599.3

Exploitation, Persistence, and Command & Control

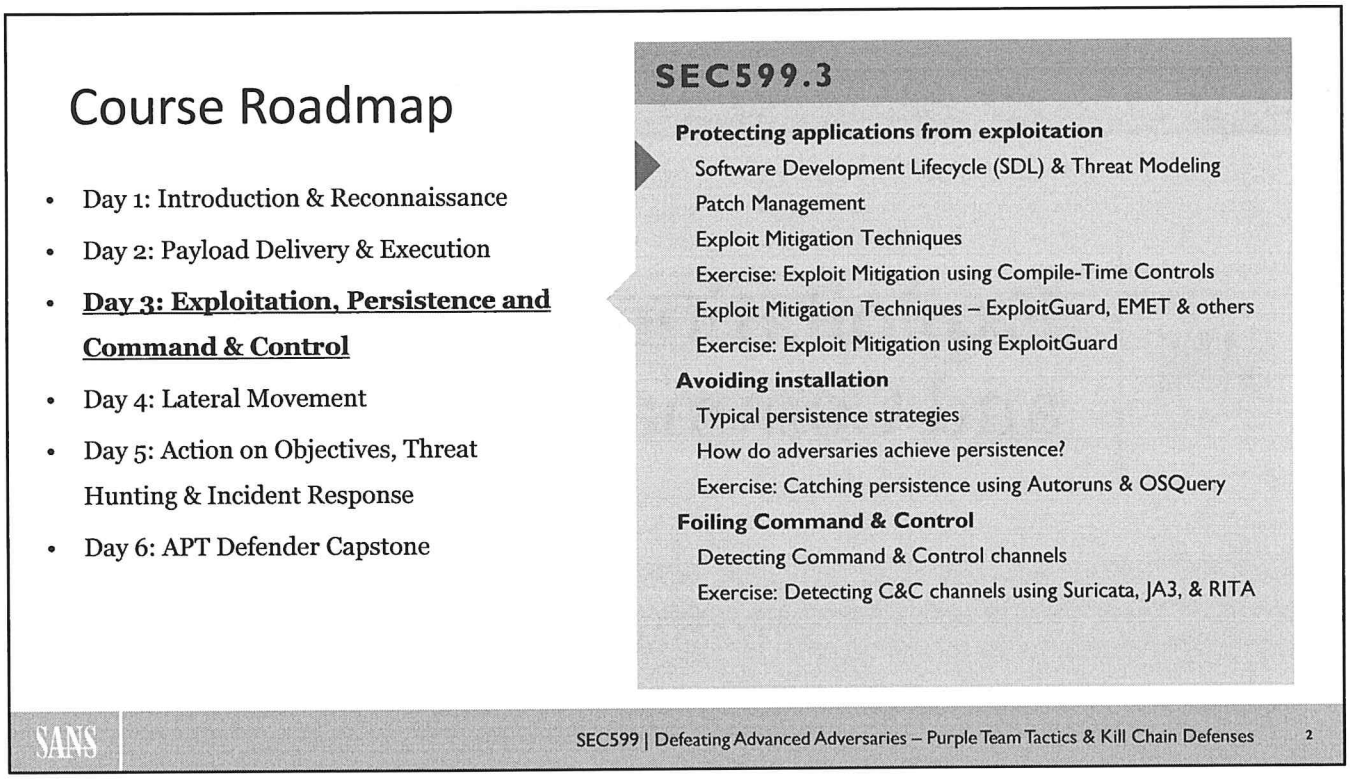
SEC599.3 | Defeating Advanced Adversaries – Purple Team Tactics & Kill Chain Defenses

SANS

Exploitation, Persistence, and Command & Control

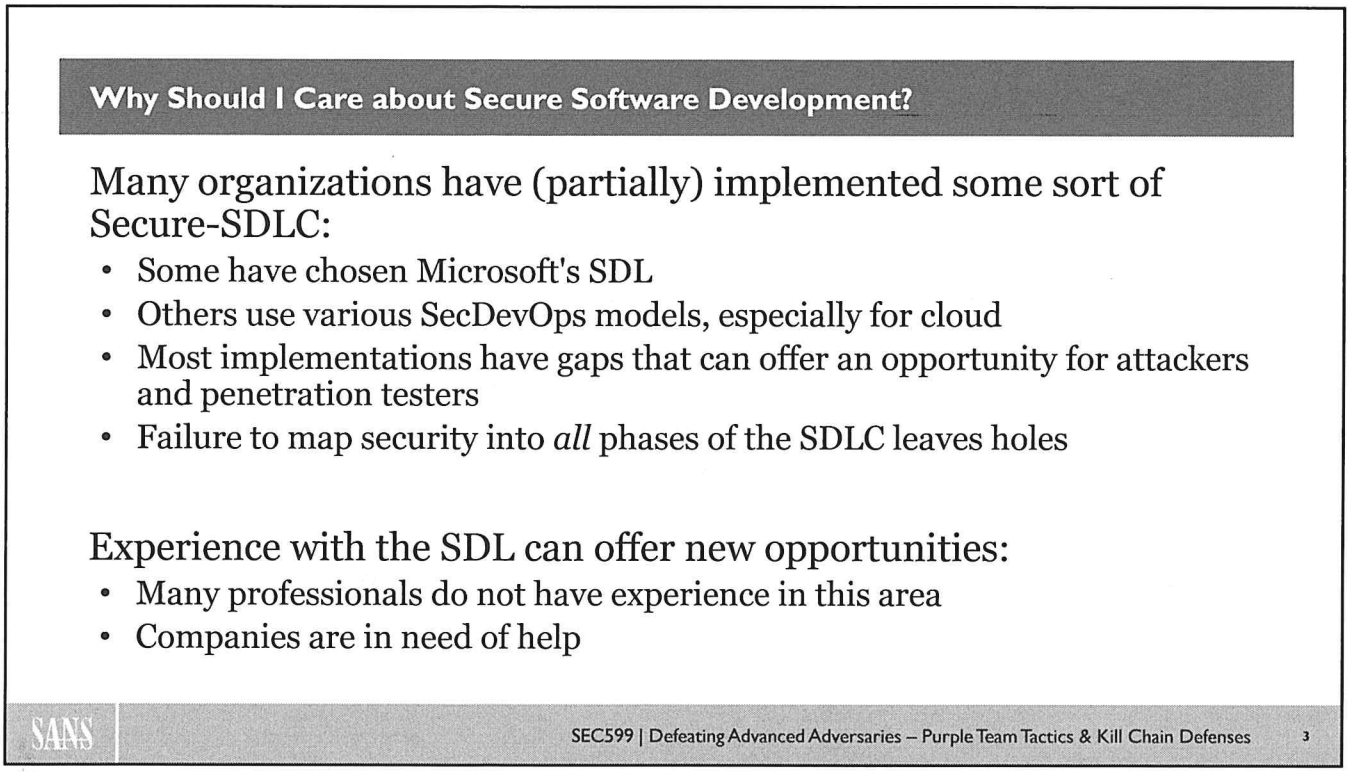
© 2019 Erik Van Buggenhout & Stephen Sims | All Rights Reserved

This page intentionally left blank.



This page intentionally left blank.

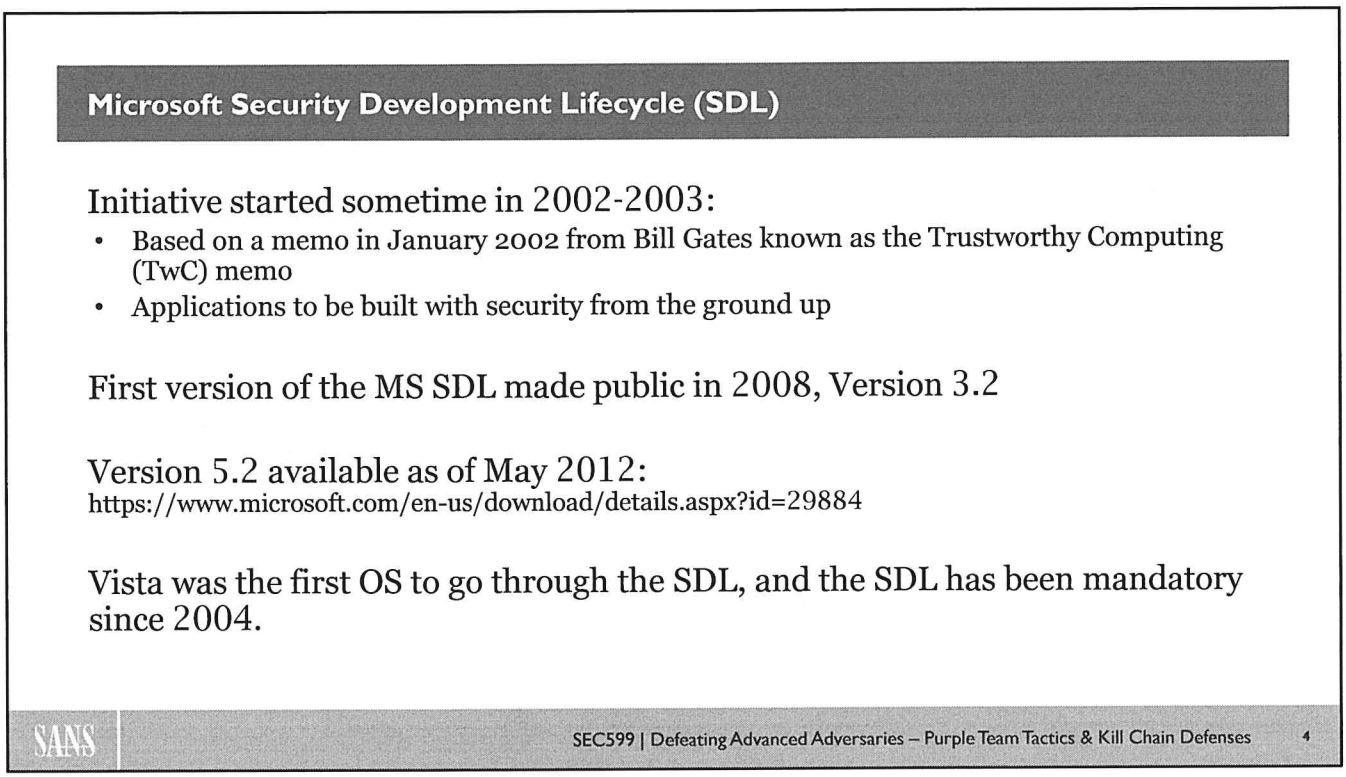
© 2019 Erik Van Buggenhout & Stephen Sims



Why Should I Care about Secure Software Development?
The question occasionally comes up about why non-developers concern themselves with Microsoft's Security Development Lifecycle (SDL) or a Secure-SDLC. The better you understand how organizations write their code, the easier it is to identify potential areas of weakness. If an organization does a good job performing peer review and static analysis, but lacks dynamic testing during the validation phase, it should be called out as a gap. This gap allows us to prioritize our time on the areas with the biggest potential for concern. Most organizations have implemented some sort of security into their development process; however, many are severely lacking. Failure to map security into each and every phase of the an SDLC leaves holes, which can be exploited.

The SDL is still a relatively new concept for most companies, and many are in need of help. Experience in this area can offer new job opportunities to a professional with the proper skills.

© 2019 Erik Van Buggenhout & Stephen Sims



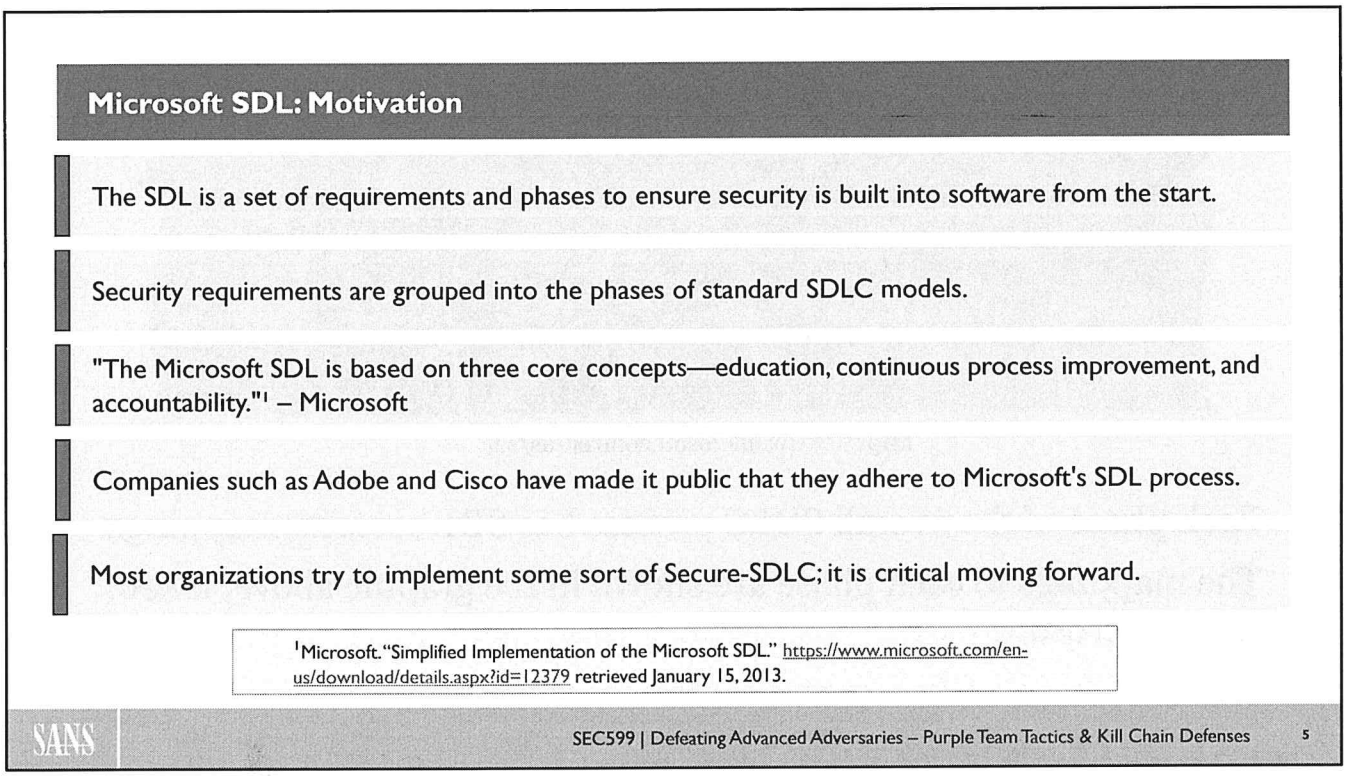
Microsoft Security Development Lifecycle (SDL)

The Microsoft Security Development Lifecycle (SDL) was started sometime in 2002-2003 to ensure that applications and operating systems are built with security from the ground up. This was during a time when Microsoft was dealing with major security issues from various pieces of malware, such as the Melissa Virus, as well as high-profile legal battles around web browser monopoly with Internet Explorer packaging. On January 15, 2002, Bill Gates sent out a memo known as the Trustworthy Computing (TwC) memo. The memo described major changes that needed to occur to ensure Microsoft and its customers were protected and that they could rely on the operating systems. The memo from Bill Gates can be read at https://www.wired.com/2002/01/bill-gates-trustworthy-computing/.

The first known version of the SDL (Version 3.2) was released to the public in 2008. Version 5.2 was the latest available version at the time of this writing and is available at https://www.microsoft.com/en-us/download/details.aspx?id=29884. Some great introductory material and presentations on the SDL are available at https://www.microsoft.com/en-us/download/details.aspx?id=16420. Microsoft Vista was the first full operating system to go through the SDL process. Microsoft also used the process to retroactively go through prior versions of code.

The contents of this module are heavily based on the Microsoft Security Development Lifecycle (SDL) and STRIDE threat modeling processes, as well as the author's experience with the implementation of Secure-Software Development Life Cycle (S-SDLC) programs in various organizations. The material written for this module references and leverages the concepts and ideas behind these models. More information on these processes can be found at https://www.microsoft.com/en-us/securityengineering/sdl/ and https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20).

© 2019 Erik Van Buggenhout & Stephen Sims

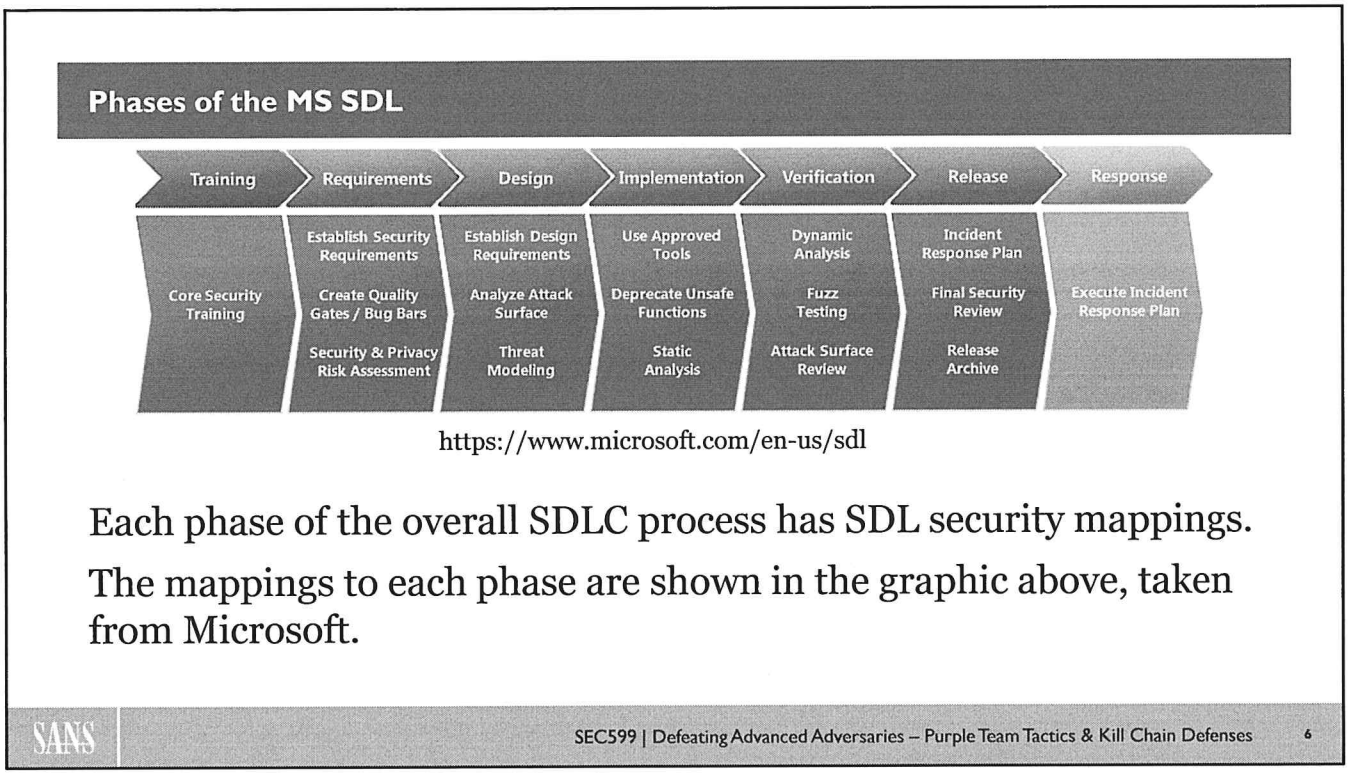


Microsoft SDL: Motivation

The Microsoft SDL is a detailed security process that must be adhered to during software development. It provides a specific group of activities to be performed during each phase of a Software Development Life Cycle (SDLC) to ensure that security is built into software from the beginning. Microsoft has various core concepts, some specific to Microsoft's deployment, such as ensuring the use of automated tools for finding bugs and other issues, tools for compliance tracking, and tools to help program managers evangelize the use of the SDL throughout various divisions and teams within the organization. Steve Lipner, Director of Security Compliance at Microsoft, has done quite a few presentations explaining how the SDL is deployed in Microsoft. Some of it can be applied to many organizations, whereas other practices are specific to Microsoft. Each organization must cater the process to their development program. During Lipner's presentation at OWASP's AppSec conference in 2010, he claimed that updates to the SDL are made only once per year and are mainly focused on the creation of new tools used for automation and fuzzing.

Companies such as Adobe and Cisco Systems have publicly stated that they adopted some or all of the Microsoft SDL. Regardless of whose process your organization adopts, the use of an overall secure SDLC is essential in this day and age.

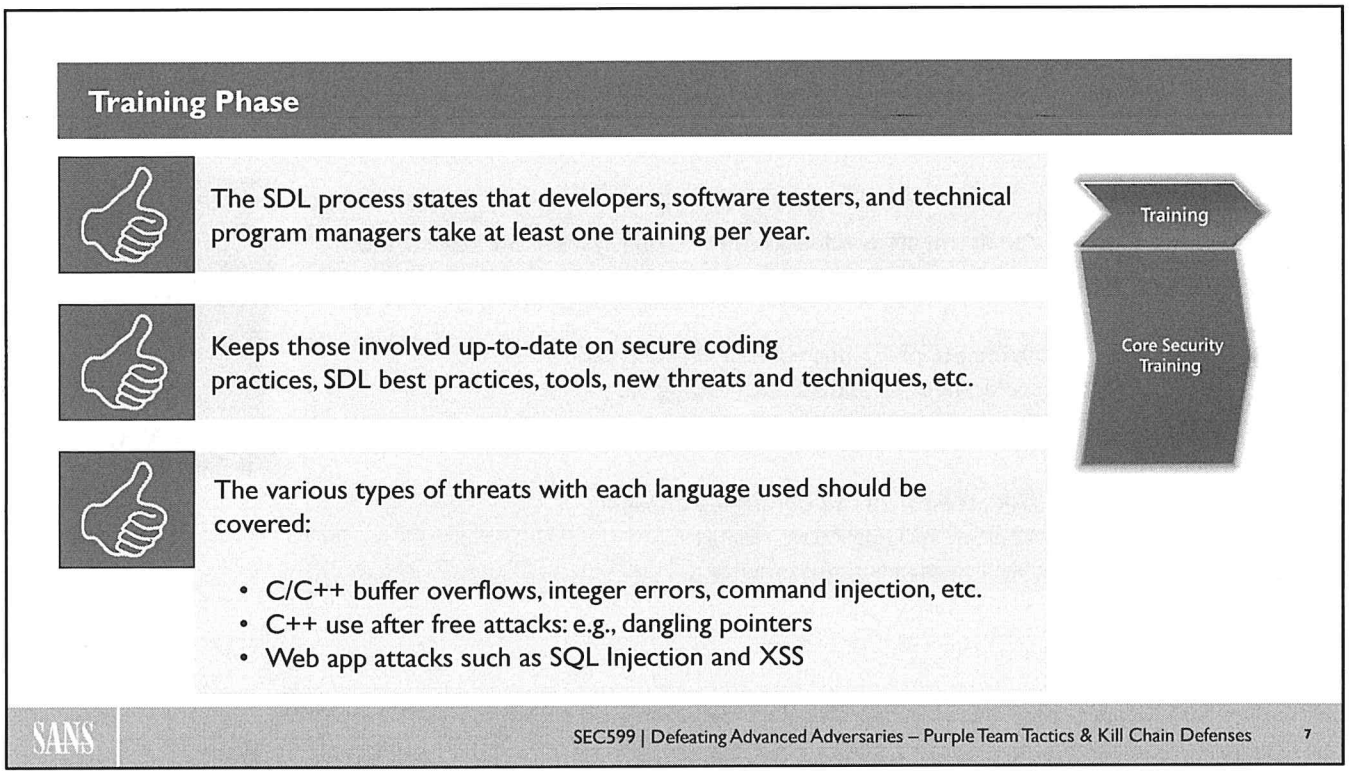
© 2019 Erik Van Buggenhout & Stephen Sims



Phases of the MS SDL

Each phase of the high-level SDLC processes listed on this slide has associated SDL security mappings. The diagram in the slide, taken from Microsoft, shows the mappings that are covered in the following slides. Microsoft changes their graph overview of the various phases periodically, so expect some differences in the way it looks over time. The SDL practices remain relatively static.

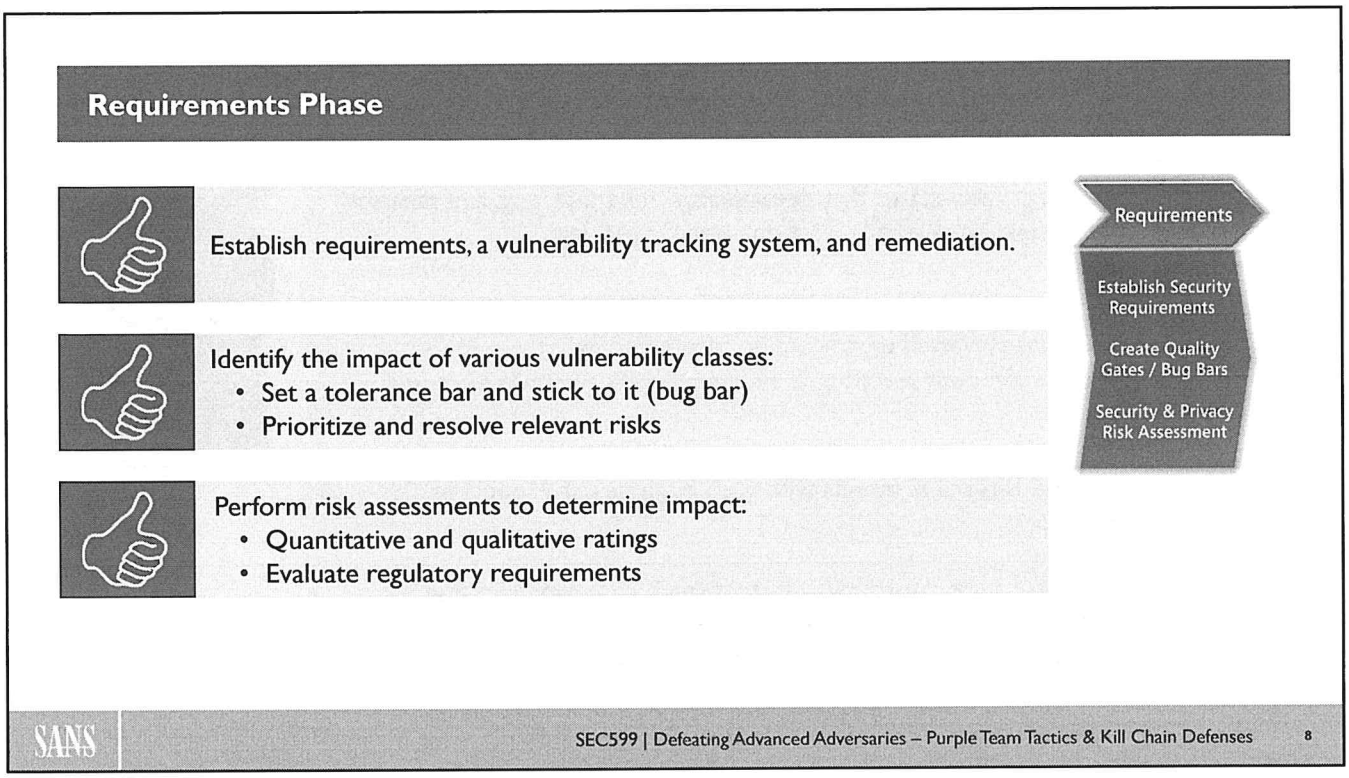
© 2019 Erik Van Buggenhout & Stephen Sims



Training Phase

According to Microsoft, leveraging the previously documented references, the SDL states that all software developers, testers, and relevant technical program managers are to attend at least one security training per year, covering each phase of the overall SDL process and specific threat types and techniques used for modeling. This requirement helps ensure that each member is up-to-date on his organization's implementation of the SDL process, new tools, threats, and commonly used techniques. The training should include any relevant languages used by the developers and it addresses vulnerability classes associated with those languages. Developers in C and C++ get training on buffer overflows, function pointer overwrites, integer errors, format string attacks, information leakage, use after free attacks, and many others. Web application developers receive training that is more focused on attacks, such as SQL injection, cross-site request forgery (CSRF), cross-site scripting (XSS), and others. Threat modeling and risk assessment techniques are also included in the training programs.

© 2019 Erik Van Buggenhout & Stephen Sims



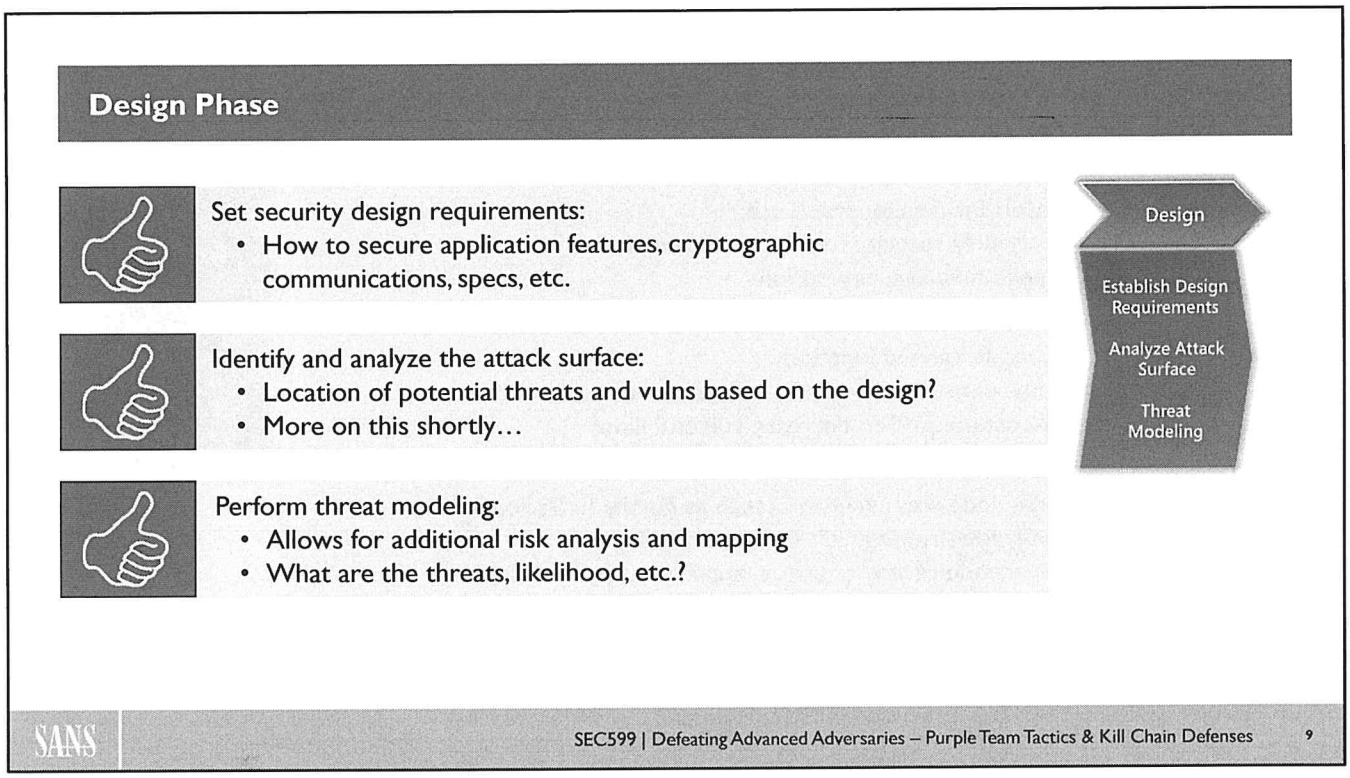
Requirements Phase

During the requirements phase, there are three main areas to cover. The first area is to establish the security requirements and to build a security team and the processes assigned to the project. This includes assigning security staff, creating a tracking system for bugs, creating a remediation process, and establishing the criteria for any privacy requirements around the data elements involved. Introduction of these items early on will help ensure a smooth process flow.

The next area is to set a tolerance bar or threshold that must be adhered to in order for the software to go into production. This typically falls in line with an organizational risk assessment process. During most risk assessment processes, the primary goal is to document all risk items, map them to policy and regulatory violations, set the initial risk rating, map in any potential mitigations, and document the likelihood and the potential impact to the organization. The difference is that with the SDL, you are setting a threshold that cannot be exceeded. This means that if it is determined during the requirements phase that no medium or high risks are permitted to go into production, they must be fixed prior to release.

The third area is centered around the identification of software features and functionality and mapping those functions to areas of concern, such as with consumer privacy, data protection, and regulatory requirements. It basically serves as a risk assessment on specific areas of the application.

© 2019 Erik Van Buggenhout & Stephen Sims



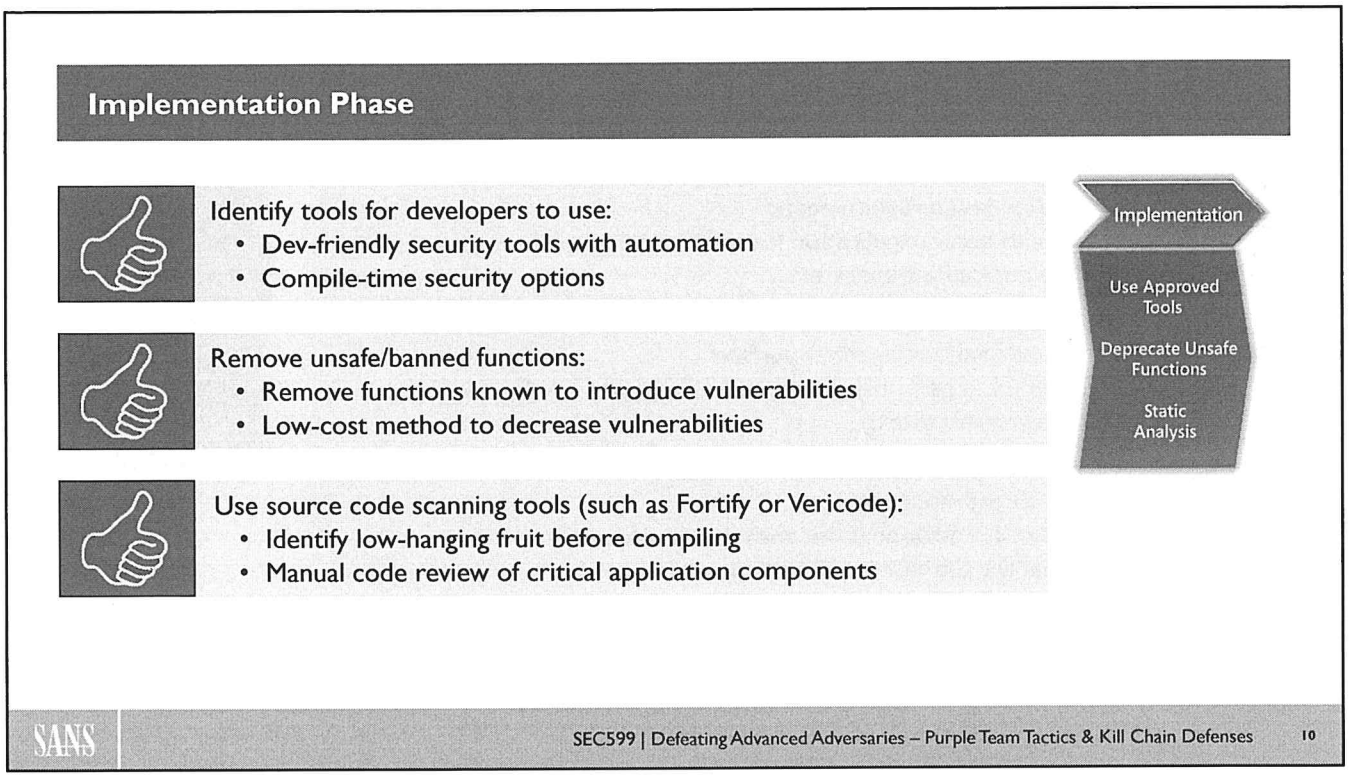
Design Phase

During the design phase, three main areas are covered. First, design requirements must be established. Leveraging the requirements phase, software features and functionality must be written to adhere to all privacy and security requirements. Because we are looking specifically at privacy requirements, secure communication and storage are major considerations. Cryptographic design must be well thought out for secure implementation and the types of cryptographic attacks must be well understood. This is a good spot to add in some peer review.

Next, we want to thoroughly understand and analyze the attack surface. We must look at the design from a high level all the way down to a low-level and identify all of the potential areas where vulnerabilities may exist and map threats to those vulnerabilities. We perform this task during the design phase so that we can make changes prior to any development as a cost-saving measure, and hopefully a time-saving measure. We will get into more on attack surfaces shortly.

Finally, we flow from analyzing the attack surface into threat modeling. This gets specifically into mapping the actual attacks to the attack surface. It is OK to get creative with the types of attacks during this phase as the team should be encouraged to think outside of the box. Threat modeling is also covered shortly.

© 2019 Erik Van Buggenhout & Stephen Sims



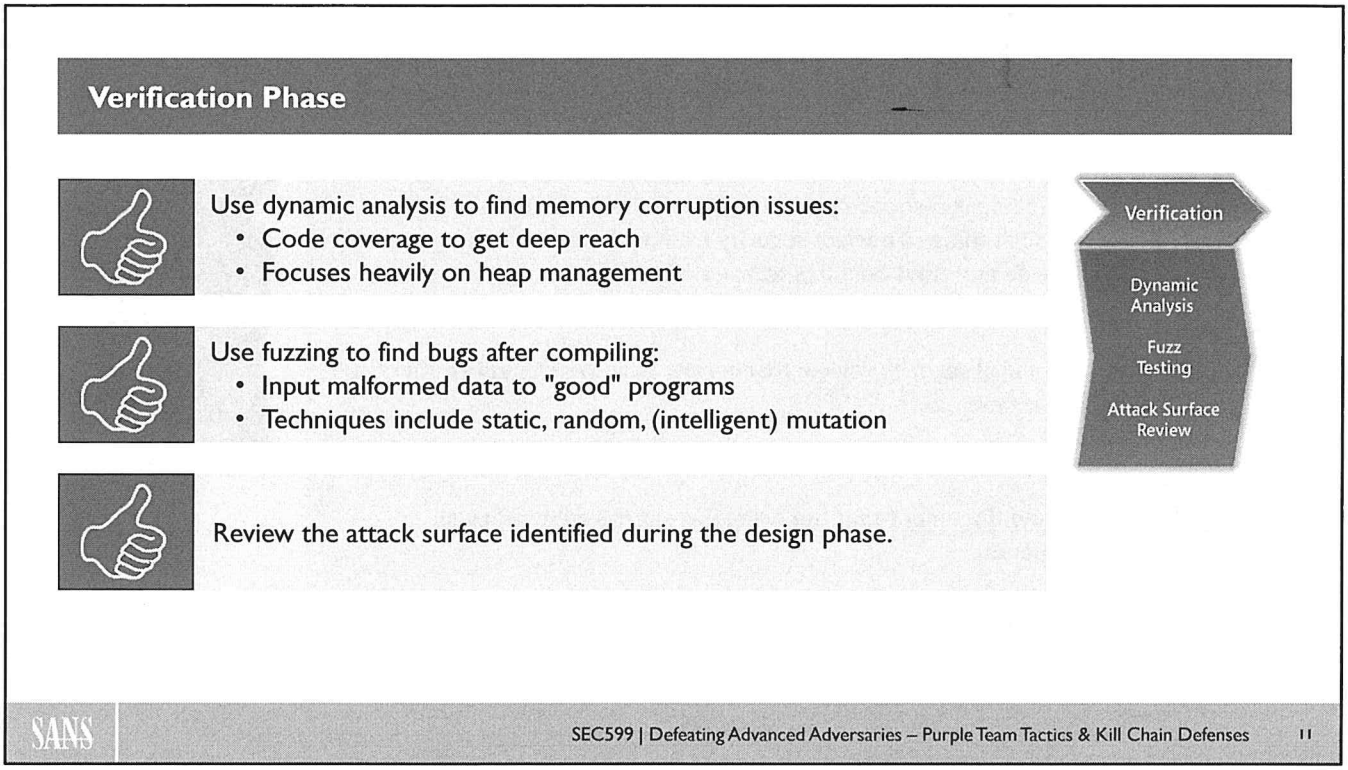
Implementation Phase

The implementation phase also has three main areas. The first area is centered around the use of approved tools. Security researchers and engineers, along with developers, should work together on solutions to help automate as much of the SDL as possible, without sacrificing security. Not every developer can be expected to be security experts and to get better support for the SDL; automation is highly desirable. Penetration testers, security engineers, incident handlers, and developers are all good resources to help identify the types of tools and exploit mitigation protections that should be used by the compiler. Depending on the target operating system where the software will be installed, compiler options such as support for address space layout randomization (ASLR), SafeSEH, stack and heap canaries, data execution prevention (DEP), and others should be designated as requirements. These types of security options and exploit mitigation controls are constantly changing and should be followed closely.

The next area is to identify any unsafe functions and add them to a list of banned functions that can be easily referenced and enforced. If possible, automating the discovery of banned functions during code review should be implemented. It is also common for operating system developers to remove unsafe functions that could cause issues if not identified during the development process. Microsoft removed support for certain functions that allow for the modification of DEP settings on Windows 7. It is important to track these types of changes.

Finally, a code review should be performed prior to compilation. Prior to the creation of automated source code scanning tools, manual review is required. This is a time-consuming process, and the chance of the reviewer missing vulnerabilities is quite high. Not all languages are supported for review; however, the bulk of the primary languages are supported by various tools such as Fortify and Veracode. In this author's experience, many of these tools are good at catching the low-hanging fruit, but they can have a bit more difficulty identifying more complex vulnerabilities. There are often a large number of false positives that must be removed prior to reaching any real areas of concern. When scanning large source code files, there can sometimes be thousands of possible vulnerabilities identified, with the majority not being a real concern. This can be frustrating for developers who are trying to ensure their code is secure and it is often better to have a separate code review team who can help remove some of the burden.

© 2019 Erik Van Buggenhout & Stephen Sims

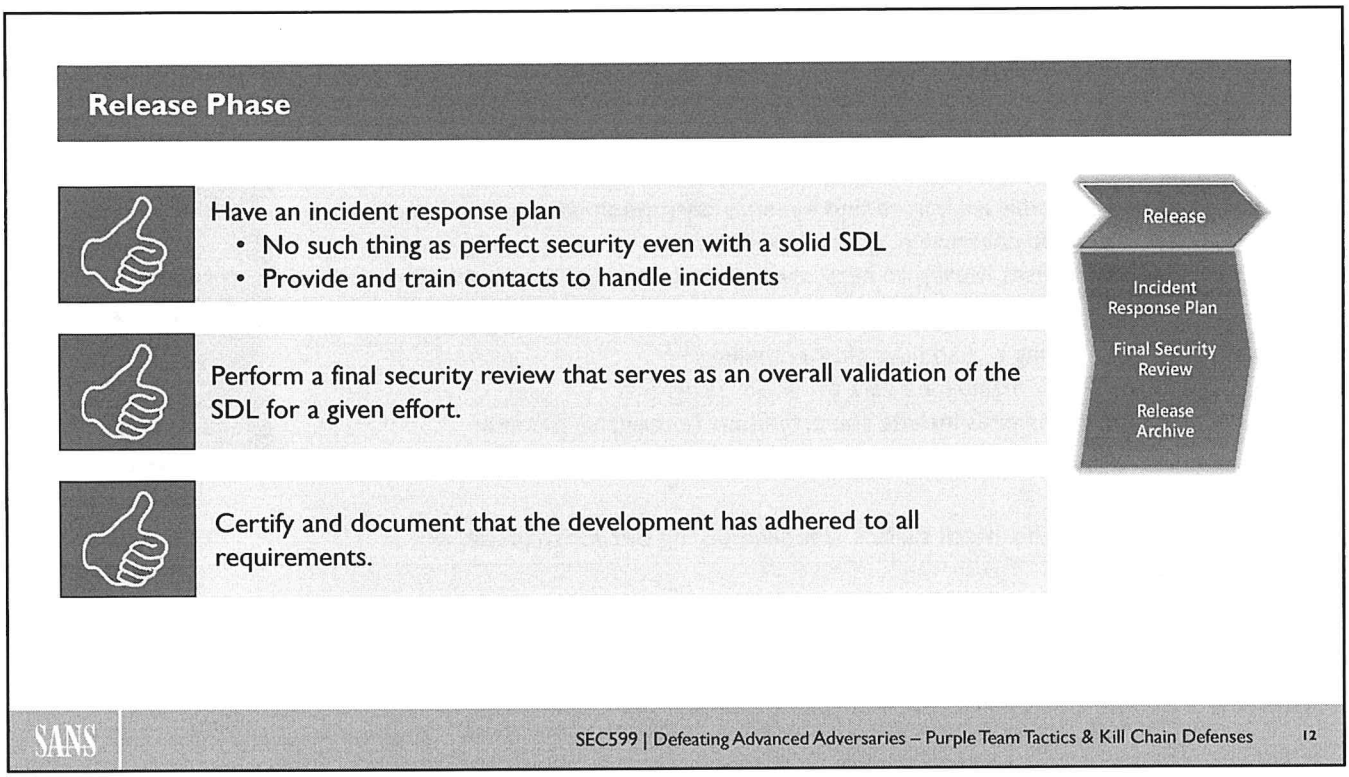


Verification Phase

There are three main areas of focus during the verification phase. This phase occurs after the source code has been compiled into an object file. At this point, regardless of how well you think you know your code, it has changed. The type and version of the compiler, the compiler and linker options used, exploit mitigation controls used, and other options used can heavily influence how your code will look on the other side. It is with this understanding that we must find ways to test all areas of input to the application and get good code coverage. Dynamic analysis tools and fuzzing tools have a similar role and the terms are often used synonymously. Dynamic analysis focuses more on identifying runtime errors, dynamic memory corruption, user privileges and rights, and some other specific areas. This often includes using dynamic tools to input data into the application and monitor behavior.

Fuzz testing or fuzzing is a useful software testing technique that involves sending malformed data to protocol implementations based on RFCs and documented standards. Programs are built to function, preferably based on standards that allow for interoperability with other vendors' products. As we know, programs can be built in many different languages using a combination of many different functions. If you have 100 developers write the same application, each one will likely be different at the source level. Fuzzing requires you to think of all of the ways that a developer could have written a piece of software and test for relative vulnerabilities. It is not that simple of a process, though, as many vulnerabilities are complex and difficult to predict. Take the vulnerability class called "use-after-free." This typically involves dynamically allocated objects that are freed and later referenced by code. An active pointer that is pointing to a freed object is a potential recipe for disaster. These types of vulnerabilities can be difficult to spot, especially during incremental code changes. Fuzzing can greatly increase the chances of finding such bugs. There are various types of fuzzing techniques covered in other courses, including static, randomized, mutation, and intelligent mutation fuzzing with tools such as the Sulley fuzzing framework by Pedram Amini and Aaron Portnoy.

At this point, we also want to review the attack surface that we analyzed during the design phase to ensure that nothing was missed. It is fairly common for changes to occur during the actual development of the software. This allows for an opportunity to ensure all threats and vulnerabilities are captured.

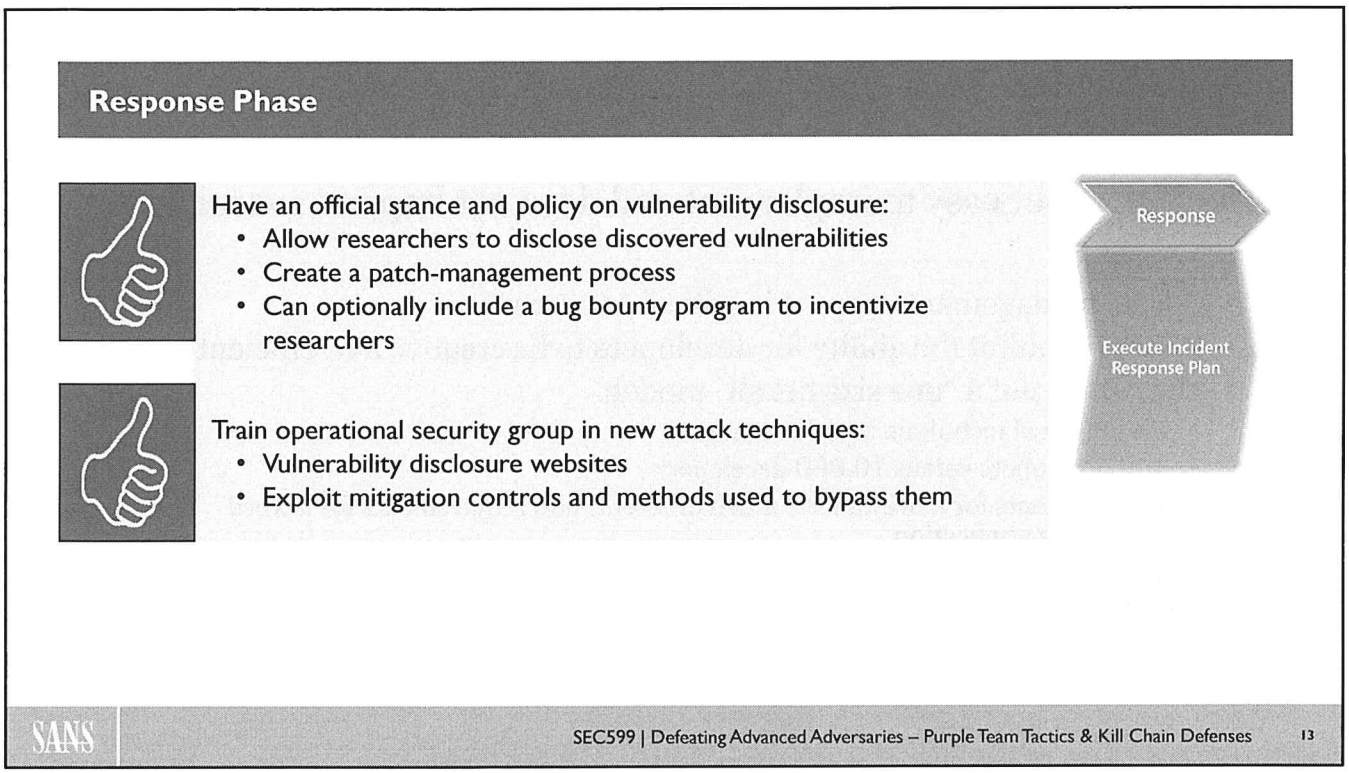


Release Phase

There are also three main phases during the release phase. The first is to ensure there is an incident response plan relative to the developed software. No matter how hard we try, there will never be such a thing as perfect security. If a bug is discovered, especially a critical bug, who are the main points of contact to quickly initiate a response? The answer to this question is exactly what this step is about. As per Microsoft, this step is also used to set up points of contact for inherited code. If code used was developed outside the group and questions arise, contacts should be available to answer questions, especially in lieu of training and documentation.

The second and third security review steps are in place to designate a time to take a holistic view of the SDL process to date. It works directly with the release archive step. The goal is to ensure that all phases and steps have been covered and documented. This serves as a crucial role in audits and adherence to regulatory requirements. It is this phase in which a good checklist and tracking system come in handy. The attack surface should be validated one final time, as well as threat modeling, risk tolerance as documented during the requirements phase, risk assessment, and all other steps.

© 2019 Erik Van Buggenhout & Stephen Sims

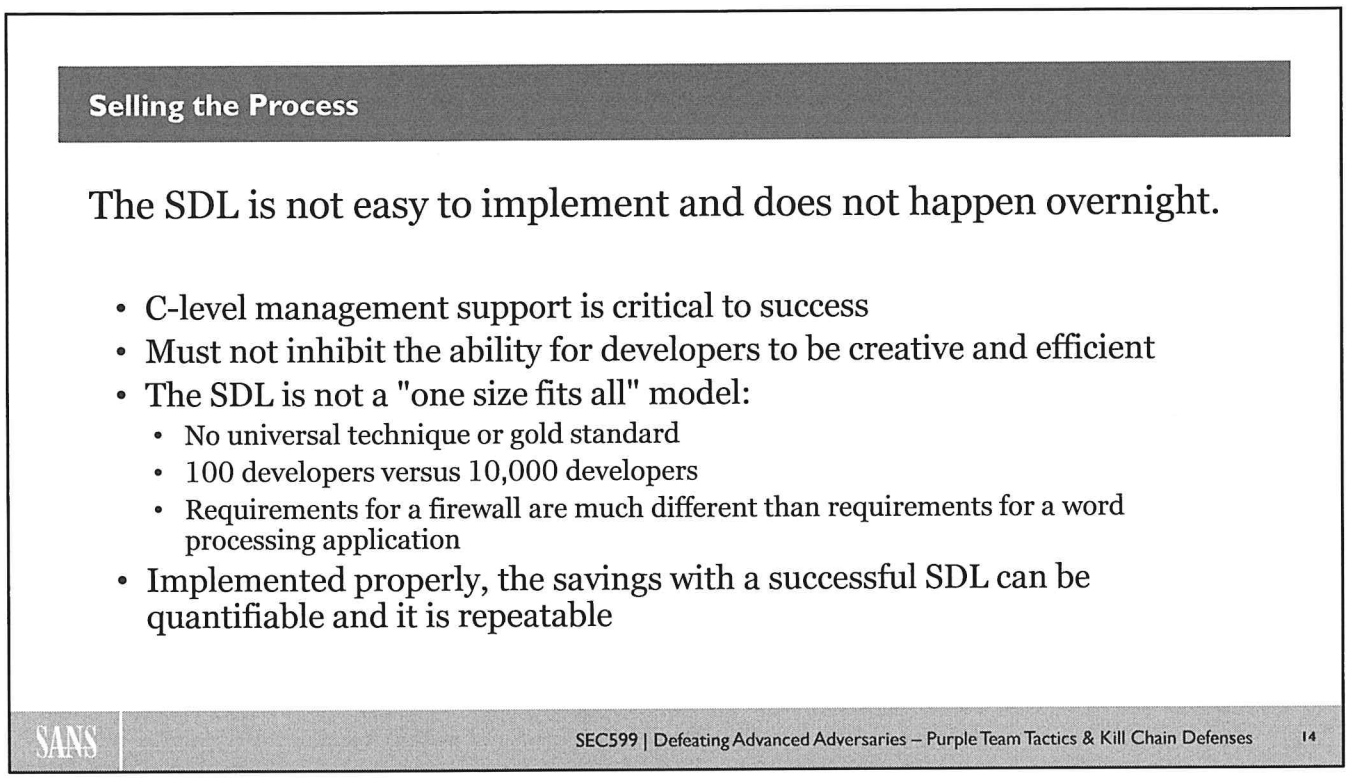


Response Phase

The final phase is centered around the implementation of an incident response process. As stated before, there is no such thing as perfect security. No matter how mature and effective your SDL process, there will always be bugs discovered and other security issues to handle. Every organization should have a vulnerability disclosure process. There are various philosophies on how disclosure should be handled, such as full disclosure, responsible disclosure, and limited disclosure (which falls somewhere in between the other two). There should be a clear-cut process for researchers and others who find a potential bug or vulnerability in your products; even if that process says that anyone reporting bugs may face legal action. This is likely not the preferred approach, but it informs those wanting to disclose a concern about your organization's stance on disclosure.

Once someone submits a finding, this is when your incident response plan goes into action. Who will respond to the individual or organization disclosing the finding? Who will take action and reach out to developers or others who should be involved? How will the submission be tracked and how long will it take to officially respond or patch the finding? How will the patch be distributed to customers if applicable? These are all processes that should be well documented and actionable.

© 2019 Erik Van Buggenhout & Stephen Sims



Selling the Process

A common question is, "How can I sell this whole SDL thing to management and get support?" This is likely as hard of a task as the actual implementation of the process. In this day and age, we are all inundated with processes and the introduction of more processes can face resistance from many angles. Always remember that the ability to factor in monetary savings into the equation will almost always get some level of attention. A properly implemented SDL should do exactly that—save money. As with any other proposal, pitching the introduction of the SDL to your development process should be well thought-out and well-presented. Interviewing various lines of business for their perspective is highly beneficial. If the security operations group is burdened with incidents stemming from poor code, you want to know that information. If management is dealing with new regulations and an audit, this can also be useful information. How can you make the company's job easier and cut costs? This should be a key element when going in to pitch the process for approval.

Executive-level support for the process is critical to its success. Lacking this support will most likely result in a poorly implemented SDL or even complete failure and resistance. This should be vocalized during the proposal. One key concern that this author has learned from developers is that the SDL must not inhibit the developers from being creative and innovative. It must also not burden them down with too much process. Development can be a stressful profession with stringent requirements and sensitivity to time. Education and the ability to automate as much of the process as possible will garner more support from developers and program managers.

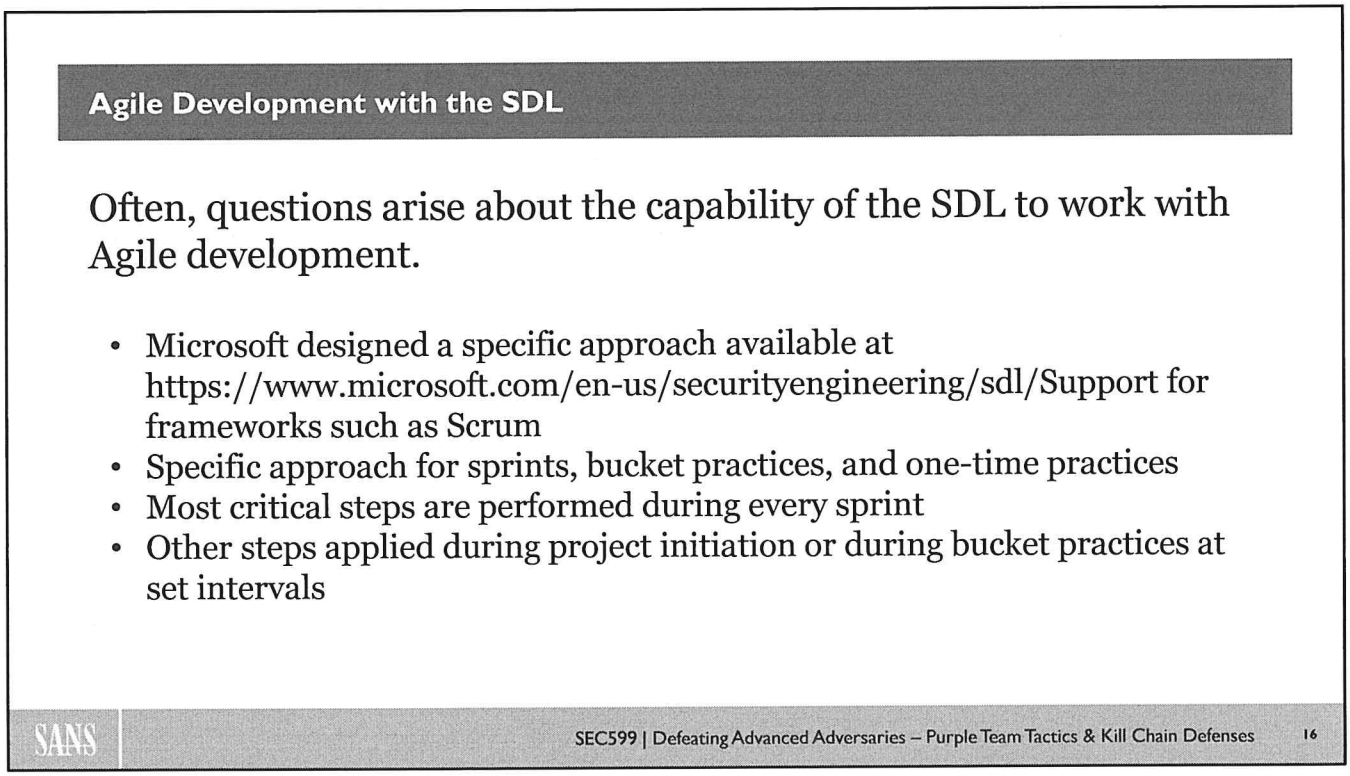
It must also be remembered that the SDL is not a "one-size-fits-all" model. Each organization can adopt the overall framework but must customize it to their needs. It is also not a process that can be implemented overnight, or even in a month. It takes experience and ongoing customization. A company that has 100 developers will need a different SDL application than a company with 10,000+ developers. Also, it cannot be a blanket application to all instances. A division working on the design of new firewall technology may need a different SDL than that of a word processing application.

© 2019 Erik Van Buggenhout & Stephen Sims

This is not to say that the framework is not applicable; it is simply saying that the application of the various steps during each phase may have to be customized to meet the needs of the organization and the security requirements.

Again, the biggest selling point is that a properly implemented SDL should result in quantifiable savings. It should make for an efficient development process, and there should be a noticeable change and decrease in code fixes. The term return on security investment (ROSI) is often a helpful approach. The general idea is that by spending time and money doing something to reduce or avoid a potential or existing risk, it will prevent a future loss that would likely be greater than the cost of mitigating the risk.

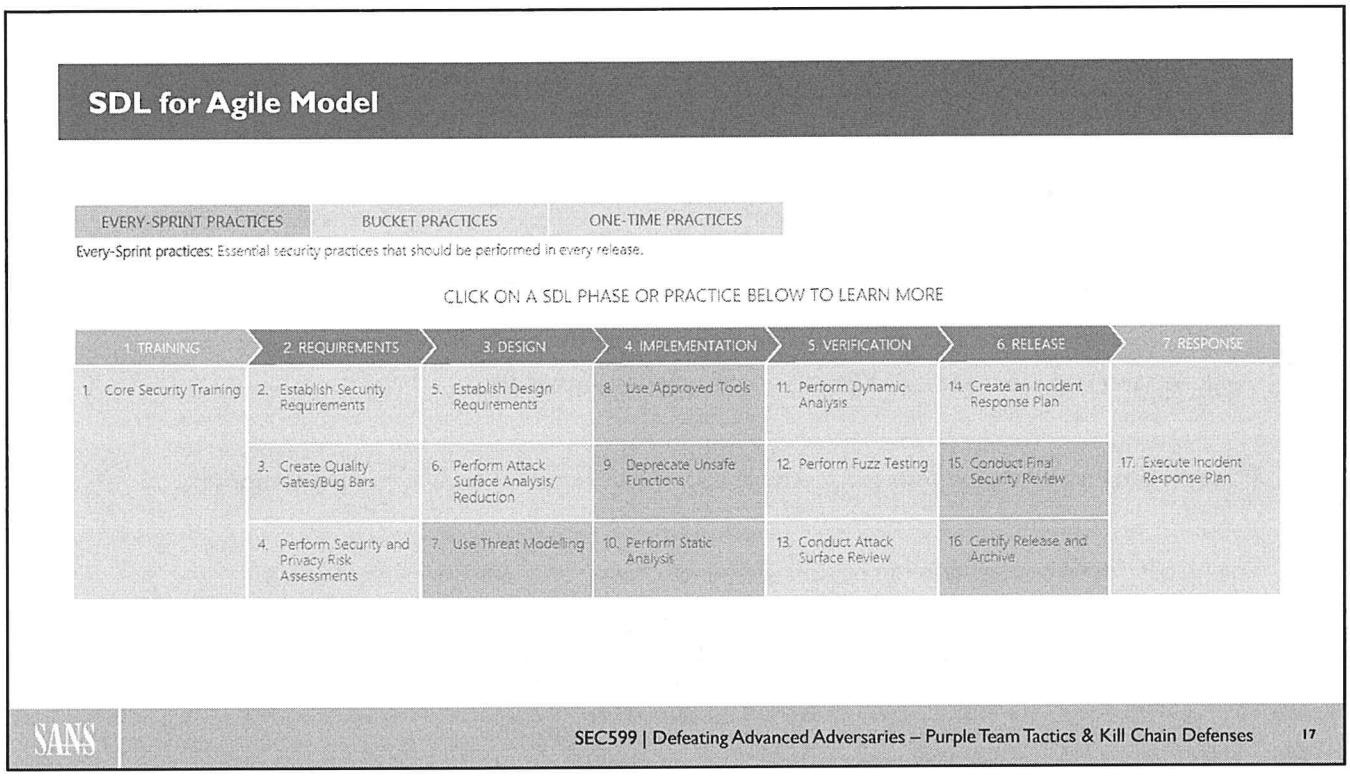
© 2019 Erik Van Buggenhout & Stephen Sims



Agile Development with the SDL

Agile development is a development process that is highly utilized and often difficult to implement. It is often seen as an inhibitor to creativity by many developers who have not successfully implemented the process and changed from models such as the waterfall model. Microsoft set up a specific application of the SDL to agile development methods that can be viewed at https://www.microsoft.com/en-us/securityengineering/sdl/. It maps specific portions and steps of the standard SDL previously covered to different development phases using the agile approach. Every agile sprint receives the most critical steps of the SDL based on the biggest areas of concern. The most important tools are run, threat modeling is performed, and code review is performed, as well as various other security reviews. A sprint is typically several weeks long and a fast-paced subset of development for the overall product. Applying all phases of the SDL to every sprint is not actionable. The other areas of the SDL that are not applied during every sprint can be applied at project initiation, such as those relative to the requirements and design or during bucket practices that occur at set intervals.

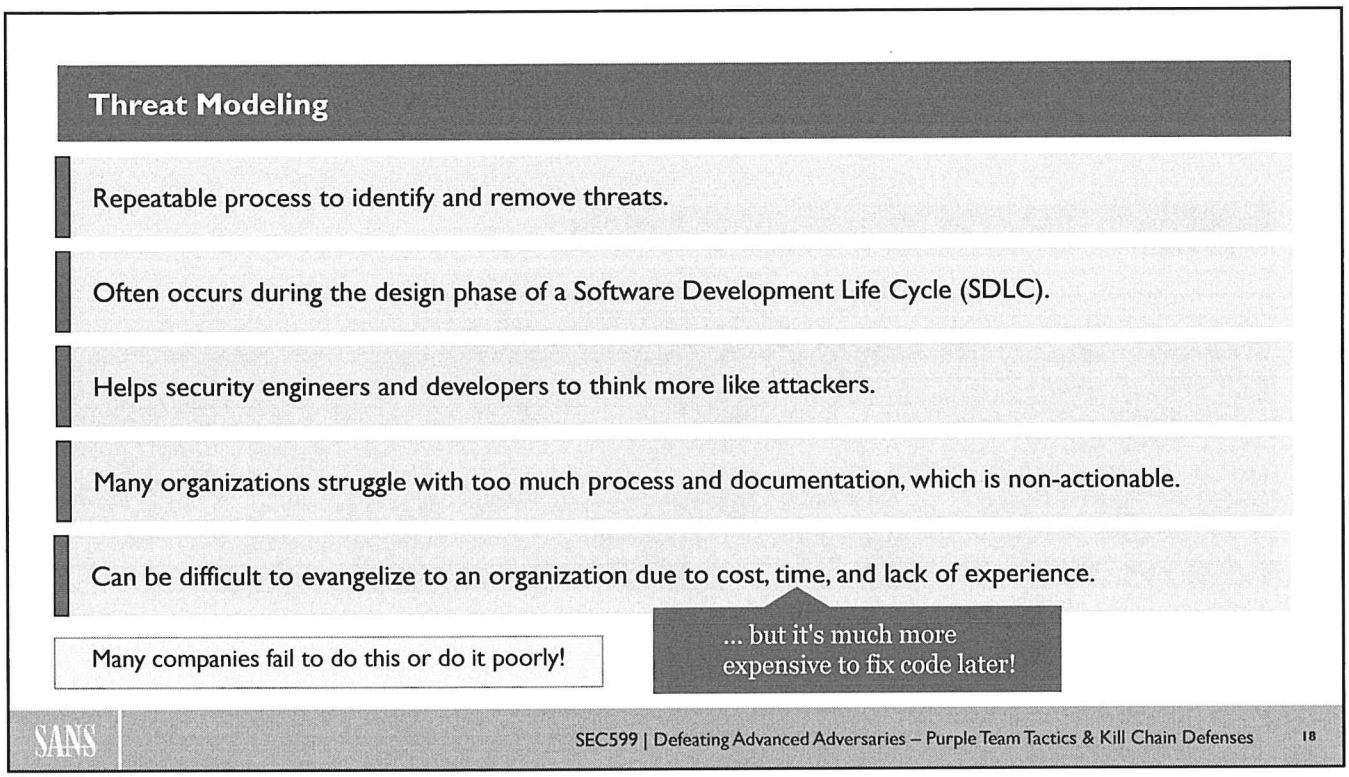
© 2019 Erik Van Buggenhout & Stephen Sims



SDL for Agile Model

As seen on Microsoft's website at https://www.microsoft.com/en-us/securityengineering/sdl/, this diagram is interactive and can help show which SDL phases apply to One-Time Practices, Bucket Practices, and Sprints.

© 2019 Erik Van Buggenhout & Stephen Sims



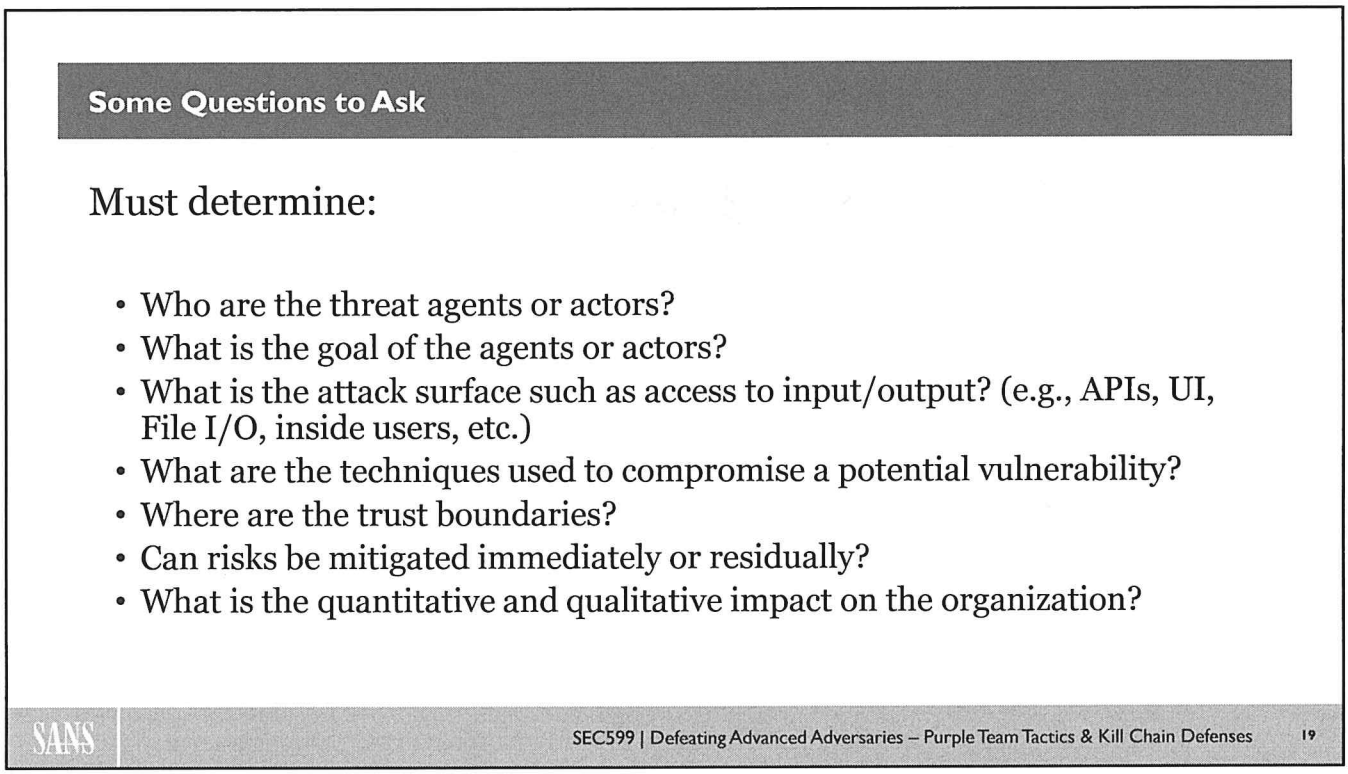
Threat Modeling

Threat modeling is an extremely valuable resource if implemented properly. Think about the cost associated with reviewing and fixing production code, or even code that has not been published yet, when a significant finding is found. Oftentimes, a vulnerability may be left in the code due to the results of a risk assessment showing that the cost would be greater to fix the bug compared to the impact to the organization if it was discovered and exploited. Regardless of that assessment and justification, it would clearly be more desirable if that bug had never been introduced in the first place. This is where threat modeling can help.

