersecurity defenders and AV/EDR solutions (although these groups are getting better at noticing these attacks).

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What Is WMI?



The Windows Management Instrumentation (WMI)

The Windows OS must manage the data and operations not only for itself but also for remote systems. WMI is the method that instruments this management and is designed for developers and administrators to use with ease via C++ development or PowerShell scripting.

What Is WMI?

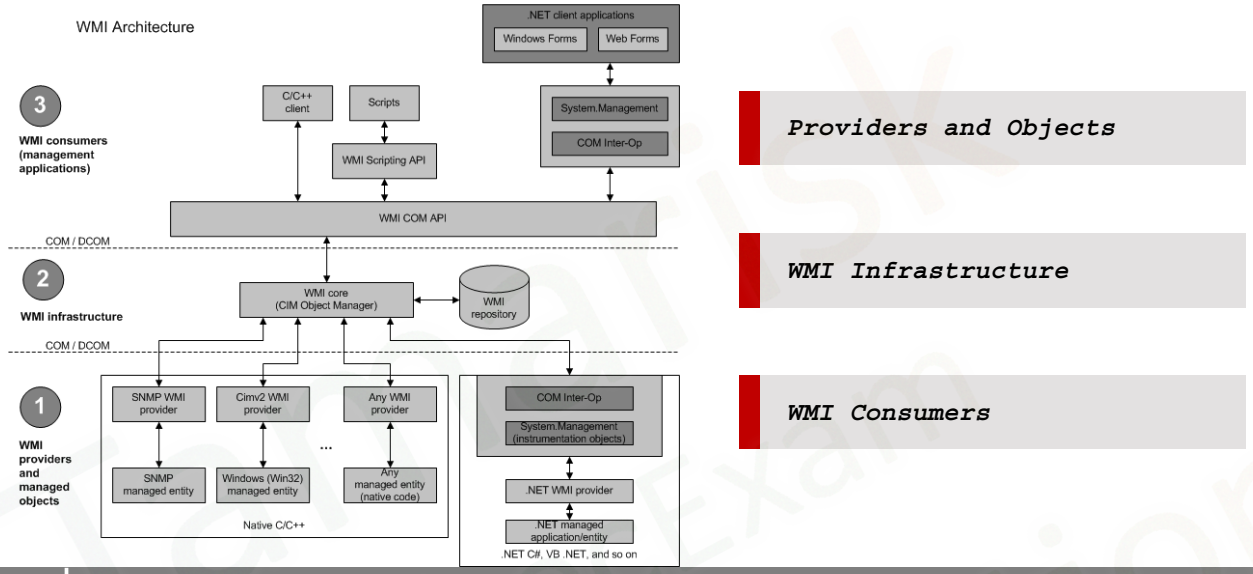
Windows generates an enormous amount of data and executes many operations. Because of this, it needs something to assist with the management of said data. Enter the Windows Management Instrumentation (WMI). WMI enables developers and administrators to query data that might be held in the WMI database. For administrators, WMI enables you to connect to and query large numbers of remote computers for whatever data you might need. In addition, you can invoke WMI methods to execute something locally or against a remote system, like creating a new process that may have crashed and did not start up again, or you need to create some new Registry keys. WMI becomes even easier to manage with PowerShell natively supporting WMI with several cmdlets, like Get-WmiObject for one. For developers, WMI is perfect and catered for our C++ development. We can create WMI providers that provide data to consumers, and we can also create our own consumers to subscribe to events, or data, that existing providers are pushing out. Before we get into events and filters, we need to understand a bit more about the back end of WMI.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

WMI Architecture



WMI Architecture

The graphic on the slide is the representation of the architecture of WMI, according to MSDN online documentation. You can follow this in stages starting at the very bottom with area 1. Area 1 is where WMI providers are doing their job of providing data that can be stored in the WMI repository, and area 2 is where data is sent to WMI consumers in area 3. One very interesting WMI provider is the Win32 provider that can give consumers a list of processes. This could prove to be another method used for process enumeration—it's just noisy. Area 2 houses the core of WMI, and its repo can store static data to be queried by consumers. This data is persistent across reboots and offers an interesting persistence mechanism that we will explore. As you might have noticed on the graphic, consumers can be created by C++ programs or scripting with PowerShell, which makes it extremely easy to utilize. In the end they all go through COM interfaces, but regardless, they enable us to create a consumer that can query events given by providers. Because there can be so many events, we must filter them by creating a filter, and filters are created using a specific query language named WQL, or the Windows Query Language. WMI adheres to a universal standard known as CIM, the Common Interface Model.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

What Is CIM?



The Common Information Model (CIM)

The CIM is an industry standard that is used by WMI to represent various items, like systems, processes, devices, and more. CIM is object oriented and gives the look and feel of a C++ class. CIM gives us three levels of classes: Core, Common, and Extended.

What Is CIM?

Standards typically have models to go by and WMI is no exception because it follows the industry standard called the Common Information Model (CIM). The CIM is used to represent data in a uniformed way, data such as systems, processes/applications, networks, devices, and more. There are numerous existing CIM classes, but developers can create new ones that can be used to represent completely customized products like networked ovens or HVAC units. These classes are very similar to that of C++ classes developers would create. At its core, CIM provides various levels of classes—three of them to be exact. The first one is Core, which is used to represent the managed objects for managed systems. Next is the Common class. These classes are an extension of the Core classes and are not tied down to any specific technology. Finally, we have the Extended class, which unlike the Common class is specific to certain technologies like operating systems. One example of a Windows Extended class is Win32_ComputerSystem. These levels of classes make CIM cross-platform, except for the Extended class, of course.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

WMI and CIM Schemas

Classes can be grouped together into what are called schemas.

CIM Schema

Classes start with CIM_ and provide the definition for the Core and Common classes. Developers can create their own as well.

Win32 Schema

Classes start with Win32_ and provide the definitions for the Extended CIM class specific for the Win32 environment. Developers can create their own here as well.

WMI and CIM Schemas

WMI and CIM classes are often grouped together to form what are called schemas, and they are typically specific to certain managed objects. Specific to the Windows SDK, there are two schemas in place. The first schema is the CIM schema, and its classes are easy to identify as they begin with the prefix CIM_. The Core and Common classes are defined as part of the CIM schema. There are many CIM classes, like CIM_Battery, CIM_Memory, CIM_ComputerSystem, CIM_Directory, etc. Perhaps the more interesting schema is the Win32 schema. It is here that the Extended class is defined that provides classes specific to the Win32 environment. Win32 is also a provider, but more on that shortly. Developers can create their own schemas, classes, etc. in either the CIM schema or the Win32 schema.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Win32 Provider and Classes



The provider provides all data specific to Windows.

Class Name	Description
Win32_Account	Information about user and group accounts
Win32_LoggedOnUser	Relates to session and user accounts
Win32_OperatingSystem	The Windows OS installed on the system
Win32_Process	A process on the system
Win32_Registry	The system registry on the system
Win32_Service	A service on the system
Win32_Thread	An executing thread in a process

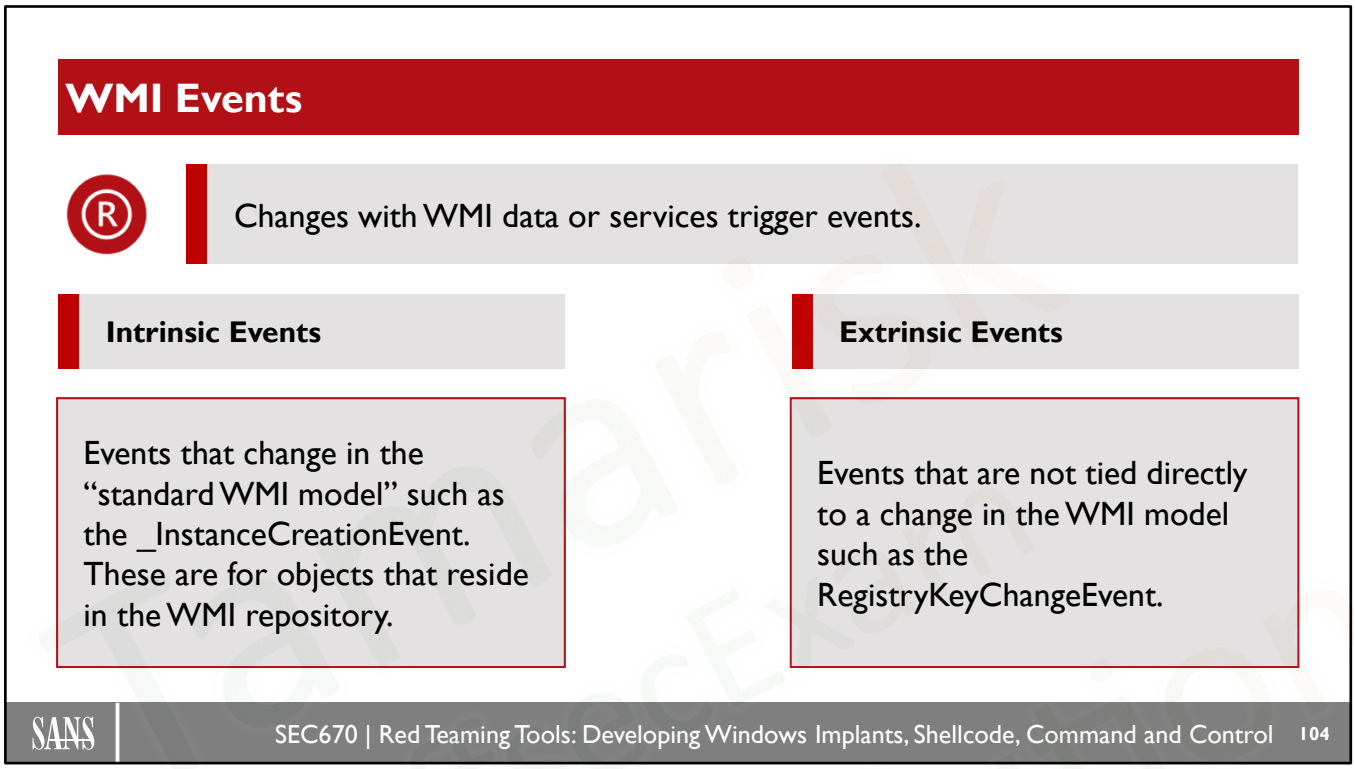
Win32 Provider and Classes

The Win32 provider is where we can fetch any data that might relate to the Windows operating system. The data can be static or dynamic, such as a list of running processes. The classes shown on the slide are but a small subset of classes that are defined by the Win32 provider. As you can see, there are several classes that can be used for very specific purposes. We can use specific classes for recon about users of the system, who is logged on like an Admin or Domain Admin, what processes are running, and more. Several classes have methods that can be invoked like the Win32_Process class. We could use it to create a new instance of the class, which in this case would be a new process. To narrow things down and cut out some of the noise, we could create filters to focus on certain events from a specific class. We can filter on any number of items, like filtering on processes using the Win32_Process class, we can filter for account logon attempts, etc. Once a filter is created, it can be used to trigger events if the filter criteria is met.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam



WMI Events

Before we would ever create a subscription to an event, we need to determine what type of event we will be receiving. WMI has intrinsic and extrinsic events with each type being quite different. The intrinsic events are created when there is some kind of change in the WMI model. The WMI repository holds objects inside of it, and WMI providers can create, delete, or modify the objects. An example of such objects are classes, instances of classes, or namespaces. When WMI providers make changes to objects, an intrinsic event occurs. To these changes, WMI conducts a polling operation like *_InstanceCreationEvent* for when an instance of a class is created. Another example of an event could be *_InstanceModificationEvent* for when an instance is modified.

Extrinsic events are not triggered by changes to WMI repository objects. Specifically, when an event cannot be tied directly to a change in the data model it must be an extrinsic event. One such example would be events that are related to the Registry. The *RegistryKeyChangeEvent* is an extrinsic event that can notify consumers that a certain Registry key has been changed. This could possibly be useful for protecting your Registry persistence mechanism.

© 2024 Jonathan Reiter

© SANS Institute 2024

Filtering Events Using WQL



Windows Query Language

Extrinsic events can be queried normally

Intrinsic events must be polled at some defined interval

```
Data Query
SELECT * FROM Win32_NTLogEvent WHERE logfile
= 'System' AND EventCode = '4625'

Event Query
SELECT * FROM __InstanceCreationEvent WITHIN
5 WHERE TargetInstace ISA "Win32_Process" AND
TargetInstance.Name = 'notepad.exe'

Schema Query
SELECT * FROM meta_class WHERE __this ISA
"Win32_Process"
```


Filtering Events Using WQL

How are events filtered? The Windows Query Language, WQL, might look similar to that of the Structured Query Language, SQL, and that is because WQL is a subset of SQL. There are certain types of queries that we can create such as Data queries, Event queries, or Schema queries. We can start off our query using the *SELECT* statement. The *SELECT* statement will be specific to the type of query, Data, Event, or Schema. It takes quite a bit of research to determine what classes you will need to query. The determination must also be made if the query will be for intrinsic or extrinsic events. Remember, intrinsic events require polling like the Event query example on the slide and polls must be done on a frequency.

The event query example on the slide is polling for *__InstanceCreationEvents* every 5 seconds. The new instance that is created must be an instance of the *Win32_Process* class with a process name of notepad.exe. These types of queries can get very complex rather quickly but yet specific for your persistence needs.

The Data query example is making a query from the *Win32_NTLogEvent* class. The logfile being chosen is the System logfile and it is looking for a specific event code of 4625, the event for failed logon attempts.

The last example query is for a Schema query to better understand the *Win32_Process* class definition. Schema queries are used for understanding the definition of a class. This type of query is quite different because we do not rely on instances of a class being present. Schema queries only support the *SELECT* * statement unlike the other query types. Also, by saying *meta_class* we indicate that this is a schema query being executed.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Testing WMI Queries

```
Get-WmiObject __EventFilter -Namespace root\subscription
Get-WmiObject __EventConsumer -Namespace root\subscription
Get-WmiObject __FilterToConsumerBinding -Namespace root\subscription

Get-WmiObject -Query "select * from Win32_Process where name='notepad.exe'"
Get-WmiObject -Query "select * from win32_ntlogevent where eventcode=4625 and \
logfile='security' and message like %alice%"


Can trigger logon events using smbclient \\\\#{target}\\C$ -U alice badpassword
```


Testing WMI Queries

PowerShell offers developers with perhaps the easiest method for testing WMI queries, thus saving us from having to develop what could be hundreds of lines of code for queries that do not work. Thanks to the *Get-WmiObject* cmdlet and we can pass a crafted query to the cmdlet's -Query parameter.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Event Triggers



Events triggering actions

Filtering for notepad.exe

Consuming the event and triggering action

```
Event Query
SELECT * FROM __InstanceCreationEvent WITHIN
5 WHERE TargetInstace ISA "Win32_Process" AND
TargetInstance.Name = 'notepad.exe'

Event Consumer
Set-WmiInstance -Class
CommandLineEventConsumer -Namespace
"root\subscription" -
Arguments@='Consumer';ExecutablePath='C:\evil
.exe';CommandLineTemplate='C:\evil.exe'
```


Event Triggers

When it comes to event triggers, nothing has been triggered yet. The only thing done so far is filtering out a specific event. The next thing is to set up our Event Consumer that will contain the action we wish to execute. When registering the consumer, there is an Action block that must be filled out. Specifically, there are five Consumer classes that we can implement. *ActiveScriptEventConsumer*, *CommandLineEventConsumer*, *LogFileEventConsumer*, *NTEventLogEventConsumer*, *SMTPEventConsumer* are the five consumers that we could use, but perhaps the most interesting to us are the *CommandLineEventConsumer* and the *ActiveScriptEventConsumer* classes. The *CommandLineEvent* is an interesting class because it will create a process when the filter it is being bound to is triggered. One of the properties of the class is the *CommandLineTemplate* that specifies the binary that is to be executed. There is also the *ExecutablePath* property that is used to specify the absolute path for the executable. The filter on the slide polls from the *__InstanceCreationEvent* class for the notepad.exe process every five seconds. The instance that will consume that event is shown using PowerShell for easier readability. It will execute the evil.exe file whenever the notepad.exe process is created.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Detecting WMI Attacks



Sysmon can be configured to detect WMI attacks.

The abuse that has been done with WMI can be detected using several tools, one of them being Sysmon. The configuration can catch the Event Filters, the Event Consumers, and our bindings of filters and consumers.

Detecting WMI Attacks

There are several methods and tools for detecting these style of attacks, so be careful before you implement this method. One of the more popular tools today is one put out by Microsoft called Sysmon. It has proven to be very formidable, and when configured correctly, can make our job much more challenging. One of the configurations for Sysmon is for detecting WMI attacks, which should be on by default, making this easier for the defenders and harder for us attackers. As an example, Sysmon would let you know when a *WmiEventConsumer* event has been created. Despite the logs being created, the decision must be made to determine if the event was malicious. Logs are one thing, but the categorization of the event is another.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discovered that WMI is impressively powerful
- Saw how events can be filtered and consumed to trigger actions
- Discussed how it can be used for persistence and elevation

Module Summary

In this module, we took a deep dive into WMI, how it is designed, how events are created, how events can be consumed, and how we can trigger process creation when a certain event is kicked off. WMI, aside from being an amazing tool for sysadmins, is also an amazing tool for attackers. Not only can we persist, but we can elevate from Admin to SYSTEM at the same time since our action in our consumer will kick off as SYSTEM.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What WMI class holds together the event filter and the event consumer?

A FilterToConsumerBinding

B Win32_Process

C CommandLineEventConsumer

Unit Review Questions
Q: What WMI class holds together the event filter and the event consumer?

A: FilterToConsumerBinding

B: Win32_Process

C: CommandLineEventConsumer

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What WMI class holds together the event filter and the event consumer?

A FilterToConsumerBinding

B Win32_Process

C CommandLineEventConsumer

Unit Review Answers
Q: What WMI class holds together the event filter and the event consumer?

A: FilterToConsumerBinding

B: Win32_Process

C: CommandLineEventConsumer

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What types of events must be polled at some interval?

A Extrinsic

B Intrinsic

C All of the above

Unit Review Questions
Q: What types of events must be polled at some interval?

A: Extrinsic

B: Intrinsic

C: All of the above

Tamarisk OffSecExam Exam solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What types of events must be polled at some internal?

A Extrinsic

B Intrinsic

C All of the above

Unit Review Answers
Q: What types of events must be polled at some interval?

A: Extrinsic

B: Intrinsic

C: All of the above

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
Lab 4.1: Persistent Service
Port Monitors
Lab 4.2: Sauron
IFEO
Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

Welcome to the bootcamp for Section 4! The challenges during the bootcamp will be very challenging but have fun with them and do not hesitate to reach out for assistance, guidance, or if you have any questions.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Bootcamp

- NotInService
- InitToWinit
- OhMyWMI
- CustomShell

Bootcamp

The bootcamp challenges today will really test your knowledge of Windows APIs. Services are important and as such, the first challenge is about services. The "Not in service" challenge requires you to create, install, and then hide your own service. The second challenge is about the *AppInit* method where you create the AppInit_DLLs key accordingly to get your DLL payload to execute without getting stuck in the infinite loop mentioned during that section. The third challenge is about using WMI to programmatically establish your foothold. You can use PowerShell for your initial testing or to get it going but the final product should be done in C or C++ as a compiled binary.

The last challenge of the day is to combine everything you have learned so far into a baby implant. The baby implant should be able to do full recon, inject into other processes, persist across reboots, create a log file, etc.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.4: NotInService

- Develop the code for your service application.
- Develop the code to install your service.
- Develop the code to hide your service.

Lab 4.4: NotInService
For "Not in service," you will be developing a custom service to be installed on the target system. Once done, you will also create the code that will install your service. Finally, after the service has been installed, you will create the code to hide it. There are a couple of ways to go about accomplishing this challenge, but the most straightforward is to use the proper Win32 APIs. Your service does not have to perform anything specific as long as it runs successfully and hidden is the main point.

Have fun!

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.5: InitToWinit

- Create the AppInit_DLLs key.
- Use an existing malicious DLL or build you own for this.
- Watch out for infinite loading situations.

Lab 4.5: InitToWinit

You can get carried away with this challenge quickly if you are not paying close attention to the details. If you need to, refer back to the Registry keys section when we discussed this method and what it can be used for. The main goal here is to become familiar with the code to implement the method, while at the same time not getting caught in a never-ending loop situation. The DLL to use should be one that you have either created already or one that you create specifically for this lab. Avoid using DLLs made by msfvenom, but for a learning experience, try one out and see what happens.

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.6: OhMyWMI

Create a permanent subscription based on system uptime.

The trigger should execute your persistence tool.

Lab 4.6: OhMyWMI

WMI was discussed in depth earlier in this section, and now it is time to put your foundations to the test. For this challenge, you are to create an installer that will establish your persistence based on system uptime. PowerShell will be your friend for this challenge as you should use it to test your queries. Once you have your query finalized, you can implement it programmatically in your code. If you finish early, you can implement more options that change what events are being subscribed to and what the trigger is.

Tamarisk OffSecExam Exam Solution

© 2024 Jonathan Reiter

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 4.7: CustomShell

- Create a basic shell.
- Implement features covered in this section.
- Implement thorough error checking.

Lab 4.7: CustomShell
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024
https://linktr.ee/offsecexam

670.5

Enhancing Your Implant: Shellcode, Evasion, and C2

Tanarisk OffsecExam Exam solution

© SANS Institute 2024
© 2024 Jonathan Reiter. All rights reserved to Jonathan Reiter and/or SANS Institute.
https://linktr.ee/offsecexam

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE (“SANS INSTITUTE”) FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this CLA. Courseware means all printed materials, including course books and lab workbooks, slides or notes, as well as any digital or other media, audio and video recordings, virtual machines, software, technology, or data sets distributed by SANS Institute to User for use in the SANS Institute course associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and User and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium, whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. User may not sell, rent, lease, trade, share, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute. Additionally, User may not upload, submit, or otherwise transmit Courseware to any artificial intelligence system, platform, or service for any purpose, regardless of whether the intended use is commercial, educational, or personal, without the express written consent of SANS Institute. User agrees that the failure to abide by this provision would cause irreparable harm to SANS Institute that is impossible to quantify. User therefore agrees to a base liquidated damages amount of $5000.00 USD per item of Courseware infringed upon or fraction thereof. In addition, the base liquidated damages amount shall be doubled for any Courseware less than a year old as a reasonable estimation of the anticipated or actual harm caused by User’s breach of the CLA. Both parties acknowledge and agree that the stipulated amount of liquidated damages is not intended as a penalty, but as a reasonable estimate of damages suffered by SANS Institute due to User’s breach of the CLA.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. A written amendment or addendum to this CLA that is executed by SANS Institute and User may accompany this Courseware.

SANS Institute may suspend and/or terminate User’s access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User, (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

The Apple® logo and any names of Apple products displayed or discussed in this book are registered trademarks of Apple, Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

Governing Law: This CLA shall be governed by the laws of the State of Maryland, USA.

Courseware licensed to User under this CLA may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department (“OFAC”), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce (“BIS”). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulation. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

All reference links are operational in the browser-based delivery of the electronic workbook.

© SANS Institute 2024

https://linktr.ee/offsecexam

SEC670.5 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control

SANS

Enhancing Your Implant: Shellcode, Evasion, and C2

© 2024 Jonathan Reiter | All Rights Reserved | Version J01_05

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control: 670.5
Welcome to Section 5 of SEC670. This section is about what you can do to enhance your implant to execute shellcode, evade antivirus, and communicate with C2 infrastructure. Let's get started.

Tamarisk OffsecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Table of Contents	Page
Custom Loaders	3
Lab 5.1: The Loader	17
Unhooking Hooks	26
Lab 5.2: UnhookTheHook	55
Bypassing AV/EDR	63
Calling Home	77
Lab 5.3: No Caller ID	110
Writing Shellcode in C	119
Bootcamp	136
Lab 5.4: AMSI No More	138
Lab 5.5: ShadowCraft	139

This page intentionally left blank.

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 5

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- **Enhancing Your Implant: Shellcode, Evasion, and C2**
- Capture the Flag Challenge

Custom Loaders
Lab 5.1: The Loader
Unhooking Hooks
Lab 5.2: UnhookTheHook
Bypassing AV/EDR
Calling Home
Lab 5.3: No Caller ID
Writing Shellcode in C
Bootcamp
Lab 5.4: AMSI No More
Lab 5.5: ShadowCraft

In this module, we will cover how to create and use a custom loader as well as how to implement a loader in your custom shell you have been working on throughout the course.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Understand what a loader is
- Understand its use cases
- Implement a custom loader

Tamarisk OffSecExam Exam solution

Objectives

The objectives for this module are to discuss at a high level what a loader is, how/when to use one, and then finally how to implement a custom loader.

© SANS Institute 2024

https://linktr.ee/offsecexam

What Is It



Load what you want, when you want

A custom loader can be used to load PE images, among other items, and execute them. This custom loading can be done locally within the current address space of a process or remotely by crossing process boundaries.

What Is It

What exactly is a loader? What is a custom loader? Is there even a difference? Not really. One could jump in and argue that a loader simply maps a PE image into memory and executes it. A custom loader, on the other hand, was built with evasion in mind in addition to mapping and executing a PE image. Custom loaders can even load a custom PE images. Custom PE images can be made to not look like traditional ones. Various headers can be moved around, there can be multiple headers as well throughout the image. This can be done to throw off scanners that might only be parsing PE files in a traditional sense. Earlier in the course, you learned in detail about the traditional layout of a PE image. The traditional layout is what most tools depend on for parsing. Also, the Window loader is also depending on this traditional layout. This layout can be modified though, and it can be used to confuse tools that parse PE images. This is where a custom PE loader could come into play.

© SANS Institute 2024

https://linktr.ee/offsecexam

An Early Start



Move over LoadLibrary, I got this.

Over a decade ago, Stephen Fewer introduced to the world his own custom version of LoadLibrary, the API that will load a DLL from disk into memory. He created the art of reflective loading images. This was a tremendous capability because images no longer had to be on disk.

An Early Start

A long time ago, in a computer far far away, Stephen Fewer took the job away from LoadLibrary and released his own custom loader dubbed reflective loading. Reflective programming as a technique is not new, but what was new is being able to load an executable image, say a DLL, from a raw buffer in memory to then call its DllMain function or an exported function. It caught on like wildfire and you can use it for injecting a DLL into your own process or even across process boundaries into a target process with each method having their own pros and cons. As time moved on, the technique advanced, other people enhanced it or ported it over to other languages. Before we move on with creating our own custom loader, let us dive in a little bit at the traditional technique: reflective DLL injection.

© SANS Institute 2024

Let Your Reflection Show



Reflective DLL Injection (RDI)

With Reflective DLL Injection, or RDI, the source DLL is manually mapped into the target's virtual address space. This means that the full path of the DLL will not be written, thus making this technique a stealthier one. It also means that we are now the system loader.

Let Your Reflection Show

You might be wondering why we need another technique to get a target process to load a DLL. Well, what if you do not want a malware analyst, or students taking FOR610/FOR710, to find the absolute path to your DLL? Or you do not want you DLL to be listed in the list of loaded modules? What if you want to process the injected DLL in its raw form like coming in from a socket? Stephen Fewer created this technique because the industry needed something that would provide us with a stealthier way of injection. If you look at the GitHub repo down below, he talks about using the concept of reflective programming to load a library from memory. Reflective programming is when a program can inspect itself, called introspection, or that it can modify itself or its own structures like a library loading itself into memory. With reflection, you are doing a lot of the heavy lifting that would normally be handled by the system loader. Here, with RDI, you handle a large chunk of what the system loader would normally do.

Because we will be acting like a lightweight system loader, we need to be able to process and read the PE headers and map them all into the target address space without error. Basically, what we are doing is implementing the ***LoadLibrary*** API. This is another reason to become intimately familiar with the PE32/PE32+ structure. Any time you are rolling your own loader, you are relying less on the system and thus flying below the radar a bit more than usual. Flying below the radar—stealth—is our goal, of course.

Reference:
https://github.com/stephenfewer/ReflectiveDLLInjection

© SANS Institute 2024

https://linktr.ee/offsecexam

RDI: LoadLibrary



Rolling your own LoadLibrary

To help understand Reflective DLL Injection better, it is best to start with the internal mechanisms of the LoadLibrary API. The system loader can spoil you with all that it does, but knowing what it does can set you free.