Threat modeling is easy to talk about and hard to implement into an actionable process. It used to be that few developers and security professionals knew exactly what threat modeling was and how it was to be implemented. With the help of various organizations such as Microsoft, Cigital, and OWASP, threat modeling has been made more actionable and dynamic. Similar to that of Microsoft's SDL, it is not a process that can just be implemented with perfect results. It takes time and effort, with much training and practice. Threat modeling is commonly performed as part of the design phase in the development process. Once the low-level diagrams are available—showing all of the data flows and processes—it is much easier to look at the attack surface and point out potential vulnerabilities. The goal is to make an actionable, repeatable process in the design phase of the Software Development Life Cycle (SDLC) to prevent vulnerabilities from being introduced into the code or overall architecture.

Many organizations get too focused and overwhelmed with documentation and process. This becomes non-actionable and slows down the development process. It is better to simplify the threat modeling process and focus on the biggest areas of concern, rather than try to accomplish too much at once and lose support for the initiative. It must also not impede the developer's ability to be creative, especially in product-based companies. This goes for the overall SDL process as well. Similarly to selling the SDL process, it can be difficult for some organizations to gain support for threat modeling. Demonstrating the process, evangelizing it, showing other companies who are using the process, and starting small can help. You must remember to map technical risks into business terms to ensure the request has teeth.

© 2019 Erik Van Buggenhout & Stephen Sims

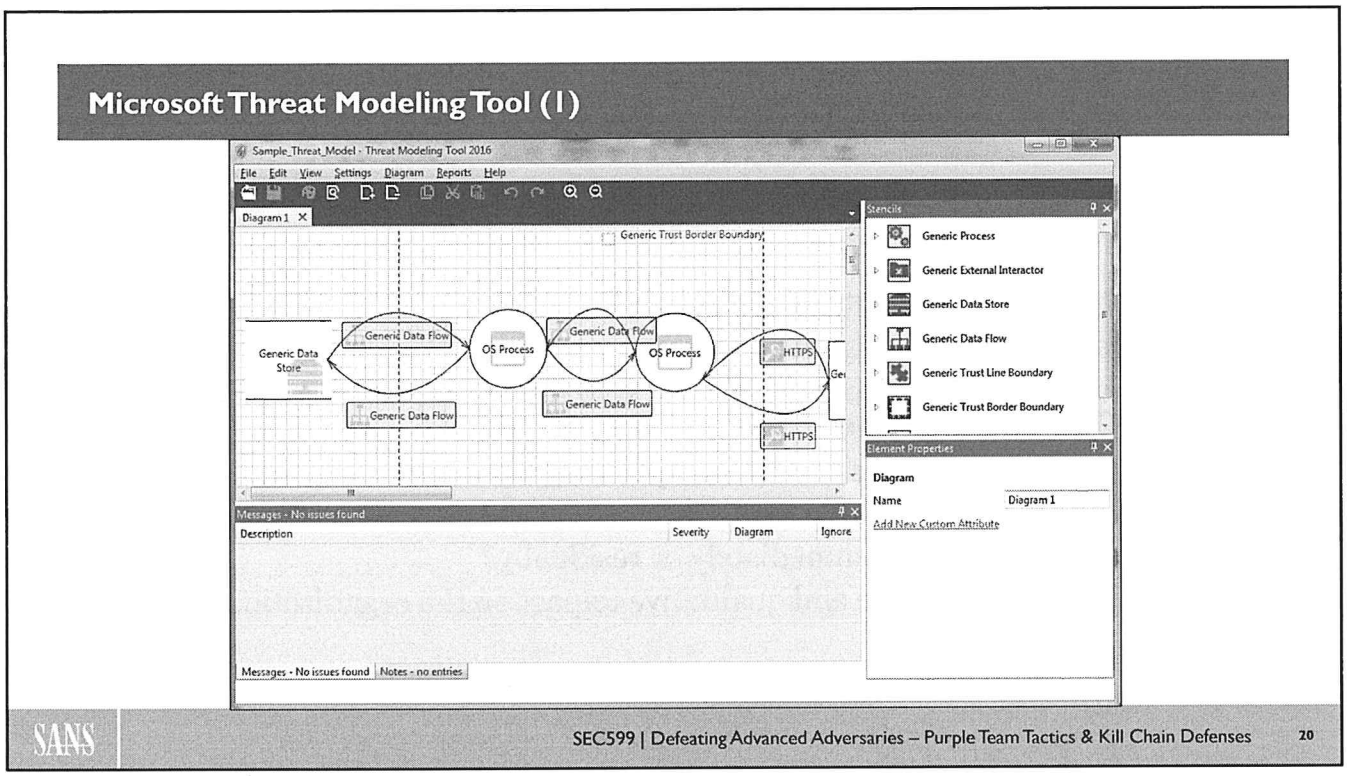


Some Questions to Ask

Once you have the design to which you want to apply threat modeling and you ensure it is sufficiently low level, there are many questions to start asking. There are various publicly available threat models such as STRIDE from Microsoft, as well as additional risk assessment models such as Microsoft's DREAD, the Department of Homeland Security's (DHS) Common Vulnerability Scoring System (CVSS), Carnegie Mellon's OCTAVE, TRIKE by Brenda Larcom and Eleanor Saitta, and many others.

We want to know about the threat agents or actors. These could be inside users with privileged access, malicious users from home on their computers or over phones, malicious software, jailbroken smartphones, and countless other threats. We want to understand their potential goals, such as harvesting credit card numbers, denial of service, and intellectual property theft. What is their attack surface? Perhaps they are able to communicate with our frontend web servers with no authentication and then have additional opportunities with authentication. Is authentication assumed once initially authenticated? What else is exposed? DNS servers, mail servers, etc. Do we have store branches? Is social engineering a possibility? How about more complex attacks like communications occurring from inside a trust boundary? Get creative. What techniques are used to exploit the attack surface and potential vulnerabilities identified? According to OWASP, the attack surfaces include all data that flows in and out of an application, the code that protects these flows, the data elements involved, and the code that protects those elements. Check out the following cheat sheet for some tips from OWASP:
https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet

Can these risks be mitigated through existing controls? Is it possible to fix them with code? Sometimes, the vulnerabilities identified during threat modeling prove challenging to fix and the fixes do not always come from code changes. You must always assume that communications coming from outside of a trust boundary could be malicious. What happens if someone breaks out of the security controls enforced by an embedded device? They can now potentially reverse engineer a mobile application, proxy the communication, and circumvent security restrictions. As we do with any type of risk assessment, we must determine the quantitative and qualitative impact to the organization. How bad could it be? How much would it cost? What is the likelihood?



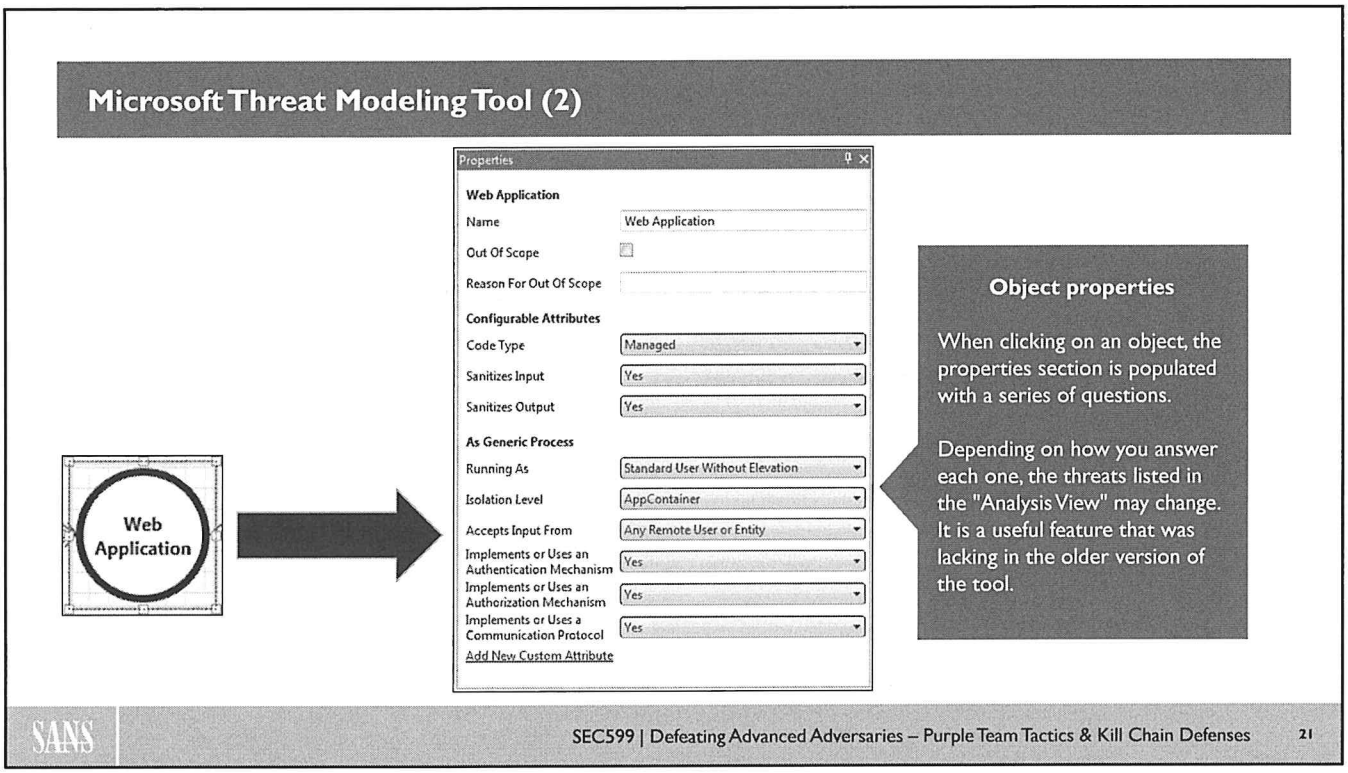
Microsoft Threat Modeling Tool (1)
Microsoft released a free tool simply called the Threat Modeling Tool. You can download the Microsoft Threat Modeling Tool version 2016 at https://www.microsoft.com/en-us/download/details.aspx?id=49168 General information about the tool can be found here: https://www.microsoft.com/en-us/securityengineering/sdl/resources. Microsoft also released a great card game called the "Elevation of Privilege Card Game" to practice threat modeling against your designs. It is available at https://www.microsoft.com/en-us/securityengineering/sdl/resources.

With the Threat Modeling Tool, you can draw your designs and have an automated tool get you started on asking the right questions. It used to require MS Visio; however, with the new version released in March 2016, Visio is no longer required. The initial screen, as shown on the slide, is the "Design View." This is where you actually draw out your designs to the whiteboard and make all of the relevant connections and data flows. One nice thing is that the "Messages" area on the bottom, which will let you know if you have likely missed a data flow. The example drawn on the slide is that of a simple network communication. The red hyphenated lines indicate a trust boundary where increased attention should be placed. From within a trust boundary, data and flows may be implicitly trusted, as where communications coming into or leaving a trust boundary should be more aggressively scrutinized.

There are other free tools available on threat modeling, such as Seasponge from the Mozilla Winter of Security 2014:

https://air.mozilla.org/mozilla-winter-of-security-seasponge-a-tool-for-easy-threat-modeling/
https://github.com/mozilla/seasponge

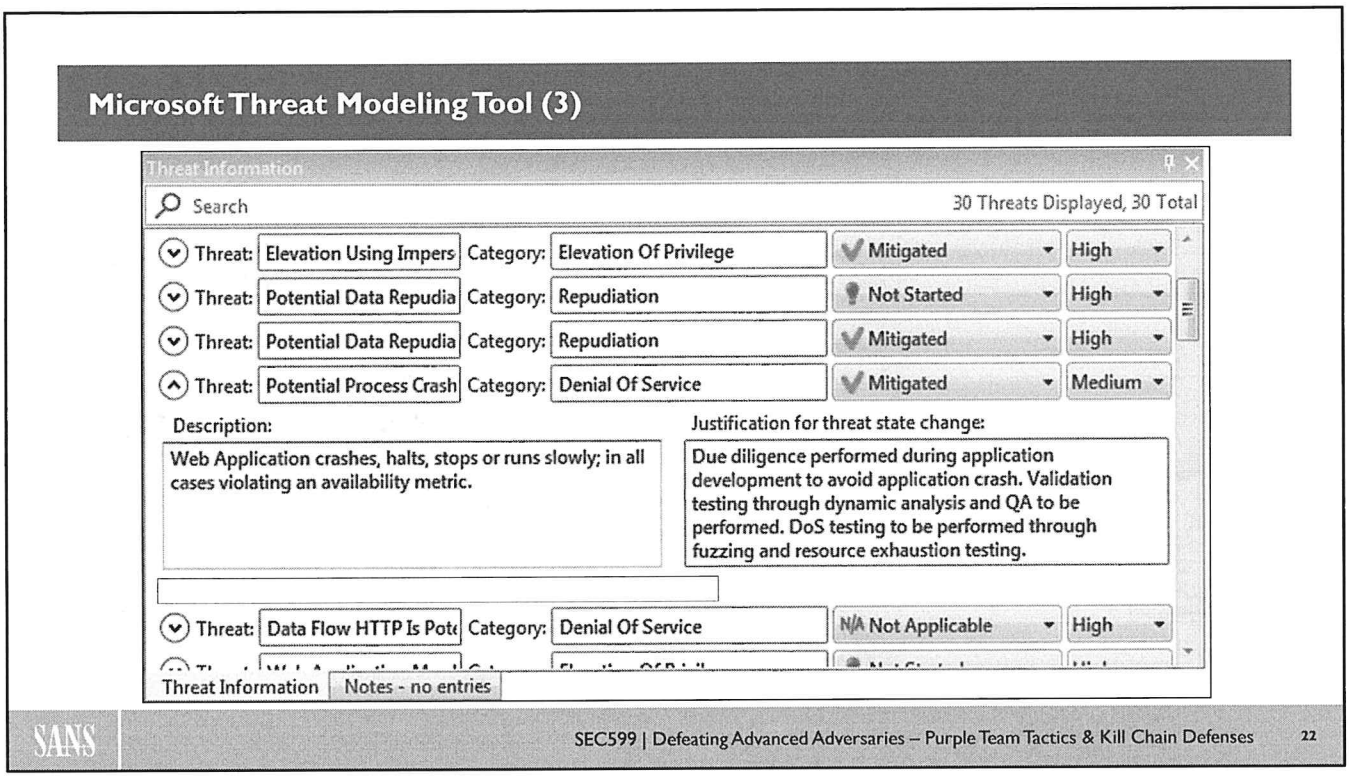
© 2019 Erik Van Buggenhout & Stephen Sims



Microsoft Threat Modeling Tool (2)

On this slide is a screen capture of the "Properties" section of the Threat Modeling Tool. When you click on certain types of objects, this region will be populated with a series of questions. Depending on how you answer each one, the threats listed in the "Analysis View" may change. It is a useful feature that was lacking in the older version of the tool.

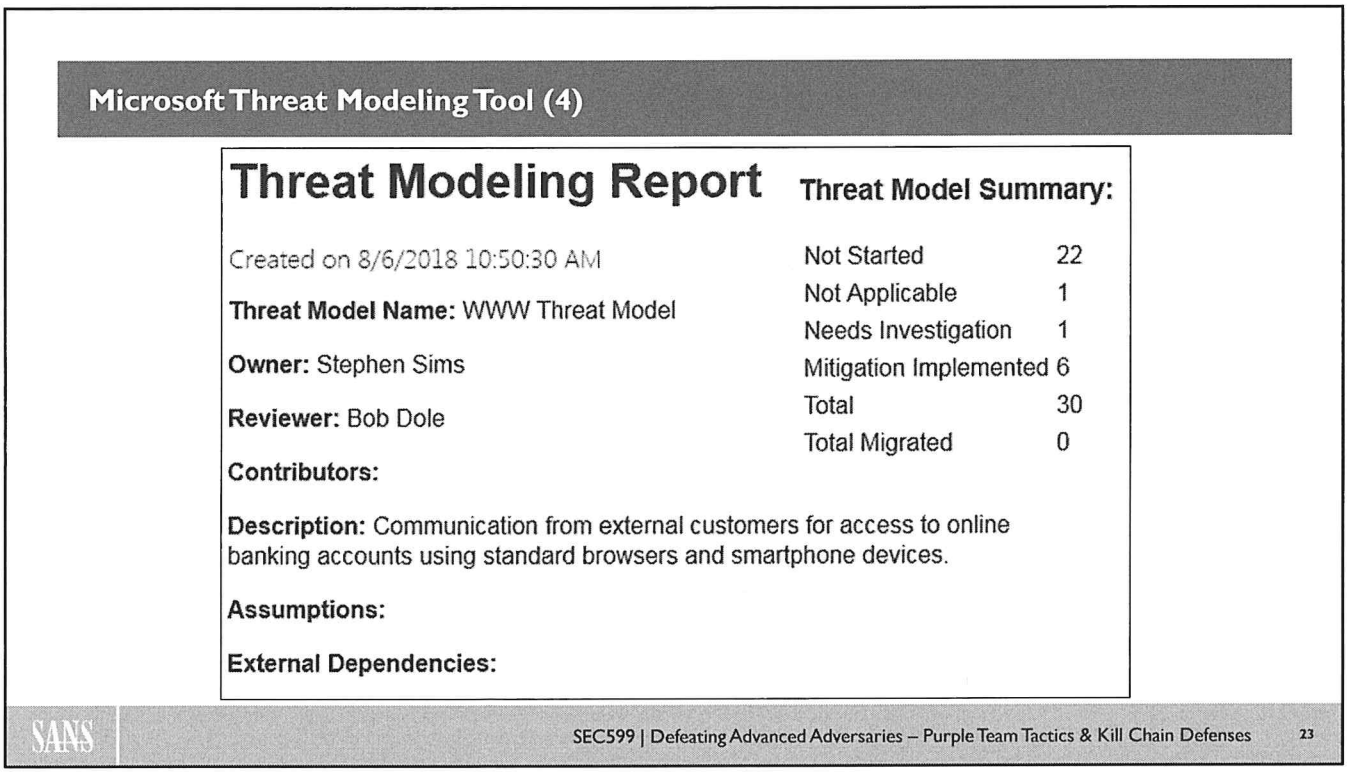
© 2019 Erik Van Buggenhout & Stephen Sims



Microsoft Threat Modeling Tool (3)
This slide shows the "Analysis View" screen. Once you have drafted your design into the design window, click on "View, Analysis View" from the ribbon bar to see what threats have been identified by the Threat Modeling Tool. It is designed to get you thinking about potential threats and add some automation for developers who may not be security experts. That being said, the tool does a great job at asking the initial questions that should be asked or making simple comments such as, "Web Application crashes, halts, stops or runs slowly; in all cases violating an availability metric." This is an example of a topic that may not be brought up without the help of the tool. Not all of them will apply to each flow and they can be removed if appropriate.

As seen on the slide, there is a Threat and Category. Following that are drop-down boxes showing the status of the risk item and the qualitative rating. On the left of each threat is a drop-down arrow that expands the description of the threat. In the example shown, you can see that the "Justification for threat state change" area on the right is populated with user-supplied content.

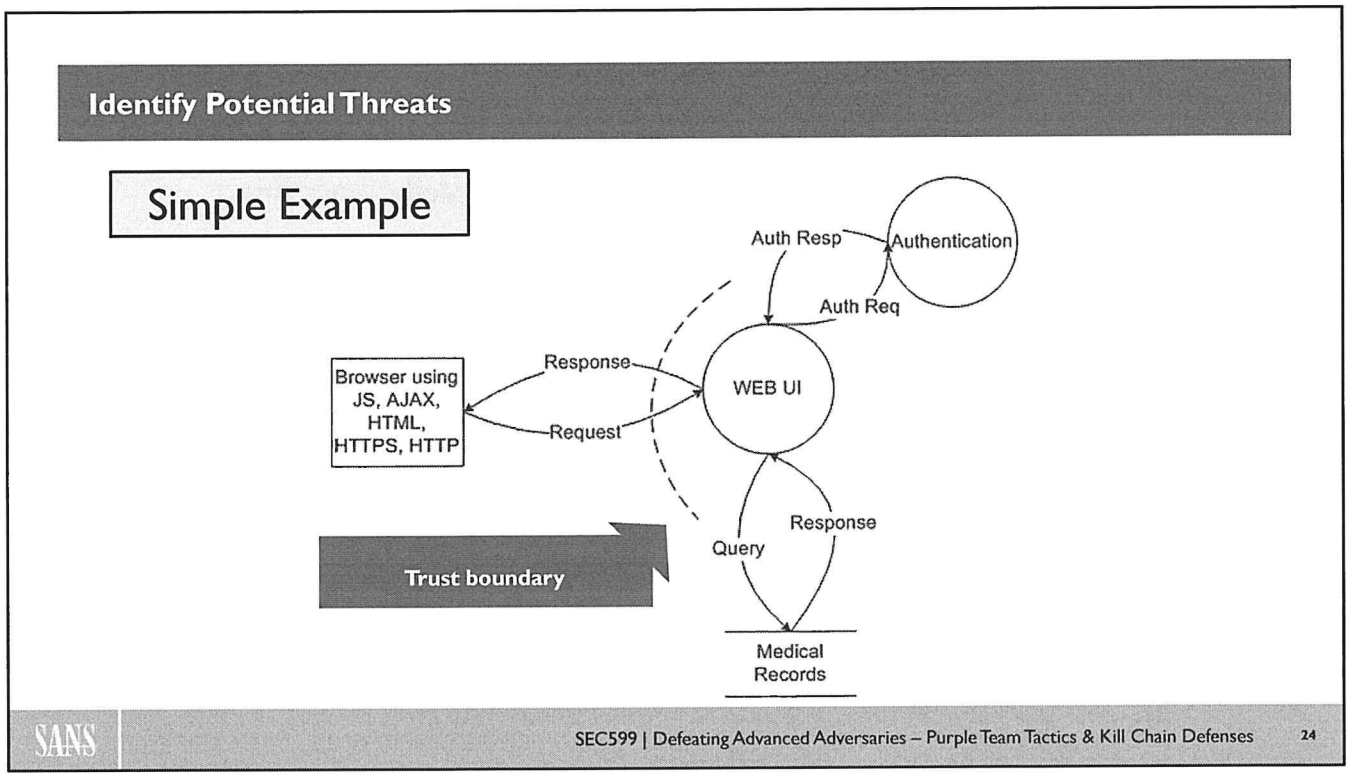
© 2019 Erik Van Buggenhout & Stephen Sims



Microsoft Threat Modeling Tool (4)
When clicking on "Reports" from the ribbon bar, you can select the option to generate a full report of the threat model. On the slide is a snippet of that report. Note: Pieces of the report were moved around to fit on the slide. As you can see, the information about the threat model author, description, assumptions, and dependencies are shown. A summary of the number of threats and the ones still needing to be triaged are also shown. Below this section in the HTML-generated document are all of the threats listed associated with each device, trust, store, or data flow shown.

The Threat Modeling Tool is available for download at https://aka.ms/threatmodelingtool.

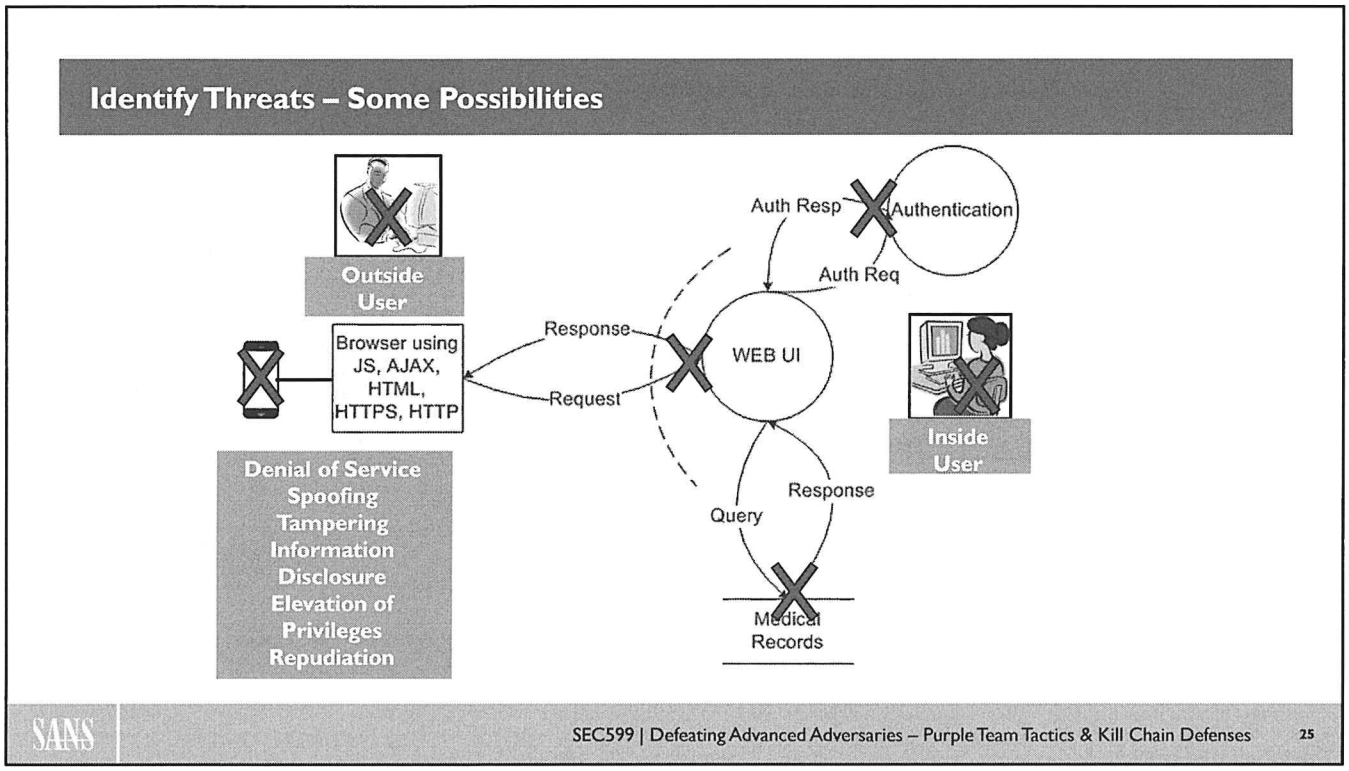
© 2019 Erik Van Buggenhout & Stephen Sims



Identify Potential Threats

On this slide is a very simple network diagram as created with the older version of the Microsoft Threat Modeling Tool. This simplified example is a good place to start when practicing threat modeling. Take a few minutes to identify potential threats. Note the trust boundary marked by the curved, hyphenated line. Trust boundaries require special attention and often influence what components will be fuzzed. On the next slide are some of the potential areas of concern that should be addressed prior to implementing the design.

© 2019 Erik Van Buggenhout & Stephen Sims



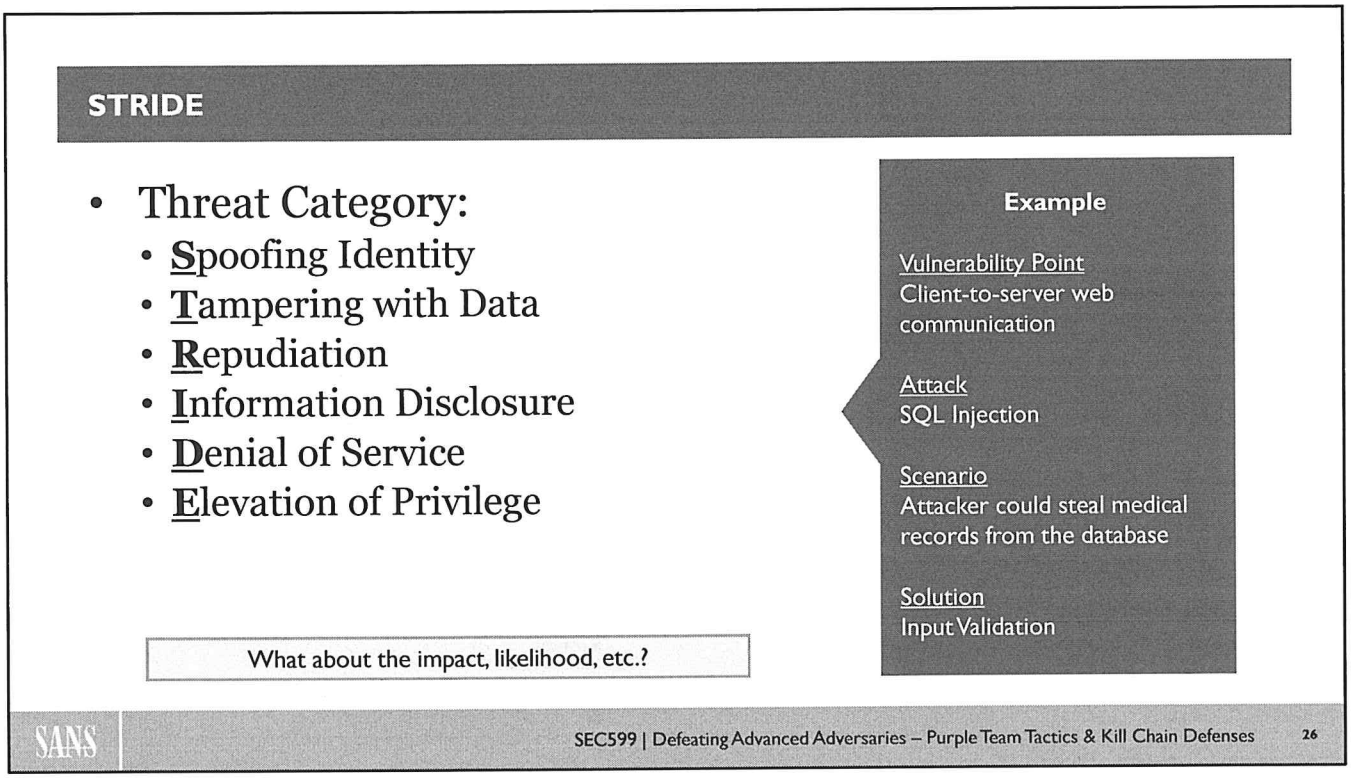
Identify Threats – Some Possibilities

On this slide are some potential threats and vulnerability spots that must be addressed. Some examples of attack categories are Denial of Service (DoS), spoofing, tampering, information disclosure, elevation of privilege, repudiation, and many others. Not all apply to each threat or vulnerability, and we can rule them out as they are addressed. The outside user could be on a personal computer, a smartphone, a kiosk in a store branch, and other possibilities. This is where it is important to think like an attacker. What about the scenario in which your company creates a smartphone application that should be protected by the controls included with an iPhone. Let us say that the attacker jailbreaks the iPhone and is able to circumvent all controls, install his own software, reverse engineer, and learn more about your smartphone application, proxy connection requests, etc. Does this change the typical attack surface? It sure does!

Again, it is easy to start with high-level designs when it comes to threat modeling, but the real power comes in when you get into low-level application designs and data flows. It is at the design stage during the SDLC that you can help prevent bugs or design flaws from being introduced by threat modeling. The more this process can be automated, the more likely it is to be adopted. We cannot expect that all of our developers will become security experts overnight, and if this can be rolled into the development process as seamlessly as possible, our chances of success increase.

Are there any missing trust boundaries that stick out? You may have noticed one should be placed between the Web UI and the "Medical Records" data store, as well as possibly between the Web UI and the Authentication.

© 2019 Erik Van Buggenhout & Stephen Sims



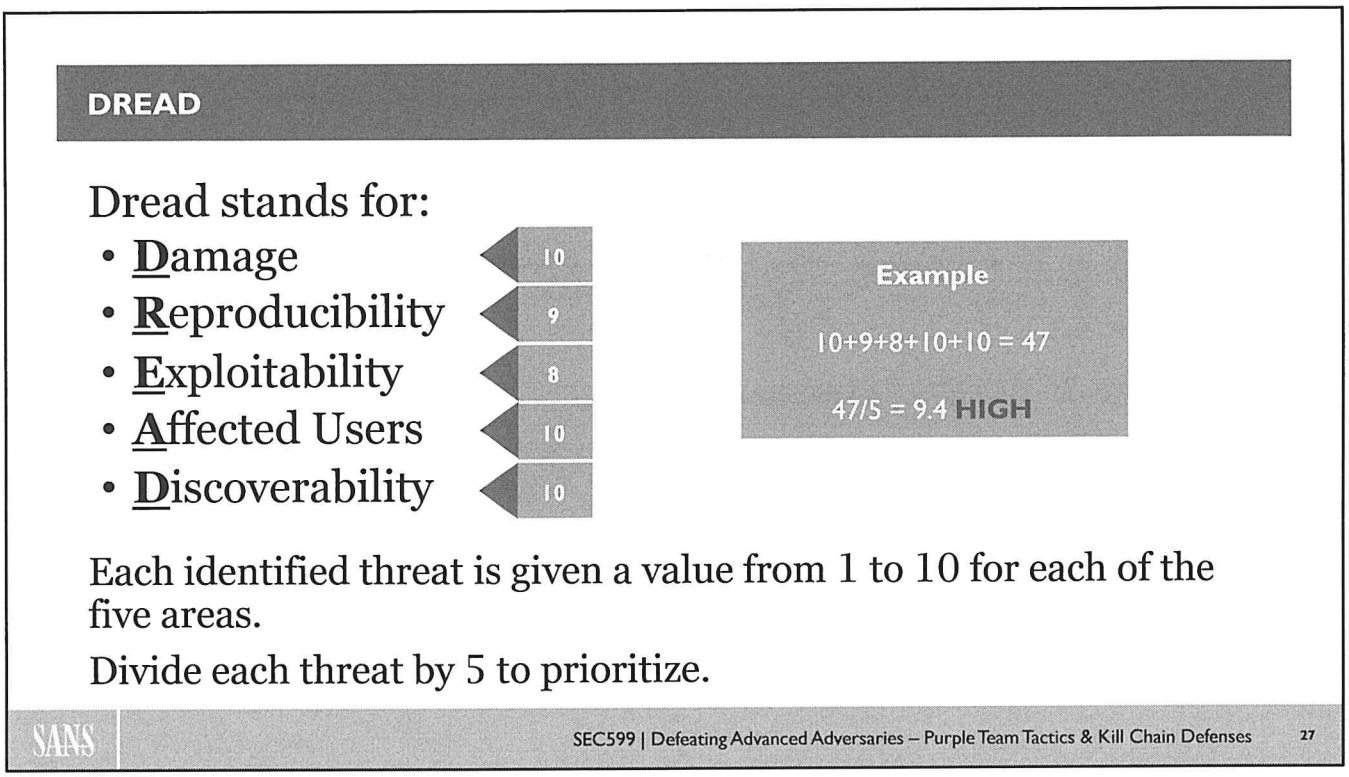
STRIDE

The Microsoft Threat Modeling Tool is based on the STRIDE threat model. As previously mentioned, there are quite a few threat models made publicly available by various organizations. The Microsoft STRIDE threat model is available at https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20).

STRIDE is an acronym that stands for "**S**poofing Identity, **T**ampering with Data, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege." Each of these is a category of threats that should be well known to all of us by this point in our careers. Under each of these threat categories are various attacks, which the Threat Modeling Tool details. The model would have you identify a potential vulnerability point within a design, such as data coming from a user into a web application. Threats to that vulnerability and the attacks associated with them are then identified, such as cross-site scripting (XSS), parameter tampering, SQL injection, etc. Under each of the potential attack types, you would then document some scenarios that could occur. Finally, some mitigations for each vulnerability can be identified.

What about the impact of an event, the likelihood, and other risk assessment modeling?

© 2019 Erik Van Buggenhout & Stephen Sims

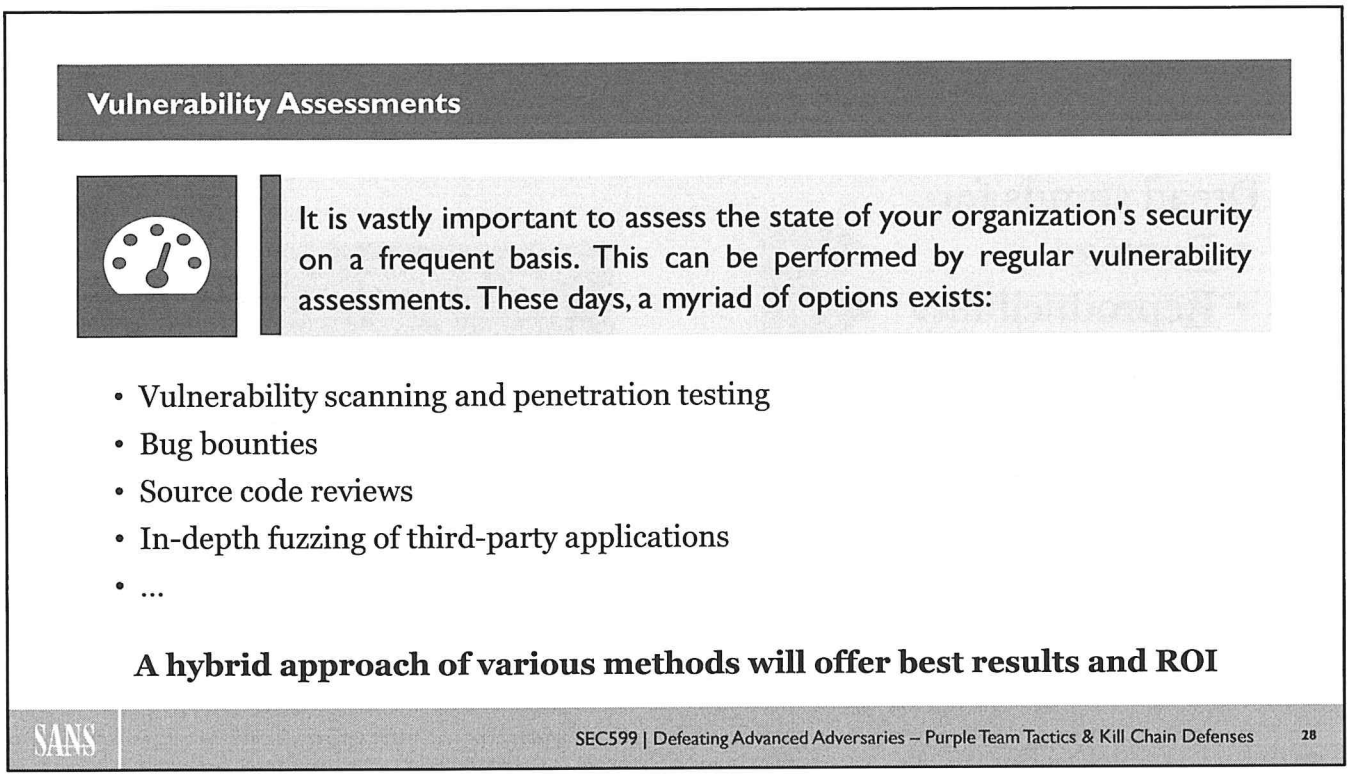


DREAD

DREAD is a multidimensional risk calculation model for prioritizing threats. Microsoft documentation on DREAD can be found at https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10).

The five areas applied to each threat include **D**amage, **R**eproducibility, **E**xploitability, **A**ffected Users, and **D**iscoverability. Damage can be compared to the impact a successful attack would have on the organization. A compromised database containing a million patient records in the worst-case scenario would be a grave impact and as such, we assign it a 10. The reproducibility pertains to the likelihood that the attack is successful and reproducible. Once an SQL injection attack is identified, it is typically easy to reproduce with success. We gave this one a 9. The exploitability pertains to the difficulty in pulling off the attack successfully. Is it a well-known attack with lots of tools and help, or is it obscure and difficult? We gave this one an 8. Affected users pertain again to the impact. In our scenario, a million patients are affected and as such we give this one a 10. Finally, we have discoverability that pertains to the likelihood that someone will find out about the vulnerability. In our example, we've assigned this one a 10, as SQL injection vulnerabilities are often easy to spot. Each organization would apply its own ratings to this threat. When we add up each of the five areas, we get 47. We divide this number by 5, representing the 5 areas in DREAD, and we come to our overall rating of 9.4, which can be considered high. We would likely want to address this threat with priority.

© 2019 Erik Van Buggenhout & Stephen Sims



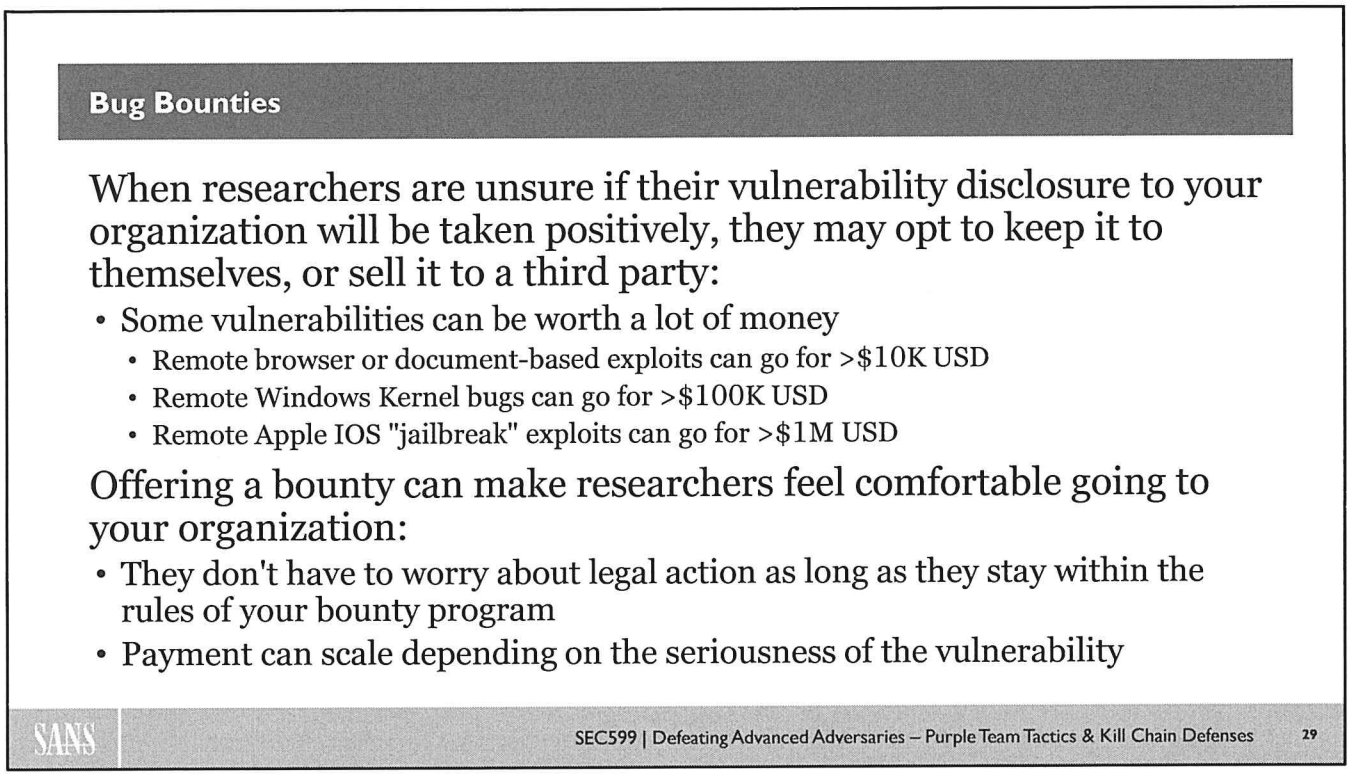
Vulnerability Assessments
It is vastly important to assess the state of your organization's security on a frequent basis. This can be performed by regular vulnerability assessments. Due to the state of the current cyber security landscape, vulnerability assessments have become the norm and the vast majority of organizations are combining different techniques to understand their cyber risk exposure. Some of the more popular ones out there include:

- Vulnerability scanning and penetration testing
- Bug bounties
- Source code reviews
- In-depth fuzzing of third-party applications (not common)

So, what approach is best? In reality, there is no silver bullet… A hybrid approach of various methods will offer best results and ROI. "Bug Bounties" are currently on the rise and an increasingly large number of organizations are rewarding security researchers with bounties for submitted bugs. Does that mean mandated penetration testing or source reviews are on the downfall? No, they are still highly useful in a structured SDLC approach, where testing is performed continuously throughout the SDLC, after which bug bounty hunting can take place once the system is in production.

Again, there is no silver bullet; there's just a lot of options to choose from. ☺

© 2019 Erik Van Buggenhout & Stephen Sims



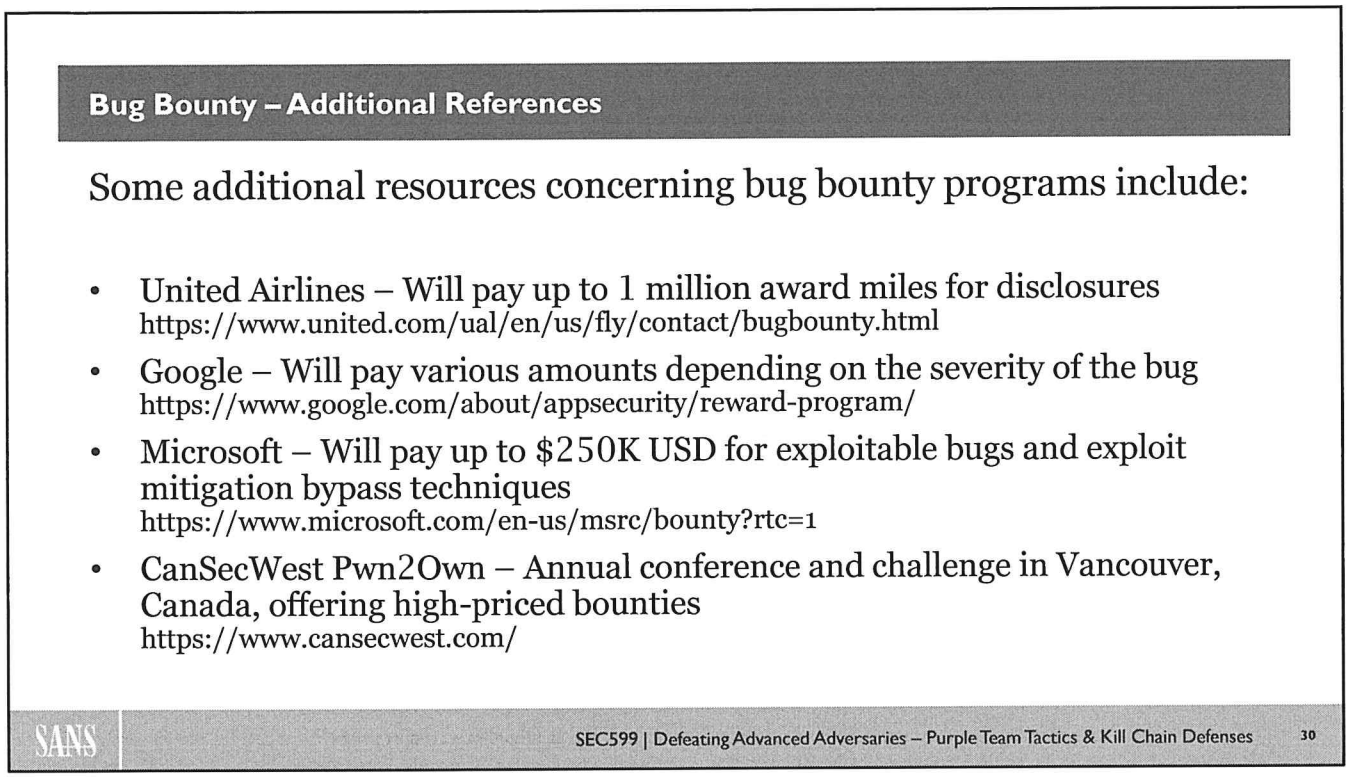
Bug Bounties

The more welcome vendors make those interested in disclosing a vulnerability feel, the more likely they won't sell it to a third party, release it publicly, or keep it to themselves. If they have to be concerned about potential legal action, many will simply avoid the risk. Some vulnerabilities can be worth a lot of money. This author has sold browser-based exploits affecting Internet Explorer for $10K - $20K USD to ethical buyers. Ethical buyers are those who disclose the vulnerability to the affected vendor and do not release details publicly, such as iDefense from Verisign and Tipping Point's Zero Day Initiative (ZDI). Windows remote code execution Kernel bugs can go for over $100K USD. Zerodium paid $1M USD to a group who disclosed an iOS remote jailbreak exploit. See here for details: https://www.zerodium.com/ios9.html. Some of the buyers are not as transparent in terms of what they do with a disclosed vulnerability. Some have a customer list that may be seen as questionable by some.

See also: https://www.macrumors.com/2016/09/30/zerodium-triples-ios-10-bug-bounty-to-1-5-million/

Offering a bug bounty program can help to make researchers interested in disclosing a bug more comfortable and more likely to disclose it to the affected vendor. As long as they follow the rules in the bounty program they should have no reason to fear legal action. The payment should scale based on the severity of the vulnerability. If the bounty is too low, then it is more likely for the disclosure to get taken elsewhere.

© 2019 Erik Van Buggenhout & Stephen Sims

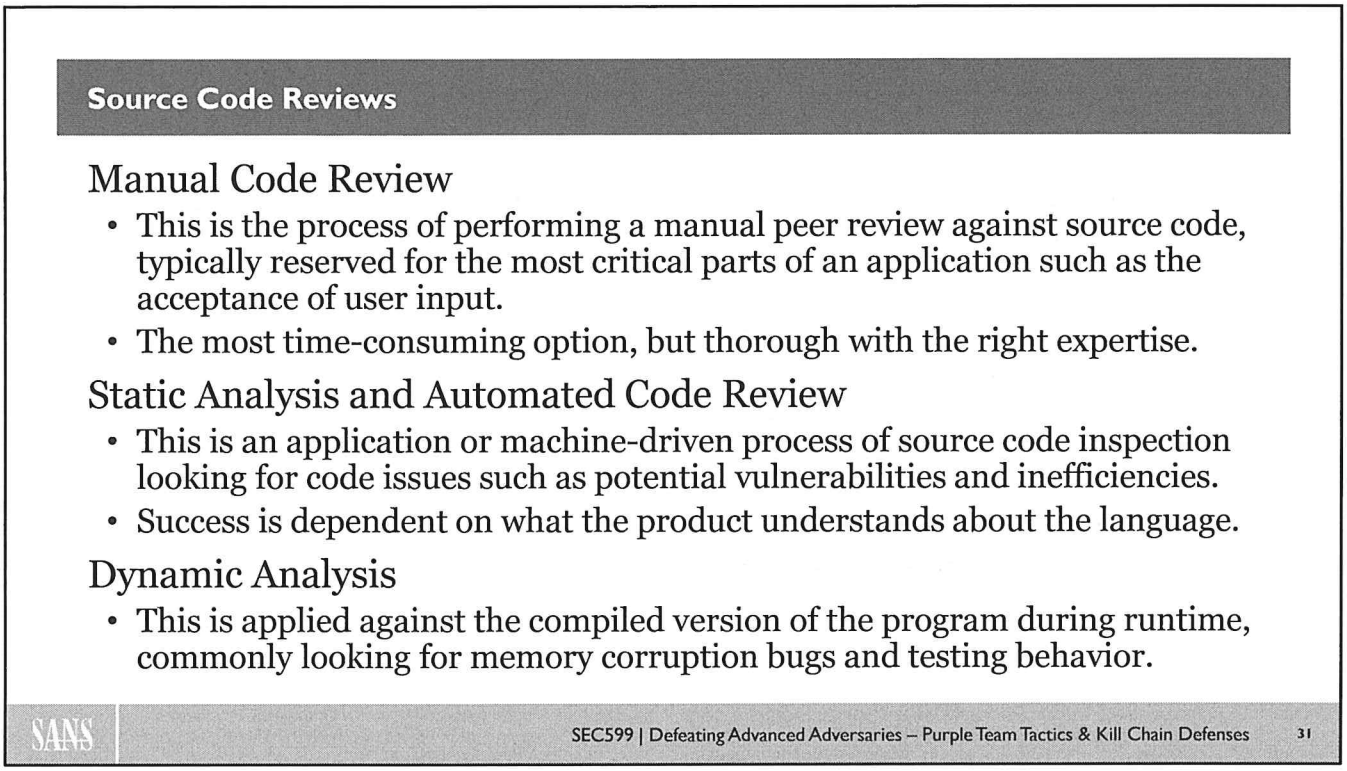


Bug Bounty – Additional References

There are a large number of vendors who offer different types of bug bounty programs. A few examples are listed on this slide, with some offering bounties such as airline award miles and others offering cash bounties in excess of $100K USD.

- United Airlines – Will pay up to 1 million award miles for disclosures
  https://www.united.com/ual/en/us/fly/contact/bugbounty.html
- Google – Will pay various amounts depending on the severity of the bug
  https://www.google.com/about/appsecurity/reward-program/
- Microsoft – Will pay up to $250K USD for exploitable bugs and exploit mitigation bypass techniques
  https://www.microsoft.com/en-us/msrc/bounty?rtc=1
- CanSecWest Pwn2Own – Annual conference and challenge in Vancouver, Canada, offering high-priced bounties
  https://www.cansecwest.com/

© 2019 Erik Van Buggenhout & Stephen Sims



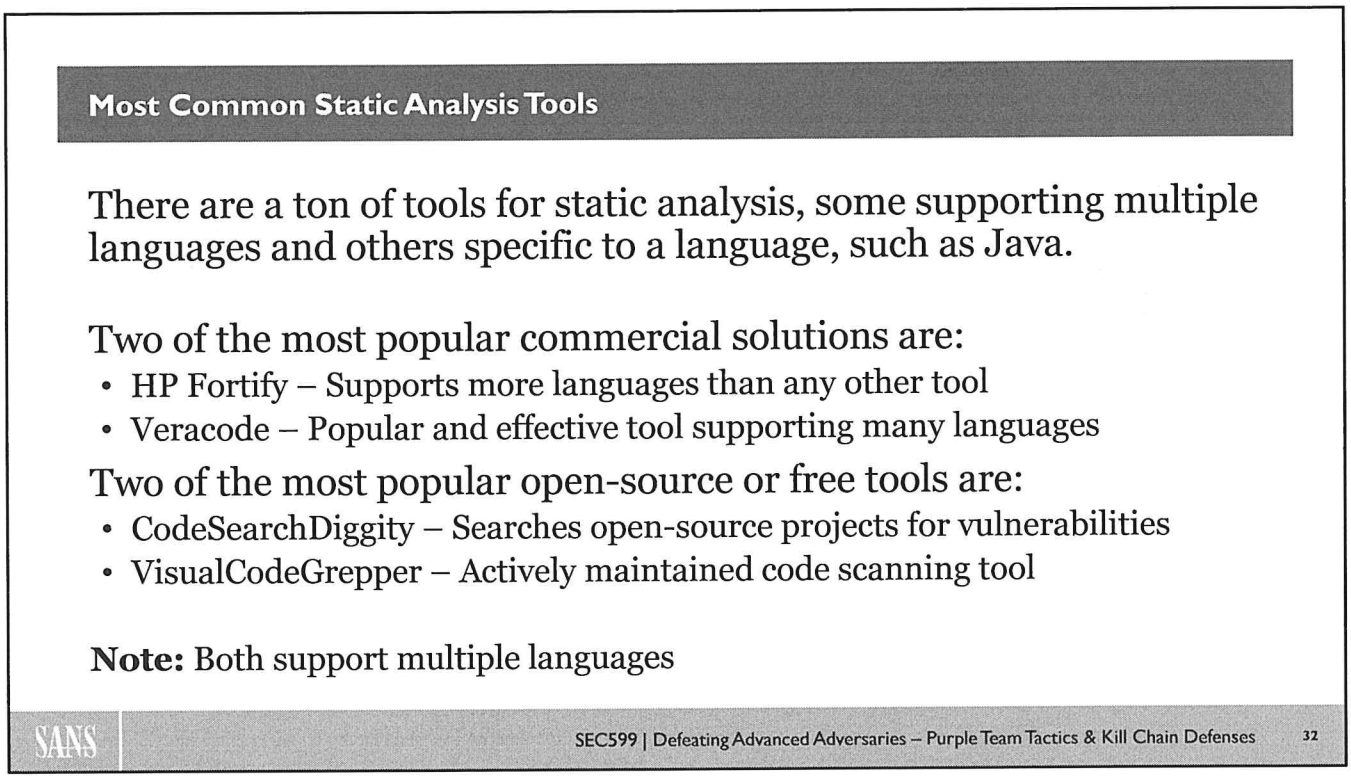
Source Code Reviews

There are various types of code reviews that can be performed. Manual code review is often seen as the most thorough but is time-consuming and typically reserved for the most critical parts of an application. This requires someone with the expertise to go line by line through code written by the development team. The quickest option is the use of static analysis tools. Tools such as the Fortify Static Code Analyzer from HP can take an entire project and quickly scan it for easy-to-spot vulnerabilities, banned functions, and programmatic inefficiencies. The good thing about static analysis tools is the speed at which they can review the code base. One issue is that they often produce a lot of false positives and miss complex bugs. The more they are taught about the supported languages being scanned, the better they can find problems. The term static analysis is also used when disassembling a compiled program for review with a tool such as the Interactive Disassembler (IDA). Dynamic analysis is typically performed during the validation phase of the SDLC once the code has been compiled. Various runtime tools are available such as Microsoft's AppVerifier for unmanaged code. (Unmanaged code refers to low-level languages such as C, C++, and assembly. Managed code refers to languages such as Java, C#, and Ruby.) These tools look for program issues typically related to problems such as heap corruption that are difficult to locate with static analysis. An example of a dynamic analysis tool for managed code is FxCop from Microsoft.

For more information on AppVerifier and FxCop check out the following links:

https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms220948(v=vs.90)
https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-3.0/bb429476(v=vs.80)

© 2019 Erik Van Buggenhout & Stephen Sims



Most Common Static Analysis Tools
If you use Google to search for "static analysis tools", you will get countless results. There are many lists put together by practitioners stating what they claim to be the best free and commercial solutions available. Some of these tools are specific to one or two languages, while others offer support for dozens of languages. There are both commercial tools and open-source or free tools. Two of the most popular commercial solutions in use today are:

HP's Fortify Static Code Analyzer – HP acquired Fortify Software in 2010, which included the Static Code Analyzer. The tool supports more languages than any other solution available at the time of this writing. Languages include C, C++, Python, Ruby, Visual Basic, Java, PHP, and many others. More information at https://www8.hp.com/us/en/solutions/business-solutions/printingsolutions/overview.html

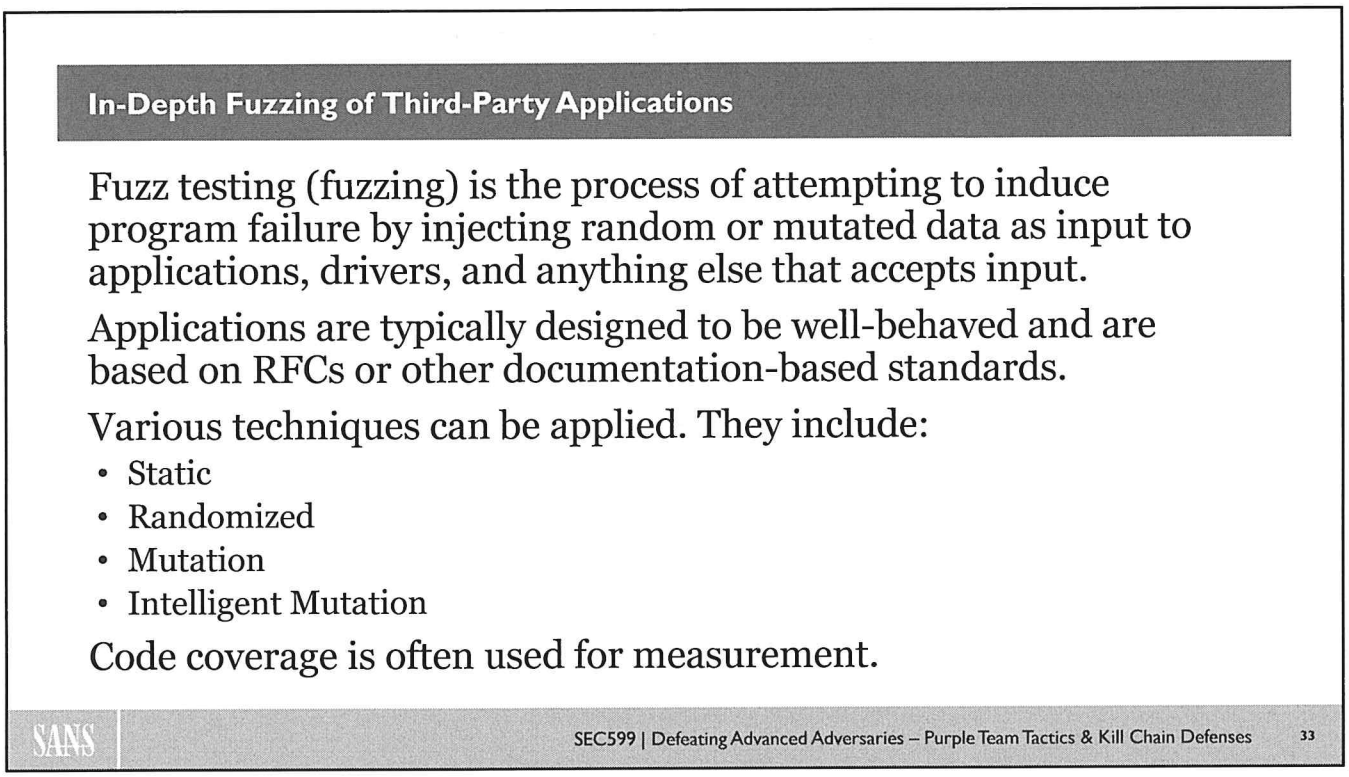
Veracode Static Analysis – The most common languages are supported, such as Java, C, C++, Objective-C, .NET, and others. More information at https://www.veracode.com/products/binary-static-analysis-sast

There are also a large number of open-source or free tools available for use. Two of the most popular:

CodeSearchDiggity – A tool included as part of the SearchDiggity project maintained by Bishop Fox that allows you to search through open-source code for vulnerabilities and other related issues. More information at https://resources.bishopfox.com/resources/tools/google-hacking-diggity/

VisualCodeGrepper – An actively maintained open-source code scanning tool supporting multiple languages including Visual Basic, PHP, C++, Java, and a couple others. More information at https://sourceforge.net/projects/visualcodegrepp/

© 2019 Erik Van Buggenhout & Stephen Sims



In-Depth Fuzzing of Third-Party Applications

Fuzz testing, also known as fuzzing, is the process of introducing random or malformed input to well-behaved network and file parsing applications, drivers, and anything else that accepts input. Googling for the definition of fuzzing will yield many different results. You have to imagine that the majority of applications are written based on a standard, such as those defined in a Request For Comment (RFC) document. Take the Session Initiation Protocol (SIP) defined under RFC 3261. SIP handles Voice Over Internet Protocol (VOIP) signaling, such as call setup and tear down. Many vendors, such as Cisco Systems and Avaya, use such protocols. The protocol is standardized so that one vendor's product is compatible with another vendor's product. These RFCs tell you the rules, but do not tell you how to write the code. There are countless ways to write the code using many different languages. When writing a fuzzer, you are attempting to test for various bug classes under many different conditions. You must think of, or have a tool do it for you, all of the ways a developer could have made a mistake and test to see if it exists. As you can imagine, this is a time-consuming process.

There are various fuzzing techniques that can be applied. Some of the most common testing techniques include static, randomized, mutation, and intelligent mutation:

- Static – Manually develop test cases that check for a specific condition, such as the existence of a buffer overflow vulnerability in a field of an IPv6 header. Even for this one check, in this one field, you would want to test for multiple sizes for the input as you do not know how large of a buffer was created by the developer or what function they chose to use to copy data into memory. This is a time-consuming technique that requires a lot of familiarity with the protocol or file format being tested.

- Randomized – This is a technique that requires little knowledge about the protocol or file format being tested. Fields are selected, and then random data is inserted into the field in an infinite loop until stopped.

- Mutation – This is a technique that requires little knowledge about the protocol or file format being tested. Fields are selected, and then mutated data is inserted to test for conditions associated with various bug classes.

© 2019 Erik Van Buggenhout & Stephen Sims

- Intelligent Mutation – This technique requires deep familiarity with the protocol or file format being tested. Test cases are written to reach specific points in an application, and then selected fields are chosen for the introduction of mutated data. This is the most comprehensive technique if properly used. Tools can be used to measure the amount of code reached to ensure coverage is maximized.

© 2019 Erik Van Buggenhout & Stephen Sims

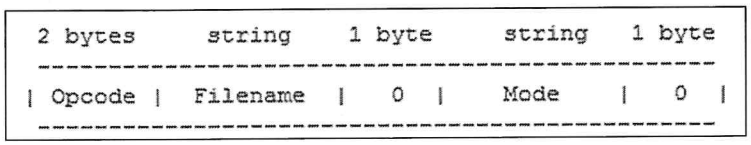
Fuzzing Example: Trivial File Transfer Protocol (TFTP)

TFTP defined in RFC 1350:

- "TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP." (Sollins, K.)

This protocol will serve as our example when going through the various fuzzing types to help add context.

- We will use the following fields taken from the RFC which indicate the request type, filename, and mode:

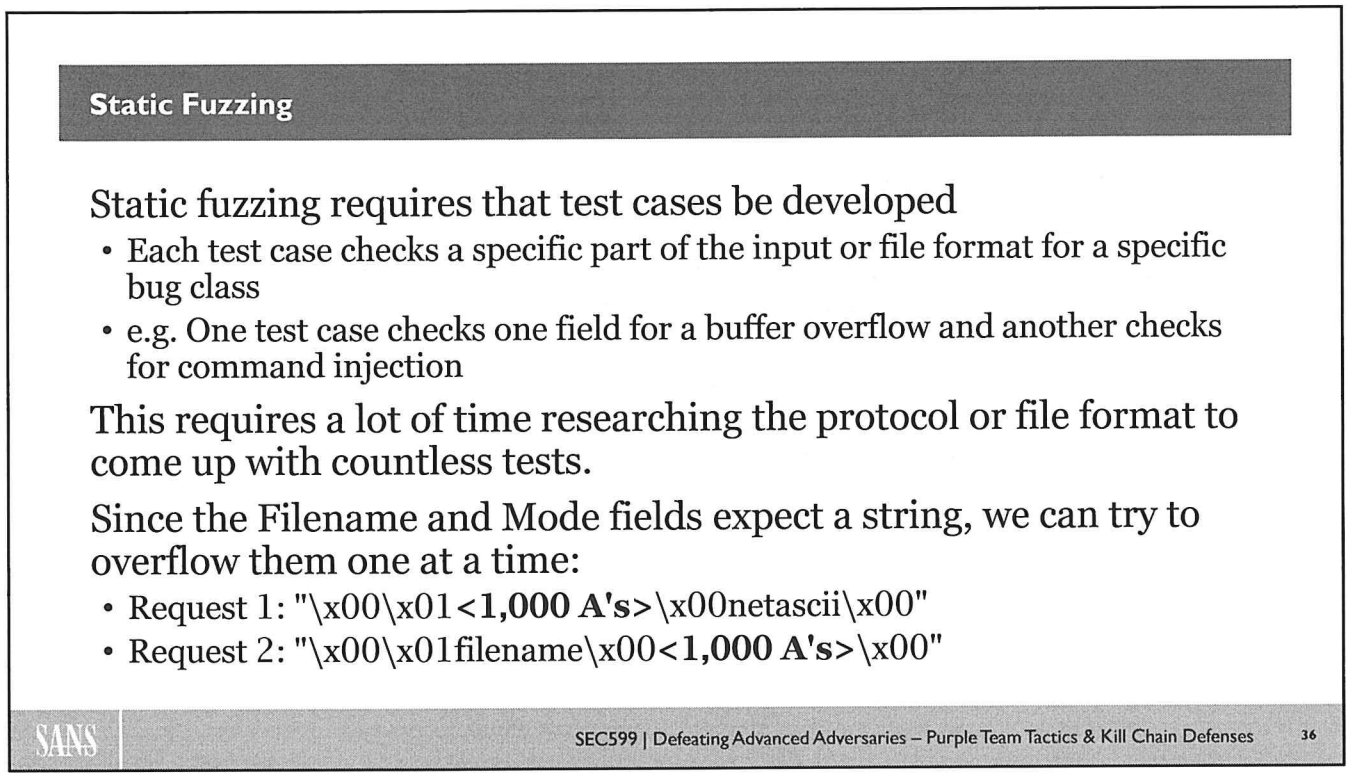


Fuzzing Example: Trivial File Transfer Protocol (TFTP)
The TFTP protocol is defined in RFC 1350. As stated in the RFC, "TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP." (Sollins, K.) This protocol was selected due to its simplicity. We will use it as an example for the various fuzzing techniques described shortly.

The image on the slide, taken from the RFC, shows the fields required when making a TFTP request. You would typically have a TFTP client that sends a request to a TFTP server. This request is most commonly a read or a write request. A read request means you would like to get something from the server and a write request means that you would like to put something onto the server. The 2-byte field for "Opcode" expects a "\x00\x01" for a read request and a "\x00\x02" for a write request. After this 2-byte field is the "Filename" field, which expects a string. In other words, it wants you to include the name of the file to read or write. A question that must be asked is, "How does the server know when it's reached the end of the filename?" The next field is a 1-byte null or 0. When the server reads in the name of the file, it knows it's reached the end when it hits the null byte. The next field is the "Mode" field which is also to be a string. Options include "netascii," "binary," and "mail." Being that this field is also a string, it terminates with a null byte.

Citations and References:
Sollins, K. "The TFTP Protocol (Revision 2)." RFC 1350 The TFTP Protocol (Revision 2). https://tools.ietf.org/html/rfc1350 (accessed February 1, 2017).

© 2019 Erik Van Buggenhout & Stephen Sims



Static Fuzzing

Static fuzzing requires the tester to spend a lot of time understanding the protocol to be fuzzed. Any documentation available, such as an RFC, would be inspected closely in order to begin creating test cases. Each test case would test for one condition. You have to remember that many languages can be used to write something such as a TFTP client or server. Each language comes with its own set of potential problems when poorly coded. Since we just saw that the Filename and Mode fields require a string and keep reading until reaching a null byte; either one could be a field that could be one we'd want to check for the presence of a buffer overflow. On the slide, we have two examples:

Request 1: "\x00\x01**<1,000 A's>**\x00netascii\x00"
Request 2: "\x00\x01filename\x00**<1,000 A's>**\x00"

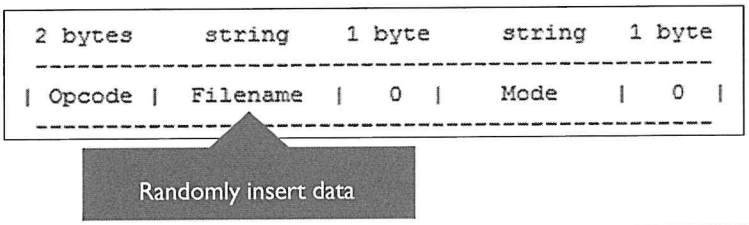
Request 1 checks the Filename field to see if a string of 1,000 A's causes an overflow. Request 2 does the same, but for the Mode field. Just because a server doesn't crash with 1,000 A's doesn't mean you would move onto testing for another bug class. It might crash at 500 A's, but not 1,000, or any number of bytes below, between, or above. It all depends on how the code was written and any number of idiosyncrasies. Each program is different and is why we would need to come up with a large number of test cases that use good samples to test for as many conditions as possible. This can result in thousands of test cases.

© 2019 Erik Van Buggenhout & Stephen Sims

Randomized Fuzzing

Randomized fuzzing is good from the perspective of not needing advanced knowledge of the protocol or file format being fuzzed.

- The tester simply selects a desired field to fuzz and the fuzzing application randomizes the input infinitely at that location
- Proper monitoring of the fuzzing session becomes critical using this technique
- If you crash the application but don't have monitoring, you may never hit that condition again



Randomized Fuzzing

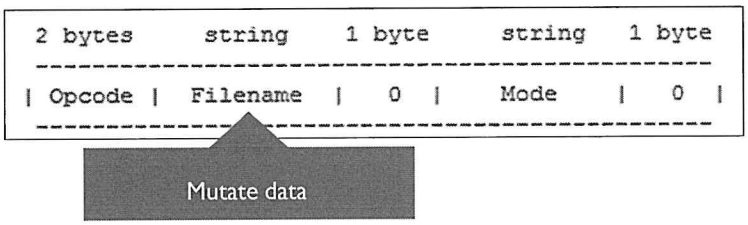
Randomized fuzzing requires no upfront knowledge about the protocol or file format being fuzzed. Typically, a sample file or network packet data is acquired. This may be a valid PNG or JPG file, a captured SSH stream, or any number of options. A field is then selected to be fuzzed. The fuzzer then replays the network stream or produces a large number of files if fuzzing a file format, randomizing the data in the selected field. This would run infinitely until stopped. As you can see, it is a simple concept that requires good monitoring to catch the crash. Take the example of fuzzing an SSH server. The fuzzer will continuously connect to the server, sending random data in the selected field. If this causes a crash and the server dies, how do you know what random data was sent to cause the crash? Since it is random, if you start it over, you may never hit that test case again. It is, therefore, critical to record the data being sent to the server so that when it crashes, the most recent samples sent are available.

© 2019 Erik Van Buggenhout & Stephen Sims

Mutation Fuzzing

Mutation fuzzing also requires little knowledge of the protocol or file format being fuzzed.

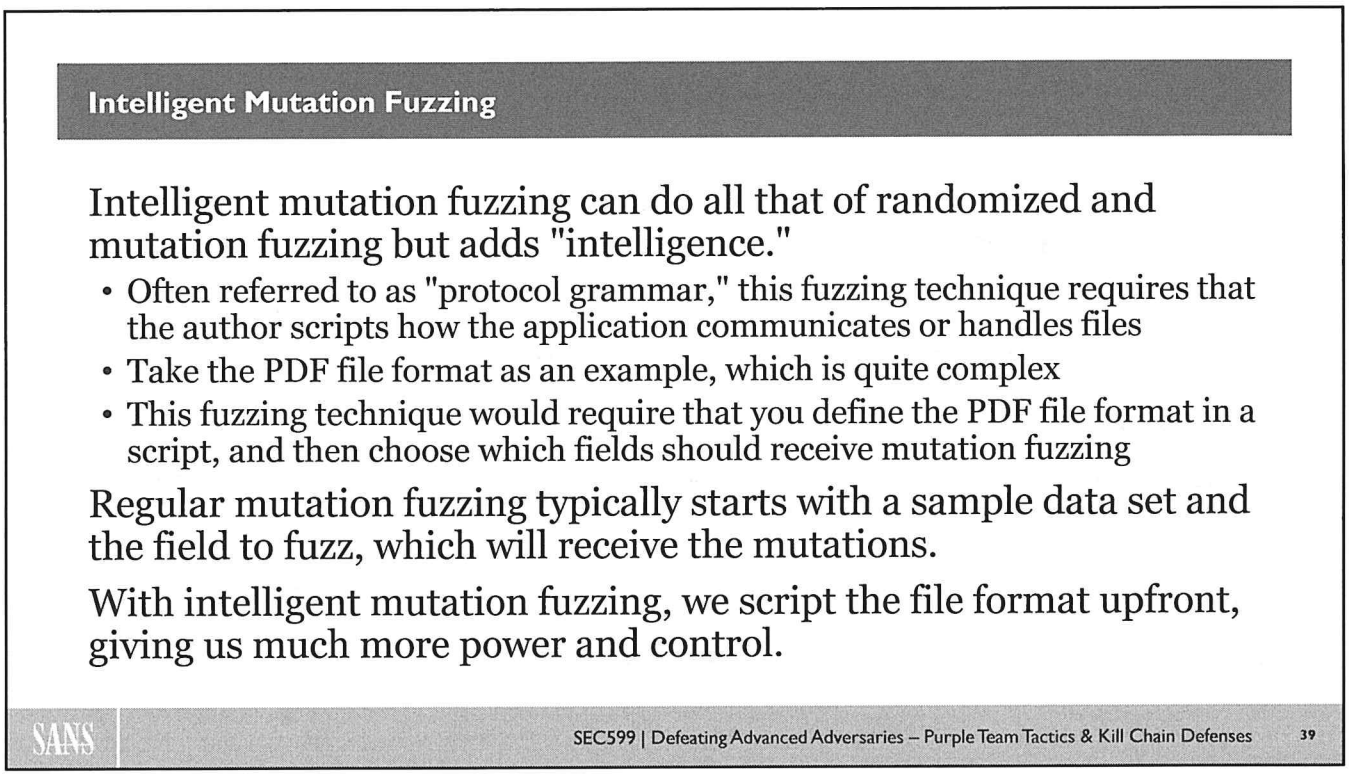
- Instead of the data being random in the selected field, the data is mutated based on various bug classes
- It is finite in that there are only so many mutations to go through in each bug class being checked
- Once exhausted, the fuzzing session ends



Mutation Fuzzing

As with randomized fuzzing, mutation fuzzing requires little to no upfront knowledge about the protocol or file format being fuzzed. Where it differs is that instead of random data being inserted, mutations are generated based on various bug classes such as buffer overflows, command injection, and others. It is finite since there are only so many mutations created per each bug class. This means that if you caused a crash, but missed the mutation that caused it, you could restart the fuzzer and likely hit the same condition.

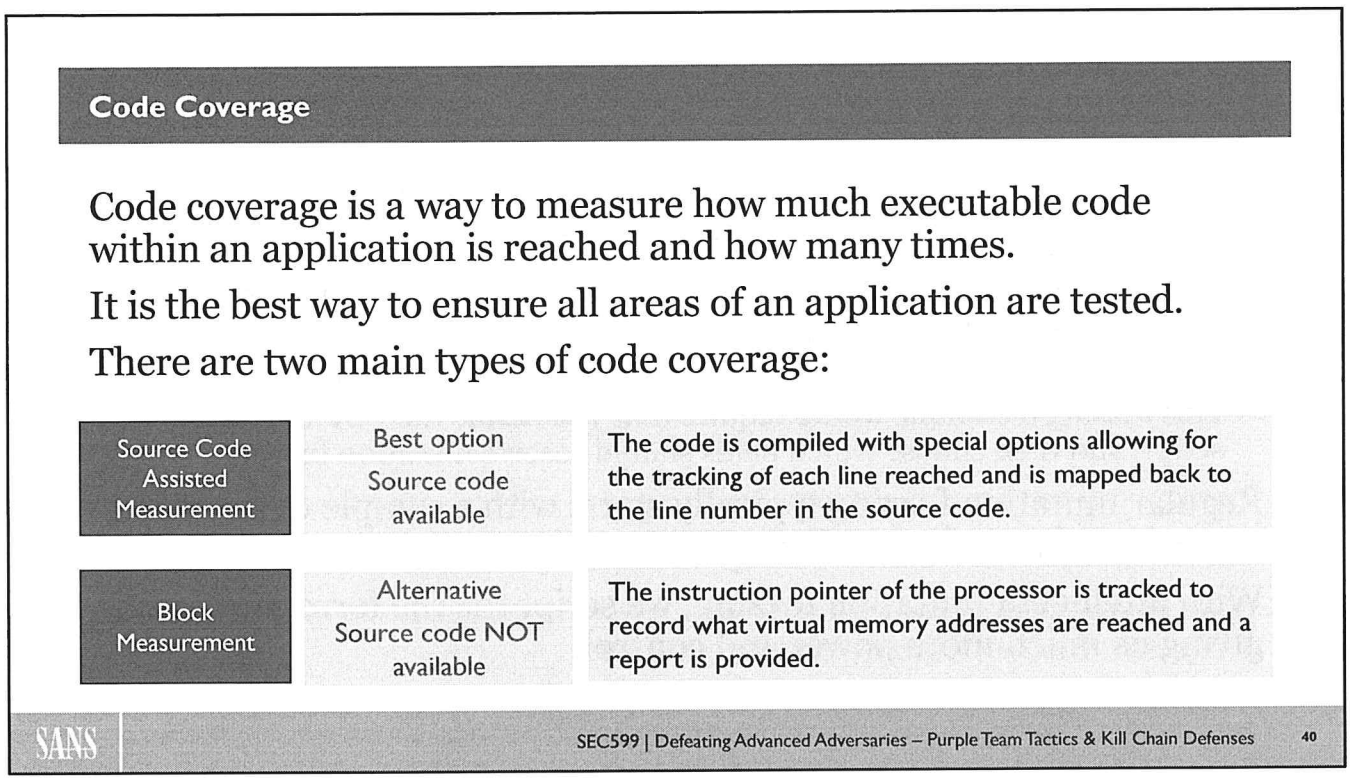
© 2019 Erik Van Buggenhout & Stephen Sims



Intelligent Mutation Fuzzing

Intelligent mutation fuzzing goes above and beyond the limitations of randomized and mutation fuzzers. It requires a lot of upfront time spent to understand the protocol or file format being fuzzed, similar to that of static fuzzing. With randomized and mutation-based fuzzing, a sample data set is typically used as a starting point. For example, if we wanted to fuzz the PDF file format, we would start with a legitimate PDF document and select fields to mutate or randomize. With intelligent mutation fuzzing, we do not start with a sample data set. Instead, the file format or network protocol grammar is scripted into a source file. Some of the fields are left as static so they are not fuzzed, and others are selected for fuzzing. If an HTTP server must always receive a request type such as PUT, GET, POST, and HEAD, then we always want to lead with those values and don't want to fuzz them. Then, we can script out the rest of the interaction with the HTTP server, choosing desired fields for mutation. This allows us to get better code coverage.

© 2019 Erik Van Buggenhout & Stephen Sims

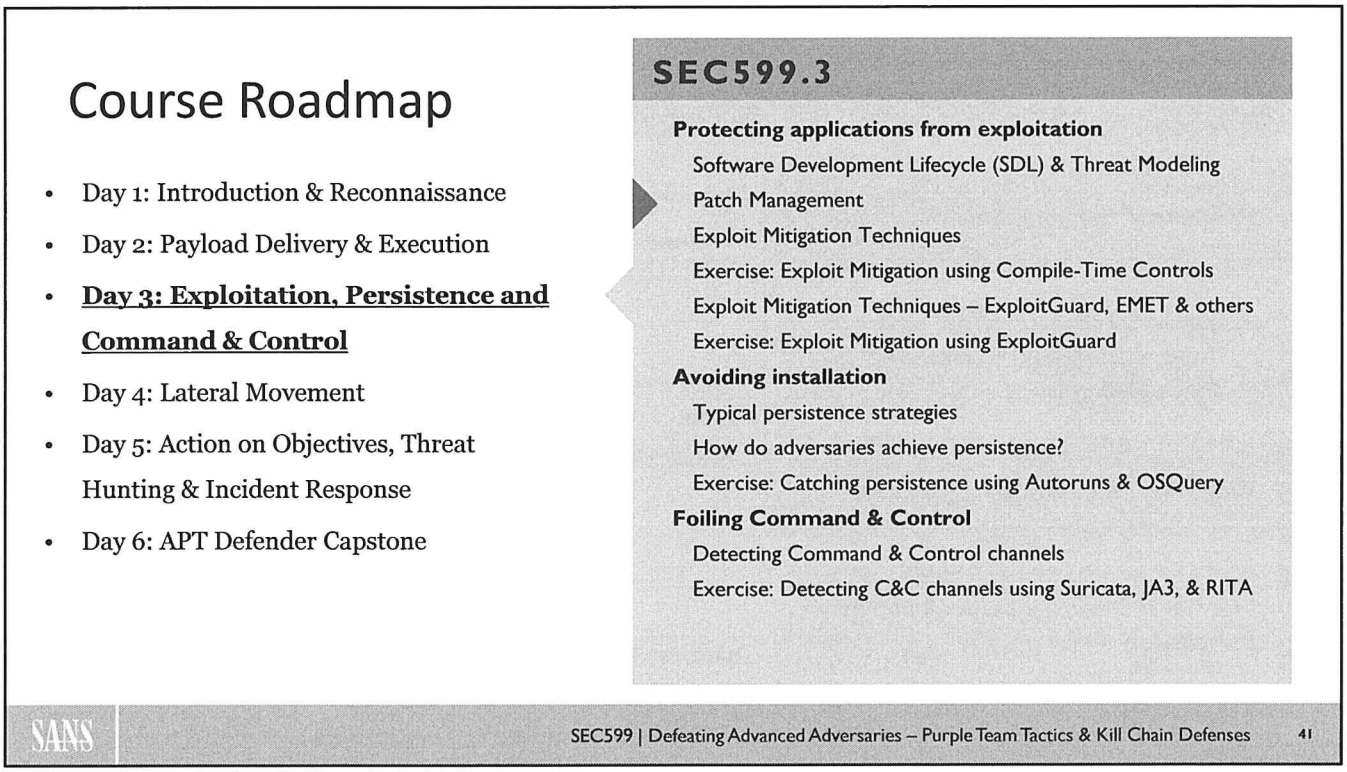


Code Coverage

Imagine if you were a law enforcement officer tasked with searching a large building for a suspect. It would likely be expected that you search each and every room. If you skip ¾ of the rooms as many are locked, then your coverage isn't very good. Now think of a large application such as Microsoft Word. There are many features and a lot of executable code. How do you know what lines of code you have reached, and which ones were not reached? You need a code coverage tool. The low-hanging fruit that is easy to reach in an application has likely been repeatedly tested and less likely to have a bug. The code that is harder to reach is more likely to have a bug.

There are two main forms of code coverage testing, source code assisted measurement and block measurement. Source code assisted measurement requires that you have the source code of the application being fuzzed, which is not always available. It also requires that you compile the source code with special debugging options to support the technique. There are various examples of source code assisted fuzzers and code coverage tools, such as the American Fuzzy Lop (AFL) by Michal Zalewski (lcamtuf) at Google. Source code line numbers are mapped to the compiled virtual addresses and then recorded at runtime during a fuzzing session. Tools such as the AFL also generate test cases (mutations) and automate the process. Block measurement can be used when source code is not available. A block is a grouping of code within a function that typically ends with a conditional branch, return, or other instruction. Imagine the statement, "If the value being checked is 0 go right and if not 0 go left." This branch would lead to another block of code within the function. In block measurement, each block's virtual memory address is recorded for later analysis.

© 2019 Erik Van Buggenhout & Stephen Sims

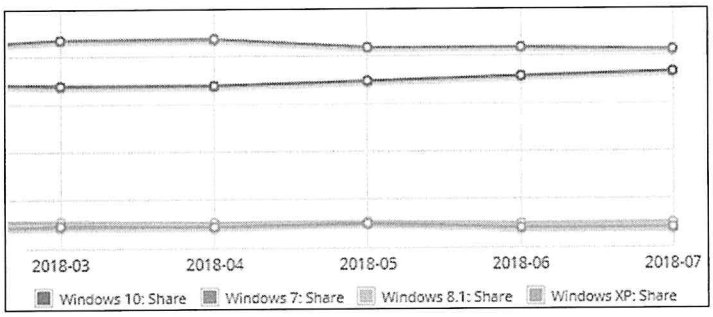


This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

OS Market Share

- Windows 7 still dominant.
- XP still at 5.23%
  - Automated Teller Machines (ATMs)
  - Embedded systems
- Windows 10 quickly gaining traction.
- Mac OS and Linux still a small number in comparison.



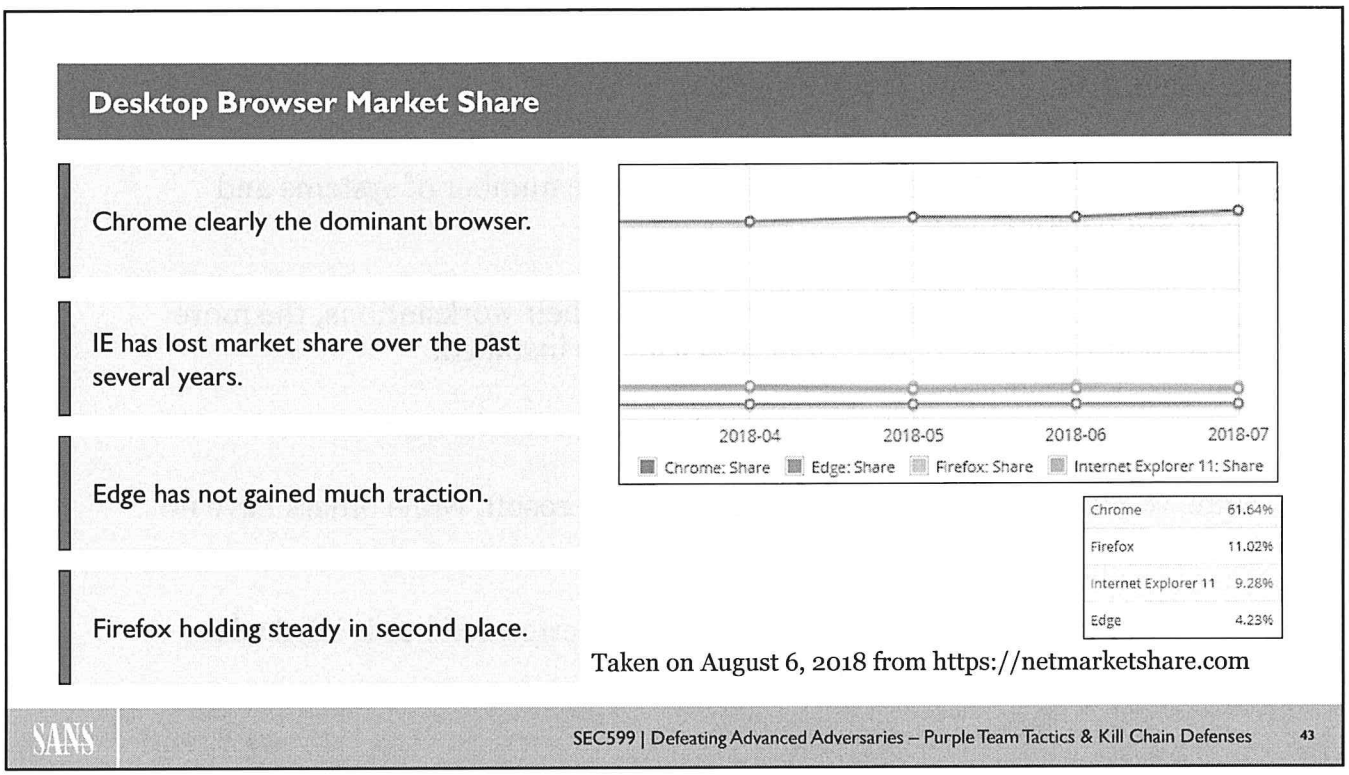
Windows 7	42.95%
Windows 10	32.98%
Windows 8.1	5.59%
Windows XP	5.23%

Taken on August 6th, 2018 from https://netmarketshare.com

OS Market Share

The image on this slide gives you an idea as to the state of OS market share, clearly showing Microsoft Windows as the dominant OS. At the time this was pulled in August 2018, Windows 7 was still the leader, with Windows 10 in second place. There are limitations in Windows 7 from a security perspective that cannot be fixed. An example is Control Flow Guard (CFG), a built-in security control we will discuss later this day. This was backported to Windows 8.1, but not Windows 7. There are also many improvements in the Kernel on Windows 10 that do not exist in Windows 7. It is critical to keep these older OSes up to date. MacOS is shown at the NetMarketShare site; however, it is low enough to not display on the graph.

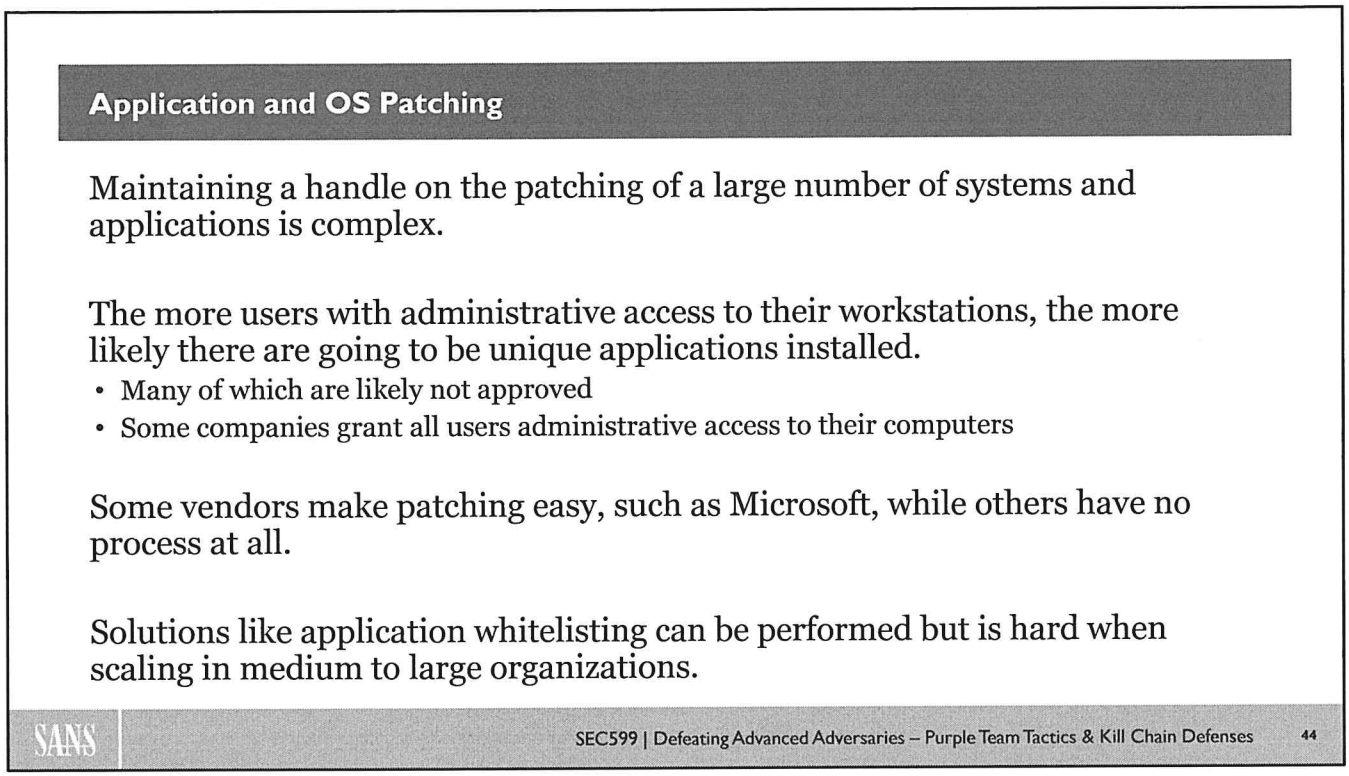
© 2019 Erik Van Buggenhout & Stephen Sims



Desktop Browser Market Share

On this slide is the desktop browser market share as of August 2018. Note that these numbers change, so it is a good idea to use the provided link to see the updated percentages. For a very long time, Internet Explorer was the dominant browser; however, Chrome has taken a large lead over the past couple of years. Microsoft Edge has struggled to gain traction. Browsers have been the source of many compromises over the years and as such, security has been greatly improved. Bug classes such as Use After Free (UAF) were one of the primary vectors in end-user compromises, but security improvements such as MemGC have greatly mitigated the technique.

© 2019 Erik Van Buggenhout & Stephen Sims

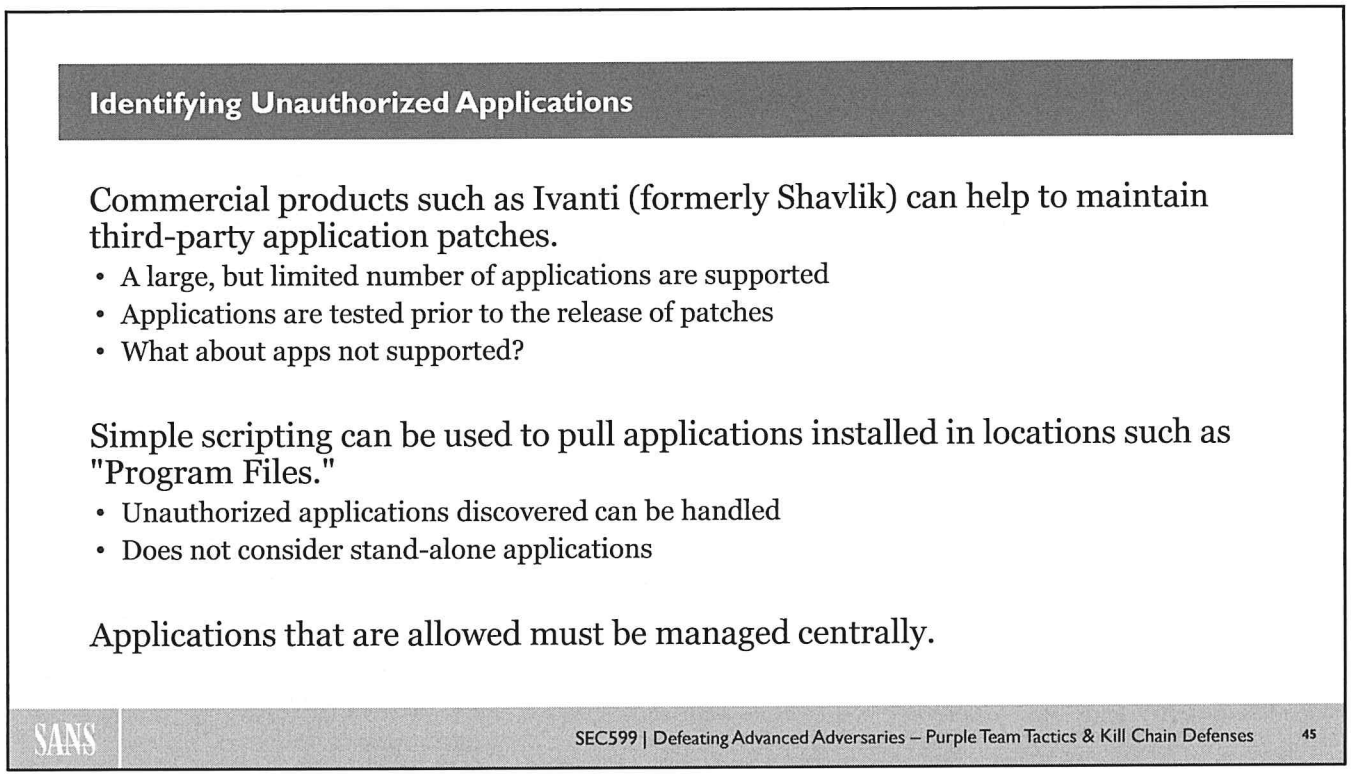


Application and OS Patching

As we just saw, it is no secret that the majority of companies use Microsoft Windows as their OS for employee workstations and laptops. Fortunately, Microsoft has a mature process for patch management with the well-known Patch Tuesday. Other OS vendors are not so clear as to when patches will be released, such as Apple. Some vendors, such as Oracle, have scheduled updates, but less frequently than Microsoft. Oracle releases updates each quarter. The mobile market makes things more complex, especially with some Android devices that are incapable of even being updated. Regardless, most organizations have a good handle on the patching of operating systems.

A different issue is around the patch management of installed applications. Some very large organizations, which will not be named, give administrative access to all employees. The reasoning for taking this approach is up to each company and their policies. A strong application whitelisting policy and enforcement can help mitigate the installation of unauthorized applications; however, whitelisting is difficult to manage. At organizations that do not grant all users administrative access, it is fairly typical for many users to be given approval to have this level of access. Take a network engineer, as an example, who is able to justify the need for administrative access to perform job-related tasks. The justification may be valid; however, it often results in the installation of third party applications that are not on the organization's approved list.

© 2019 Erik Van Buggenhout & Stephen Sims



Identifying Unauthorized Applications
At one organization this author worked at, we wrote a script to pull the name and version of all applications installed on every system where the user had administrative access. The result was a staggering 1,100+ applications. Some of these were the same application but different versions. When Googling for the words "<application name> vulnerability" there were countless hits. The point of this example is to demonstrate the complexity of trying to manage patches for that many applications not on the approved list. This also did not include stand-alone applications, but instead only applications installed in "Program Files" on Windows systems.

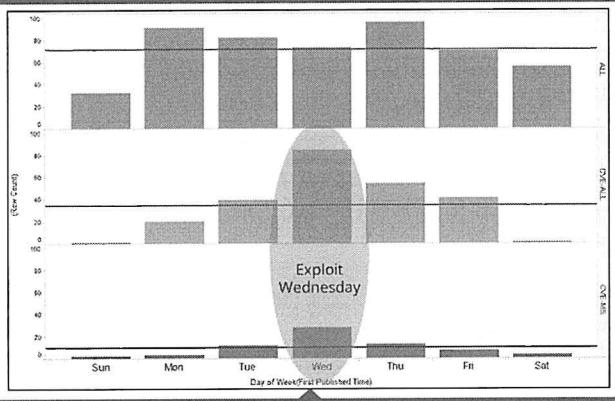
There are various commercial products available to aid in managing patches to third-party applications, such as Ivanti (formerly Shavlik). These solutions typically focus on the most common applications such as Flash, browsers, document readers, Oracle, etc.... Part of the service offered by these solutions includes the testing of the patches to ensure applications are not negatively affected. You would need a process for applications that are not supported.

More information:
https://www.ivanti.com/solutions/needs/manage-my-os-and-third-party-application-patches

© 2019 Erik Van Buggenhout & Stephen Sims

Microsoft Patch Tuesday

- Microsoft releases patches on the **second Tuesday of each month**, for now...
- Effort to help **simplify the patching process**: Random patch releases caused many users to miss patches.
- Patch delay of max. 30 days has security concerns: **Emergency patches are released out-of-cycle.**
- Many **exploits** released in the days following a patch (="Exploit Wednesdays").



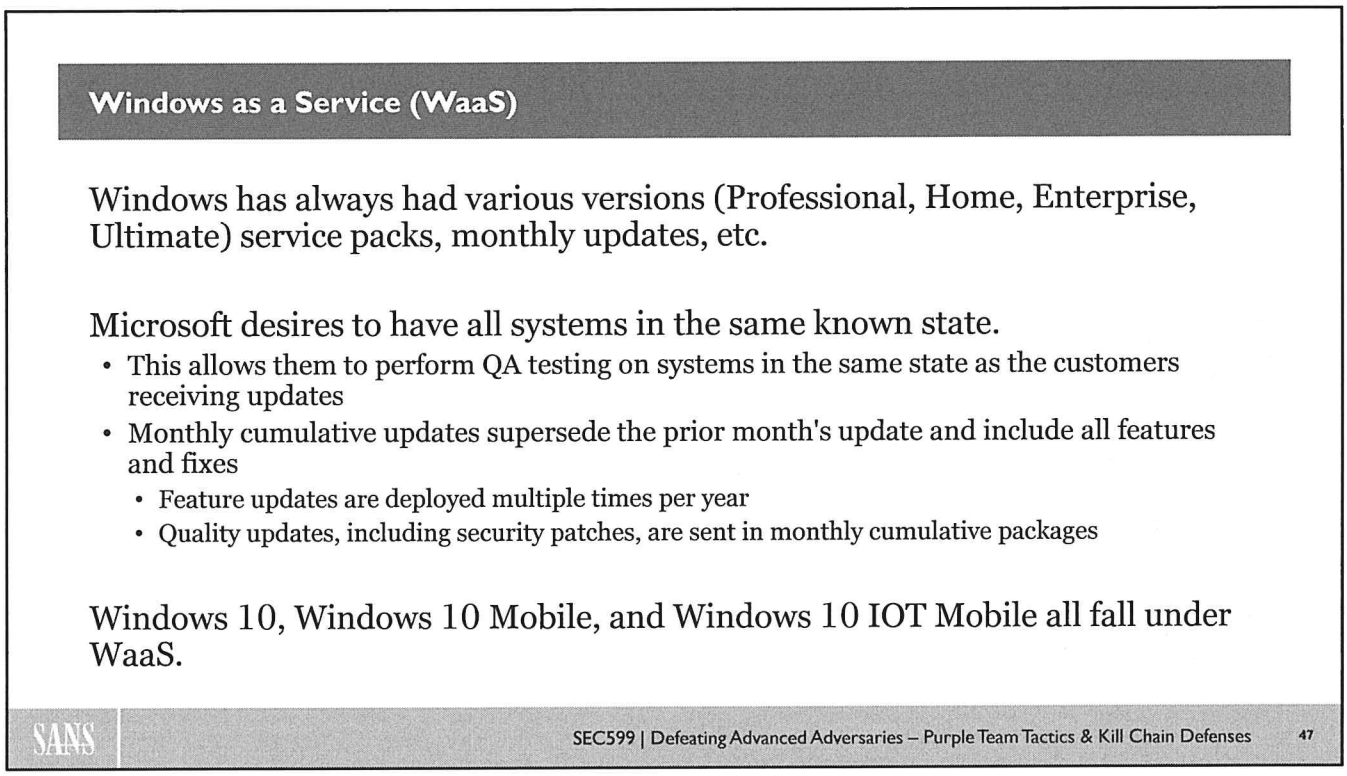
Exploit Wednesdays
Analysis of a Russian hacker forum's traffic has shown evidence for exploit Wednesdays. While generic forum posts remain approximately constant throughout the week, there is a strong increase in traffic related to generic and Windows CVEs on Wednesdays.

Microsoft Patch Tuesday
In October 2003, Microsoft started "Patch Tuesday." This came after complaints from users and administrators who stated it was difficult to keep up with patching their systems when it was unknown as to when patches would be released. The patches were released by Microsoft as they were approved. Users and administrators had to be constantly ready to handle the release of new patches. It is now well known that the second Tuesday of each month, Microsoft will release patches, both security-related and functionality or maintenance-related. The idea was that it would simplify the patching process. Advanced alerts are sent out from Microsoft to inform and prepare users of the nature of each patch. Most organizations have adapted to "Patch Tuesday" and have a process in place to test patches, followed by deployment to their systems. There are many services available to assist with deployment, from automatic updates on each Microsoft OS to Windows Server Update Service (WSUS) helping with large-scale patch management and deployment. Third-party applications are also available.

There are concerns around the waiting period in between patch releases from Microsoft. It is no secret that many exploit developers wait for patches to be released so that they can compare the patched version of a function or library to that of the unpatched version. Tools such as IDA Pro and BinDiff can quickly locate changes to the code. An experienced reverse engineer can locate the vulnerability within the unpatched code and write programs to reach the location within the affected program. This results in the release of cutting-edge exploits, which often prove lucrative to an attacker because many organizations do not quickly patch their systems. Exploits are sometimes released the following day after a patch is deployed by Microsoft, which caused the term "Exploit Wednesdays" to get adopted in the IT landscape. There is also the issue around attackers intentionally waiting until the day after patch Tuesday to release new unknown known exploits, knowing that it will likely not be patched for up to 30 more days. Microsoft does occasionally release out-of-band patches for critical updates; however, systems are often left unpatched for weeks. Workarounds are often provided, but this is only a temporary fix and is not always practical. Patch diffing is not only used by the bad guys. Those working for organizations often reverse engineer patches to determine the effect to the organization of patch application or to determine the impact of the vulnerability. Intrusion Detection System (IDS) signatures can also be developed from a thorough understanding of a vulnerability, as well as developing modules for vulnerability scanning and penetration testing frameworks.

Exploit Wednesday source: Recorded Future: https://www.recordedfuture.com/hacker-forum-traffic/

© 2019 Erik Van Buggenhout & Stephen Sims



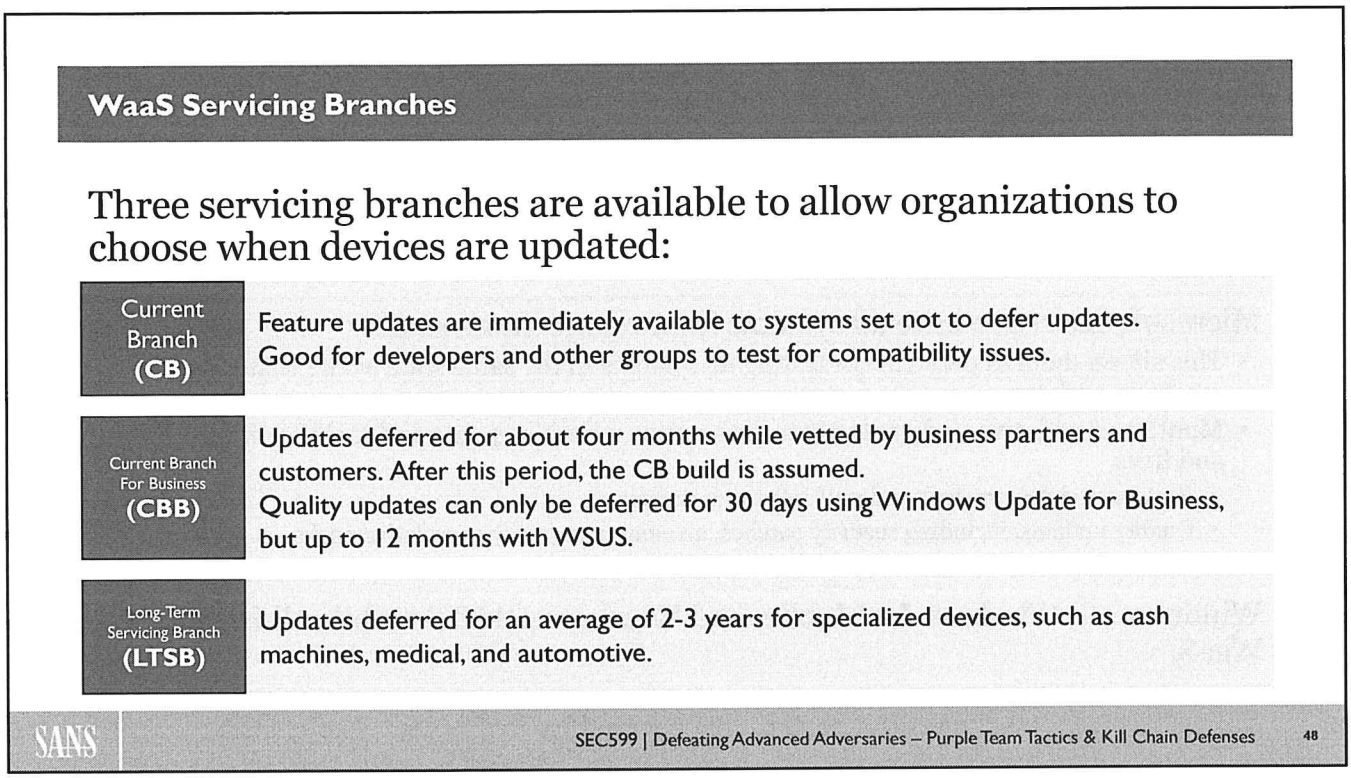
Windows as a Service (WaaS)
Prior to the release of Windows 10 Beta, Microsoft started to release bits of information about the idea of Windows as a Service (WaaS). Little information was initially available, and many websites tried to define what it could mean. Even today, it is still complicated to digest with the introduction of servicing branches and various deferral options. Microsoft desires for all systems to be in the same state. This makes their life easier as the systems out in production around the world look more like the ones in their testing labs. Knowing the state and build of all systems out there should result in fewer compatibility problems. Each month, a cumulative update is made available that supersedes the prior month's update. These cumulative patches include all updates for OS version. There are two types of updates: Feature and quality. They wish to do away with things like "Service Pack 2" and "Revision 3" and standardize on not more than two supported builds. The quality updates include security patches. Picking and choosing which updates to apply results in systems that are in many different states. Cumulative roll-ups help to ensure a system has all necessary patches to remain secure and support newer features.

Although cumulative updates are now pushed out for all systems, ever since October 2016, Windows 10, Windows 10 Mobile, and Windows 10 IOT Mobile are the only ones falling under the official WaaS practice.

Reference:
Halfin, Dani. "Overview of Windows as a service." TN Overview of Windows as a service. https://docs.microsoft.com/en-us/windows/deployment/update/waas-overview (accessed January 29, 2017).

More Information:
https://docs.microsoft.com/en-us/windows/deployment/update/index

© 2019 Erik Van Buggenhout & Stephen Sims



WaaS Servicing Branches
Three servicing branches are available from Microsoft to help with patch distribution. It actually gets quite complex when evaluating the various update server options such as Windows Update for Business, Windows Server Update Services (WSUS), and the System Center Configuration Manager. The references provided below and associated links at those locations are very useful in trying to understand how your Windows 10 organization can architect the right solution. The three branches are:

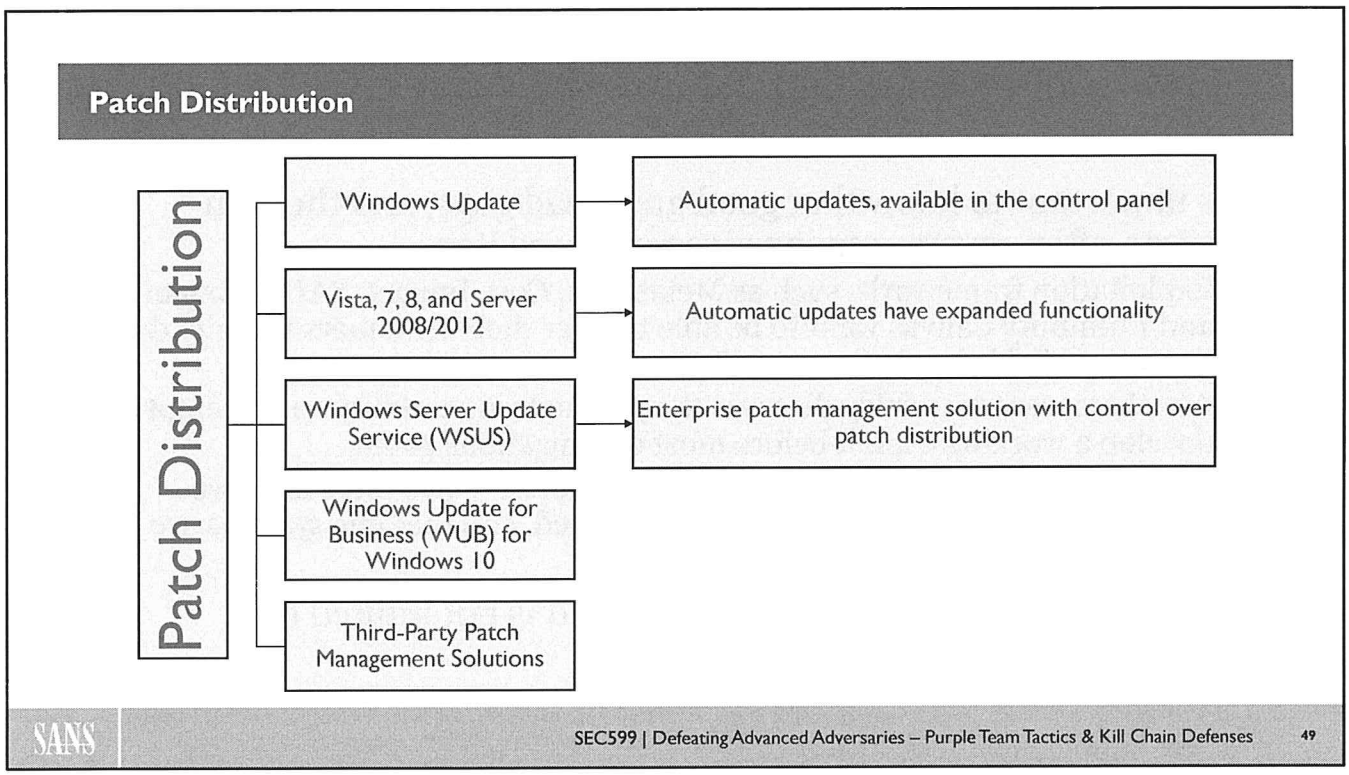
Current Branch (CB) – This branch makes features available as soon as they become available so that groups such as developers and QA can begin ensuring there are no compatibility problems, or those looking to take advantage of the new features can get started as soon as possible.

Current Branch for Business (CBB) – This branch is designed for wide-scale deployment to an enterprise. All of the features and such made available in the CB are moved to the CBB after a few month's vetting process involving customers and business partners. Quality updates can be deferred by different amounts depending on the Group Policy Option (GPO) settings pushed out and the patch distribution server option being used. This is likely going to continue to change and evolve as feedback is received by Microsoft.

Long-Term Servicing Branch (LTSB) – This branch is designed for specialized devices such as ATM/cash machines, medical devices, automotive devices and others. These are devices that have a specific focus or role and do not utilize the features made available by Microsoft.

References:
Halfin, Dani. "Overview of Windows as a service." TN Overview of Windows as a service. https://docs.microsoft.com/en-us/windows/deployment/update/waas-overview (accessed January 29, 2017).
Halfin, Dani. "Managing updates using Windows Update for Business." TN Managing updates using Windows Update for Business. https://docs.microsoft.com/en-us/windows/deployment/update/waas-manage-updates-wufb (accessed January 29, 2017).

© 2019 Erik Van Buggenhout & Stephen Sims



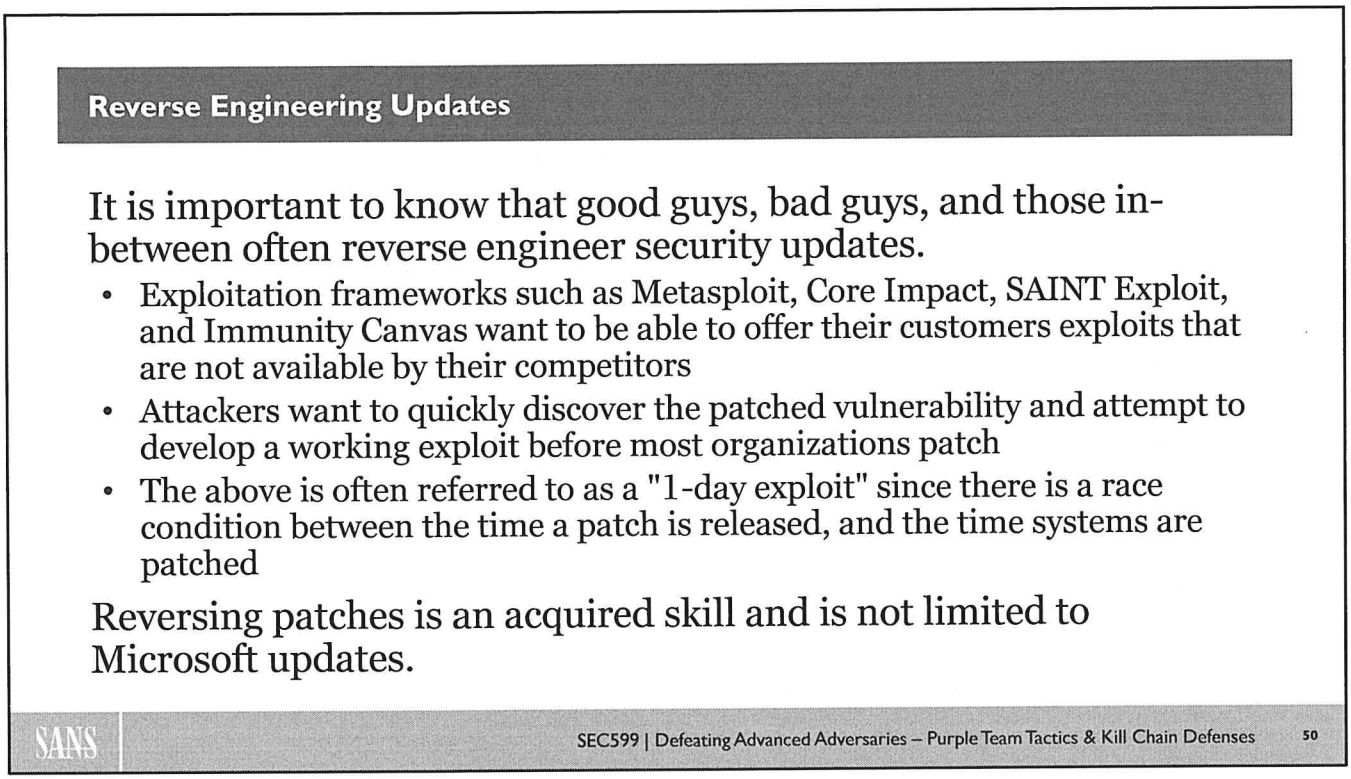
Patch Distribution

This slide serves as a simple high-level overview of the Microsoft patch distribution process. Many organizations do not permit end users to connect to Microsoft to obtain patches. Instead, a centralized enterprise patch management process controls patch distribution. The reasoning behind such a solution ranges from system consistency to security to application stability. The ability for each user to connect at any time to the Microsoft update site and install desired patches renders the system builds to be highly inconsistent. Some patches have even been known to introduce new vulnerabilities. Other patches have been known to cause applications to break or behave differently than when the patch was not installed. All these issues make it desirable to control the distribution and installation of patches on end user systems and servers.

Automatic updates have been installed by default on Windows systems since Windows ME, XP, and Windows 2000 Server. Automatic updates can be used to check for updates, check for updates and download them, and check for updates, download, and install them. Enterprise patch management often takes advantage of Windows Server Update Service (WSUS) servers to communicate directly with Microsoft update servers. Updates can be scheduled and sent directly to the WSUS servers over HTTP or HTTPS. Administrators then have the ability to first test the patches prior to deployment. Automatic updates on each end user system can be configured to communicate only with the enterprise WSUS servers. Administrators can select which patches they want pushed out and when. They also have the ability to set whether a patch can be deferred by the user and how soon a reboot is required if applicable. Windows Update for Business (WUB) is available starting with Windows 10. Update deferral is more limited, and Microsoft sees it as more of a constant stream of updates that should be installed as soon as possible. Check out SANS SEC505 "Securing Windows and PowerShell Automation" for more information on securely architecting Windows domains and a building a patch management process.

Third-party patch management solutions such as PatchLink and Lumension are available, often offering additional services and support for different operating systems.

© 2019 Erik Van Buggenhout & Stephen Sims

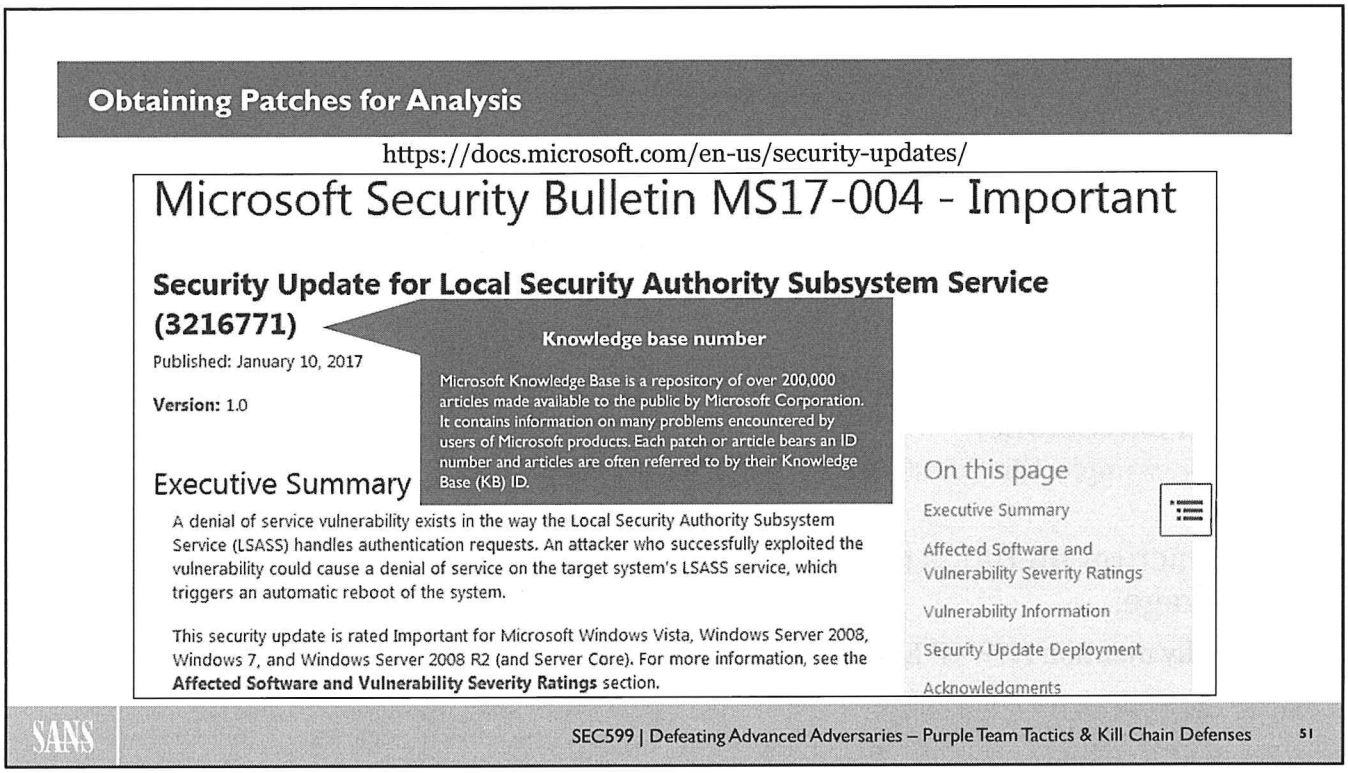


Reverse Engineering Updates

If you think about a security update, it should quickly become obvious that people might be interested in the contents of that update as it contains sensitive information to those with the right skillset. This is something that is performed by good guys, bad guys, and those in-between. Think about it from the perspective of a vendor who maintains an exploitation framework as a product, such as Immunity Canvas, Core Impact, Metasploit, or SAINT Exploit. Most disclosures are done privately and not found in the wild. This means that the vendor has been given the technical details and not the public. The vendor then creates a patch and distributes it to their customers. If you have someone with the expertise to take the patch and reverse engineer it to find the fix, that information can be used to potentially write a working exploit. Now your product has exploits available to privately disclosed vulnerabilities that are not possessed by your competitors.

If Microsoft releases patches on the second Tuesday of the month, and then someone reverse engineers the patches and quickly gets an exploit working, that exploit would be valuable to penetration testers, attackers, and security vendors. This is often referred to as a 1-day exploit since those performing the reversing are attempting to quickly locate the fix and get a working exploit built before organizations patch. The more time that goes by the less valuable the exploit as more systems are patched.

© 2019 Erik Van Buggenhout & Stephen Sims

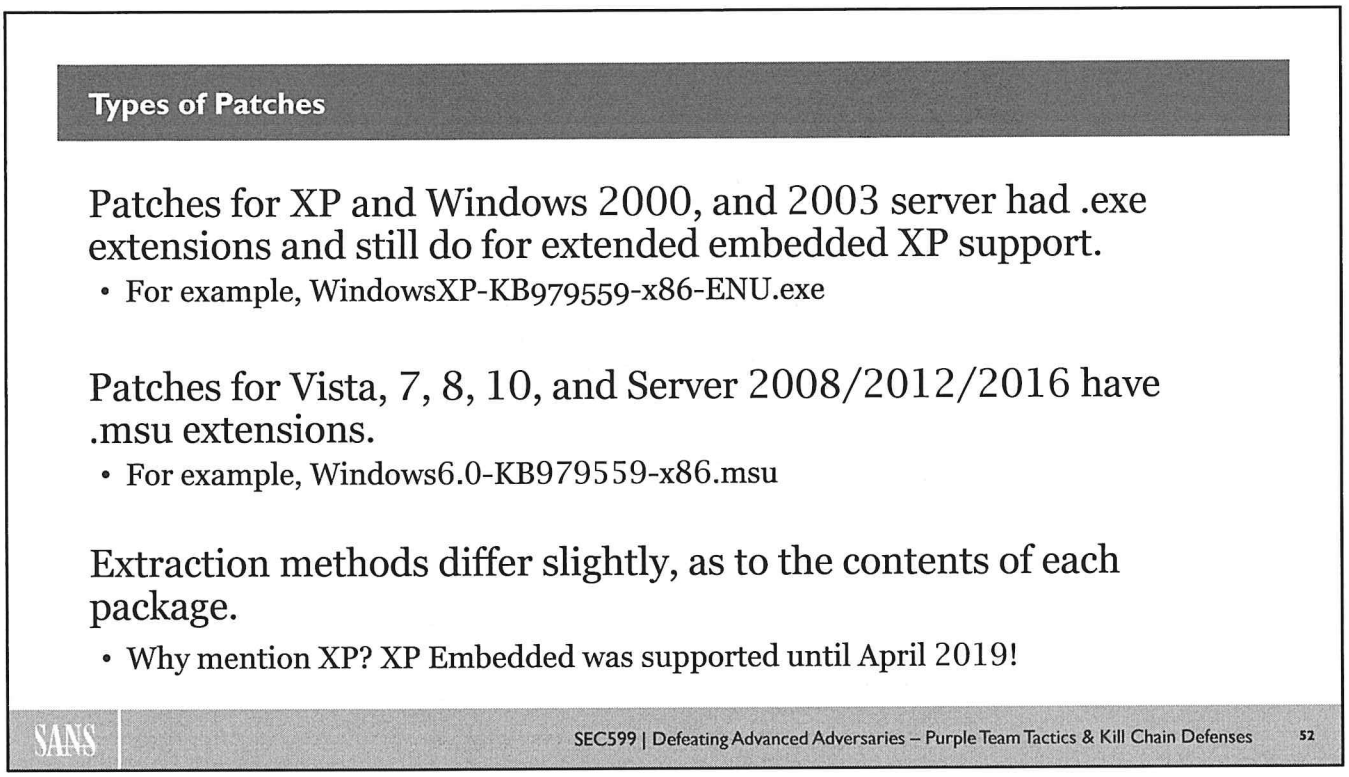


Obtaining Patches for Analysis

Microsoft TechNet provides us with the ability to directly acquire patches. Available at https://docs.microsoft.com/en-us/security-updates/, we can search for a specific update and download the appropriate patch for a given operating system. Patches are released in a couple of different formats, depending on the OS level. The cumulative updates that started in October 2016 have made the process of identifying the individual patches more difficult. They used to be in a stand-alone format and easy to extract.

Microsoft Knowledge Base is a repository of over 200,000 articles made available to the public by Microsoft Corporation. It contains information on many problems encountered by users of Microsoft products. Each patch or article bears an ID number and articles are often referred to by their Knowledge Base (KB) ID.

© 2019 Erik Van Buggenhout & Stephen Sims



Types of Patches

Most patches distributed by Microsoft have a .msu extension; however, legacy patches had a .exe extension. Patches for Windows XP, 2000 Server, and Server 2003 had the .exe extension, while Windows Server 2008/2012/2016, Windows Vista, and Windows 7/8/10 have the .msu extension. For example, a patch for a Windows XP system would look like:

WindowsXP-KB979559-x86-ENU.exe

The same patch on Server 2008 would look like:

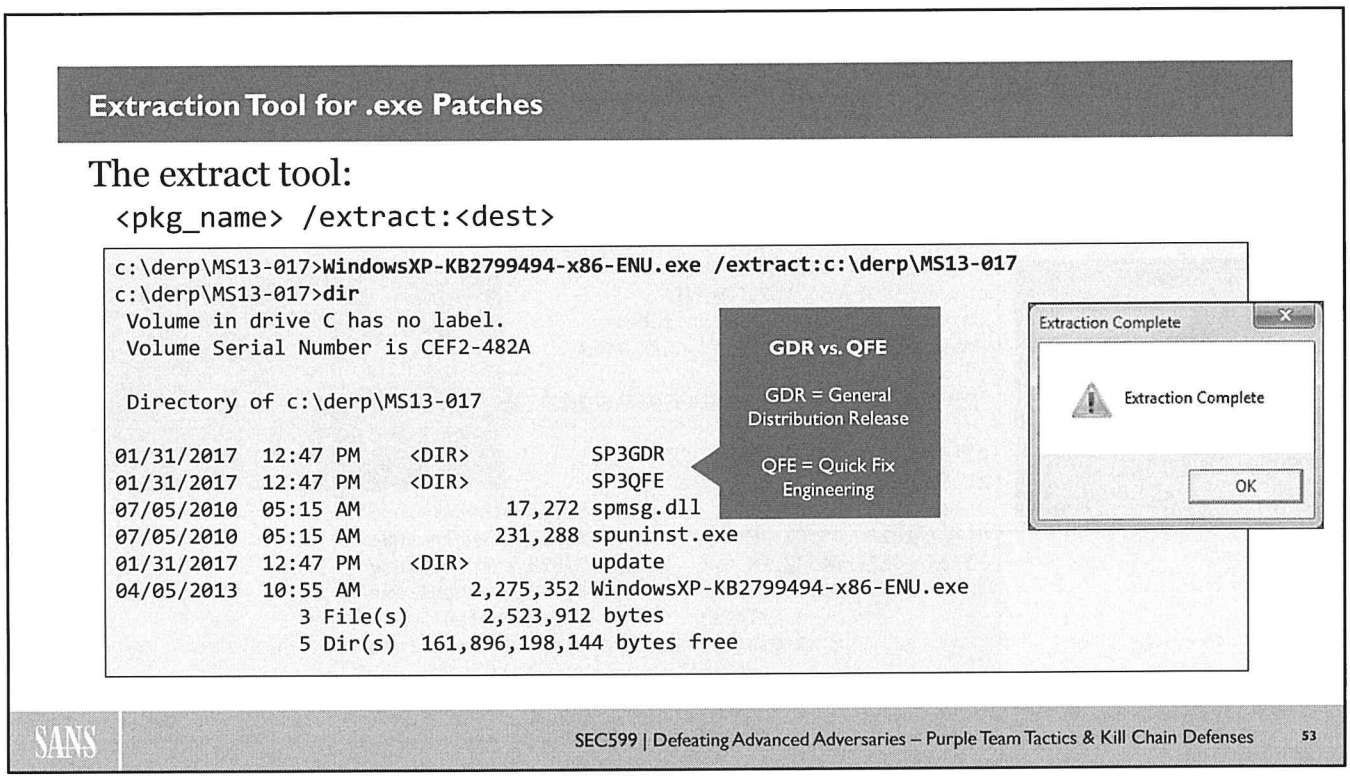
Windows6.0-KB979559-x86.msu

You may be wondering why it's worth mentioning XP. Good question. Patches for XP embedded were still available until April 9, 2019. There are people out there who acquire the embedded version patches, use a registry hack, so that they can stay on XP. See this website as an example: https://www.ghacks.net/2014/05/24/get-security-updates-windows-xp-april-2019/

From the perspective of reverse engineering patches, XP is also still of interest. Many vulnerabilities affect many or all versions of Windows. You would have to imagine that the code on XP is likely less complex than Windows 10 and therefore, reversing the same vulnerability on XP could save time.

Contents within the patch files differ depending on the OS, as do the tools to extract them manually. The .exe patch files tend to be much simpler to get to the wanted files, whereas the .msu patch files may require additional examination, especially with cumulative updates.

© 2019 Erik Van Buggenhout & Stephen Sims



Extraction Tool for .exe Patches

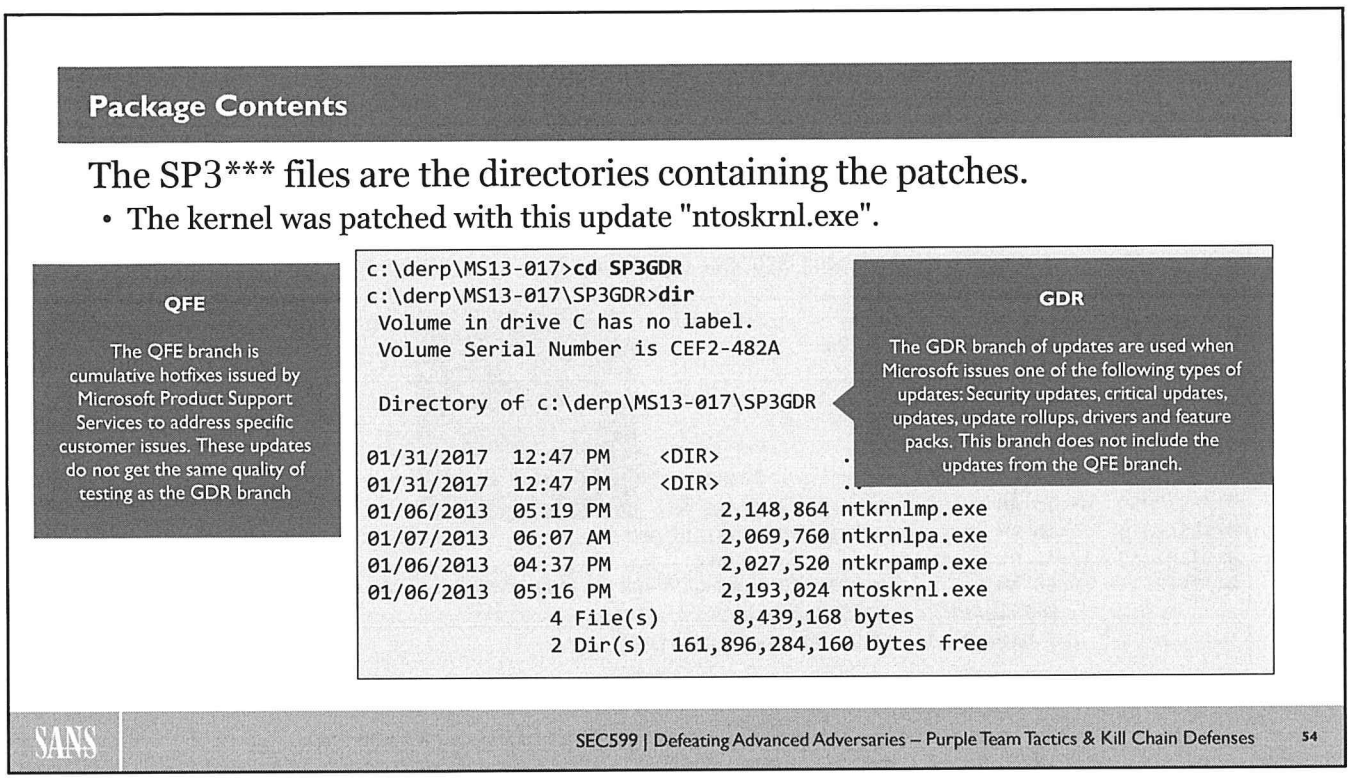
The extract tool can be used via the command line to extract patches with the .exe extension. Simply type in the name of the patch file containing the .exe extension, followed by /extract:<dest>. For example:

C:\derp/MS13-017> **WindowsXP-KB2799494-x86-ENU.exe /extract:c:\derp\MS13-017**

If successful, you get the pop-up box on the screen stating that extraction was successfully completed. Proceed to review the contents of the package.

You may have noticed that there are two folders: One with GDR in the title and the other with QFE. GDR stands for General Distribution Release and QFE stands for Quick Fix Engineering.

© 2019 Erik Van Buggenhout & Stephen Sims



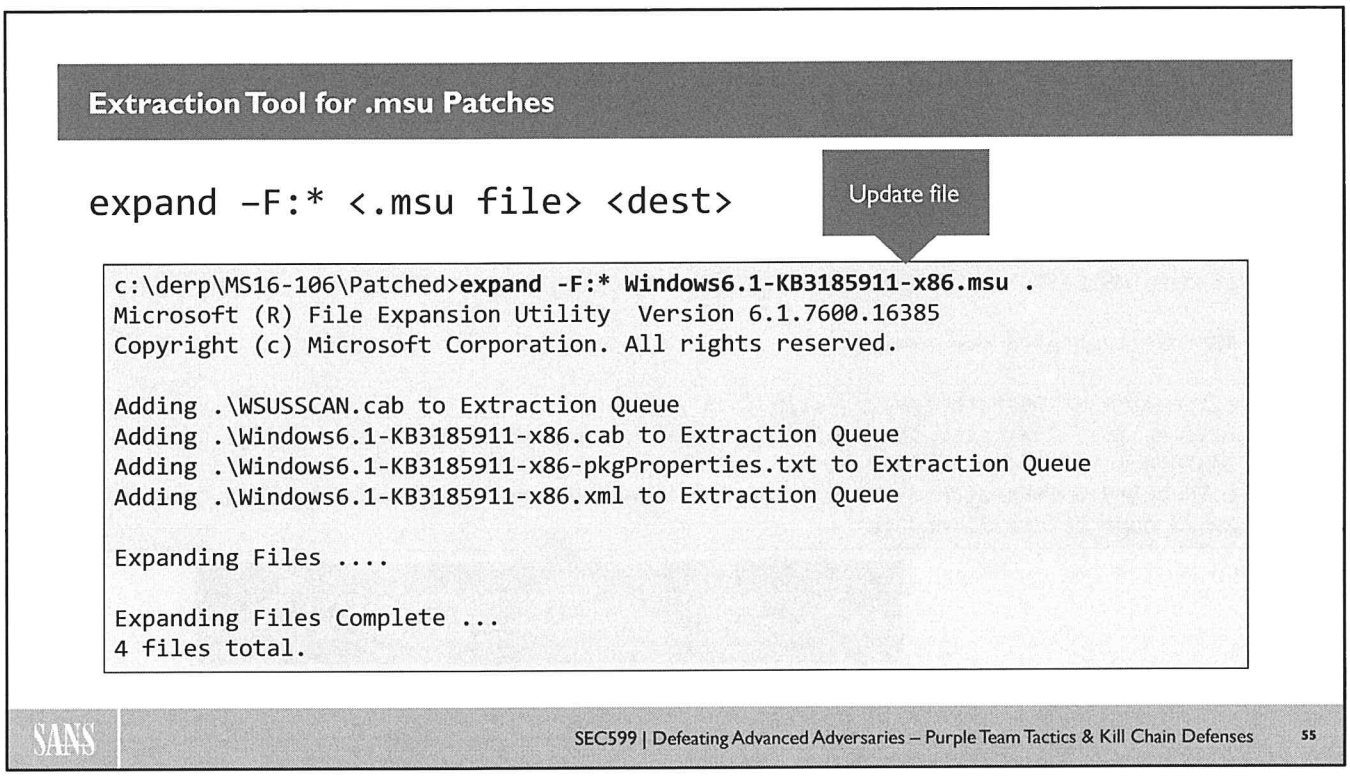
Package Contents
The package contents of this update are shown in the screenshot. As you saw on the last slide, there are two directories listed for XP SP3 called SP3GDR and SP3QFE. The contents of the directory SP3GDR contains multiple files, such as "ntoskrnl.exe." This is actually the name of the Windows Kernel and therefore the Kernel was patched in this fix. Command switches were used to limit the output to fit the image onto the slide.

"The GDR branch of updates are used when Microsoft issues one of the following types of updates: Security updates, critical updates, updates, update rollups, drivers, and feature packs. This branch does not include the updates from the QFE branch.

The QFE branch is cumulative hotfixes issued by Microsoft Product Support Services to address specific customer issues. These updates do not get the same quality of testing as the GDR branch."

Citation:
Jphillips59. "QFE vs. GDR." QFE vs. GDR Microsoft (Windows) Support – Neowin Forums. https://www.neowin.net/forum/topic/332694-qfe-vs-gdr/ (accessed January 31, 2017).