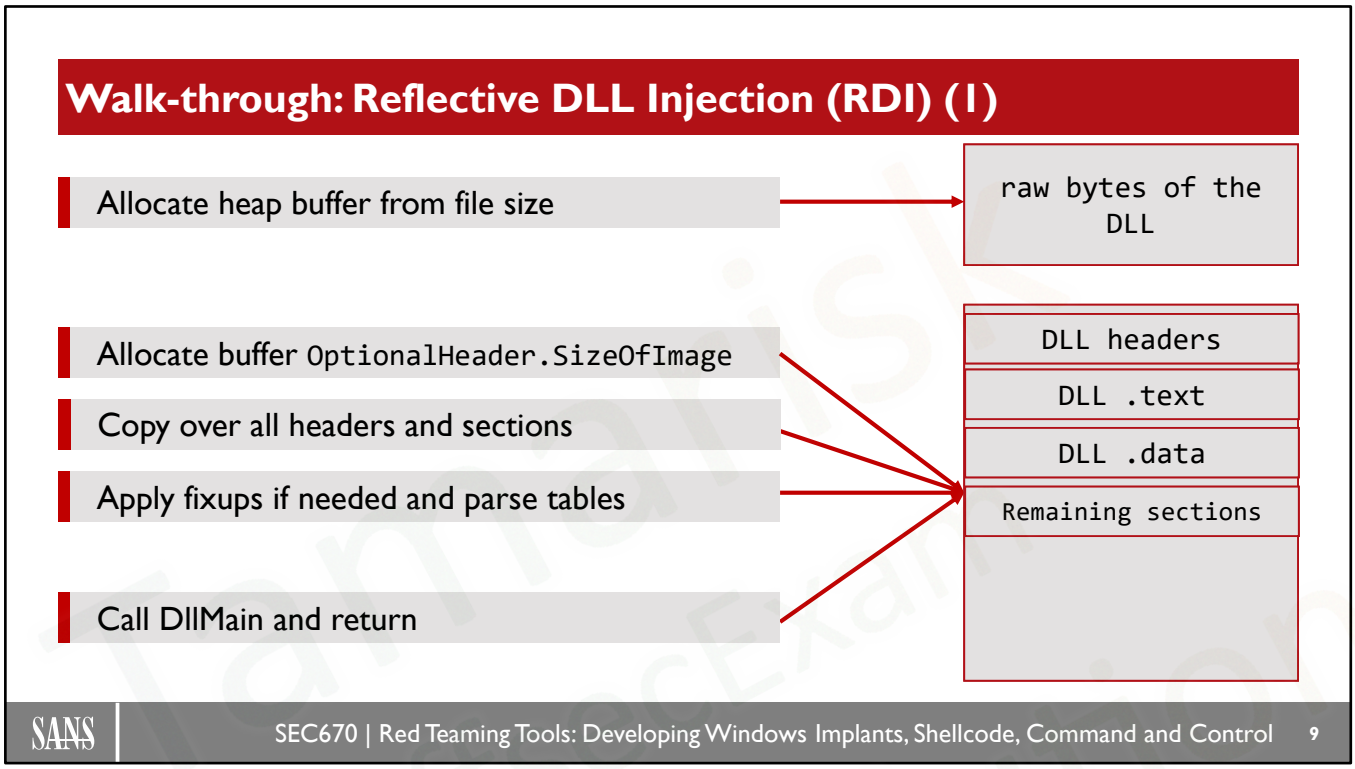
RDI: LoadLibrary

Rolling your own version of the ***LoadLibrary*** API can be thought of manually mapping a DLL into memory. We can give our own version of the API its own name, so we could call it ManualLoadLibrary to be clear. We can explore several versions, but we will stick with the most common and perhaps the simplest. Also, we will assume the DLL is on disk, which makes learning this method a little bit easier. One of the first tasks is to use the ***CreateFileA*** API to grab a handle to the DLL. With the DLL handle, we can call the ***ReadFile*** API and read the raw data of the DLL into a memory buffer. The address of the memory buffer is going to serve as the base address of the DLL. Having the base address, we can start to parse the various structures of the PE headers. Other important items are the section headers—that way we can grab the size of the image from *OptionalHeader.SizeOfImage*. The image size is used to allocate another chunk of memory using the ***VirtualAlloc*** API. The system loader will try to allocate that chunk of memory using the module's *OptionalHeader.ImageBase* because that value indicates the preferred base address. If that allocation cannot be done, it uses whatever the system allocates for it. If that is the case, relocations and fixups will take place. In other words, the base is being relocated and everything else must be adjusted according to the new base. This is the reason why RVAs are so important to understand. One thing that can make our own version mimic the real one more closely is to keep the memory permissions for each section since not every section must be marked as Read, Write, and Executable. Furthermore, some sections can be completely discarded and freed if they are marked as IMAGE_SCN_MEM_DISCARDABLE or IMAGE_SN_MEM_NOT_CACHED. After all of that is done, the loader must notify the DLL that it has been loaded and attached to a process. It normally does this using the flag DLL_PROCESS_ATTACH, which is always checked in the DLL using a switch statement. We can implement this part using the *AddressOfEntryPoint* and casting it to a function pointer whose signature matches that of the DllMain. The call could look something like this:

(*EntryPoint)((HINSTANCE)base, DLL_PROCESS_ATTACH, 0);

© SANS Institute 2024

https://linktr.ee/offsecexam



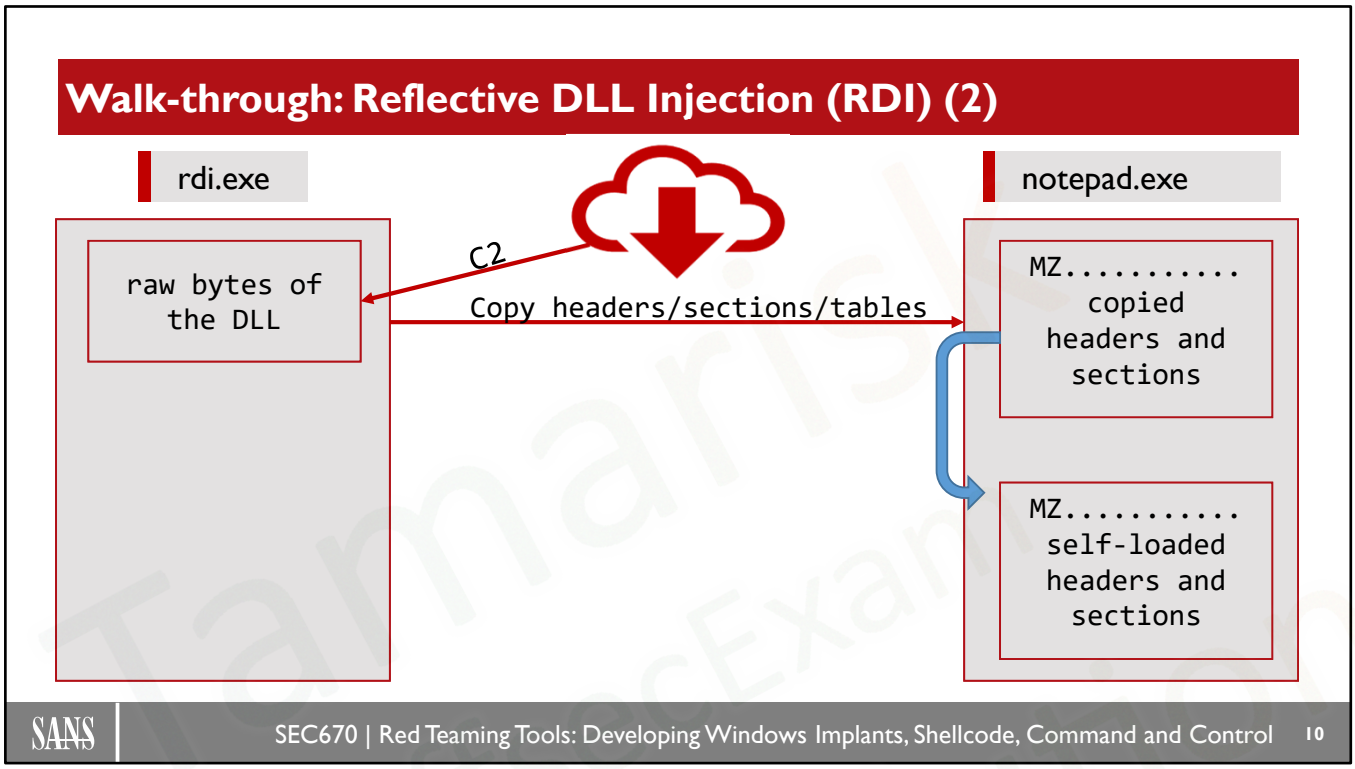
Walk-through: Reflective DLL Injection (RDI) (1)

Instead of trying to talk about it more, let us try to visualize the steps involved with RDI. One of the first things we might want to do is create a chunk of memory on the heap, and its size is determined by the DLL's file size. This will have the raw bytes of the DLL and from its base, we can start to parse the contents of the PE headers and sections. Before we can parse and move them, we need to allocate some more space, but this time we will grab the size from the ***OptionalHeader*** struct member *SizeOfImage*. With that value being known, we can proceed with copying over the headers into that new chunk of memory. Depending on the address of our given chunk of memory, we might need to process the relocations section just like the system loader would do. To do this, we need to determine the delta of the base address of our image and the address where we are in memory; the buffer, if you will. Once you are done processing the relocations section, any imported functions would need to be processed as well. Processing the IAT is similar logic to processing the EAT of a binary like we did earlier during Section 3—the export parser. Finally, after all fixups have been fixed up, sections have been copied over, and imported functions have been relocated to their new home, it is time to call DllMain.

This might still sound very confusing and intimidating without looking at some code but let us continue the discussion as we add another small step in the direction of stealth.

© SANS Institute 2024

https://linktr.ee/offsecexam



Walk-through: Reflective DLL Injection (RDI) (2)
Instead of having our DLL on disk and calling the ***CreateFile*** and ***ReadFile*** APIs, we can grab the DLL over our C2 channel and process everything just like we talked about on the previous slide. The functions of the Reflective DLL being pulled down onto the target should be position independent, meaning they can be executed regardless of where they are located in the process' virtual address space. Stephen Fewer exports a function called ***ReflectiveLoader*** that implements the heavy lifting that was mentioned on the previous slide. This function must be called by the rdi.exe and when it executes, it will reflectively load itself in Notepad's virtual address space. According to Fewer's GitHub repo, the ***ReflectiveLoader*** function will find its own location in memory, because it must be position independent. Remember that almost every process has Kernel32.dll and Ntdll.dll mapped in its address space? Well, ***ReflectiveLoader*** is going to take advantage of that and resolve several APIs of interest: ***LoadLibraryA***, ***GetProcAddress***, and ***VirtualAlloc***. ***VirtualAlloc*** is needed because it is going to allocate a contiguous chunk of memory so that it can load itself into it. ***GetProcAddress*** is needed because it needs to find the address of standard routines noted in the import table. ***LoadLibraryA*** is needed to load an imported module into memory. Now, in order for us to call DllMain, we must determine the entry point for the DLL. This can be achieved using the *AddressOfEntryPoint* member of the *OptionalHeader* struct.

And with all of that, it is finally time to take a look at some code.

© SANS Institute 2024

https://linktr.ee/offsecexam

RDI: Differences



Noting the differences between Classic and RDI methods of injection

1. Obtain process handle
2. Allocate memory for DLL path
3. Write the DLL path
4. Create remote thread to load the library into target process

1. Allocate local buffer and read in raw DLL bytes
2. Obtain process handle
3. Allocate remote memory
4. Copy over all sections keeping section permissions
5. Apply fixups for "rebasing"
6. Execute AddressOfEntryPoint

RDI: Differences

Now that we have discussed what RDI is and how to implement it programmatically, we should be able to spot the differences between the two types of DLL injection methods: Classic and Reflective. Let us cover some of those differences here. With Classic DLL Injection, the DLL must reside somewhere on disk and that is not something you want to do unless if you can help it. RDI can take the DLL as raw bytes from a socket as part of a staged payload (or stageless), like what Metasploit does. Classic injection writes the absolute path of the DLL in the target's virtual address space, whereas RDI places the PE headers and sections in the target's address space. Again, RDI can get away with this because it will never call ***LoadLibrary*** on that DLL. Classic injection depends on ***LoadLibraryA*** and lets the system loader take care of the loading process, which does have its advantages. RDI brings its own loader so it can load itself. The DLL used for RDI exports a position independent function, so it does not matter where it gets placed in the target process' virtual address space. So, not only can RDI take a DLL from memory, but it can also take one over a socket. Finally, gone are the days that a DLL needs to be on disk to be loaded.

© SANS Institute 2024

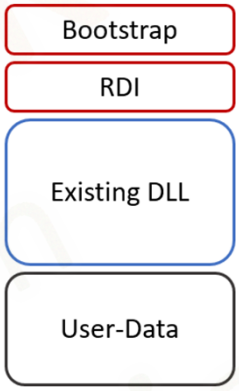
https://linktr.ee/offsecexam

Shellcode Reflective DLL Injection



Using shellcode to reflectively inject a DLL

sRDI is just like RDI but with a few twists. First, everything is converted to position independent shellcode. Second, your DLL does not need to be compiled with RDI. Third, the PE loader portion has been converted to shellcode.



Shellcode Reflective DLL Injection

What happens when you convert a reflective loader into assembly? You get Shellcode Reflective DLL Injection, or sRDI. It is the positioning of independent shellcode with which we can utilize and inject into a process using some of the injection techniques we have already covered up to this point. Basically, you take a malicious DLL that you made yourself, and run it against sRDI tools to make it into a position independent DLL. While getting started with this method and to ease debugging, it could be best to use a simple DLL like one that pops a message box or one that simply creates a new process. sRDI, according to its developer Nick Landers (monoxgas), can serve as a complete PE loader that provides support for various items like section permissions and TLS callbacks. sRDI can also help execute some shellcode locally by using a new custom function called ***GetProcAddressR*** that will look up exported functions just like the Win32 API version does. All the code for the project is hosted on the GitHub repo with more detailed explanations.

References:
https://www.netspi.com/blog/technical/adversary-simulation/srdi-shellcode-reflective-dll-injection/
https://github.com/monoxgas/sRDI

© SANS Institute 2024

https://linktr.ee/offsecexam

When To Make A Custom Loader



Sometimes, all the time, or never

There is a time and a place for everything including when to use your own loader. Perhaps one of the best times is when you do not want to have anything on disk. Reflective loading is commonly used to describe this method of manually loading something in memory.

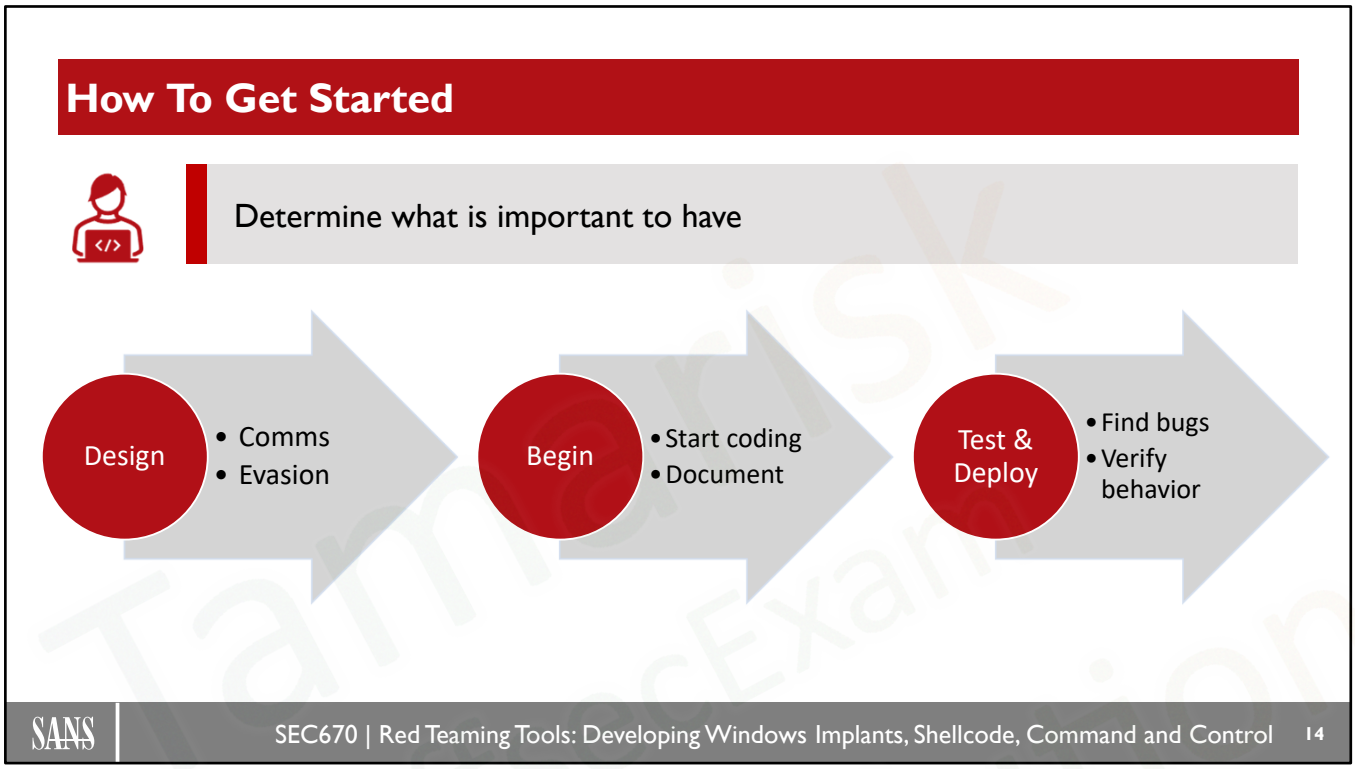
When To Make A Custom Loader

Having a custom loader is a great feature to have in your arsenal. Adding it as a feature to your custom shell would be amazing because it would allow you to pull down PE image over a socket and execute them without the need to drop to disk. Consider this scenario, say you are given the requirement from a red team to make a beaconing implant that calls out to their C2 setup every so often. The red team is planning on using your beaconing implant as a standalone executable, for whatever reason, maybe for persistence. This standalone does not have all the bells and whistles baked into it because the red team does not want everything to be discovered in one fell swoop. So, the extra features come in the format of a PE image, like a DLL. The red team could also pass another EXE to your implant, and have it executed in a sacrificial lamb of a process where the entire point of that new process is to house your new image.

When you do not want anything additional to be dropped to disk, you are not left with many other options to get that image loaded. Manually loading images also gives you the ability to unload them at will without triggering too many events.

© SANS Institute 2024

https://linktr.ee/offsecexam



How To Get Started

Making a custom loader can take a bit of time, especially when it is your first attempt at taking on such a project. You might have several questions about how exactly to even get started making one. One of the first things that is recommended with any project is to have a design phase. The design phase is where you spend a few days or up to a week to nail down the key components of the project, in this case, the loader. Some of the questions that might surface during the design review could be the following: should evasion be included; what about encryption; is the PE image going to be encrypted or packed; once loaded, should the loader code delete itself from memory; should it beacon out; should the comms be encrypted; is the image going to be loaded within our own process or another process; etc. Those questions just barely scratch the surface with what should be talked about during the design phase.

Once you are finished with the design phase, it is time to begin implementing the chosen design. You can determine what coding standards or practices you will adhere to, if any, but at a minimum your team should have their own standard in place. The language should have been chosen as well during the design phase and if it is C++, then start making some classes that can be used over and over. Next, you can move over and start getting into the weeds with everything. It might be good to have solid documentation along the way as you build it all out that way the developer coming along after you knows what you code is doing, without question.

Your testing comes into play continuously as you build it out. Final testing comes once you are ready to turn it over to the red team for release and deployment. Before that release, you should have squashed any bugs that may have crawled around in your code. To aid in testing, give your loader traditional images, malformed images, encrypted, packed, encoded images. Also, see what it does if you give it a Word document or Excel spreadsheet. Does it gracefully handle the unexpected file format, or does it crash and burn? All of these questions should have been answered before the red team starts using it.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation



Inside own process

Validation of file format and architecture

Process headers and all sections

Rebase, apply fixups, build import table, execute

```
// file format check
Offset 0x00 == “MZ”;
// architecture check
FileHeader.Machine == x64;
// process all headers/sections
PIMAGE_DOS_HEADER; PIMAGE_NT_HEADERS64;
PIMAGE_FILE_*; PIMAGE_OPTIONAL_*;
PIMAGE_SECTION_*; etc.
// process all data dirs
PIMAGE_DATA_DIRECTORY;
// build the tables; IAT, EAT, etc.
// process any base relocations
// kick off main
OptionalHeader.AddressOfEntryPoint;
```


Implementation

For the sake of time, we are skipping the design phase and will use a pre-chosen design that implements the bare minimum to get the job done. For this loader, we will be choosing C++ to get this done, after all this is a C++ course. You will be leveraging your newly found knowledge of the structure of PE images. You created a PE parsing utility already to obtain a function’s address inside of a loaded DLL. This time, you will be doing the loading all on your own. As you go through this, here are some things to make sure get done:

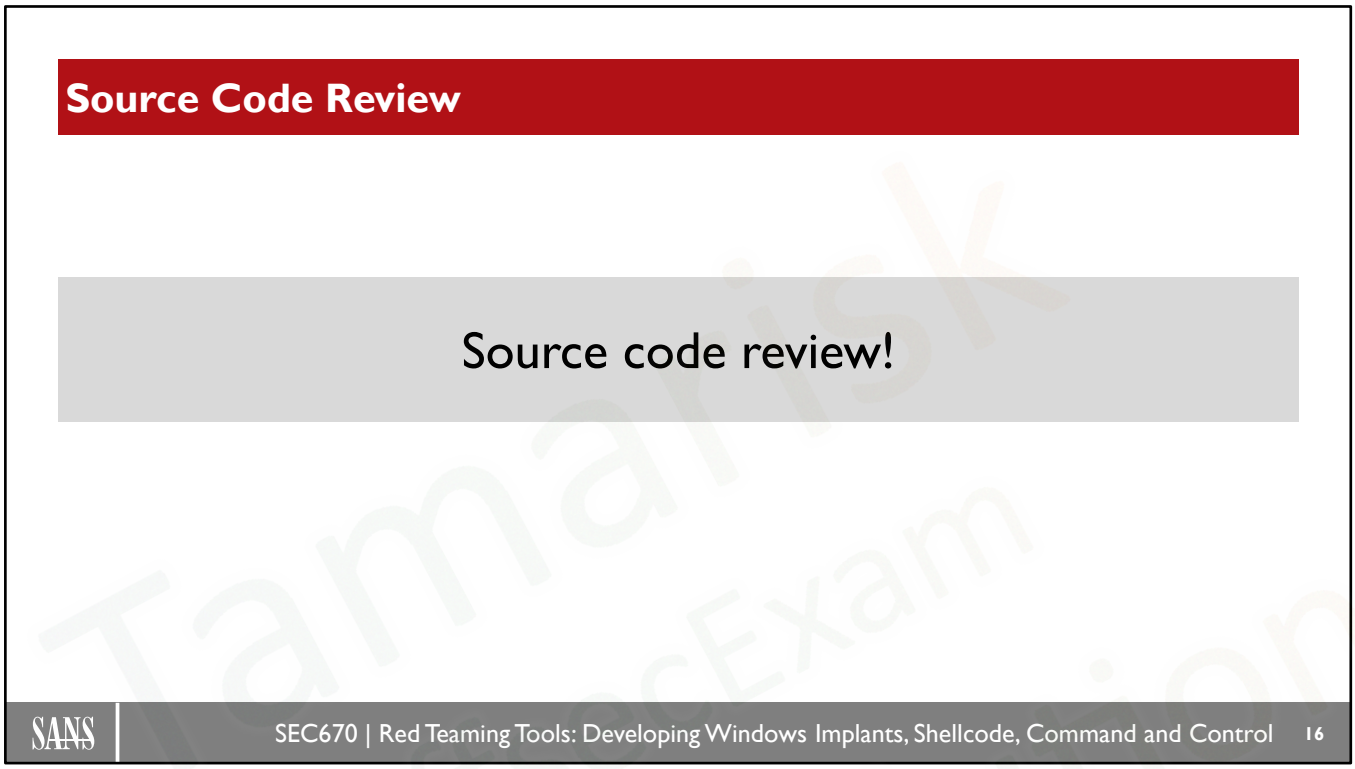
- Validate file format
- Validate architecture
- Process headers
- Process sections
- Apply fixups
- Build the tables, mainly imports
- Execute entry point

If you are making a loader that needs to handle Wow64 images, then the architecture validation would be a necessity. Wow64 processes are becoming less and less necessary these days and as such, we will just focus on x64 images and systems.

When it finally comes time to execute the image, you must know if the image is a DLL or EXE image. The reason for this is the signature of the entry point will be different. GUIs have WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR lpCmdLine, int nCmdShow) or wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow), Consoles have main(INT argc, PCHAR argv[], PCHAR envp[]) or wmain(INT argc, PWCHAR argv[], PWCHAR envp[]), DLLs have DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved), and native applications have NtProcessStartup(PPEB peb). Once you know the signature of the function, call it. Calling the function can be done with a thread or function pointer.

© SANS Institute 2024

https://linktr.ee/offsecexam



Source Code Review
Time to jump into the source code and explain it.

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 5.1: The Loader



Explore executing shellcode locally and over process boundaries.

Please refer to the eWorkbook for the details of the lab.

Lab 5.1: The Loader
Please refer to the eWorkbook for the details of this lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

Tamarisk OffsecExam Exam Solution

What's the Point?

The point of this lab was to explore the methods of executing shellcode locally within your own process and across process boundaries into a target process on the same host. There are many methods that can be used to execute shellcode via injection, these labs explored the most tried and true methods of getting the job done. These methods can be further enhanced by using non-standard APIs to help avoid detection.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Discussed what a loader is

Covered various loader methods and tools

Looked at how to create a loader from scratch

Module Summary

In this module, we went back in time to understand how the method of manually loading images first came on to the scene. Then we went over a few enhancements that also brought newer methods as the years went forward. Finally, we discussed the overall operation for making a custom loader and reinforcing the concepts in a lab.

Tamarisk OffSecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the preferred way to load a DLL coming over a socket?

A. Write it to disk then call LoadLibrary()

B. Keep it in memory and load it reflectively

C. None of the above

Unit Review Questions
Q: What is the preferred way to load a DLL coming over a socket?

A: Write it to disk then call LoadLibrary()

B: Keep it in memory and load it reflectively

C: None of the above

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the preferred way to load a DLL coming over a socket?

A Write it to disk then call LoadLibrary()

B Keep it in memory and load it reflectively

C None of the above

Unit Review Answers
Q: What is the preferred way to load a DLL coming over a socket?

A: Write it to disk then call LoadLibrary()

B: Keep it in memory and load it reflectively

C: None of the above

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the function signature for a native application?

A. NTSTATUS NtProcessStartup(PPEB peb)

B. DWORD NtMain(int argc, const char* argv[])

C. INT main(int argc, const char* argv[]);

Unit Review Questions
Q: What is the function signature for a native application?

A: NTSTATUS NtProcessStartup(PPEB peb)

B: DWORD NtMain(int argc, const char* argv[])

C: INT main(int argc, const char* argv[])

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the function signature for a native application?

A. NTSTATUS NtProcessStartup(PPEB peb)

B. DWORD NtMain(int argc, const char* argv[])

C. INT main(int argc, const char* argv[]);

Unit Review Answers
Q: What is the function signature for a native application?

A: NTSTATUS NtProcessStartup(PPEB peb)

B: DWORD NtMain(int argc, const char* argv[])

C: INT main(int argc, const char* argv[])

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What structure and field member refers to the program's main function?

A IMAGE_OPTIONAL_HEADER.ImageBase

B IMAGE_FILE_HEADER.PointerToSymbolTable

C IMAGE_OPTIONAL_HEADER.AddressOfEntryPoint

Unit Review Questions
Q: What structure and field member refers to the program's main function?

A: IMAGE_OPTIONAL_HEADER.ImageBase

B: IMAGE_FILE_HEADER.PointerToSymbolTable

C: IMAGE_OPTIONAL_HEADER.AddressOfEntryPoint

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What structure and field member refers to the program's main function?

A IMAGE_OPTIONAL_HEADER.ImageBase

B IMAGE_FILE_HEADER.PointerToSymbolTable

C IMAGE_OPTIONAL_HEADER.AddressOfEntryPoint

Unit Review Answers
Q: What structure and field member refers to the program's main function?

A: IMAGE_OPTIONAL_HEADER.ImageBase

B: IMAGE_FILE_HEADER.PointerToSymbolTable

C: IMAGE_OPTIONAL_HEADER.AddressOfEntryPoint

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 5

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- **Enhancing Your Implant: Shellcode, Evasion, and C2**
- Capture the Flag Challenge

Custom Loaders
Lab 5.1: The Loader
Unhooking Hooks
Lab 5.2: UnhookTheHook
Bypassing AV/EDR
Calling Home
Lab 5.3: No Caller ID
Writing Shellcode in C
Bootcamp
Lab 5.4: AMSI No More
Lab 5.5: ShadowCraft

In this module, we will go over hooks and how to unhook hooks to re-hook your own hooks. In other words, a lot of hooking.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Learn about syscalls
- Discuss why we would need to unhook hooks
- Learn how to find hooked functions
- Learn how to re-hook hooked functions

The objectives for this module are to learn about syscalls, discuss why we would even need to unhook functions. The presence of a hooked function could indicate that another malicious actor is on the box with us or that some AV product has been installed and implemented its own hooks for functions that it thinks it needs to "watch." The latter option is most likely going to be the situation you might come across during your engagements, but you never know. For us to unhook hooked functions we will first need to find the hooked function, obviously. How best can we do that and what sources do we trust?

© SANS Institute 2024

https://linktr.ee/offsecexam

What Is A Syscall



Syscall me maybe

Syscall is short for system call. A syscall is a mechanism used to transition code from user mode to kernel mode and back to user mode again. Each syscall has a number tied to it, an index into a table linking it to a specific kernel mode function to invoke when kernel mode code begins execution.

- NtOpenProcess
- NtCreateProcess
- NtCreateToken
- NtAccessCheck
- NtCreateThread
- NtLoadDriver

What Is A Syscall?

What happens when code transitions into the kernel? How does that process even begin? If you are familiar with interrupts or if you have previous exploit dev experience, you might already understand a little bit as to how user mode code interrupts the kernel to ask for some help. What is happening here is a call is being made into the kernel to execute a specific routine. Who are you going to call unless you know the number? The syscalls that are invoked are all tied to specific numbers. For example, in ntdll.dll, there is a table of functions that are known as syscalls, and some are listed on the slide. Each one of those has a number that acts as an index for who to call in the kernel. Typically, these syscall numbers are found to be sequential with the first syscall number being zero (0), the next one would be one (1), then two (2) and so on. This sequential ordering is not guaranteed whatsoever so care must be taken when relying on it to resolve a syscall number. In fact, even the numbers for the syscalls listed above have changed over the different versions of Windows, so you cannot even rely on memorizing or hardcoding their numbers.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's Your Number



Syscall my number

System Call Symbol	Windows XP (hide)	
	SP1	SP2
NtAcceptConnectPort	0x0060	0x0060
NtAccessCheck	0x0061	0x0061
NtAccessCheckAndAuditAlarm	0x0026	0x0026
NtAccessCheckByType	0x0062	0x0062
NtAccessCheckByTypeAndAuditAlarm	0x0056	0x0056
NtAccessCheckByTypeResultList	0x0063	0x0063
NtAccessCheckByTypeResultListAndAuditAlarm	0x0064	0x0064
NtAccessCheckByTypeResultListAndAuditAlarmByHandle	0x0065	0x0065

Windows 10 (hide)								
1507	1511	1607	1703	1709	1803	1809	1903	1909
0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029
0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063
0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059
0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064
0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065
0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066

What's Your Number?

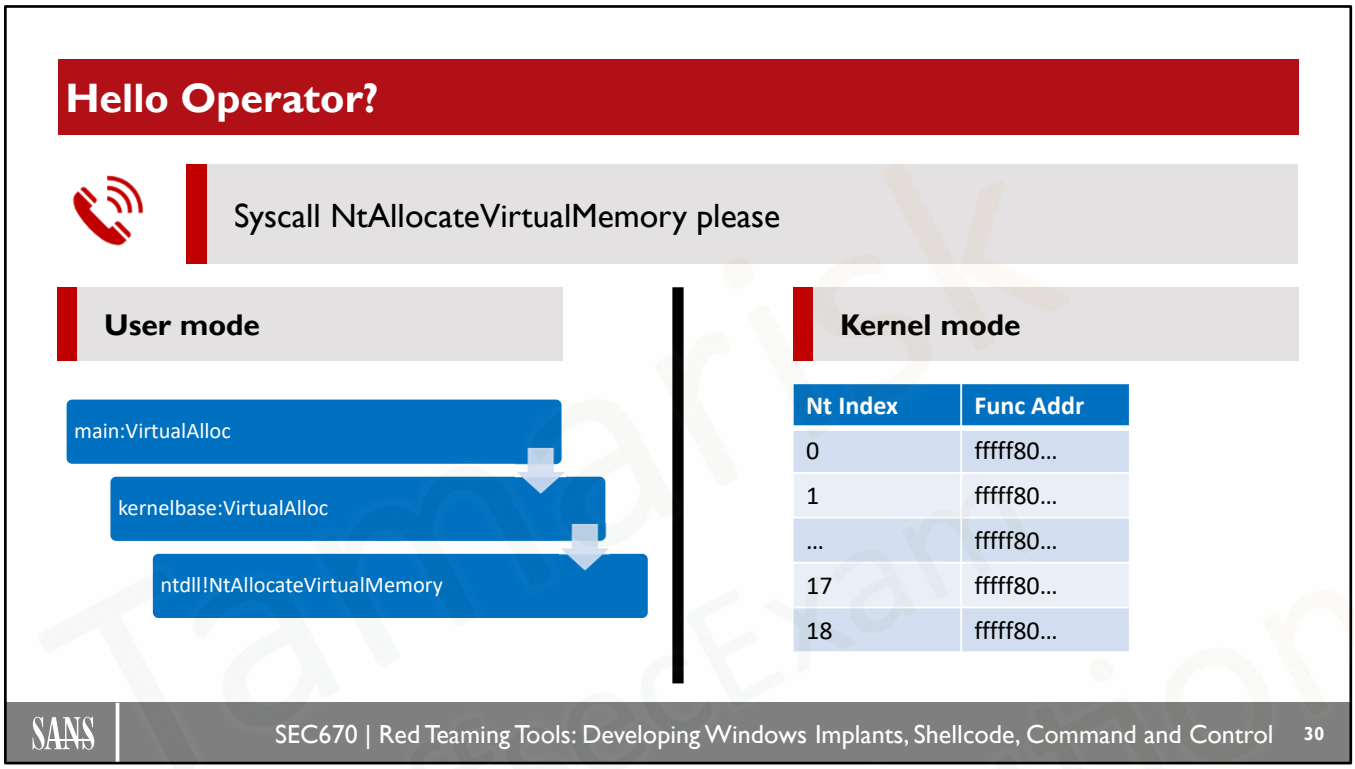
Many attempts have been done to document each one and arguably one of the more popular documents is the syscall table by Mateusz Jurcsyk (j00ru). The screenshot on the slide is from j00ru's HTML version of the syscall table. The table shows syscalls from Windows XP SP1 to Windows 10 and Windows 11. The online tool offers a feature to highlight a certain number of a syscall, like syscall 0x00 for example. Depending on the version of Windows you have, syscall 0x00 will either be ***NtAccessCheck*** on Windows 10 through Windows 11 or ***NtMapUserPhysicalPagesScatter*** on Windows XP SP1 through Windows 7 SP1. You can now start to understand how the syscall numbers vary quite a bit as newer versions of Windows comes out and as Microsoft adds more syscall functions to Windows itself. Many shellcoders who go through the pain of manually writing their shellcode, have often used a hardcoded value. Those coming from Linux do not have to worry about syscall numbers changing. Interrupt 0x80 will probably always be the same 32-bit syscall across all versions of Linux, maybe. Coming back to Windows, you cannot hard code your numbers without limiting yourself. The limit is narrowing your portability that your implant can run on multiple versions of Windows. If you are only targeting one and only one version of Windows 10, then maybe you would be able to get away with hard coding syscall numbers. Hard coding anything like syscall numbers or kernel offsets is ever recommended unless there is absolutely no other way around it.

Since we cannot 100% rely on hard coding numbers, what can be done about it?

Reference:
https://github.com/j00ru/windows-syscalls/tree/master

© SANS Institute 2024

https://linktr.ee/offsecexam



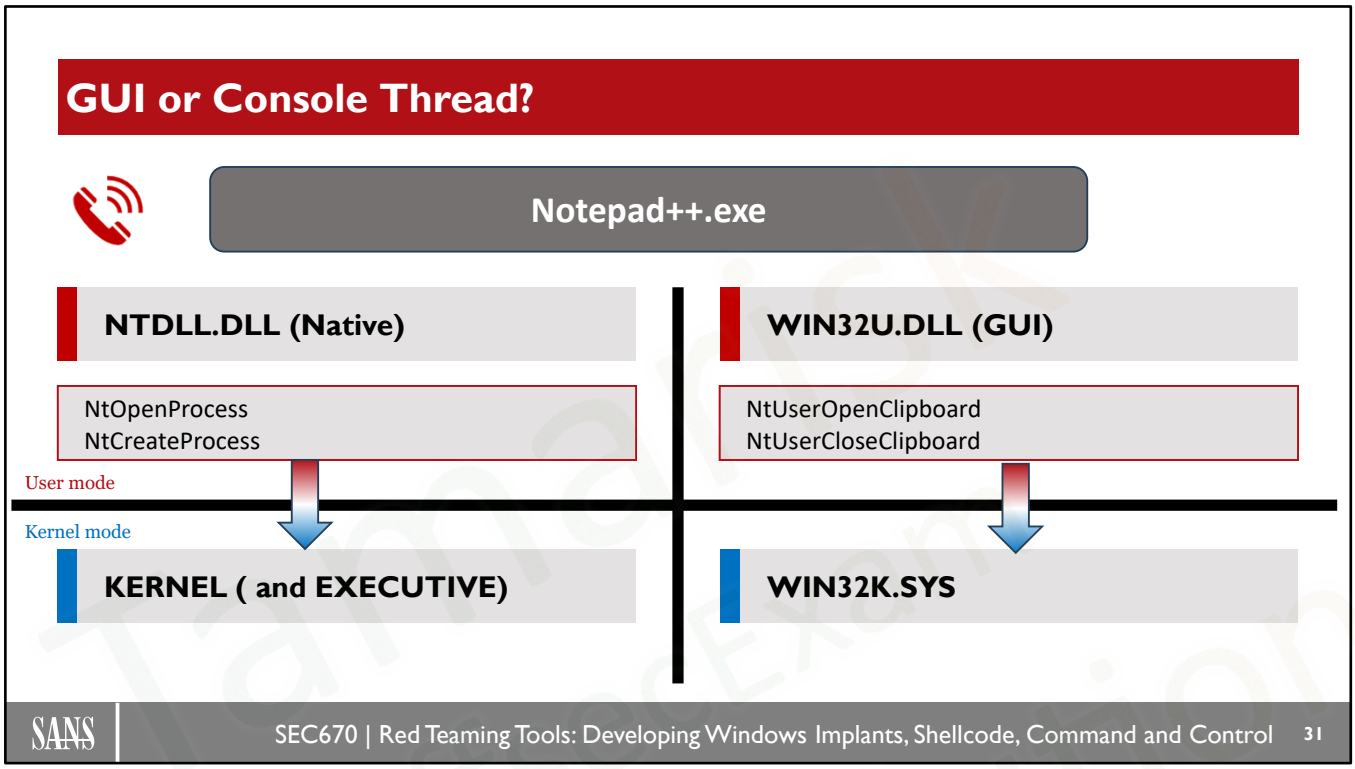
Hello Operator?

Without diving deep into the weeds with the full transition into the kernel, how does a user mode process get the help of the kernel and its syscalls? When a user mode application needs to create an objects, like a process object, or when it needs to allocate pages of memory, it will eventually need the help of the kernel. There is a process that takes place before this happens because typically, your user mode process does not go directly into a syscall. If you call ***VirtualAlloc*** from your main function, eventually it would be forwarded to kernelbase.dll and then ultimately ntdll.dll, which as you recall is the last stop right before jumping into the kernel. Ntdll.dll is not alone though, there is another last stop DLL before the kernel and that would be win32u.dll. Win32u.dll is for all GUI threads that are running GUI windows and applications. It also has a series of syscall numbers that are tied to kernel routines. Once the code flow jumps into ntdll or win32u, a stub is prepared that sets up specific registers with their proper values. Perhaps one of the more important values is the syscall number itself. After a few table lookups and sanity checks to make sure the syscall number is not too high of a number than what is supported in the table, the routine will execute and do what needs to be done.

You might be wondering, if we have two different user mode DLLs that are the last stop before the kernel, how does the kernel know what is coming in and from what DLL? Are not all threads the same? Does it even matter? Yes, why, yes, it does matter.

© SANS Institute 2024

https://linktr.ee/offsecexam

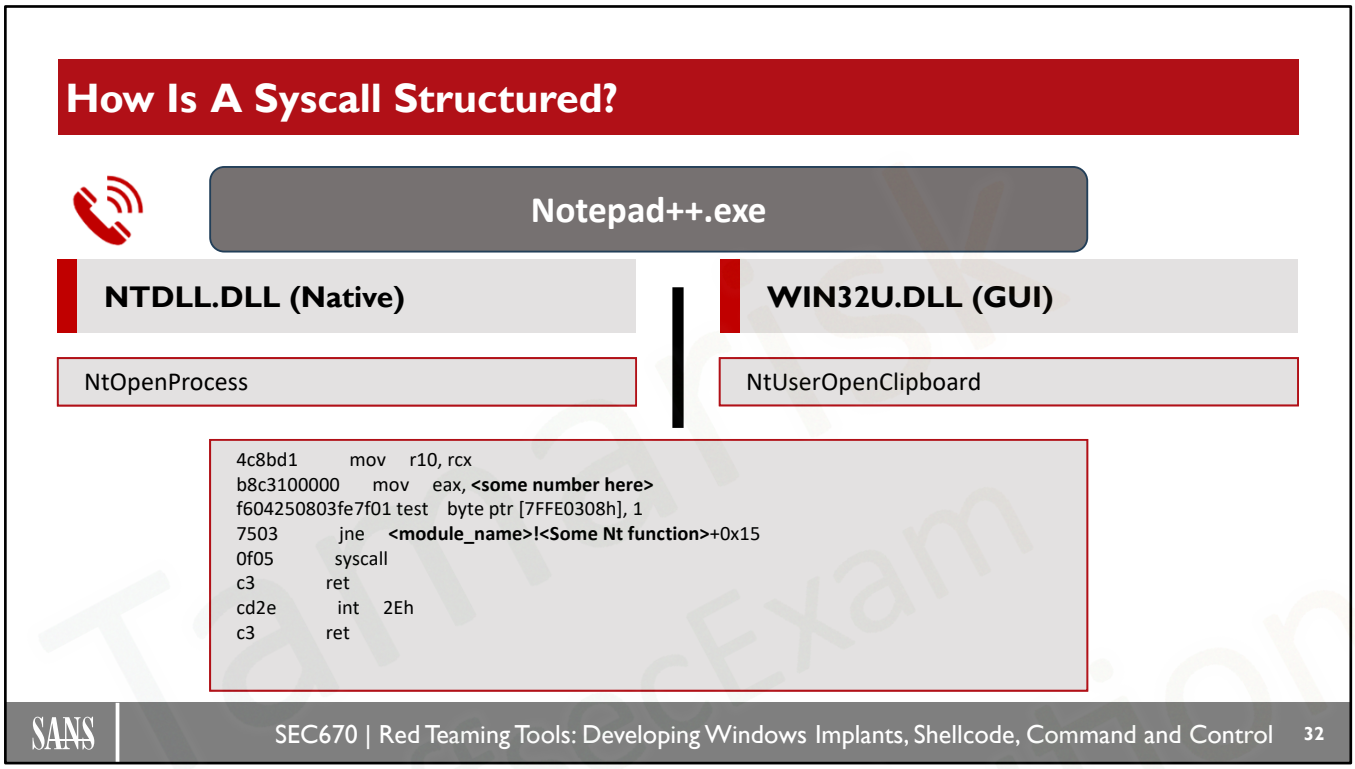


GUI or Console Thread?

It matters. All threads are not equal and when they invoke a syscall, where they end up and how they get there differs. They all will wind up in the kernel, but their journey to get there is a bit different. Let us look at a user mode process like notepad++.exe. As you are typing your awesome stuff into the window, you might at some point want to copy something over to another process, so you use the famous keyboard combination of CTRL+C to copy the selected text to the clipboard. Behind the scenes, there will be some syscalls that make this happen. One of the calls could be to ***NtUserOpenClipboard*** and if you look at the slide, you will see that it will wind up being invoked inside the kernel driver win32k.sys. This is a bit different from native syscalls, which are the more popular ones being discussed and used today in the field of implant development. Each kernel component implements its own table that determines where the flow is going to go. There are technically only two (2) tables that are indexed with syscalls, but there could be a max of four (4). It makes sense to split these up into different portions of the kernel as it keeps the modules smaller in size but also keeps the functionality separate, which would be easier to maintain. Each one of these syscalls, GUI or native, could be hooked by some EDR to have their desired level of introspection with how a function is being called and with what parameters. What does a syscall even look like?

© SANS Institute 2024

https://linktr.ee/offsecexam



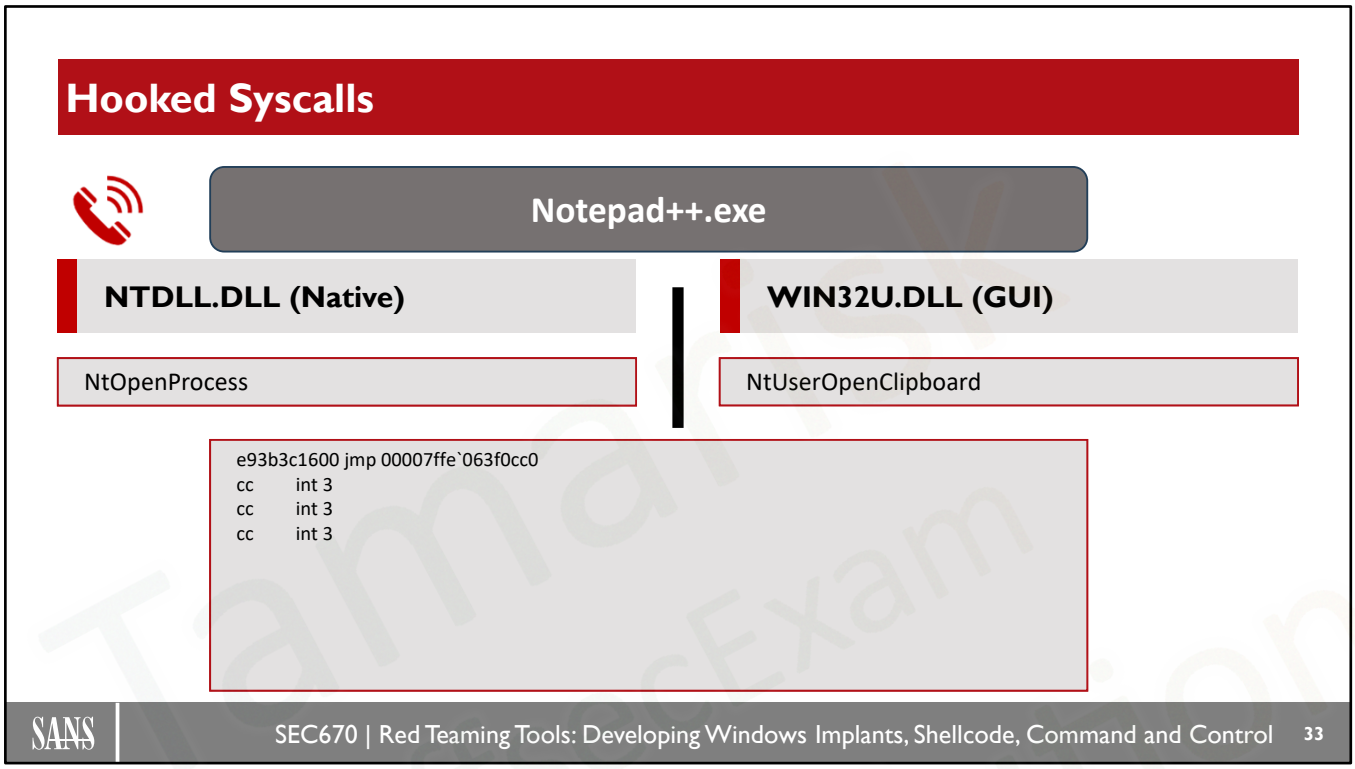
How Is A Syscall Structured?

Believe it or not, syscalls have a signature just like regular functions. The major differences with syscalls though is they do not set up the same things a called function does. For instance, a typical function will enter its prolog, create a stack frame of a particular size, save some registers that might be clobbered, it might optionally do something with the 64-bit shadow space for the four (4) registers holding some arguments, and on the way out it will do the opposite during its epilog. A syscall does none of that and is incredibly small, in fact, the small size could be why some refer to them as stubs. The example on the slide showcases the stub of a generic syscall. The instructions in the stub are broken out as follows:

- The contents of RCX are moved into R10
- The syscall number is moved into EAX
- A test is done against 0x7FFE0000 + 0x308 with the value 0x1: KUSER_SHARED_DATA->SystemCall
- A jump is taken depending on the test results
- The syscall instruction is invoked and the transition to kernel mode begins
- The stub returns
- The interrupt instruction is invoked with 0x2E if the jump is taken
- The stub returns

© SANS Institute 2024

https://linktr.ee/offsecexam

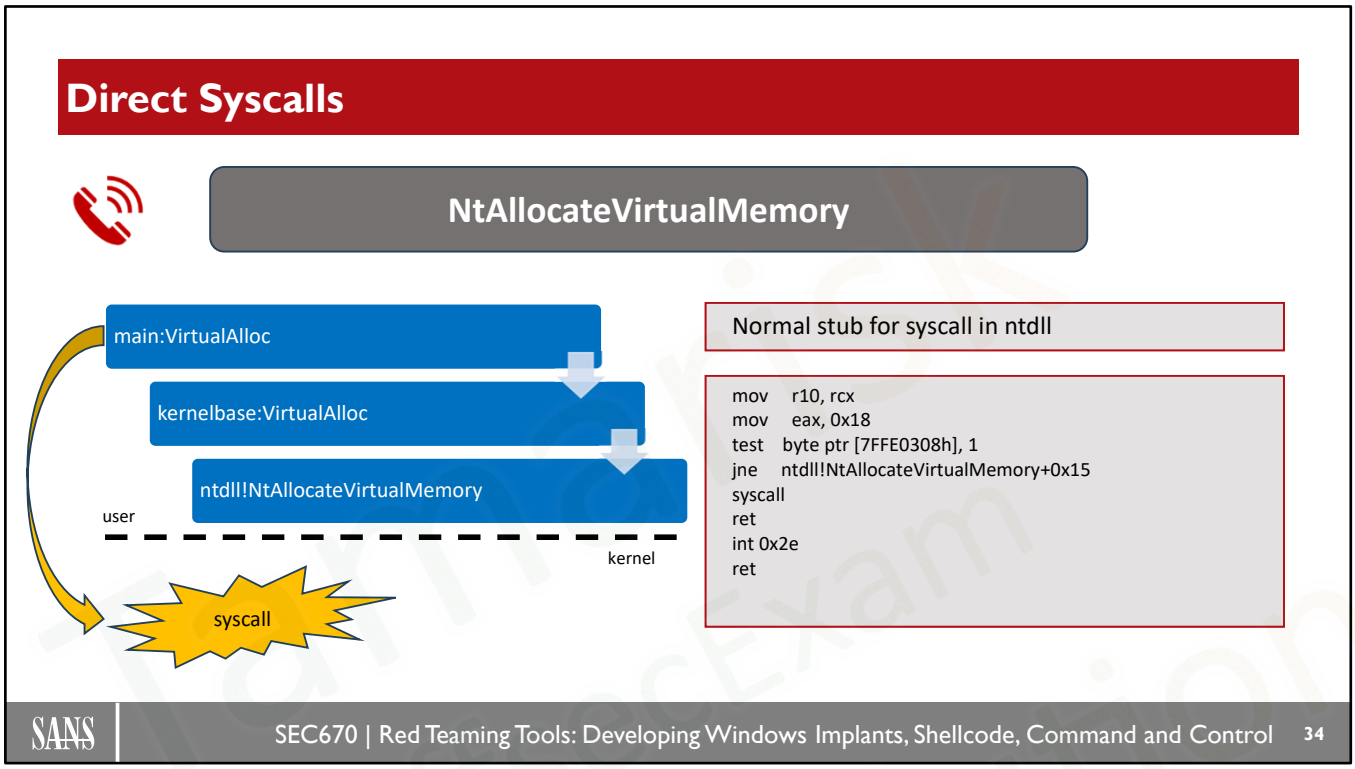


Hooked Syscalls

Security products will often implement any number of user mode hooks. In fact, there have been a few people that take the time to document all the hooked functions and in what modules they are hooked; ntdll, kernelbase, win32u, etc. One of those efforts is hosted by VX-Underground on their GitHub repo. The whitepaper is called AntiVirus artifacts with first, second, and third editions. The best way for security products to protect their customers from malware is to inspect the commonly used functions malware developers are using and see what is being provided for arguments. The product has some sort of logic implemented to look at the argument values and decide if the usage of the function is suspicious. There is no single API that is malicious, all functions are benign out of the gate. However, it can be the combination of several APIs that can make code appear to be suspicious. The APIs being used by malware developers are the same ones being used by Windows itself and by the security product. So, for security products to protect their customer, they must implement a hook so their code can run and inspect everything. The example on the slide is one example of a hook being implemented by a security product called Bitdefender. Native and GUI syscalls can all be hooked by a security product. The downside for us as implant developers is that we no longer know what syscall number was being used for that syscall. We must now implement some kind of restore operation to get things back to where they were before the hooks were implemented.

© SANS Institute 2024

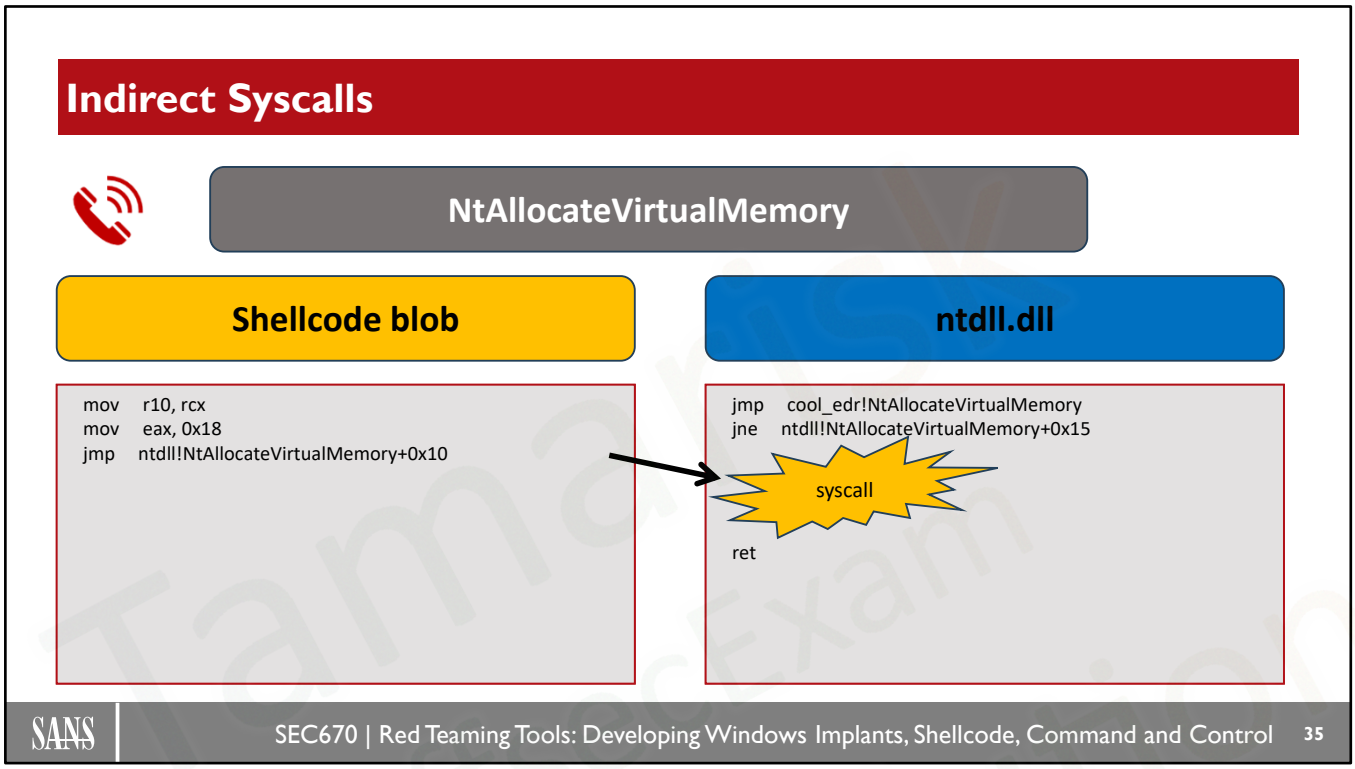
https://linktr.ee/offsecexam



Direct Syscalls

Hello operator, I would like to make a direct call to ***NtAllocateVirtualMemory***. Direct syscalls are as much interesting as they are old, but they are still being used to bypass some EDRs that are not up to speed with everything. This technique has been around for at least 10 years or so, but here is the gist of it. Instead of making your call to something like ***VirtualAlloc*** or ***VirtualAllocEx***, which eventually boils down to ntdll where the syscall gets invoked, we just bypass all of that. So, in our little implant, we make our own syscall stub and invoke it on our own. This also means that we must know the syscall number before we make the call. There are many techniques that exist today to help you recover those numbers either statically or dynamically. What is great about direct syscalls is that we will not be prone to any EDR user-mode syscall hooks. However, when EDRs have kernel-mode components, they can easily look at the call stack and will see that the invocation did not come from ntdll.dll's address space like it should have. Of course, there are other methods that implement a spoof of the call stack or even clone another thread's call stack to appear more legitimate. We can do a little bit better than this though, we must do better.