© 2019 Erik Van Buggenhout & Stephen Sims



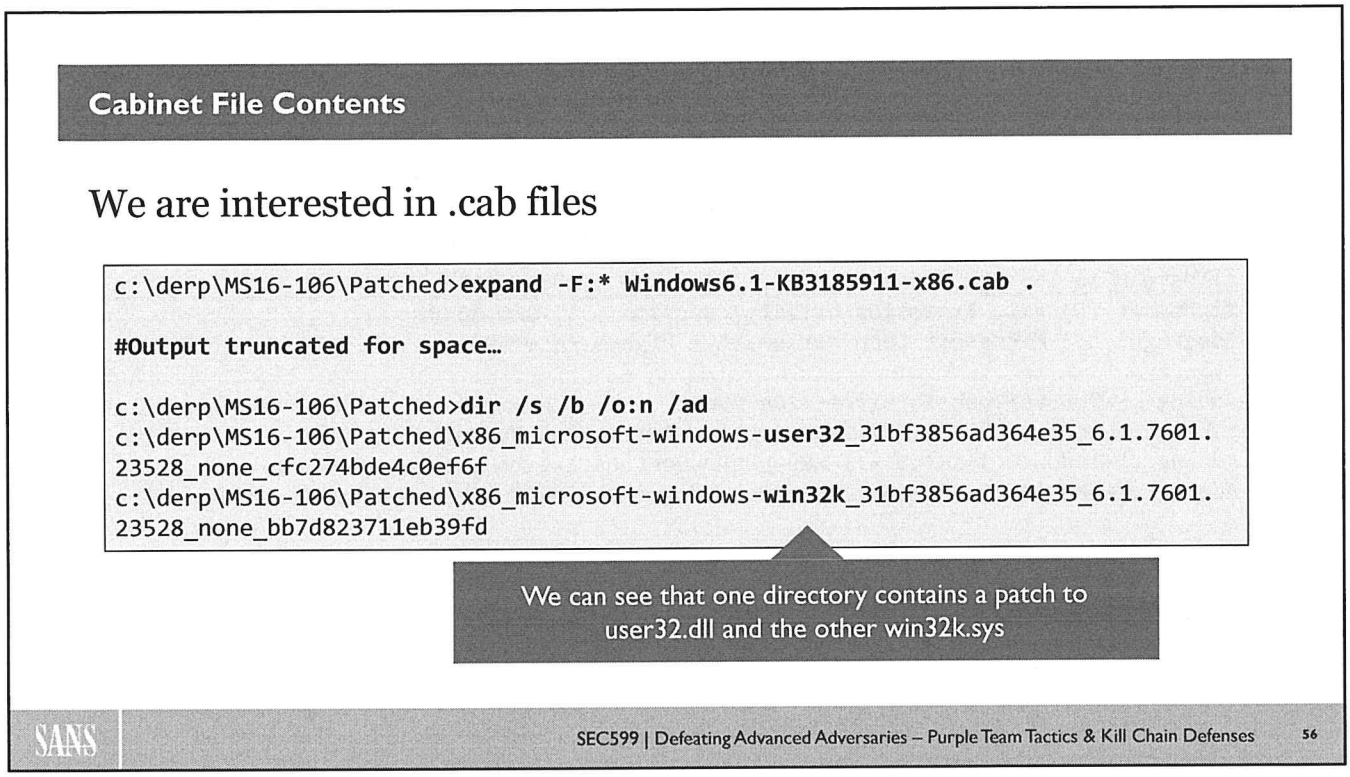
Extraction Tool for .msu Patches

For Windows Vista, 7, 8, 10 and Server 2008/2012/2016, the expanded tool can unpack packages with the .msu extension. As shown on the slide, the file Windows6.1-KB3185911-x86.msu is expanded with the following command:

expand –F:* Windows6.1-KB3185911-x86.msu .

Four files are unpacked and can be seen.

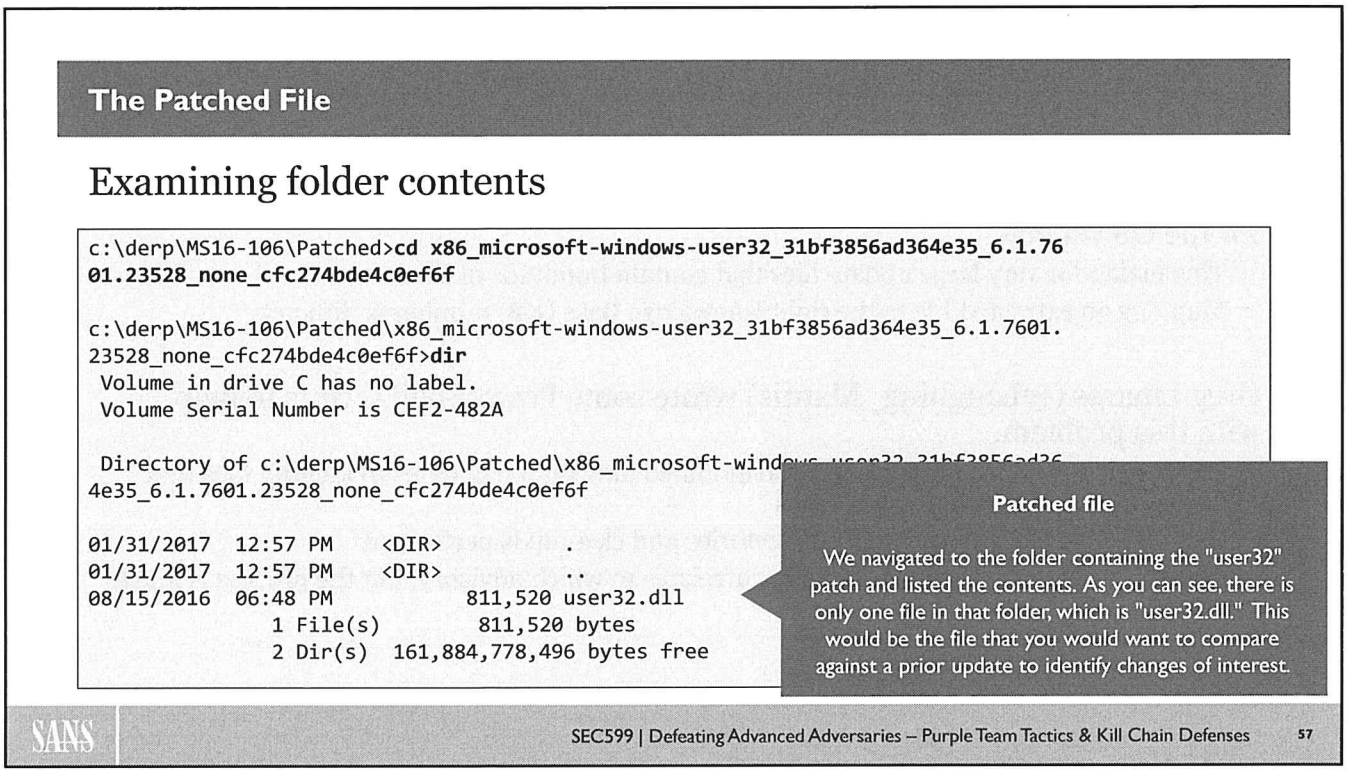
© 2019 Erik Van Buggenhout & Stephen Sims



Cabinet File Contents

As seen on the prior slide, several files were extracted from the .msu file. We must now use the same method to extract the .cab file. A lot of output is displayed on the screen when extracting the .cab file and as such, it was truncated from the output on the slide for spacing purposes. A customized "dir" command is then issued to limit output to directories only. You can see there are two folders, one containing a reference to the name "user32" and the other "win32k."

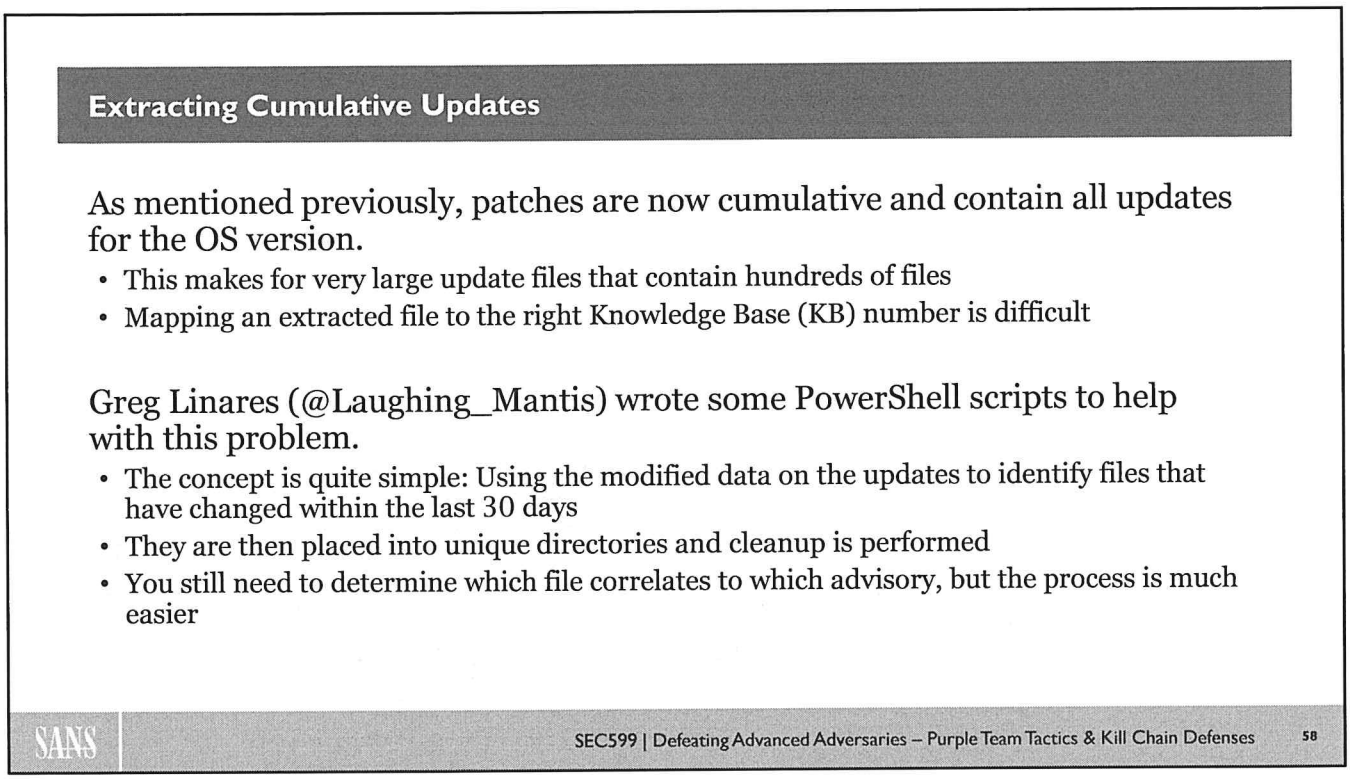
© 2019 Erik Van Buggenhout & Stephen Sims



The Patched File

We have now simply navigated to the folder containing the "user32" patch and listed the contents. As you can see, there is only one file in that folder, which is "user32.dll." This would be the file that you would want to compare against a prior update to identify changes of interest. More on this shortly.

© 2019 Erik Van Buggenhout & Stephen Sims



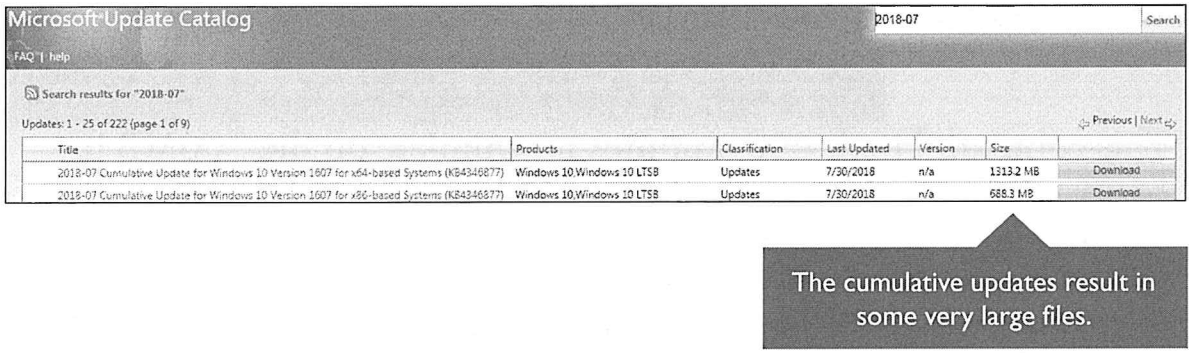
Extracting Cumulative Updates

As previously mentioned, the cumulative updates are very large and contain all patches for the OS version. When extracting them, there are hundreds to thousands of files. This makes it very challenging to sort through them to find the desired file and map it correctly to the Knowledge Base (KB) number. Greg Linares, known as @Laughing_Mantis on Twitter, created a couple of PowerShell scripts to help with this issue (you can find an example here: https://pastebin.com/0mYXJGC5). The idea is quite simple, extract everything, delete all the junk we do not care about, and sort the files over 30 days old into an "old" directory. This allows you to focus on the files that have data modified within the past 30 days. You still need to map the files to the correct KB number, but now you are only looking at 10 or so folders as opposed to a very large amount.

© 2019 Erik Van Buggenhout & Stephen Sims

Obtaining a Cumulative Update for Windows 10

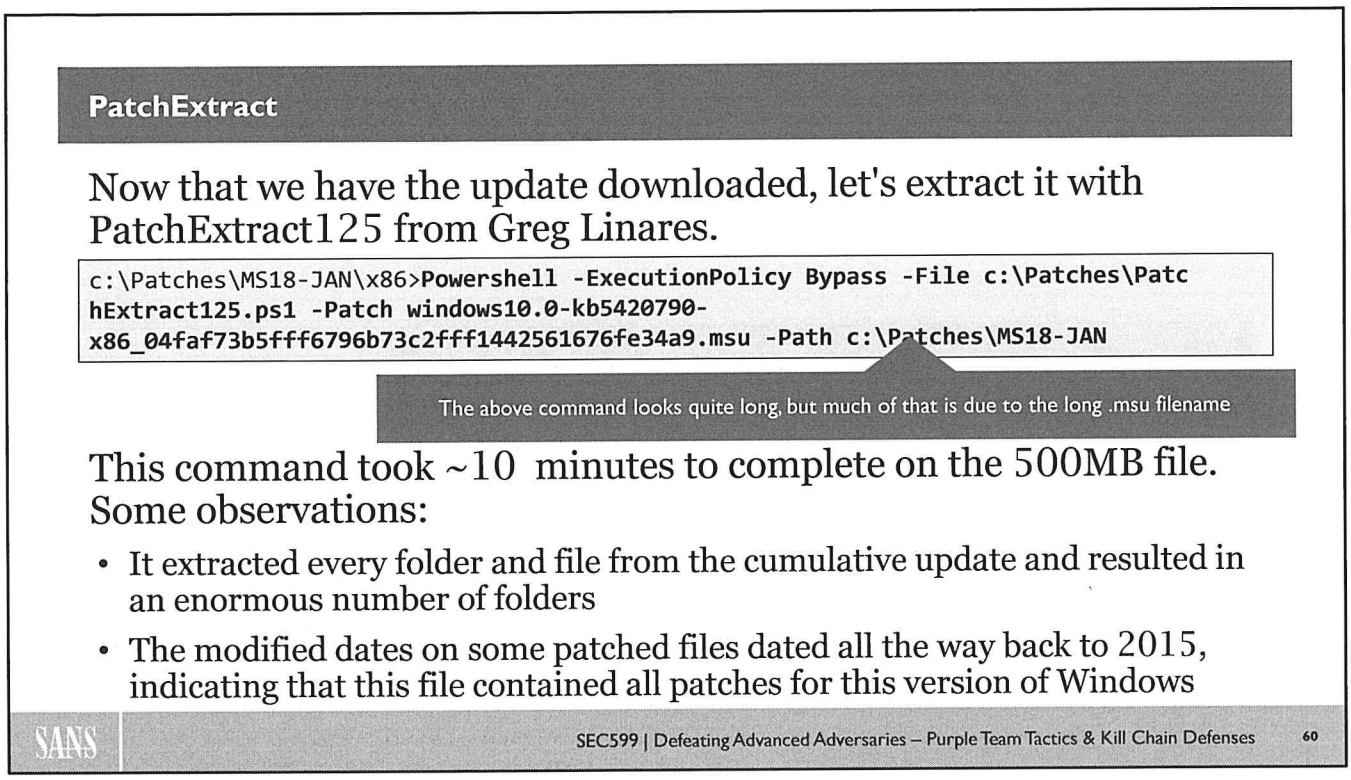
The following screenshot shows the cumulative update file for July 2018:



Obtaining a Cumulative Update for Windows 10

This slide simply shows a screenshot of the July 2018 cumulative update for Windows 10. As you can see, the x86 file is 688MB and the x64 file is over 1GB. Changes in the way Microsoft is managing incremental updates to reduce the file size may allow for this to be smaller for patching: https://www.theverge.com/2016/11/3/13511012/microsoft-windows-10-unified-update-platform-features

© 2019 Erik Van Buggenhout & Stephen Sims



PatchExtract

With the update downloaded, let's extract it with the PatchExtract tool from Greg Linares.

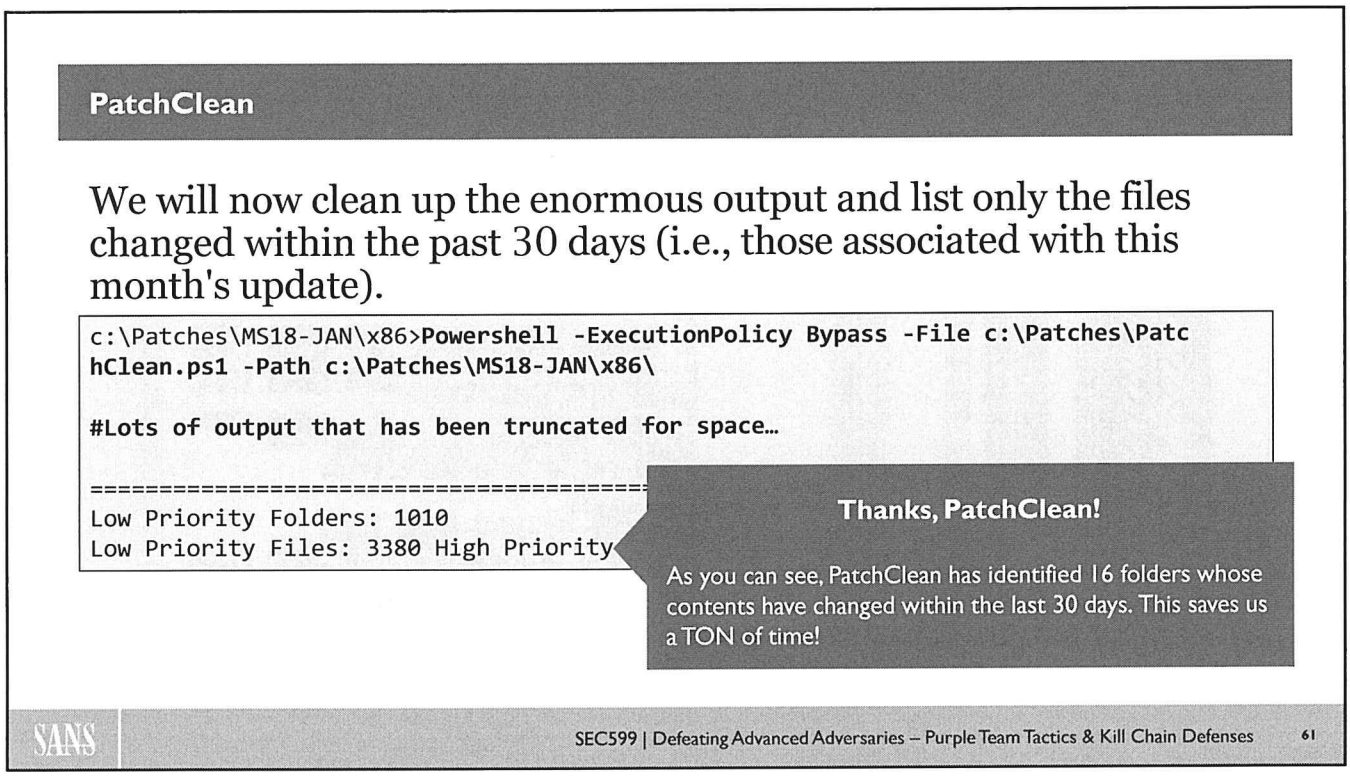
```
C:\Patches\MS18-JAN\x86>Powershell -ExecutionPolicy Bypass -File c:\Patches\Patc
hExtract125.ps1 -Patch windows10.0-kb5420790-
x86_04faf73b5fff6796b73c2fff1442561676fe34a9.msu -Path c:\Patches\MS18-JAN
```

The command is rather long due to the .msu filename; however, we're simply telling it what script to execute "PatchExtract125.ps1," then the name of the .msu file with the "-Patch" switch, and then the path where to put the extracted files with "-Path."

Depending on the size of the .msu file (500MB in this case), it can take quite a while to extract all of the files. It took ~10 minutes for this file. The result is excluded from the slide as it is quite a lot of output, as well as over a thousand files and folders. When looking at a couple of sample files the dates were all the way back into 2015, showing that the .msu file contains all patches for this version of Windows.

PatchExtract can be found at: https://pastebin.com/u/Laughing_Mantis

© 2019 Erik Van Buggenhout & Stephen Sims



PatchClean

With all of the files and folders from the cumulative update extracted, we want to know which ones are associated with this month's update. The PatchClean script will go through and put every folder that contains a folder with a "Date Modified" time of >30 days into a folder called "Old." It will leave only folders with files in them that have a "Date Modified" time within the last 30 days. We run PatchExtract with:

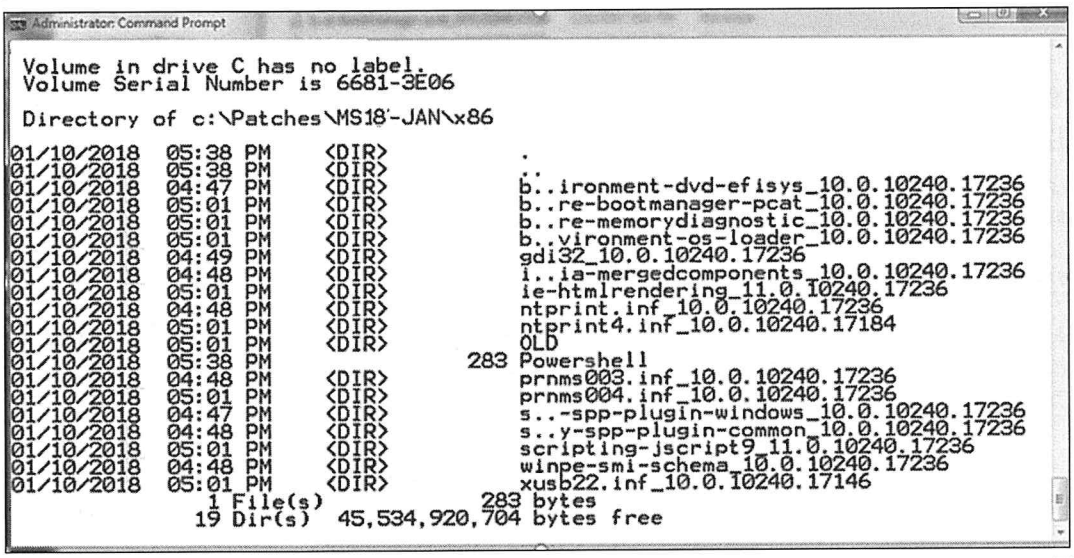
c:\Patches\MS18-JAN\x86>**Powershell -ExecutionPolicy Bypass -File c:\Patches\PatchClean.ps1 -Path c:\Patches\MS18-JAN\x86**

The result, as shown on the slide, is 16 high priority folders. That is much less than the >1,000 folders and >3,800 files extracted from the cumulative update.

PatchClean is also available at: https://pastebin.com/u/Laughing_Mantis

© 2019 Erik Van Buggenhout & Stephen Sims

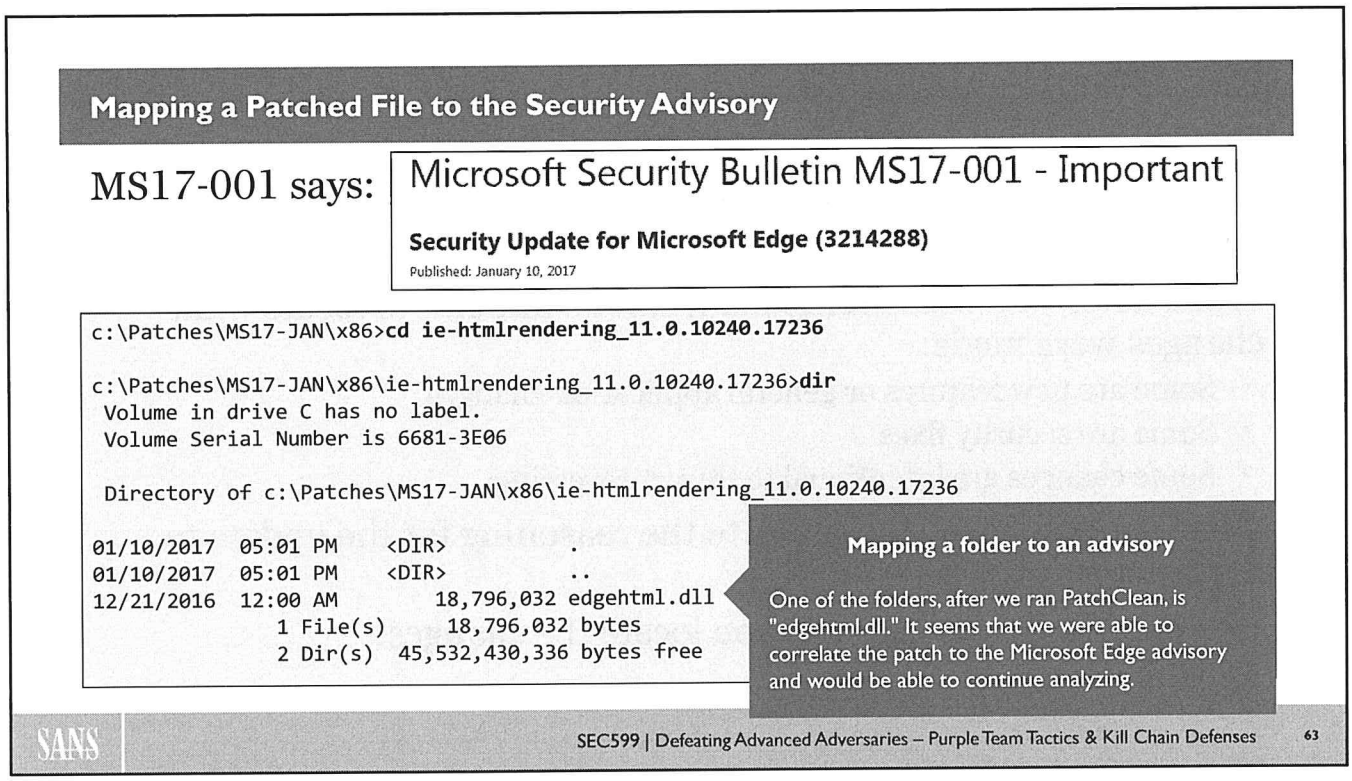
Patch Extraction Results



Patch Extraction Results

This slide simply shows the results in the remaining folders. These should be easily mappable to security advisories on the Microsoft website. Let's try to do one on the next slide.

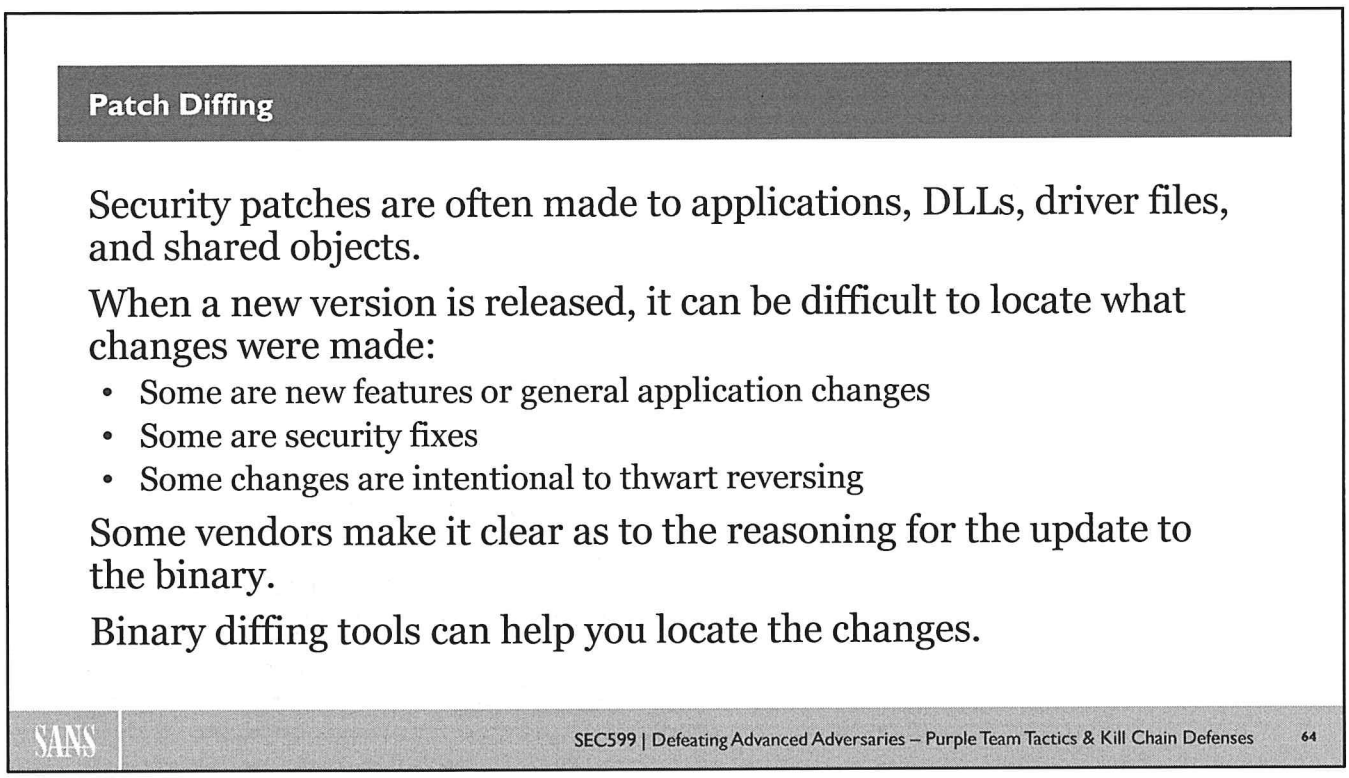
© 2019 Erik Van Buggenhout & Stephen Sims



Mapping a Patched File to the Security Advisory

When looking at MS17-001 we see that the security bulletin applies to the Microsoft Edge browser. One of the folders, after we ran PatchClean, is "edgehtml.dll." It seems that we were easily able to correlate the patch to the advisory and would be able to continue analyzing.

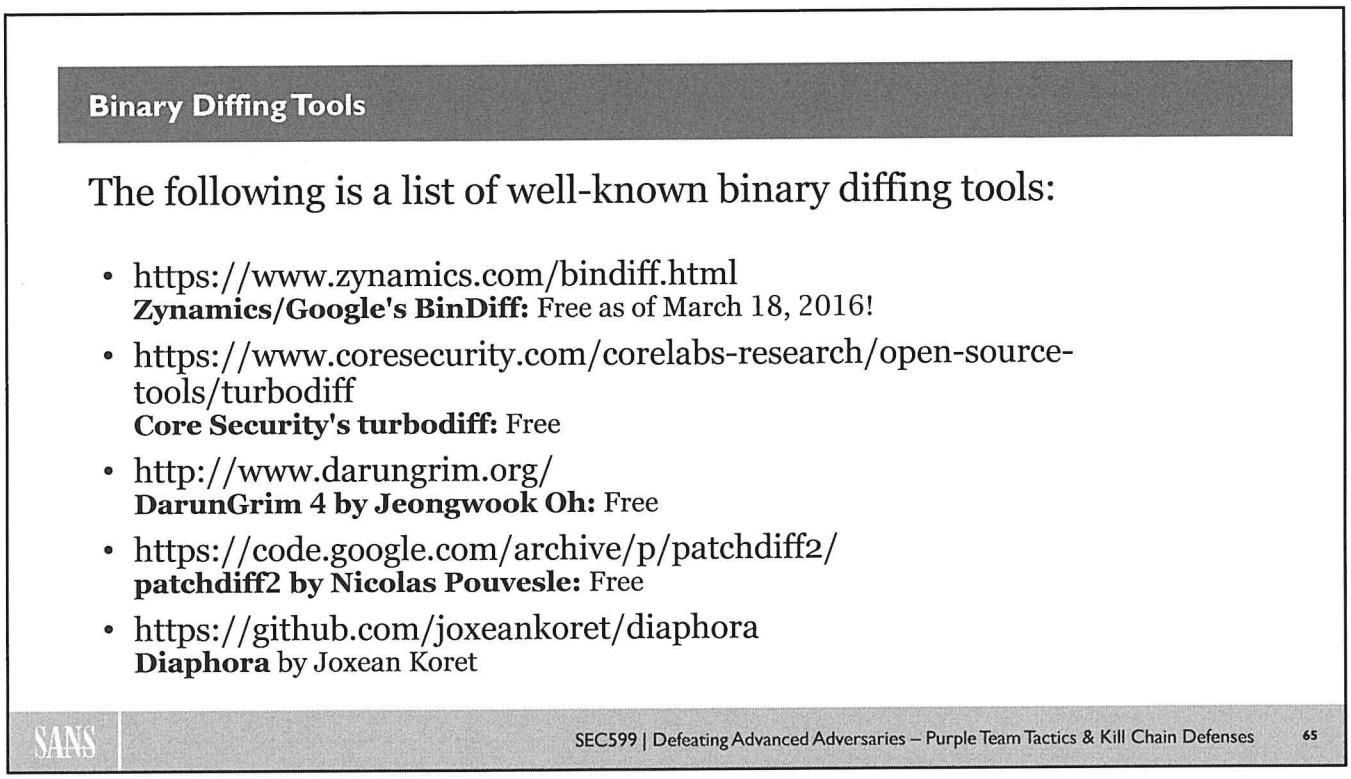
© 2019 Erik Van Buggenhout & Stephen Sims



Patch Diffing

As we are all aware, new versions of applications come out all the time, as do patches to existing DLLs, drivers, and shared objects. Some of these changes are simply new features rolled out or fixes to performance problems. Other changes are vulnerability patches that are certainly of interest. If someone can take the unpatched version of a binary and diff it against the patched version, the code changes may become visible, shining a light on an otherwise unknown vulnerability. Those systems that are properly patched would be safe, leaving anyone who has not patched their system exposed to a potential 1-day exploit. Some vendors make it clear as to the reasoning behind an update, whereas others attempt to hide their intentions. Either way, binary diffing tools can often help us locate code changes that could potentially reveal the patched vulnerability. This is a lucrative practice because many organizations do not patch their systems quickly.

© 2019 Erik Van Buggenhout & Stephen Sims

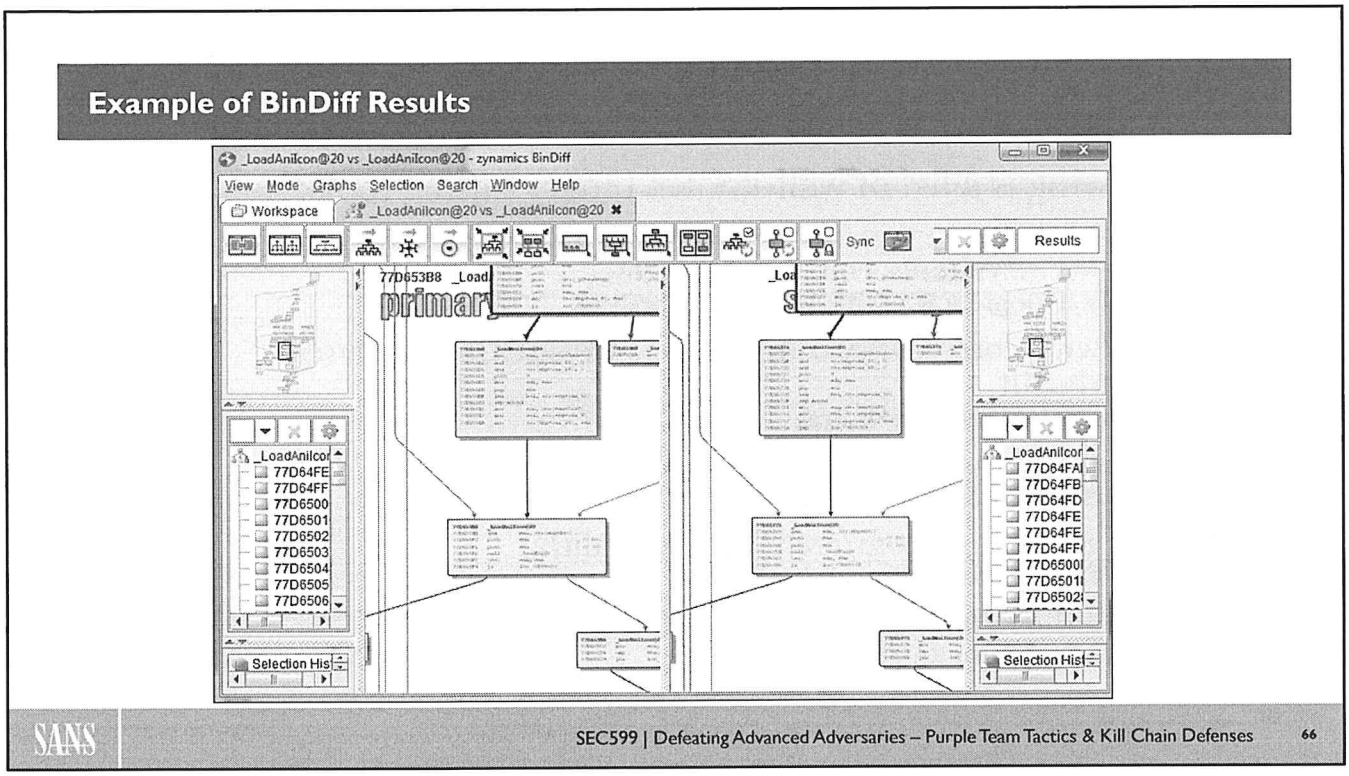


Binary Diffing Tools

There a few well-known binary diffing tools, most of them free; although many have specific dependencies on versions of IDA.

- **BinDiff**: Created by Zynamics, acquired by Google in 2011 – https://www.zynamics.com/bindiff.html
- **Turbodiff**: Created by Core Security – https://www.coresecurity.com/corelabs-research/open-source-tools/turbodiff
- **DarunGrim 4**: Written by Jeongwook Oh – http://www.darungrim.org/
- **Patchdiff2**: Written by Nicolas Pouvesle – https://code.google.com/archive/p/patchdiff2/
- **Diaphora:** Written by Joxean Koret – https://github.com/joxeankoret/diaphora

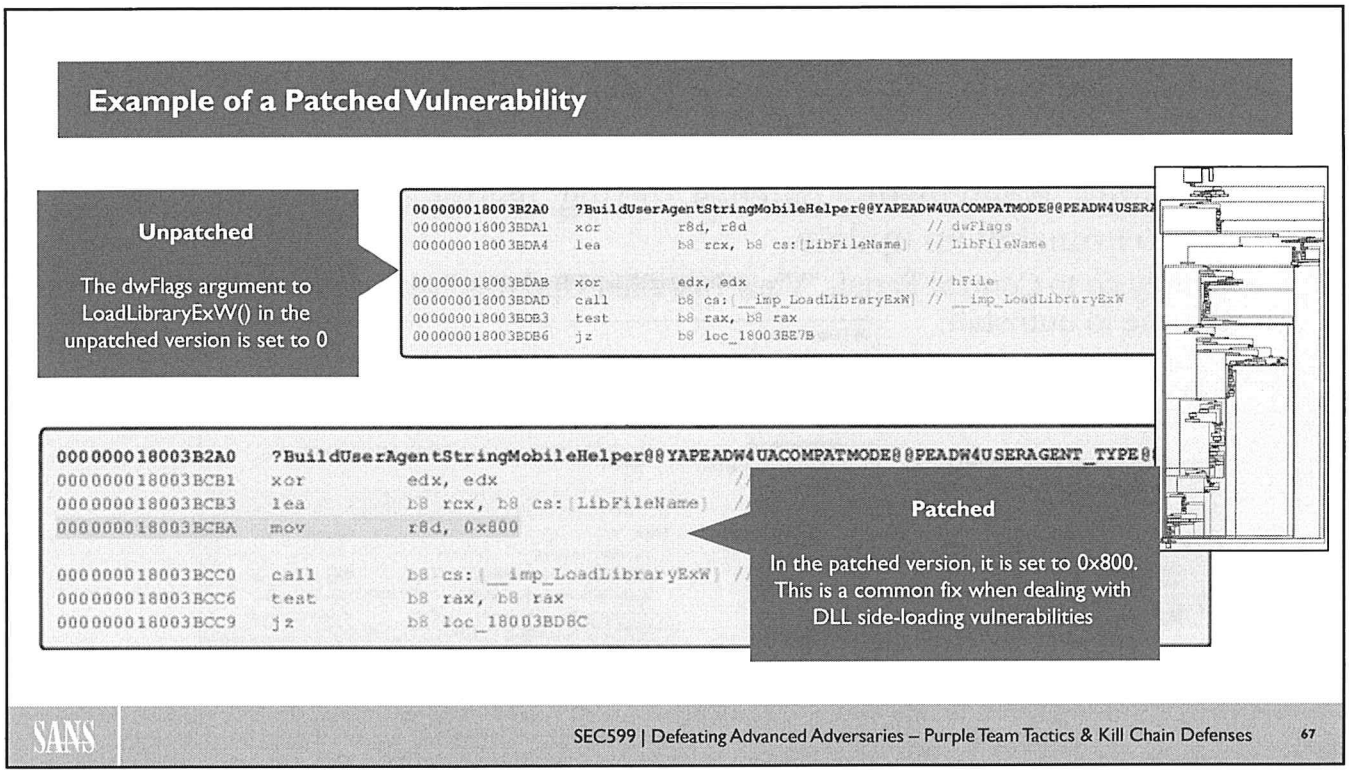
© 2019 Erik Van Buggenhout & Stephen Sims



Example of BinDiff Results

What you see on this slide is the visual diff of two versions of the same file. On the right, would be the patched version and on the left, is the version from the last time it was patched. When comparing these two versions together, the only changes should be related to the most recent security fix. Some DLLs and other binaries might have over 20,000 functions. Using these diffing tools, we can greatly reduce the number of files we would have to look at when trying to identify modified code in relation to a vulnerability.

© 2019 Erik Van Buggenhout & Stephen Sims



Example of a Patched Vulnerability

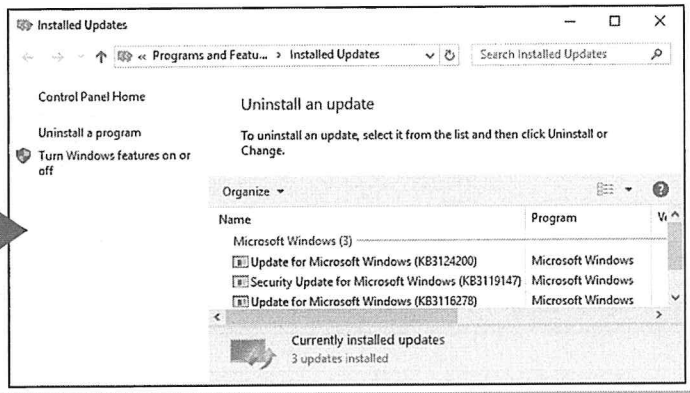
When running a Visual Diff and analyzing the changes, the block shown on this slide shows the issue. The dwFlags argument to LoadLibraryExW() in the unpatched version is set to 0, while in the patched version, it is set to 0x800. This is a common fix when dealing with DLL side-loading vulnerabilities. A dwFlags argument of 0 results in the potential loading of DLLs from outside of safe locations. A dwFlags option of 0x800 states that DLLs may only be loaded from the "%SystemRoot%\Windows\System32" folder.

© 2019 Erik Van Buggenhout & Stephen Sims

Uninstalling a Patch

Sometimes, when testing patches, diffing, testing exploits, etc., you need to uninstall an update.

- Simply go to Control Panel, "View Installed Updates," and double-click on the one to uninstall:



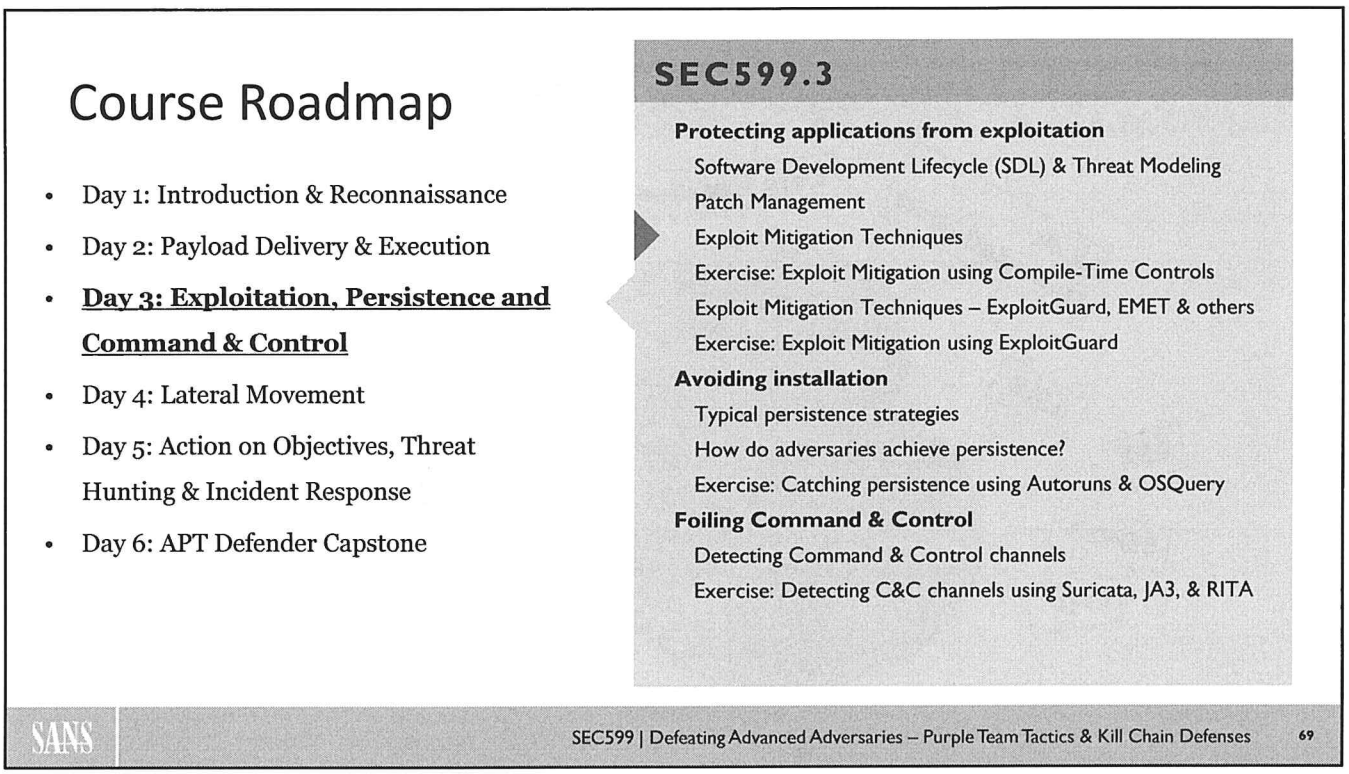
This is an example from a Windows 10 base build with minimal updates, hence the low number listed.

SANS SEC599 | Defeating Advanced Adversaries – Purple Team Tactics & Kill Chain Defenses 68

Uninstalling a Patch

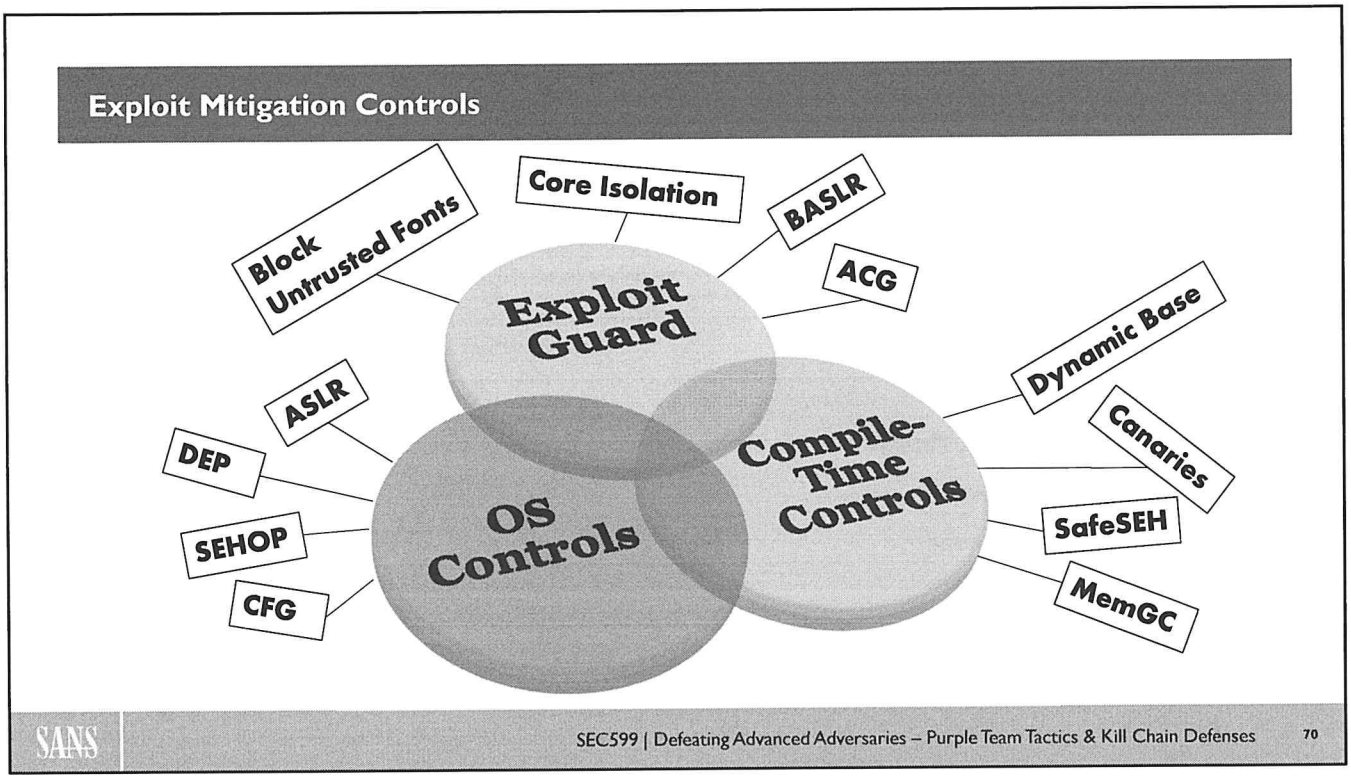
Sometimes, a patch was already applied to a system you want to test, or you may want to uninstall an update for any number of reasons. The process is simple because Windows archives the old versions of patched DLLs and other files. Simply go to your control panel and type "View Installed Updates" into the search field. A box with the installed updates will appear as shown on the slide. When you find the update you want to uninstall, double-click it, and you will be asked if you are sure you want to uninstall this update. This is an example from a base build with minimal updates, hence the low number listed.

© 2019 Erik Van Buggenhout & Stephen Sims



This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

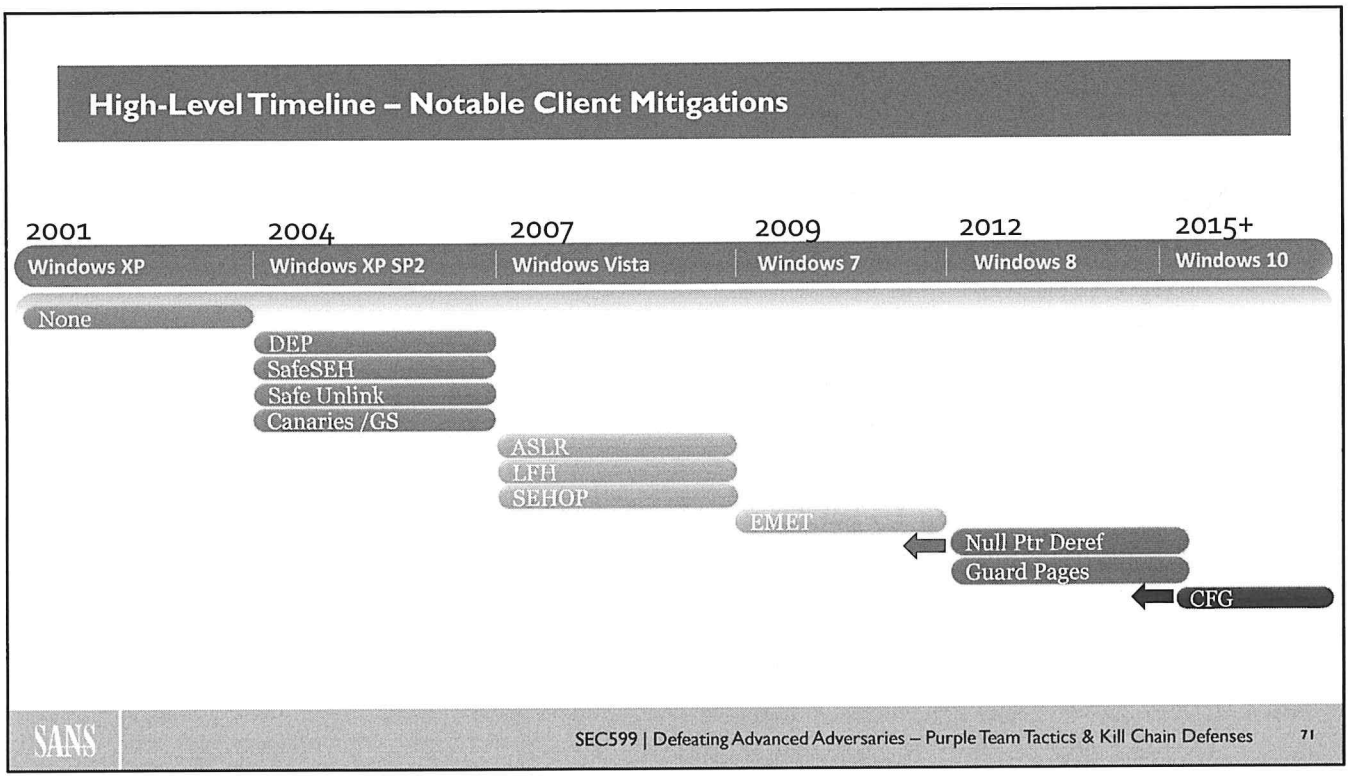


Exploit Mitigation Controls

First, let's briefly discuss the role of exploit mitigations. We are all aware of the concept of "Defense in Depth." The idea is that any one control may fail, so we want as many as possible without impacting application or system performance too significantly. If we only utilize a single control such as Data Execution Prevention (DEP) and an attacker figures out a way to disable it, then there is nothing left protecting the application or system from compromise. By layering on various controls, it can stop or at least greatly increase the difficulty to achieve exploitation.

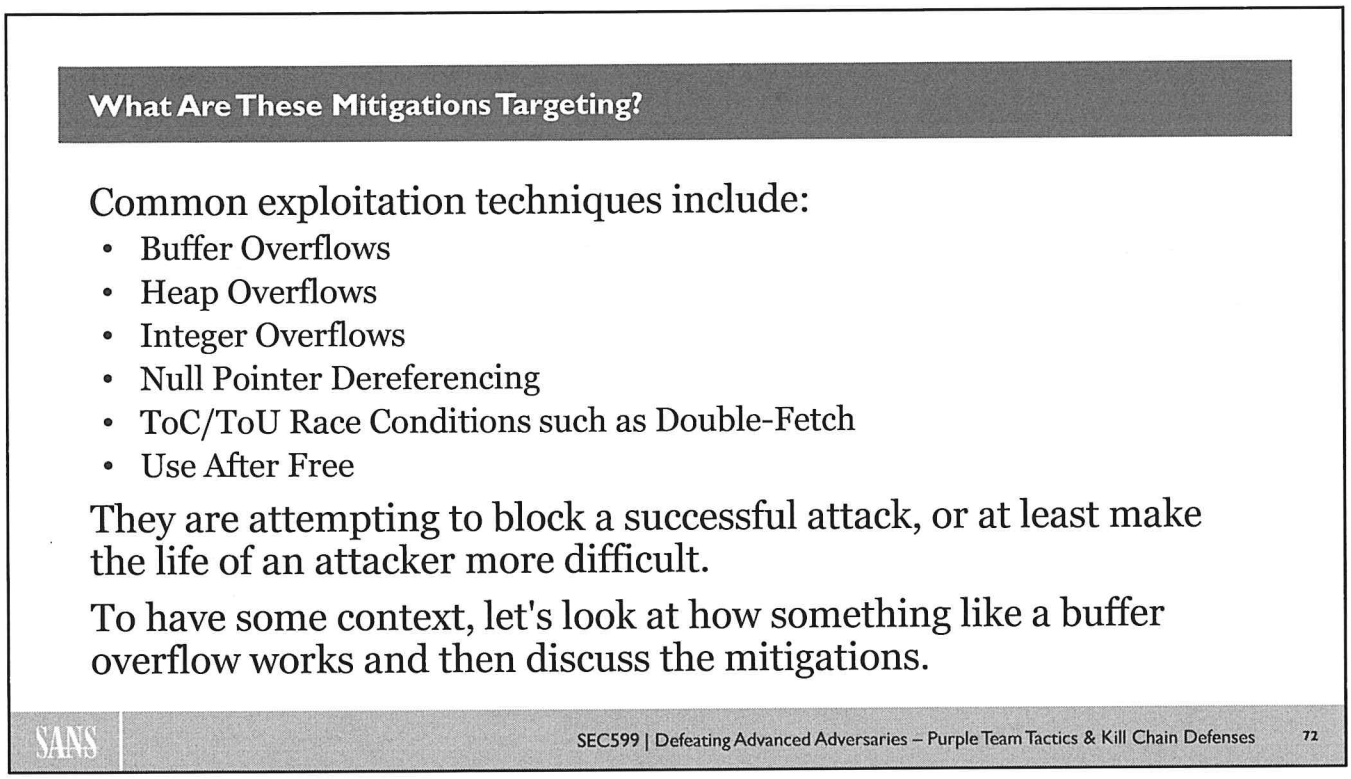
The basic Venn diagram on the slide shows two categories of exploit mitigation: "OS Controls" and "Compile-Time Controls." OS controls include protections such as Address Space Layout Randomization (ASLR), DEP, Structured Exception Handling Overwrite Protection (SEHOP), and Control Flow Guard (CFG). The operation system must support these controls, and sometimes even the hardware. Each OS is different, but they are typically designed to be controls that cannot be turned off by an application. They are system enforced. Compile-time controls are exactly how they sound; they are controls that are added during compile time. These often insert code or metadata into the program. Examples include stack and heap canaries, MemGC, SafeSEH, and Dynamic Base. We will discuss a sample of the most prominent controls in this module. As we increase the number of controls and move into the merged areas of the circles, our protection should increase.

© 2019 Erik Van Buggenhout & Stephen Sims



High-Level Timeline – Notable Client Mitigations
This slide shows a high-level timeline of exploit mitigations added or made available over the years. This is not comprehensive by any means, and we will address many of them in this module.

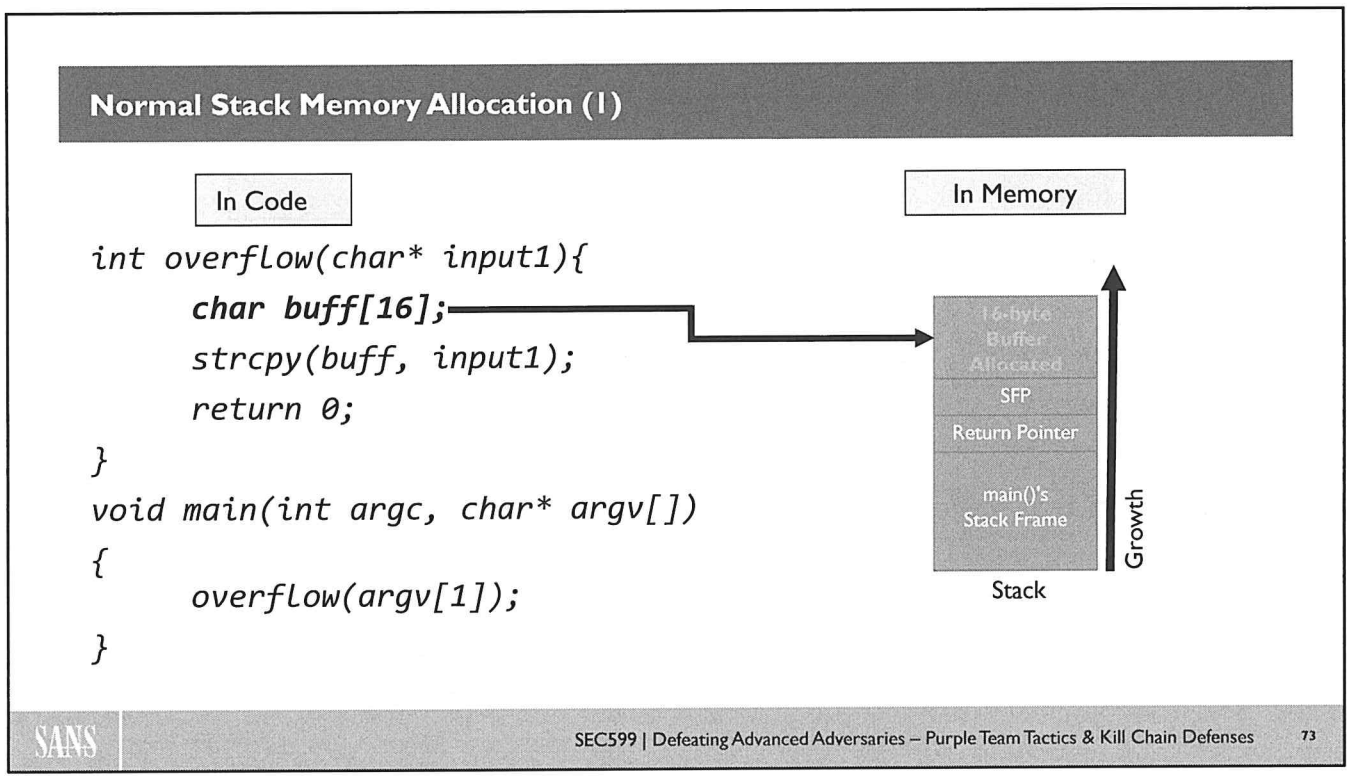
© 2019 Erik Van Buggenhout & Stephen Sims



What Are These Mitigations Targeting?

There are many different bug classes, each varying in difficulty in relation to discovery, exploitability, reliability, etc.... Some common bug classes related to low-level languages such as C and C++ include buffer overflows, heap overflows, integer overflows, null pointer dereferencing, race conditions, and use after free. The exploit mitigations that we are discussing focus their efforts on preventing the exploitability of these vulnerabilities. The majority of operating systems and large applications are written in C and C++, as well as assembly, and Objective-C. The main benefits of using low-level languages is their ability to directly access memory, processor registers, and other low-level functionality. This access also allows for costly mistakes. Let's take a look at a basic buffer overflow that will provide context when discussing the mitigations.

© 2019 Erik Van Buggenhout & Stephen Sims

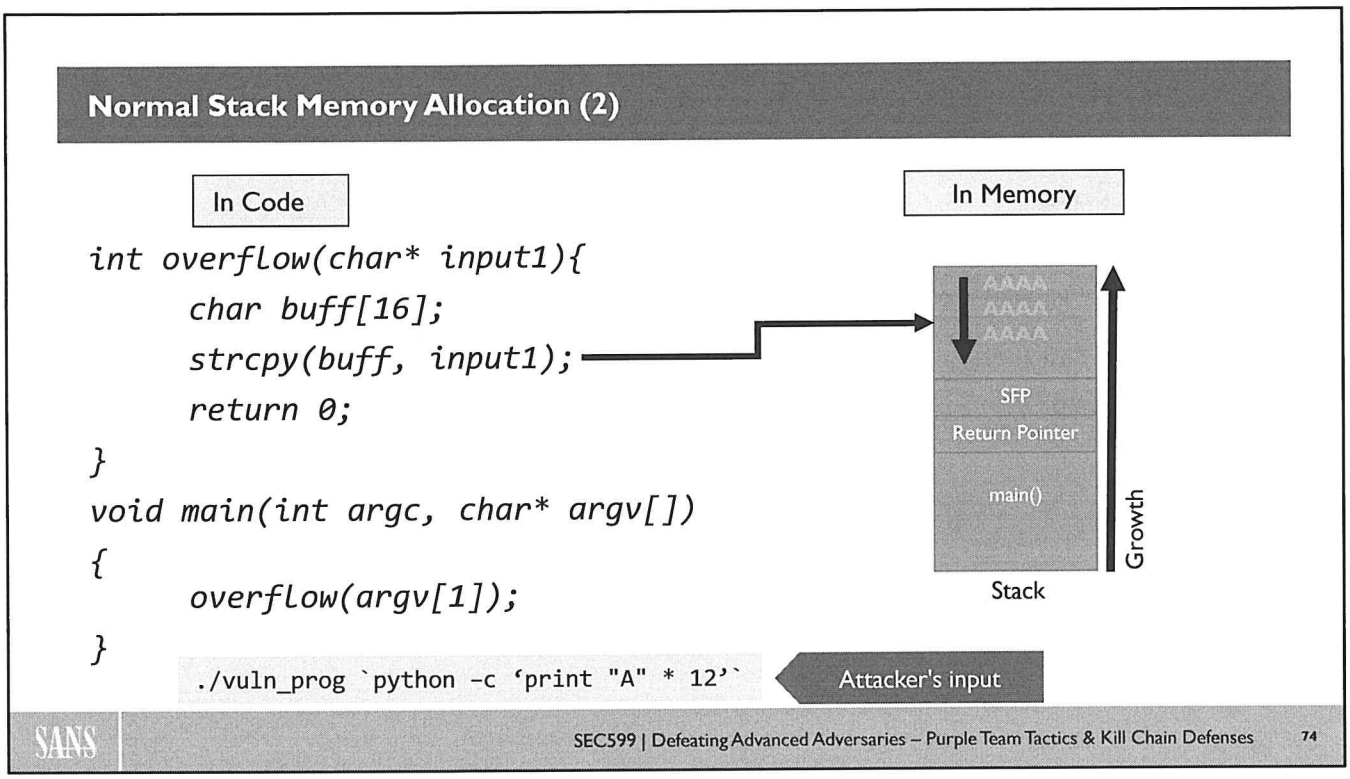


Normal Stack Memory Allocation (1)
On the left of this slide is the C source code:

```
int overflow(char* input1){                    // This is a function called overflow() which contains a
buffer overflow
   char buff[16];                  // Allocating a 16-byte buffer called buff
   strcpy(buff, input1);                       // The vulnerable strcpy() function copying user-supplied
data to the buffer
   return 0;                                   // Returning a status of 0 if all is find
}
void main(int argc, char* argv[])  // This is the main() function with which all C programs begin.
{
    overflow(argv[1]);                         // The main() function calls the vulnerable overflow()
function, passing stdin
}
```

On the right is the stack of the process once the program is compiled and run. The stack is a region in memory that stores data associated with function calls. Each function called gets its own allocation of memory on the stack called a stack frame. The stack grows from high memory towards low memory. If function A() calls function B(), then function B()'s stack frame will be built on top of function A()'s. As the code in a function finishes, control is passed back to the calling function via the return pointer. In the image on the right, you can see that main()'s stack frame is at the "bottom of the stack", on top of which stack frames for other functions (e.g. overflow) are built.

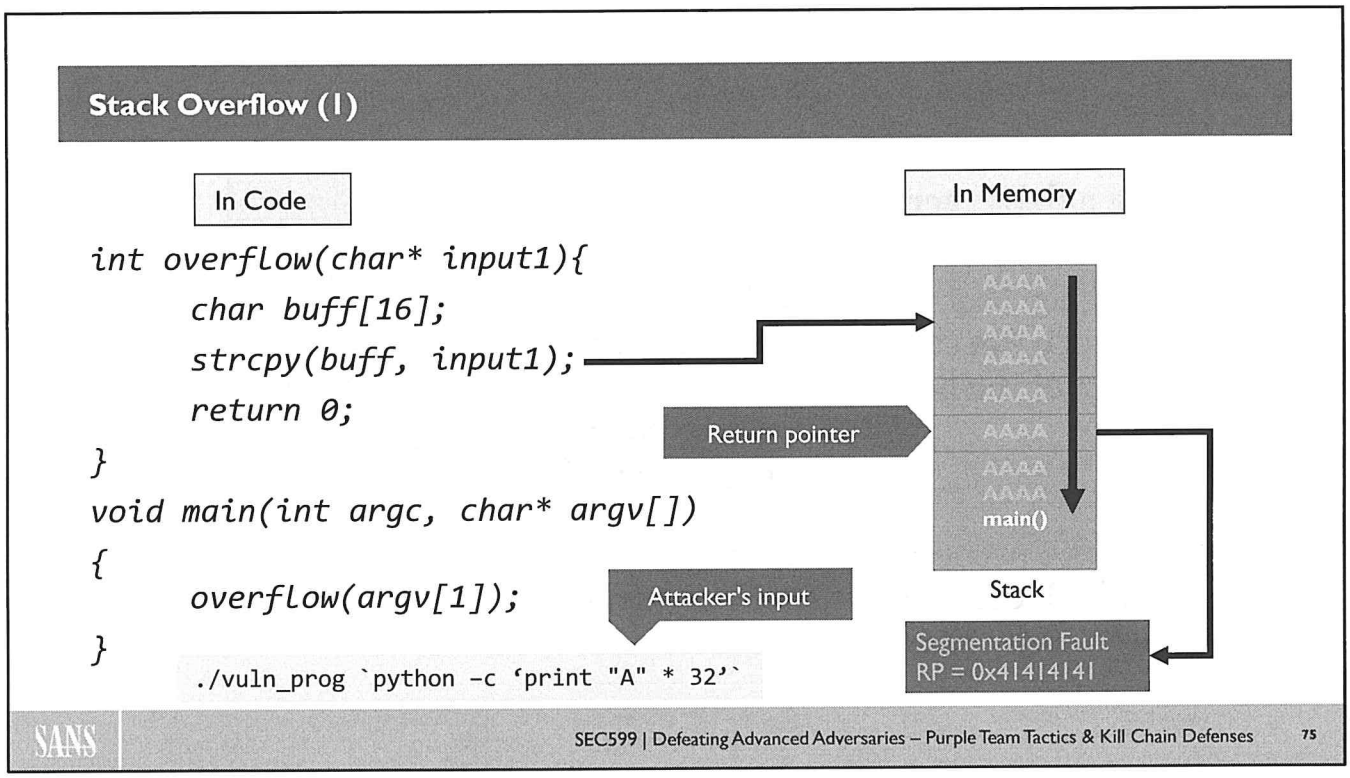
© 2019 Erik Van Buggenhout & Stephen Sims



Normal Stack Memory Allocation (2)

On this slide, you can see that the attacker is using Python to send 12 A's as input into the program. This input goes into the program via standard-in (stdin). "Argv" is the argument vector. At argv[0] is the name of the program and argv[1] would be the first argument passed by the attacker, which is the 12 A's. If a second argument was passed, it would be reachable at argv[2]. Again, it is simply the argument vector. In the main() function, the overflow() function is called and passed argv[1] as an argument. The overflow function takes a pointer "char* input1" to the argument and names it input1. The 16-byte buffer is then allocated, and then strcpy() is called, copying the 12 A's into the 16-byte buffer. The problem with strcpy() is that it does not provide any bounds checking. In other words, there is no size argument to limit the amount of data copied into the allocated buffer. So, if the attacker passed in 100 A's, strcpy() would happily copy all of it into the buffer, overwriting important items such as the return pointer back to the main() function.