© SANS Institute 2024

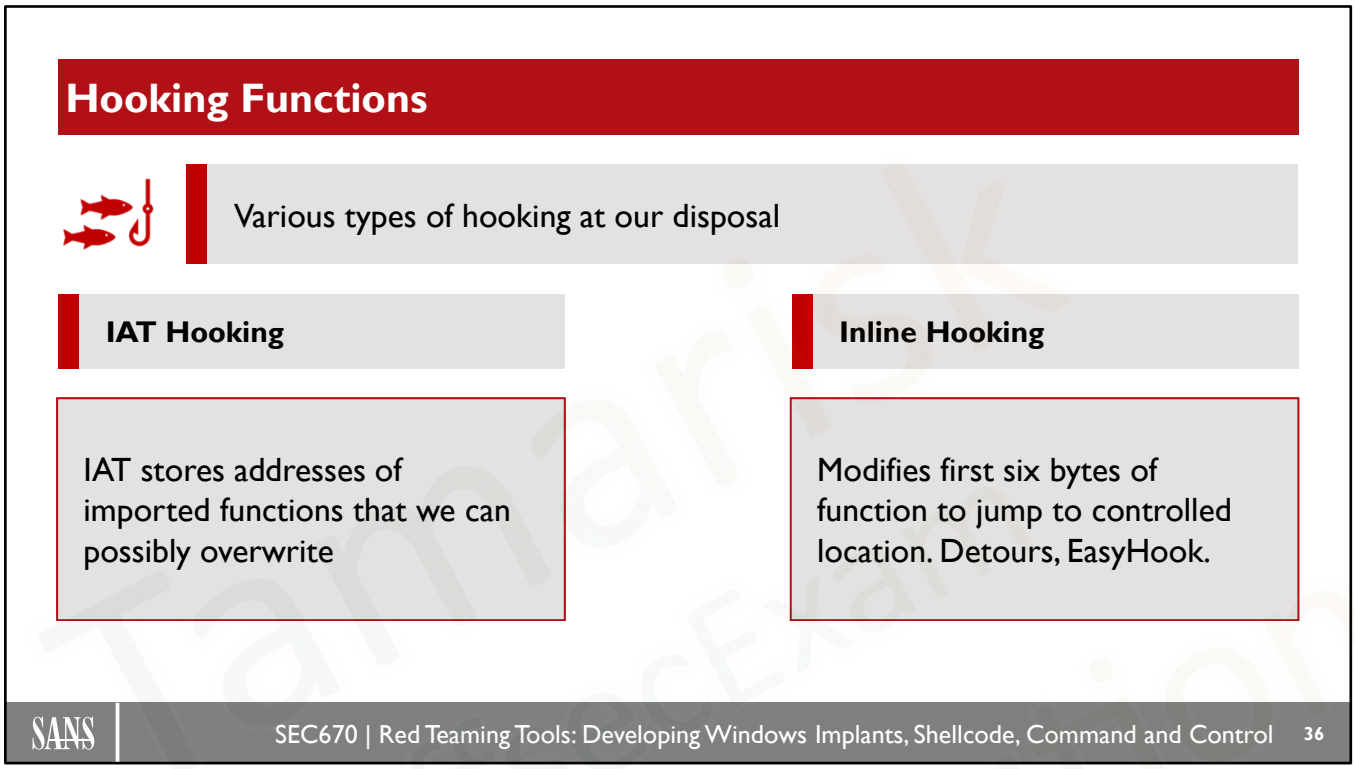


Indirect Syscalls

Hello operator, I would like to make an indirect call to ***NtAllocateVirtualMemory***. Looking at the slide here, you may have noticed that not a lot has changed with our approach. The biggest difference to note here is that we are not going to invoke the syscall instruction inside our malware.exe. Instead, we are going to jump over into ntdll.dll where the syscall instruction is located in memory. It should be good-to-go because it is not very common for security product's to modify too much of the syscall stub other than the first several bytes that they overwrite with their own jump. You should also note that we are still in a similar problem with knowing what syscall number is tied to the one we need. Just like what was mentioned on the previous slide, there are many ways we can go about finding those and those methods would still be useful for these indirect syscalls. One nice advantage we have with indirect calls is that we at least have a much better-looking call stack because we are technically coming from ntdll.dll's address space and not our own. This method will easily bypass a security product that is only inspecting the first or second return address. If one were to start diving deeper and deeper, then we would start to look a bit more suspicious.

© SANS Institute 2024

https://linktr.ee/offsecexam



Hooking Functions

Before jumping into restoring hooks, we need to discuss the two most used hooking methods out there: IAT and inline hooking. There could be several reasons why someone might want to hook, or intercept function calls. Perhaps a developer wants to verify their API is working as it should and implements an inline hook to double check the behavior of their API. You might also want to gain deeper knowledge with an application you are hired to reverse engineer and so you implement hooks to have the deeper insight. Also, it is fun to act as the debugger so hooking APIs can allow you to look at certain internal OS mechanisms like how a debugger might. You might also be a developer for some antivirus solution and your protection is done via hooking functions that malware typically use. In our scenario, we want to end up hiding a process.

Before we jump straight into the code and implement some hooks, let us talk about the process involved with hooking as well as some of the types of hooking we can leverage. The overall goal of hooking is to get your custom function to be executed instead of or before the real function. This way, your function can view the values of the parameters and possibly modify the values. Hooking is extremely popular in the game hacking community and their online forums are arguably the best place to learn how to hook functions and Windows internals in general. Game hackers have their own “cat and mouse” game with the anti-cheat drivers that are trying to protect their intellectual property like Counter-Strike: Go. Hooks can be done in both user mode and kernel mode with rising degrees of complexity. This course will strictly be staying in user mode.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unhooking Hooks: Why?



Restore intended use of a function

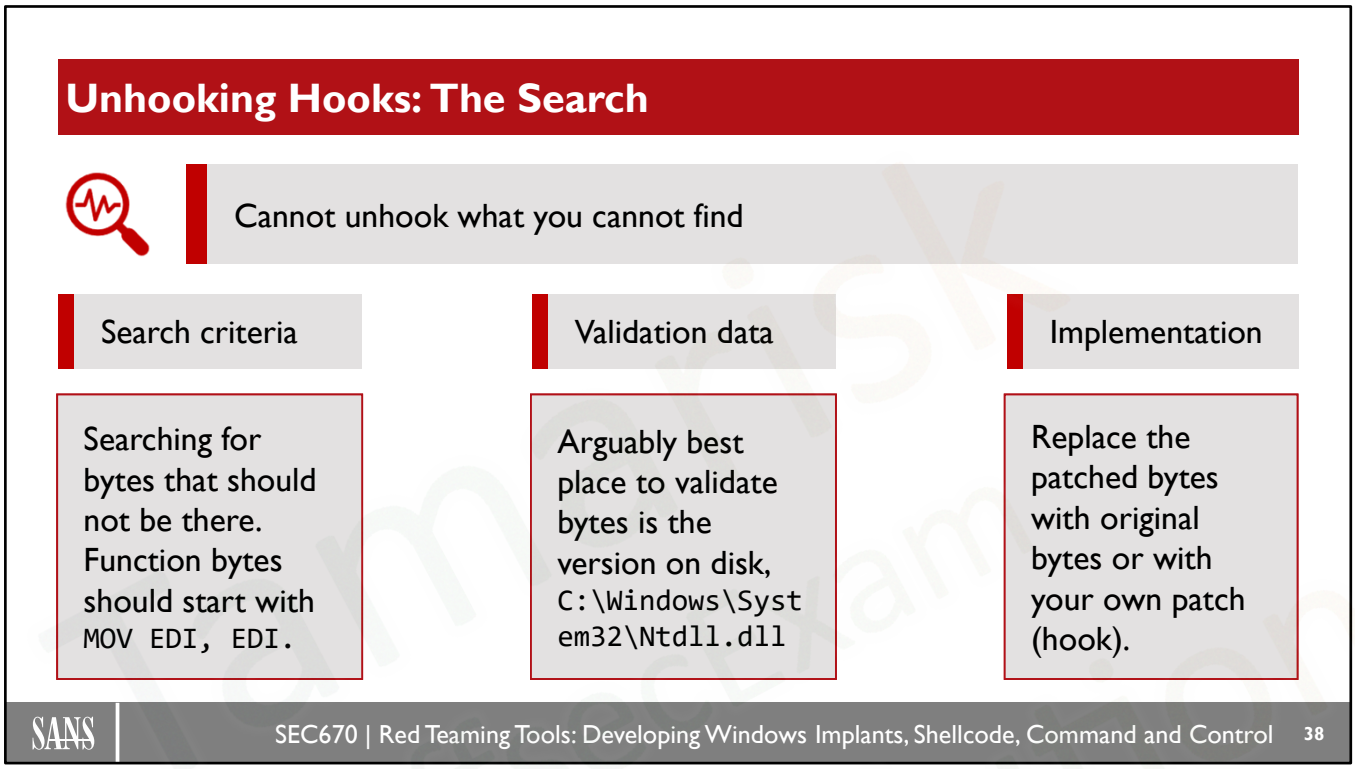
Hooks are, by design, implemented to change the behavior of a function. If we are trying to clean up and restore items of interest, it would be a good idea to also restore the original, intended use of a function like *NtMapViewOfSection*.

Unhooking Hooks: Why?

Perhaps a better question is who else out there might be hooking functions besides us? As mentioned previously, nation states and security products could be a solid bet. Security products might not be changing the behavior of a function but instead, looking at the arguments being passed to it. What a certain security products does with those parameter values is a bit of a mystery. There must be some logic in place that decides as to whether there is something suspicious happening. If the security product determines that there is something it does not like, such as a suspicious series of API calls with parameters that are typically used for injection, it will alert the user to it and possibly kill the process and/or quarantine any associated file. Obviously, this type of result is not desirable for developers or red team operators. Think back to Section 2 where we discussed how important it is to perform recon or information gathering about a target system. The same kind of thing could apply here, such as doing some recon to determine what APIs are being hooked. You could have a list of APIs that are important for your capability and if you see that those are being hooked, you will then take the action of unhooking it by any number of methods.

Nation states and other hackers are most likely modifying the behavior of the hooked function so they can have introspection and possibly modify the results of a function. If we want a function to execute as the developer originally designed it, then we must unhook that function and cut off the capability of whatever entity installed the hook. One downside to this is that some security products might be looking to see if their hooks are still being implemented, and if they see that we spit out their hook, they will most likely unhook us and re-hook the function. It can be a fun but annoying battle to fight. There would be nothing wrong with implementing a hooking integrity feature that would be on the lookout for if/when the hooks you put in place are changed.

© SANS Institute 2024



Unhooking Hooks: The Search

As discussed on the previous slide, you could come across functions that have already been hooked. You have a few options at that point with what you want to do, but first things first. Instead of accidently stumbling over a hooked function, how do you go about intentionally looking for a hooked function? Well, for starters, you can act like some AV solution and scan the first 5+ bytes of every function to see if what you except to be there is really there. If not and you see something like a **jmp** instruction instead, then odds are it has been hooked. Instead of searching through every function, you could be more focused in your search by only looking for functions that you are interested in hooking yourself. If you only want to hook ***NtQuerySystemInformation*** then just look for that single API.

For a moment, let's say you found a hooked function. Cool. Now what? Now, we validate what the original bytes were, and we do that by going to disk where the non-tampered version of the DLL should be located. We can read in the Ntdll.dll right from the System32 folder and see if the bytes in memory match the bytes on disk. If not, then we have just completed our validation of locating a hooked function. A word of caution here: there is no guarantee that the DLL on disk has not been modified either. It is possible to patch files on disk with the right permissions, and if that is the case then you would need to look at the same module on a mirrored system with all the same updates and patches. That would hopefully not be the case, but just know what to do if you ever come across that situation.

After validating the bytes from the file on disk, we can copy those and overwrite the hook to restore the original functionality of the function, if that is our desire. Perhaps though, our real goal is to re-hook the function with our own relative jump and keep that function hooked. We effectively just ejected an AV solution out of its own game, and it might not like it.

© SANS Institute 2024

https://linktr.ee/offsecexam

IAT Hooking



An array of addresses

AKA: Function pointer hooking

IAT is typically read-only
Must make it writeable

1. Parse PE headers to find import table
2. Locate module that implements the hooked function
3. Locate the function in the found module
4. Change page protections to PAGE_READWRITE, save old permissions
5. Overwrite function pointer
6. Restore previous page protections

IAT Hooking

If you recall from the PE header module, the IAT is one of 16 entries in the array named *DataDirectory*. The table itself is another array—an array inside of an array—that stores that addresses of imported functions. If your program uses the ***VirtualAlloc*** API, it will be listed as an imported function from Kernel32.dll and its address will be in the IAT. The IAT is typically read only but we can manipulate that and change the page protections to be writeable. One of the first items on our to-do list is to get a handle to our own base address and start parsing our own PE headers because we are trying to make our way down to the DataDirectory under the OptionalHeader. Once we are there, we can start looping over the modules looking for whichever one is responsible for implementing the targeted function. This could be Kernel32.dll or Ntdll.dll. Once the module has been found, we can then start looping over entries in that module looking for the targeted function, like ***NtQuerySystemInformation***. Once we find it, we can overwrite that function pointer. Again, one item we cannot forget about is modifying the page protections because if we attempt to overwrite the entry before we modify it, we will trigger an access violation. To do this we would use the ***VirtualProtect*** API, being sure to store the old permissions so we can clean up after ourselves once we overwrite the entry. The more you understand what structures and headers point to what, the quicker you can develop the code to get this done.

© SANS Institute 2024

https://linktr.ee/offsecexam

Inline Hooking



- Modifies bytes of function
- Inserts `jmp` instruction
- Hook beginning of function
  Hook mid function
  Hook end of function

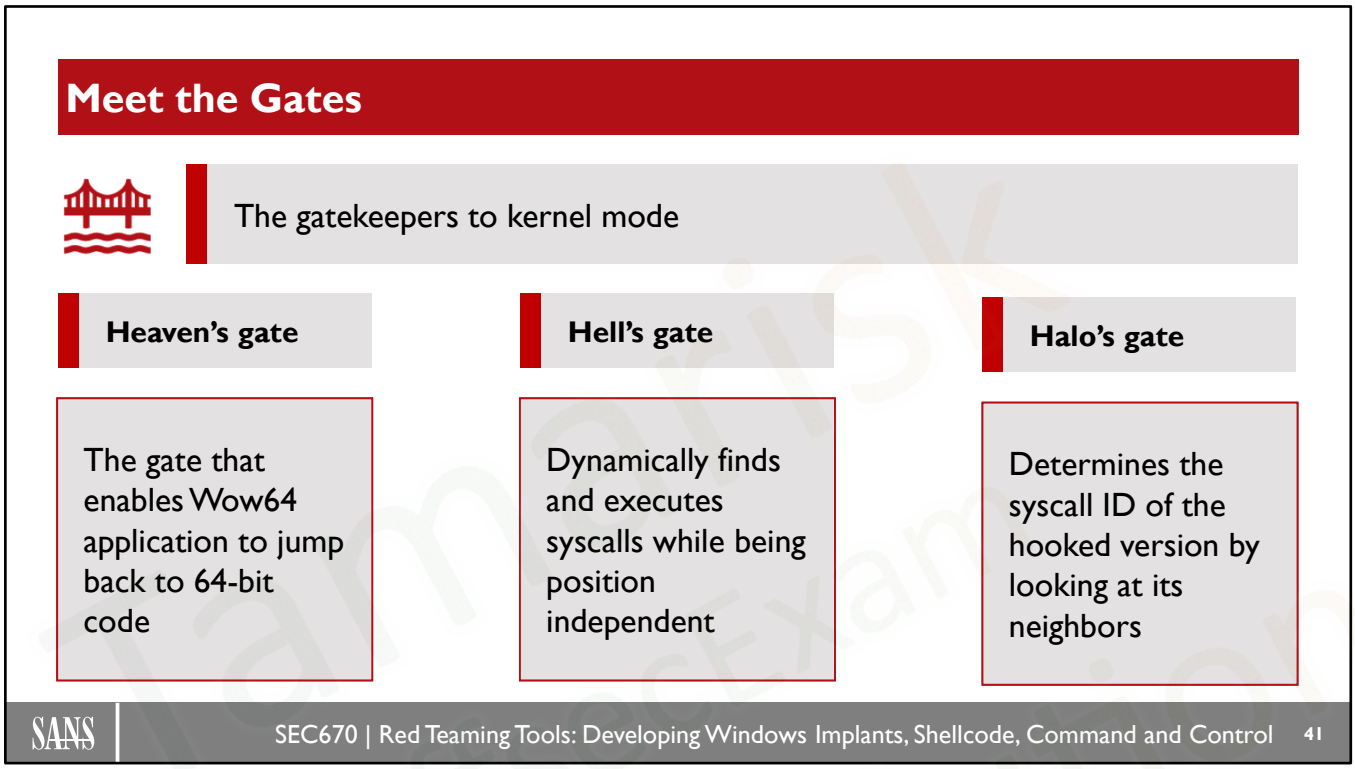
1. Obtain memory address of function
2. Read and save 5+ bytes of the function
3. Patch in the jmp
4. Your function executes
5. Clean up patched bytes
6. Execute original function

Inline Hooking

Perhaps the easiest way to learn how to implement an inline hook is to do so locally in the target process. External hooking is completely possible, but it is much more difficult and requires a deeper understanding of shellcoding. For this course, we will stick to local inline hooking to establish a solid foundation. One of the first items on the to-do list is to determine the address of the function in memory. This can be obtained using the ***GetProcAddress*** API since it returns the address of our desired procedure. Once we have the address, we can read 5+ bytes from the process' memory and save those bytes for later use. With the bytes safely saved off, we can overwrite them with our jmp instruction. The jump will be relative and as such it is impossible to calculate this at the time of development. We must do the offset calculation during runtime so we can calculate the relative offset to our hook function. Our hook function will be executed and inside it we can do pretty much whatever we want. Once the damage is done, we can then proceed to restore the original bytes by repatching the function and then executing it.

© SANS Institute 2024

https://linktr.ee/offsecexam



Meet the Gates

The general idea with these gates is to evade your intentions by not making direct Nt* API calls, but instead, invoking the system call yourself. This can be done by creating your own syscall stub, identifying the proper system call number dynamically, and others. We already touched on Heaven's gate previously, but it is mentioned again here as a collective on a single slide. As a refresher, this is how Wow64 applications get back to 64-bit code and invoke a syscall. Hell's gate takes the direct utilization of syscalls to a more dynamic level and achieves pure position independence. Halo's gate brings a new twist to using direct syscalls by locating neighbor syscalls to the one that is hooked. It looks left and right of the hooked call and by using simple math, identifies the syscall number for the one you need. We will take a deeper look at each one as well as dissect some source code that implements each method.

© SANS Institute 2024

https://linktr.ee/offsecexam

Hell's Gate



Dynamically locate and invoke syscalls

Completely position independent

Relies on Ntdll.dll

```
PPEB pPeb = (PPEB)__readgsqword(0x60);
//// hardcoded 2nd entry
(pPeb->  \
 LoaderData->     \
  InMemoryOrderModuleList.Flink->Flink \
   - 0x10)
//// check opcodes
if ((*(PBYTE)TargetFunction + 3) == 0xB8)
{
 // more opcode checks
}
```


Hell's Gate

Hell's gate is impressive because it does not rely on static syscall IDs. Be very cautious about hard coding values in your tools. The syscall IDs can change with any update and are not consistent between versions. There has been amazing research done by j00ru to document the IDs of practically every syscall from Windows XP to Windows 10. It is an amazing resources to have the information documented! Now, when it comes to creating your own stubs to invoke the syscall yourself, there are a few items we need to complete before we can do that. The first one is to dynamically locate and determine the syscalls for the system we are operating on. Hell's gate can also dynamically invoke the syscalls after self-resolving them. Another great advantage here is that it is completely position independent! RtlMateusz and am0nsec did an excellent job documenting and sharing their research in their paper located in their GitHub repo along with their code base.

References:
https://j00ru.vexillium.org/syscalls/nt/64/
https://github.com/am0nsec/HellsGate

© SANS Institute 2024

https://linktr.ee/offsecexam



Halo's Gate

Halo's gate offers a refreshing twist on using direct syscalls. The downside to Hell's gate is that it does not account for the possibility that the function in Ntdll.dll is already hooked. Should the function be hooked, then Hell's gate will fail. Where Halo's gate helps with this is taking advantage of the fact that the syscall IDs in Ntdll.dll are in numerical order. So, you will not find syscall 4F first. 4F will come immediately after 4E, and so on. Knowing this, if a function is already hooked, we can still find the syscall ID for it by looking at its neighbors. The hooked function would be the middle of your search and you would look forward, ascending in numbers, and backward, descending in numbers. This is effectively looking at your house in the middle of your street and looking at the address to the neighbor on your left and on your right. By knowing the addresses of the houses on either side of you, your house address could then be determined. We can do this programmatically very easily. Once the number is determined, we can invoke the syscall ourselves without having to repair the hooked function. This is great just in case the AV/EDR solution were to ever check the integrity of its hooks. It would check and see that its hook was still there!

Check out Reenz0h's blog where he did an awesome job documenting everything. We thank him for sharing his knowledge with everyone.

Reference:
https://blog.sektor7.net/#!res/2021/halosgate.md

© SANS Institute 2024

https://linktr.ee/offsecexam

Heaven's Gate



Hooking Wow64 and the gate to 64-bit code

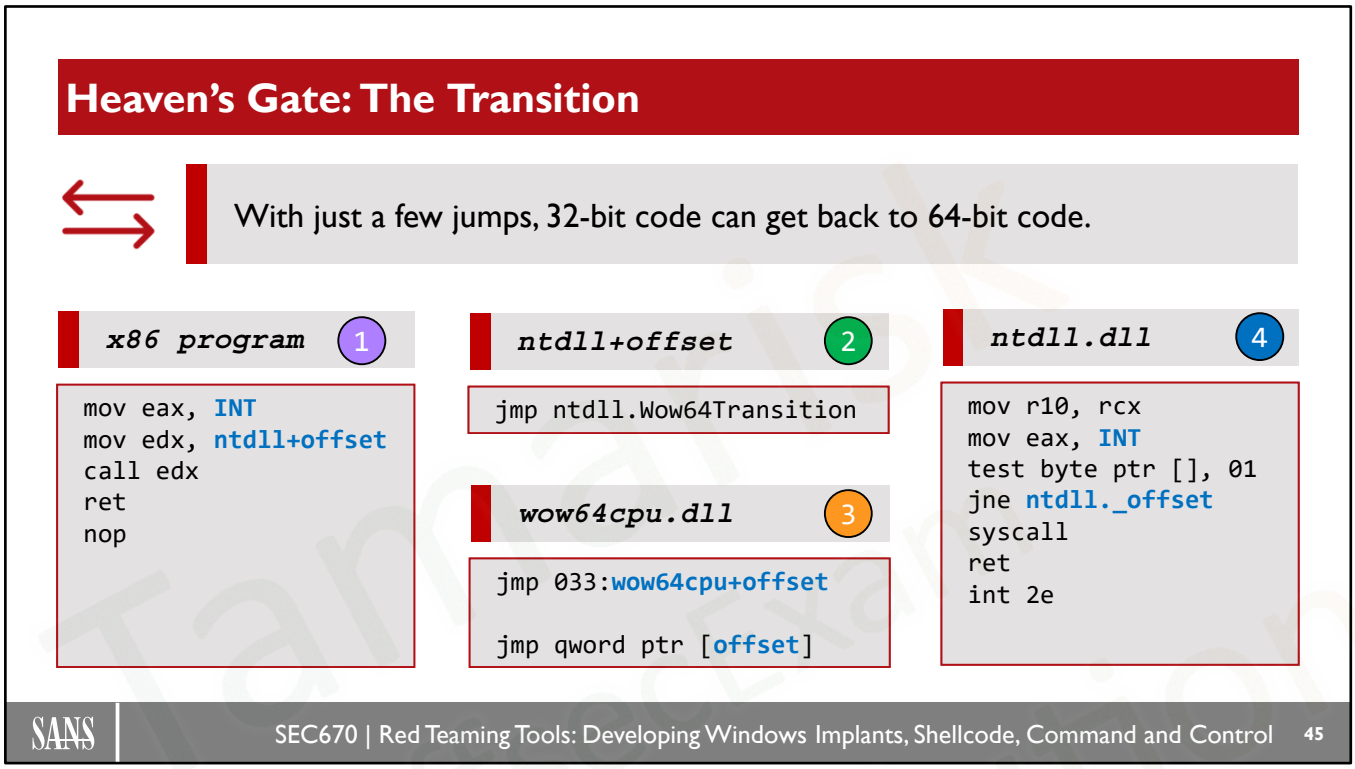
32-bit processes on 64-bit systems have an interesting method when it comes to making syscalls. Thanks to Windows 32 on Windows 64, this is all made possible. The transition from 32-bit code to 64-bit code has been dubbed Heaven's Gate.

Heaven's Gate

There might come a time when you need to implement some hooks in a 32-bit application, or more technically speaking, a Wow64 application. Ntdll.dll implements the logic for the system loader and thus is responsible for initializing the user mode portion of the new process. Because ntdll.dll is the system loader, ntdll.dll is loaded in every process. When it comes to Wow64 applications, 64-bit code always kicks things off starting with the system loader. Once it is done doing its 64-bit business, the 32-bit version of ntdll.dll is loaded. Depending on your debugger application, you would be able to see two entries listed for ntdll.dll. One of them would be mapped in 64-bit address space, while the other one would be in the 32-bit space. So, knowing this, would it make sense to hook Wow64 functions like ***NtAdjustTokenPrivileges***? Not really, because at some point there will be a transition that moves away from 32-bit code and executes 64-bit code. So, what if we implement our hook at a lower level, say the 64-bit version of ntdll.dll? Before we jump into some source code review, we need to fully understand how the 32-bit APIs in the 32-bit version of ntdll.dll transition to the 64-bit version. There are some other DLLs that aid in this process too as we will see shortly.

© SANS Institute 2024

https://linktr.ee/offsecexam



Heaven's Gate: The Transition

Because Windows takes pride in their ability to retain backwards compatibility, 32-bit applications can still be launched on 64-bit versions of Windows. What makes this interesting is the fact that two versions of ntdll.dll will be in the process. The 32-bit functions and syscalls must execute somehow, and all of it is made possible by something dubbed Heaven's Gate and a few system DLLs like wow64cpu.dll. When a 32-bit program needs to invoke a syscall, like ***NtAdjustTokenPrivileges***, ***NtCreateProcess***, or *Nt** something, a transition back to 64-bit code must take place. Afterall, this is a 64-bit system, so it makes sense.

1. The application will populate the desired syscall into the EAX register and make a call to some function at some offset into ntdll.dll. Notice though, that there is no syscall instruction here. It cannot happen just yet because we are still in 32-bit mode.
2. This is where code flow hops into the 32-bit version of ntdll.dll and from there we execute a far jump into the code segment of the wow64cpu.dll.
3. Here, inside wow64cpu.dll is another jump that will land us in the 64-bit version of ntdll.dll.
4. This is where we see the syscall for whichever one has been loaded into the EAX/RAX register from the user application.

© SANS Institute 2024

Unhooking Hooks: 32-Bit Example



What a prolog hook might look like for 32-bit (Wow64)

Non-hooked function

```
nop
nop
nop
nop
nop
mov edi, edi
push ebp
mov ebp, esp
```

Hooked function

```
jmp rel32     E9 xx xx xx xx
jmp 0xFB (-5) EB F9
```


Unhooking Hooks: 32-Bit Example

32-bit functions are very interesting because they typically begin with several NOP instructions, or no operations. If you are not familiar with the NOP instruction, then do not worry because it literally does nothing but waste one CPU cycle. It has no effect on registers or the stack—it simply executes and advances the Instruction Pointer to the next instruction to execute. The NOP instruction takes up a single byte and in comparison, a MOV instruction takes up 2 bytes. In total, there are 7 bytes before you would get to the traditional prolog starting with the PUSH EBP instruction. You might also sometimes see the CC instruction instead of the NOPs. The CC instruction takes up a single byte as well and simply translates to an interrupt. The NOPs, or CCs, never get executed anyway, so it does not matter what is there during runtime. However, what does get executed is the MOV EDI, EDI instruction. Because the same register is being used with the MOV instruction, it really behaves just like a NOP instruction would but because it takes up 2 bytes of space, we can overwrite those with a short JMP instruction, which will also take up 2 bytes of space. The idea here is to jump backward by 5 bytes to the beginning of the NOPs, or CCs, whichever is there. At that point, we can then implement a bigger jump and utilize all 32-bits, which would give us a range of +/-2GB. Pretty convenient for hooking, isn't it? It's as if Microsoft had us in mind when creating this.

But what is the MOV EDI, EDI anyway? There are other general-purpose registers that could be used instead, right? Perhaps, but according to Raymond Chen, whose blog you should read every morning before work, Microsoft talked with CPU manufacturers to determine what the best 2-byte NOP could be and thus, MOV EDI, EDI was born. Microsoft also uses these bytes as a mechanism to implement hot patches for a function. Hot patches are implemented when an API must be patched but the system cannot be rebooted just yet.

Reference:
https://devblogs.microsoft.com/oldnewthing/20110921-00/?p=9583

© SANS Institute 2024

https://linktr.ee/offsecexam

Unhooking Hooks: 64-Bit Example



What a prolog hook might look like for 64-bit

Non-hooked function

```
mov r10, rcx
mov eax, 41
....
```

Hooked function

```
mov rax, 1122334455667788
jmp rax
nop
nop
nop
```


Unhooking Hooks: 64-Bit Example

Just like the previous slide that covered 32-bit inline hook, we have a visualization for a 64-bit inline hook. Even though there is no typical prolog like what you might normally see for x86, the JMP instruction is used in a similar fashion. The JMP instruction is quite interesting for both 32-bit and 64-bit programs. For 32-bit you have far jumps, which is what was used on the previous slide along with a short jump. For 64-bit, it becomes interesting because you cannot jump to an absolute address. It must be an offset like a relative 32-bit offset, but that would obviously limit our jumping capabilities to 2GB in either direction. The JMP instruction is 5 bytes total, 1 byte for the actual opcode and the remaining 4 bytes are for the relative offset. Because this is 64-bit, 2GB is nothing in the scheme of the entire address space. What must be done instead is to take advantage of the MOV instruction. An absolute address can be moved into the RAX register and then we can JMP to the RAX register after the MOV is executed. In theory, you could use any register, but we are clobbering the RAX register, which is typically just fine.

64-bit inline hooks are more challenging because you need to take into account the variable instruction length, unlike for 32-bit where we know exactly how many bytes we are dealing with starting with the 5 NOPs before the function and the 2-byte JMP. You might have to keep track of at least 12 bytes because that is the minimum number of bytes required for the MOV; JMP RAX; technique. The NOPs on the slide are simply there for padding to show what it might look like if you have to account for 15 bytes in your hook.

© SANS Institute 2024

https://linktr.ee/offsecexam

Trampolines



What happens after your hooked function executes?

Hooked *NtQuerySystemInformation*

Could get stuck in a loop

```
mov rax, 1122334455667788
jmp rax

// hooked code here

// call original function
call NtQuerySystemInformation

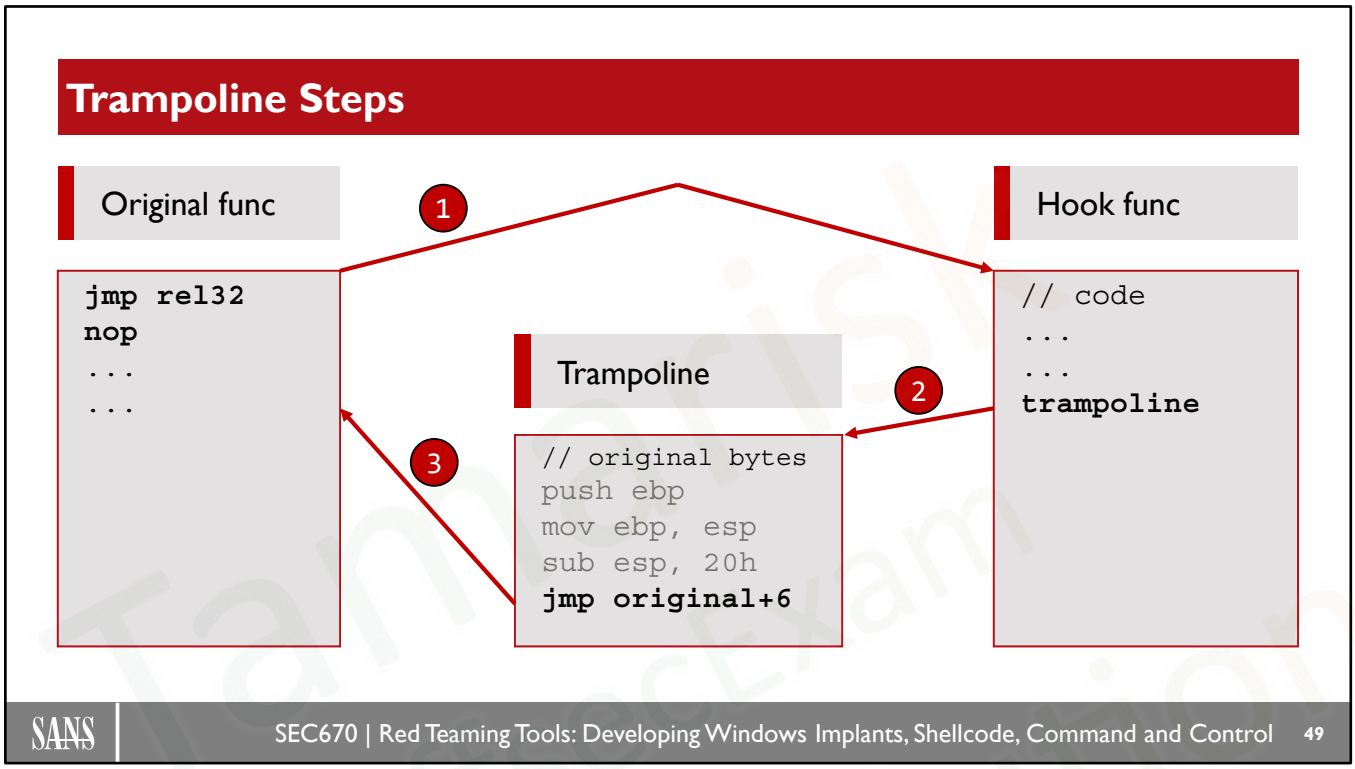
// enters loop
```


Trampolines

So, you have your inline hook implemented in ***NtQuerySystemInformation*** and your hook function gets executed. Awesome! Now what? Well, now you would probably want to call the original function and see what it returns or maybe pass in the modified arguments. Regardless of what you want to do, the original function should be called at some point. If you were to make the call to the original function, your hook is just going to get hit again and bounce you right back to your hook function. This is not what you want to happen because you will be caught in an endless hooking loop. What you need to do is somehow get back to the proper location in the original function. The proper location is immediately *after* the bytes you overwrote. Maybe that is something like ***NtQuerySystemInformation****+12* bytes for 64-bit hooks and +6 bytes for 32-bit hooks.

© SANS Institute 2024

https://linktr.ee/offsecexam



Trampoline Steps

This visualization shows what the entire process looks like from start to finish. The original function shown on the left-hand side of the slide has already been hooked, as can be seen by the relative jump followed by a NOP. The jump takes us to where the actual hook function is located, and it is here that our malicious needs can be carried out. Inside the hook function, we can eventually call the original function via a specially crafted trampoline. The trampoline holds the original bytes that were stolen from the original function and then a jump plus an offset from the start of the original function. The offset is extremely important to remember because we do not want to get into what some call hook recursion where we keep hooking ourselves.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unhooking Hooks: A Fresh Copy



There is another way to unhook hooks.

The system libraries that are stored in C:\Windows\System32 can serve as a validation for your integrity checks, but instead of copying the first several bytes from disk to memory, just copy in the entire TEXT section.

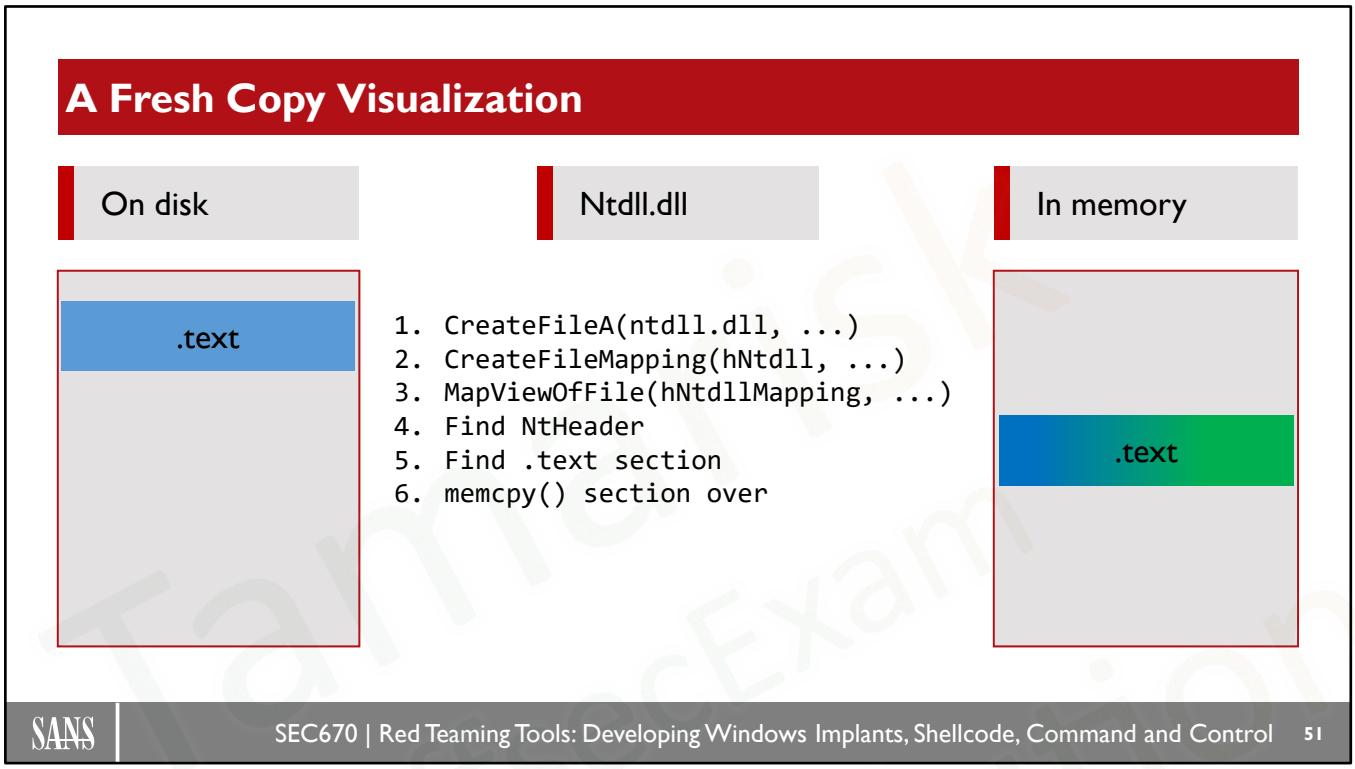
Unhooking Hooks: A Fresh Copy

When it comes to unhooking a function, there are several methods we can implement. One method is to use the original version of the function as found in the DLL on disk. This method has been dubbed "a fresh copy" where a clean copy of a system library found in the C:\Windows\System32\ folder can be used as a trusted source. If we take Ntdll.dll as an example, since it is a popular target by EDRs and AVs, we can read in the library straight from disk and copy over the tampered version in memory. This would effectively restore the original functionality of every single function, hooked or not. This is called grabbing a fresh copy of Ntdll.dll. There are detections that can be implemented to catch us unhooking a hook by an AV solution, but they are not very common, for now. However, it is good to know now so you can try to account for it when developing your tools that are utilizing some hooking engine or custom hooking engine.

Again, if an attacker wanted, and had the correct permissions, they could patch the on-disk version of Ntdll.dll. So, if you used this method to restore a fresh copy in memory, you would actually be re-implementing the attacker's hooks.

© SANS Institute 2024

https://linktr.ee/offsecexam



A Fresh Copy Visualization

Perhaps the best way to copy over a fresh copy of NTDLL on disk is to create a file mapping. The process is not complicated at all once you become familiar with the few APIs involved. The first action to execute is to obtain a module handle to the DLL using the ***CreateFile*** API, which will return a handle to us. From the handle we can use the ***CreateFileMapping*** API to create a mapping object, which will be available to us via a file mapping handle. Using the file mapping handle, we can use the ***MapViewOfFile*** API to map the object into our address space. There is no handle being returned from this API, but rather a pointer to the base address and this address will be used to work our way through the PE headers. Once we have the address to the first section, we can start looping over the sections until we find the .text section. This is the section we need to copy over the tampered .text section in memory. At this point, everything should be back to normal.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unhooking Hooks: A Suspended Copy



Yet another way to unhook hooks

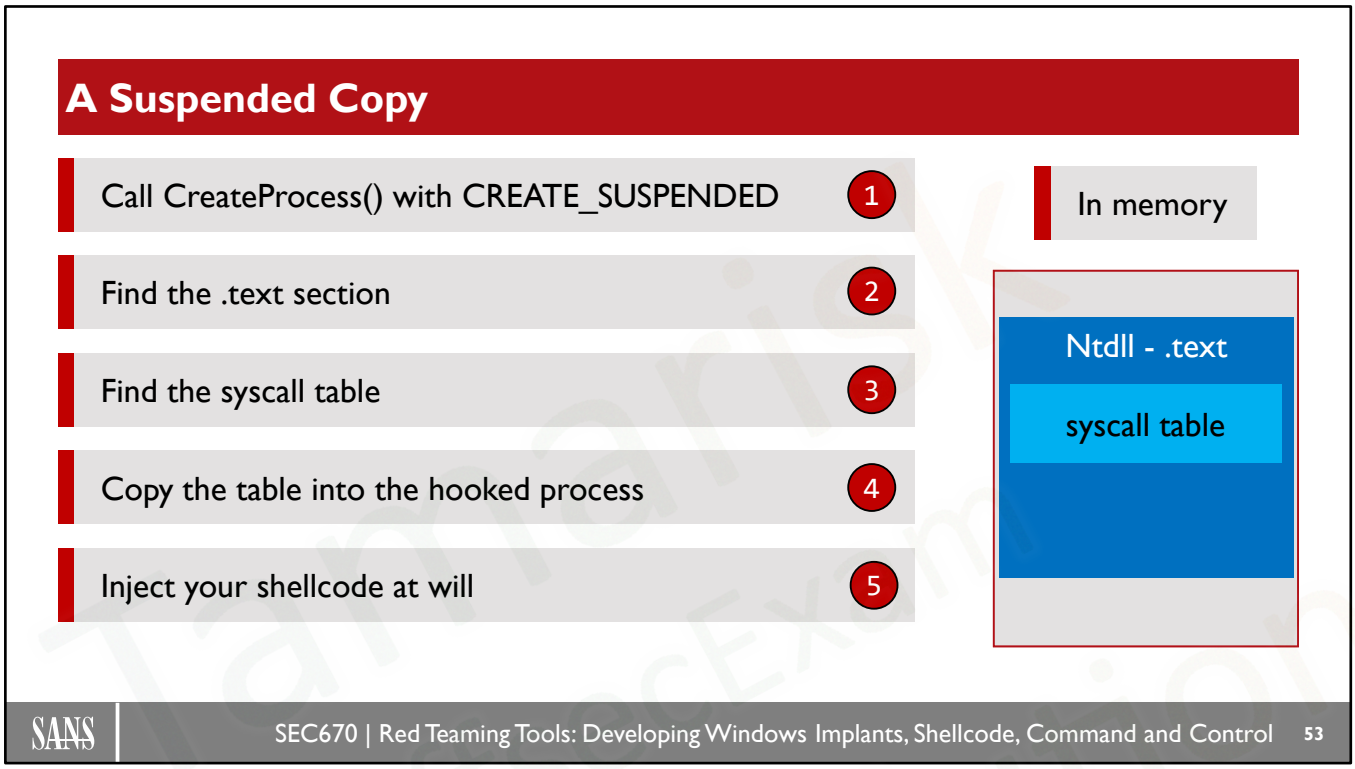
What modules are loaded when you create a process in the suspended state? A debugger can show you that only Ntdll.dll should be implicitly loaded because it is getting ready to load the image into memory. If a process is created in the suspended state, its thread does not execute yet, thus no AV/EDR hooks can be implemented yet.

Unhooking Hooks: A Suspended Copy

Grabbing the .text section of Ntdll.dll from disk is not the only option. For this one, think back to when we created a process in the suspended state so we could hollow it out and make some slight modifications. Did you happen to notice what modules are loaded at that point? You should only see Ntdll.dll because no actual program code has been executed just yet. The main thread has not been allowed to run because we suspended it via passing the suspended flag to the ***CreateProcess*** API. The reason Ntdll.dll is there is because it is trying to do its job of mapping the image from disk and into memory, but it only got so far. This is good news because the AV/EDR solution has not been provided the opportunity to implement its hooks! There is nothing to hook at this point since the process is in the process of being initialized. At this phase of initialization, we can grab the information we need from Ntdll.dll and use it to fix our tampered hooked version. Simple and effective.

© SANS Institute 2024

https://linktr.ee/offsecexam

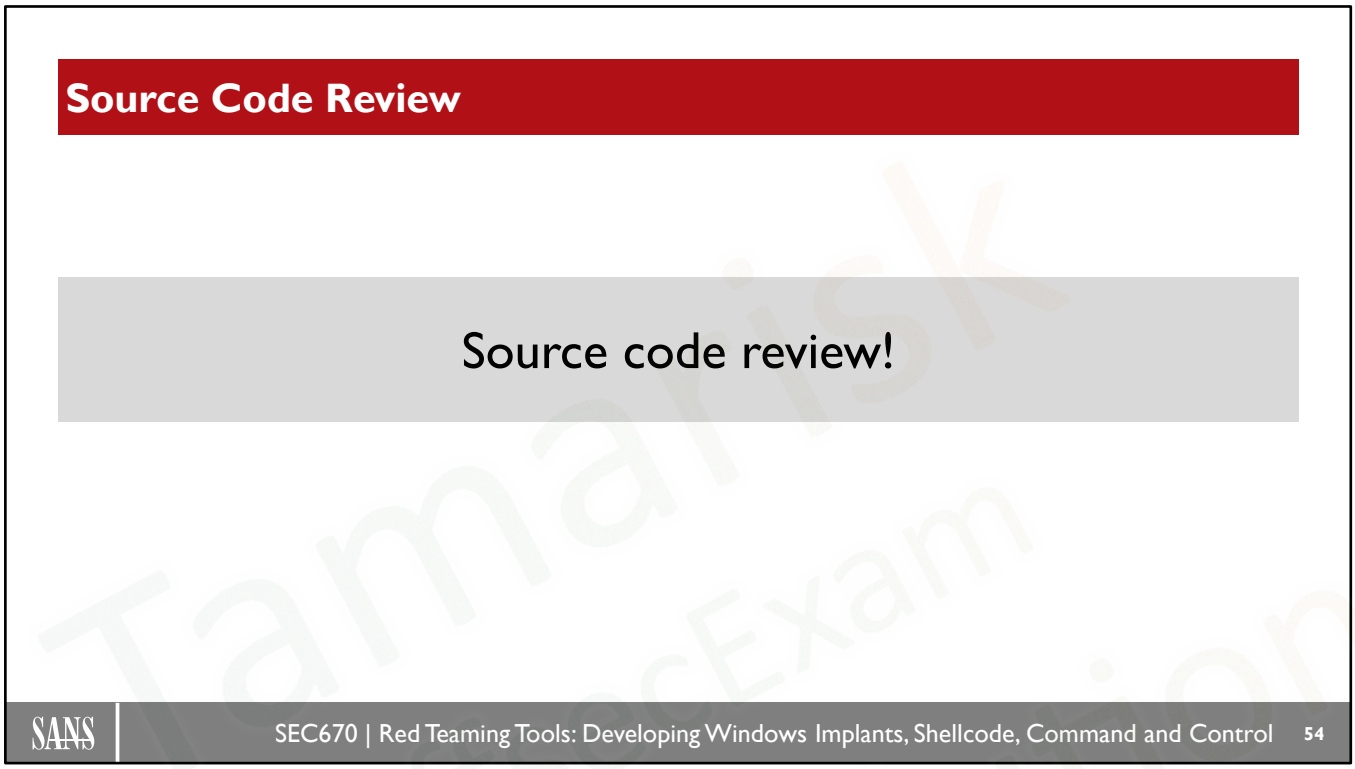


A Suspended Copy

The process of grabbing the ntdll syscall tables from the suspended process is not very complicated because most of the logic you have seen already. The only new item with this technique is finding the syscall table that resides in the .text section. There is no perfect way to search for the table boundaries, so you can just implement a loop and seek out the pattern of bytes that we know to be a syscall. This is similar to pattern scanning when trying to locate non-exported functions, except we are looking for syscall patterns. To summarize, we are attempting to overwrite a tampered syscall table with a clean syscall table from a suspended process.

© SANS Institute 2024

https://linktr.ee/offsecexam



Source Code Review
Time to jump into the source code and explain it.

Tamarisk
OffSecExam
Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 5.2: UnhookTheHook



Test your unhooking skills against Bitdefender and others.

Please refer to the eWorkbook for the details of the lab.

Lab 5.2: UnhookTheHook
Please refer to the eWorkbook for the details of this lab.

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to explore one of several methods of clearing out user mode hooks that an EDR's DLL might implement. The kernel mode side of an EDR will be discussed in follow-on courses like SEC770.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

More Techniques



Never just one way to do a thing

Syswhispers1, 2, 3

Syswhispers3: WoW64, EGGs, direct syscall jumps in both WoW64 and x64, direct syscall jumps to random syscalls

```
py syswhispers.py --preset common -o syscalls_common -m egg_hunter
```





More Techniques

As with all things programming, there is never just one single way to get something done. Some methods might seem similar when you look at the source code, but they can still produce a different signature. The beauty of programming! The same thing goes for how we invoke syscalls, how we find the syscall numbers that have been hooked, etc. One tool that can be used to help us out is Syswhispers, their GitHub link is referenced down below, but that project is designed to produce a few files to use in your own project. Specifically, it will generate header files and assembly files in pairs for syscalls found in the current image kernel, which is typically ntoskrnl.exe. One way to run their python script is to have it generate egg-hunting patterns as a way to bypass the "mark of the syscall." This will generate a few files that need to be imported into your VS project. You should see a header, a source file, and an assembly file. If you attempt to build your project right after adding those files, you will probably be greeted with a few errors. You probably did not enable MASM support in your project as it is something that is not enabled by default. This setting is under *Build Customizations* under the Project's settings. One of the last things you must do is change a property for the assembly file. Right click on it and choose properties and then set its *Item Type* to *Microsoft Macro Assembler*. Now, you should be all set to carry on with the build of your project. One of the items to keep in mind is that if you are targeting a GUI application that will invoke GUI syscalls, syswhispers3 does not currently support them, but they might in a future update.

Reference:
https://github.com/klezVirus/SysWhispers3

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed why we would unhook hooks
- Found hooked functions
- Explored various ways to unhook hooks
- Implemented your own hooks

Module Summary

In this module, we discussed several reasons as to why you might want to unhook functions that were already being hooked by something else or someone else's malware. Many AV solutions will also hook functions that they think are commonly used in malware. They will also pay attention to the call order of APIs because that could be an indicator of suspicious activity.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



At what stage of process creation will you not have any hooks implemented?

A Suspended

B Terminated

C Running

Unit Review Questions
Q: At what stage of process creation will you not have any hooks implemented?

A: Suspended

B: Terminated

C: Running

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



At what stage of process creation will you not have any hooks implemented?

A Suspended

B Terminated

C Running

Unit Review Answers
Q: At what stage of process creation will you not have any hooks implemented?

A: Suspended

B: Terminated

C: Running

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



Does unhooking hooks truly blind a Security Product?

A Yes, because it will no longer have introspection into that process

B Depends, there could be a kernel module still watching

C Only if it's Defender

Unit Review Questions
Q: Does unhooking hooks truly blind a Security Product?

A: Yes, because it will no longer have introspection into that process

B: Depends, there could be a kernel module still watching

C: Only if it's Defender

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



Does unhooking hooks truly blind a Security Product?

A Yes, because it will no longer have introspection into that process

B Depends, there could be a kernel module still watching

C Only if it's Defender

Unit Review Answers
Q: Does unhooking hooks truly blind a Security Product?

A: Yes, because it will no longer have introspection into that process

B: Depends, there could be a kernel module still watching

C: Only if it's Defender

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 5

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- **Enhancing Your Implant: Shellcode, Evasion, and C2**
- Capture the Flag Challenge

Custom Loaders
Lab 5.1: The Loader
Unhooking Hooks
Lab 5.2: UnhookTheHook
Bypassing AV/EDR
Calling Home
Lab 5.3: No Caller ID
Writing Shellcode in C
Bootcamp
Lab 5.4: AMSI No More
Lab 5.5: ShadowCraft

In this module, we will cover the concepts of bypassing AV solutions and possibly EDR solutions.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss reasons to avoid detection
- Explore various methods to avoid detection
- Discuss the good and the bad

Objectives

The objectives for this module are to discuss some of the reasons why you would want to avoid detection, explore some of the various implementations that have been created to assist you in avoiding detection, and discuss the good and the bad of some of those methods.

Tamarisk OffSecExam Exam Solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Why Avoid Detection?



AV/EDR solutions can give away your presence.

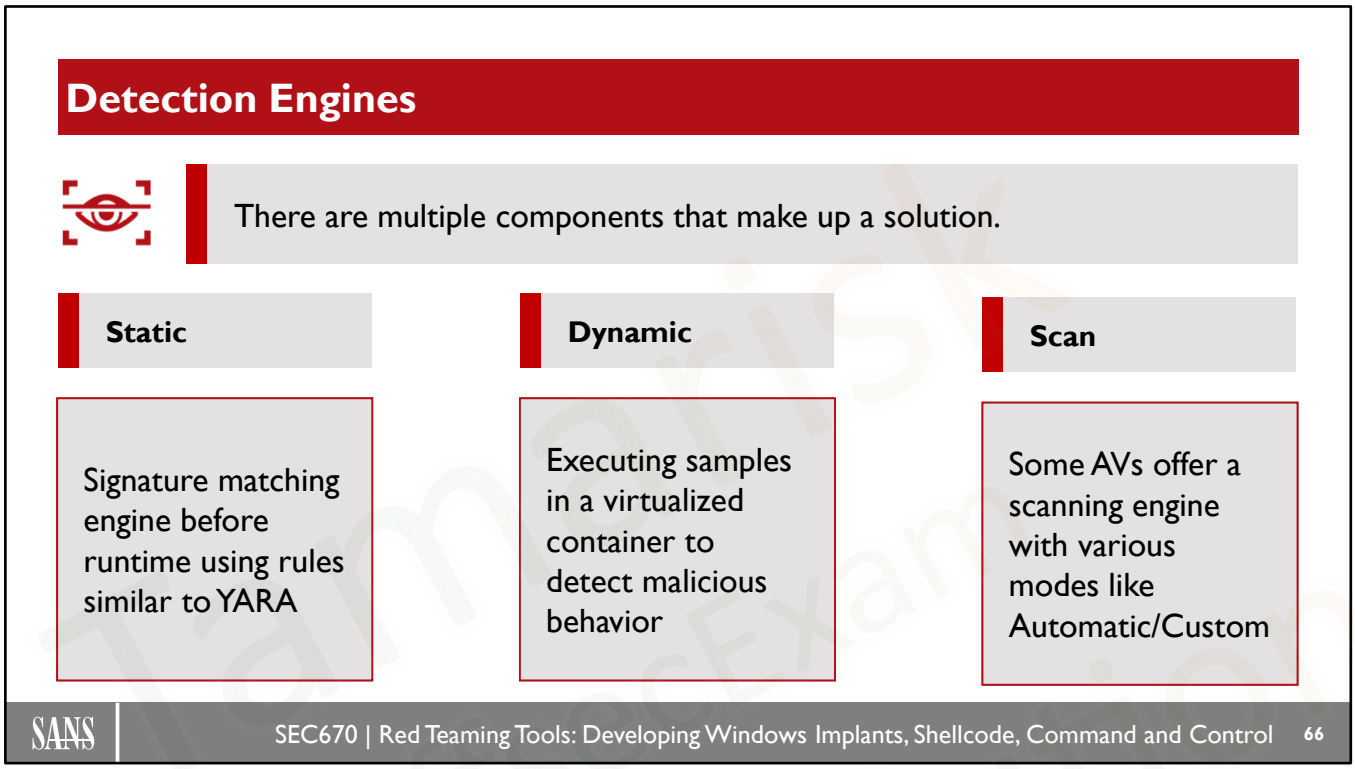
Why would you want to be detected? Unless you are intentionally trying to test an AV solution, you want to remain undetected for as long as possible. To do that, you must find a method that enables you to bypass whatever solution is being used on the target.

Why Avoid Detection?

Nobody wants to be detected immediately after gaining access to the target system, unless that is your goal. Perhaps though, you have several levels of avoidance you want to implement against your blue team so they can determine the effectiveness of their detections. With each method you implement, you increase the complexity and difficulty for being detected. As is the case with most everything, there is no single solution to a problem just like there is no single bypass that will work for every single solution on the market today. Regardless, there are some methods we will explore that are extremely effective, depending on what product you are going up against. We will look at how some AV and EDR solutions go about detecting malicious activity like hooking functions of interest. We already talked about unhooking those hooks but now we will explore additional methods.

© SANS Institute 2024

https://linktr.ee/offsecexam



Detection Engines

When it comes to protecting users and the system, it is best if the AV solution can detect the threat before it has a chance to execute, naturally. At this stage, this would fall under static analysis where a solution could use rules comprised of static signatures to detect a threat. YARA rules are very popular with malware analysts, and they offer a robust capability with how their rules are created. The downside to static rules is that they can be bypassed by changing your code base, thereby changing your tool's signature. It is great to bypass the static signatures, but we still must deal with the dynamic portion and hopefully bypass it as well. There are a few methods that could be implemented to help bypass this stage like delaying execution, exhausting resources, encryption, etc. Some AV solutions like Bitdefender, for example, offer a scanning engine that can leverage the power of machine learning to reduce false detections and provide live updates to match emerging threats. Bitdefender's scanning engine can be in Automatic mode and switched to a Custom mode for complete customization of scanning endpoints. Before we go any further, let us talk about some gates that you will have to come across.

© SANS Institute 2024

https://linktr.ee/offsecexam

Choose Your Parent Process



What if you could choose your parent process?

One method that can be used as an addition to avoid detection is to choose your parent process. There are certain processes that should never spawn other processes, including PowerShell or CMD prompt; browsers, or office applications.

Choose Your Parent Process

If you were developing an AV solution, one of the checks you might implement is the relationship between processes. Watching explorer.exe spawn a number of processes of all kinds is fine but watching a browser process spawn a PowerShell process should set off some red flags suggesting that something is definitely not right. It also would be very uncommon to see an Excel document spawn a cmd prompt. What if I told you that there was a way that we could choose the process who created us? Well, Windows gave us a pretty easy way to do just that. With that ability, we can choose to have the explorer.exe process be the parent process instead of Chrome.exe. The more we can blend in and look normal, the more time we might be able to give ourselves. We always want ample time on target to ensure our objectives are accomplished or to secure our way back into a network for future attacks.

© SANS Institute 2024

https://linktr.ee/offsecexam

Choose Your Parent Process: Implementation



Are you my parent?

Requires just two API calls

Helps avoid detection

```
//// make pointer variable
PPROC_THREAD_ATTRIBUTE_LIST pAttrList;

//// get the buffer size needed
InitializeProcThreadAttributeList(...);

//// make the real call
InitializeProcThreadAttributeList(...);

//// update the attribute before creation
UpdateProcThreadAttribute(...);
```


Choose Your Parent Process: Implementation

As with any new method, there will be a new set of APIs and/or structures to understand. Everything that is about to be done, is done before the process is ever created, which is the point because why bother changing the parent process ID after execution? As previously mentioned, we do not want to be spawning a cmd prompt from an Excel or Word document from a VBA macro payload. That would really draw some attention to our activities, and of course we do not want any scrutiny into what we are doing. The two new APIs are ***InitializeProcThreadAttributeList*** and ***UpdateProcThreadAttribute***. The pseudo code on the slide is showing what your code could look like. The first item is to create a pointer variable that will hold the buffer for our attribute list that we will allocate some space on the process heap for. After that is done, you will notice two calls to initialize the attribute list, and this is a necessity despite it seeming to be redundant. We use the first call to let the system tell us how many bytes we need to allocate on the heap for our attribute list. With the proper-sized buffer allocated, we can make the real call and then finally we can update the attribute list. At this point in the course, you have seen many APIs where we must do this two-call method.