© 2019 Erik Van Buggenhout & Stephen Sims



Stack Overflow (1)

On this slide, the attacker has sent in 32 A's instead of only 12. As you can see, strcpy() happily wrote past the 16-byte buffer allocation, overwriting the return pointer back to main() among other things. When the process went to return control to main, it returned to "AAAA." The letter "A" in hex-ASCII is "0x41." The "0x" on the front of the number indicates that it is a hexadecimal value or Base-16. So "AAAA" maps to "0x41414141" which is why we are seeing that show up in the segmentation fault at the bottom. The attacker now knows they can gain control of the program. At this point, they would use a debugger or disassembler to determine the exact number of bytes before reaching the return pointer.

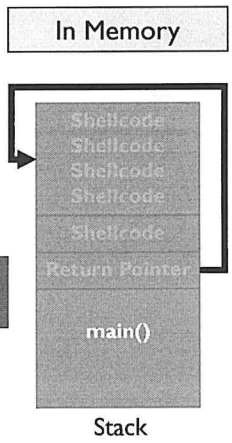
© 2019 Erik Van Buggenhout & Stephen Sims

Stack Overflow (2)

1. Using a debugger, the attacker gets the static memory address of the buffer.
2. They place their shellcode into the buffer and overwrite the return pointer with the address of the buffer.

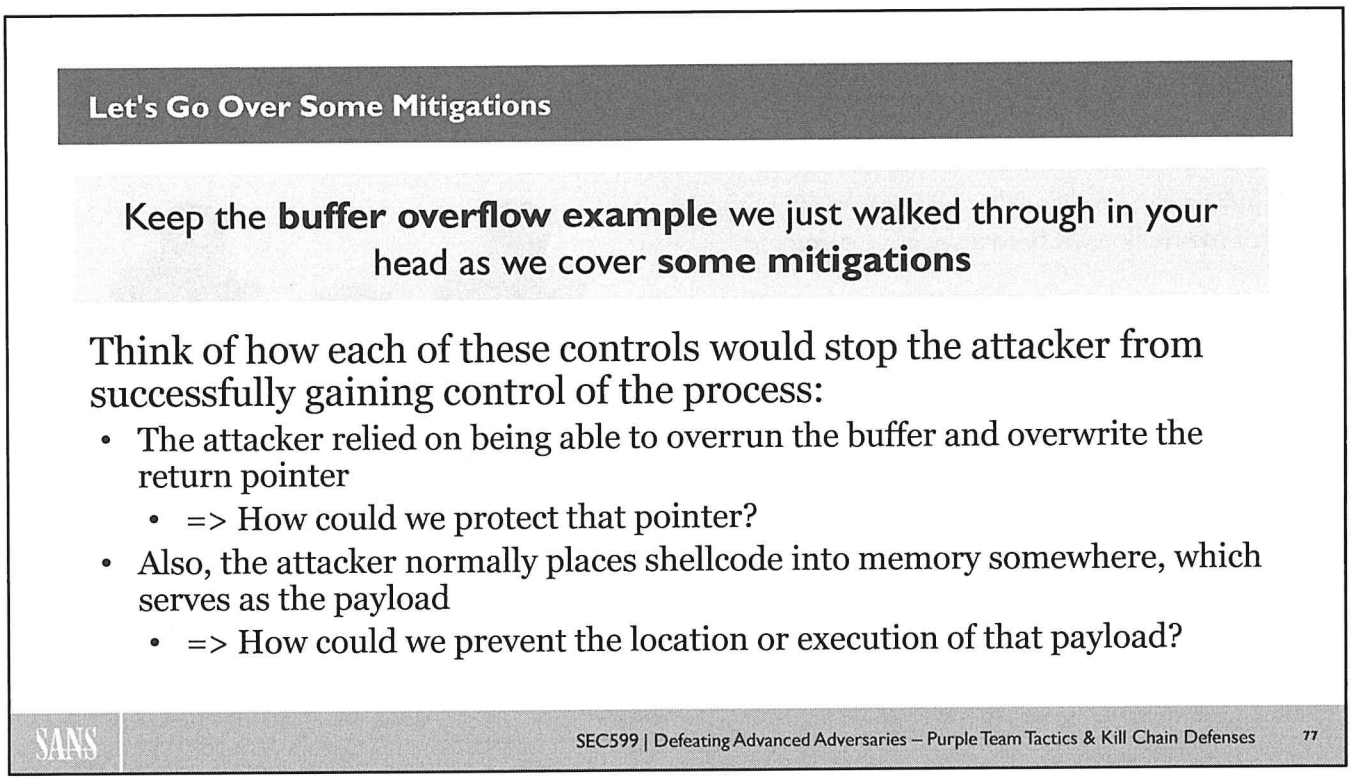
Segmentation Fault
RP = Address of Shellcode

3. When the process goes to return control to main(), control is instead passed to the attacker's shellcode.



Stack Overflow (2)
Once the attacker determines the exact number of bytes to get to the return pointer, as well as the address of the buffer, they can insert their shellcode into the buffer first, and then overwrite the return pointer back to main() with the address of their shellcode. The exploit mitigations will focus on things like randomizing the addressing in memory, preventing execution in writable regions, place guards into memory, and many other approaches.

© 2019 Erik Van Buggenhout & Stephen Sims



Let's Go Over Some Mitigations

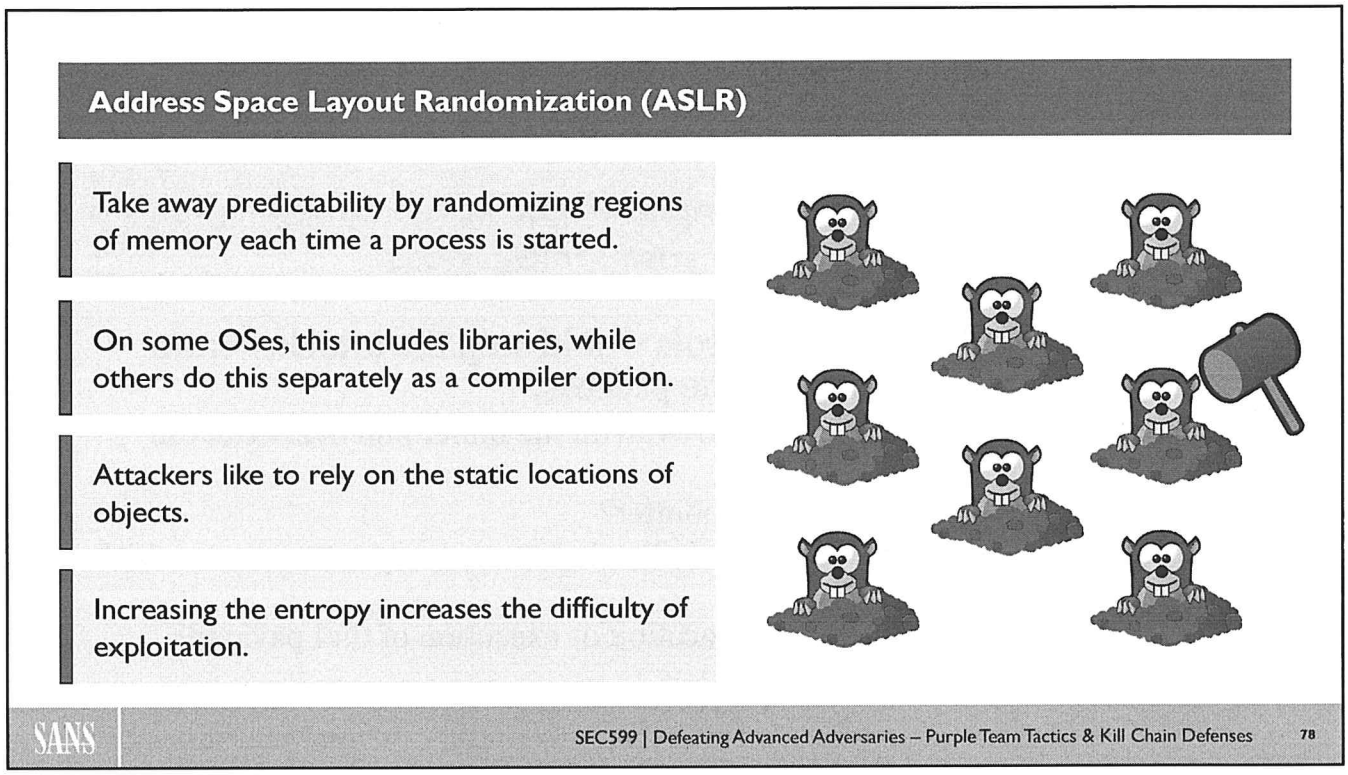
Now, let's keep the buffer overflow example we just discussed in our mind as we cover some mitigating controls. Try to think how each of the introduced controls could possibly stop the attacker from successfully gaining control of the process (and thus being able to execute malicious code).

There are two interesting things to note:

1. The attacker relied on being able to overrun the buffer and overwrite the return pointer. How could we protect that pointer?
2. Furthermore, the attacker usually attempts to place shellcode into memory somewhere, which will serve as a payload that is to be executed. How could we prevent the location or execution of that payload?

The author recognizes this is not the easiest topic of the day, but if we want to protect against advanced adversaries, it's vital we understand how modern exploit mitigation strategies work. As always, should you have questions or would look some additional guidance on this subject, please don't hesitate to contact your instructor or TA. Let's have a look!

© 2019 Erik Van Buggenhout & Stephen Sims

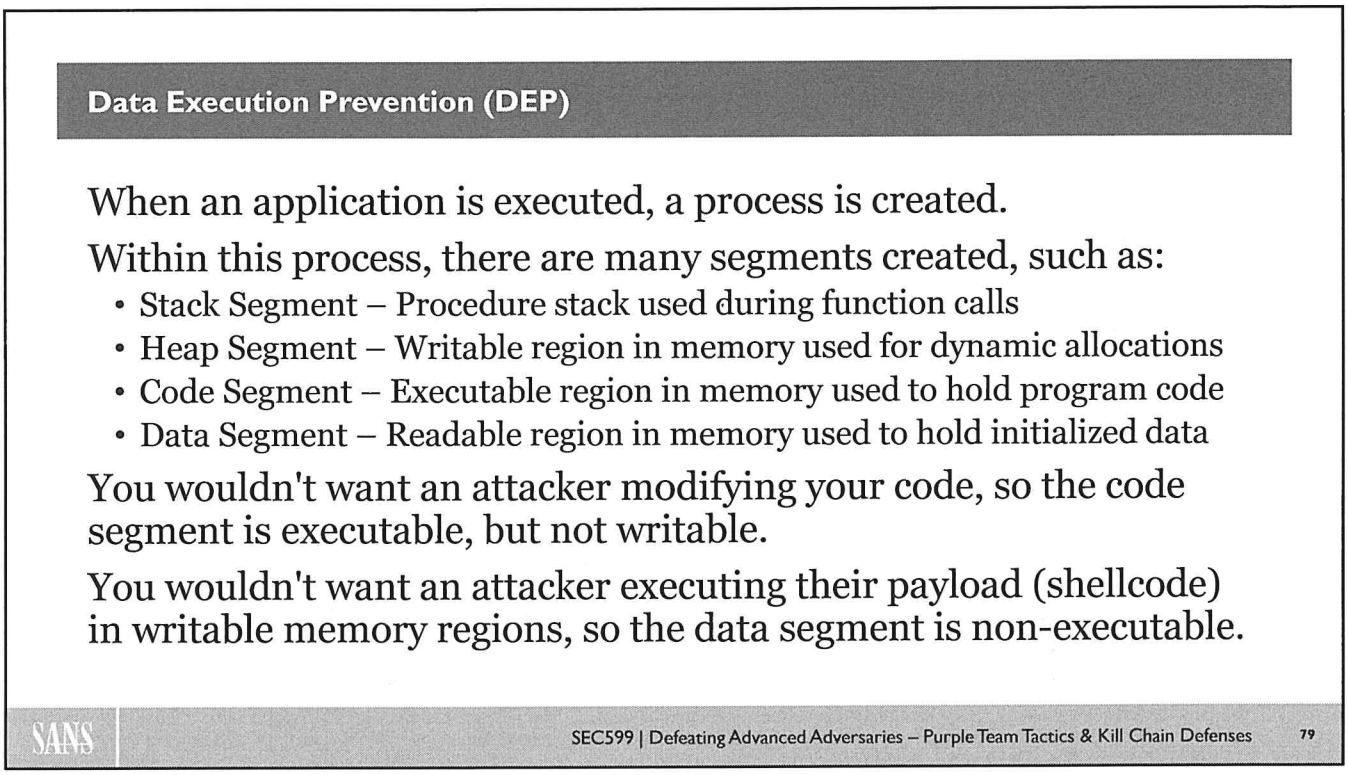


Address Space Layout Randomization (ASLR)
Imagine if each time you went to retrieve the salt and pepper from your kitchen that it was in a different location. Even though the last time you used it, you put it in a specific location for retrieval later, the next time you went to use it, the location changed again. This would likely cause frustration. The concept is not so different from ASLR. Attackers like for things to be static in memory no matter on what system the vulnerable application is running. This allows for them to use static addressing in their exploits. If an attacker gains control of a process and attempts to execute their payload, but the location has changed, it will likely crash and fail.

On some operating systems, the libraries are randomized as part of the overall system setting for ASLR. Many Linux variants include a file called "randomize_va_space" which stores a value of 0, 1, or 2. If the value stored is a 0, then ASLR is off, including libraries. If the value is a 1 or a 2, then ASLR is on. On the Windows OS, ASLR cannot be turned off for the system; however, there is a compile-time control known as "DynamicBase" which determines if a Dynamic Link Library (DLL) is to be randomized once loaded into a process.

Take the earlier example of the salt and pepper. If your kitchen is rather small, and you know that the salt and pepper must be stored somewhere in the kitchen, then the chances of you guessing the right spot are pretty good in a short number of tries. At least when comparing it to an example where your kitchen is the size of a sports stadium. If this were true, the chances of successfully guessing the right location of the salt and pepper is greatly lessened.

© 2019 Erik Van Buggenhout & Stephen Sims

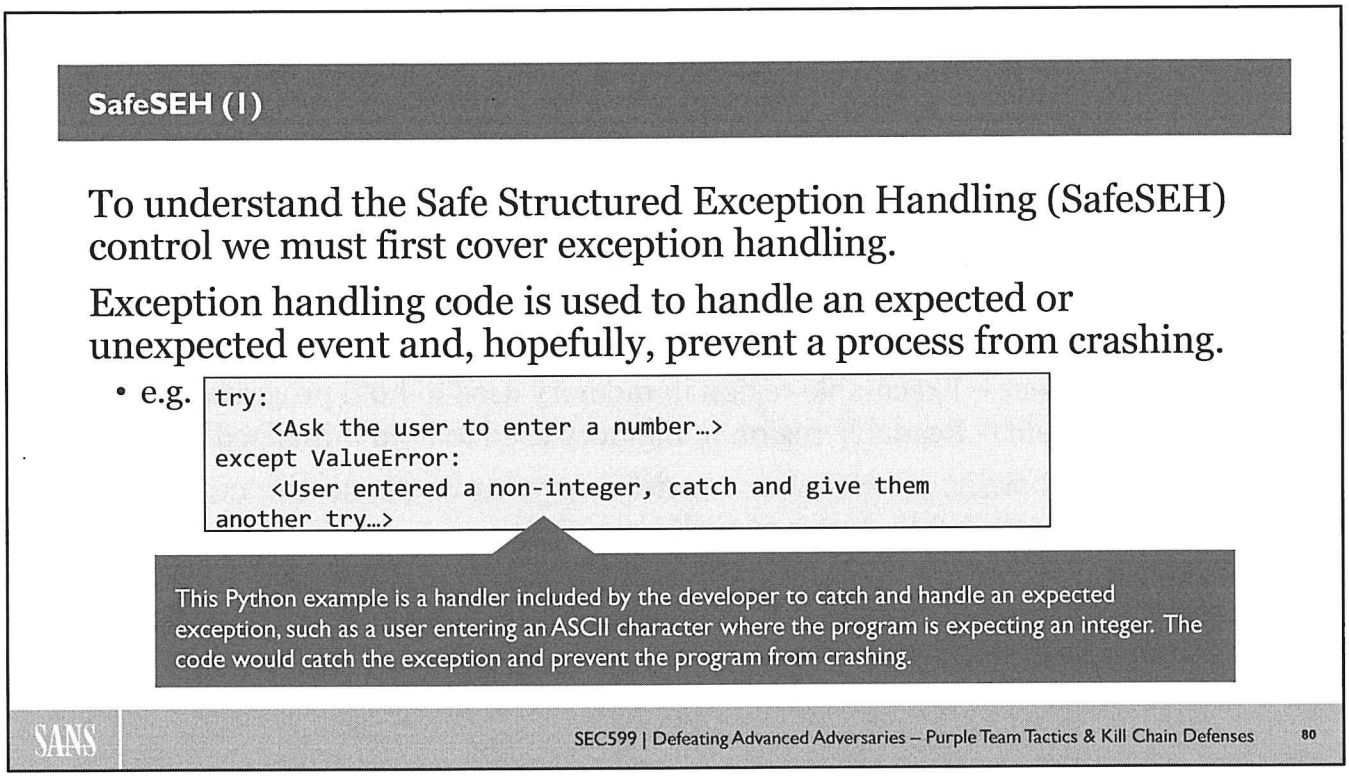


Data Execution Prevention (DEP)

DEP is a mature OS control that is quite simple to explain. Executable code resides in its own segment in a process called the code segment. You certainly wouldn't want anyone to be able to modify the code in your program. When thinking about the three basic permissions, read/write/execute, then it is easy to see why the code segment should be executable but not writable. When dealing with writable segments of memory within a process such as the stack and heap, it is clear that they need to be writable. Now, since those regions are writable, we have concerns around an attacker inserting their own malicious code into these areas and somehow causing code execution to occur. We can mitigate this concern by marking this memory region as writable, but not executable.

DEP must be supported by the processor. In RAM, when an allocation occurs, a unit of measurement known as a page is used. On most OSes, a page of memory is 4KB for alignment purposes. They are allocated by the processor. As pages of memory are allocated, a special bit is set, known as the eXecute Disable (XD) or No eXecute (NX) bit, depending on the hardware architecture. Regardless, they are identical as to their role. The bit simply determines if the page is writable or executable, as the idea behind the control is that you cannot be both.

© 2019 Erik Van Buggenhout & Stephen Sims



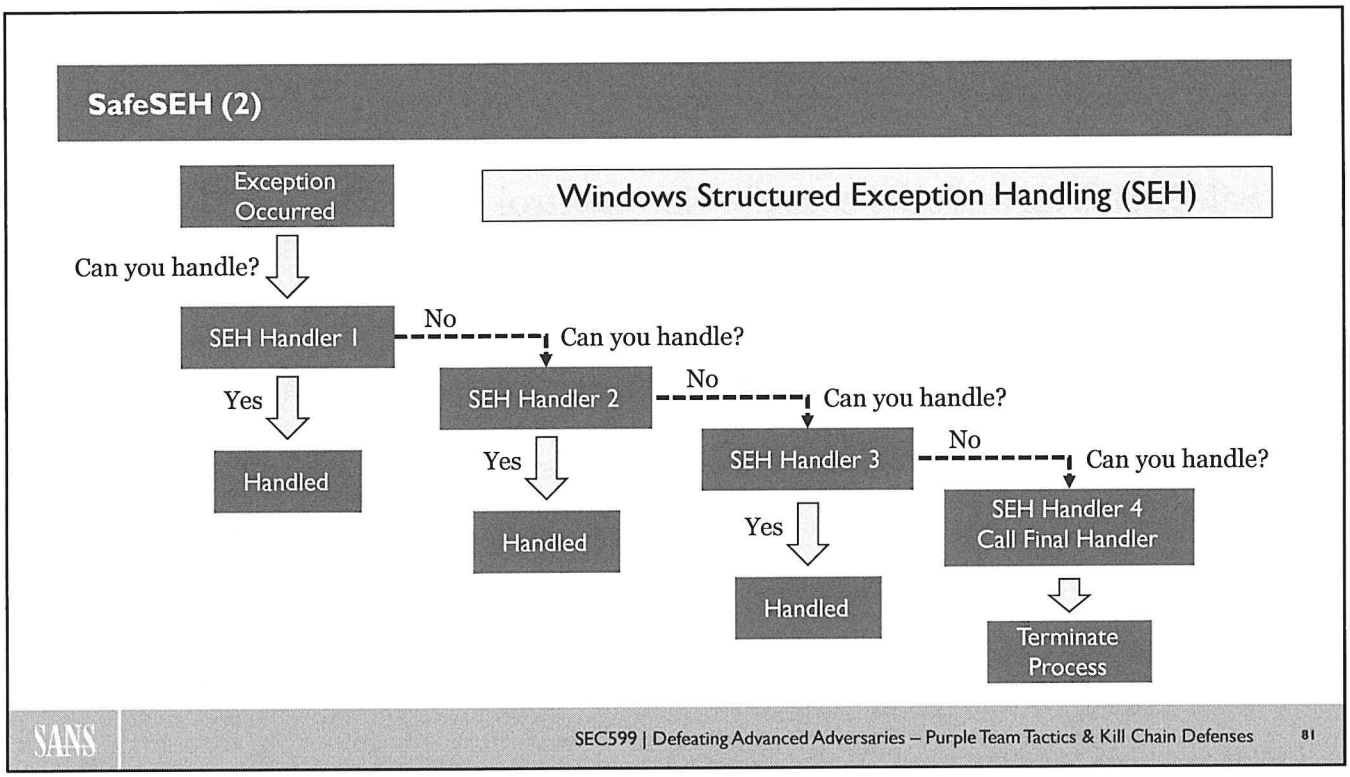
SafeSEH (1)

To explain the Windows Safe Structured Exception Handling (SafeSEH) compile-time exploit mitigation control, we must first cover basic exception handling and Windows SEH in general. An exception is something that occurs within a process, such as an anomaly, or an intentional attempt to cause a program fault, that potentially affects the stability of the process. If we have exception handling code within the program, then the exception that has occurred can potentially be handled, preventing the process from crashing. Some exceptions can be anticipated, while others are unexpected. Look at the following Python-like example:

```
try:
  <Ask the user to enter a number…>
except ValueError:
  <User entered a non-integer, catch and give them another try…>
```

The "try" and "except" syntax is used to create a handler in Python. We are basically saying that we wish to try the following code, but if an exception occurs, we wish to have it handled. We are asking the user of the script to enter in an integer value. If the user enters anything other than an integer value, Python will throw a "ValueError" exception, which can be anticipated. When this type of exception is experienced, our code would catch it and pass control back to the "try" block again. Handlers are supported by almost all programming languages.

© 2019 Erik Van Buggenhout & Stephen Sims



SafeSEH (2)

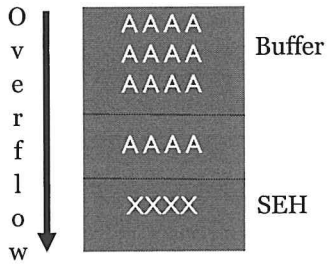
The windows SEH mechanism is used to handle various types of faults that cannot be handled or are not handled by developer code. An example of an exception that would cause control to shift the SEH mechanism is when the processor attempts to read from or write to a memory address that is not mapped into the process. Processes only take up the physical memory and virtual addressing that is required to run the program. If they run out of memory, more can be requested. If something causes a process to attempt to read or write to an address that is not mapped from virtual memory to physical RAM, then an exception occurs. Control would move to the SEH chain. It is called a chain because, as you can see on the slide, if one exception handler cannot handle the exception, control is passed down the chain until it is handled, or it reaches the end. If the end is reached and none of the handlers were able to handle the exception, then the program is terminated. The handler code resides in various DLLs such as ntdll.dll. The pointers to or addresses of the handlers are stored on the procedure stack for the thread.

© 2019 Erik Van Buggenhout & Stephen Sims

SafeSEH (3)

SafeSEH is an optional compile-time control that builds a table of all valid handlers in a DLL; the table stores the addresses of each valid handler.

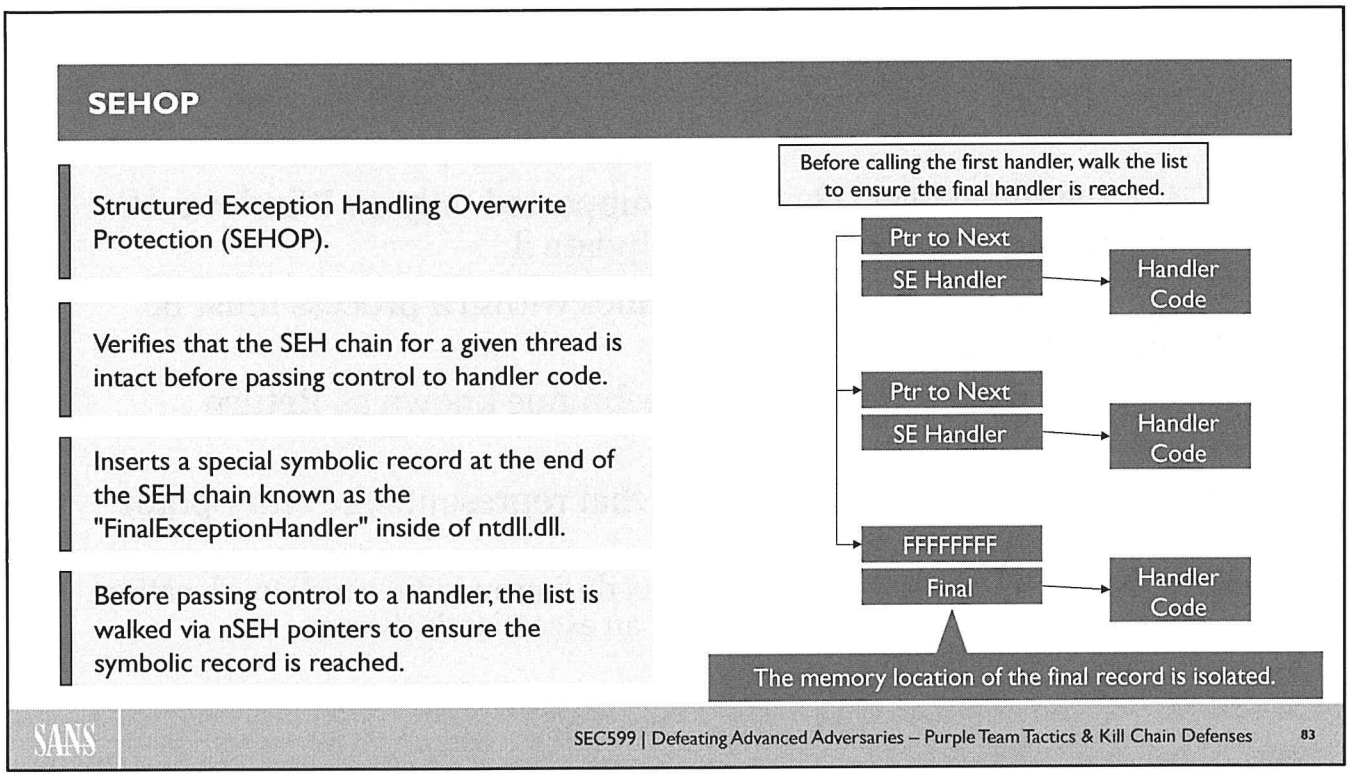
SafeSEH is aimed at stopping buffer overflows where an attacker attempts to overrun a buffer, overwriting the address of a handler to hijack control.



SafeSEH (3)

SafeSEH is a compile-time control that builds a table of all valid handlers inside of a DLL. Each handler has a starting memory address within a module. This is the address stored in the SafeSEH table. A common attack technique is for attackers to overwrite the location in memory where the address of a handler is stored. In case you are curious, the SEH chain resides on the stack for each thread within a process. An attacker would overwrite the handler address in memory with the address of their choosing. There are common techniques used by attackers to gain control of a process via SEH overwrites. If an attacker overwrites one of these SEH addresses and the address written by the attacker points into a SafeSEH-protected DLL, they would be caught, and the process terminated. It is not seen as a very effective control as all modules (DLL's) loaded into the process must participate in the control. A single module that does not participate in the control will render this exploit mitigation worthless.

© 2019 Erik Van Buggenhout & Stephen Sims



SEHOP

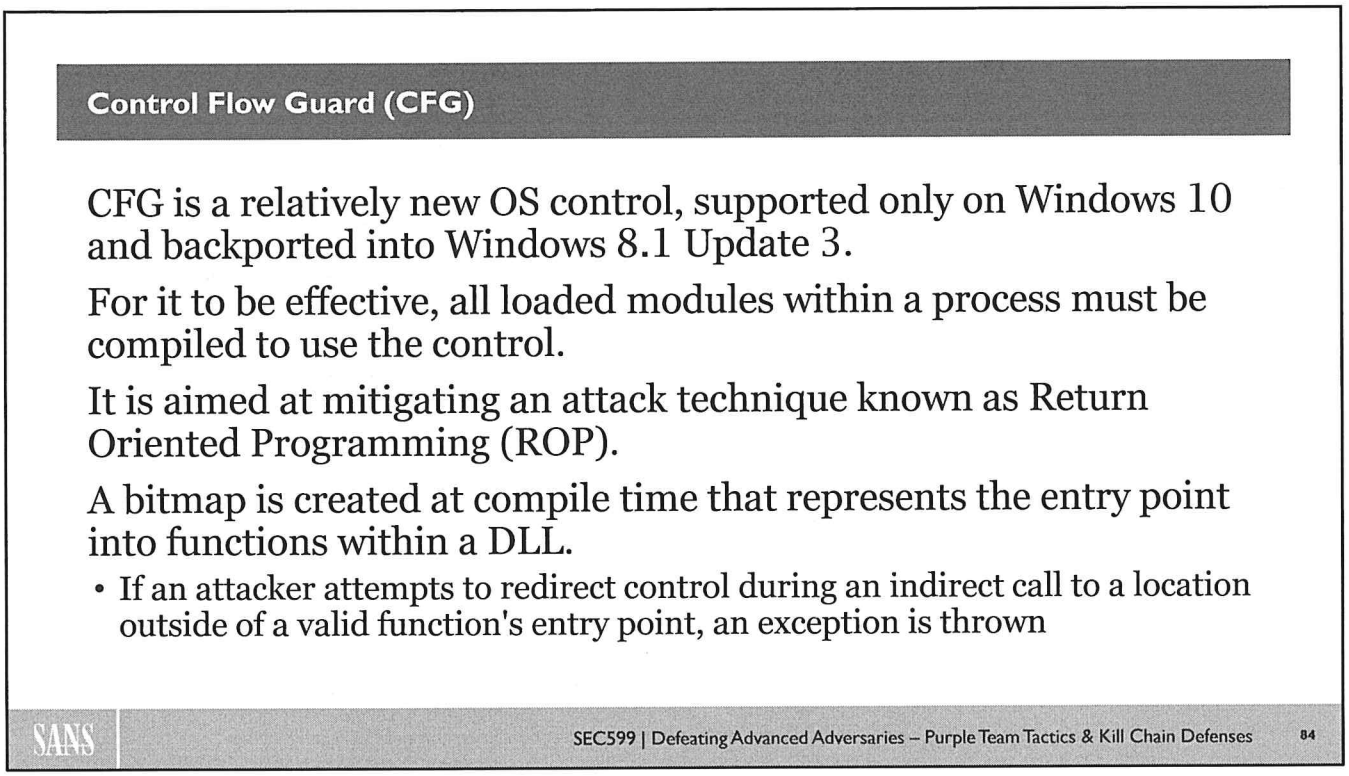
The Structured Exception Handler Overwrite Protection (SEHOP) control was added into Server 2008 and Vista; however, it is disabled by default on almost all versions of Windows. This is due to the potential lack of application support for the protection, although it can be enabled typically through the use of Microsoft's Enhanced Mitigation Experience Toolkit (EMET) or Exploit Guard. Normally, if you follow the nSEH pointers down the stack, you will reach the end of the list. If a handler has been overwritten, it is likely that walking the pointers will no longer reach the end of the list.

As described by Matt Miller (Skape) at Microsoft, the SEHOP control works by inserting a special symbolic record at the end of the SEH chain. Prior to handing control off to a called handler, the list is walked to ensure that the symbolic record is reachable.

References:

Miller, Matt (2009-02-2). Preventing the Exploitation of Structured Exception Handler (SEH) Overwrites with SEHOP. Retrieved January 11, 2017, from technet.Microsoft.com Website https://msrc-blog.microsoft.com/2009/02/02/preventing-the-exploitation-of-structured-exception-handler-seh-overwrites-with-sehop/

© 2019 Erik Van Buggenhout & Stephen Sims

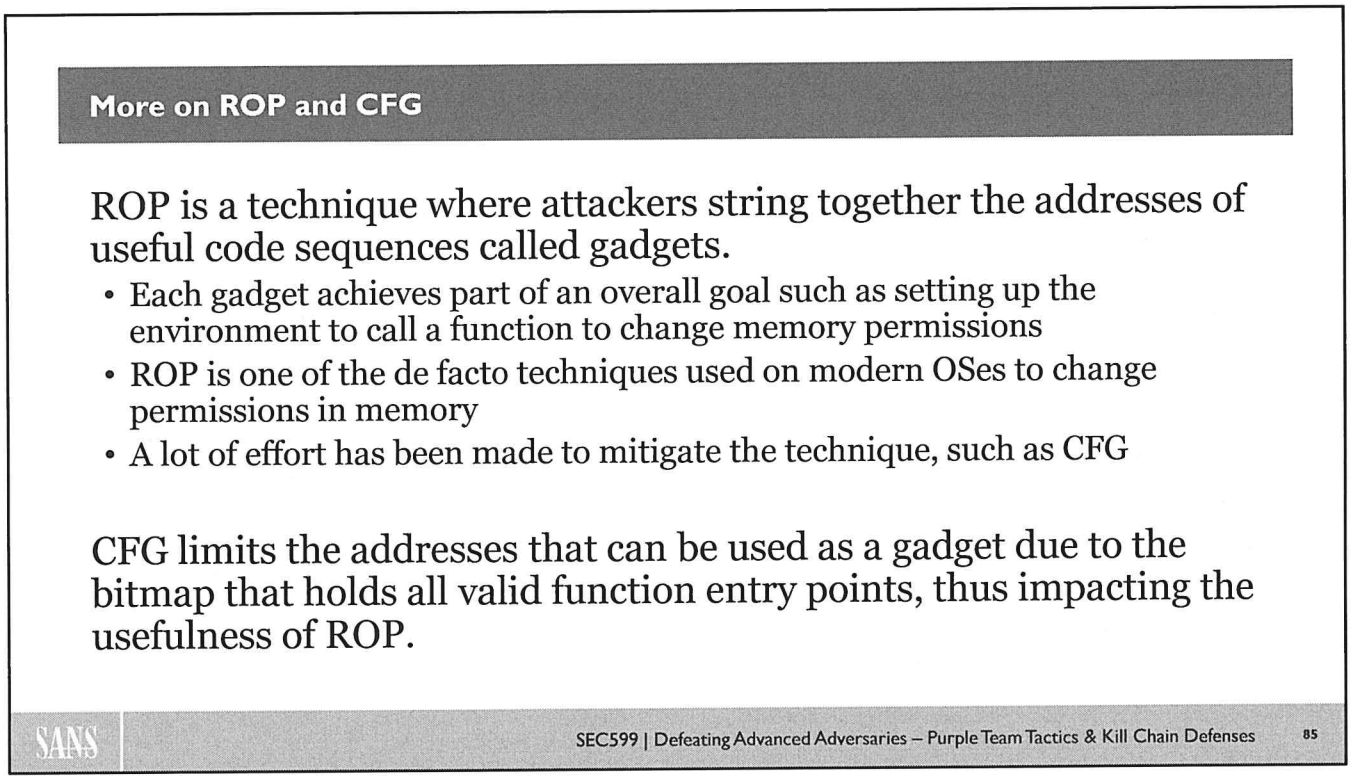


Control Flow Guard (CFG)
CFG is a newer control that requires support by the OS and also that each module (DLL) be compiled with the control as it requires code insertion. It is supported on Windows 10 and was backported into Windows 8.1 Update 3 and is also supported in Server 2016. Again, aside from the OS support requirement, CFG is only effective if all loaded modules (DLLs) are compiled to support the control. The control is aimed at mitigating a common attack technique known as Return Oriented Programming (ROP).

CFG works by creating a bitmap of all valid function entry points from within a DLL at compile time. When an indirect call to a function occurs within a module protected by CFG, the bitmap is checked to ensure that the address is that of a valid function entry point. If it is not, an exception is thrown. To understand indirection, we can use a simple analogy. If you decide to have pizza for dinner, you could go to the store, buy the ingredients, and make it yourself, or, you could order from a pizza delivery restaurant. If you make the pizza yourself, you can feel safe that the pizza has not been contaminated in any way as you are the preparer. If you order the pizza from a restaurant and have it delivered, there may be various points of concern. First, you did not witness the making of the pizza, and second, the pizza may have gone through an adventure during delivery to which you are not privy. We do not need to go into the details. This would-be indirection is heavily based on trust. CFG attempts to help secure this trust.

For more on CFG, visit: https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard

© 2019 Erik Van Buggenhout & Stephen Sims



More on ROP and CFG

As stated previously, ROP is a technique often used by attackers to achieve a goal. Often, the goal of ROP is to change the permissions in memory where attacker code resides. We previously discussed DEP and how it marks writable regions of memory as non-executable. By using ROP, one could set up the arguments to a function call such as VirtualProtect() that allows you to change the permissions in memory. ROP works by identifying useful short sequences of code within executable modules and stringing them together to accomplish their goal. These code sequences are referred to as gadgets. We string the gadgets together, which formulates our ROP chain. As mentioned, once an attacker gains control of a process, they may wish to change memory permissions, so they can have their shellcode executed. To do this, a system call must be made to a function like VirtualProtect() or VirtualAlloc(). Control of the process is passed to the gadgets, which returns to each successive gadget, each performing a piece of the overall goal to set up the arguments to the desired function call. CFG limits the number of useful gadgets by only allowing indirect calls to go to addresses indicated in the CFG bitmap.

© 2019 Erik Van Buggenhout & Stephen Sims

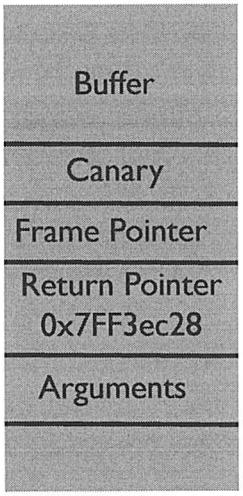
Stack Canaries / Security Cookies

When a function is called, a return pointer is pushed onto the stack frame for that function.

- The return pointer is used to return control to the caller once the called function is finished
- Overwriting this pointer due to the use of an unsafe function such as strcpy() can result in control hijacking

A canary is a special value placed above the return pointer for protection.

- In order to overwrite the return pointer, the canary must also be overwritten
- If the value is unknown to the attacker, then it won't match when it is checked prior to returning control to the caller

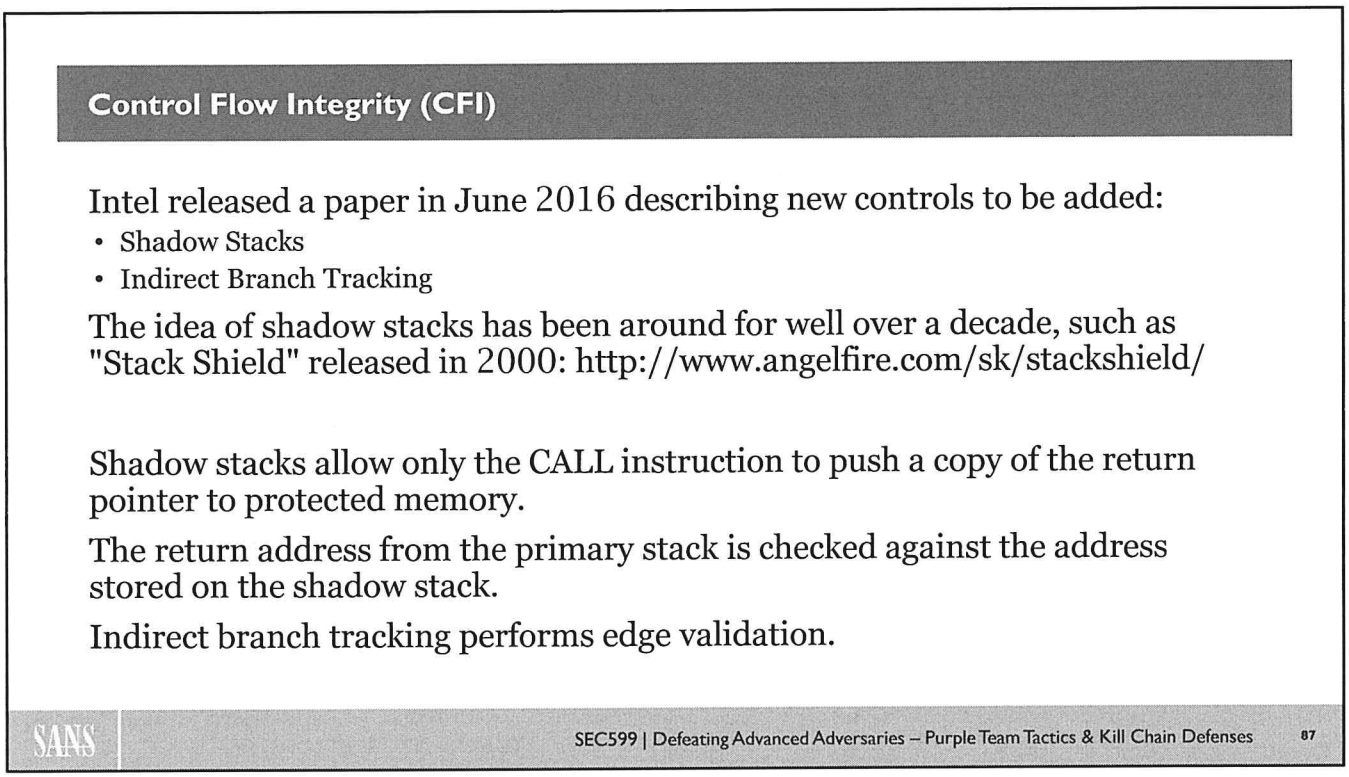


Stack Canaries / Security Cookies

Stack canaries, also called security cookies in the world of Microsoft, are a compile-time control that inserts code into functions deemed as needing protection. During a normal function call, an address known as the return pointer is pushed into memory onto something known as the procedure stack. Each function call gets its own stack frame on the procedure stack. A stack frame is nothing more than a small amount of memory to store items such as arguments, buffer space, and variables such as the return pointer. Once the function is finished, its stack frame is torn down. The return pointer is used to return control to a specific point in the program just after the occurrence of the function call. Since it is stored in writable memory, it is prone to being overwritten. If overwritten, control of the process can be hijacked by an attacker.

The canary serves as a guard that is pushed onto the stack frame above the return pointer. In order for an attacker to reach the return pointer during an overwrite attempt, they must also overwrite the canary. Most canaries are random and thus an attacker would not know what value to write to that position during an overflow. Prior to returning control to the calling function, the canary is checked to ensure it has not been damaged. If the canary check fails, then an exception is thrown, and the process terminated.

© 2019 Erik Van Buggenhout & Stephen Sims



Control Flow Integrity (CFI)

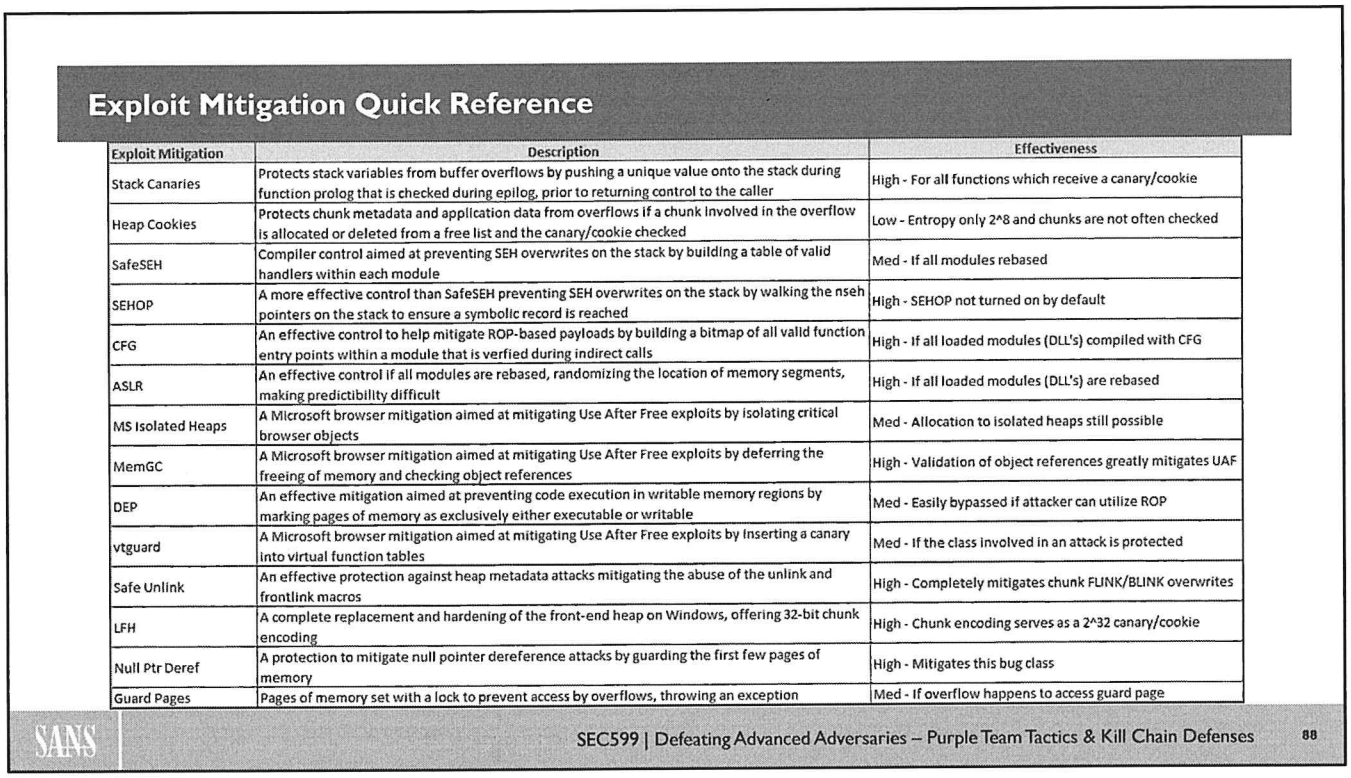
In June 2016, Intel released some press releases and a detailed PDF on Control Flow Enforcement (CET), their new upcoming control to prevent code reuse attacks and Return Oriented Programming (ROP) techniques. https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

The primary controls introduced in this paper are Shadow Stacks and Indirect Branch Tracking. Each of these ideas has been proposed in various forms for well over 10 years. An example is "Stack Shield" released back in 2000. http://www.angelfire.com/sk/stackshield If properly integrated into the processor architecture, each control could have a moderate impact on code reuse techniques. The grsecurity team released a short posting as to their opinion on how Intel's implementation plan of these controls is lacking. https://forums.grsecurity.net/viewtopic.php?f=7&t=4490#P9

Shadow stacks work by marking certain pages of memory as protected, allowing only the CALL instruction the ability to write a copy of the return addresses used in the call chain. The return pointer on the actual stack is tested against the copy stored on the shadow stack. If there is a mismatch, an exception is thrown.

Indirect branch tracking takes advantage of the new instruction "ENDBR32" for 32-bit or "ENDBR64" for 64-bit. This instruction is inserted after each valid call instruction. If this is not the next instruction, an exception is thrown. The instruction has the same effect as a NOP and simply is used for validation.

© 2019 Erik Van Buggenhout & Stephen Sims



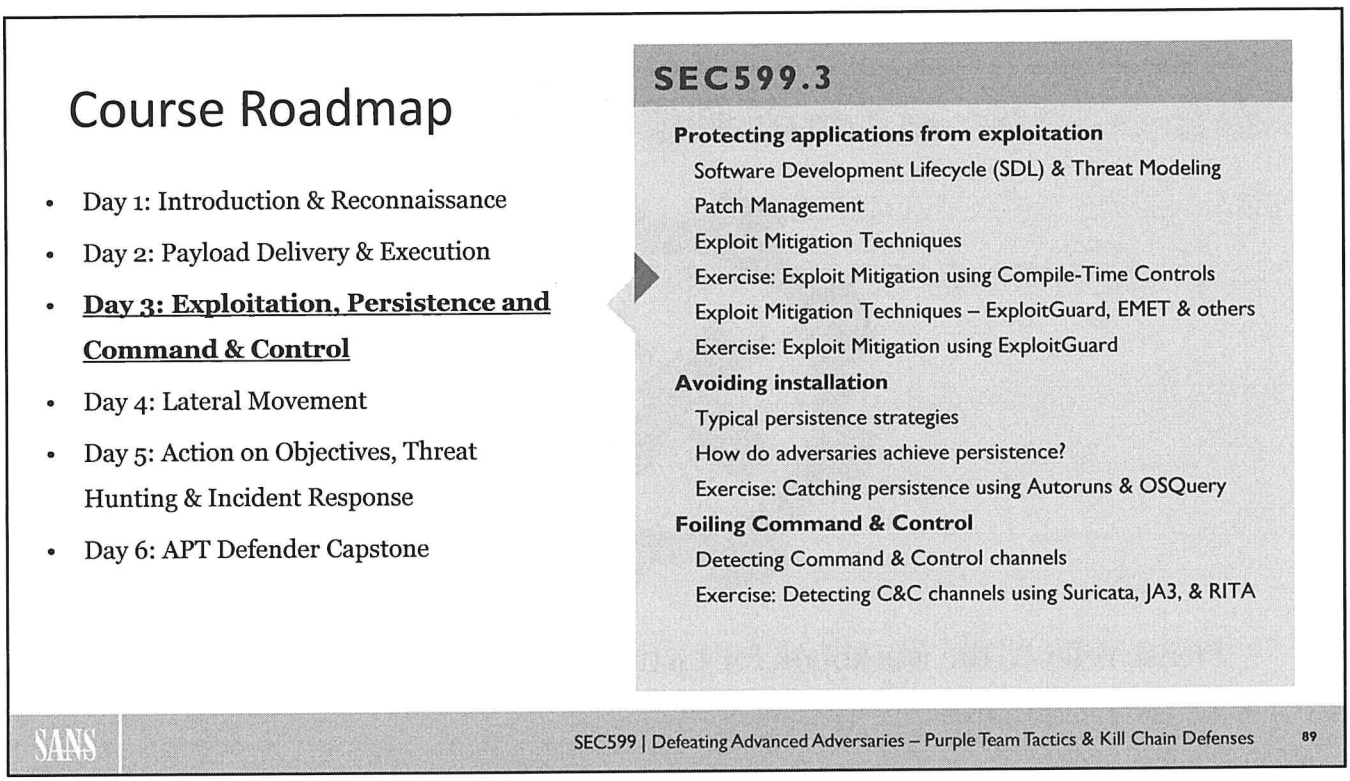
Exploit Mitigation Quick Reference

Exploit Mitigation	Description	Effectiveness
Stack Canaries	Protects stack variables from buffer overflows by pushing a unique value onto the stack during function prolog that is checked during epilog, prior to returning control to the caller	High - For all functions which receive a canary/cookie
Heap Cookies	Protects chunk metadata and application data from overflows if a chunk involved in the overflow is allocated or deleted from a free list and the canary/cookie checked	Low - Entropy only 2^8 and chunks are not often checked
SafeSEH	Compiler control aimed at preventing SEH overwrites on the stack by building a table of valid handlers within each module	Med - If all modules rebased
SEHOP	A more effective control than SafeSEH preventing SEH overwrites on the stack by walking the nseh pointers on the stack to ensure a symbolic record is reached	High - SEHOP not turned on by default
CFG	An effective control to help mitigate ROP-based payloads by building a bitmap of all valid function entry points within a module that is verfied during indirect calls	High - If all loaded modules (DLL's) compiled with CFG
ASLR	An effective control if all modules are rebased, randomizing the location of memory segments, making predictibility difficult	High - If all loaded modules (DLL's) are rebased
MS Isolated Heaps	A Microsoft browser mitigation aimed at mitigating Use After Free exploits by isolating critical browser objects	Med - Allocation to isolated heaps still possible
MemGC	A Microsoft browser mitigation aimed at mitigating Use After Free exploits by deferring the freeing of memory and checking object references	High - Validation of object references greatly mitigates UAF
DEP	An effective mitigation aimed at preventing code execution in writable memory regions by marking pages of memory as exclusively either executable or writable	Med - Easily bypassed if attacker can utilize ROP
vtguard	A Microsoft browser mitigation aimed at mitigating Use After Free exploits by inserting a canary into virtual function tables	Med - If the class involved in an attack is protected
Safe Unlink	An effective protection against heap metadata attacks mitigating the abuse of the unlink and frontlink macros	High - Completely mitigates chunk FLINK/BLINK overwrites
LFH	A complete replacement and hardening of the front-end heap on Windows, offering 32-bit chunk encoding	High - Chunk encoding serves as a 2^32 canary/cookie
Null Ptr Deref	A protection to mitigate null pointer dereference attacks by guarding the first few pages of memory	High - Mitigates this bug class
Guard Pages	Pages of memory set with a lock to prevent access by overflows, throwing an exception	Med - If overflow happens to access guard page

Exploit Mitigation Quick Reference

We have not covered all of the exploit mitigations shown on this slide, but this can serve as a quick reference to see what each control does from a high level and get an idea as to their effectiveness. The effectiveness is subjective and based on the experience and experiences of each exploit writer. These are the ratings as given by this author.

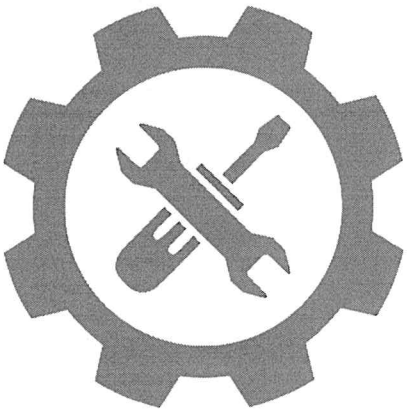
© 2019 Erik Van Buggenhout & Stephen Sims



This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Exercise: Exploit Mitigation Using Compile-Time Controls



Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

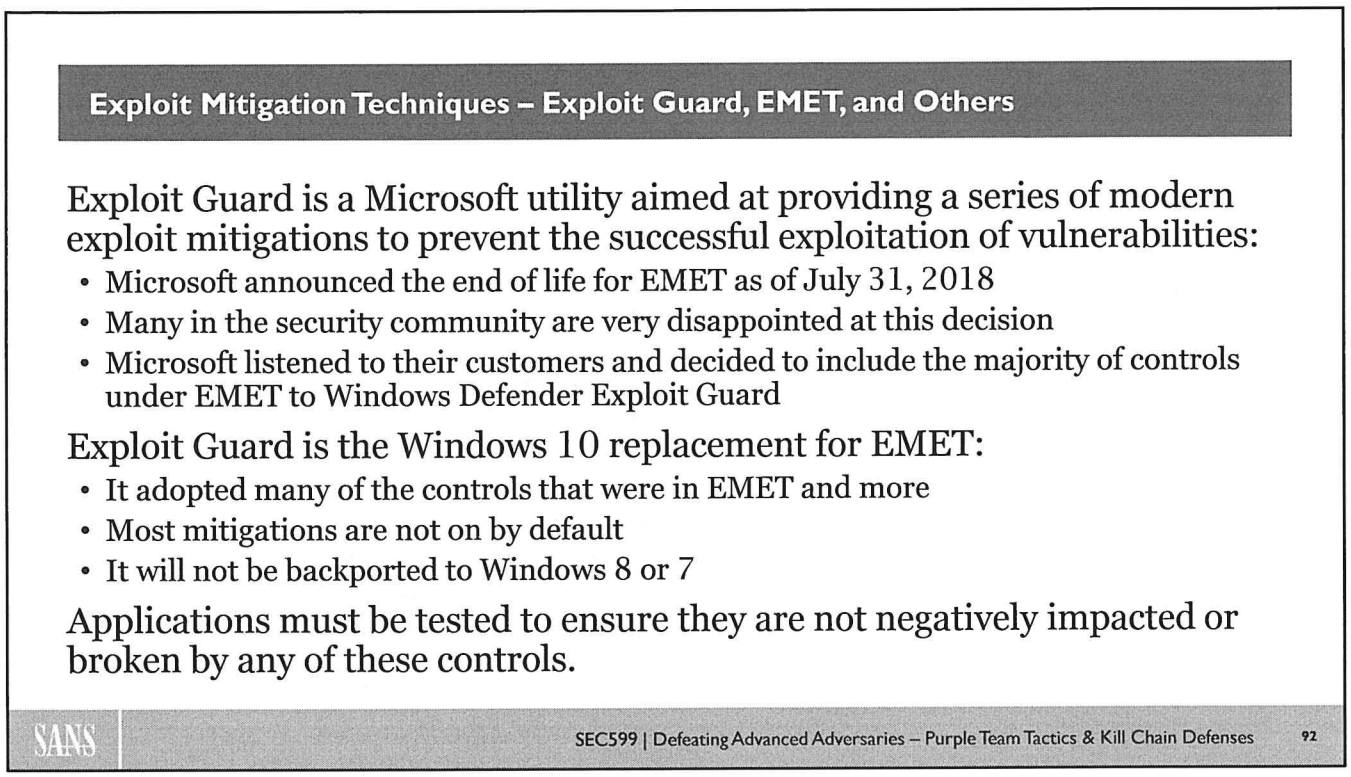
Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims



Exploit Mitigation Techniques – Exploit Guard, EMET, and Others

Microsoft's EMET utility was released back in 2009 around the same time as Windows 7. It offered numerous exploit mitigations aimed at providing defense-in-depth to applications and prevent the successful exploitation of vulnerabilities. EMET version 5.52 was the latest release from Microsoft prior to its end of life. All recent EMET releases focused on resolving disclosed bypass techniques. Sadly, Microsoft announced in 2016, that support and development of the product will end on July 31, 2018. Initially, Microsoft meant to discontinue support in January 2017, but due to feedback from customers, they agreed to push back the date. The exact reasoning for the discontinuation of EMET by Microsoft is unclear, though it likely has to do with a low adoption rate over the years and a focus on Windows 10 security and beyond. EMET had a low adoption rate within organizations, which may have partially led to Microsoft's decision to discontinue support.

Microsoft's recommendation is to migrate to Windows 10 for improved security. It is very unlikely that support will become available for Windows 8 or 7. Exploit Guard started with the Fall Creators Update of Windows 10 in October 2017. Many of the mitigations or protections from EMET have been worked into Exploit Guard, as well as some new ones. The majority of these mitigations are not on by default. Each application must be tested to ensure there is no negative impact associated with any of the protections. This also includes performance issues. Some of the newer protections are quite aggressive and are likely to prevent some applications from even starting.

© 2019 Erik Van Buggenhout & Stephen Sims

Application Testing

Microsoft tests the various exploit mitigations against their applications to ensure they are not broken:

- They may also opt to disable certain protections at a per-application level if one is causing trouble
- It is not possible to test under all conditions

Organizations must test internal and third-party applications under Exploit Guard enforcement to ensure stability.

The controls may also cause a performance hit:

- This is typical of any exploit mitigation

Application Testing

Another point of frustration is around application testing. It requires someone to go through all applications being considered for Exploit Guard's various protections by your organization and check to see if any of them cause an issue. It is difficult to say at what point you have done enough testing to deem an application safe to use with Exploit Guard. Then, if an update is released to resolve a bypass, technique testing would again be required to ensure the new version doesn't pose any new issues. It is simply not possible to test all scenarios under which an application may run. This is very much similar to how quality assurance (QA) testing occurs during software development. Applications like Microsoft Word, Excel, PowerPoint, Internet Explorer, Edge, Adobe Flash, Java, and several others are tested by Microsoft for compatibility. If you think about the main applications that fall victim to exploitation at an organization, that list alone should be quite effective.

There are also concerns about a system performance hit due to the additional controls. The protections certainly result in extra code execution to perform enforcement. This can slow down an application; however, on a modern workstation, it shouldn't be too big of an issue.

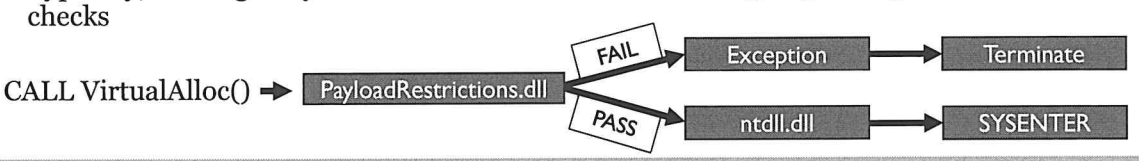
© 2019 Erik Van Buggenhout & Stephen Sims

How Does Exploit Guard Work?

The module PayloadRestrictions.dll is loaded into all processes designated for protection by Exploit Guard.

Many of the controls simply "hook" application flow at specific points:

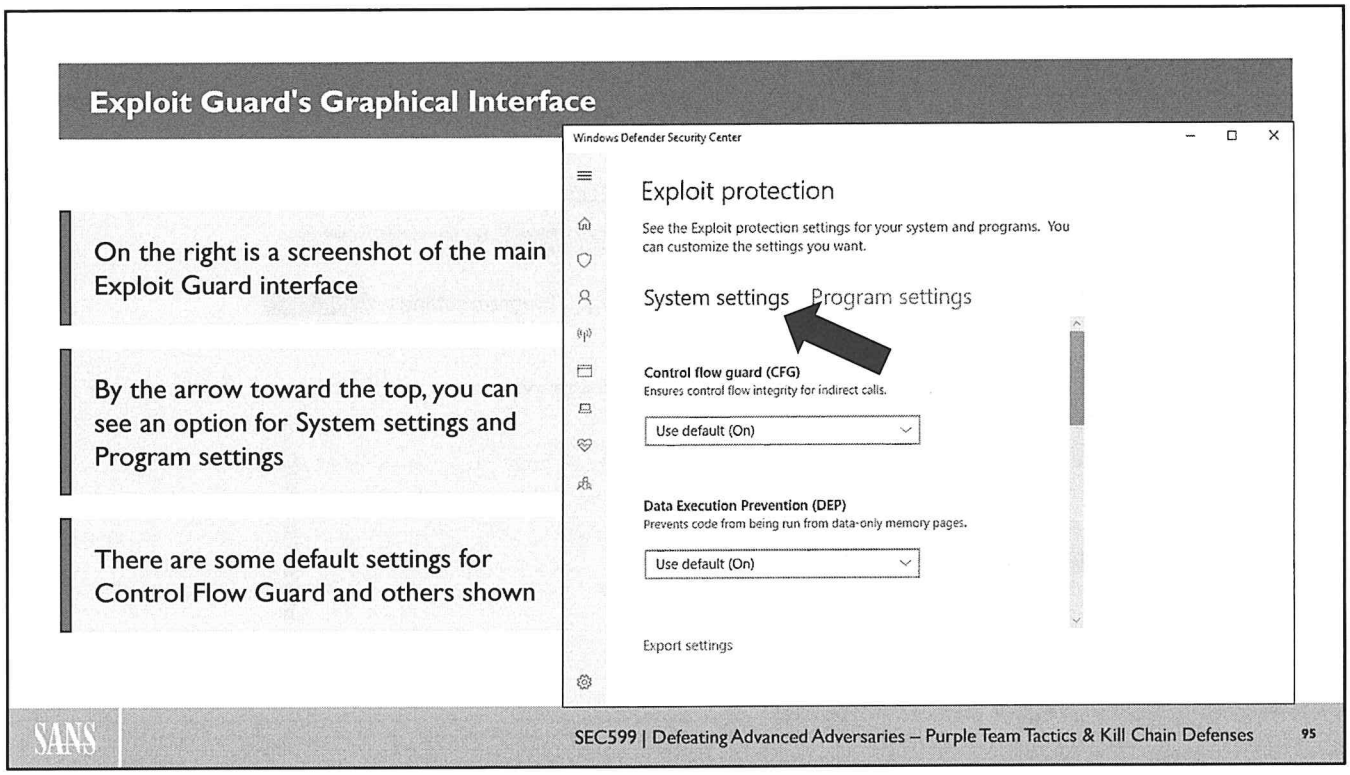
- An example of hooking is when a table of pointers to various functions is overwritten with pointers to different code
  - This is commonly used by malware, endpoint protection suites, and anti-exploitation products
  - Typically, the originally intended function is reached after going through a series of checks



How Does Exploit Guard Work?

A big question is likely, "How do the protections under Exploit Guard work?" Some of the controls are system-level controls such as DEP, where Exploit Guard can control the settings as opposed to going through the system control panel. The more specific per application controls that are native to Exploit Guard often work by hooking. This is very similar, if not identical, to how many endpoint protection and antivirus products work, as well as malware. Imagine an application wanting to call a function that is deemed critical. Microsoft classifies various functions as critical, such as those with the ability to change permissions in memory, allocate new memory, and many others. When the application goes through the normal channel of calling a critical function, the address of that function has been overwritten with an address inside of PayloadRestrictions.dll. This allows Exploit Guard to perform any checks, and if all looks good, control is passed to the desired critical function. We will look at specific examples of controls coming up soon.

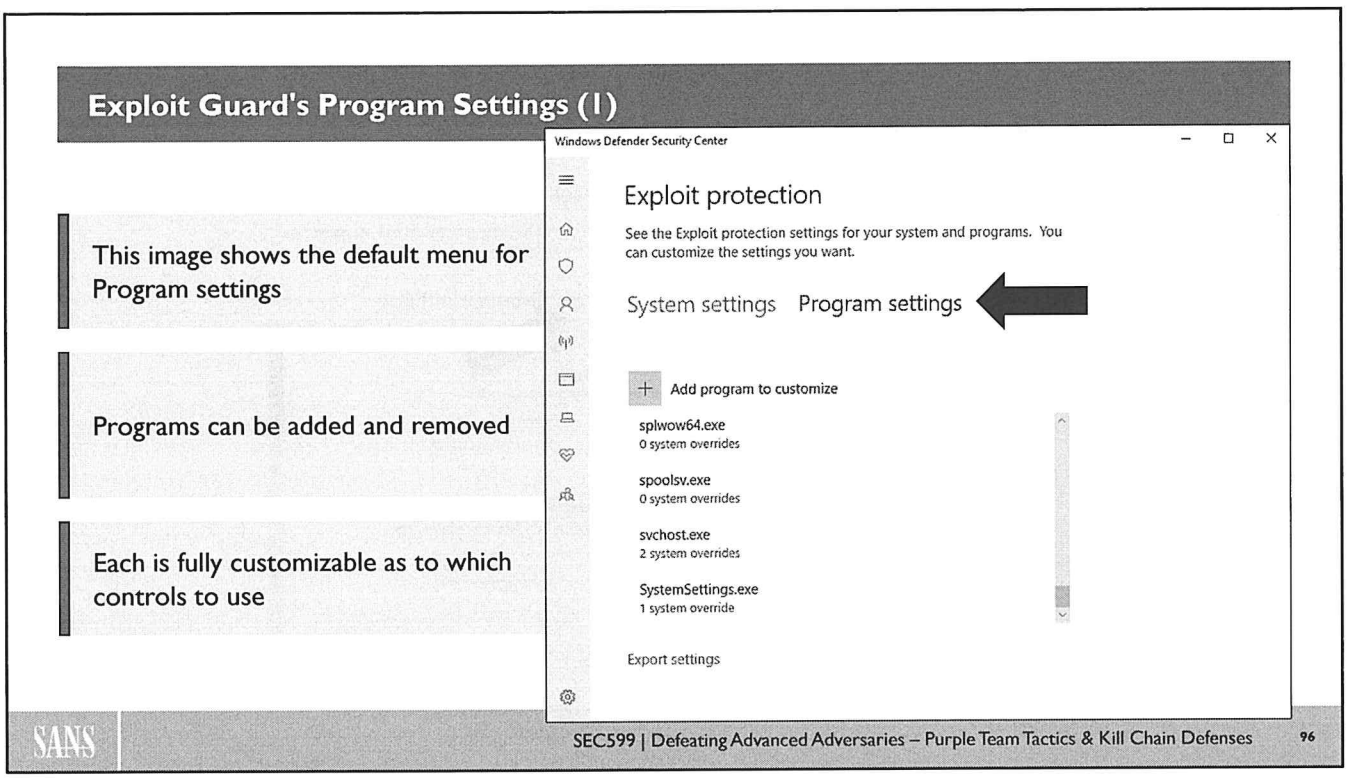
© 2019 Erik Van Buggenhout & Stephen Sims



Exploit Guard's Graphical Interface

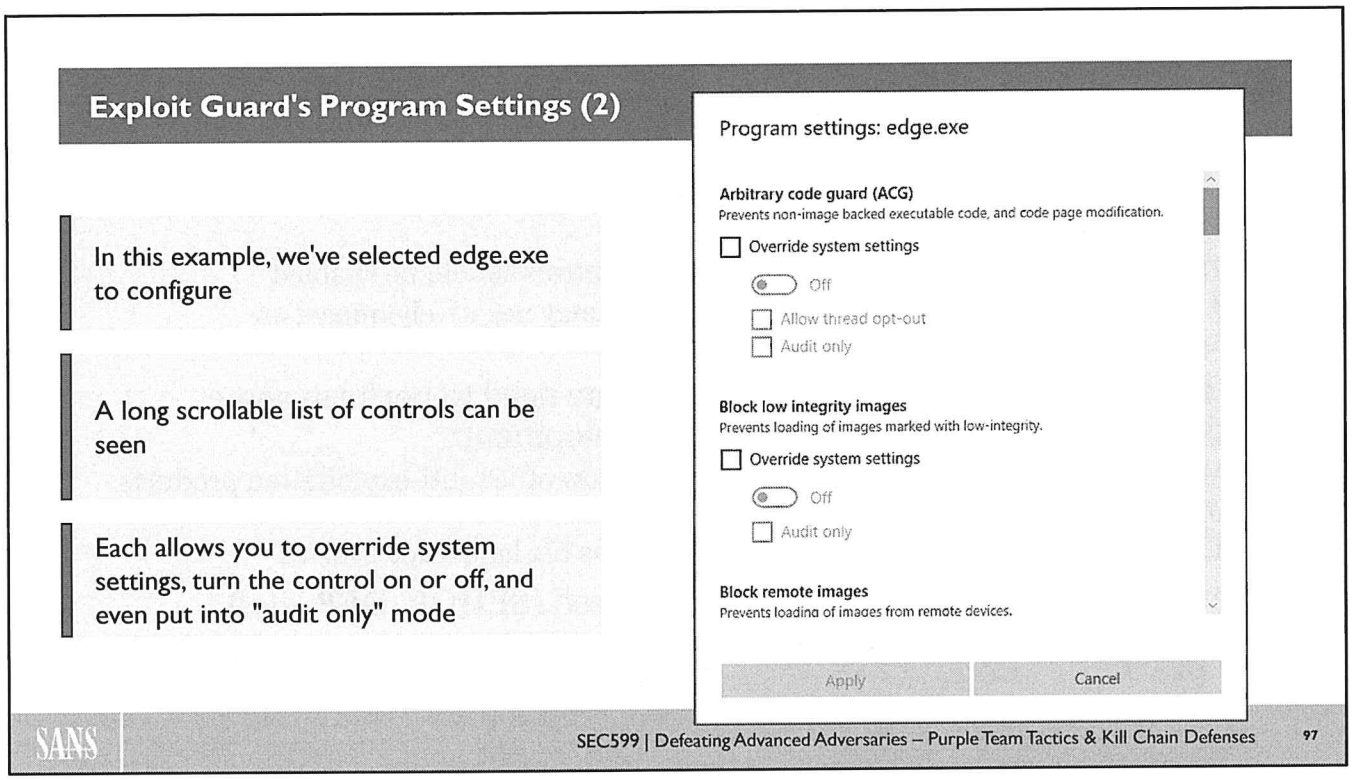
On this slide is a screenshot of the primary Exploit Guard interface. The easiest way to get here on a Windows 10 system with Exploit Guard installed is to click on the Start button, and type in "Exploit Protection." The two main items to point out are the System settings menu and the Program settings menu, as marked by the black arrow. On the image shown, we are looking at a piece of the System settings, showing Control Flow Guard (CFG) and Data Execution Prevention (DEP). As you can see, it also scrolls down further to show additional system options, which allow you to apply controls as a global setting. We will cover the various controls shortly.