© SANS Institute 2024

https://linktr.ee/offsecexam

InitializeProcThreadAttributeList



InitializeProcThreadAttributeList

Used to initialize an attribute list for process and thread creation

Has BOOL return type

```
BOOL InitializeProcThreadAttributeList(
 LPPROC_THREAD_ATTRIBUTE_LIST lpAttrList,
 DWORD dwAttrCount,
 DWORD dwFlags,
 PSIZE_T lpSize
);

// EXAMPLE
InitializeProcThreadAttributeList(NULL, 1,
0, &AttrListSize);

pAttrList = (...)HeapAlloc();

InitializeProcThreadAttributeList(pAttrLis
t, 1, 0, &AttrListSize);
```


InitializeProcThreadAttributeList
This API has a very descriptive name that gives away what it does—initialize an attribute list, and what it applies to: processes and threads. MSDN specifically says this API is for process and thread creation, which is exactly what we need to change a few attributes. Let us start by taking a look at the API's parameters.

lpAttributeList, is a pointer to the attribute list, but it can be NULL for the first call so we can figure out what size buffer we need to hold the number of attributes desired. The example on the slide shows this passing NULL, indicating 1 attribute, and a place to store the size, *lpSize*.

dwAttributeCount, will be the number of attributes to be added to the list. For this example, we are only adding a single item.

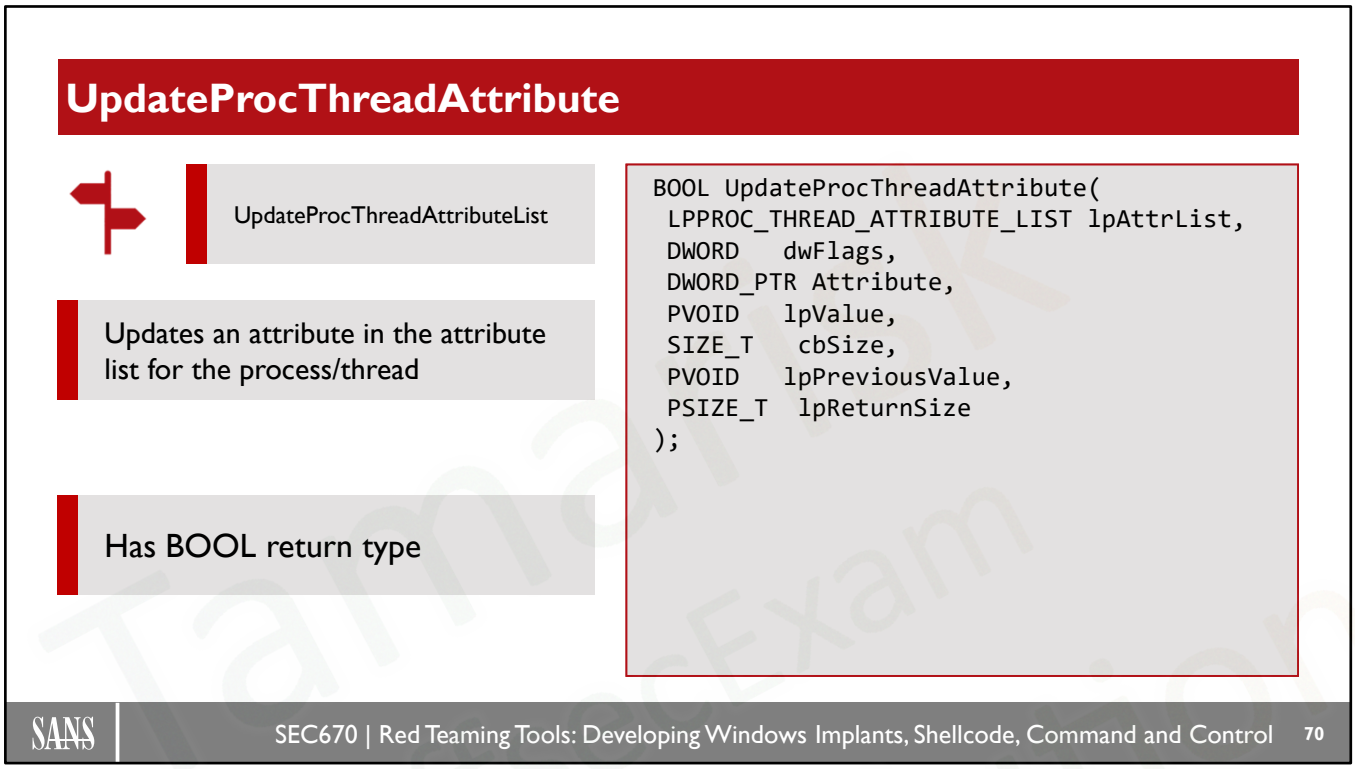
dwFlags, is a reserved parameter, so it must always be 0.

lpSize, is used to store the number of bytes needed for the buffer if *lpAttributesList* is NULL. If it is not NULL, then this will be the size of the attributes list buffer in bytes.

When we are done using the list, it is a good idea to call the ***DeleteProcThreadAttributeList*** API to free up the memory that we allocated on the heap earlier.

© SANS Institute 2024

https://linktr.ee/offsecexam



UpdateProcThreadAttribute

The ***UpdateProcThreadAttribute*** API is used when you want to update an attribute for a process or a thread. This would typically not be called until you have initialized your attribute list using the ***InitializeProcThreadAttributeList*** API. For our example, the attribute we will want to update is the parent process, but it is not obvious where to indicate that, so let us take a look at the API's parameters.

lpAttributeList, is just like the ***IntializeProcThreadAttributeList*** API, so nothing new here.

dwFlags, is also just like the *dwFlags* for the ***IntializeProcThreadAttributeList*** API, so nothing new here either.

Attribute, is our parameter of interest. It is the attribute key that will be updated and can be one of several values, but the one we are interested in is PROC_THREAD_ATTRIBUTE_PARENT_PROCESS. It is almost as if Microsoft knew that implant developers would want to change their parent PID.

lpValue, is a pointer to the attribute value that we will pass in as a handle to the process we chose to be the parent.

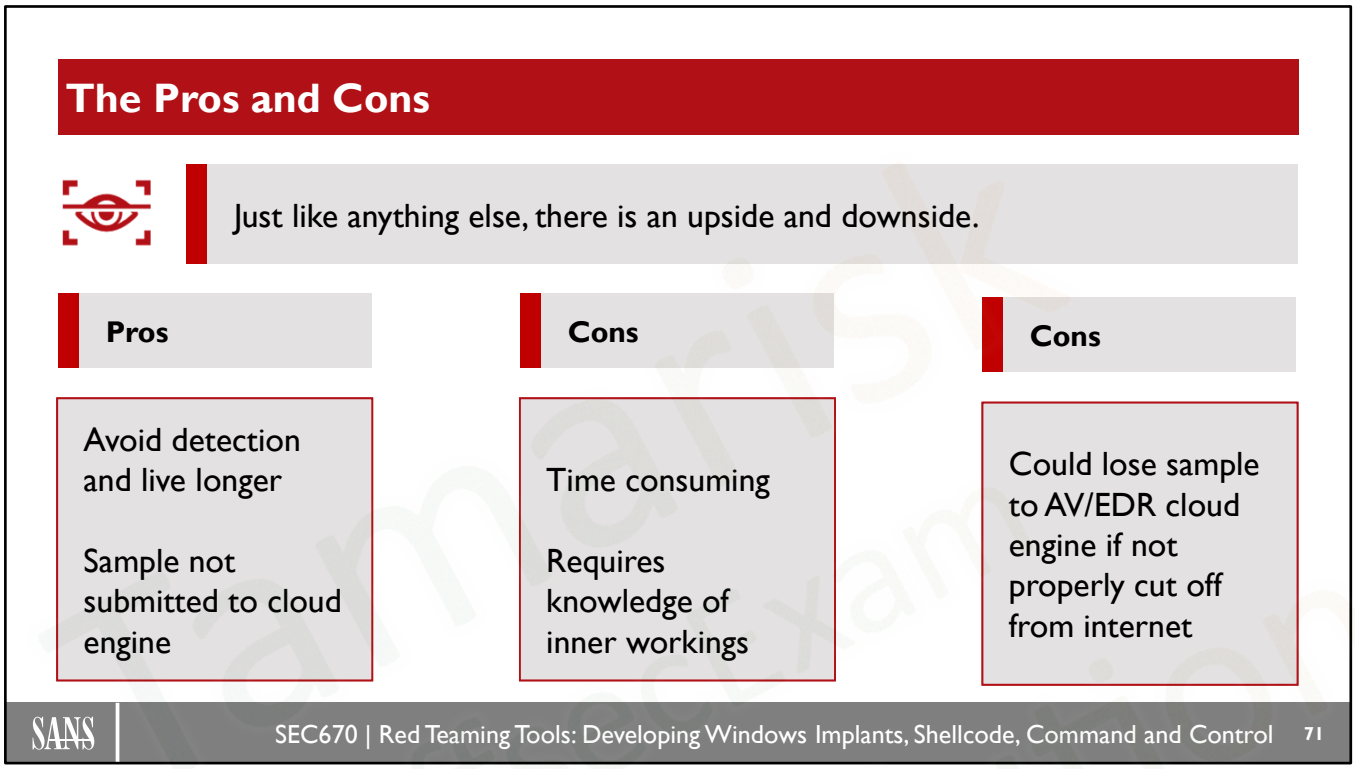
cbSize, is the size of the attribute value that was passed in for the *lpValue* parameter.

lpPreviousValue, is reserved, so it must be NULL.

lpReturnSize, is also reserved, so it must be NULL.

© SANS Institute 2024

https://linktr.ee/offsecexam



The Pros and Cons

Bypassing AV/EDR solutions is not an easy task. There are a few pros and cons with doing so and some of the pros might seem obvious. One obvious pro is that by finding a bypass, our presence on the target will not be detected. It should go without saying that the longer you can remain undetected, the longer you can maintain your presence on the target. Detections can also mean your sample gets picked up and submitted to their cloud engine for detailed analysis and the creation of a signature. The signature would then be shared with all the other systems that have their product installed. Creating bypasses can take a lot of time and research, which is best done without an internet connection because you do not want your implant to be picked up each time you are conducting a test of your bypass technique. You might also have to do some reverse engineering and debugging to see how your tools are being caught, which can also take a lot of time. Bottom line: be sure your bypass is solid before dropping some of your coveted tools on a protected target because all of that hard work and effort could be lost. If you are not certain you will not be detected, then clean yourself off the target and do so quickly.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



Learned bypassing AV/EDR should be a requirement

Discussed how avoiding detection allows you to remain on target longer

Discovered several publicly available methods exist to assist with avoidance

Module Summary

In this module, we discussed a small number of methods that can be taken to help you bypass detection and stay on target longer. There are many more that are out there and perhaps you will find new ones as you continue to go down this path.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is one risk with dropping a capability to disk with an unknown security product on the target?

A. You could lose your capability

B. You could be detected

C. All of the above

Unit Review Questions
Q: What is one risk with dropping a capability to disk with an unknown security product on the target?

A: You could lose your capability

B: You could be detected

C: All of the above

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is one risk with dropping a capability to disk with an unknown security product on the target?

A. You could lose your capability

B. You could be detected

C. All of the above

Unit Review Answers
Q: What is one risk with dropping a capability to disk with an unknown security product on the target?

A: You could lose your capability

B: You could be detected

C: All of the above

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the structure that can be used to change your parent process?

A PROC_THREAD_ATTRIBUTE_LIST

B KPROCESS

C KUSER_SHARED_DATA

Unit Review Questions
Q: What is the structure that can be used to change your parent process?

A: PROC_THREAD_ATTRIBUTE_LIST

B: KPROCESS

C: KUSER_SHARED_DATA

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the structure that can be used to change your parent process?

A PROC_THREAD_ATTRIBUTE_LIST

B KPROCESS

C KUSER_SHARED_DATA

Unit Review Answers
Q: What is the structure that can be used to change your parent process?

A: PROC_THREAD_ATTRIBUTE_LIST

B: KPROCESS

C: KUSER_SHARED_DATA

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 5

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- **Enhancing Your Implant: Shellcode, Evasion, and C2**
- Capture the Flag Challenge

Custom Loaders
Lab 5.1: The Loader
Unhooking Hooks
Lab 5.2: UnhookTheHook
Bypassing AV/EDR
Calling Home
Lab 5.3: No Caller ID
Writing Shellcode in C
Bootcamp
Lab 5.4: AMSI No More
Lab 5.5: ShadowCraft

In this module, we will cover how your implant could call home to your C2 framework, redirector, listening post, or similar.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Discuss various ways to callback
- Understand Windows Sockets
- Check in for pending tasks to execute
- Sending results back

Objectives

The objectives for this module are to discuss a number of ways to implement internet connectivity with your implant and how to check in with your C2 listening post for pending tasks. We can also discuss sending the results of commands back to your listening post with the next connection.

© SANS Institute 2024

https://linktr.ee/offsecexam

Calling Home



You are lost and forgotten about until you call home.

There are several reasons to call back home and there are several methods of doing so. We can make our own sockets, we can make web requests, talk to other services or protocols, the list of options is massive. We will discuss a few but, in the end, it is up to you to determine what is best for your mission objectives.

Calling Home

One of the first actions an implant typically carries out is an attempt to call back home to let the mothership know it is alive and well. Think of it like a check-in to say, I was just executed on computer <ComputerName> with internal IP address <IPv4> and external IP address <IPv4> on <date> <time>. The check-in would not be exactly like that, but you get the idea of what could go into it. Now that we have an idea of what a check-in might look like, how do we do it? Well, there are a few ways we could accomplish this and there also several libraries that offer some functionality for this too, like Boost. Boost is massive package to download and install, and it bloats your program with strings, program size, and other junk. Overall, I would not recommend using Boost for any of your implants. We can create our own sockets to make connections like reverse shells. There are also internet APIs we can include that would enable the ability to connect to web pages and make requests. Our C2 Listening Post could serve tasks via a web page. The simplest C2 of them all would be a lightweight Python C2 handler that hosts a few web pages like, /register or /tasks.

© SANS Institute 2024

https://linktr.ee/offsecexam

A Reverse Shell



Don't call me, I'll call you

Reverse shells are mostly preferred over bind shells because they can poke through the firewall and connect back to your C2. The shell part is where a cmd.exe process is created once the connection has been established. This is nothing as advanced as a Meterpeter session, but it is good to learn how to implement in code.

A Reverse Shell

TCP reverse shells are arguably the most common in the exploit development world, and even in the penetration testing world. For starters, reverse shells do not leave a local port listening 24/7 like a bind shell does. Also, most outbound connections are not blocked when they are initiated from behind a firewall. If you tried connecting to a bind shell to a target behind a firewall, there is a high chance that it will be rejected. Simplifying it a bit, a reverse TCP connection means that the victim, or target, is going to create the SYN packet and send it out to you, the attacker, or your C2 Listening Post. A Bind shell means the target will open a LISTENING local port and bind to that port. Typically, they would bind on 0.0.0.0:<port> so it could accept connections on any interface. If you were to run a netstat command, you could see the local port in the LISTENING state until a connection has been established. You as the attacker, or from the Listening Post, would send off that SYN packet to make a connection. If you have time, experiment with both reverse and bind shells in code to understand more how they differ programmatically.

The shell part of this comes when the connection creates a new process like PowerShell.exe or CMD.exe. This is the shell that we can use to interact live on the target. We can also create our own custom Windows shell that accepts its own commands for when a red team operator wants to go interactive on a target. One item to note when creating shells: be sure to not have the program blip on the user screen. We do not want to make the user think that something is going on with their system. Later, we will look at how we can prevent that from happening.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation (1)



Make the shell

Only if the connection succeeded

Send all handles over the socket;
STDIN, STDOUT, STDERR

```
// did the connection succeed
INT result = connect(...);

if (result != 0)
{
 return result;
}

// tell STARTUPINFO use socket for handles

// start the CMD prompt
CreateProcessA(..., "cmd.exe",...);
```


Implementation (1)
At its most basic form, a shell can be created upon successful connection to the other end, whether that be your C2 Listening Post, a test system to make sure everything is operating the way it should, or something custom you have rigged. On the slide, the connect function is being called and obviously you would pass it the required arguments, but we are more concerned about the logic and flow at this point. Success for the connect function is 0 so we make sure to check the returned value and return from the routine early should it fail. When a successful connection is made, the ***CreateProcessA*** API will be called, which will initiate the cmd.exe process. Due to the lack of space on the slide, the STARTUPINFOA struct was not able to fit, but it is a very important one to fill out properly whenever you will be creating new processes. This is mainly because the STD handles need to be redirected over the connection instead of to their normal location, the terminal window. The next slide shows the details regarding the STARTUPINFO structure.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation (2)

```
// tell STARTUPINFO use socket for handles
STARTUPINFOA StartInfo;
SecureZeroMemory(&startup, sizeof(STARTUPINFOA)); // or StartInfo = { 0 };

startup.hStdInput = (HANDLE)theSocket;
startup.hStdOutput = (HANDLE)theSocket;
startup.hStdError = (HANDLE)theSocket;
startup.dwFlags = (STARTF_USESTDHANDLES);

// the process information struct
PROCESS_INFORMATION ProcInfo = { 0 };

// start the CMD prompt
CreateProcessA(..., “cmd.exe”,...);
```


Implementation (2)

This is what was missing on the previous slide. What is happening here is after we declare and initialize the STARTUPINFOA struct, we fill out some of the struct members, so the new process knows how it is supposed to start up. Specifically, it needs to know that the STD handles for input, output, and stderror are to be sent over the socket. The variable named *theSocket* is what is holding the socket as returned from a call to ***WSASocket***, as we will learn about shortly. One thing to note is that when specifying this flag, you must set the *bInheritHandles* flag to TRUE. This is because the process being created needs to inherit the three STD handles. Play around and experiment with it during your free time and see what happens if you do not let the process inherit the handles.

© SANS Institute 2024

https://linktr.ee/offsecexam

Making Sockets



The reverse TCP connections need to be a socket.

Windows sockets somewhat resemble Linux sockets but with the major caveat of the Win32 APIs involved with creating them. You will see some of the same structures being used for both Windows and Linux.

Making Sockets

If you are familiar with Linux sockets, then your learning curve will be quite minimal for learning Windows sockets. Windows, of course, decided to bring their own APIs into play for sockets just like they did for so many other areas. For starters, there is a specific version of the Winsock.dll that must be used depending on what Windows Socket specification we are wishing to use. It is best to use the current specification, which, at the time of this course, is 2.2. There are also specific structures that only pertain to Windows like the WSADATA structure. Windows still has the traditional sockaddr_in structure, which we will be using for our reverse connection. The sockaddr_in struct is filled out similar to how it would be done on Linux. During your time as a developer, if your application needs to create sockets you will become very familiar with the WSA* family of Windows Socket APIs. We will implement a few of them today, but there are many, many more that are available to experiment with on your own time.

© SANS Institute 2024

https://linktr.ee/offsecexam

WSAStartup



WSAStartup()

Used to initiate Winsock usage

Has INT return type

```
int WSAStartup(
 WORD   wVersionRequired,
 LPWSADATA lpWSAData
);

// EXAMPLE

WSADATA wsaData;

INT result = WSAStartup(MAKEWORD(2,2), \
  &wsaData)

// TODO: error check
```


WSAStartup

To get things going with our reverse shell, we need to tell the system to initialize a few items. One of the items is initializing the usage of the Winsock.dll for our process. This is where ***WSAStartup*** comes to play. The API has two parameters described below.

wVersionRequired, is a WORD type that according to MSDN online documentation, is still waiting to be determined. We know it to be the specification of the Windows Socket that we are going to use, which will be 2.2. 2.2, is the most recent version but you could go back to 1.0. Because of the way it needs the version number, we must use the MAKEWORD macro that is defined in the Windef.h header file. It takes two arguments, a lowByte and highByte.

lpWSAData, is a pointer to a predefined WSADATA variable that will hold whatever details the function fills out for us.

Should the function fail, there could be several error codes that could be returned. Success for this API is 0. Also, to get the last error for WSA* functions, use the ***WSAGetLastError*** API.

© SANS Institute 2024

https://linktr.ee/offsecexam

WSASocket



WSASocket()

Used to create a socket

Has SOCKET return type

```
SOCKET WSAAPI WSASocketA(
 int         af,
 int         type,
 int         protocol,
 LPWSAPROTOCOL_INFOA lpProtocolInfo,
 GROUP        g,
 DWORD        dwFlags
);

// sister function
socket();
```


WSASocket

Before we can try to callback and check in with our C2 Listening Post, we need to create a socket and set it up accordingly. The ***WSASocket*** API is what we will be using to get this done. The return type is a SOCKET, so we will be saving this in a predefined SOCKET variable type. If the API fails, it will return the value INVALID_SOCKET and we should follow up with ***WsaGetLastError***. The ***WSASocket*** API has several parameters, and their names are not as descriptive as other APIs. Regardless, they are described below.

af, is an INT type that is used to indicate the address family specification. We will be using AF_INET, or 2 for IPv4.

type, is an INT type that specifies the type of the socket. Sockets can be RAW, TCP, UDP, etc. We will be using TCP, which has streams so SOCK_STREAM, or 1, is how you can remember to tie TCP to it.

protocol, is another INT type to indicate what address family specific and socket type specific protocol to use. The constants typically begin with IPPROTO_. We will be using IPPROTO_TCP as this is really the only option for an address family of AF_INET and socket type of SOCK_STREAM.

lpProtocolInfo, is a pointer to a predefined WSAPROTCOL_INFO struct that describes the socket's characteristics. NULL is just fine for our use case.

g, is a GROUP type if you want to add the socket to an existing group or create a new group. We do not care about this so 0 is what we will use.

dwFlags, is of DWORD type and can be used to describe more attributes of the socket. NULL is fine for this parameter too.

The sister function, ***socket***, is very similar but Microsoft decided to make socket-related APIs have the ability to be asynchronous; Windows Socket Asynchronous. It is not recommended that you mix and match these APIs around in your code. If you would like to use the async capabilities, there is an overlapped flag that can be passed to make the socket have overlapped I/O capabilities.

© SANS Institute 2024

https://linktr.ee/offsecexam

Making a Windows Shell



Making an interactive Windows shell

A custom Windows shell should give the feel of an interactive command-line tool. Some built-in Windows commands have an interactive mode like diskpart. Here, the shell should have your most basic commands for an implant, like directory listings, changing directories, uploading/downloading files of interest, running commands, etc.

Making a Windows Shell

The basic idea for a shell is to implement a loop that never ends until you terminate the process or execute a command to gracefully exit the process. The loop's entire purpose in life is to keep presenting a command prompt to the user to enter commands into. To handle all of the commands for the tool, there would be some logic that would validate the command that was entered and send it off to the appropriate function for action. You can make this feel like a typical Windows built-in command that can even have a help menu or context aware help menus, and really advanced ones can implement a command suggestion if there were typos. At its most basic form, some of the commands that you would want to implement would give it the ability to display directory listings, change into a new directory, run other commands, and even upload/download files. When developing this out, it would be recommended to create one piece of it at a time instead of trying to implement everything all at once. As you build it out it can become more and more advanced, like being able to accept binary files that contain shellcode like a file named shellcode.bin. The capabilities are limited by you, the developer, and the resources at your disposal.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation



Basic shell logic

Loop never breaks to continuously process commands

Logic to read the command line and process the args passed

```
// the pseudocode
while (TRUE)
{
 printf("RAT670 $> ");

 ReadCommandLine();

 ProcessArgs();

 ExecuteArgs();
}
```


Implementation

A contrived example on the slide shows what the root logic might look like for a shell. To kick things off, we have a while loop with the argument of TRUE so that it never ends. Typically, this endless loop is something developers try to avoid, but we are intentional about having this one. Inside the loop, we have a print function that displays whatever shell prompt we would like to display to the user. Each shell on each OS has their own look to it, so feel free to be creative here; even the shell between the Windows command prompt and PowerShell look completely different. The shell prompt helps identify the tool or the mode. After displaying the shell's prompt, basic as it is, we are ready to accept and process shell commands. Once we validate the commands and the arguments that are passed to them, if any, we can execute them. This would be a bare bones shell, but it is something to get started with and build on as you progress.

Examples of custom shell commands could be the following list:

- checkos
- survey
- runcmd <Windows built-in command>
- processkill <pid or name>
- processlist
- processstart <name>
- selfhide
- selfdestruct
- help
- help <shell command>
- put <filename> <remote destination path>
- get <filename> <local destination path>

© SANS Institute 2024

https://linktr.ee/offsecexam

WinINet



HTTP or FTP, no problem for WinINet

The Windows Internet (WinINet) API is very robust when there is a need for your application to interface with HTTP or FTP servers. WinINet can handle authentication, FTP and HTTP sessions, HTTP cookies, IPv6, and more. WinINet can be used to verify if a system has an internet connection.

WinINet

The Windows Internet (WinINet) APIs are what give your application robust internet capabilities. In fact, WinINet was developed for Internet Explorer to use. If you did not have a minimal installation of Internet Explorer on the system, then you might not be able to use their APIs. Obviously, the API family can handle HTTP sessions, but it can also handle FTP sessions. If your application needs authentication, then that is supported as well in the form of a user prompt, typically. For this reason, WinINet APIs cannot be used in services. It is common for malware to use this API family, and some use it to check if the system has an active internet connection before doing anything else. A commonly used API for this check is ***InternetCheckConnection*** or an older one—***InternetGetConnectedState***. We could implement the same kind of check to see if the system is really connected to the internet before we go trying to connect to our C2.

© SANS Institute 2024

https://linktr.ee/offsecexam

InternetOpen



InternetOpenA()

Used to create the main internet handle for the session

Has HINTERNET return type

```
HINTERNET
InternetOpenA(
 LPCSTR lpszAgent,
 DWORD dwAccessType,
 LPCSTR lpszProxy,
 LPCSTR lpszProxyBypass,
 DWORD dwFlags
);
```


InternetOpen

The ***InternetOpen*** API is used to create and initialize an application's intended use of the WinINet API family. Behind the scenes, the system is preparing internal structures in anticipation of future API calls. There should never be a need to call this API multiple times unless you are trying to go through a number of different proxies, or other changes. Typically, though, a single call will be enough until your program is done. On that note, when you are all said and done with using the session and the WinINet APIs, you should call the ***InternetCloseHandle*** API to shut it all down.

The parameters are described below.

lpszAgent, is a pointer to a constant NULL-terminated string that is to be used as the User-Agent or the name of the program that is calling the WinINet APIs.

dwAccessType, indicates what the type of access should be, like INTERNET_OPEN_TYPE_PROXY for when you want to pass all request to the proxy.

lpszProxy, is another pointer to a constant NULL-terminated string that identifies the name(s) of proxy server(s) that should be used when you pass INTERNET_OPEN_TYPE_PROXY for dwAccessType.

lpszProxyBypass, is the same type as lpszProxy, but is used to identify any hostnames or IP addresses that do not need to be passed to the proxy when you pass in INTERNET_OPEN_TYPE_PROXY for dwAccessType.

dwFlags, MSDN only indicates three possible values for this: INTERNET_FLAG_ASYNC for asynchronous requests, INTERNET_FLAG_FROM_CACHE for only showing what is in the cache and errors out otherwise, and finally INTERNET_FLAG_OFFLINE, which is pretty much the same thing as INTERNET_FLAG_FROM_CACHE.

© SANS Institute 2024

https://linktr.ee/offsecexam

InternetConnect



InternetConnect()

Used to open a session for the site passed in; can be HTTP or FTP

Has HINTERNET return type

```
HINTERNET
InternetConnectA(
 HINTERNET   hInternet,
 LPCSTR    lpszServerName,
 INTERNET_PORT nServerPort,
 LPCSTR    lpszUserName,
 LPCSTR    lpszPassword,
 DWORD     dwService,
 DWORD     dwFlags,
 DWORD_PTR   dwContext
);
```


InternetConnect

The ***InternetConnect*** API is called when you want to create an internet session. The session can be for HTTP or for FTP, nothing else. Upon success, the API will return a HINTERNET handle for the session, but should the API fail, it will return NULL. To see why it really failed, you should make a call to ***GetLastError***. The parameters are described below.

hInternet, is the internet handle as returned by a successful call to ***InternetOpen***.

lpszServerName, NULL-terminated string identifying the hostname of the request being created. An IP address can be just fine here too but doing so makes your traffic stick out a bit more.

nServerPort, indicates the port that is to be used. Can be INTERNET_DEFAULT_FTP (21), INTERNET_DEFAULT_HTTP_PORT (80), INTERNET_DEFAULT_HTTPS_PORT (443).

lpszUserName, NULL-terminated string of the username to use for the request. The function will use "Anonymous" when the parameters is NULL and the *dwService* is INTERNET_SERVICE_FTP.

lpszPassword, NULL-terminated string of the user's password. FTP sessions will use the user's email address if NULL is passed.

dwService, indicates what type of service should be used for the session, HTTP, FTP, or GOPHER.

dwFlags, is dependent on what service is passed to dwService but can indicate that an FTP session use passive semantics when the flag INTERNET_FLAG_PASSIVE is passed.

dwContext, for callbacks, has program specific data that identifies the context for the program.

© SANS Institute 2024

https://linktr.ee/offsecexam

InternetOpenUrl



InternetOpenUrlA()

Used to open a specific URL for either FTP or HTTP

Has HINTERNET return type

```
HINTERNET
InternetOpenUrlA(
 HINTERNET hInternet,
 LPCSTR  lpszUrl,
 LPCSTR  lpszHeaders,
 DWORD   dwHeadersLength,
 DWORD   dwFlags,
 DWORD_PTR dwContext
);
```


InternetOpenUrl

The ***InternetOpenUrl*** API is used when you have a URL, not an IP address, that you need to query. The API will return a handle for the URL or NULL if it fails. You can always call ***GetLastError*** for the specifics, but in addition you can call the ***InternetGetLastResponseInfo*** API.

The parameters are described below.

hInternet, is the handle provided by calling ***InternetOpen***.

lpszUrl, is the NULL-terminated string of the URL to read data from and can only be one of three options: ftp:, http:, or https:.

lpszHeaders, are the headers that you might want to send to the server.

dwHeadersLength, is the length of the headers, if any, in characters.

dwFlags, is a parameter where many flags can be chosen. For example, there are flags to ignore certs, allow redirects from HTTP to HTTPS, use keep-alives, force download from origin server instead of from the cache, and so on.

dwContext, is used if there is any application specific value that would tie this operation with any application data. If you do not have anything, just use 0.

© SANS Institute 2024

https://linktr.ee/offsecexam

HttpOpenRequest



HttpOpenRequestA()

Used to create a HTTP request handle from an existing session

Has HINTERNET return type

```
HINTERNET
HttpOpenRequestA(
 HINTERNET hConnect,
 LPCSTR  lpszVerb,
 LPCSTR  lpszObjectName,
 LPCSTR  lpszVersion,
 LPCSTR  lpszReferrer,
 LPCSTR  *lplpszAcceptTypes,
 DWORD   dwFlags,
 DWORD_PTR dwContext
);
```


HttpOpenRequest
The ***HttpOpenRequest*** API is used for when you want to create your request session for some HTTP request. The API does not actually send out the request just yet but is still another step in preparing the HTTP request. Upon success, an internet handle is returned, but the API can also return NULL if it fails. If it does fail, be sure to call our go-to API, ***GetLastError***. The parameters are described below.

hConnect, is the internet handle returned from a successful call to ***InternetConnectA*** and must be passed.

lpszVerb, is the NULL-terminated string to the HTTP verb that should be used for the request. The verb must be in all caps. If you pass NULL here, the API will assume you meant to pass in GET.

lpszObjectName, is the name of the target resource like a file name or the root of the page. You could use something like L"/robots.txt" or L"tasks" to pull down the robots.txt file or the tasks your C2 has for you.

lpszVersion, would be the NULL-terminated string of the version to use, but we can just pass in NULL to use the default version of HTTP/1.1.

lpszReferrer, is the referrer to which this document or file was found, but we will just be using NULL for no referrer.

**lplpszAcceptTypes*, is a pointer to an array of NULL-terminated strings indicating what types you accept as a client program. If you do not pass in anything here, other than NULL, you are saying that you do not accept any type. However, some servers will see that as saying you only accept TEXT documents.

dwFlags, indicate the internet options to be used for the request. Some of the options you could use are INTERNET_FLAG_KEEP_CONNECTION for when you are using NTLM authentication, INTERNET_FLAG_NO_AUTH for no automatic authentication attempts, INTERNET_FLAG_PRAGMA_NOCACHE for when you do not want a cached version but make the origin server send a fresh copy, and INTERNET_FLAG_SECURE for when you want to use SSL. There are others that can be used but these could be the most relevant options.

dwContext, is used if there is any application specific value that would tie this operation with any application data.

© SANS Institute 2024

https://linktr.ee/offsecexam

HttpSendRequest



HttpSendRequestA()

Used to send the HTTP request to the target HTTP server

Has BOOL return type

```
BOOL
HttpSendRequestA(
 HINTERNET hRequest,
 LPCSTR  lpszHeaders,
 DWORD   dwHeadersLength,
 LPVOID  lpOptional,
 DWORD   dwOptionalLength
);
```


HttpSendRequest

The ***HttpSendRequest*** API is used for when you are finally ready to send out the HTTP request. Upon success, TRUE is returned and FALSE upon failure. If it does fail, be sure to call our go-to API, ***GetLastError***, because FALSE does not mean a thing when it comes to failure reasons. The parameters are described below.

hRequest, is the internet handle returned from a successful call to ***HttpOpenRequestA*** and must be passed.

lpszHeaders, is the parameter to use to point to extra headers that you would want to send out with the HTTP request. Of course, it must be NULL-terminated.

dwHeadersLength, is the size of additional headers in characters.

lpOptional, is the pointer for any additional data that you want to immediately send out after the headers. You would use this for PUT or POST operations. If you do not have anything, then just pass in NULL.

dwOptionalLength, is the size of the optional data and if you do not have any optional data, then just pass in 0.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation

```
HINTERNET hInetOpen = NULL;
hInetOpen = InternetOpenA(lpszAgent, INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);

HINTERNET hConnect = NULL;
hConnect = InternetConnectA(hInetOpen, "www.sans.org", 443, NULL, NULL,
  INTERNET_SERVICE_HTTP, 0, 0);

HINTERNET hRequest = NULL;
hRequest = HttpOpenRequestA(hConnect, "GET", NULL, NULL, NULL, dwFlags, 0);

BOOL status = FALSE;
status = HttpSendRequestA(hRequest, NULL, 0, NULL, 0);
```


Implementation

The general concept for the pseudo-code on the slide is to create our internet session via the ***InternetOpen*** API. With the session handle, we can use that for our call to ***InternetConnect***, which will give us our connection handle to the target server: sans.org. After we get the connection handle, we can pass that when we make our call to ***HttpOpenRequest*** where we indicate what kind of HTTP request we are going to be making. If that succeeds, we then make our call of the ***HttpSendRequestA*** function to send the request on the line.

There are other versions of this example that can use the Unicode versions of the APIs, but also other ***WinINet*** APIs like ***InternetOpenURL***, ***InternetReadFile***, ***InternetWriteFile***, and more.

For an oldie but goodie example of these APIs, you can check out a very old version of the Metasploit Framework before it was re-written in Ruby. The GitHub repo can be found here: https://github.com/metasploit/framework2.

© SANS Institute 2024

https://linktr.ee/offsecexam

WinHTTP



Microsoft Windows HTTP Services

The HTTP services Windows provides is a nice option to have in addition to WinINet. WinHTTP supports great features like being able to run from inside of a service and being able to impersonate. WinHTTP also supports IPv6, automatic redirects, and AutoProxy.

WinHTTP

The Windows HTTP services are not meant to be a replacement for WinINet but rather another option that can be used for non-GUI based programs. WinINet can create a prompt to the user for credentials, which is completely fine for applications like web browsers, but an implant should not be prompting the user for anything, unless perhaps it is trying to get the victim to enter credentials. WinHTTP is a great option, though, for when we want to install a malicious service that has internet capabilities, which would not be uncommon as there are many services that do have such capabilities. There is nothing that mandates using WinHTTP over WinINet; they both can be used, and you should use them both to understand better how they work. WinHTTP works around objects like the ***WinHttpRequest*** object along with WinHTTP sessions. There are also key WinHTTP APIs that create HINTERNET handles that other functions need to operate. APIs like ***WinHttpOpen***, ***WinHttpConnect***, and ***WinHttpOpenRequest*** return a HINTERNET handles that you would need to pass around to other functions.

© SANS Institute 2024

https://linktr.ee/offsecexam

WinHttpOpen



WinHttpOpen

Used to initialize use of WinHTTP functions

Has HINTERNET return type

```
WINHTTPAPI
HINTERNET
WinHttpOpen(
 LPCWSTR pszAgentW,
 DWORD  dwAccessType,
 LPCWSTR pszProxyW,
 LPCWSTR pszProxyBypassW,
 DWORD  dwFlags
);
```


WinHttpOpen

The ***WinHttpOpen*** API must be called first to indicate that your program is going to be calling additional WinHTTP functions. The system will initialize some internal structures as preparation for your program's future API calls. When the API is successful, it will return a HINTERNET handle that must be stored in a variable to use for the next WinHTTP API, ***WinHttpConnect***. You will see a pattern with how the HINTERNET return values are passed into additional functions. Should this API fail, 0 will be returned and you should call ***GetLastError*** to see the details. The HTTP session is valid until you call the ***WinHttpCloseHandle*** API. The parameters are described below.

pszAgentW, is a pointer to a constant wide char string that holds the name of the application. It will also be in the User Agent when being sent out.

dwAccessType, is the type of access being requested. For Windows 8.1 and beyond, we will be using WINHTTP_ACESS_TYPE_AUTOMATIC_PROXY will take the system proxy settings.

pszProxyW, is pointer to the variable holding the name of the proxy to use, but if you pass in WINHTTP_ACESS_TYPE_AUTOMATIC_PROXY then this value must be WINHTTP_NO_PROXY_NAME.

pszProxyBypassW, is a value that must be WINHTTP_NO_PROXY_BYPASS unless *dwAccessType* is WINHTTP_ACCESS_TYPE_NAMED_PROXY. Then you can pass it a list of hostnames of IP addresses that you do not want to be routed through the proxy.

dwFlags, can be used if you want the operations to be handled asynchronously. Passing WINHTTP_FLAG_ASYNC indicates that you have a callback ready to be notified of when the operation completes.

© SANS Institute 2024

https://linktr.ee/offsecexam

WinHttpConnect



WinHttpConnect

Used to choose the target server

Has HINTERNET return type

```
WINHTTPAPI
HINTERNET
WinHttpConnect(
 HINTERNET   hSession,
 LPCWSTR    pswzServerName,
 INTERNET_PORT nServerPort,
 DWORD     dwReserved
);
```


WinHttpConnect

The ***WinHttpConnect*** API uses the handle that ***WinHttpOpen*** returns. The API also returns an HINTERNET handle upon success and NULL otherwise. This is the connection handle created by using the HTTP session earlier. The name of this API might mislead you into thinking that a request is being sent to the target server, but nothing is being sent just yet. We are simply going through the building blocks for our request one API at a time. The parameters are described below.

hSession, is the HTTP session created by calling the ***WinHttpOpen*** API and is required to be passed.

pswzServerName, is a pointer to the constant wide char string of the target server to eventually connect. The string must be NULL-terminated as indicated by the pswz prefix. The value can also be an IP address.

nServerPort, cannot be NULL, but can be any valid TCP port number. There are constants that can be used as well, like INTERNET_DEFAULT_HTTP_PORT to indicate the use of TCP port 80.

dwReserved, is obviously reserved and therefore must be 0.

© SANS Institute 2024

https://linktr.ee/offsecexam

WinHttpOpenRequest



WinHttpOpenRequest

Used to create the request handle

Has HINTERNET return value

```
WINHTTPAPI
HINTERNET
WinHttpOpenRequest(
 HINTERNET hConnect,
 LPCWSTR  pwszVerb,
 LPCWSTR  pwszObjectName,
 LPCWSTR  pwszVersion,
 LPCWSTR  pwszReferrer,
 LPCWSTR  *ppwszAcceptTypes,
 DWORD   dwFlags
);
```


WinHttpOpenRequest
The *WinHttpOpenRequest* API is used to create the handle for our request. It is important to know that nothing is being sent out even at this point. This is the final step in building out our request as we should now have everything that we need to send the request to the target server. Upon success, the API will return a HINTERNET handle that will be used by the sending API coming up next. The parameters are described below.

hConnect, is the connection handle created earlier by calling ***WinHttpConnect*** and is required to be passed.

pwszVerb, is a NULL-terminated string of the HTTP verb to use for the request like GET. Note that it is a constant wide char pointer, so L"GET" should be used. Also note that the string must be in uppercase format as this is required by Internet Engineering Task Force. If you pass NULL here, then GET is used as the default verb.

pwszObjectName, is the name of the target resource like a file name or the root of the page. You could use something like L"/robots.txt" or L"tasks" to pull down the robots.txt file or the tasks your C2 has for you.

pwszVersion, would be the NULL-terminated string of the version to use, but we can just pass in NULL to use the default version of HTTP/1.1.

pwszReferrer, is the referrer to which this document or file was found, but we will just be using WINHTTP_NO_REFERRER.

**ppwszAcceptTypes*, is a pointer to an array of NULL-terminated strings indicating what types you accept as a client program. To keep things simple, we can just use WINHTTP_DEFAULT_ACCEPT_TYPES.

dwFlags, are the internet flags that can be used like WINHTTP_FLAG_REFRESH to indicate you do not want the cached version of the page. Another option would be to use the WINHTTP_FLAG_SECURE flag to have the connection use SSL.

© SANS Institute 2024

https://linktr.ee/offsecexam

WinHttpSendRequest



WinHttpSendRequest

Used to send the HTTP request

Has a BOOL return type

```
BOOL
WinHttpSendRequest(
 HINTERNET hRequest,
 LPCWSTR  lpszHeaders,
 DWORD   dwHeadersLength,
 LPVOID  lpOptional,
 DWORD   dwOptionalLength,
 DWORD   dwTotalLength,
 DWORD_PTR dwContext
);
```


WinHttpSendRequest

The ***WinHttpSendRequest*** API is used to finally send the request to the target server. Since this is a BOOL return type, TRUE indicates a successful call while FALSE indicates a failure. You would have to call ***GetLastError*** to get a clearer picture as to what might have gone wrong. The parameters are described below.

hRequest, is the handle that ***WinHttpOpenRequest*** returned to us and is required to be passed.

lpszHeaders, is the parameter to use to point to any extra headers that you would want to send out with the HTTP request. If you do not have anything, then pass in WINHTTP_NO_ADDITIONAL_HEADERS.

dwHeadersLength, indicates how many characters there are in the additional headers, if any.

lpOptional, is for any data that you would want to send right after sending the headers like you would be doing for any POST or PUT operation. Outside of those it would not really make sense. If you don't have any, then you can use WINHTTP_NO_REQUEST_DATA. If you do not have any additional header and dwHeadersLength is 0, just pass NULL here.

dwOptionalLength, indicates the length in bytes of the optional data. If you do not have anything to send, then pass in 0.

dwTotalLength, indicates the length in bytes of the total data that is going to be sent. Specifically, this is the Content-Length header of your HTTP request.

dwContext, is a pointer to some pointer-sized precision variable that stores a defined value that would be passed to a callback function.

© SANS Institute 2024

https://linktr.ee/offsecexam

WinHttpReceiveResponse



WinHttpReceiveResponse

Used to receive the HTTP response

Has a BOOL return type

```
WINHTTPAPI
BOOL
WinHttpReceiveResponse(
 HINTERNET hRequest,
 LPVOID  lpReserved
);
```


WinHttpReceiveResponse

The ***WinHttpReceiveResponse*** API is used whenever you need to get the response back from your ***WinHttpSendRequest*** call. Upon success, the API will have obtained the HTTP status code, like 200 OK, along with the response headers. You would then have to query for that information using the ***WinHttpQueryHeaders*** API, a very fitting name for that API—something Microsoft is very good at doing. To obtain the information from the full response body, you would have to call an API like ***WinHttpReadData*** or ***WinHttpQueryDataAvailable***. The parameters are described below.

hRequest, is the handle that ***WinHttpOpenRequest*** returned to us and is required to be passed.

lpReserved, is obviously reserved and must be NULL.

© SANS Institute 2024

WinHTTP: A Simple Example

```
// create HTTP session
HINTERNET hSesssion = WinHttpOpen(...);

// choose target server
HINTERNET hConnect = WinHttpConnect(hSession, L"www.sans.org", ...);

// create request handle
HINTERNET hGet = WinHttpOpenRequest(hConnect, L"GET", NULL, NULL, NULL, NULL, 0);

// send the request
BOOL bResults = WinHttpSendRequest(hGet,...);

// receive the response
bResults = WinHttpReceiveResults(hGet, NULL);
```


WinHTTP: A Simple Example

This is a very simple example of what your code could look like. It is being used to show the order that functions are being called and to show how the HINTERNET handles are utilized by the other WinHTTP APIs. It is not until you send your request that the types change to BOOL. Not all arguments are shown here due to the limited space as it would reduce the readability of the code. The missing parameters would distract from the main purpose of this example, which is to make a simple GET request for the root page of www.sans.org.

© SANS Institute 2024

https://linktr.ee/offsecexam

Checking In



Calling home at a certain interval

After the initial call home, it is good to perform a check-in every so often to report your status. This is also a good time for your C2 Listening Post to issue commands, if there are any. Not checking in at the expected times could indicate compromise or failure to execute further instructions.

Checking In

At this point, let us say that our implant has performed the first call back home. We told our configured C2 LP that we are alive and well and that we were going back to sleep for some random time before we check in again. From here on out, we are going to call these requests check-ins, or health checks. Some also refer to this as beaconing by sending out beacons to the C2 LP. The check-ins are important because on the C2 side, we can expect when an implant should be checking in and we would make sure the connection is ready to receive that check-in. We also would know how many missed check-ins there were, if any. One missed check-in here and it might not mean anything, but several missed check-ins in a row or some pre-determined number missed in a 24-hour period could indicate that our implant has been compromised, which is not a good sign at all. At this point, we would hope that the logic in the implant would also be tracking missed check-ins and uninstall itself from the target. Aside from that, check-ins give the C2 side a chance to issue any commands for the implant to carry out. Commands can be anything from surveys to creating an interactive shell similar to what Meterpreter can do when you execute the *shell* command. You can see how important check-ins are at different levels.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation



The check-in

As long as we are running

Check-in for tasking and then sleep

```
// the check-in
while (alive)
{
 try {
  // save the response from the check-in
  auto response = CheckIn();
  // were we tasked to do anything?
  CheckTasks(response);
 }
 // catch any exception that might occur
 catch () {};
 // sleep for a bit before next check-in
 sleep(jitter);
}
```


Implementation

As mentioned previously, the check-in is vital for not only the implant itself but also the C2 LP for which the implant is configured to check-in. The overall logic for the check-in function can be made without being too complex. At its most basic form, and as can be seen on the slide, we can implement a loop that performs the check-in and checks to see what tasks, if any, have been sent to the implant. We can wrap the code around a try/catch block because we do not want any unexpected exception crashing our implant. We can save the results from our check-in so we can parse the response for any taskings. After that process is done, we can tell the implant to hang out for a bit before it attempts to check in again. The variable jitter is being passed for our pseudo sleep function because we do not want the check-ins to happen at a fixed interval, say every 30 seconds. If we have a check-in frequency hard coded, then that value is what we want to apply some jitter so we can mix it up a bit more. Going with a frequency of 30 seconds, the jitter could be +/- some value, meaning one check-in could follow at exactly 30 seconds, another could follow at 27, then 25, then 32, then 38, and so on. The process would continue while the implant is running.

© SANS Institute 2024

https://linktr.ee/offsecexam

Sending Results



Hey, I have your answer.

Our C2 LP gave us a task, now what? The task must be executed, the results must be stored somewhere, and then the results must be sent back to the LP. The results can be sent back during a check-in, but they do not always have to wait for the next check-in.

Sending Results

At this point, we made our first call back home and we are checking in every so often, 30 seconds with a jitter. After several check-ins, the C2 LP finally sent down a task to be executed. We finally have something to do! Determining how to execute the task will be discussed later, but for now let us say that it has been executed already. The results are being stored in memory at the moment, but we could also encrypt them to a results log file whose format is only known to us. We wait for the next check-in time to expire, and we prep the results to be sent over. We can prep the results with several methods: JSON, encrypting, encoding, or something completely custom. Once you determined how the results will be stored and prepped, they can be added as a payload with a HTTP POST, or similar, request to the LP.

© SANS Institute 2024

https://linktr.ee/offsecexam

Implementation



The results

Check if a task was given

Execute the task and return results

```
// checking for tasks in the response
CheckTasks(response)
{
 // JSONify the response
 if (taskFound) {
  // each task should have a unique ID
  taskId;
  // run the task and save results
  taskResults = RunTask();
  // return the task results
  return taskResults;
 }
}
```


Implementation

The pseudo-code on the slide is extremely abbreviated, but the main idea is presented for how it could work. At its most basic level, we parse the response from the C2 LP into a JSON format so we can easily identify if there was a task. Since we were finally given a task, we execute the logic that performs the task. Also, since we now have a task, we need to give it a unique ID that way each task given to us can be tracked. UUIDs or GUIDs can be used to serve this purpose instead of using a 64-bit number. Windows has an API that can help us create one: ***UuidCreateSequential***. If you do not want to bring in another library, then you can always just roll your own, if you are up for the challenge. Now, we can run the task with its newly created Task ID and save off the results somewhere. From here, there are several ways that we can get the results back to the LP. We can have a dedicated thread go check for results and then send them off, or we can just send them off right away.

More complicated versions could make use of a thread pool that would process an internal queue of results that are pending processing for being sent off. The job of the thread pool would be to encrypt and then encode the data before being sent, and then finally send off the data to the C2 LP.

In the end, it really depends on how you want to implement it and how much time you have to dedicate to having something more structured.

© SANS Institute 2024

Serialization



Packing and unpacking

Make sure your data is in the right order for sending over the network

Use an intrinsic function, when possible, then a std:: function, then a library, and then finally manually

```
typedef struct _STUFF {
ULONG IpAddress;
std::string HostName;
std::string UserName; } STUFF, *PSTUFF;

// 16-bit, 2 bytes
_byteswap_ushort()

// 32-bit, 4 bytes
_byteswap_ulong()

// 64-bit, 8 bytes
_byteswap_uint64()

// pack into a BYTE vector
std::vector<BYTE> bvData;
auto nBytes = alpaca::serialize(STUFF, bvData);

// C++ 20
auto stuff = std::bit_cast(raw_data);
```


Serialization

This is a very small snippet of pseudo-code that describes at a high level how to pack and unpack data that is about to be transmitted on the wire. When dealing with data transmitted over the network, you need to keep in mind the endianness. The host computer is going to be in little endian where the little end is the least significant bit (LSB), which is at bit position zero (0). Take a 4-byte value as an example: 0xC0A867A4. You can use that value in that order in source code, but once the program gets it into memory, you will see it ordered a bit differently, probably something like this: A4 67 A8 C0, which is now in little endian order. The value is not in the correct order is being used for sockets or being sent over to your C2 infrastructure. The endianness for that is call big endian where the LSB is flipped on its end. There are several functions that can be used to switch endianness when needed. One of the better ones is to use an intrinsic function known to the compiler internally called, ***_byteswap_ulong()***. Call that intrinsic function with the value 0xC0A867A4 and it will be put into the correct endianness: ***_byteswap_ulong(****0xC0A867A4****).*** So now in memory it will look like this: C0 A8 67 A4, now it is ready to hit the wire or to be used with a socket.

Since this class uses C++, there are more things we can take advantage of instead of plain old C. For example, there is an awesome library called “alpaca” that can serialize/deserialize the data into a byte vector ready for transmitting. When using C++ 20, you can use ***std::bit_cast*** to change how data is reinterpreted. For example, if your C2 infrastructure is sending over a struct-packed data of information, you can quite easily unpack that using the ***std::bit_cast*** right into your own custom struct.

Reference:
https://github.com/p-ranav/alpaca/

© SANS Institute 2024

https://linktr.ee/offsecexam

Encryption of Data



CNG APIs

Need to specify what algorithm should be used and its properties

Can create or import a key followed by encrypting or decrypting the data

```
// basics steps for CNG API usage
// open the algorithm provider
BCryptOpenAlgorithmProvider(, BCRYPT_AES_ALGORITHM,,)

// set algorithm properties
BCryptSetProperty(,, BCRYPT_CHAIN_MODE_GCM,,)

// create a key
BCryptGenerateSymmetricKey()

// encrypt the data
BCryptEncrypt();

// decrypt the data
BCryptDecrypt();
```


Encryption of Data

Your data that is being sent back and forth between your implant and C2 infrastructure, you should really encrypt it. Many companies will have proxies like F5 BIG-IP or Blue Coat Proxy SG that can host certificates, inspect data, and more! When those types of products implement SSL interception to then look at the HTTPS traffic, your implant's traffic is going to get snooped on when it is actively talking. How they implement this is by having two HTTPS connections, one that goes from the appliance and your website, and one from the appliance and your host. The appliance will be making its own certificate that will be presented to the host and your host either must be configured to trust it or just be setup to automatically trust it. There are only a handful of sites or categories of sites and traffic that cannot be intercepted. Communications with your healthcare provider, your lawyer, your bank, etc. all are protected by law under HIPAA and other rulings. Your little implant's traffic though, it is all fair game unless your red team is being clever and getting their infrastructure categorized as something that cannot legally be intercepted.

Now that we know some interception is going to happen, it is best to encrypt the actual data that is inside those packets. Some implants will use a stream cipher like RC4, but it is strongly recommended against it. It is best to use a block cipher like AES. AES can be implemented in different modes with some of them being terrible and others being the best. If you have taken SEC660, you will know a bit more about AES and the block modes in a deeper detail that this course will not dive into. What is recommended is to use AES with counter mode (AES-CTR) or Galois counter mode (AES-GCM). There are modern Win32 APIs that can be used for this that are implemented in the bcrypt.h header file. The only downside is the next-gen APIs will not work on older systems. For those older targets, you will have to use the wincrypt APIs. The encrypted data should not be able to be broken, well, at least not very easily and quickly. In short, plan on your implant's traffic being intercepted and encrypt your data using AES-CTR or AES-GCM. For more information about CNG APIs, check out the link below.

Reference:
https://learn.microsoft.com/en-us/windows/win32/seccng/about-cng

© SANS Institute 2024

https://linktr.ee/offsecexam

TLS Comms



Winlnet and WinHttp

Both offer similar capabilities, and both behave relatively the same

INTERNET_FLAG_SECURE

```
// create the internet session for the implant
InternetOpen(,INTERNET_OPEN_TYPE_DIRECT,,,)

// initiate the connection to the WWW
InternetConnect(,,,,,INTERNET_SERVICE_HTTP,,)

// open the request for the HTTPS session
HttpOpenRequest(,,,,,,INTERNET_FLAG_SECURE,)

// for self-signed certs
InternetSetOption(,INTERNET_FLAG_IGNORE_CERT_CN_INVALID |
SECURITY_FLAG_IGNORE_UNKNOWN_CA)

// send out the request
HttpSendRequest()

// see what might need to be processed
InternetReadFile()
```


TLS Comms

Move over SSL, hello TLS. The Transport Layer Security (TLS) is what should be implemented for your implant's communications when you are trying to look like legit HTTPS traffic. In a nutshell, SSL and TLS work in similar fashion, but TLS is more secure and should be the only thing used. One of the first things done is the TLS handshake. The handshake is where both client and server will begin to establish secure communications with each other and send over the public key. This handshake will become the session where both parties will create keys for the session, which will then be used for all future encryption and decryption of data. TLS is great because one of the many amazing features about it is that it has a message authentication code (MAC) baked right into it. This is great because we can rest assured that data has not been tampered with, allegedly. For your implant's communications, they should be wrapped inside a TLS session and Windows has some Win32 APIs to handle it.

WinHttp and WinInet are the two most used family of APIs for HTTP communications. There are pros and cons to each family, but what we are going to focus on here for the moment is WinInet. When establishing your session with your C2 infrastructure, you will need to make sure your HTTP request handle has the proper flags specified. Flags such as INTERNET_FLAG_SECURE would be enough to just get you going. If using self-signed certs within your C2 infrastructure, you will have to account for the warnings that might get generated when invalid or unknown certs are discovered. To handle that you would use a combination of the following which can all be bitwise ORd together:

- INTERNET_FLAG_IGNORE_CERT_CN_INVALID
- SECURITY_FLAG_IGNORE_UNKNOWN_CA
- SECURITY_FLAG_IGNORE_CERT_DATE_INVALID
- SECURITY_FLAG_IGNORE_CERT_CN_INVALID
- SECURITY_FLAG_IGNORE_WEAK_SIGNATURE
- SECURITY_FLAG_IGNORE_REVOCATION

© SANS Institute 2024

https://linktr.ee/offsecexam

Cert Pinning



Let's put a pin in it

Windows, Apple, Android, and others all do certificate pinning

Pros and cons, more pros depending on your situation

```
// ask for server's cert chain context
InternetQueryOption(INTERNET_OPTION_SERVER_CERT_CHAIN_CONTEX
T)

// find first of common name, unit name, org name
CertGetNameString(CERT_NAME_SIMPLE_DISPLAY_TYPE)

// get encrypted key hash for cert context
CertGetCertificateContextProperty(CERT_HASH_PROP_ID)

// convert hash to hex bytes
std::stringstream ss;
ss << std::hex;
for (; i < hashLen; ) {
   ss << static_cast<INT>(certHash[i]);
}

// free the cert chain context
CertFreeCertificateChain()
```


Cert Pinning

Cert pinning is just shorthand for certificate pinning and is a special technique that will pin the certificate in the program itself. You can think of this pin like a signature, or thumbprint, that will be pinned in your implant and validated against the thumbprint of the cert you are talking to at the moment for that TLS session. Should the two thumbprints not match, then the entire conversation is rejected, and the implant can take whatever courses of actions it needs. Perhaps it goes to sleep for a bit before trying again, maybe it tries a different domain, or in a paranoid environment it self-destructs. What is kind of great about cert pinning is being able to detect if someone is intercepting your communications like the old school interception of Man-in-the-Middle (MITM). When a high-level certificate authority is compromised, it can be bad for everyone trusting that CA. A compromise like this took place back in 2011 with DigiNotar and Google decided to pin many of their websites' certs directly into Chrome so that any other cert being presented would be rejected. Check out the link down below for more details about cert pinning and the history of it.

For our implant, we can pin the cert we are expecting to see when we call back to the C2 infrastructure. From a coding perspective, there is not a lot of extra code that needs added to implement cert pinning. You can also implement cert pinning with both WinHttp and WinInet despite what some blogs and online documentation might tell you. For WinInet, you can query the option for the flag INTERNET_OPTION_SERVER_CERT_CHAIN_CONTEXT, which will obtain the certificate-chain for your C2 infrastructure. It will then duplicate it as a PCCERT_CHAIN_CONTEXT struct that can be used with any of the crypto APIs that take in that type. When you are done with this, it is imperative to call ***CertFreeCertificateChain*** to free up those resources.