© 2019 Erik Van Buggenhout & Stephen Sims



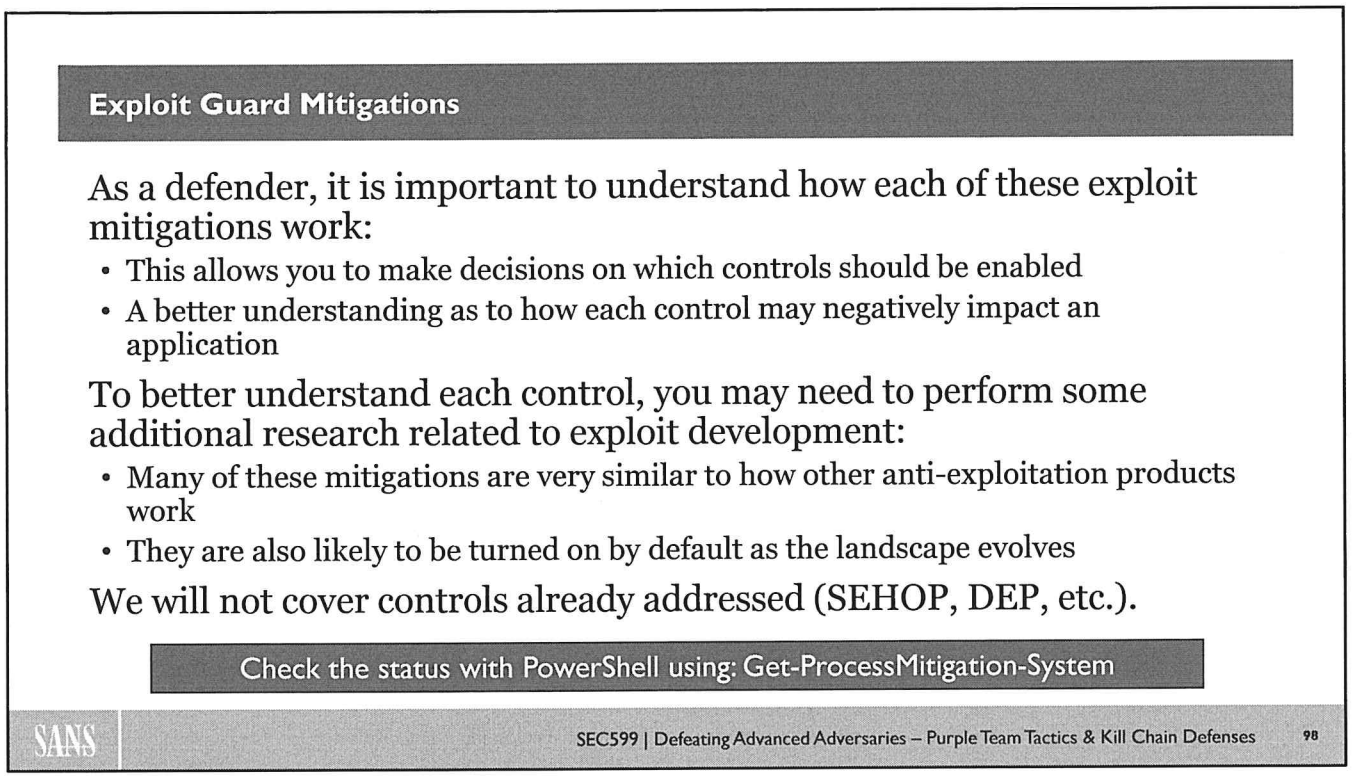
Exploit Guard's Program Settings (1)
On this slide is a screenshot of the Program settings window. You can see the scrollable list of applications. Each allows you to set the specific controls for each program.

© 2019 Erik Van Buggenhout & Stephen Sims



Exploit Guard's Program Settings (2)
This slide shows an example of the settings at the per-program level. We have selected edge.html as the program to configure. A long scrollable list can be seen with each control available. Exploit Guard allows you to configure the controls at a very granular level in the event certain controls have compatibility issues with a program. There is even the option to put a control into "audit only" mode so that you can see if a control would have caused an issue, or simply to use exploit guard as a detection tool as opposed to prevention.

© 2019 Erik Van Buggenhout & Stephen Sims

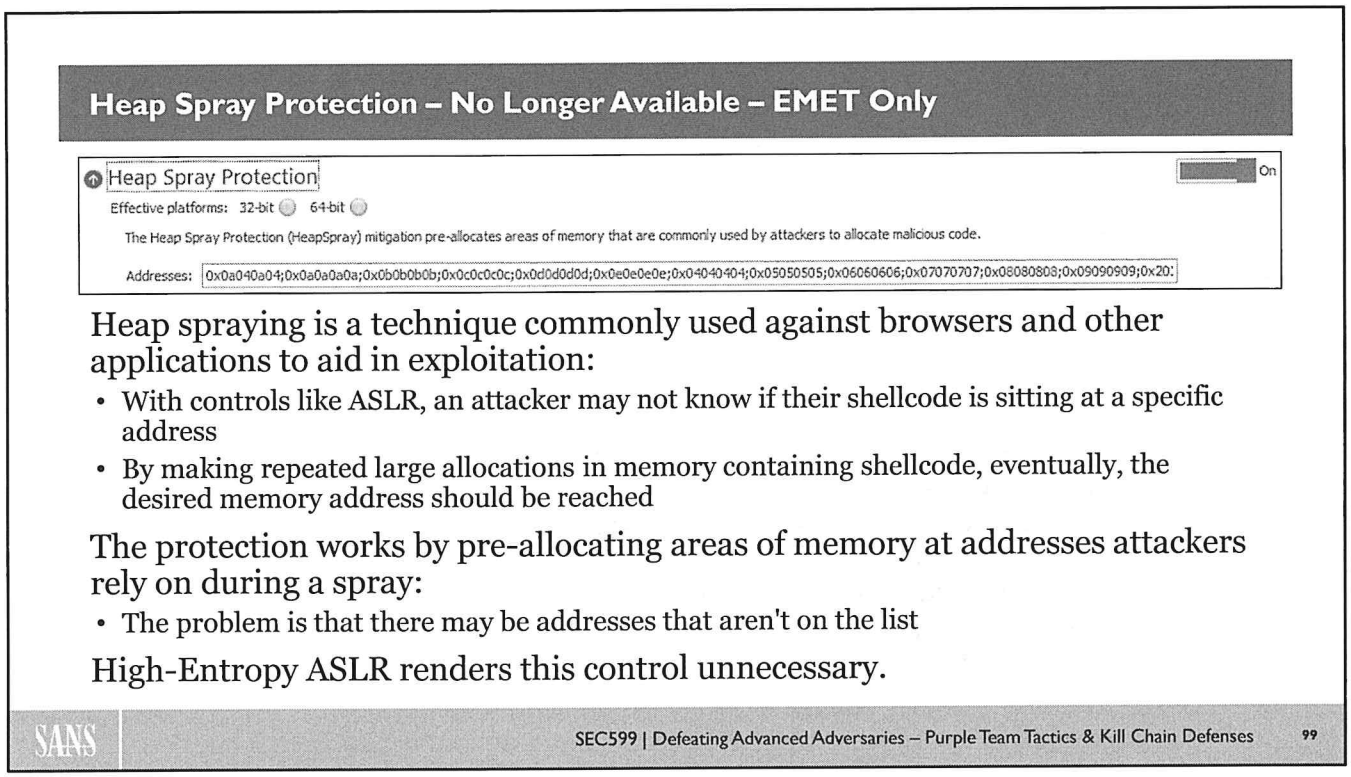


Exploit Guard Mitigations

As a defender, the phrase "Offense must inform the defense" often comes up, but what does that really mean? It is certainly subjective as to how it is to occur. From this author's perspective, to become proficient in a technical area, you must put in the time and do the work. You cannot wait for penetration testers, exploit developers, and malware experts to continuously provide information, and even if they could, do you have the prerequisite knowledge to understand what is being said? Take application debugging and reverse engineering as an example. Tools such as the Interactive Disassembler (IDA) and WinDbg are commonly used. They are unintuitive tools each requiring countless hours and practice. An exploit mitigation may be related to a very niche way in how low-level instructions are executed by the processor. Failure to have this prerequisite knowledge can limit your understanding as to the effectiveness. This paragraph is by no means an effort to discourage you. If anything, it should serve as a motivator to say that everyone who is an expert in areas such as malware reversing, and exploit development paid their dues. There are few corners available to cut.

The hope is that since Exploit Guard is integrated into Windows 10, that more companies will opt to use the tool. Many of the commercial anti-exploitation products out there also utilize the same types of controls. As more and more systems move to Windows 10, and as Windows 10 further evolves, it is highly likely that the controls once only available only in EMET will be turned on by default. We will not cover controls that we already covered, such as DEP and SEHOP.

© 2019 Erik Van Buggenhout & Stephen Sims



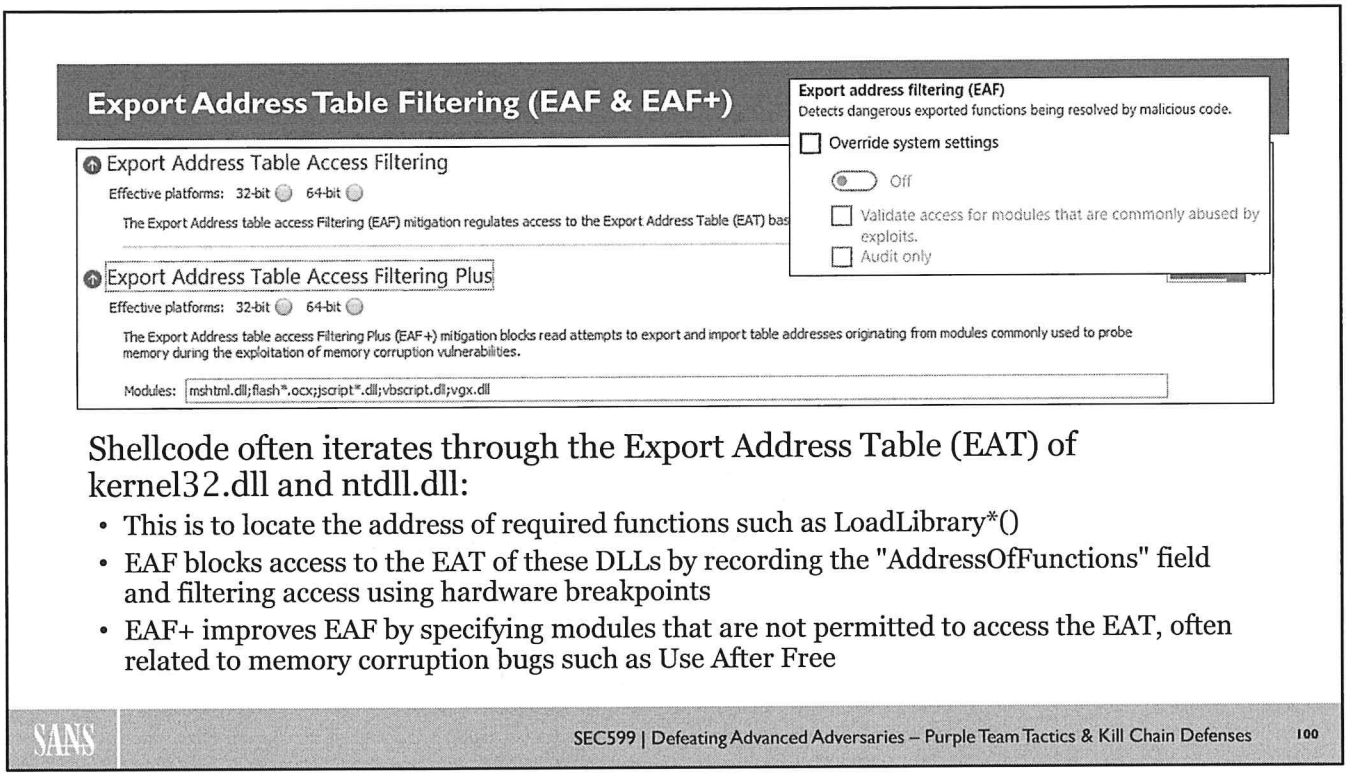
Heap Spray Protection – No Longer Available – EMET Only

Heap spraying is a technique first made public by the researcher Berend-Jan Wever, who goes by the handle "Skylined" as part of his "Internet Exploiter" exploit associated with CVE-2004-1050. The original exploit can be found at https://www.exploit-db.com/exploits/612/. Very little information about how the technique worked was released originally; however, Skylined recently released an article on heap spraying at https://blog.skylined.nl/20161118001.html.

On this slide is a screenshot from EMET. There are a couple of ways that heap spraying can aid during an exploit. One benefit is to help deal with ASLR and the difficulty in knowing if your shellcode will be at a predictable address. Imagine if you were in a small room. You can stand anywhere in the room, but you still only take up the same amount of space. Now, imagine if someone filled ¾ of the room with boxes, pushing you over to the remaining ¼, limiting the space available for you to stand. Now, imagine that it is not you standing there, rather, it is shellcode. If we fill up memory by repeatedly making large allocations that contain our shellcode, we will eventually fill up so much memory that we will have extended the region of memory down to a predictable address. Heap spraying also helps attackers during Use After Free (UAF) exploitation.

The protection works by pre-allocating the commonly used address that attackers rely on during a heap spray. On the slide, you can see an example of some of these addresses, such as 0x0b0b0b0b and 0x0c0c0c0c. The main issue with the protection is that there are many predictable addresses that can be used, so tracking all of them can be difficult to effectively manage. This control is no longer supported on the latest versions of 64-bit Windows 10 as the control High-Entropy ASLR (HEASLR) renders the control unnecessary due to the enormous amount of entropy with each run of a process.

© 2019 Erik Van Buggenhout & Stephen Sims



Export Address Table Filtering (EAF & EAF+)

The majority of shellcode for Windows relies on walking through the Export Address Table (EAT) of a DLL in order to resolve the location of its functions. A DLL is a library of functions available for use to applications. An application needs to know where inside the DLL a desired function is located. To make this easy, DLLs include an EAT. There is a field called "AddressOfFunctions" which is simply a pointer to an array of pointers, each pointing to the relative virtual address offset of the functions available for use by an application. EAF works by recording the "AddressOfFunctions" field and creating an exception handler. Hardware breakpoints are used when attempting to access the EAT of kernel32.dll and ntdll.dll. The exception handler created by EMET filters access via the hardware breakpoints, breaking access attempts by shellcode. Breakpoints are used by debuggers to pause execution when hitting a specific memory address or under a certain condition. Hardware breakpoints utilize debug registers built into the processor.

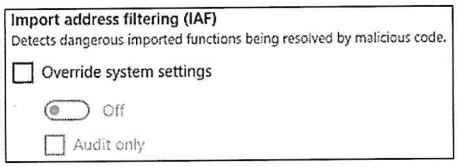
EAF+ improves the EAF protection by allowing you to specify modules commonly involved in memory corruption bugs such as Use After Free (UAF) and denying them from reading or writing to export and import address tables of modules such as ntdll.dll, kernel32.dll, and kernelbase.dll. Included by default, as shown above, is mshtml.dll, flash modules, and Visual Basic modules.

This control is a bit more unnecessary on 64-bit Windows 10 running Exploit Guard, as other controls, to be discussed, compensate. On the slide you can see the EMET version of the control on the left, with the Exploit Guard version on the top right. With the Exploit Guard version of EAF, the address of a CALL to a critical Windows function must come from within the program's code segment itself, and not from other locations such as the heap.

© 2019 Erik Van Buggenhout & Stephen Sims

Import Address Filtering (IAF)

The IAT (Import Address Table) is writable and used during dynamically linked function calls



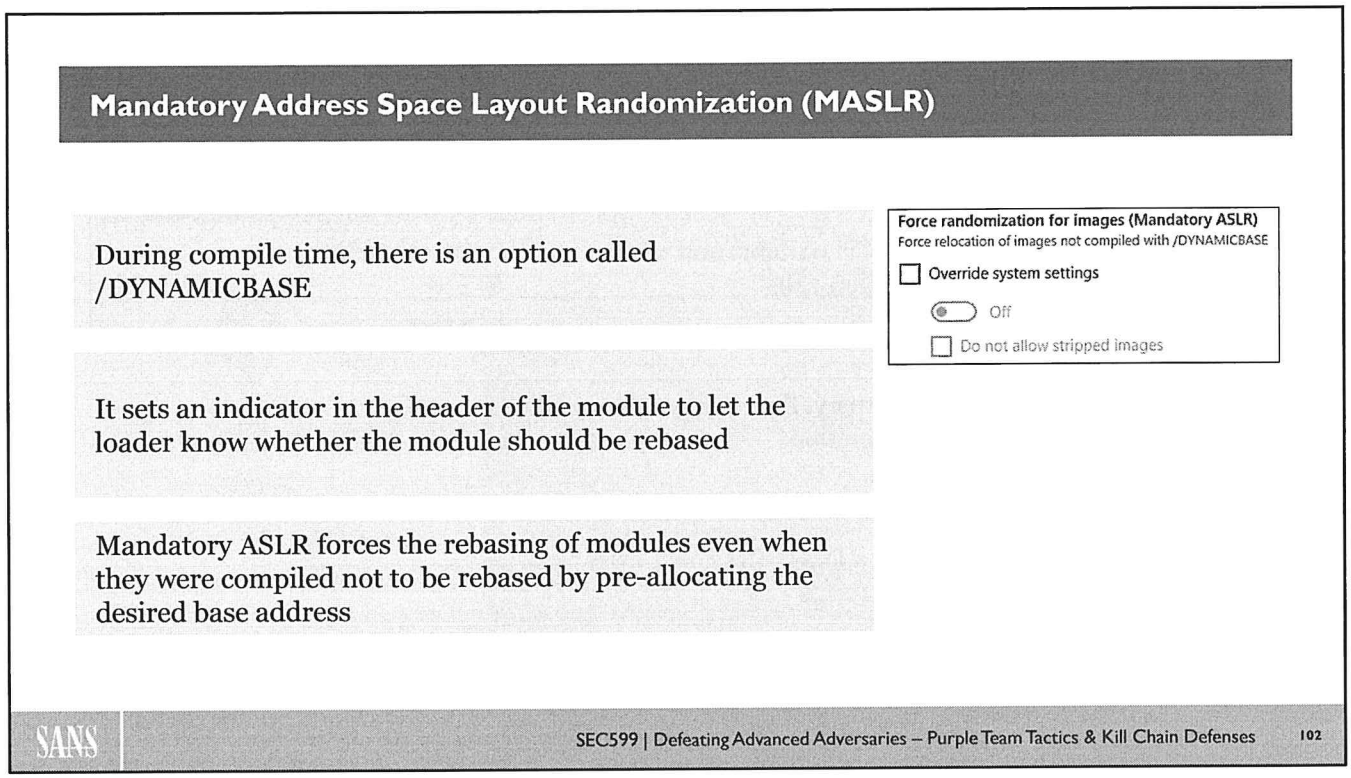
If an attacker can overwrite an entry, they can get their code called instead of the intended function

With IAF, all functions listed in a DLL's IAT must exist within the image's load address range

Import Address Filtering (IAF)

The Import Address Table (IAT) is similar to that of the Global Offset Table (GOT) on Linux. They store the resolved addresses of dynamically linked functions. A term known as "lazy linking" is often used to describe the way in which dynamically linked functions are resolved. It means that a function that is not statically linked into the program may not be resolved until it is needed. Regardless, these tables are writable. Since these tables are used to call functions in an indirect manner, an attacker could overwrite an entry, gaining code execution. Import Address Filtering (IAF) first checks functions being called to ensure that their addresses listed in the IAT reside within the memory allocated during the loading of the relevant module.

© 2019 Erik Van Buggenhout & Stephen Sims



Mandatory Address Space Layout Randomization (MASLR)

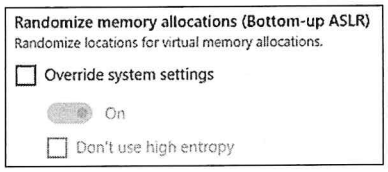
When compiling a DLL with Visual Studio, there is an option called /DYNAMICBASE. Remember, DLLs and modules are the same thing, so the words are used interchangeably. When compiled with this option, the header of the DLL is set with an indicator that it is to be rebased when loaded into a process. ASLR, as controlled by the OS, randomizes segments such as the stack and the heap, but DLLs are randomized separately and at a per-DLL level. It is often that an exploit mitigation control can be bypassed due to a single module not participating in a control. Mandatory ASLR (MASLR), also known as ForceASLR, mitigates this issue by forcing the rebasing of all loaded DLLs, regardless of the compiler option set. In theory, this should not cause an issue with an application; however, if there are static addresses used by the application, then a crash could occur. The issue of non-rebased modules is typically with the use of third-party applications that bring along custom modules to which the application is dependent.

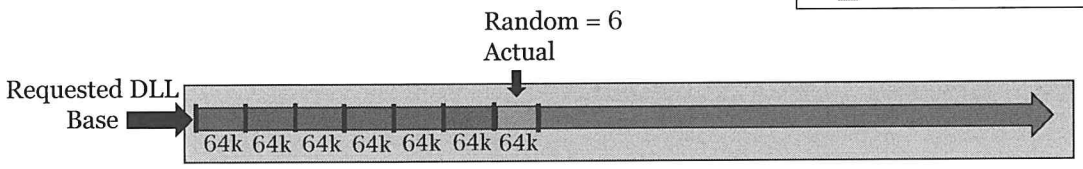
© 2019 Erik Van Buggenhout & Stephen Sims

Bottom-Up Address Space Layout Randomization (BASLR)

Works alongside of MASLR:

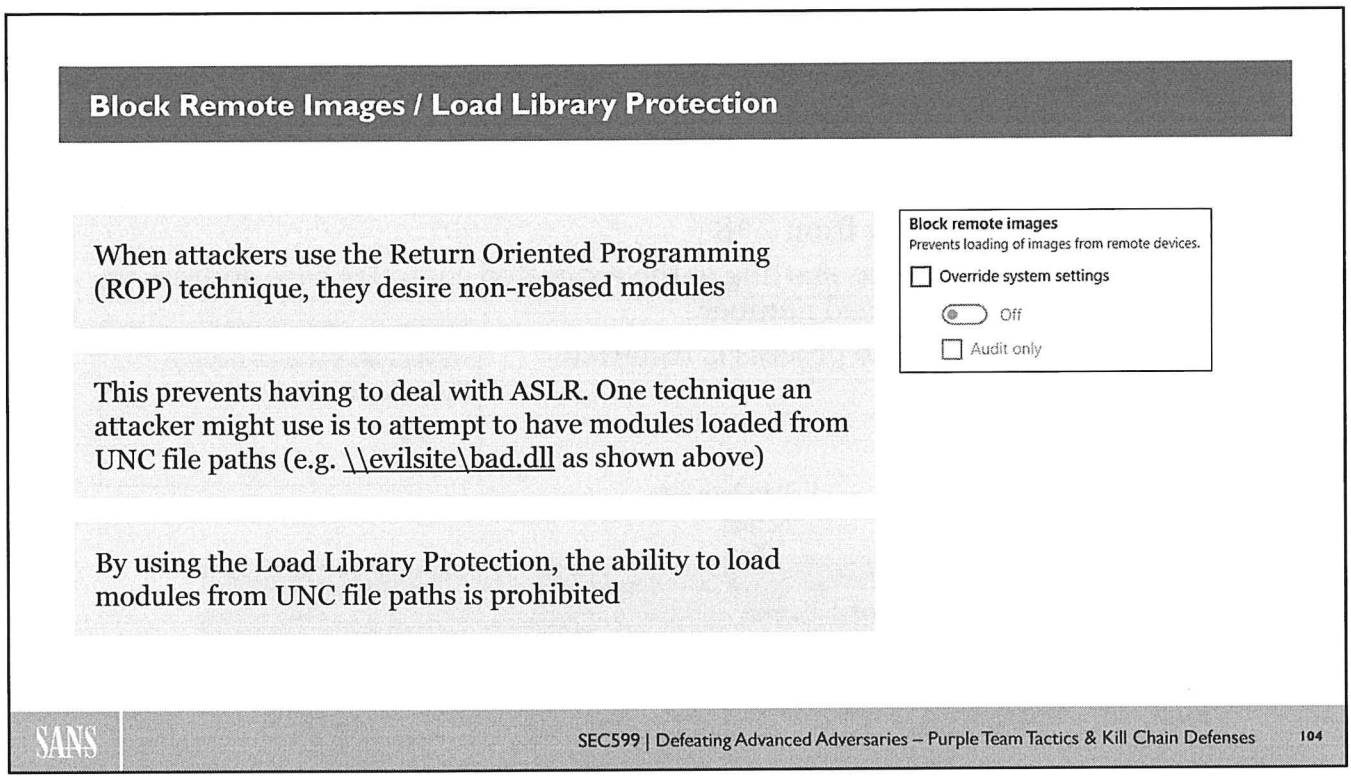
- Select a random number from 2^8
- Block all 64kb allocations starting at the requested compiler base address up until the randomly selected number
- Repeat this each time the process is restarted





Bottom-Up Address Space Layout Randomization (BASLR)
Mandatory ASLR blocks the requested compiler base address. It may be easy to determine where the rebase will occur in a repeatable manner as the next available base address is selected. BASLR improves the randomization by selecting a random number between [0, 256] and blocks that number of 64kb allocations from the compiler base address up until that point. This number will change with each process invocation, improving security.

© 2019 Erik Van Buggenhout & Stephen Sims



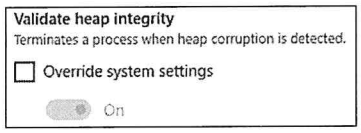
Block Remote Images / Load Library Protection

Exploit Guard has multiple controls focused specifically on mitigating Return Oriented Programming (ROP). Block Remote Images, also known as Load Library Protection, is one of these controls. The easiest way for an attacker to create an ROP chain to use in an attack is to have non-rebased modules inside the process from which they can use static addressing. An attacker can attempt to have the application load DLLs from across the network via a UNC file path. This can be leveraged to load modules, which contain code desired by the attacker. The Block Remote Images / Load Library Protection from Exploit Guard blocks modules from being loaded via UNC file paths.

© 2019 Erik Van Buggenhout & Stephen Sims

Validate Heap Integrity

Memory corruption bugs on the heap can be difficult to detect



The Validate Heap Integrity control performs behaviors similar to that already implemented by the Low Fragmentation Heap (LFH), such as:

- Randomizing the allocations from FreeLists
- Encoding chunk metadata in the headers

It also utilizes guard pages that should not be accessed

Validate Heap Integrity

In userland, the heap has a frontend allocator and a backend allocator. The frontend allocator used to be the Lookaside Lists. Starting with Windows Vista, the Lookaside Lists were no longer available to use, leaving us with the Low Fragmentation Heap (LFH). By design, the frontend allocators are used to service allocation requests where the size is often used. Take a browser as an example. As common HTML elements, such as a CButton or Span, are allocated, their size is always the same. If a threshold is met, LFH is triggered for that heap and services those allocations. This is used to improve performance. The LFH has controls in place to help increase the security of the heap. Well-known examples include randomizing allocations out of a FreeList, as well as encoding the first 32-bits of the chunk header, which serves as a security cookie or canary. The Validate Heap Integrity control carries on these types of protections to backend allocations as well. It also places guard pages onto the heap. Guard pages should never be accessed, or an exception is raised. If an attacker performs an overflow attempt that touches a guard page or attempts an arbitrary write to a guard page address, they will be caught.

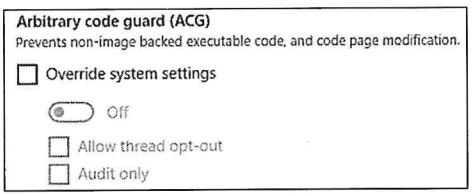
© 2019 Erik Van Buggenhout & Stephen Sims

Arbitrary Code Guard (ACG)

Formerly called MemProt on EMET

During a buffer overflow, an attacker will often place their shellcode into or just past the overflowed buffer with the hopes of execution:

- With DEP typically enabled, an attacker will often utilize ROP to call VirtualProtect() or VirtualAlloc() to change permissions on the stack
- ACG checks the destination address passed to a critical function to ensure it's not on the stack
- JIT code must be factored in, such as C#.NET
- Applications may require major changes



Arbitrary Code Guard (ACG)
Since the dawn of exploitation, it has been common for an attacker to overflow a buffer on the stack, put their shellcode into or past the overflowed buffer, and return control to this location to execute their payload. It is fairly standard for DEP to be enabled on modern OSes. Attackers typically use Return Oriented Programming (ROP) to call a function such as VirtualAlloc() or VirtualProtect() to change the permissions on the stack or other locations where their shellcode is located; otherwise, an exception would be thrown when trying to execute code in a write-only region. Formerly called MemProt from EMET, ACG works by evaluating the address passed to VirtualProtect(), VirtualAlloc(), or other similar functions to ensure that the execute permission is not being set on an existing or new allocation.

© 2019 Erik Van Buggenhout & Stephen Sims

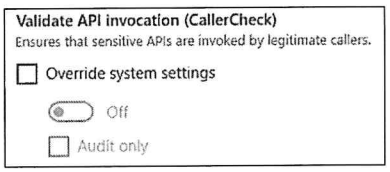
Validate API Invocation

The intended way for functions to be called is via the "Call" instruction

This instruction first pushes the return pointer onto the stack, so control can be passed back to the caller upon completion of the called function, and then redirects control to the called function

When attackers use ROP, they utilize the "Ret" instruction to jump to critical functions such as VirtualAlloc() and VirtualProtect()

The Validate API Invocation (formerly the Caller Check on EMET) control works by disallowing these critical functions to be reached via a "Ret" instruction



Validate API Invocation

In many processor architectures, there is a "Call" instruction. This is the intended way for functions to be called (e.g. call memcpy). It performs two operations. First, the address of the instruction after the "Call" instruction is pushed onto the stack, serving as the return pointer. This return pointer is used when the called function is finished, allowing for control to be returned to the calling function. The second thing the "Call" instruction does is it redirects control to the actual called function. When attackers utilize ROP as part of their exploit, they often rely on the "Ret" instruction to return to the start of critical functions such as VirtualProtect() or VirtualAlloc(). The Validate API Invocation control works by disallowing critical functions from being reached via a "Ret" instruction. This control was known as the Caller Check on EMET.

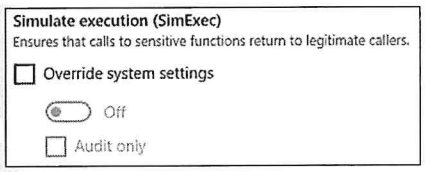
© 2019 Erik Van Buggenhout & Stephen Sims

Simulate Execution (SimExec)

Typically, when returning from a function, it will be some time before reaching another "Ret" instruction

Simulate Execution works by simulating the instructions after the return pointer address to look for the presence of a Ret

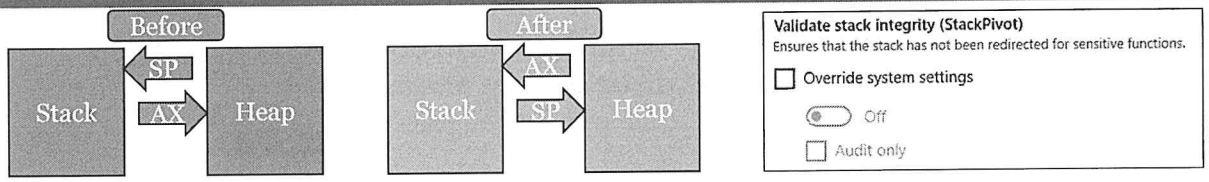
A number of instructions are simulated to look for ROP characteristics. EMET used 15 instructions by default



Simulate Execution (SimExec)
The Simulate Execution (or "SimExec") control is like the opposite of the Caller Check. The Call Check makes sure that we are reaching critical functions via a valid "Call" instruction. SimExec works by simulating a predetermined number of instructions (15 was the default on EMET) that exist, starting with the address to which the return pointer is pointing. It is looking for the existence of instructions commonly used with ROP, most often a "Ret" instruction. When returning from a function call such as VirtualProtect() it is typically a while before you would hit another "Ret" instruction; however, this is not always the case and has been known to cause issues with applications.

© 2019 Erik Van Buggenhout & Stephen Sims

Validate Stack Integrity / Stack Pivot Protection – XCHG RAX, RSP



During memory corruption exploits such as Use After Free (UAF), it is common to steal the stack pointer away from the stack and point it to attacker-controlled memory such as the heap.

- This is due to three special instructions unique to the stack pointer:
  - RET – Redirect execution to the address pointed to by the stack pointer
  - PUSH – Push the desired value onto the stack at the address held in the stack pointer
  - POP – Pop the value pointed to by the stack pointer into the designated register

Stack Pivot protection works by ensuring that the stack pointer points to the stack by checking the TIB (Thread Information Block) for stack limits.

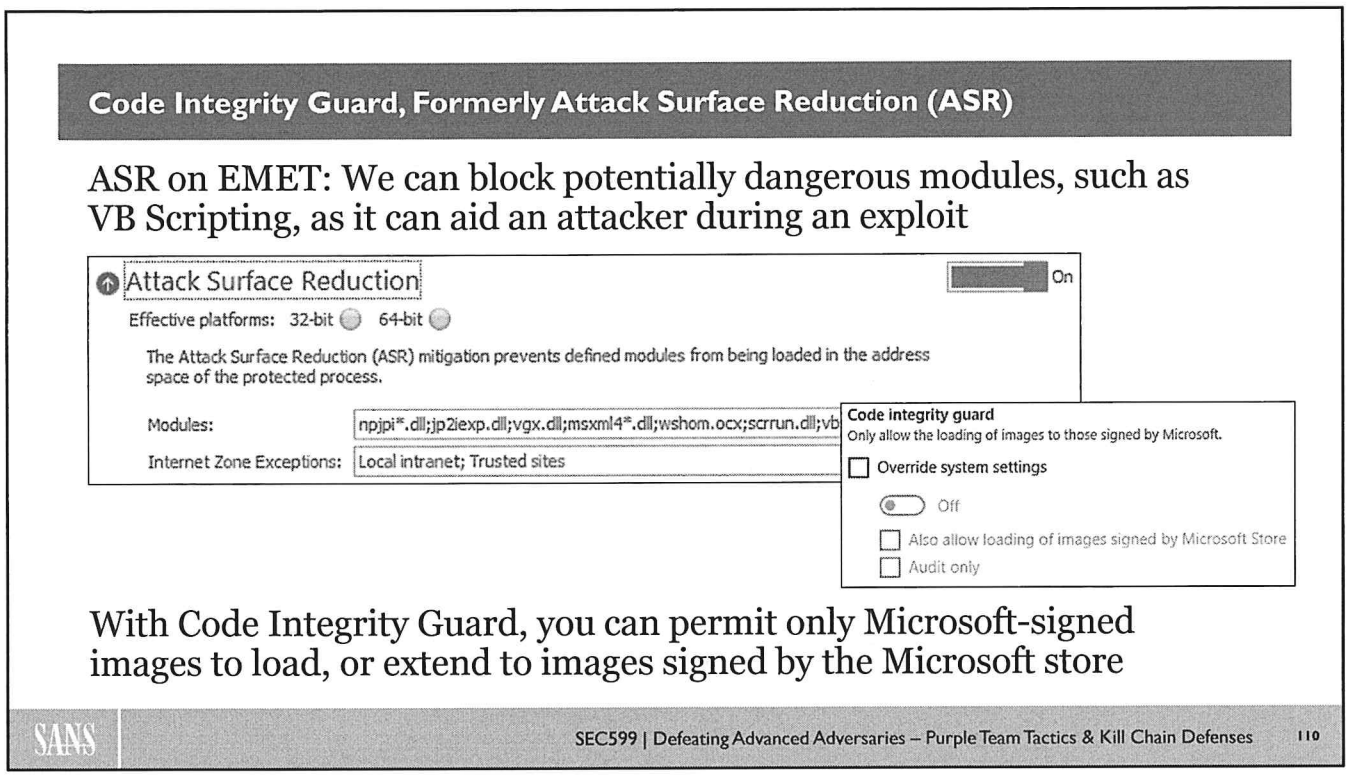
Validate Stack Integrity / Stack Pivot Protection – XCHG RAX, RSP

During memory corruption exploits such as Use After Free (UAF), a common technique is to steal the stack pointer away from pointing to the stack region. This is typically accomplished by using an instruction like "XCHG RAX, RSP". This would cause the EAX register to now point to the stack and the ESP register to point to a region such as the heap. This would be useful if the attacker controls memory on the heap and wishes to leverage unique and powerful instructions like "POP," "PUSH," and "RET" in relation to ROP. Registers are hardcoded variables integrated into the processor cores. They are used for arithmetic operations, storing addresses to memory locations, and many other purposes. Examples of registers include EAX, RIP, CR3, EFLAGS, ESP, R11, etc. The stack pointer is a register (ESP on 32-bit and RSP on 64-bit) that is designed to point to the top of the stack while under the context of a given thread. Those three special instructions are very useful to attackers.

RET – Redirect execution to the address pointed to by the stack pointer
PUSH – Push the desired value onto the stack at the address held in the stack pointer
POP – Pop the value pointed to by the stack pointer into the designated register

The Stack Pivot protection works by checking the stack addressing limits from within the TIB (Thread Information Block) to ensure that the stack pointer is pointing to a stack location.

© 2019 Erik Van Buggenhout & Stephen Sims



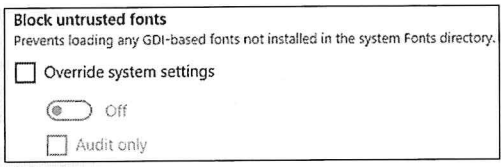
Code Integrity Guard, formerly Attack Surface Reduction (ASR)

There are quite a few modules that have been involved in many exploits over the years due to the functionality they provide. A couple of examples include vgx.dll (Vector Markup Language support), vbscript.dll (Visual Basic Scripting support), and jp2iexp.dll (Java plug-in). Attack Surface Reduction (ASR) allows you to specify any DLL you wish to never be loaded into a process. With Exploit Guard, we have Code Integrity Guard which replaces ASR. This allows you to limit the loading of modules to those signed by Microsoft. You can also extend it to images signed by the Microsoft store. It also ensures modules are not being loaded from untrusted locations, such as "Downloads."

© 2019 Erik Van Buggenhout & Stephen Sims

Block Untrusted Fonts

Requires that Graphical Device Interface (GDI) fonts be only loaded from the Windows Fonts directory



Much of the code related to the rendering and processing of fonts is done in Kernel mode

The infamous Duqu APT campaign is an example where a malicious font was used to compromise systems

Block Untrusted Fonts
Font protection is a rather simple mitigation to explain. If a process such as MS Word is running and a document indicates the desire for a font to be loaded outside of %WINDIR%/Fonts, it is not permitted. As with most mitigations, support can be turned on or off at a per-process level. As noted on the slide, the infamous Duqu APT, which targeted the Iranian nuclear program, utilized a 0-day font bug allowing for Kernel-level code execution.

Reference:
https://docs.microsoft.com/en-us/windows/security/threat-protection/block-untrusted-fonts-in-enterprise

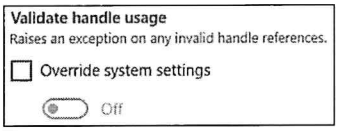
© 2019 Erik Van Buggenhout & Stephen Sims

Validate Handle Usage

The Validate Handle Usage control checks handle references to ensure they are valid

An example of using a handle is when a new process is created and needs to inherit the handle to a file descriptor or socket

If an attacker can modify the address of a handle, they may be able to run arbitrary code

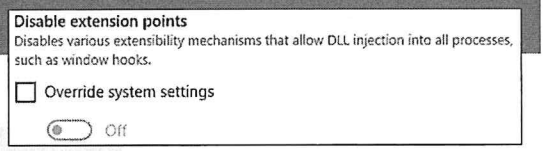


Validate Handle Usage

Handles are used as a way to pass resources within a process, between processes, and other scenarios. A handle may be of various types, such as file descriptors, sockets. STDIN/STDOUT, process IDs, and various others. Handles can be inherited or duplicated. If an attacker can modify the address of a handle and cause the handle to get inherited, they may be able to run arbitrary code. The "Validate Handle Usage" control checks to make sure that references to handles are valid. This can be performed by building a table of valid handles upon creation and ensuring that any references are listed in the table.

© 2019 Erik Van Buggenhout & Stephen Sims

Disable Extension Points



This control disables some ways in which applications can or could be extended or hooked over the years

A big example is with the AppInit_DLLs registry key where any DLLs listed would be loaded into each process upon invocation

There is no "Audit Mode" with this control



Disable Extension Points
An infamous attack technique used over the years is DLL injection. This is where we force a process to load a potentially malicious DLL containing an attacker's or malware's desired functionality. There are various ways in which the injection can be performed, such as that with hooking where you monitor a process for specific events. When one occurs, an action can be taken prior to passing it further onward down a hook chain. An example of an action that can be performed is the loading of a DLL. Another common example is the use of the AppInit_DLLs registry key. DLLs listed at this location are loaded into each process upon invocation. The "Disable Extension Points" mitigation blocks these techniques. As noted, there is no "Audit Mode" available with this control.

You can get more information on the AppInit_DLLs registry key from the following link:
https://support.microsoft.com/en-us/help/197571/working-with-the-appinit-dlls-registry-value

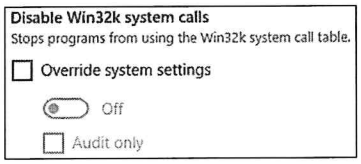
© 2019 Erik Van Buggenhout & Stephen Sims

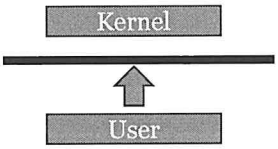
Disable Win32k System Calls

The Win32k System Call Table is full of functionality that runs under the context of System

Most applications do not need this ability... There are over 1,000 functions available, some of which previously being involved in vulnerabilities

This control greatly reduces the attack surface by blocking access to the Win32k System Call Table, but still allowing for NT-based system calls





Disable Win32k System Calls
This control prevents a process from being able to access the Win32k system call table. This is a large attack surface that has been known to have vulnerabilities from information disclosure to remote code execution. Most programs use the regular NT path of getting into the System context for privileged operations. The NT method typically involves using the SYSENTER instruction from within an NTDLL function. Without the "Disable Win32k System Calls" control applications can also utilize the Win32k system call table, which has over 1,000 functions that run from within the context of System. If a process does not need this capability, the control can be turned on, greatly reducing the attack surface.

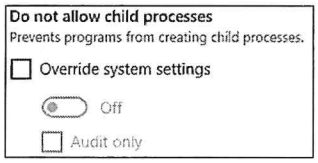
© 2019 Erik Van Buggenhout & Stephen Sims

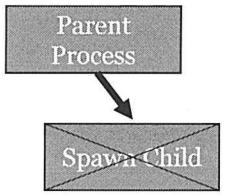
Do Not Allow Child Processes

A common goal of exploitation is to create a new process once the victim process is compromised

Often, even Proof of Concept code spawns the Windows Calc.exe program to prove success

This mitigation blocks the ability for a process to call the CreateProcess function

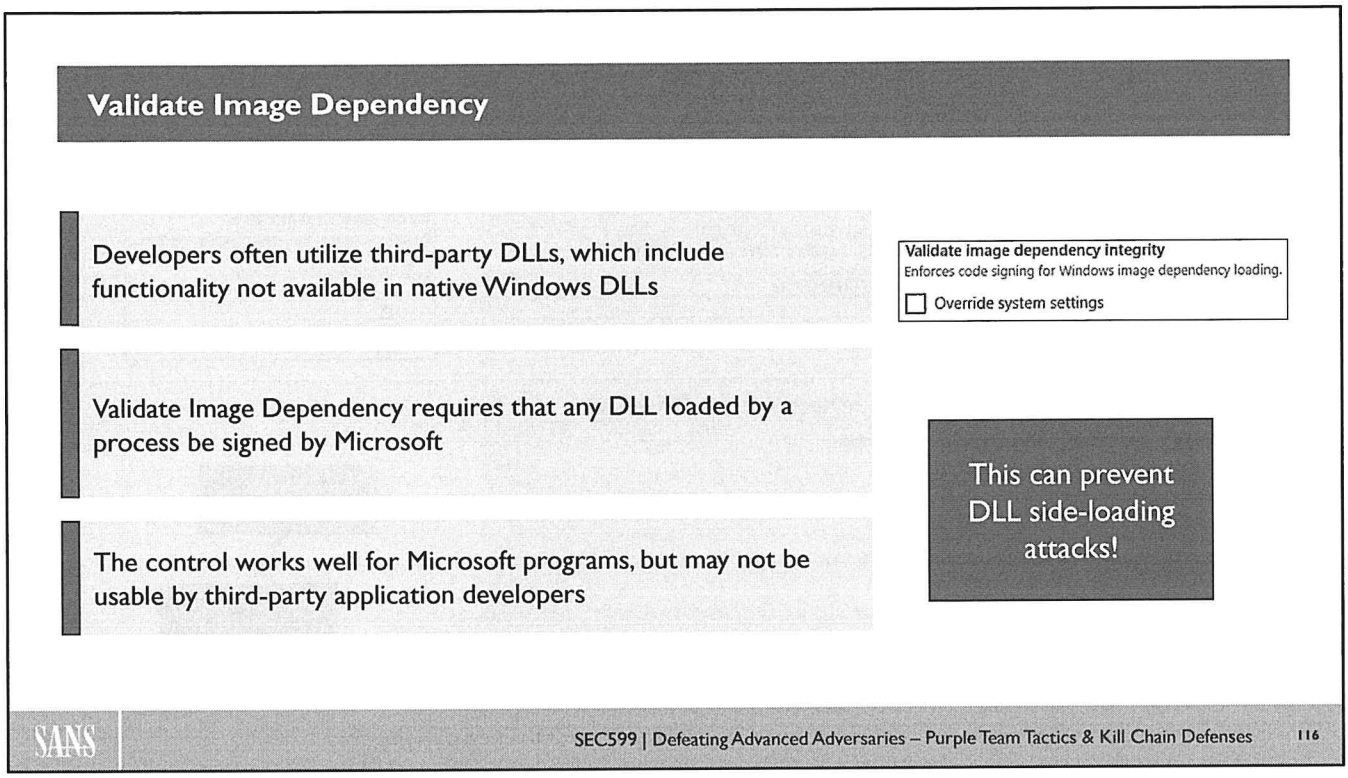




Do Not Allow Child Processes

The idea behind this control is simple. Block the ability for a process to spawn a child process using the CreateProcess function. It is not uncommon for an exploit to spawn a child process during exploitation to fulfill some goal. By preventing this capability, an attacker's options are more restricted, especially if you combine it with other controls that mitigate an attacker's ability to load modules into the compromised process.

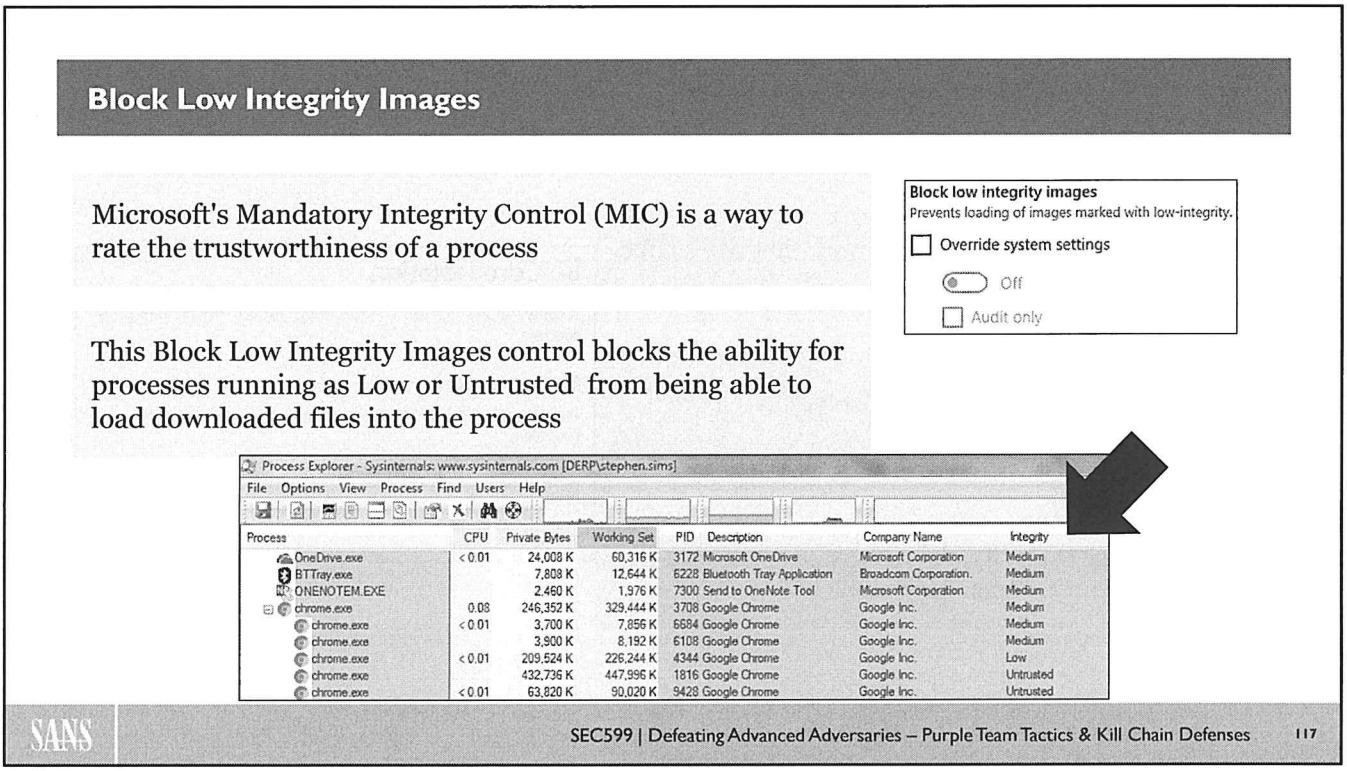
© 2019 Erik Van Buggenhout & Stephen Sims



Validate Image Dependency

DLLs are image files that contain functionality available to developers. Microsoft makes available to developers a large number of DLLs, and would prefer if only those DLLs are used. There are certainly cases where a third-party application developer may require functionality unavailable in any Microsoft DLL, or perhaps they need the behavior to differ. The "Validate Image Dependency" control mandates that all DLLs loaded into a protected process be digitally signed by Microsoft. If the DLL is not signed, it cannot be loaded into the process. This may not be suitable for all third-party applications and should be thoroughly tested. The positive thing about this control is that it can prevent DLL side-loading bugs from being exploitable. If a process goes to load a module that it cannot locate on the filesystem, an attacker could potentially trick a user into putting a malicious version of that DLL into one of the load locations. They typically would create a custom malicious DLL to perform some malicious actions. If the DLL is not signed by Microsoft, and the controls are on, the bug would not be exploitable.

© 2019 Erik Van Buggenhout & Stephen Sims

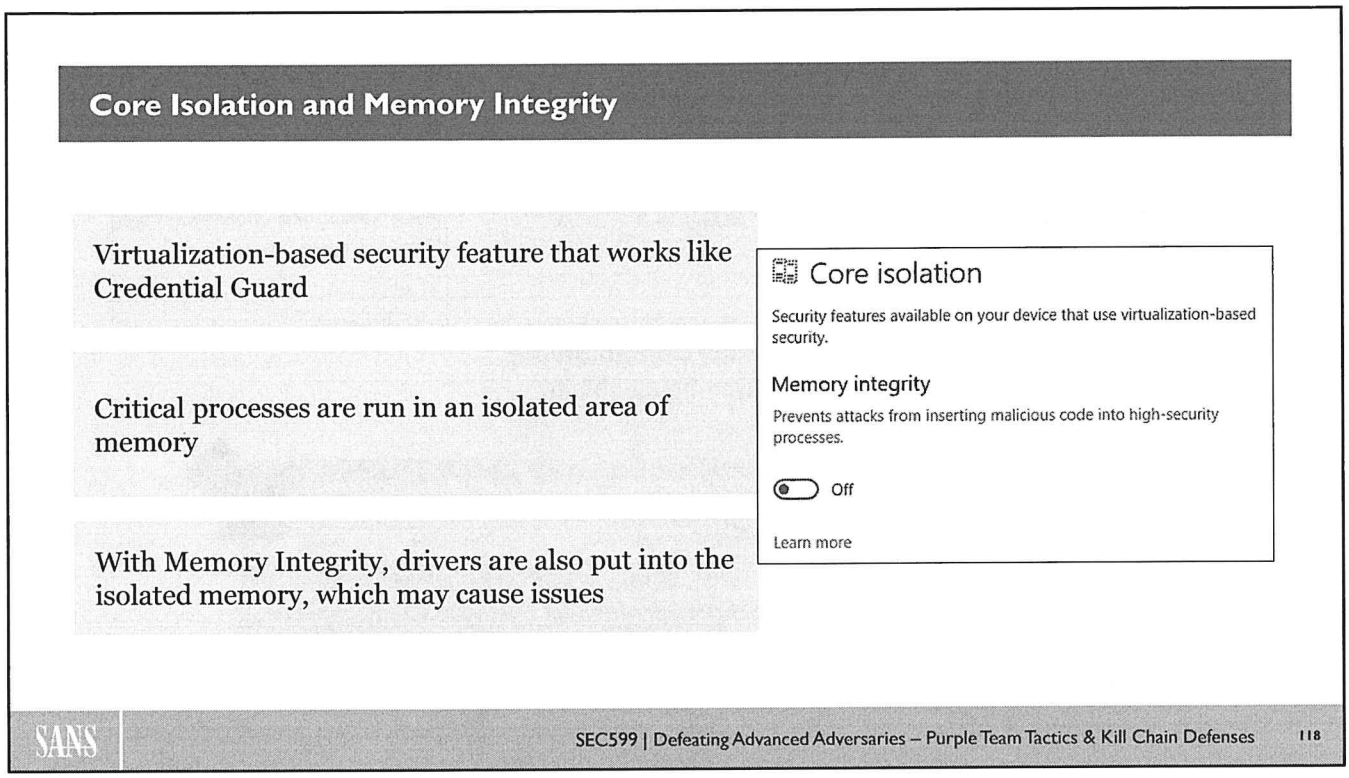


Block Low Integrity Images

Microsoft introduced Mandatory Integrity Control (MIC) with Windows Vista. It allows for an integrity level to be assigned to a process in order to increase the security around access control. Internet Explorer 7, which came with Vista as the default browser, could run in "Protected Mode" which utilized MIC. The browser would run with a low integrity level, preventing it from being able to make changes at a higher integrity level. As you can see from the screenshot of Process Explorer on the slide, to the right is the integrity column. Some processes are running as Medium, and others as Low or Untrusted. Google Chrome is running its browser windows as Untrusted, the lowest and most secure level. What the "Block Low Integrity Images" controls does is to prevent files that may have been downloaded by a process running with low integrity from being loaded into that process, further improving security.

For more information on Microsoft's Mandatory Integrity Control (MIC) see the following MSDN article: https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/bb625963(v=msdn.10)

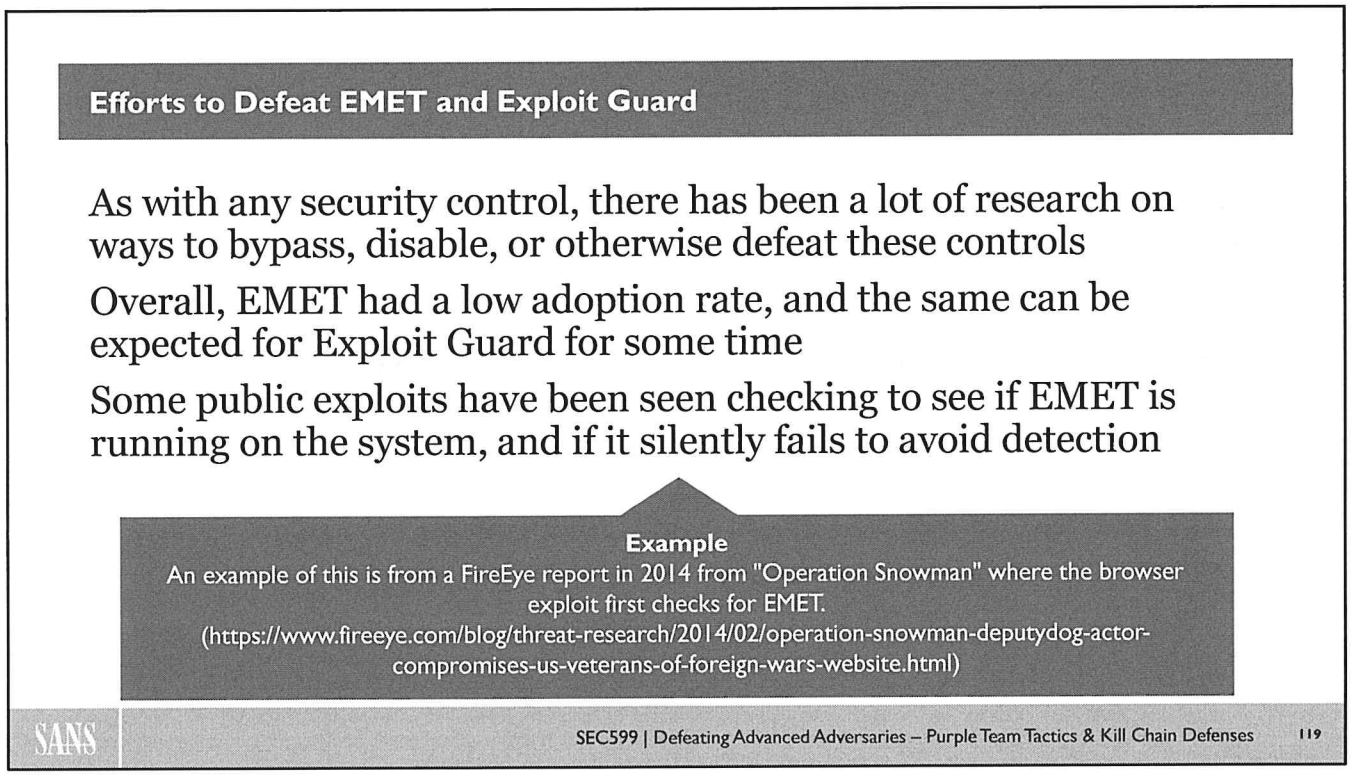
© 2019 Erik Van Buggenhout & Stephen Sims



Core Isolation and Memory Integrity
The Windows 10 Spring 2018 Creators Update brought us new controls called "Core Isolation" and "Memory Integrity." Core Isolation takes advantage of virtualization technology with Hyper-V, similar to how Credential Guard protects the LSASS process. Critical OS processes are placed into the protected area of memory, preventing tampering from being possible via traditional techniques. Memory Integrity adds on additional security by placing device drivers into this same protected region of memory. This is disabled by default as the control could negatively impact processes and prevent them from working properly.

Reference:
https://www.howtogeek.com/357757/what-are-core-isolation-and-memory-integrity-in-windows-10/

© 2019 Erik Van Buggenhout & Stephen Sims

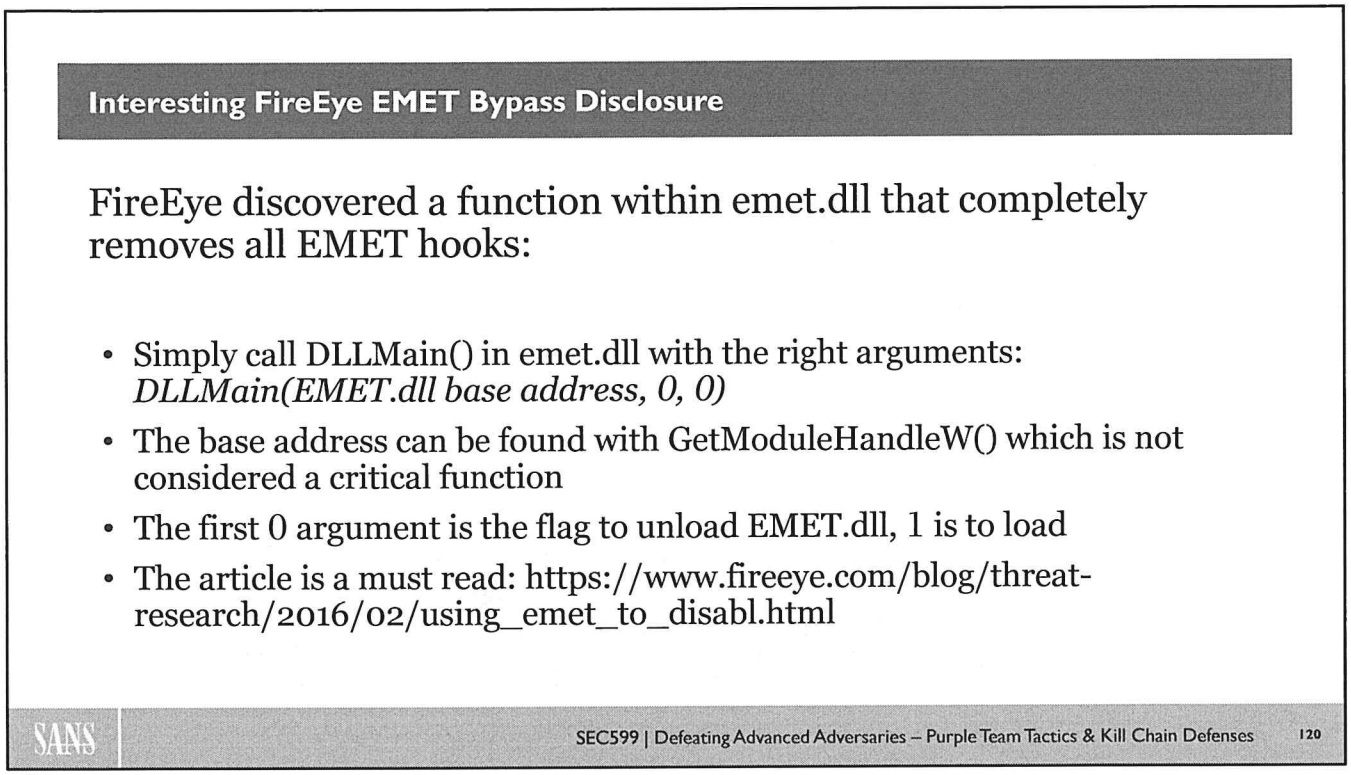


Efforts to Defeat EMET and Exploit Guard

Every time a new security feature or device is introduced into the wild, researchers, attackers, and others look for ways to defeat its controls. This is actually a good thing from a security perspective as too much trust has been given to vendors of security products, such as antivirus software. EMET had a relatively low adoption rate over the years due to numerous reasons. This is likely part of the reasoning for Microsoft's discontinuation of EMET in 2018. Many of the users and organizations using EMET are from interesting lines of work, including government, defense, critical infrastructure and others. This would also add to the draw of finding ways around the tool.

FireEye released a report in 2014 showing a browser exploit that first checked the victim to see if they had EMET running. If so, it silently fails to avoid detection. This was clearly an effort to keep the exploit unknown for as long as possible. You can check out the details here: https://www.fireeye.com/blog/threat-research/2014/02/operation-snowman-deputydog-actor-compromises-us-veterans-of-foreign-wars-website.html

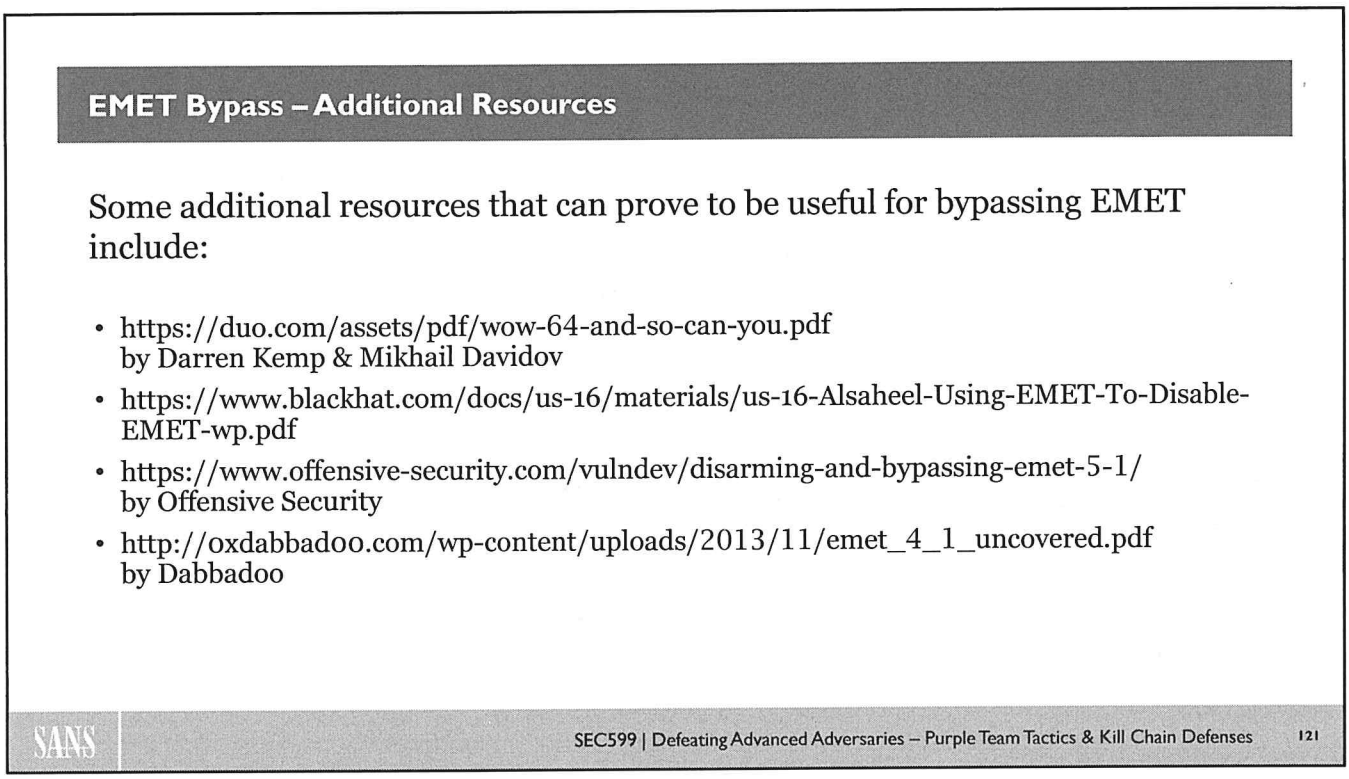
© 2019 Erik Van Buggenhout & Stephen Sims



Interesting FireEye EMET Bypass Disclosure
FireEye also released an interesting article on a bypass they discovered. To summarize, once you gain control of the process, you need to deal with EMET prior to attempting the disabling of DEP and execution of your shellcode. Much of the research has been quite complex in relation to methods to bypass EMET; however, this technique simply requires that you locate the base address of emet.dll and replay the DLLMain() function with an argument of 0 to unload all hooks. Pretty amazing.

Reference:
Alsaheel, Abdulellah. Pande, Raghav. "Using EMET to Disable EMET." FireEye Using EMET to Disable EMET. https://www.fireeye.com/blog/threat-research/2016/02/using_emet_to_disabl.html (accessed February 1, 2017).

© 2019 Erik Van Buggenhout & Stephen Sims



EMET Bypass – Additional Resources
Some additional resources that can prove to be useful for bypassing EMET include:

https://duo.com/assets/pdf/wow-64-and-so-can-you.pdf
by Darren Kemp & Mikhail Davidov

https://www.blackhat.com/docs/us-16/materials/us-16-Alsaheel-Using-EMET-To-Disable-EMET-wp.pdf
by FireEye

https://www.offensive-security.com/vulndev/disarming-and-bypassing-emet-5-1/
by Offensive Security

http://0xdabbad00.com/wp-content/uploads/2013/11/emet_4_1_uncovered.pdf
by Dabbadoo

© 2019 Erik Van Buggenhout & Stephen Sims

Malwarebytes

Malwarebytes is a commercial anti-malware product for Windows, Mac OS, and Android devices:



- A free version is offered with limited functionality, as well as a trial version

Protections are offered against malware, exploitation techniques, and ransomware.

For the purpose of this module, we are looking at the anti-exploit portion of the product:

- Similar to Microsoft's EMET and an alternative as EMET is EOL since 2018.
- Focuses on the most commonly exploited applications such as IE, Chrome, Office, Flash, etc.

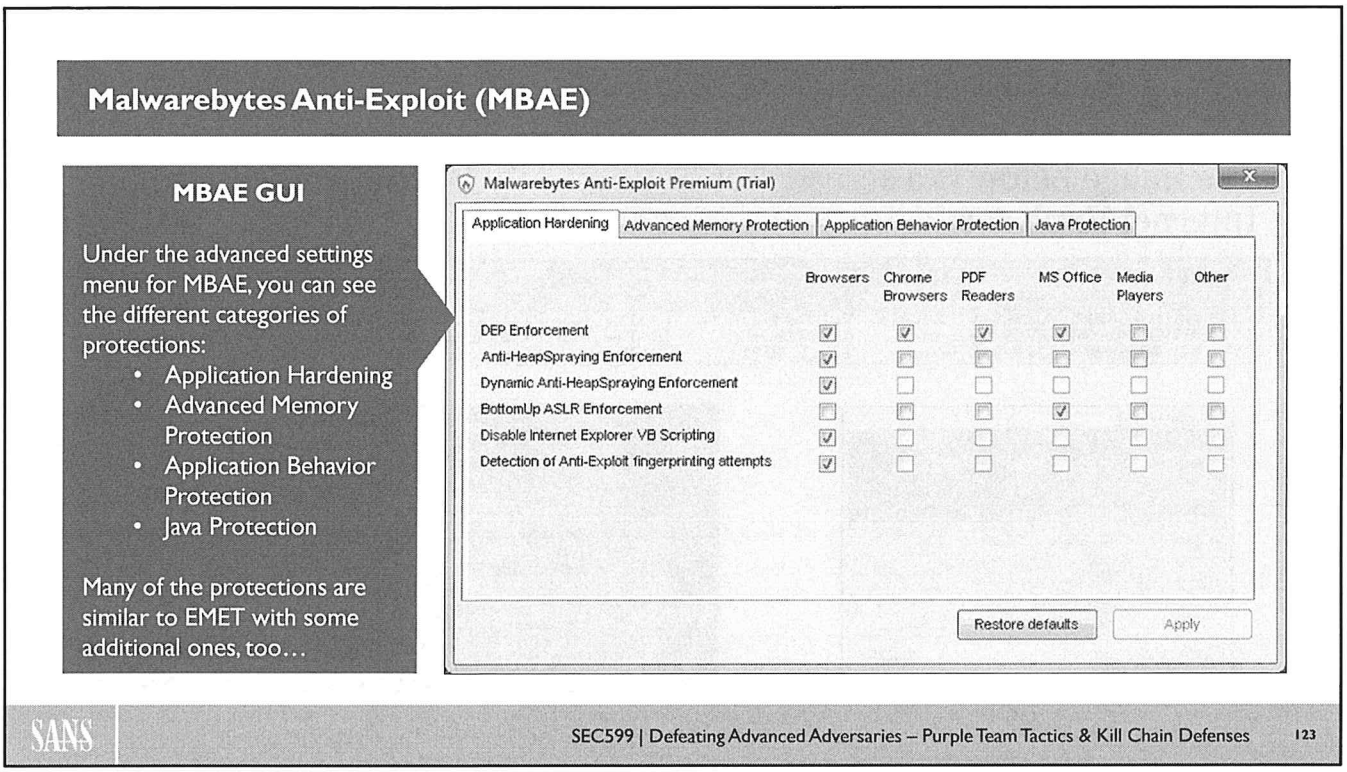
Malwarebytes
Malwarebytes Anti-Exploit (MBAE) is a commercial alternative to Microsoft's Enhanced Mitigation Experience Toolkit (EMET), which as stated previously, is end of life as of mid-2018. Organizations might still run Windows 7, which justifies the case to consider alternative products. Malwarebytes has been around for over 10 years initially offering only anti-malware type features, such as the removal of Spyware and Adware, as well as identifying common infections through scanning. As the product evolved, real-time scanning was incorporated, as well as anti-exploitation functionality like EMET and Ransomware protection. A free version is available once the 14-day trial expires offering anti-malware and anti-spyware protection, as well as rootkit detection, as stated on their website at https://www.malwarebytes.com/mwb-download/.

For the purposes of this module, our attention is focused on the anti-exploitation functionality. To try and simplify configuration and focus on the most common targets involved in exploitation, MBAE focuses on browsers, Flash, MS Office Suite, PDF readers, and media players.

References:
Malwarebytes. "Malwarebytes Endpoint Security." MBAEBGuide.pdf https://www.malwarebytes.com/pdf/guides/MBAEBGuide.pdf (January 26, 2017).

Malwarebytes copyrighted image taken from https://plus.google.com/+Malwarebytes

© 2019 Erik Van Buggenhout & Stephen Sims



Malwarebytes Anti-Exploit (MBAE)
When looking at the trial version of MBAE, you can go to the "Advanced settings" menu from the control panel to bring up the image on the slide. The tabs at the top include "Application Hardening," "Advanced Memory Protection," Application Behavior Protection," and "Java Protection." Many of the controls should look familiar as they are similar to Exploit Guard and EMET, including "DEP Enforcement," "Anti-HeapSpraying Enforcement," "BottomUp ASLR Enforcement" and many others, some not offered by EMET. The techniques behind the controls are very similar, if not identical to Exploit Guard and EMET. MBAE and Exploit Guard or EMET should not be running on the same system as both will inject a DLL into the applications chosen for protection when started and attempt to apply the same types of hooks. This will likely end badly.

MBAE has additional controls over Exploit Guard and EMET focusing on Macro abuse of WMI and Visual Basic for Applications (VBA), as well as Java protections focused on common payloads such as Meterpreter.

Check out the following reference for more information:
https://www.malwarebytes.com/pdf/guides/MBAEBGuide.pdf

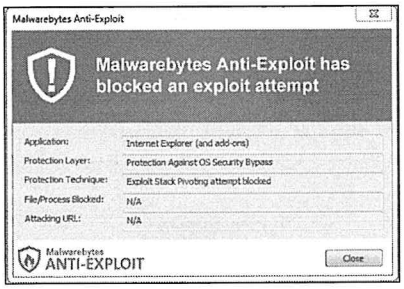
© 2019 Erik Van Buggenhout & Stephen Sims

Looking at MBAE with Immunity Debugger

The image below is a screenshot of Immunity Debugger attached to Internet Explorer:

Executable modules

Base	Size	Entry	Name	File version	Path
7C340000	00056000	7C34229F	MSVCR71	7.10.3052.4	C:\Program Files\Java\jre6\bin\MSVCR71.dll
6D730000	0004F000	6D74821B	ssv	6.0.340.4	C:\Program Files\Java\jre6\bin\ssv.dll
689B0000	00064000	689D9A18	mbae	1.9.1.1291	C:\Program Files\Malwarebytes Anti-Exploit\mbae.dll



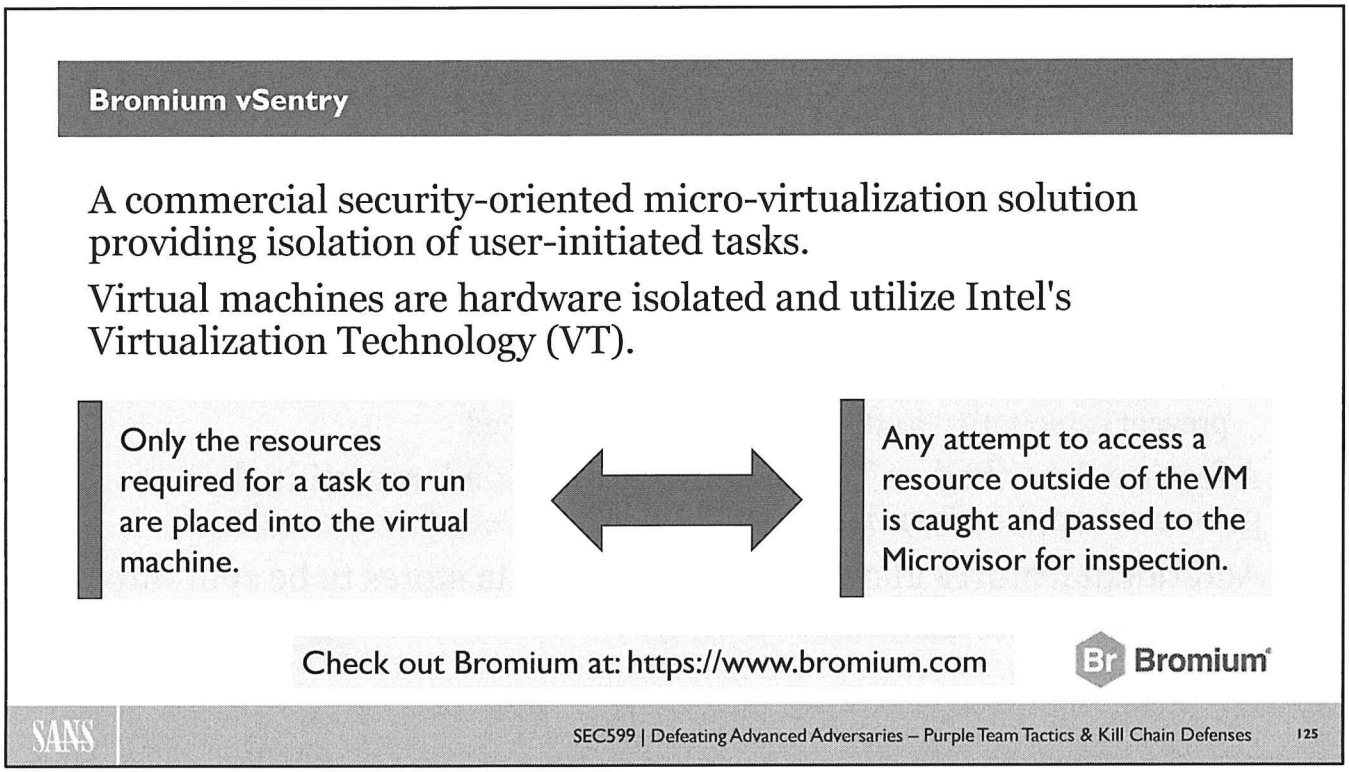
Process injection

The arrow on the right is pointing to the mbae.dll module that was injected into the process. This is the exact behavior of EMET with the emet.dll module.
If both emet.dll and mbae.dll were in the process at the same time, they would likely be fighting for the same control. When trying to run IE with both running, IE failed to start. ☺

Looking at MBAE with Immunity Debugger
On this slide is a screenshot from an Immunity Debugger session attached to Internet Explorer. MBAE is installed on the system and you can see that the module mbae.dll is injected into the process as indicated by the arrow. This behavior is identical to Exploit Guard and EMET. If both PayloadRestrictions.dll or emet.dll and mbae.dll were loaded into this process at the same time, the would likely be fighting for the same control. As a test to see what would happen, this author ran EMET, protecting Internet Explorer, and also MBAE. The process failed to start after several attempts with no alerts from EMET or MBAE and no logs shown in Windows Event Viewer.

Furthermore, the alert box shows what appeared after running a browser exploit on a system protected by MBAE that uses the stack pivoting technique covered earlier.

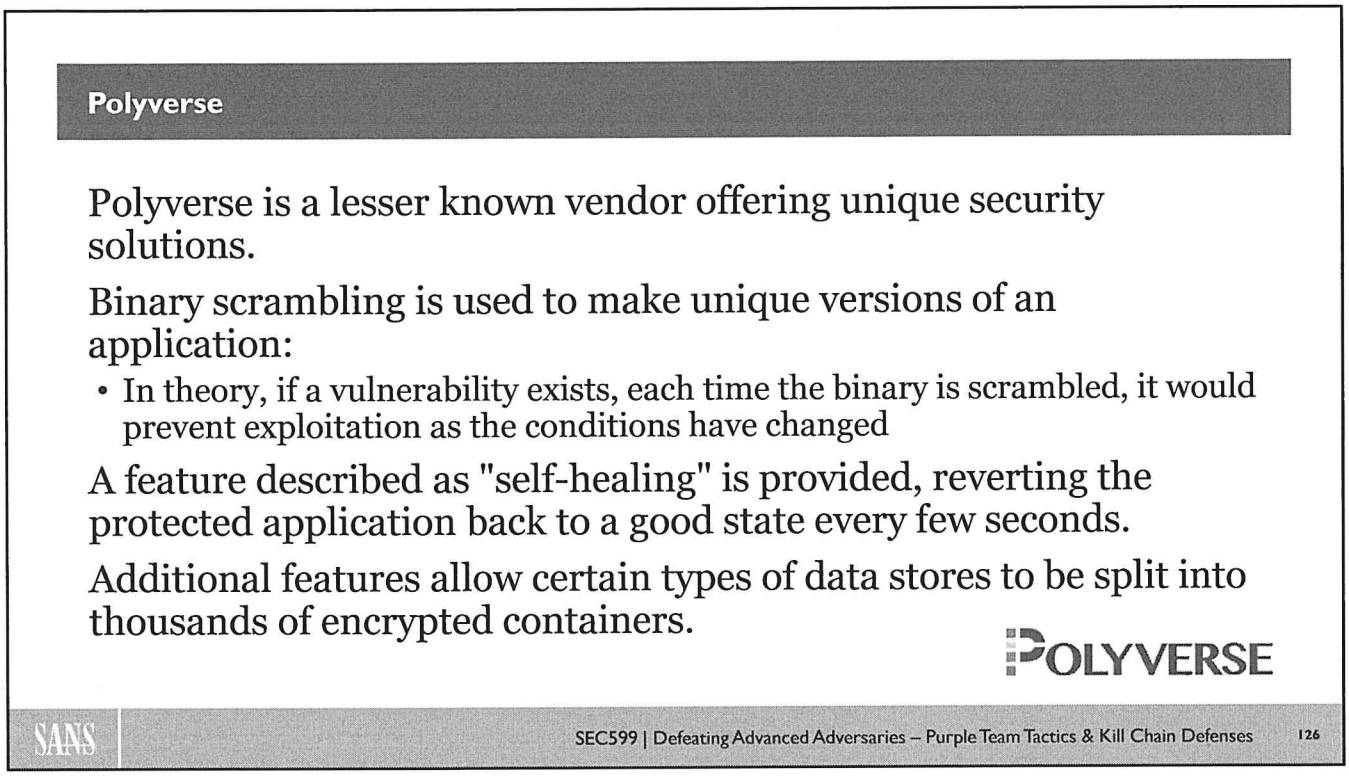
© 2019 Erik Van Buggenhout & Stephen Sims



Bromium vSentry
A couple of commercial security solutions offer micro-virtualization where portions of the operating system or processes are contained within their own virtual machine, preventing wide-scale system access. Bromium is a vendor offering a product called vSentry which isolates user-initiated tasks using Intel's Virtualization Technology (VT). Each task is provided with only the resources necessary to properly run. When a task attempts to interact with another task or anywhere outside of its own VM, hardware interruptions occur, and the request is passed to the Microvisor for inspection and application of a set of mandatory access controls.

References:
Bromium Secure Platform | https://www.bromium.com/our-tech/bromium-secure-platform/
Bromium copyrighted image taken from: https://www.bromium.com/

© 2019 Erik Van Buggenhout & Stephen Sims



Polyverse
Another contender in the space of vendors such as Bromium is Polyverse. There is not too much publicly available information about the internal technology of their product; however, unique features include the concept of binary scrambling and "self-healing." You may be familiar with the idea of polymorphic malware. This is malware that attempts to evade detection by constantly changing so that signatures go unmatched. Binary scrambling takes advantage in a similar fashion in that the binary is changed from its original state. If you are familiar with assembly code, you know that there are many ways for instructions to achieve the desired result.

Let's say we want the x64 RAX processor register to hold a value of 0x40. In assembly, we have several ways to accomplish this goal. The following is a simple example:

```
e.g. 1
xor rax, rax                    # Zero out the RAX register
mov al, 0x40                    # Move 0x40 into the lower byte of the RAX register (al stand for
accumulator low)

e.g. 2
mov rax, 0xffffffc0             # Move 0xffffffc0 into the RAX register
neg rax                         # Compute the two's complement of the value stored in the RAX register,
resulting in 0x40
```

Another feature offered by Polyverse is what they describe as "self-healing." This is simply the application reverting back to a known good state every few minutes. Additional advanced features offer the splitting of a data store such as a database into thousands of encrypted containers. This is all definitely a technology to keep an eye on.

References:
Polyverse copyrighted image taken from https://polyverse.io/

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

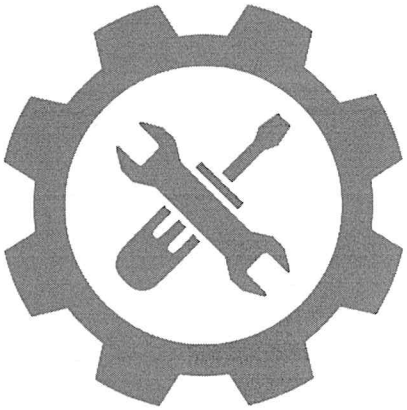
Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Exercise: Exploit Mitigation Using ExploitGuard



Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

Protecting applications from exploitation

- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

Avoiding installation

- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control

- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Persistence?

Upon successful exploitation, adversaries typically want to **persist** their access on the target environment (e.g. to survive reboots, user logoff, ...) There's two main categories of persistence strategies:

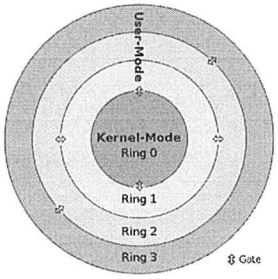


Image Source: Wikipedia

User space

- "Hiding in plain sight"
- Do not always require administrative access to system
- Examples: Web shells, scheduled tasks, user profile, ...

Kernel

- Has additional capabilities to hide itself from investigations
- Require administrative access to system
- Examples: Device drivers, Loadable kernel modules, ...

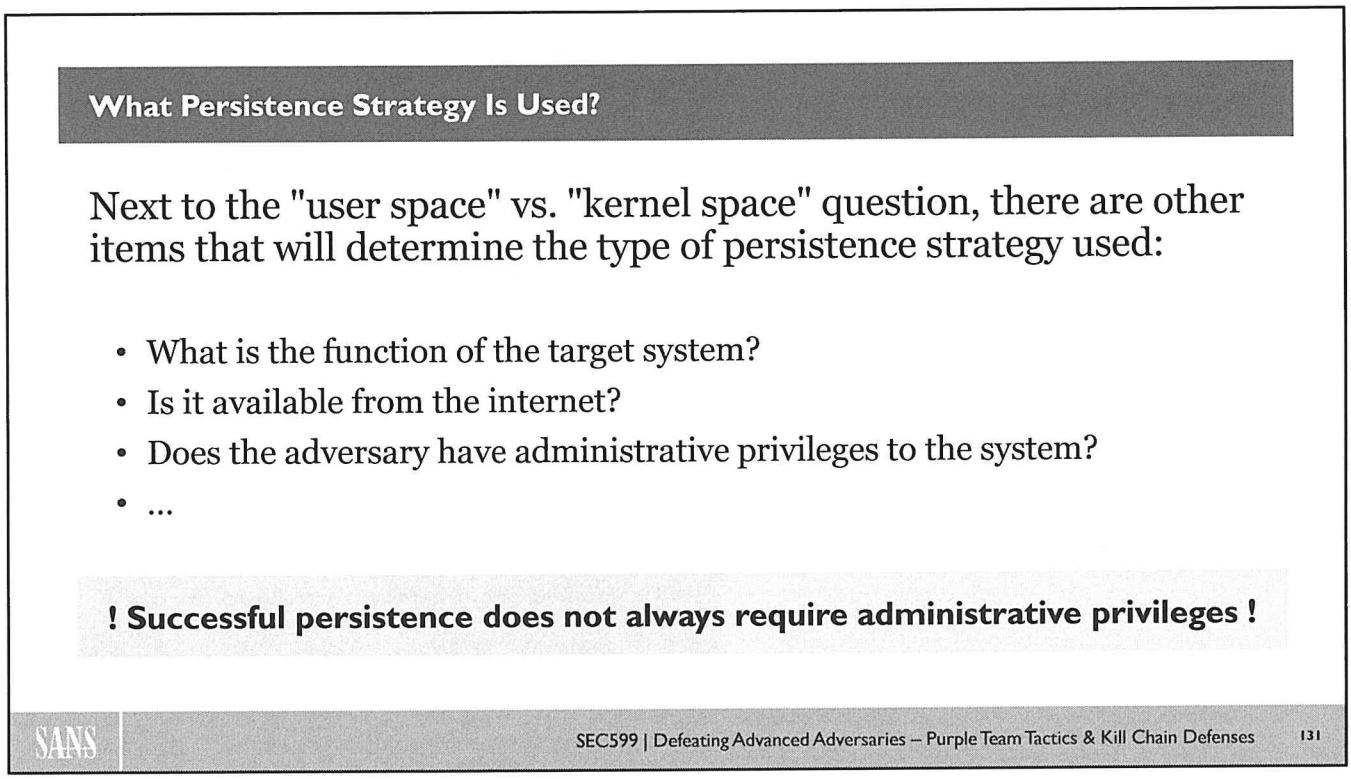
Both user space & kernel space persistence strategies are used by APTs!

Persistence?
Upon successful exploitation, adversaries typically want to persist their access on the target environment (e.g. to survive reboots, user logoff, ...). Depending on the privileges available to the adversary, they could choose to hide in two main parts of the victim system:

- User space: User space is the memory area where application software (and a limited number of drivers) execute. All "user" interactions typically occur in user space.
- Kernel space: Kernel space is strictly reserved for running a privileged operating system kernel, kernel extensions, and most device drivers. Code used for persistence in kernel-mode is typically referred to as a "rootkit", as it interacts with low-level parts of the OS and can thus hide itself better from investigations.

Advanced adversaries have been known to use both locations for persistence. You might think they would always prefer kernel-mode persistence; however, this is not true: The very presence of an unknown item in kernel space (e.g. a "rootkit") provides a signal something is suspicious... They might thus prefer to compromise the user environment and thus, limit themselves to "hiding in plain sight". Adding a scheduled task, setting a "Run" registry key, adapting the user profile, ...

© 2019 Erik Van Buggenhout & Stephen Sims



What Persistence Strategy Is Used?
We already discussed the "user space" vs. "kernel space" question in the previous slide. There are, however, a number of other questions that will determine what type of persistence strategy is opted for by the adversary:

- What is the function of the target environment?
  If it's a workstation, we can assume that reboots and user logons will occur frequently and thus a persistence strategy related to the user profile could be a viable option.
- Is it available from the internet?
  If it is, the adversary could opt to place a backdoor in an existing service (e.g. web shell) and use that to directly reconnect to the system.
- Does the adversary have administrative privileges to the system?
  If administrative privileges are available, the adversary could opt to attempt installing a rootkit that operates in kernel-mode.

Even if administrative privileges are helpful to the adversary, persistence can be achieved without having them as well!

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

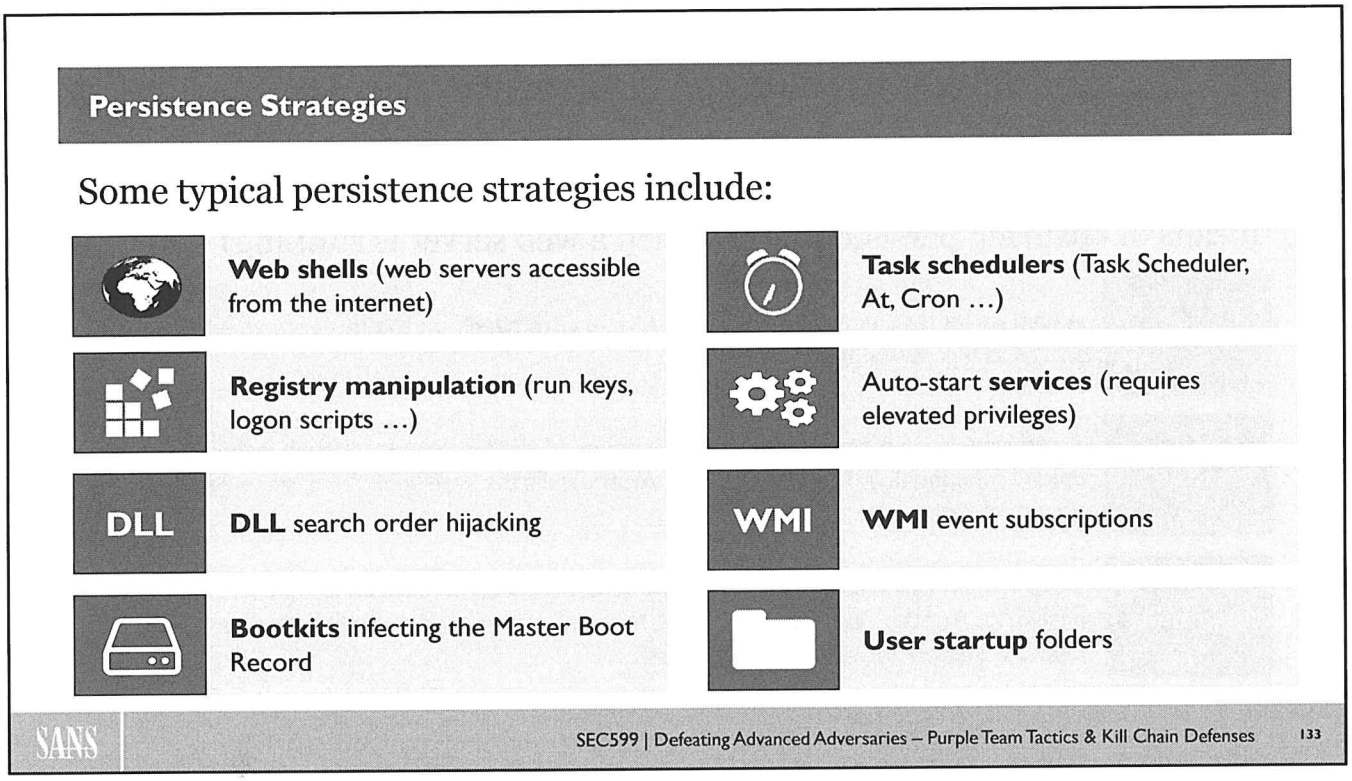
Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

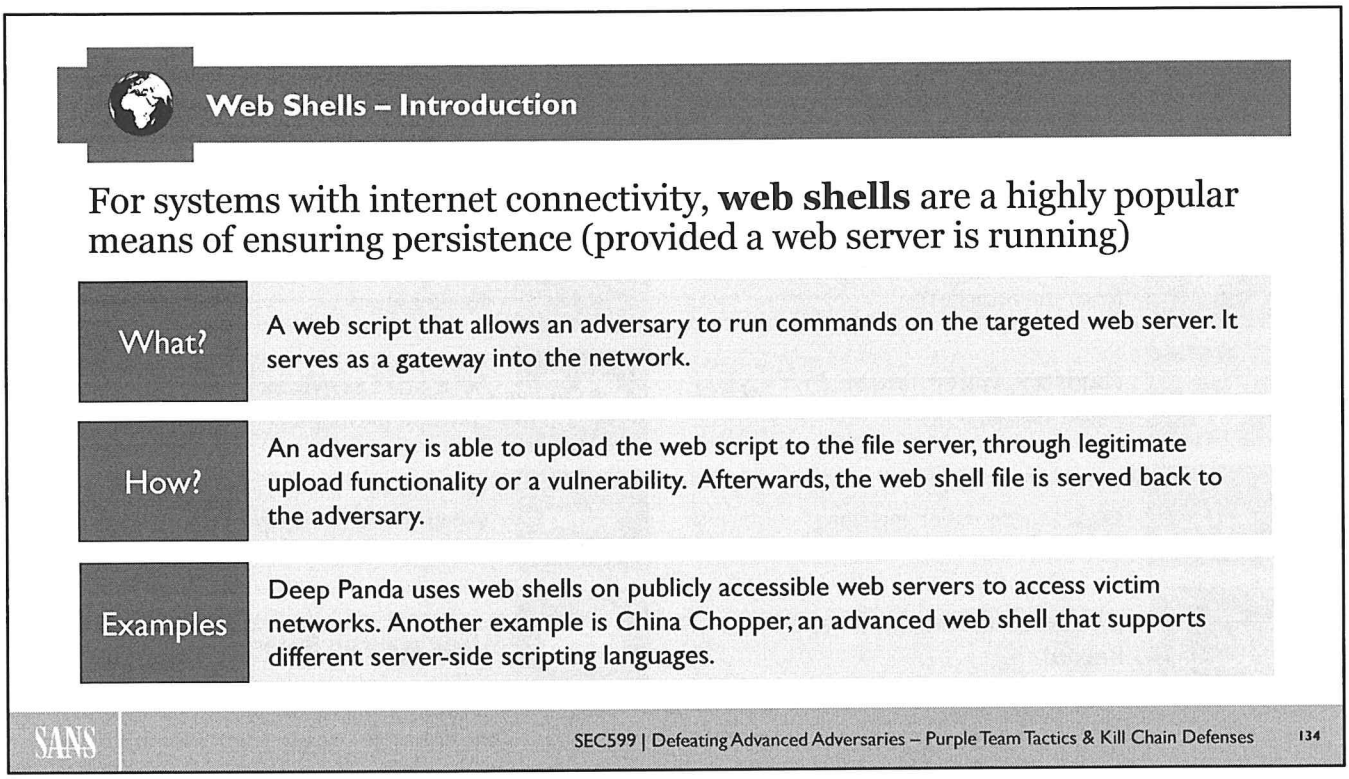


Persistence Strategies
A number of common persistence strategies are listed below. These will be explained in more detail in the following slides.

- Web shells: Only useful for internet-accessible systems. Often used as an initial entry point in an environment.
- Task schedulers: All major operating systems have task schedulers available (Task Scheduler, At, Cron, …). A highly popular means of persistence.
- Registry manipulation: A number of registry keys are used during the startup / user logon process. These can be abused by adversaries to add malicious code to be executed upon start-up / logon.
- Auto-start services: Many systems use a number of services that will automatically launch at startup (often even with elevated privileges). Adversaries could add additional services that appear to have normal names or functions (will require administrative privileges);
- DLL search order hijacking: Highly interesting technique abusing the way Windows prioritizes the loading of DLLs;
- WMI event subscriptions can be used to persist payloads on Windows-based systems;
- User startup folders: Much like registry manipulation, adversaries could implement shortcuts / scripts in a user's startup folder, which is executed upon logon;
- Bootkits are used to affect the system upon startup. Malicious code that infects the Master Boot Record (MBR) will run even before the Operating System is launched.

We will analyze all of these techniques in the upcoming section! Please note that this overview covers some of the most popular methods, but it's certainly not exhaustive... Check out MITRE's ATT&CK framework for additional techniques!

© 2019 Erik Van Buggenhout & Stephen Sims



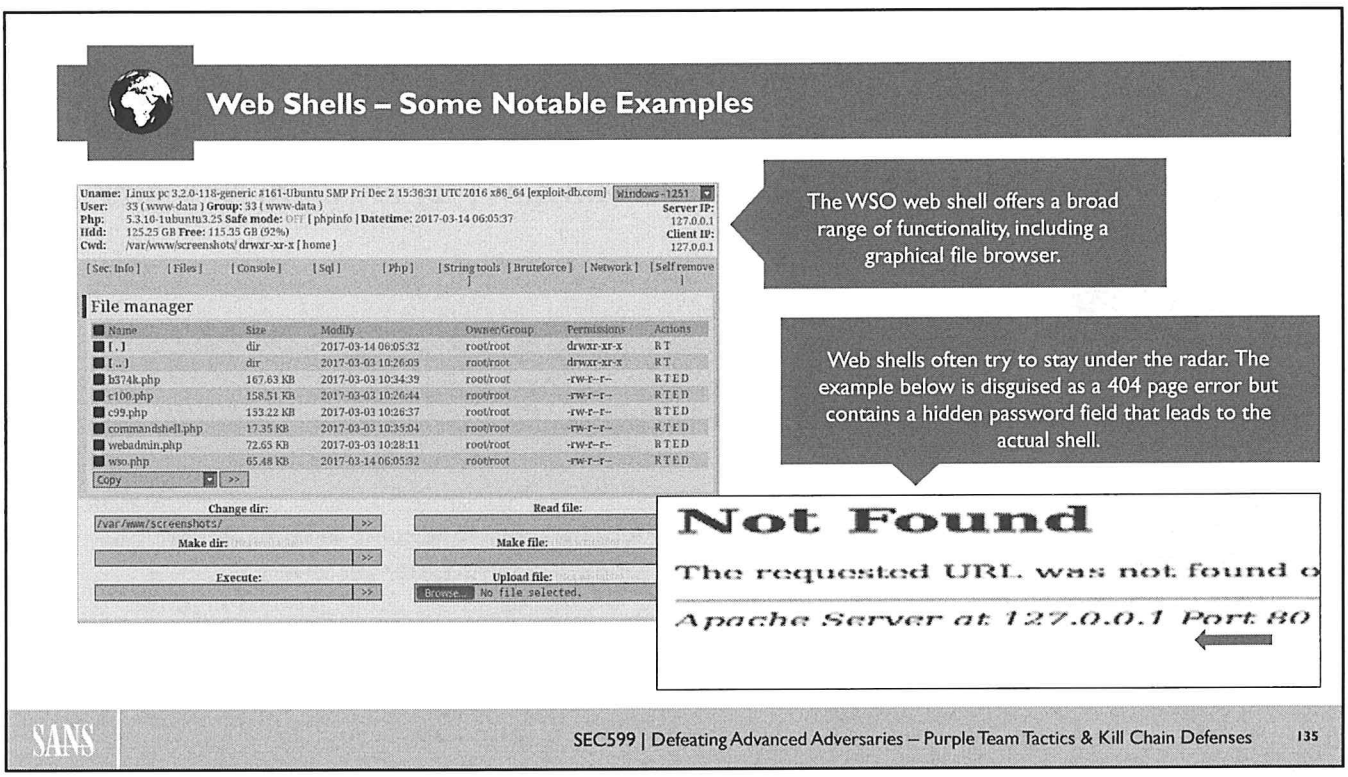
Web Shells – Introduction
A web script allows an adversary to run commands on the targeted web server. It can serve as a gateway into the network. Web shells can provide a simple interface that allows to run single commands or they can consist of an advanced GUI with multiple types of functionality, such as direct file access, database connections, or network reconnaissance to explore the internal network.

For an adversary to be able to abuse a web shell on a web server, the web shell first has to be uploaded. This could either be done through a legitimate upload function provided by the web server or might be possible due to a vulnerability present in the web application or web server software. Once the adversary has been able to upload the web shell, it has to be served back. If the file is accessible, but not interpreted as web script, and thus shown back as simple text, the adversary will not be able to execute commands. The web server has to interpret the web shell's script and serve that back to the adversary.

Some famous examples are web shells used by the Deep Panda threat group, who used them as primary access back into victim organizations in the defense, legal, telecommunication and financial industries. This was an interesting approach as web shells were mostly only seen as a first stage into obtaining a foothold in the target network, after which they would be abandoned as soon as a second stage malware was communicating back to the adversaries. The usage of web shells gave Deep Panda some advantages, such as the absence of C&C beacon traffic, a low detection rate by AV products, and the ease to switch source IP addresses, making it difficult for the defenders to block known C&Cs. More information on Deep Panda web shells can be found here: https://www.crowdstrike.com/blog/mo-shells-mo-problems-deep-panda-web-shells/

An advanced example of a web shell is the China Chopper shell, which supports server payloads for many different kinds of server-side scripting languages and contains functionality to access files, connect to a database, and open a virtual command prompt. An analysis of China Chopper can be found here: https://www.fireeye.com/blog/threat-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html.

© 2019 Erik Van Buggenhout & Stephen Sims



Web Shells – Some Notable Examples

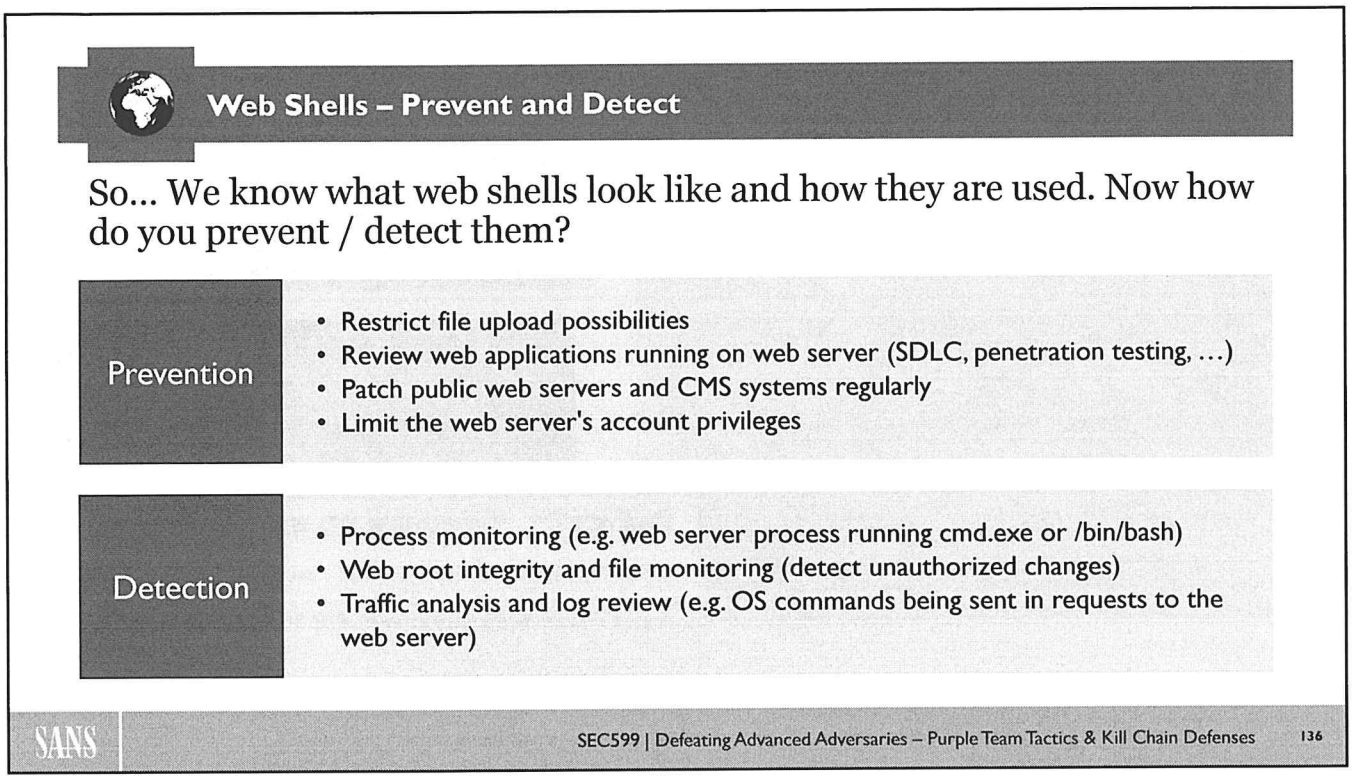
A number of shells offer the creation of a botnet in as little as a click, launching standalone processes that either connect to a command and control server or listen for commands over an insecure TCP connection. Some allow performing port scans to find potentially exploitable services. Others enable fraudsters to schedule denial of service attacks. There are shells dedicated to sending bulk spam emails, testing stolen credentials against popular websites (such as PayPal or Amazon), cracking passwords, and automatically defacing websites. With such a wide array of powerful features, it is unsurprising how popular web shells are with cyber criminals.

A popular and feature-rich web shell is WSO. WSO offers, among others file management, such as browsing directory contents in a GUI, but also both bind shell and back connect options. Selecting one of these options will launch a standalone process that will connect to or listen for a connection from a remote command and control server—an easy method for the creation of a botnet.

Web shells will often try to stay under the radar to avoid detection by server admins or other hackers. A particularly common ploy is that of fake error pages, used by some variants of the C99 web shell. These shells attempt to recreate the default Apache error pages, usually 404 Not Found or 403 Forbidden. When viewed in a web browser, these fake pages can easily be mistaken for legitimate error messages. However, when compared side-by-side, discrepancies can be found by looking for incorrect or omitted version numbers, hostnames, URLs, and HTML titles. These fake error pages also contain hidden password fields, which provide access to the web shell: Some variants simply set the background and border colors to match the page background, while others add JavaScript that reveals the password form when the port number is clicked.

Another notable method for avoiding detection is prefixing the web shell scripts with small excerpts of image file headers—most commonly those from the GIF89a specification. When processed by the PHP interpreter, these bytes are ignored and passed through to the web browser, displaying the text "GIF89a". Automated tools such as the open-source utility file use these magic bytes as a fingerprint to identify the file type, mistaking the malicious PHP script for an image. Reference: https://news.netcraft.com/archives/2017/05/18/web-shells-the-criminals-control-panel.html

© 2019 Erik Van Buggenhout & Stephen Sims



Web Shells – Prevent and Detect

It is easier to prevent web shells from being abused as a persistence mechanism in your server or network than to detect them.

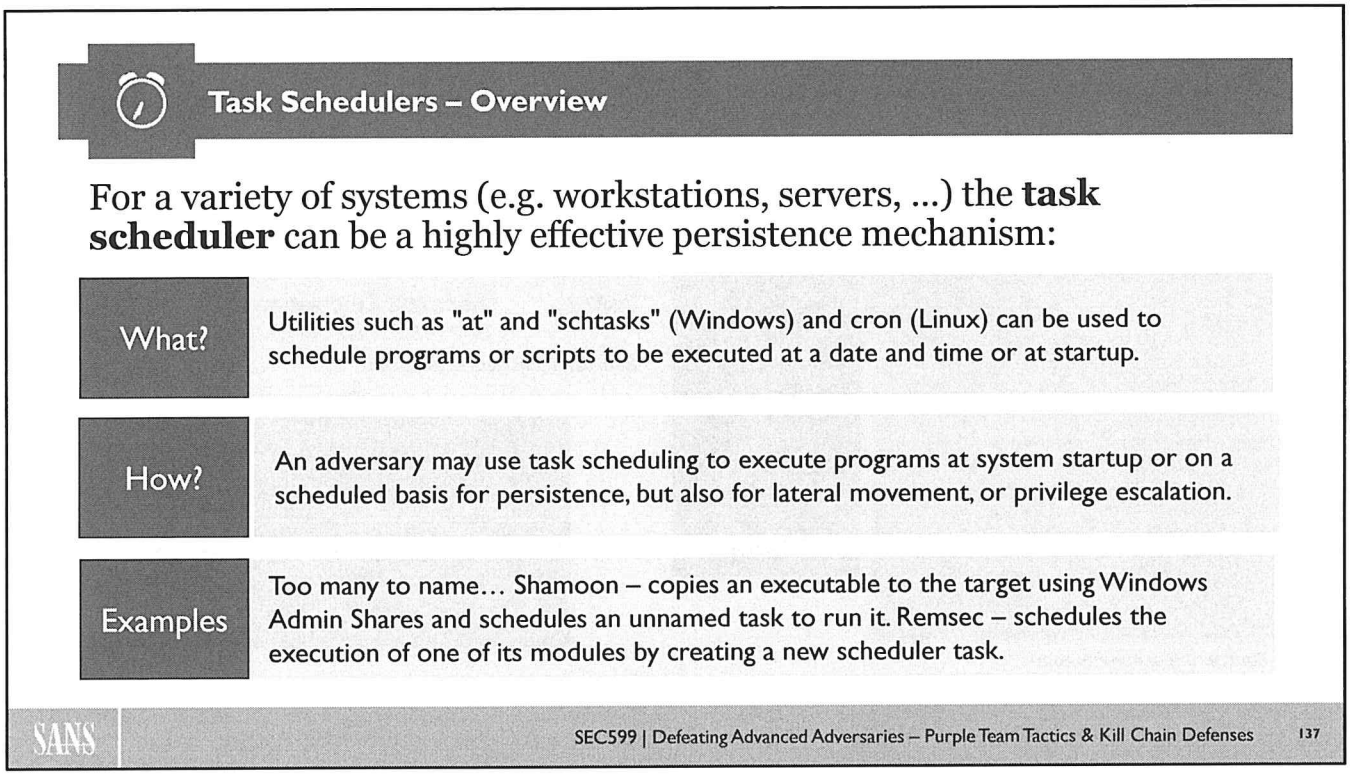
If your web application has a file upload function, it should have some restrictions. First of all, it's possible to limit the types of files that can be uploaded (based file header for example). It could be wise to disallow users to upload web scripts of various types. In case all kinds of files are allowed to be uploaded, it's possible to restrict how users access these files. Avoid having the web server interpret uploaded scripts, for example, by changing the filenames for all uploaded files or by only allowing users to download them.

In case file upload is not allowed, measures should be taken to avoid unforeseen file upload. Make sure adversaries cannot abuse known vulnerabilities that could allow remote code execution or file inclusion by regularly patching your externally reachable web servers.

In case adversaries do succeed in uploading and abusing a web shell, it is still possible to limit the possible damage they can do. Audit account and group permissions of the web server's account and make sure it does not have local root privileges or access to unnecessary files and folders. If the web server is part of an Active Directory, make sure accounts that are used to manage it do not overlap with permissions for the internal network.

In case all prevention measures failed, the web shell might still be detected. However, they can be difficult to detect, since they do not initiate connections. The portion of the shell that resides on the web server might also be small and innocent looking. Process monitoring could be used to detect suspicious actions such as command execution or file access outside the web directory. Often, an adversary will try to go for the /etc/passwd file. File monitoring could be used to detect web shell code being injected into other web application files. In case files are changed that do not match updates to the web content, this might be an indication of unauthorized changes by an adversary. Additionally, log authentication attempts to the server to avoid potential brute forcing and monitor network traffic for suspicious activity to and from the web server and the internal network.

© 2019 Erik Van Buggenhout & Stephen Sims



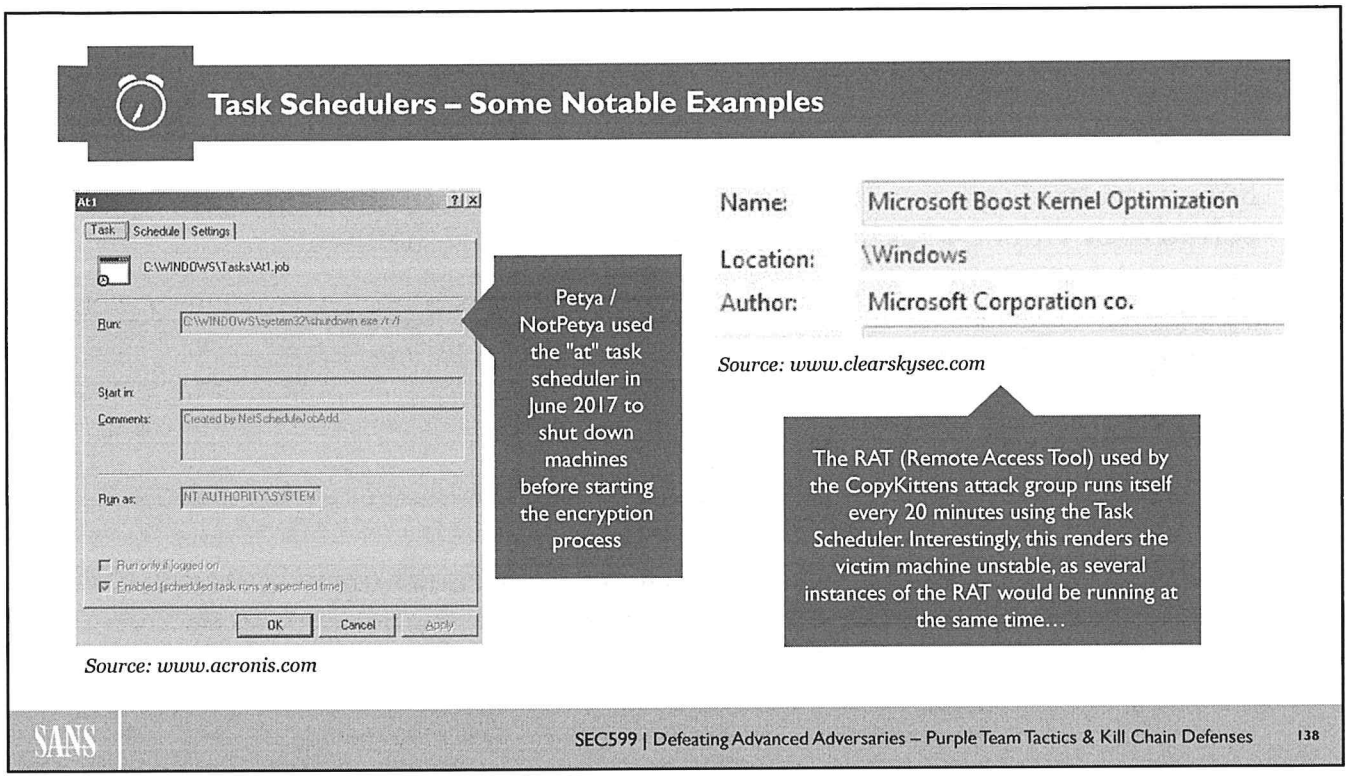
Task Schedulers – Overview
Utilities such as "at" and "schtasks", along with the Windows Task Scheduler, can be used to schedule programs or scripts to be executed at a date and time or at startup. The account used to create the task must be in the Administrators group on the local system. A task can also be scheduled on a remote system, provided the proper authentication is met to use RPC and file and printer sharing is turned on.

An adversary may use task scheduling to execute programs at system startup or on a scheduled basis for persistence, but also for lateral movement, or privilege escalation to SYSTEM, or running a process under the context of a specified account.

A famous case that makes use of scheduled tasks is Shamoon. It copies an executable payload to the target using Windows Admin Shares and schedules an unnamed task to run the malware. Remsec – schedules the execution of one of its modules by creating a new scheduler task. https://www.fireeye.com/blog/threat-research/2016/11/fireeye_respondsto.html

Another well-known example is Remsec, a modular backdoor aimed at espionage, which was used by Strider, a.k.a. ProjectSauron. The execution of one of its modules was scheduled using a new scheduler task. An analysis by Kaspersky can be found here: https://securelist.com/files/2016/07/The-ProjectSauron-APT_Technical_Analysis_KL.pdf.

© 2019 Erik Van Buggenhout & Stephen Sims



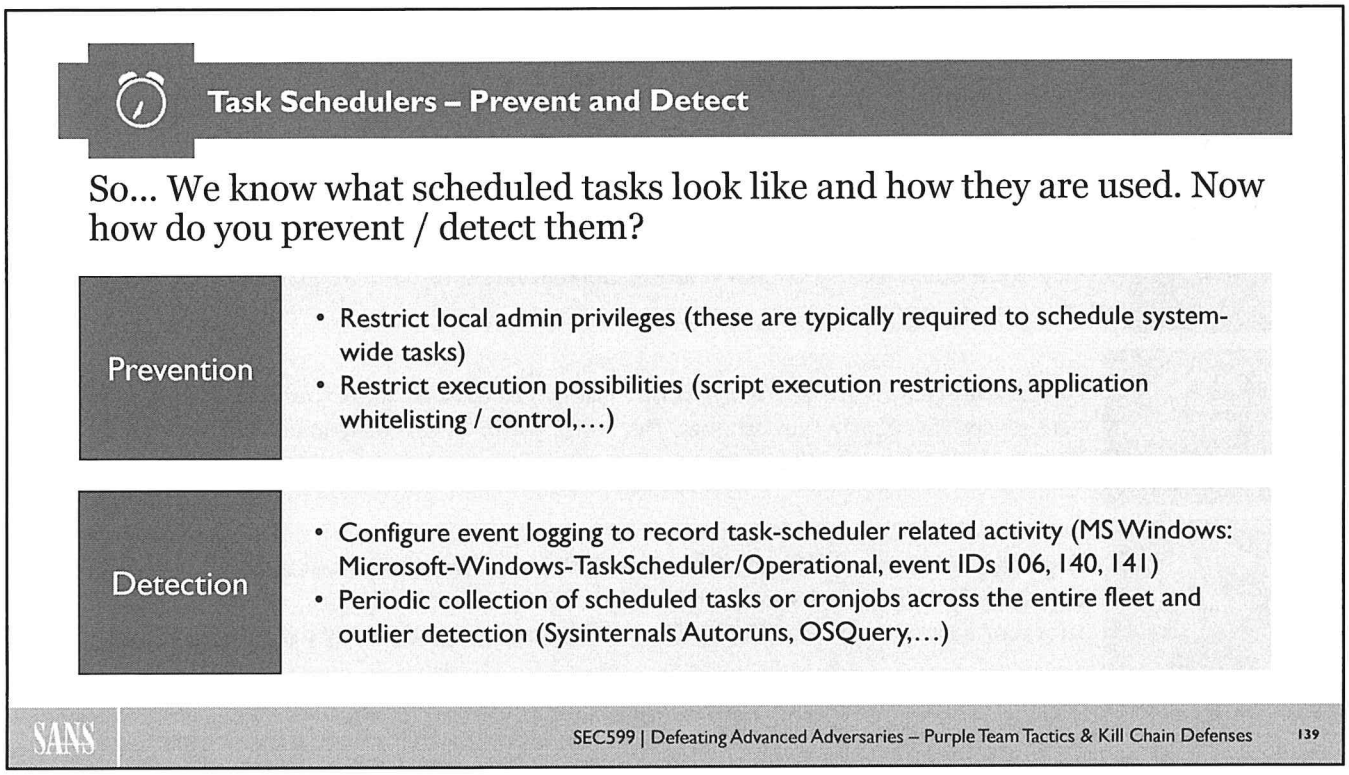
Task Schedulers – Some Notable Examples

So, let's have a look at some notable examples of persistence through task schedulers. Due to the simplicity of this attack, it's one of the most popular persistence strategies. Some examples include:

- The Metasploit Meterpreter has a built-in "automated persistence" method that will rely on a scheduled task being added;
- Petya / NotPetya used the "at" task scheduler in June 2017 to shut down machines before starting the encryption process;
- The RAT (Remote Access Tool) used by the CopyKittens attack group runs itself every 20 minutes using the Task Scheduler. Interestingly, this renders the victim machine unstable, as several instances of the RAT would be running at the same time…

From a forensic perspective, scheduled tasks (both using the Task Scheduler and "at") do leave some artifacts we can use for further analysis and investigation.

© 2019 Erik Van Buggenhout & Stephen Sims



Task Schedulers – Prevent and Detect
So... We know what scheduled tasks look like and how they are used. Now how do you prevent / detect them?

It's a good idea to restrict local admin privileges to ensure only administrative users can create scheduled task on a system. This is typically the case on Windows systems. Furthermore, some of the controls that were previously discuss for "execution prevention" can be useful:

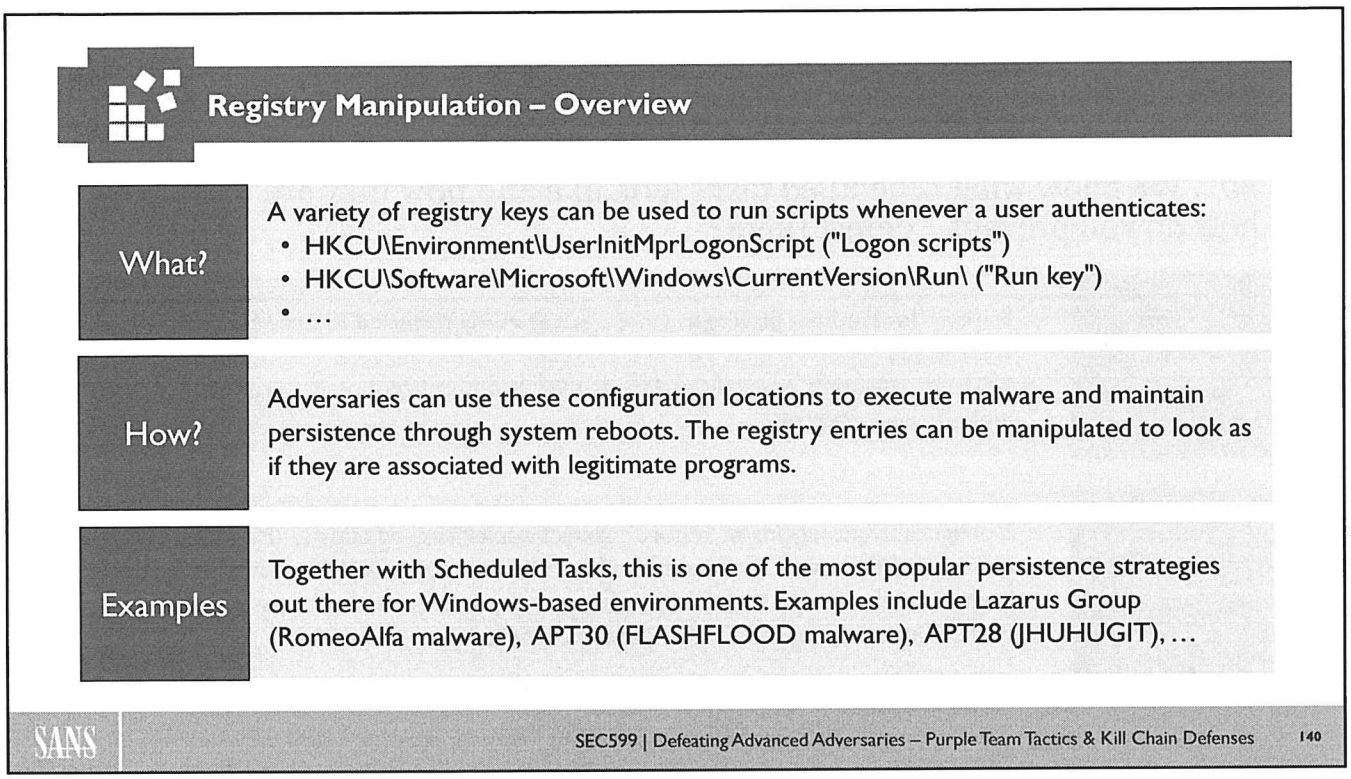
- PowerShell Constrained Language Mode to prevent PowerShell-based execution in Scheduled Tasks;
- Application whitelisting or control to prevent unknown application / executables from running.

In order to detect suspicious tasks that are being scheduled, configure event logging to record task-scheduler related activity. For Windows, you can review "Microsoft-Windows-TaskScheduler/Operational" and look for the following event IDs:

- 106 - Scheduled task registered
- 140 - Scheduled task updated
- 141 - Scheduled task removed

Finally, periodic collection of scheduled tasks or cronjobs across the entire IT environment can be performed using tools such as Sysinternals Autoruns, OSQuery,... The results of these tools can then be used to perform analysis, outlier detection,...

© 2019 Erik Van Buggenhout & Stephen Sims



Registry Manipulation – Overview
Adding an entry to the "Run keys" in the Registry or startup folder will cause the program referenced to be executed when a user logs in. The program will be executed under the context of the user and will have the account's associated permissions level. Some examples include:

- Login scripts under HKCU\Environment\UserInitMprLogonScript
- "Run" registry keys under a variety of possible locations, including HKCU\Software\Microsoft\Windows\CurrentVersion\Run\

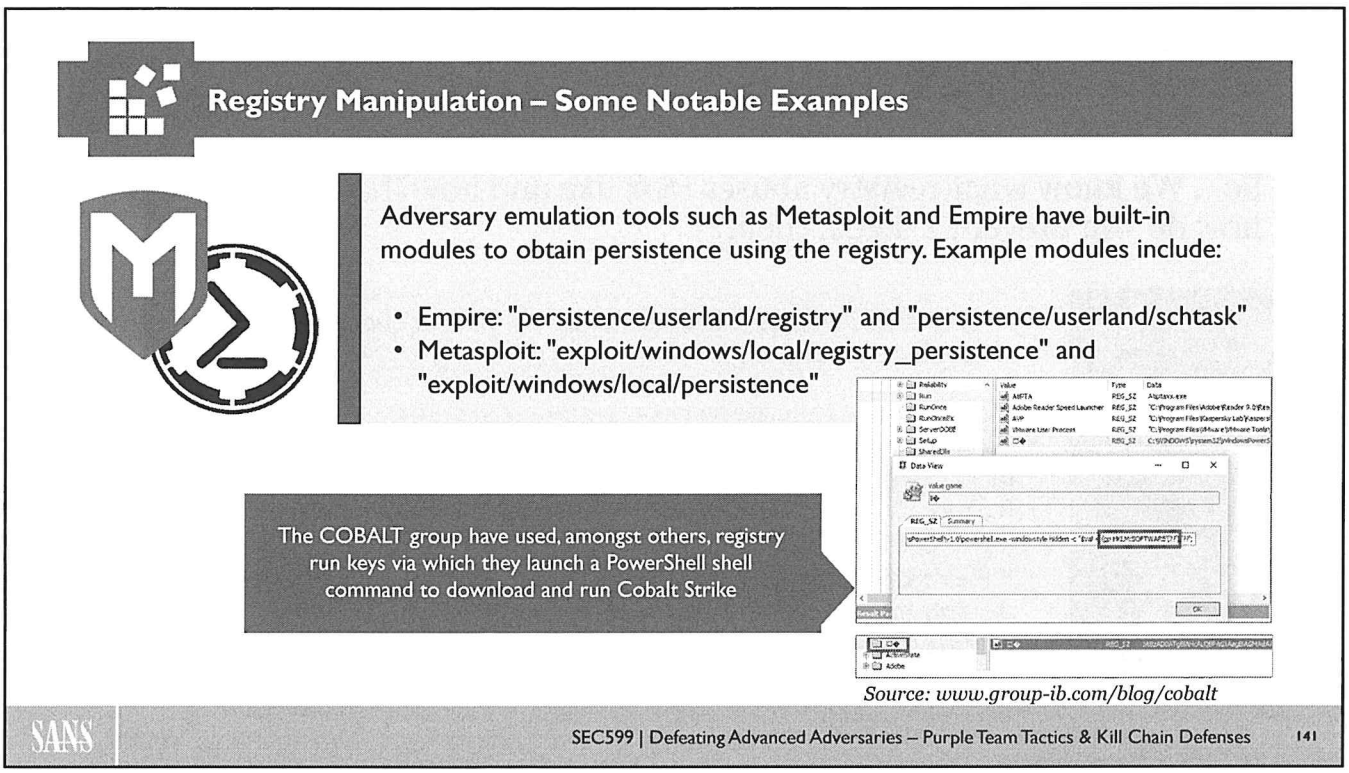
Adversaries can use these configuration locations to execute malware and maintain persistence through system reboots. The Registry entries can be manipulated to look as if they are associated with legitimate programs.

Lazarus Group made use of RomeoAlfa malware, which has persistence by saving itself in the Start menu folder.

FLASHFLOOD – Malware developed by APT30 that can exfiltrate data across air-gaps. It creates an entry in the Run key. The following reference contains a section on the FLASHFLOOD malware: https://www2.fireeye.com/rs/fireye/images/rpt-apt30.pdf.

JHUHUGIT is a piece of malware used by APT28. It registers a Windows shell script under the Registry key HKCU\Environment\UserInitMprLogonScript to establish persistence. A good analysis of the APT28 group, including the use of Logon scripts is located here: https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-part1.pdf.

© 2019 Erik Van Buggenhout & Stephen Sims



Registry Manipulation – Some Notable Examples
There are quite a few samples that have used the registry to host (part of) their persistence mechanism.

Adversary emulation tools such as Metasploit and Empire have a wide variety of different built-in modules to obtain persistence. Several of them leverage the registry. Some examples include:

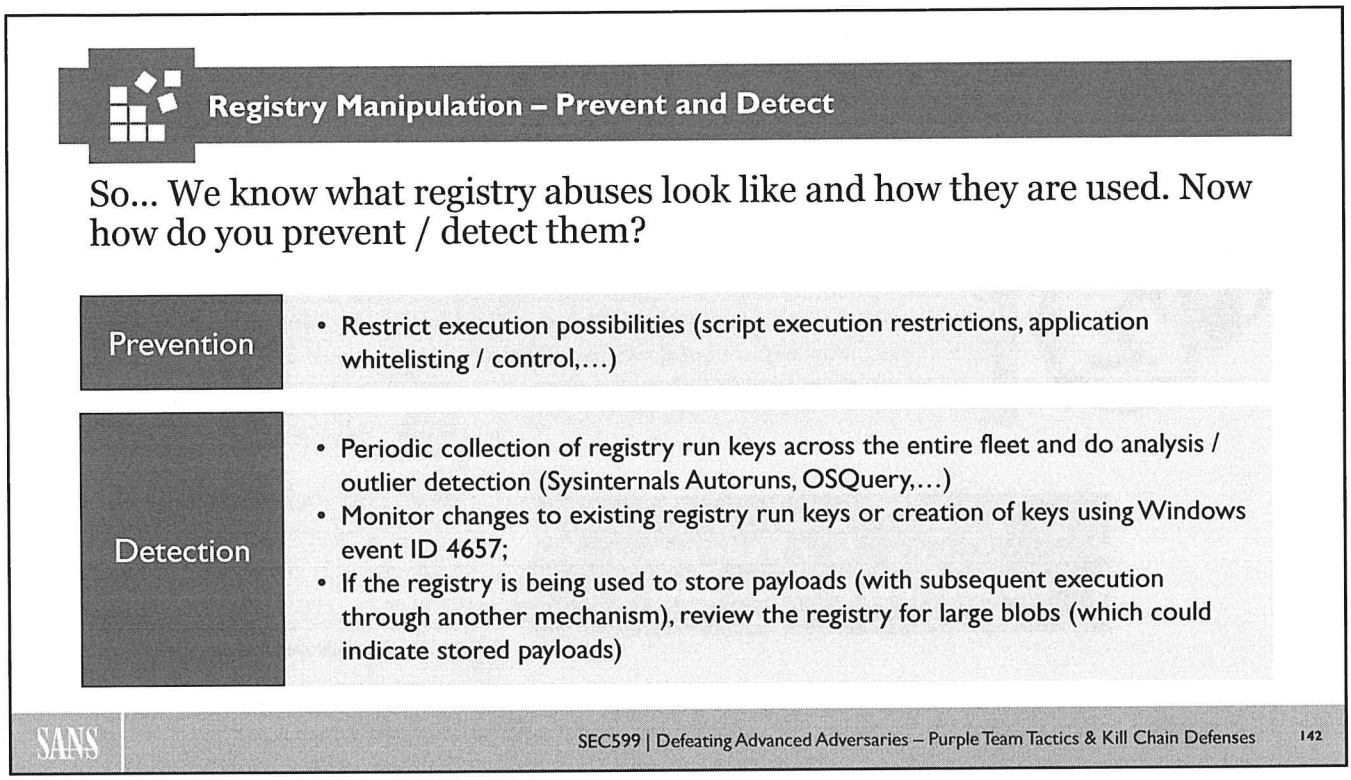
- For Empire, "persistence/userland/registry" and "persistence/userland/schtask". While the "schtask" might surprise you, it's actually an interesting way of storing the actual payload in the registry, while using a Scheduled Task to extract and run the actual payload from the registry.

- For Metasploit, "exploit/windows/local/registry_persistence" and "exploit/windows/local/persistence" offer very similar features as Empire described above.

This very popular persistence mechanism is used by a great variety of groups. An example is the COBALT group, they have been observed using registry run keys to establish persistence. In this specific example, the registry run key executed a PowerShell shell via which they downloaded and ran Cobalt Strike.

References:
http://www.powershellempire.com/?page_id=221
http://www.powershellempire.com/?page_id=223
https://www.rapid7.com/db/modules/exploit/windows/local/registry_persistence

© 2019 Erik Van Buggenhout & Stephen Sims



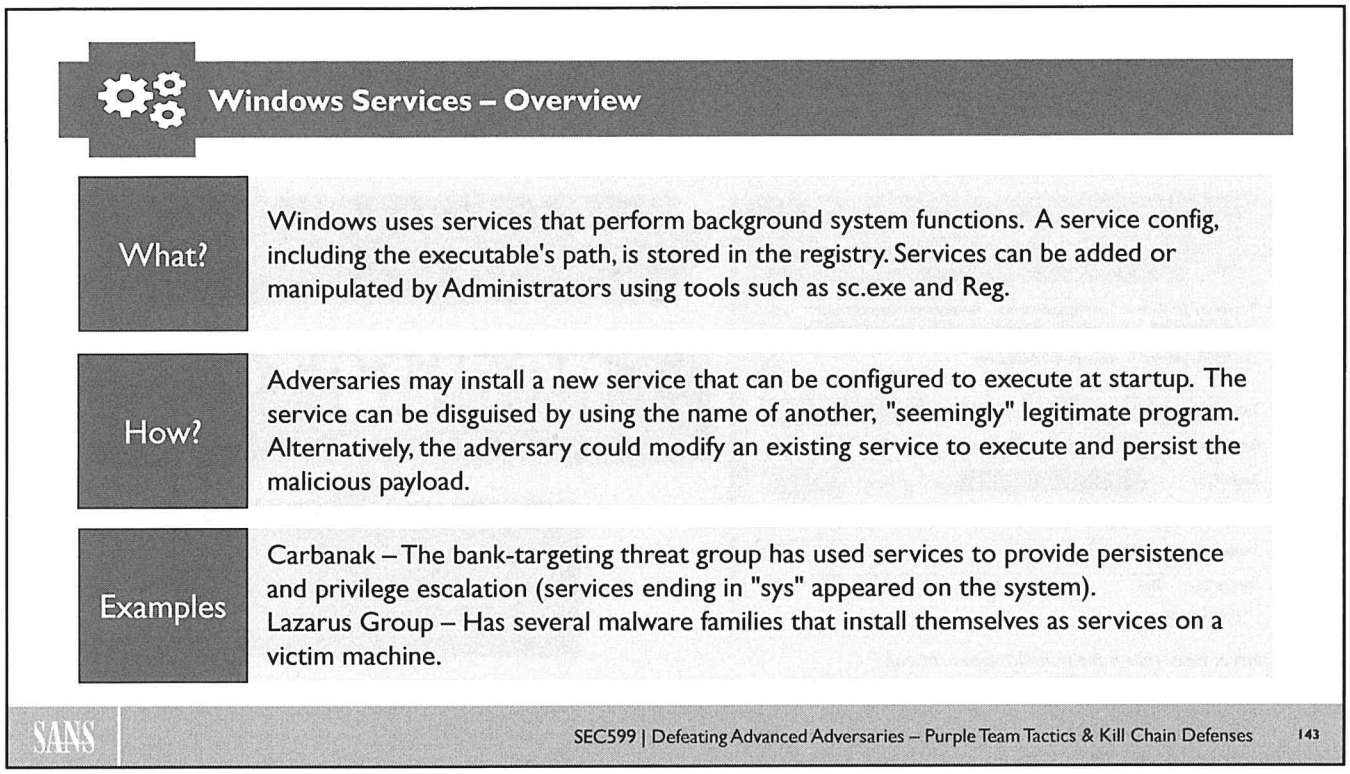
Registry Manipulation – Prevent and Detect
So how can we prevent or detect abuse of the registry as part of persistence mechanisms?

- From a preventive point of view, there's not much that can be done, as normal, standard, users are allowed to create registry run keys. When additional executables are being referenced, however, there is an opportunity to prevent launching of these executables by, again, using script execution restrictions or application whitelisting / control;

- In order to detect registry abuse, there are a few options available:

  - Defenders can periodically collect registry run keys across the entire fleet and perform analysis / outlier detection (Sysinternals Autoruns, OSQuery,...).
  - Central collection and monitoring of Windows events related to changes to registry keys using Windows event ID 4657;
  - If the registry is being used to store payloads (with subsequent execution through another mechanism), the registry could be periodically reviewed for large blobs (which could indicate stored payloads). This is, however, prone to false positives and will require manual analysis.

© 2019 Erik Van Buggenhout & Stephen Sims



Windows Services – Overview
When booting, Windows can start programs or applications called services that perform background system functions. A service's configuration information, including the file path to the service's executable, is stored in the Windows Registry. Service configurations can be modified using utilities such as sc.exe and Reg.