Reference:
https://learn.microsoft.com/en-us/azure/security/fundamentals/certificate-pinning

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 5.3: No Caller ID



Use HTTP libraries to implement HTTP communications.

Please refer to the eWorkbook for the details of the lab.

Lab 5.3: No Caller ID
Please refer to the eWorkbook for the details of this lab.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

What's the Point?

What's the point?

What's the Point?
The point of this lab was to explore HTTP functions to implement HTTP requests.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Discussed calling home to say we made it
- Discussed how to call home
- Discussed how to become internet capable
- Discussed checking for pending tasks and sending results back to a listening post

Module Summary

In this module, we discussed how and why your implant should call home. We need some method of telling our C2 infrastructure that we made it. We also took a look at how we could check for any pending tasks as well as send back the results of a previously executed task.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is the difference between the WSASocket() and socket() APIs?

A WSASocket allows for redirected handles

B Socket is asynchronous

C WSASocket is a CRT API

Unit Review Questions
Q: What is the difference between the WSASocket() and socket() APIs?

A: WSASocket allows for redirected handles

B: Socket is asynchronous

C: WSASocket is a CRT API

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is the difference between the WSASocket() and socket() APIs?

A WSASocket allows for redirected handles

B Socket is asynchronous

C WSASocket is a CRT API

Unit Review Answers
Q: What is the difference between the WSASocket() and socket() APIs?

A: WSASocket allows for redirected handles

B: Socket is asynchronous

C: WSASocket is a CRT API

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



If using the WinHTTP library, what API is used to create the session?

A WinHttpOpen

B WinHttpConnect

C WinHttpCreate

Unit Review Questions
Q: If using the WinHTTP library, what API is used to create the session?

A: WinHttpOpen

B: WinHttpConnect

C: WinHttpCreate

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



If using the WinHTTP library, what API is used to create the session?

A WinHttpOpen

B WinHttpConnect

C WinHttpCreate

Unit Review Answers
Q: If using the WinHTTP library, what API is used to create the session?

A: WinHttpOpen

B: WinHttpConnect

C: WinHttpCreate

Tamarisk OffsecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



If using the WinInet library, what API is used to create the session?

A InternetConnect

B InternetOpen

C InternetCreate

Unit Review Questions
Q: If using the WinInet library, what API is used to create the session?

A: InternetConnect

B: InternetOpen

C: InternetCreate

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers

If using the WinInet library, what API is used to create the session?

A InternetConnect

B InternetOpen

C InternetCreate

Unit Review Answers
Q: If using the WinInet library, what API is used to create the session?

A: InternetConnect

B: InternetOpen

C: InternetCreate

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 5

Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- **Enhancing Your Implant: Shellcode, Evasion, and C2**
- Capture the Flag Challenge

Custom Loaders
Lab 5.1: The Loader
Unhooking Hooks
Lab 5.2: UnhookTheHook
Bypassing AV/EDR
Calling Home
Lab 5.3: No Caller ID
Writing Shellcode in C
Bootcamp
Lab 5.4: AMSI No More
Lab 5.5: ShadowCraft

In this module, we will cover how you could craft shellcode using C.

© SANS Institute 2024

https://linktr.ee/offsecexam

Objectives

Our objectives for this module are:

- Introduce shellcode creation using a higher-level language
- Coding guide for creating shellcode
- Items to keep out of your final product

Objectives

The objectives for this module are to introduce the technique of creating shellcode using a higher-level language like C. We will discuss an overall guide for how to write your C code to produce effective shellcode as well as what are some items we will not want in the final product. The compiler will almost always do something we do not want so we need to override certain items.

© SANS Institute 2024

https://linktr.ee/offsecexam

Shellcode in C



Not an expert in assembly? No problem!

To get shellcode, you do not have to program in pure assembly, unless you like a challenge. You can use a higher-level language like C and allow the compiler to do some of its magic and optimizations that we cannot manually do in assembly.

Shellcode in C

Writing anything in pure assembly is a challenge that not many take on as it is not for the faint of heart. There are certain portions of the Windows Kernel that are done in pure assembly because of the need for efficiency and perhaps a few other reasons. Exploit developers sometimes create and/or modify ROP gadgets, written in pure assembly, when crafting an exploit against a target. Implant developers sometimes must create small features in pure assembly like creating syscall stubs. Regardless of the role, you might need to develop some part of your implant in assembly, thus requiring a solid understanding of assembly. Assembly does offer the capability for more advanced items but implementing some of those only comes with time and advanced knowledge of assembly. When getting started, it is common to make mistakes and have errors, and a very common error is stack alignment. We want to keep making sure our stack is properly aligned when doing shellcode because the CPU requires it, and it comes into play when making calls to Win32 APIs because they expect to be called with proper alignment. Because we do not have to be experts in assembly, we are going to take full advantage of using a higher-level language and convert it to our shellcode.

Doing development using Visual Studio is great because of the IntelliSense, and with the new AI auto completion in Visual Studio 2022, developing code just became a lot easier. You do not get any of those features, and more, when developing in assembly. Another advantage is with cross-compilation using Visual Studio, we can target x86 and x86_64, or even ARM if so desired, all from one source code project.

© SANS Institute 2024

https://linktr.ee/offsecexam

Coding Techniques



Assembly is great, but C can be even better.

There are several requirements for coding in C. One is the code must still be position independent (PIC). Our code must not have external references or external dependencies. We also must not have references to the data sections like we would have with initialized strings.

Coding Techniques

It should go without saying but it must be said anyway: we must maintain position independency when creating our shellcode. This requirement does not change just because we are using C to create our shellcode. The position independent requirement also means that we cannot delegate the responsibility of resolving functions/dependencies because those would become external references to somewhere else. A hard-and-fast requirement is that our code should not have any external references unless we resolve them manually on our own. In the end, it would be best not to have external references whatsoever. Also, we do not have the luxury of being loaded by the system loader and having it resolve our APIs for us. We cannot even rely on dynamic resolution using ***GetProcAddress*** because we do not even know the address of ***GetProcAddress***, so we are going to have to find that on our own.

Another cool feature we can take advantage of is compiler intrinsic functions. Compiler intrinsic functions are functions that are not stored in any DLL or other library—they are internal to the compiler itself, hence the name compiler intrinsic. Intrinsic functions have lower overhead than a typical function call would because intrinsic functions are forced inline. Also, since we can no longer write inline assembly for x64 Visual Studio programs, we must use intrinsic functions instead.

The intrin.h header file is an excellent resources for looking at how some of the intrinsic functions are prototyped.

Reference:
https://docs.microsoft.com/en-us/cpp/intrinsics/compiler-intrinsics?view=msvc-170

© SANS Institute 2024

https://linktr.ee/offsecexam

Intrinsic Functions

```
#include <intrin.h>
// specify single intrinsic function
#pragma intrinsic(memset)
#pragma intrinsic(__movsb)
// comma separated list of functions
#pragma intrinsic(strcmp,memcmp,memcpy,strcpy)

UCHAR decryptedShellcode[10];
UCHAR encryptedShellcode[10] = "\xFC\xC4\x90\x90\x90\x90\xCC\xCC\xCC";

// decryption happens before the move happens

__movsb(decryptedShellcode, encryptedShellcode, 10);
```


Intrinsic Functions

Some C runtime (CRT) functions are available as an intrinsic function, which is very nice to leverage so we can avoid calls to those functions. To enable this, we use a pragma directive like *#pragma intrinsic(memcmp)* or *#pragma intrinsic(strcmp)*. Some other ones are *__writemsr* (for Kernel fun), *__readfsdword* (for reading a DWORD value from FS segment register), *__readgsqword* (for reading a QWORD value from the GS register), or *__writegsqword* (for writing a QWORD value to an offset of the GS register).

The examples on the slide are showing how to tell the compiler that it should use the intrinsic equivalent of the functions specified. The first few examples show how to specify a single function like ***memset***. The next example shows how to make a list of functions that should be made into intrinsic functions. The functions can be called just like you would normally do. The example of calling an intrinsic function shown is of the __movsb which will take the contents of *encryptedShellcode* and move them into *decryptedShellcode*. There is a massive list of CRT functions that have intrinsic equivalents, so get to know them.

References:
https://docs.microsoft.com/en-us/cpp/intrinsics/compiler-intrinsics?view=msvc-170
https://docs.microsoft.com/en-us/cpp/intrinsics/x64-amd64-intrinsics-list?view=msvc-170
https://docs.microsoft.com/en-us/cpp/c-runtime-library/run-time-routines-by-category?view=msvc-170

© SANS Institute 2024

Avoiding Strings/Global Variables



Strings can easily give you away.

Avoid references to the data section.

Can still be found via static analysis in IDAPro or Ghidra

```
// change this
char charName[] = “GetProcAddress”;

// or this
char* charName = “GetProcAddress”

// to this
char charName[] = {‘G’,‘e’,‘t’,‘P’,...,0};

// avoid global vars
int g_c2port = 4444;
char* g_charName = “LoadLibraryA”;
```


Avoiding Strings/Global Variables

Strings can be a dead giveaway as to what your implant can do, and they are also kind of easy to obfuscate or hide. When a malware analyst is ripping apart your implant, one of the areas an analyst will most likely look at would be strings. The Sysinternals suite offers a tool called *strings* that will comb through your implant and display anything that looks like a string. Most strings are NULL-terminated, so that could be one identifying factor when automated tools are looking for strings. When you declare and initialize string variables, like the two variables on the slide, those will easily be found using the strings program or the Strings subview window in IDAPro. We want to avoid this, so one easy step we can implement is creating an array of the individual letters of our string. We of course need to NULL terminate the array with either a ‘\0’ or 0. Please note that is not an ASCII 0, (‘0’), because of the escape character in front of it (’\’). The first two variables will be stored in the .data/.rdata section because that is where initialized variables are stored. Using the array technique, we avoid that, but even still, the hex bytes can be found and manually converted to individual chars using a tool like IDAPro of Ghidra.

© SANS Institute 2024

https://linktr.ee/offsecexam

Viewing Strings

Address	Length	Type	String
.rdata:0040...	0000002C	C	The Stack Segment starts at address: \t %p \n
.rdata:0040...	00000019	C	Please enter an number:
.rdata:0040...	0000000C	C	PID is %d\n\n
.rdata:0040...	0000001A	C	One + Your number = %d \n\n
.rdata:0040...	00000044	C	This program is held open so it may be attached with a debugger. \n\n
.rdata:0040...	0000002B	C	The Data Segment starts at address: \t %p \n
.rdata:0040...	0000002A	C	The BSS Segment starts at address: \t %p \n
.rdata:0040...	0000002B	C	The Code Segment starts at address: \t %p \n
.rdata:0040...	0000002D	C	w32_sharedptr->size == sizeof(W32_EH_SHARED)
.rdata:0040...	0000001E	C	%s:%u: failed assertion `%s'\n
.rdata:0040...	0000002B	C	../../gcc/gcc/config/i386/w32-shared-ptr.c
.rdata:0040...	00000027	C	GetAtomNameA (atom, s, sizeof(s)) != 0

Viewing Strings

This is the strings subview window in IDAPro of an example program called memtest2.exe. On the left side is the address where the string can be located, followed by the string's length, type, and the string itself. The memtest2.exe binary is holding strings in the read only data section, which is where your strings will be stored by default unless you intentionally change it. An assumption could be made about this program that there are several *printf()* statements that will be printing some of these strings out to the terminal. The purpose of the program could be that it simply obtains its own process ID, then displays some information about its sections. For our implant, we do not want the reverse engineering process to be that easy, but rather we want to make the analyst work hard for that paycheck. Obfuscate, encrypt, or otherwise hide your strings as best as possible.

© SANS Institute 2024

https://linktr.ee/offsecexam

Code Generation



Must change build settings

No need for security checks

No need for hotpatching

```
// Linker settings
No Entry Point: YES /NOENTRY (for DLLs only)
Entry Point: MyEntryPoint (this is your new main)
References: No /OPT:NOREF
Additional Dependencies: Empty
Ignore All Default Libraries: /NODEFAULTLIB
Generate Debug Info: No
Generate Map File: Yes /MAP
SubSystem: Native /SUBSYSTEM:NATIVE


// C/C++ settings
Security Check: No /GS-
Runtime Library: Multi-threaded /MT
Enable C++ Exceptions: No
SDL Checks: /SDL-
Debug Information Format: None
```


Code Generation

We want to generate the best code possible and to do that, we must turn off some options that Visual Studio thinks should be there and options that might be there by inheriting parent project settings. In a normal application, some of these options would make sense to have enabled because we want to build a secure application for our customer base. Implant developers, exploit developers, etc., do not care at all about any of that. We do not care about security cookies because nobody is going to be attempting overflows against our functions. We also do not care about other items like Control Flow Guard, making our image hotpatchable, or function-level linking. All of these options can be found in the project's settings, and you can start turning them off or choosing the values shown on the slide. Since we need self-contained, position independent shellcode, these options help get us there.

© SANS Institute 2024

https://linktr.ee/offsecexam

Encrypting/Decrypting Your Shellcode: AES (1)



AES Encryption

These versions are deprecated.

Use the Crypto Next Gen APIs instead.

```
// make an AES decrypting function
void AESDecrypt( args go here )
{
 CryptAcquireContextA();
 CryptCreateHash();
 CryptHashData();
 CryptDeriveKey();
 CryptDecrypt();

 CryptReleaseContex();
 CryptDestroyHash();
 CryptDestroyKey();
}
```


Encrypting/Decrypting Your Shellcode: AES (1)

Having raw shellcode in your binary can instantly be flagged by various AV/EDR solutions. We can easily use AES encryption and decryption in our implant thanks to the family of CryptoAPIs. At first it might not seem very intuitive to use and get up and running with it, but after a bit of source code reading along with the MSDN documentation, you will get better at it. It is mainly about contexts, hashes, and keys. When you are done you can free it all and destroy the hash along with the key. The pseudo code on the slide defines a custom function to use for decrypting an AES encrypted blob of data; in our case this would be the shellcode. The first API call is to get a handle to the container that the crypto provider offers. One key parameter is the provider type, which we will choose as PROV_RSA_AES. The call to ***CryptCreateHash*** gives us our handle to the hashed object, which is then passed to the call to ***CryptHashData***. The hash data API will keep adding the hashed data to the hash object. Finally, we call the ***CryptDecrypt*** API to decrypt the shellcode so that it is available for execution. The remaining APIs are for cleanup purposes.

© SANS Institute 2024

https://linktr.ee/offsecexam

Encrypting/Decrypting Your Shellcode: AES (2)



AES Encryption

Using CNG APIs

More advanced and extensible

```
// make an AES decrypting function
void AESDecryptCNG( args go here )
{
 BCryptOpenAlgorithmProvider();
 BCryptGetProperty();
 BCryptGetProperty();
 BCryptSetProperty();
 BCryptGenerateSymmetricKey();
 BCryptDecrypt();
}
```


Encrypting/Decrypting Your Shellcode: AES (2)
All of the APIs on the previous slide have all been deprecated, but you can still use them in your code thanks to Microsoft's tremendous effort for backwards compatibility. It is rare for Microsoft to remove APIs that have been deprecated; that is a move that Apple does. The APIs that Microsoft would like us to use are the NextGen crypto APIs (CNG). The functions on the slide are the functions that are aimed at replacing the deprecated APIs. Thankfully, the older APIs have not been removed as of this writing, but Microsoft says they can be removed without any warning. Let us walk through the flow of decrypting a blob of data using the CNG APIs.

Our first API call is to ***BCryptOpenAlgorithmProvider*** because we need to get the CNG provider initialized for our future CNG API calls. The API returns a handle to a variable that we pass it. The next call will be used to get the property value of the chosen algorithm and get the size of the buffer for the key object. The first call we pass NULL for the buffer parameter to purposefully fail the function so it will give us the size needed. The second call we can take the provided size and make our official call with a newly allocated buffer large enough for storing the key object. Then we can set a property for the chosen algorithm like BCRYPT_CHAIN_MODE_CBC. Following that, we can generate our key using the ***BCryptGenerateSymmetricKey*** API, which will take a provided key and create a key object that is then used for symmetric key encryption. Finally, we can decrypt our data using the ***BCryptDecrypt*** API. A very similar process and ordering of API calls can be done for encryption.

© SANS Institute 2024

https://linktr.ee/offsecexam

Encrypting/Decrypting Your Shellcode: XOR



- Using XOR
- XORing data with a key
- Same routine for encrypt and decrypt

```
// make an XOR decrypting function
1 void XorIt( args go here )
2 {
3  DWORD i = 0;
4  for (; i < sizeof(scode); i++)
5  {
6   scode[i] = (BYTE)scode[i] ^ key;
7  }
8 }
```


Encrypting/Decrypting Your Shellcode: XOR

There has been some discussion as to whether or not XORing data is encoding it or encrypting it. If you read about encryption, you would see there is a key that is involved, then you might say that XORing data is indeed encryption. Of course, to decrypt the data you need to know the key that was used for the XOR operation. So, yes, XOR could technically be a form of encryption despite not getting much entropy, if any at all. Encoding and decoding does not involve keys whatsoever. All you need to know is the encoding mechanism and some are, for the most part, visually identifiable, like Base64. Encoding shellcode that has NULL bytes could be a good method of getting shellcode into a buffer that can later be decoded before execution.

The small snippet of pseudo code on the slide shows what a function might look like that performs XORing of bytes. The bytes could come in as a pointer to an array, like an array of shellcode, or something similar. The loop would iterate over each index of the array and perform the XOR operation with a provided key. The result would then be stored in that same indexed position in the array, as noted at line 6.

© SANS Institute 2024

https://linktr.ee/offsecexam

Encoding/Decoding Your Shellcode: Base64

```
// can use certutil
certutil.exe -encode scode.bin scode.b64

// place resulting bytes in variable
char scode[] = "ASfdbgfndGBAthyh==";

// decode shellcode
CryptStringToBinaryA(...);

// encode raw shellcode
CryptBinaryToStringA(...);
```


Encoding/Decoding Your Shellcode: Base64

For transforming, or encoding your shellcode, there are several tools that can assist you with this. Online tools are available if you want to give up your shellcode to them, but why take that chance in the first place? We want to use something that is either native to the OS or something that we developed in house. On the slide are a few examples of tools and APIs that are readily available. Certutil.exe, a Windows command-line tool, was designed to be used for certificates, but it has features that we can use for other purposes, like taking shellcode and encoding it to a Base64 file. In addition, you can create a tool in Python to do your encoding and have the logic in your Windows tool to do the decoding. Windows provides a few APIs that can also perform encoding and decoding of data. They are described below.

The ***CryptStringToBinaryA*** , or W version, is one optional API to use to decode the Base64 encoded shellcode blob. The API has seven parameters.

1. LPCSTR *lpszString* is the pointer to string to be converted.
2. DWORD *cchString* is the number of characters in string pointed to by *lpszString* not including the NULL byte.
3. DWORD *dwFlags* is the format to be converted, CRYPT_STRING_BASE64.
4. BYTE **pbBinary* is the pointer to the buffer that will receive the bytes.
5. DWORD **pcbBinary* is the pointer to variable that holds the size of the buffer pointer to by *pbBinary.*
6. DWORD **pdwSkip* is used when bytes are needed to be skipped like when using a header. We will just use NULL here.
7. DWORD **pdwFlags* is the pointer to variable that holds the flags used for the conversion.

Another API is the ***CryptBinaryToStringA***, or W, version. This API has five parameters.

1. const BYTE* *pbBinary* - pointer to the array of bytes we are converting
2. DWORD *cbBinary* - how many elements are in the array, the size of it, but not in bytes
3. DWORD *dwFlags* - the desired resulting format for the string, CRYPT_STRING_BASE64
4. LPSTR *pszString* - the buffer that will store the converted string
5. DWORD* *pcchString* - the size of the destination buffer in chars. NULL indicates we want the API to calculate this for us.

© SANS Institute 2024

https://linktr.ee/offsecexam

Module Summary



- Covered writing shellcode in C
- Saw how higher-level languages make writing shellcode easier
- Discussed how you must know your compiler flags to keep the compiler in check
- Discussed how you can still attempt shellcoding by hand

Module Summary
In this module, we talked about how we can leverage the knowledge of C to create some shellcode. We can also leverage various compiler flags to get the job done.

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions



What is one example of an API that can decrypt Base64 encoding?

A CryptStringToBinary

B CryptBinaryToString

C DecryptB64

Unit Review Questions
Q: What is one example of an API that can decrypt Base64 encoding?

A: CryptStringToBinary

B: CryptBinaryToString

C: DecryptB64

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers



What is one example of an API that can decrypt Base64 encoding?

A CryptStringToBinary

B CryptBinaryToString

C DecryptB64

Unit Review Answers
Q: What is one example of an API that can decrypt Base64 encoding?

A: CryptStringToBinary

B: CryptBinaryToString

C: DecryptB64

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Questions

When creating shellcode using C, what do you want to avoid?

A Bugs

B External references

C The heap

Unit Review Questions
Q: When creating shellcode using C, what do you want to avoid?

A: Bugs

B: External references

C: The heap

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Unit Review Answers

When creating shellcode using C, what do you want to avoid?

A Bugs

B External references

C The heap

Unit Review Answers
Q: When creating shellcode using C, what do you want to avoid?

A: Bugs

B: External references

C: The heap

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Section 5

Course Roadmap

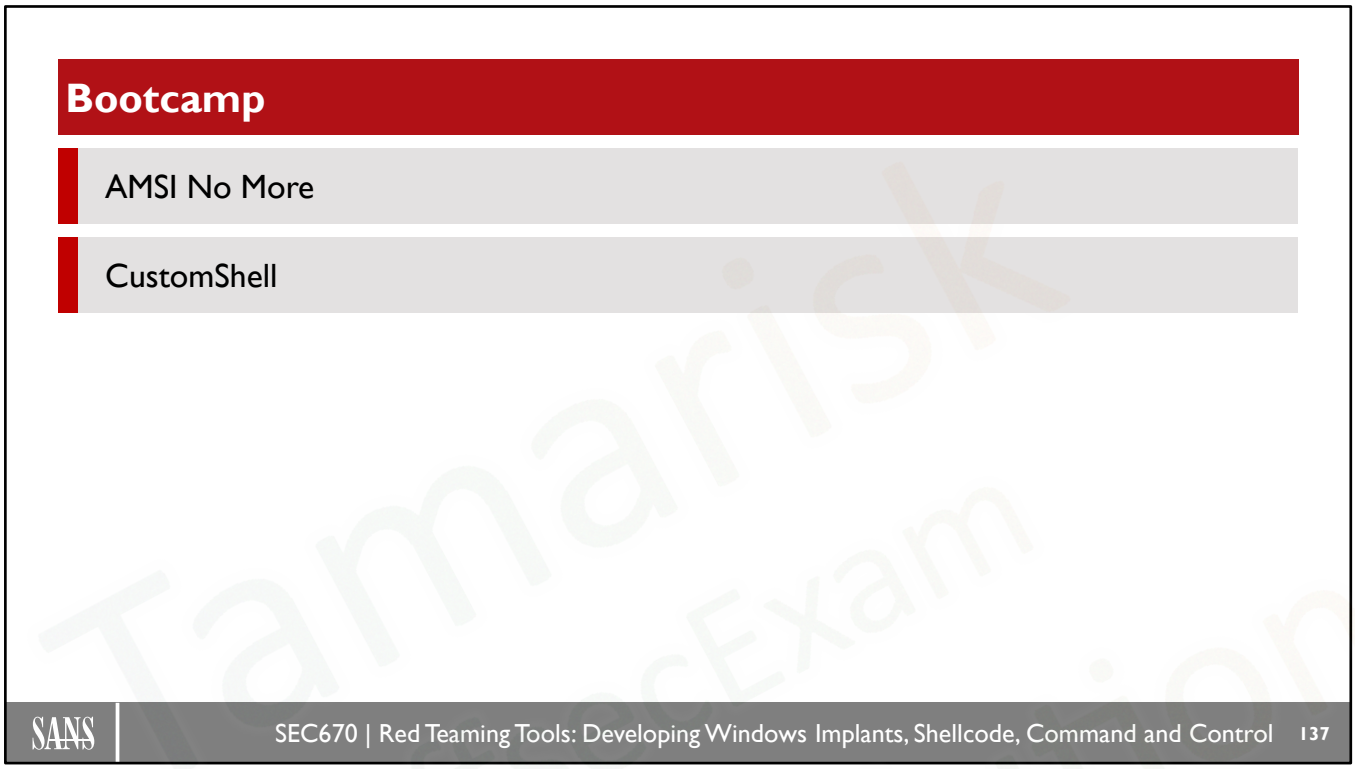
- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- Persistence: Die Another Day
- **Enhancing Your Implant: Shellcode, Evasion, and C2**
- Capture the Flag Challenge

Custom Loaders
Lab 5.1: The Loader
Unhooking Hooks
Lab 5.2: UnhookTheHook
Bypassing AV/EDR
Calling Home
Lab 5.3: No Caller ID
Writing Shellcode in C
Bootcamp
Lab 5.4: AMSI No More
Lab 5.5: ShadowCraft

This bootcamp module is going to present some of the most challenging coding challenges you have faced in this course. This final bootcamp is meant to be more hands off, meaning there will be no detailed walk-through or explanation of every step. This is because all of the foundational blocks have been covered and reinforced via labs.

© SANS Institute 2024

https://linktr.ee/offsecexam



Bootcamp
For this bootcamp, the first challenge is to use your knowledge of code caves and function hooking to disable AMSI in a PowerShell process.

The second challenge is to continue with the building of your own custom shell.

The third challenge is to create some shellcode using C or C++ and your knowledge of various compiler and linker flags. The purpose here is not to compile the entire program as PIC but only certain functions that you will inject into other processes!

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 5.4: AMSI No More

- Patch a PowerShell process with amsi.dll loaded
- Observe how data is being passed in for analysis
- Explore various methods to patch amsi.dll

Lab 5.4: AMSI No More
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam

Lab 5.5: ShadowCraft

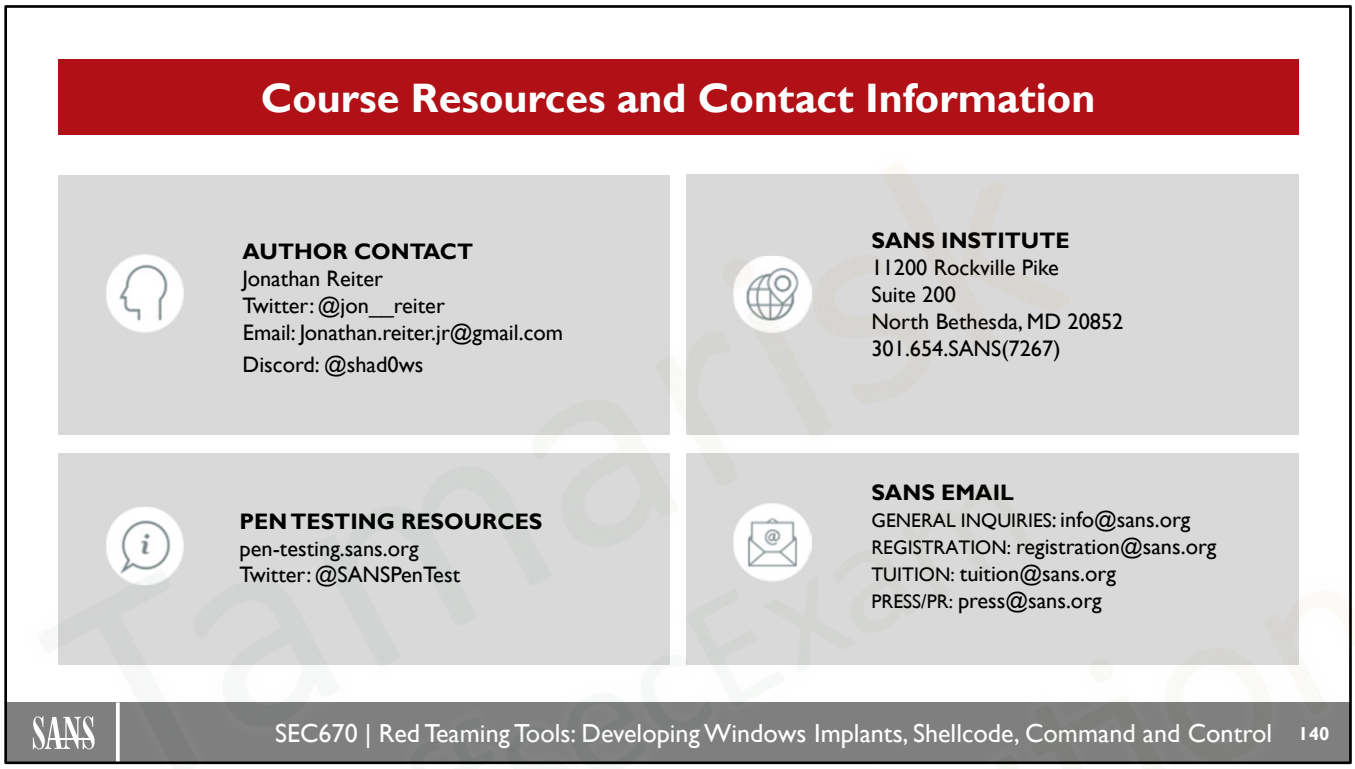
- Create a basic shell
- Implement features covered in this section
- Implement thorough error checking

Lab 5.5: ShadowCraft
Please refer to the eWorkbook for the details of this bootcamp challenge.

Tamarisk OffSecExam Exam solution

© SANS Institute 2024

https://linktr.ee/offsecexam



Course Resources and Contact Information
Please feel free to reach out to the course author.