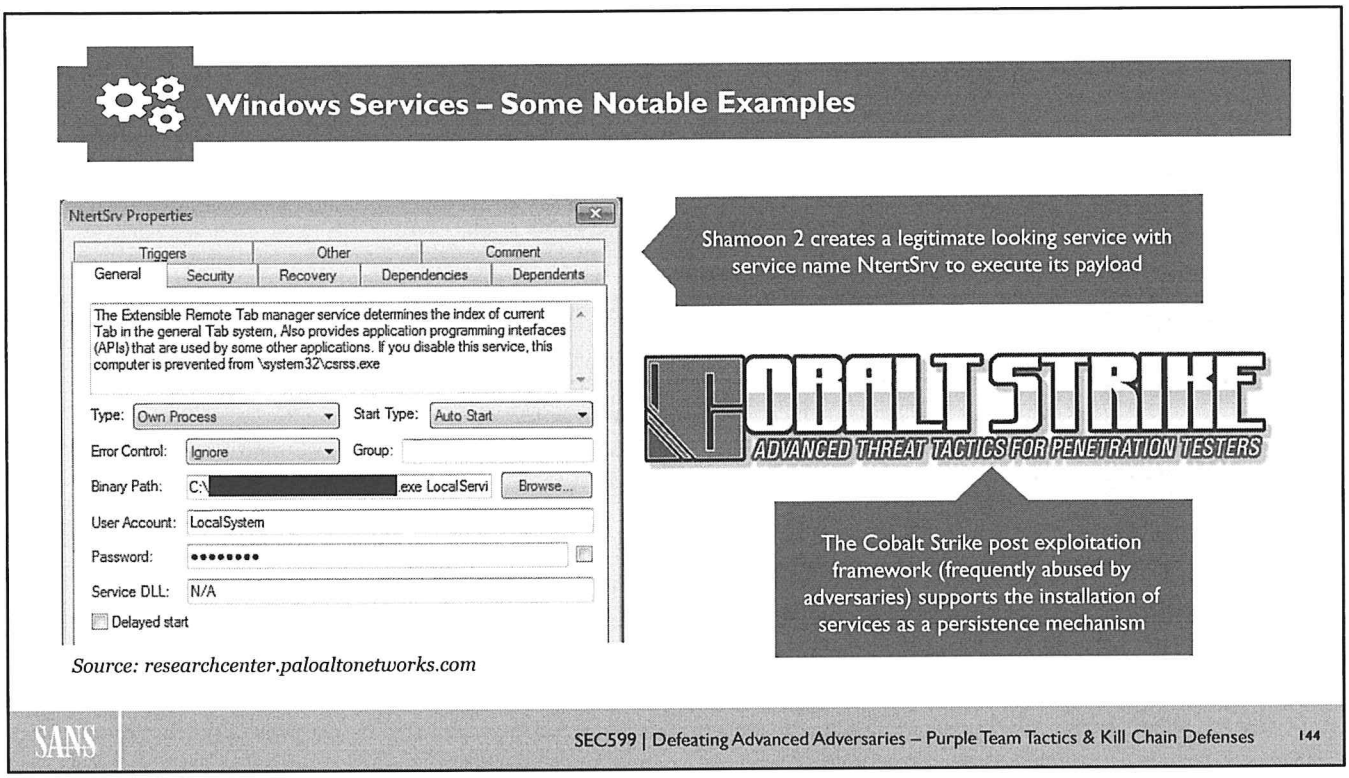
Adversaries may install a new service that can be configured to execute at startup by using utilities to interact with services or by directly modifying the Registry. The service can be disguised by using the name of another, legitimate program. Services may be created with administrator privileges but are executed under SYSTEM privileges, so an adversary can use a service to escalate privileges from administrator to SYSTEM. Services can also be executed directly.

Instead of creating a new service, an adversary can also modify an existing service to execute and persist the malicious payload. The usage of existing services is a type of masquerading that may make detection analysis more challenging. Modifying existing services could interrupt their functionality or enable services that are disabled or otherwise not commonly used.

Carbanak is a threat group that mainly targets banks. Their malware makes use of services to provide persistence and privilege escalation. An analysis on the Carbanak APT can be found here: https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/bb625963(v=msdn.10).

Lazarus Group is a threat group that was responsible for the attack against Sony Pictures Entertainment. Several of their malware families install themselves as new services on victim machines. A very detailed report on Operation Blockbuster, carried out by the Lazarus Group, can be found here: https://www.operationblockbuster.com/wp-content/uploads/2016/02/Operation-Blockbuster-Report.pdf

© 2019 Erik Van Buggenhout & Stephen Sims



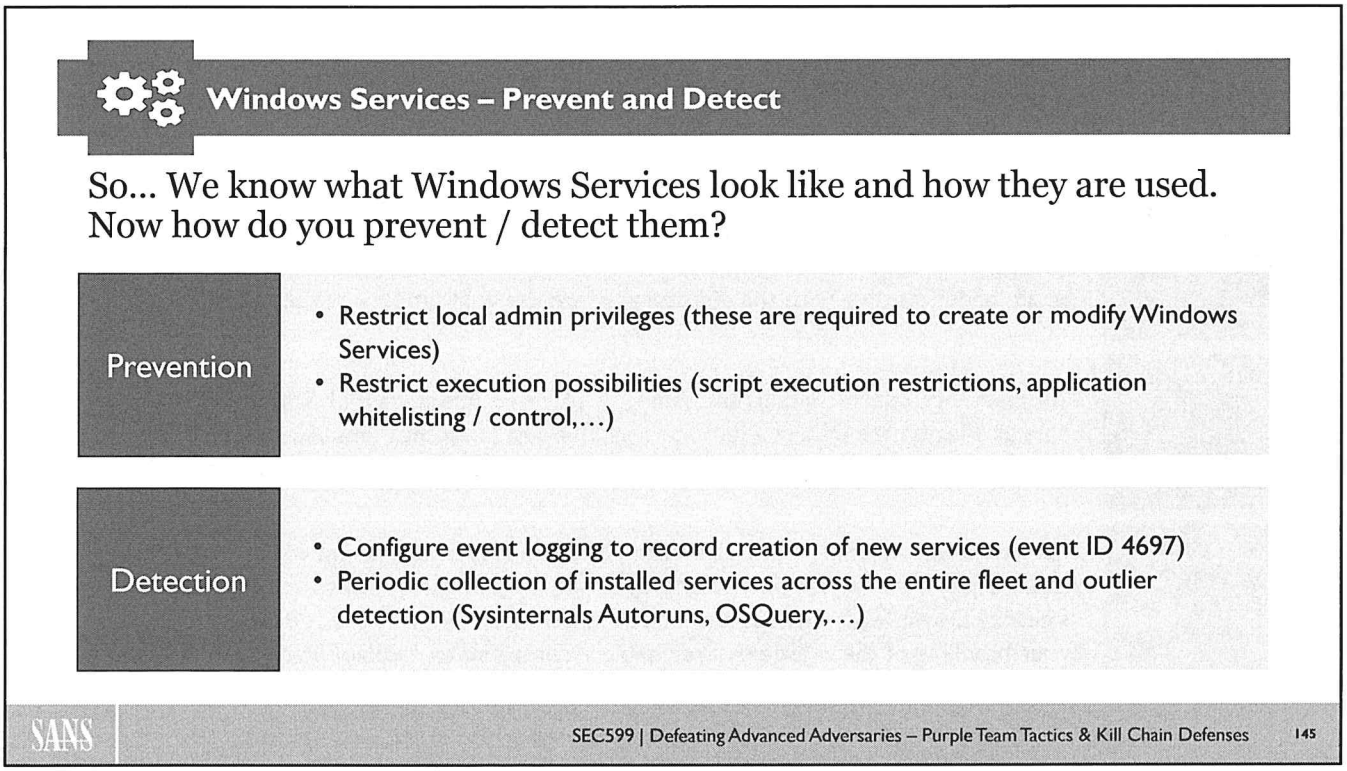
Windows Services – Some Notable Examples

Creating Windows Services is a highly effective persistence mechanism, and as such, frequently used by a multitude of adversaries.

One such example is observed in the Shamoon 2 attack on an organization in Saudi Arabia. In this attack, the payload is dropped in the System32 folder on the host after which a service is created to ensure persistence. A legitimate looking name "NtertSrv" is given to the service to blend in with the other services. An analysis on Shamoon 2 can be found here: https://unit42.paloaltonetworks.com/unit42-shamoon-2-return-disttrack-wiper/

The Cobalt Strike post-exploitation framework has, among others, the ability to create a new service on the system in order to ensure persistence. The configuration of these services, including the file path pointing to the executable launched by the service, is stored in the Windows Registry.

© 2019 Erik Van Buggenhout & Stephen Sims



Windows Services – Prevent and Detect
So how can we prevent or detect abuse of the Windows Services as part of persistence mechanisms?

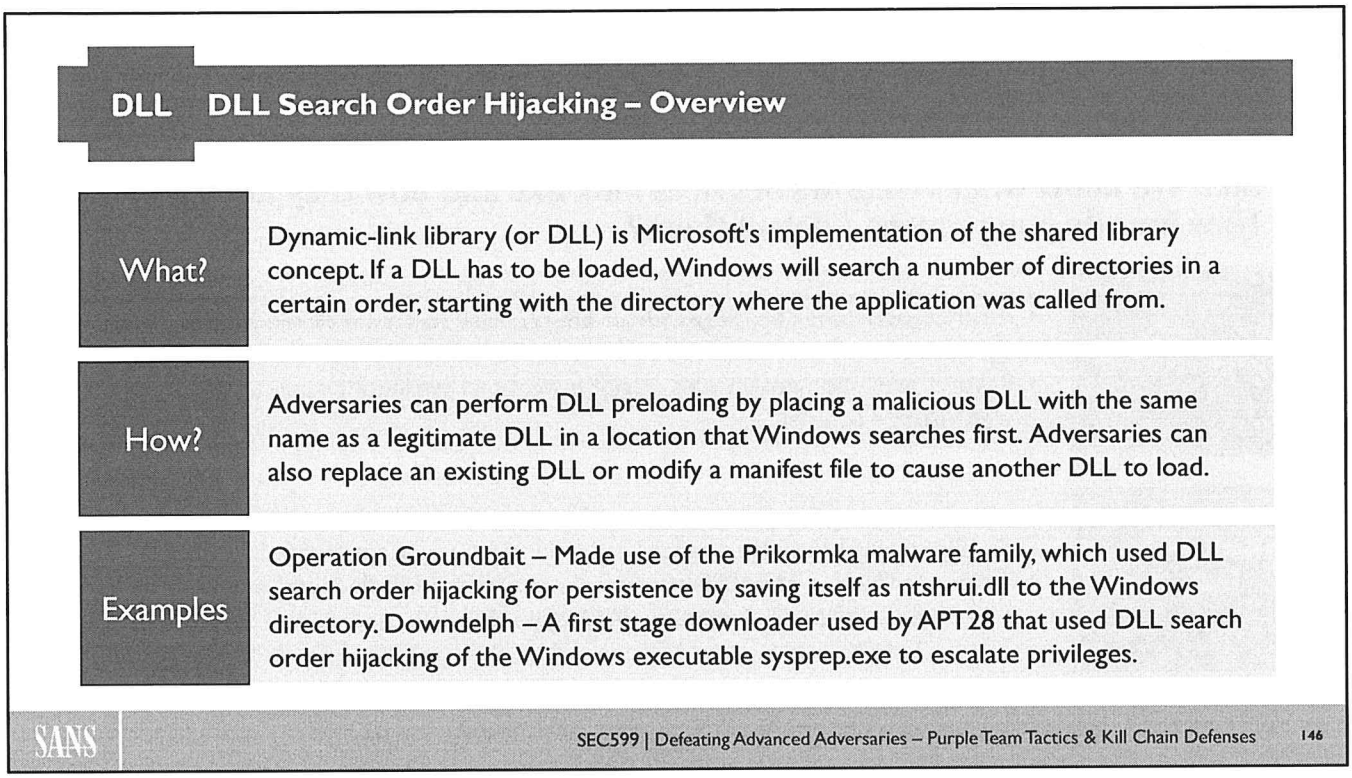
From a prevention perspective, there are two specific items that will prevent the attacker from installing a Windows service:

- Restriction of local administrator privileges and remediation of privilege escalation vulnerabilities (in order to create a Windows service, local admin privileges are required)
- Prevent the usage of system utilities or potential malicious software via which an adversary could install a Windows service. This can be achieved by, for example, script execution restrictions, application whitelisting, etc.

In order to detect abuse of Windows Services, there are some options available:

- Configure the Windows Event log service to record the creation of new services. When triggered, this event will get the Event ID 4697.
- Periodically review the Windows Services installed across the entire fleet. Outlier detection or frequency analysis can help identifying abnormal or newly installed services. The information on installed services in the entire fleet can be obtained by using tools such as Sysinternals, Autoruns, OSQuery, etc.

© 2019 Erik Van Buggenhout & Stephen Sims

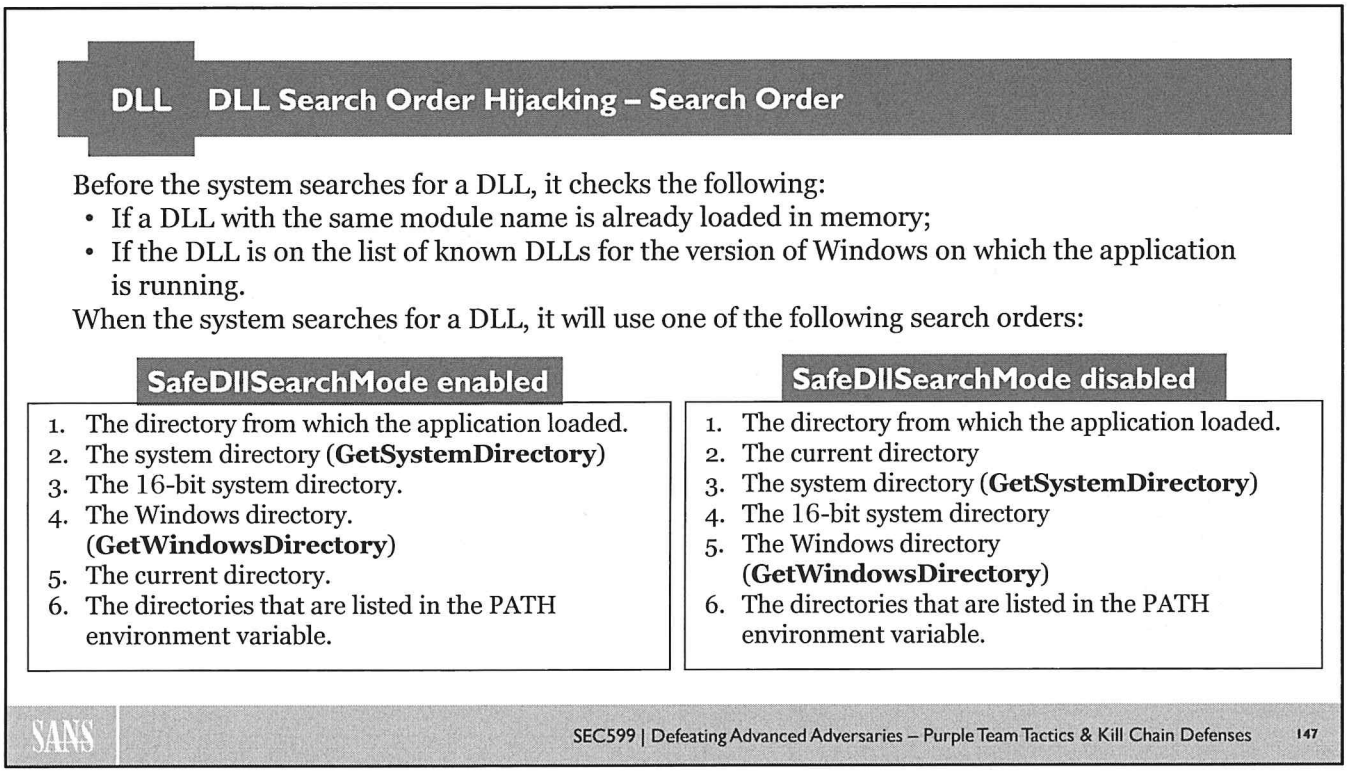


DLL Search Order Hijacking – Overview
Dynamic-link library (or DLL) is Microsoft's implementation of the shared library concept. Windows will follow a specific search order when a DLL has to be loaded. Adversaries can perform DLL preloading, by placing a malicious DLL with the same name as a legitimate DLL in a location that Windows searches first. That way, when Windows encounters the malicious DLL, it will be loaded instead of the legitimate one. Adversaries can also replace an existing DLL or modify a manifest file to cause another DLL to load. The malicious DLL can also be configured to load the legitimate DLLs they are meant to replace, which means the application keeps functioning as it normally would.

Operation Groundbait was mostly observed in Ukraine and used for surveillance. It made use of a malware family called Prikormka (which roughly translates to groundbait from Russian), which employed DLL search order hijacking for persistence by saving itself as ntshrui.dll to the Windows directory so it will load before the legitimate ntshrui.dll saved in the System32 subdirectory. Info on Operation Groundbait can be found here: https://www.welivesecurity.com/wp-content/uploads/2016/05/Operation-Groundbait.pdf

Another example is Downdelph, a first stage downloader that was (although rarely) used by APT28. It also made use of DLL search order hijacking, but targeted the Windows executable sysprep.exe, with the goal of escalating privileges. An analysis can be found here: https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-part3.pdf

© 2019 Erik Van Buggenhout & Stephen Sims



DLL Search Order Hijacking – Search Order
Before the system searches for a DLL, it checks the following:

- If a DLL with the same module name is already loaded in memory, the system uses the loaded DLL, no matter which directory it is in. The system does not search for the DLL.
- If the DLL is on the list of known DLLs for the version of Windows on which the application is running, the system uses its copy of the known DLL (and the known DLL's dependent DLLs, if any). The system does not search for the DLL. For a list of known DLLs on the current system, see the following registry key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs.

The standard DLL search order used by the system depends on whether safe DLL search mode is enabled or disabled. Safe DLL search mode places the user's current directory later in the search order.

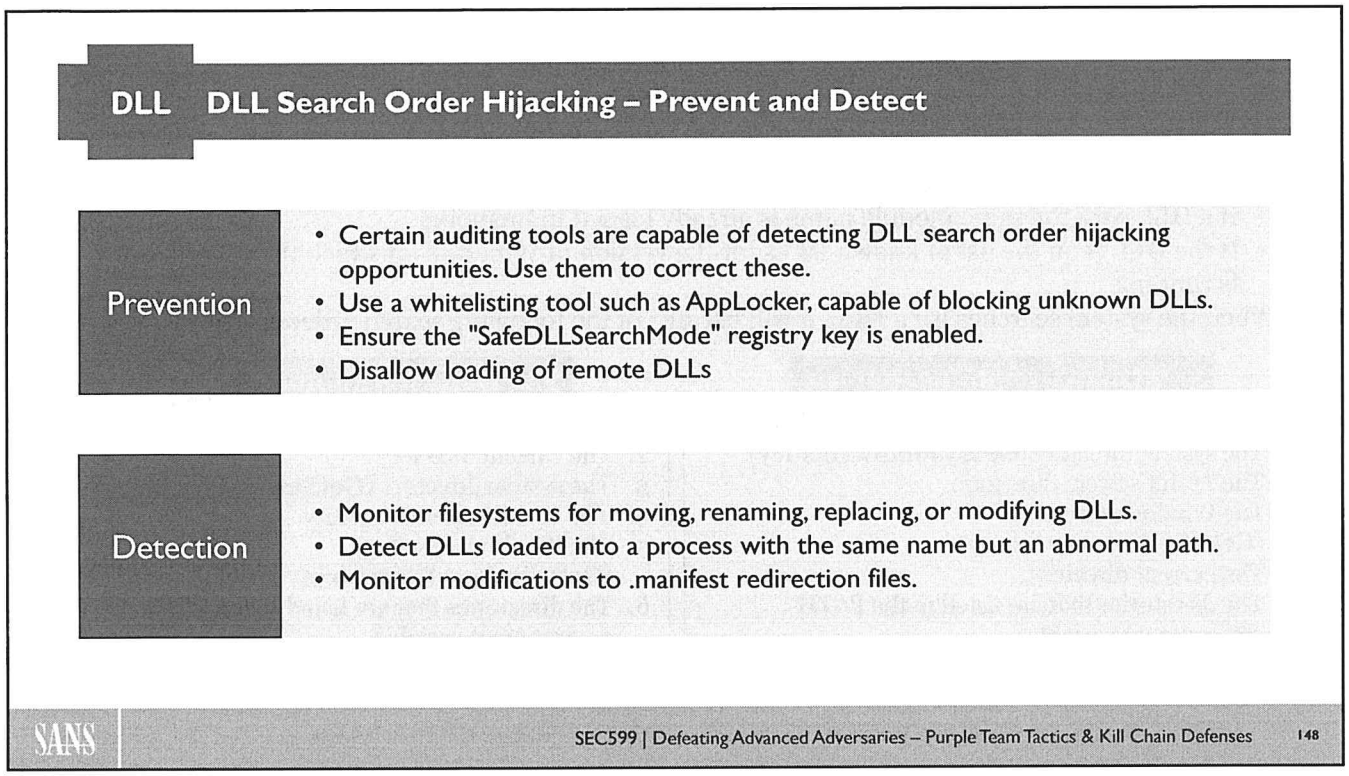
If **SafeDllSearchMode** is enabled, the search order is as follows:

- The directory from which the application loaded.
- The system directory. Use the **GetSystemDirectory** function to get the path of this directory.
- The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
- The Windows directory. Use the **GetWindowsDirectory** function to get the path of this directory.
- The current directory.
- The directories listed in the PATH environment variable. Note that this does not include the per-application path specified by the **App Paths** registry key. The **App Paths** key is not used when computing the DLL search path.

If **SafeDllSearchMode** is disabled, the search order is as follows:

- The directory from which the application loaded.
- The current directory.
- The system directory. Use the **GetSystemDirectory** function to get the path of this directory.
- The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
- The Windows directory. Use **the GetWindowsDirectory** function to get the path of this directory.
- The directories listed in the PATH environment variable. Note that this does not include the per-application path specified by the **App Paths** registry key. The **App Paths** key is not used when computing the DLL search path.

© 2019 Erik Van Buggenhout & Stephen Sims



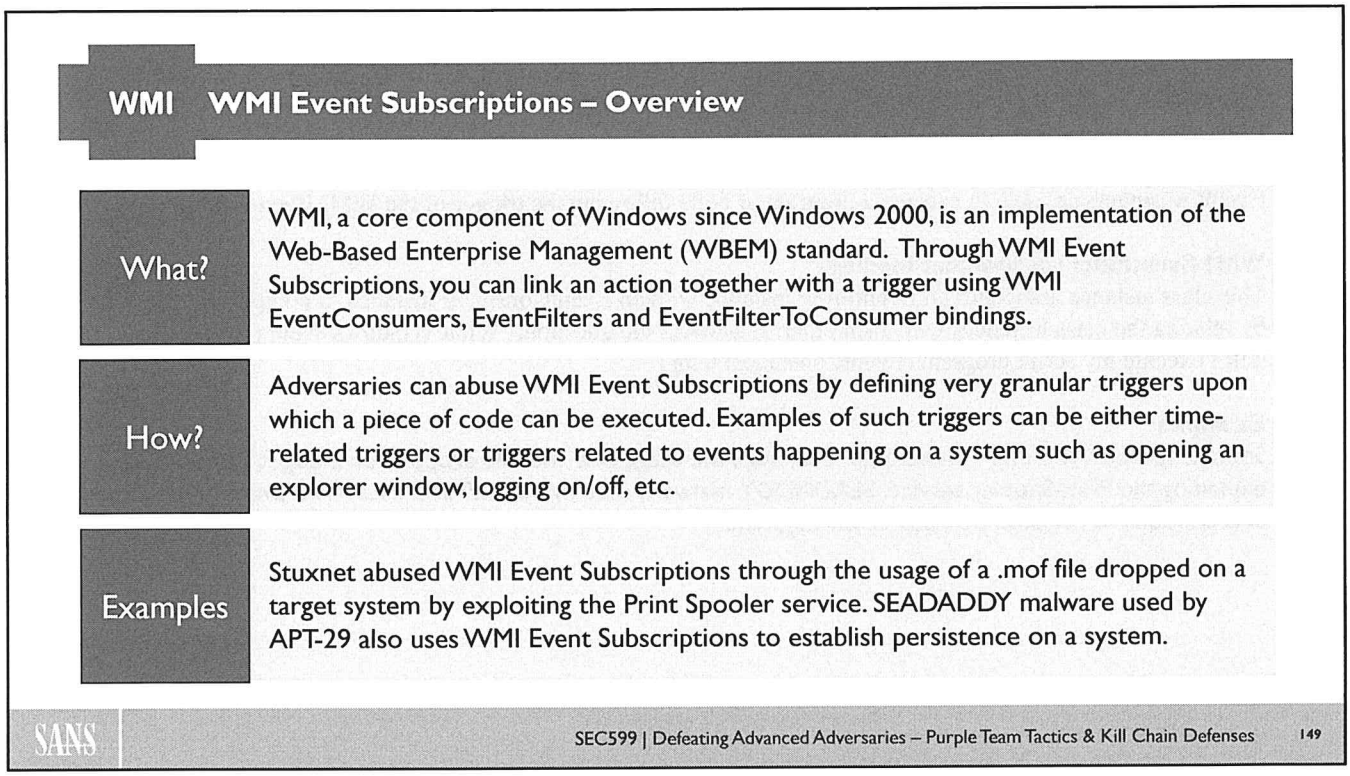
DLL Search Order Hijacking – Prevent and Detect
In order to prevent and detect DLL search order hijacking attacks, there are a few controls we can consider:

- Certain auditing tools are capable of detecting DLL search order hijacking opportunities. You can use these tools to identify potential DLL search order issues and correct them. An example tool is Powerup (https://github.com/PowerShellMafia/PowerSploit/tree/master/Privesc).
- Additionally, a whitelisting tool such as AppLocker, capable of blocking unknown DLLs, can be used as well.
- Finally, in order to reduce the opportunity of DLL search order hijacking vulnerabilities, ensure the "SafeDLLSearchMode" registry key is enabled.
- Disallow loading of remote DLLs, which could take place if an application sets its current directory to a folder on a share.

In order to detect DLL search order hijacking attacks, the following controls can be considered:

- Monitor filesystems for moving, renaming, replacing, or modifying DLLs. Changes in the set of DLLs that are loaded by a process (compared with past behavior) that do not correlate with known software, patches, etc., are suspicious.
- Monitor DLLs loaded into a process and detect DLLs that have the same filename but abnormal paths.
- Modifications to or creation of .manifest and .local redirection files that do not correlate with software updates are suspicious.

© 2019 Erik Van Buggenhout & Stephen Sims



WMI Event Subscriptions – Overview

WMI, which stands for Windows Management Instrumentation, is an implementation of the Web-Based Enterprise Management standard, which is a set of system management technologies that have been developed to unify and facilitate the management of hosts in an enterprise environment. WMI uses the Common Information Model (CIM), which is an industry standard to represent systems, applications, networks, devices, and other managed components. WMI has been a core component of Windows since Windows 2000.

As the name Windows Management Instrumentation implies, this is a set of tools that allows you to manage devices and applications in a Windows environment. This includes remotely changing system settings, properties, and permissions.

WMI provides a uniform access mechanism to a huge collection of Windows management data and methods, and it is very easy to access. WMI offers access to this information via script, C++ programming interfaces, .NET classes (system.management), a command-line tool (WMIC) and others. Remote WMI connections are made through DCOM. An alternative for DCOM could be to use Windows Remote Management (WinRM), which obtains remote WMI management data using the WS-Management SOAP-based protocol.

A subscription is the term used for WMI persistence, and it consists of the following three items:

- An Event Consumer: An action to perform upon triggering an event of interest.
- An Event Filter: The event of interest.
- A Filter to Consumer Binding: The registration mechanism that binds a filter to a consumer

The possibilities with WMI subscriptions are endless, as an EventFilter can trigger on about everything resulting in something else being executed by the EventConsumer.

WMI Event Filter

A WMI filter consists of a query that is checked against the WMI data on the target machine; the answer is always true or false. This is, for example, used a lot in group policies (e.g. apply GPO to workstations and not servers). WMI Filter is a mandatory class entry creation process to activate event consumer class instances. Event filters are triggers or autostart methods to execute event consumer entries.

© 2019 Erik Van Buggenhout & Stephen Sims

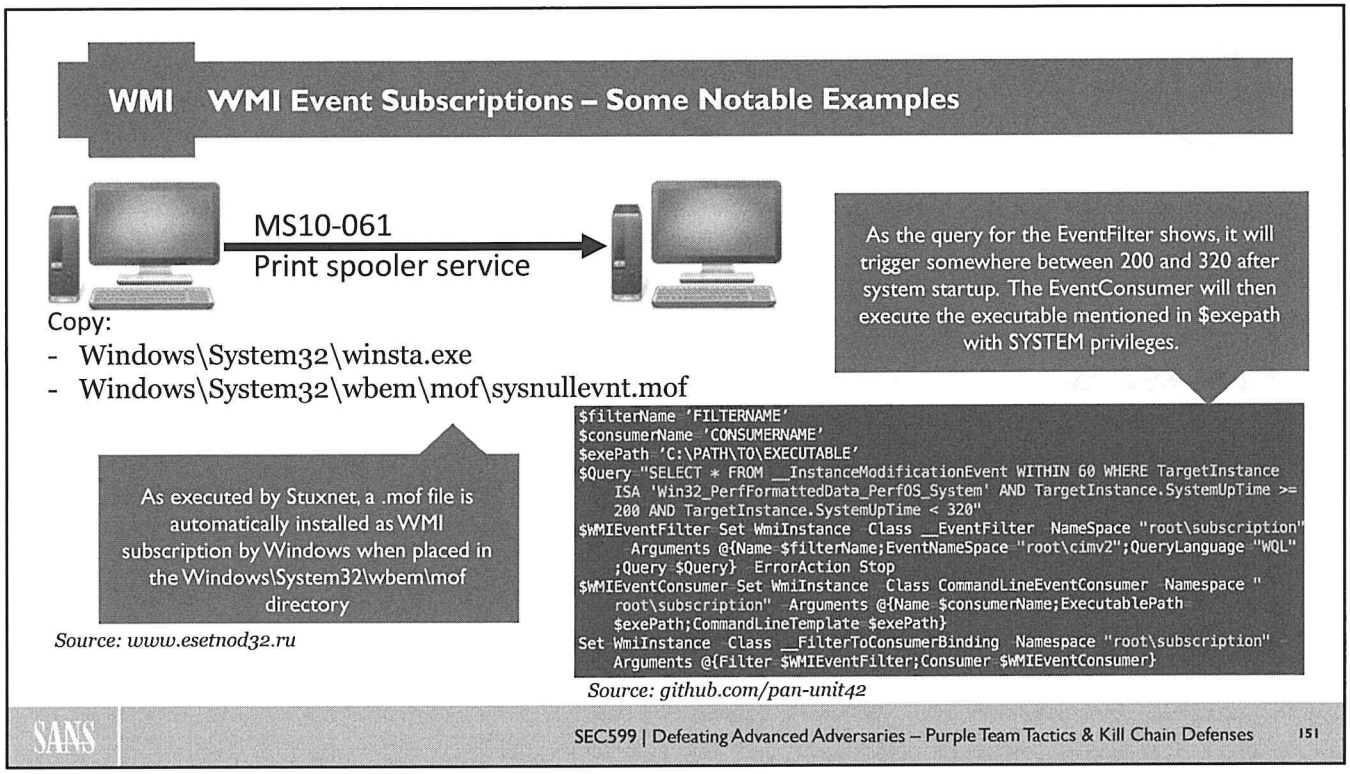
WMI Event Consumers
A WMI consumer is a management application or script that interacts with the WMI infrastructure. Such an application can query data, enumerate data, run provider methods, or subscribe to events by calling either the COM API for WMI or the Scripting API for WMI.

EventConsumers are able to execute a program or code following the trigger of the WMI EventFilter.

WMI EventFilterToConsumer binding
This class instance associates an EventFilter instance with an EventConsumer instance. It completes the cycle by relating the class instances with each other. It answers the question, "What Windows event (EventFilter) will I execute my script program (EventConsumer) with?"

Examples
Stuxnet abused WMI Event Subscriptions through the usage of a .mof file dropped on a target system by exploiting the Print Spooler service. SEADADDY malware used by APT-29 also uses WMI Event Subscriptions to establish persistence on a system.

© 2019 Erik Van Buggenhout & Stephen Sims



WMI Event Subscriptions – Some Notable Examples
The Stuxnet malware contained 4 zero-day exploits, among one of which was one for the Print Spooler Service.

Hosts with file and printer sharing turned on were vulnerable to this attack. The vulnerability allowed a remote user using a guest account to write in the %SYSTEM% directory of the target host. The attack here was performed in two stages:

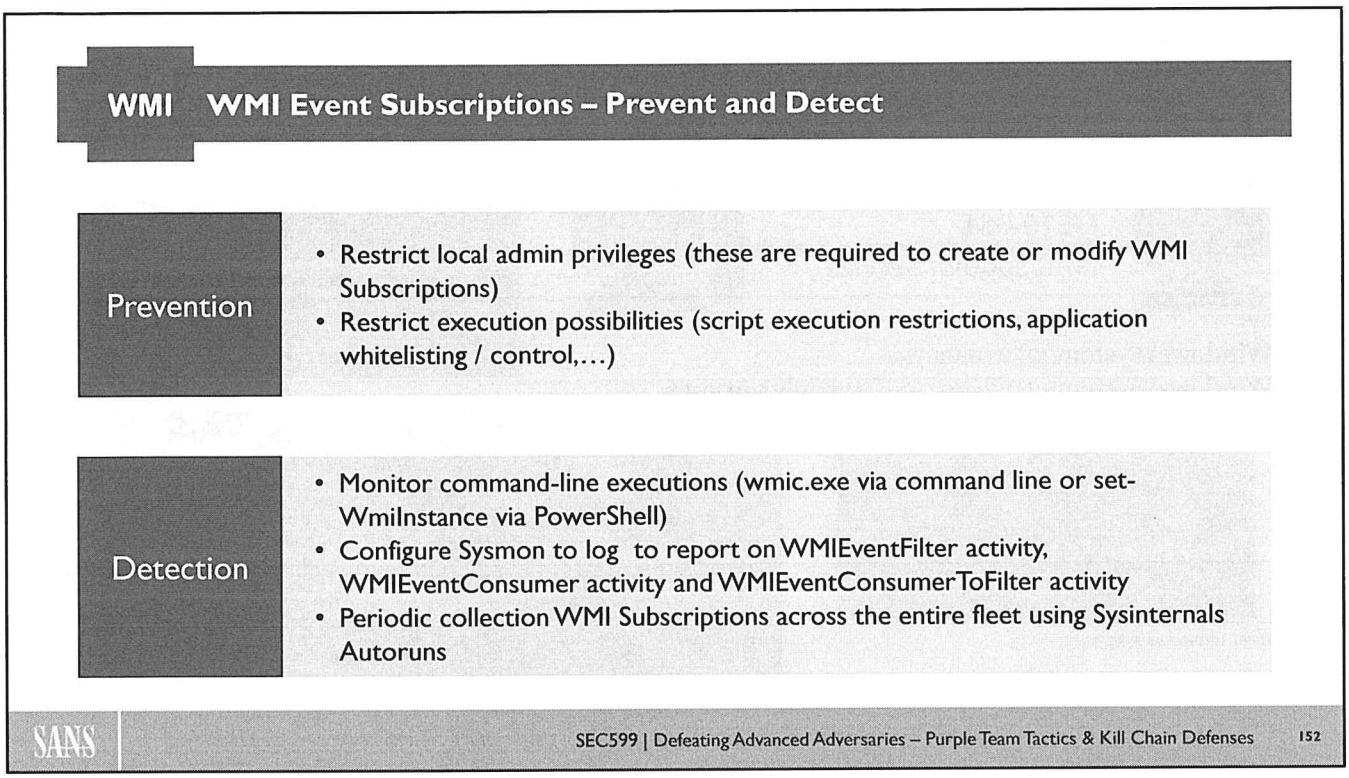
- First the malware copies the dropper and an additional file into Windows\System32\winsta.exe and Windows\System32\wbem\mof\sysnullevnt.mof (exploiting the MS10-061 vulnerability)
- In the second stage, the dropper is executed.

The interesting part here is the .mof file, a .mof file contains all the information to comprise a WMI subscription. When a .mof file is dropped in that specific \wbem\mof directory, it is automatically read and installed by the Windows operating system. In certain specific circumstances, the EventFilter would trigger and execute the winsta.exe executable, which results in the infection of the system.

The SEADADDY malware uses WMI as a persistence mechanism. The SEADADDY malware is known to be used by APT-29. It's a WMI backdoor that operates as follows:

- The EventFilter, defined in $Query is set to trigger somewhere between 200 and 320 seconds after system startup.
- The EventConsumer will, when the EventFilter triggers, execute the executable mentioned in $exepath.

© 2019 Erik Van Buggenhout & Stephen Sims



WMI Event Subscriptions – Prevent and Detect
So how can we prevent or detect abuse of the WMI Event Subscriptions as part of persistence mechanisms?

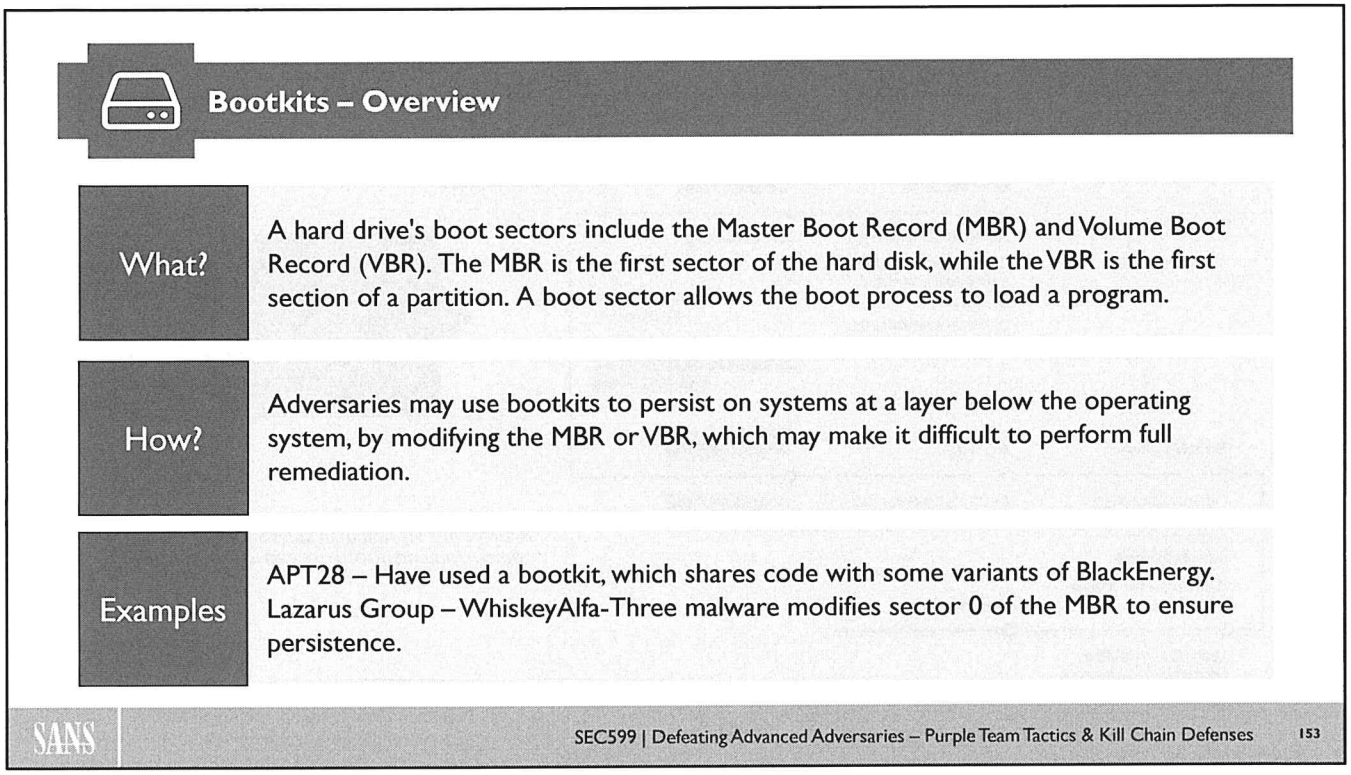
From a prevention perspective, there are two specific items that will prevent the attacker from installing a Windows service:

- Restriction of local administrator privileges and remediate privilege escalation vulnerabilities (in order to create a WMI Event Subscription, local admin privileges are required).
- Prevent the usage of system utilities or potential malicious software via which an adversary could configure an WMI Event Subscription. This can be achieved by, for example, script execution restrictions, application whitelisting, etc.

In order to detect WMI Event Subscription creation, the following controls can be considered:

- Configure Windows to log command-line execution in the Windows Event log, "A new process has been created".
- Enable logging for WMI activity by executing "wevtutil.exe sl Microsoft-Windows-WMI-Activity/Trace /e:true" (this might flood you with events as WMI is used by lots of legitimate applications).
- Create a WMI Event Subscription that triggers on the creation of a new WMI Event Subscription.
- Configure Sysmon to log Event ID 19, 20, 21 to report on WMIEventFilter activity, WMIEventConsumer activity and WMIEventConsumerToFilter activity.
- Use Sysinternals Autoruns to collect all WMI subscriptions across the entire fleet.

© 2019 Erik Van Buggenhout & Stephen Sims



Bootkits – Overview

A boot sector is a region of a hard drive that contains machine code to be loaded into random-access memory (RAM) by a computer's built-in firmware. The purpose of a boot sector is to allow the boot process of a computer to load a program (usually, but not necessarily, an operating system) stored on the same storage device. A hard drive's boot sectors include the Master Boot Record (MBR) and/or Volume Boot Record (VBR):

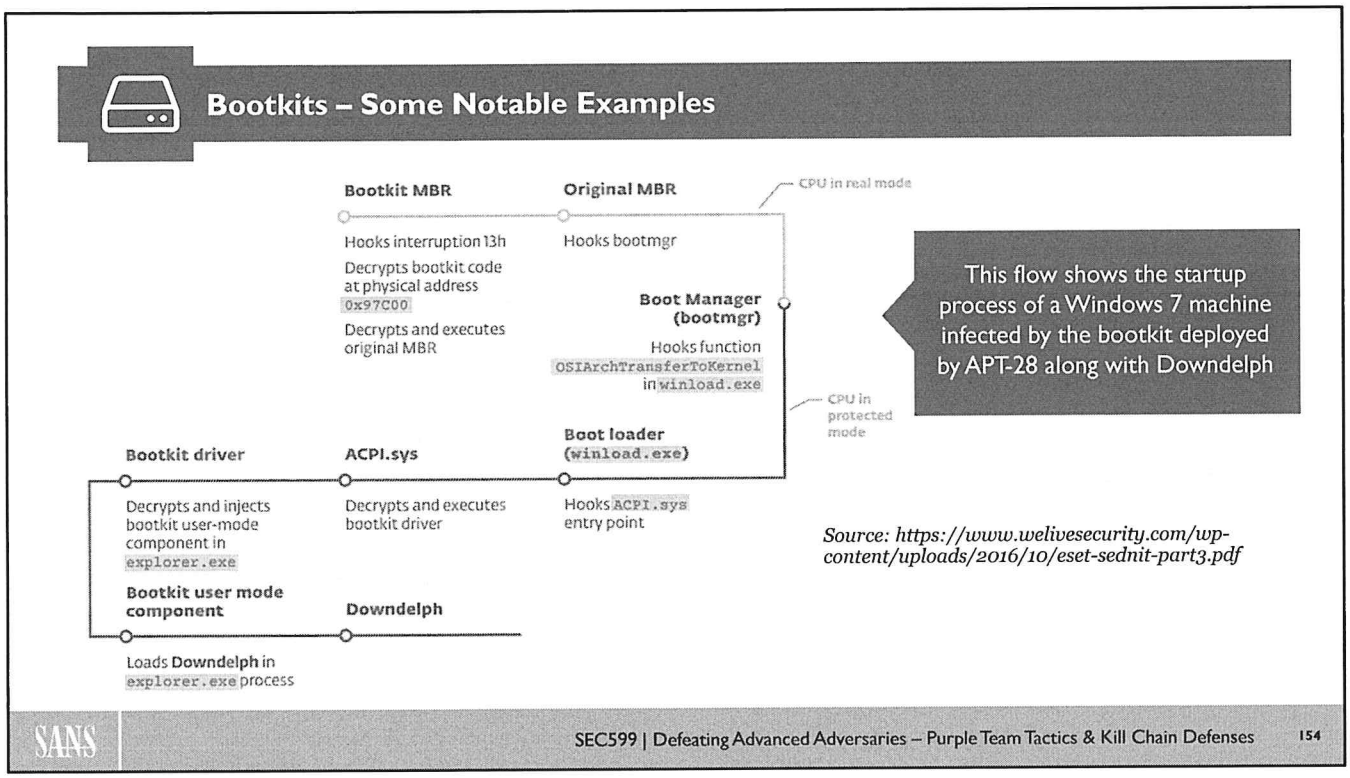
- A Master Boot Record (MBR) is the first sector of a data storage device that has been partitioned. The MBR sector may contain code to locate the active partition and invoke its Volume Boot Record.
- A Volume Boot Record (VBR) is the first sector of a data storage device that has not been partitioned or the first sector of an individual partition on a data storage device that has been partitioned. It may contain code to load an operating system (or other stand-alone program) installed on that device or within that partition.

Adversaries may use bootkits to persist on systems at a layer below the operating system, by modifying the MBR or VBR, which may make it difficult to perform full remediation. The MBR is the section of disk that is first loaded after completing hardware initialization by the BIOS. An adversary who has raw access to the boot drive may overwrite this area, diverting execution during startup from the normal boot loader to adversary code. The MBR passes control of the boot process to the VBR. Similar to the case of MBR, an adversary who has raw access to the boot drive may overwrite the VBR to divert execution during startup to adversary code.

Once again, APT28 comes into play. They have used a bootkit that shares some of its code with variant of the Black Energy malware. More information here: https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-part3.pdf

The Lazarus Group is no stranger to us, either. Their WhiskeyAlfa-Three malware modifies sector 0 of the MBR to ensure persistence of their malicious programs. Detailed info on this (and other) malware can be found here: https://operationblockbuster.com/wp-content/uploads/2016/02/Operation-Blockbuster-Destructive-Malware-Report.pdf

© 2019 Erik Van Buggenhout & Stephen Sims

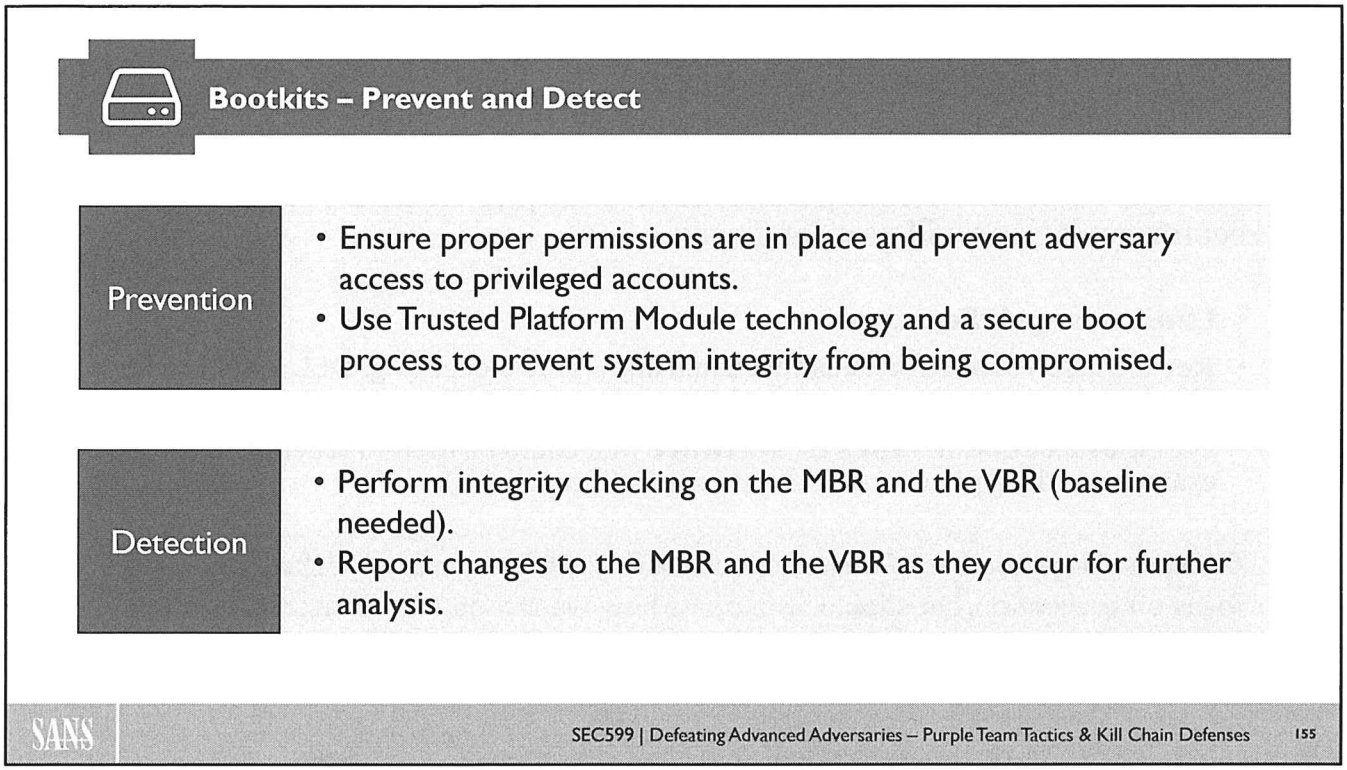


Bootkits – Some Notable Examples

APT28 has deployed a bootkit along with Downdelph to ensure its persistence on the victim. The bootkit shares code with some variants of BlackEnergy. This bootkit employed by APT-28 overwrites the MBR with a custom version, maintaining an encrypted copy of the original MBR code in the second sector. Full details on the operation of this bootkit can be found here:

https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-part3.pdf

© 2019 Erik Van Buggenhout & Stephen Sims

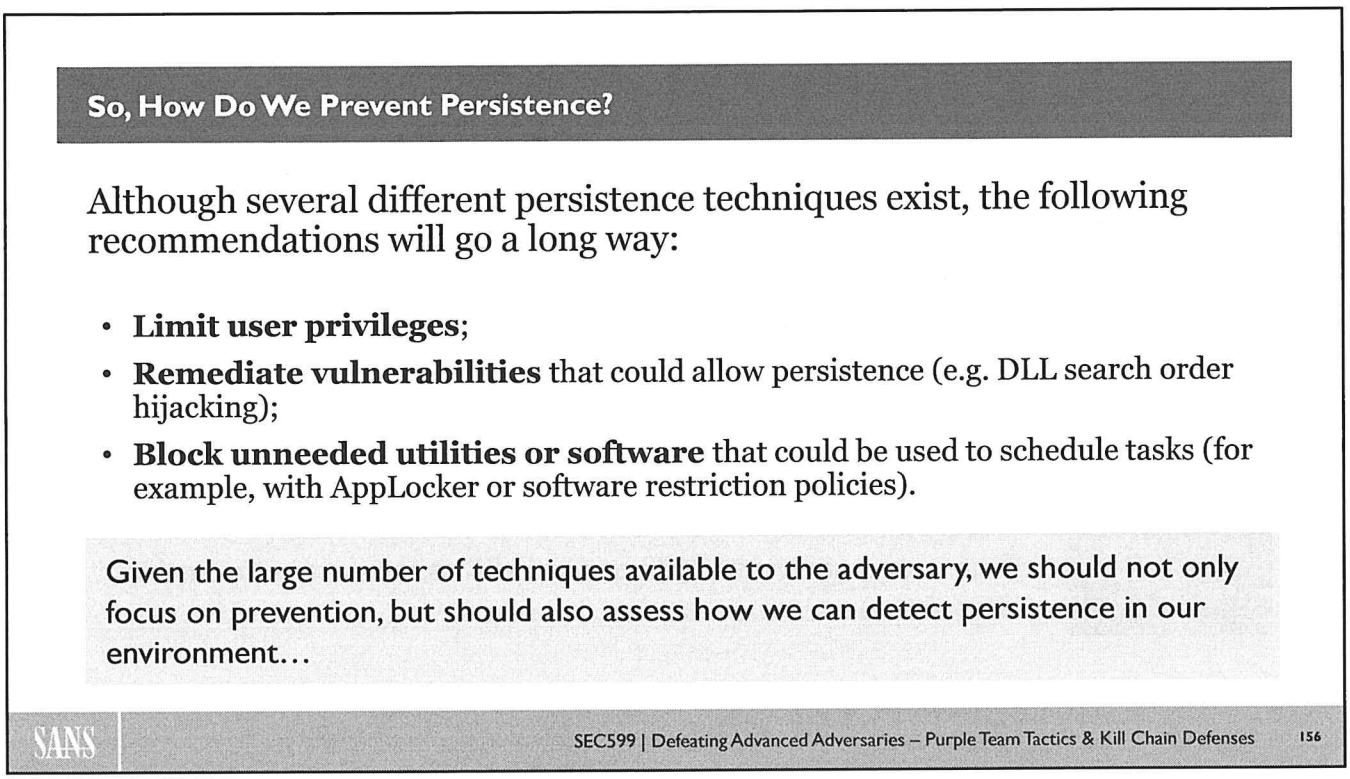


Bootkits – Prevent and Detect
Ensure proper permissions are in place to help prevent adversary access to privileged accounts necessary to perform modifications to the boot sectors. Use Trusted Platform Module (TPM) technology and a secure or trusted boot process to prevent system integrity from being compromised.

A TPM is a microcontroller that can be used to store platform measurements that help ensure that the platform remains trustworthy. The primary purpose of a TPM is to ensure the integrity of the platform. In this context, "integrity" means "behave as intended", and a "platform" is generically any computer platform—not limited to PCs or a particular operating system: Start the power-on boot process from a trusted condition and extend this trust until the operating system has fully booted and applications are running. A good summary on TPM is available here: https://www.trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf

Perform integrity checking on the MBR and the VBR. To do this, take snapshots of the MBR and the VBR and compare against known good samples (i.e., a baseline). In case changes to the MBR or VBR have taken place, further analysis should be performed to determine malicious activity.

© 2019 Erik Van Buggenhout & Stephen Sims

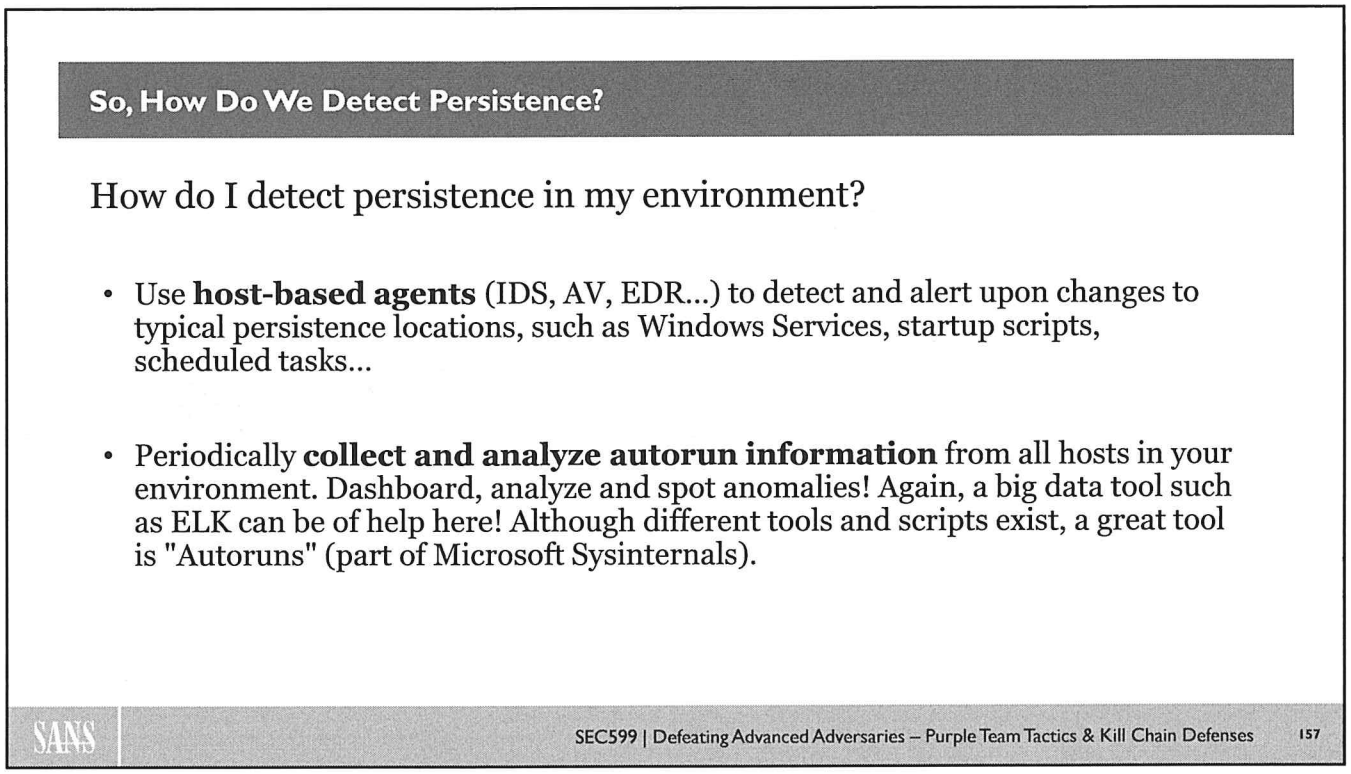


So, How Do We Prevent Persistence?
Several different persistence techniques exist, some of which require a specific approach, which we have highlighted in the techniques above. The following recommendations will, however, go a long way to prevent persistence from succeeding in your environment:

- Limit user privileges to prevent the installation of bootkits or Windows Services. We will discuss this in more depth during the Privilege Escalation and Active Directory sections of this course.
- Remediate vulnerabilities that could allow persistence (e.g. DLL search order hijacking).
- Block unneeded utilities or software that could be used to schedule tasks (for example, with AppLocker or software restriction policies).

As a number of different persistence opportunities exist and it will be difficult to deny an adversary to persistence once he / she actually obtains access to an environment, we strongly advise taking efforts to detect persistence as well!

© 2019 Erik Van Buggenhout & Stephen Sims



So, How Do We Detect Persistence?
Although adversaries are continuously changing and adapting the way they penetrate our networks, they only have a number of options available for persistence. Two main strategies exist to detect persistence strategies in your environment:

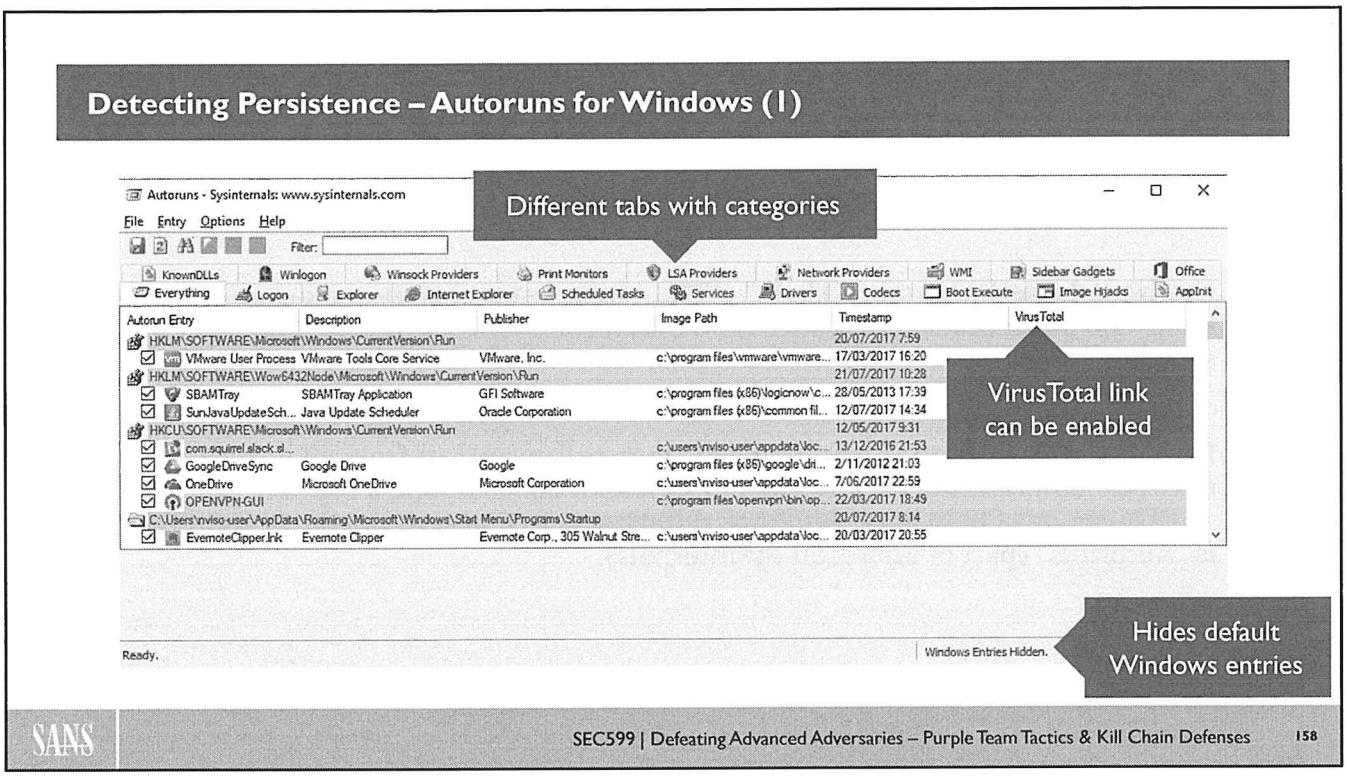
Host-based agents (IDS, AV, EDR, ...) can help you detect and alert upon changes to typical persistence locations such as Windows Services, startup scripts, scheduled tasks...

Detecting persistence often relies on finding anomalies in your environment. Monitor scheduled task creation from common utilities using command-line invocation. Legitimate scheduled tasks could be created during installation of new software or through system administration functions, so beware of false positives. Monitor process execution from the Windows Task Scheduler, taskeng.exe, and changes to the Windows Task Scheduler stores (%systemroot%\System32\Tasks) for change entries related to scheduled tasks that do not correlate with known software, patch cycles, etc. Data and events should not be viewed in isolation but as part of a chain of behavior that could lead to other activities, such as network connections made for C&C, learning details about the environment through discovery, and lateral movement.

Tools such as Sysinternals Autoruns may also be used to detect system changes that could be attempts at persistence, including listing currently scheduled tasks. Look for changes to tasks that do not correlate with known software, patch cycles, etc. Suspicious program execution through scheduled tasks may show up as outlier processes that have not seen before when compared with historical data.

Monitor processes and command-line arguments for actions that could be taken to create tasks. Remote access tools with built-in features may interact directly with the Windows API to perform these functions outside of typical system utilities. Tasks may also be created through Windows system management tools such as Windows Management Instrumentation (WMI) and PowerShell, so additional logging may need to be configured to gather the appropriate data.

© 2019 Erik Van Buggenhout & Stephen Sims



Detecting Persistence – Autoruns for Windows (1)
Autoruns has "the most comprehensive knowledge of auto-starting locations of any startup monitor." It attempts to show you a full overview of what programs or scripts are configured to run during system boot or user login or when built-in Windows applications such as Explorer or Internet Explorer starts. Some of the example locations it monitors include:

- Run, RunOnce and other Registry keys
- Explorer shell extensions
- Scheduled tasks
- Auto-start services
- …

Autoruns is, by default, configured to hide default, known Windows entries, providing you with an in-depth view of what is executed once the computer boots.

So what do all of the colors mean?

- If an entry is highlighted in **pink**: No publisher information was found, or if code verification is on, that the digital signature either doesn't exist or doesn't match, or there is no publisher information.
- If an entry is highlighted in **green**: When comparing against a previous set of Autoruns data, this entry wasn't there last time.
- If an entry is highlighted in **yellow**: The startup entry is present, but the image (or file) it refers to doesn't exist!

In the screenshot above, the "Slack" and "OpenVPN" entries appear to lack publisher information (which can be seen using the color code and the "Publisher" field)!

© 2019 Erik Van Buggenhout & Stephen Sims

Detecting Persistence – Autoruns for Windows (2)



Optimize Output

In the screenshot to the left, we have optimized the Autoruns output and are left with a total of 15 entries. We achieved this by enabling the VirusTotal link (hash lookup) and hiding all entries that are clean on VT!

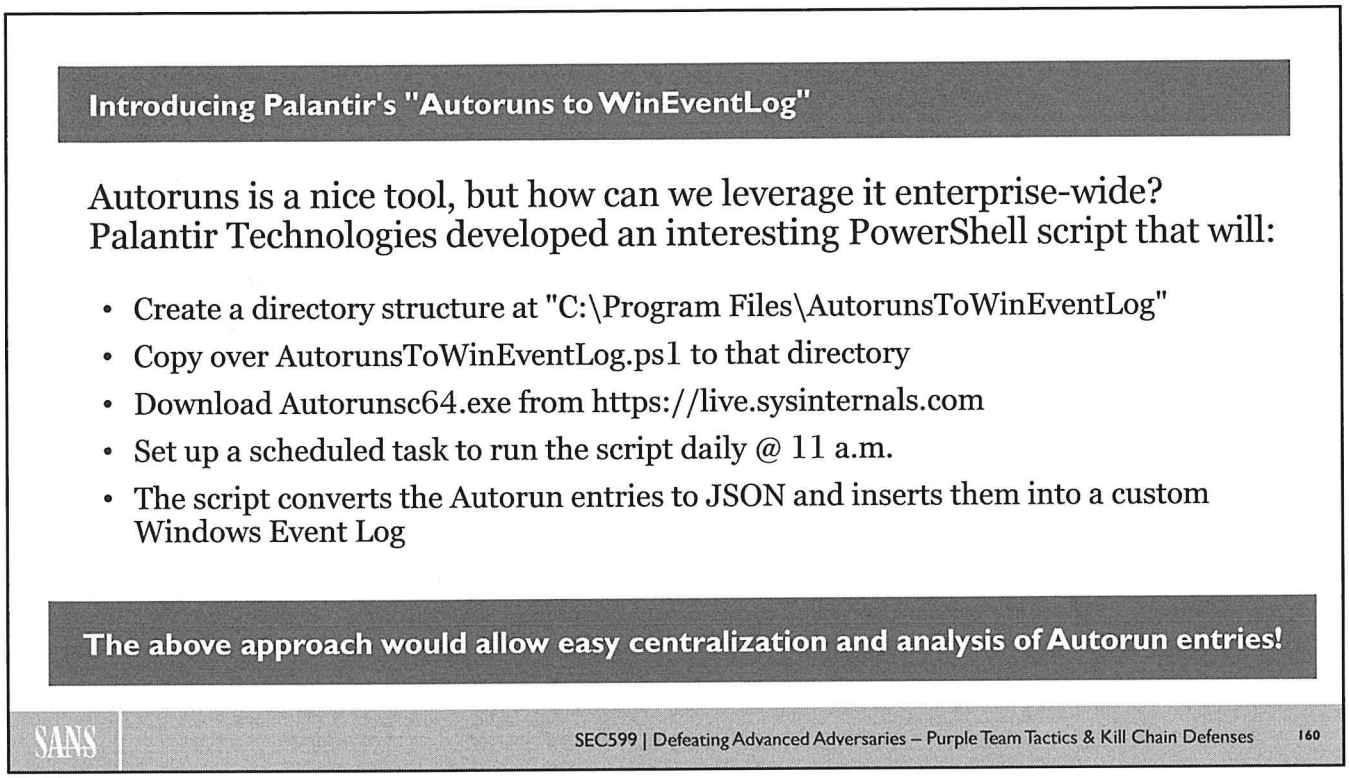
Note the presence of yellow entries, a clean-up might be in order!

Detecting Persistence – Autoruns for Windows (2)

In the screenshot on the slide, we have optimized the Autoruns output and are left with a total of 15 entries. We achieved this by enabling the VirusTotal link (hash lookup) and hiding all entries that are clean on VT! We are also hiding all Autoruns locations that have 0 entries. This list is rather manageable and can be further investigated!

Note the presence of yellow entries, which indicates entries exist, but the image (file) that is referred to is no longer there. A cleanup might be in order in this case!

© 2019 Erik Van Buggenhout & Stephen Sims



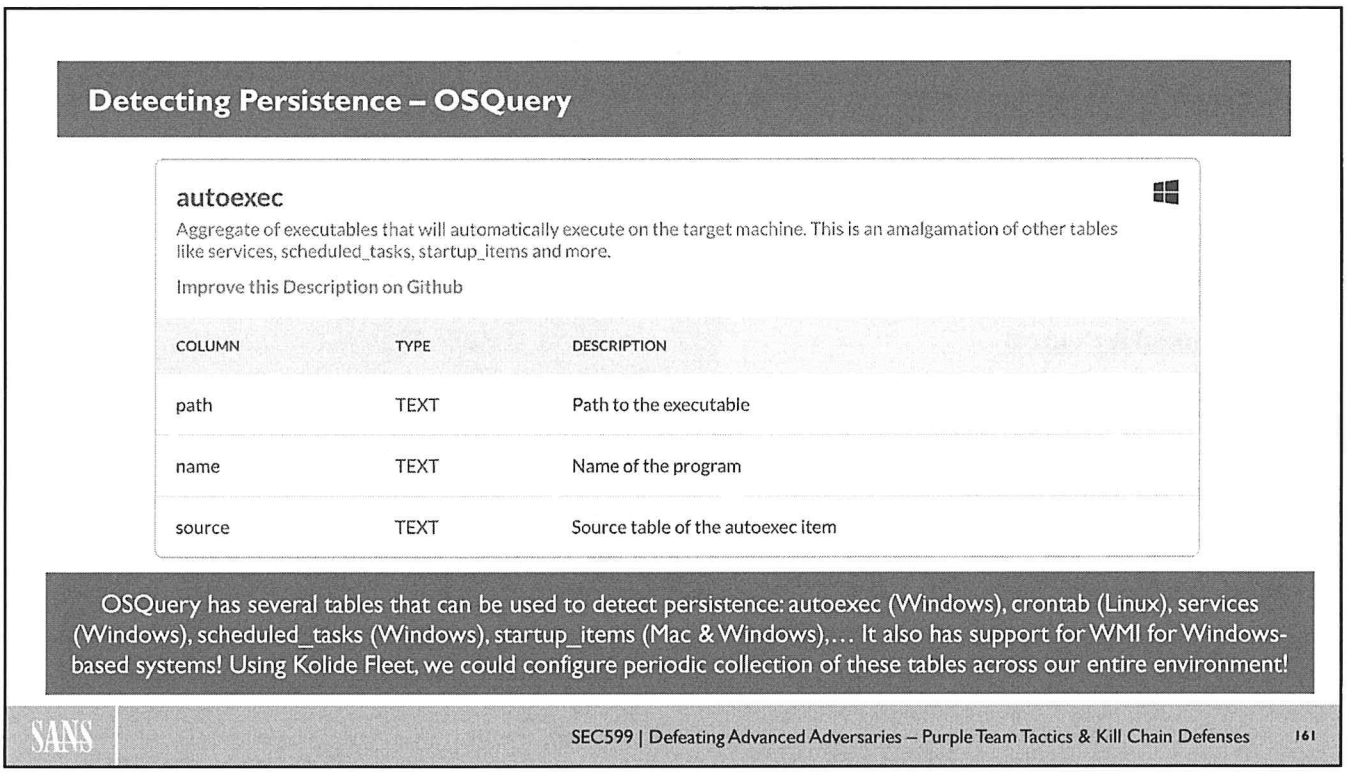
Introducing Palantir's "Autoruns to WinEventLog"
Autoruns is a nice tool, but how can we leverage it enterprise-wide? Palantir Technologies developed an interesting PowerShell script that will:

- Create a directory structure at "C:\Program Files\AutorunsToWinEventLog"
- Copy over AutorunsToWinEventLog.ps1 to that directory
- Download Autorunsc64.exe from https://live.sysinternals.com
- Set up a scheduled task to run the script daily @ 11 a.m.
- The script converts the Autorun entries to JSON and inserts them into a custom Windows Event Log

By using such an approach, the script allows for facilitated centralization and analysis of Autorun entries! You can find the script here:

https://github.com/palantir/windows-event-forwarding/tree/master/AutorunsToWinEventLog

© 2019 Erik Van Buggenhout & Stephen Sims



Detecting Persistence – OSQuery
Should you already have OSQuery installed, another option would be to fetch autorun information using OSQuery! OSQuery has several tables that can be used to detect persistence. This includes:

- autoexec (Windows), which is an amalgamation of other Windows tables;
- crontab (Linux)
- services (Windows)
- scheduled_tasks (Windows)
- startup_items (Mac & Windows)

It also has support for WMI for Windows-based systems! Using Kolide Fleet, we could configure periodic collection of these tables across our entire environment. We could also configure Kolide Fleet to perform "differential" queries, thereby only collecting entries that were changed!

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

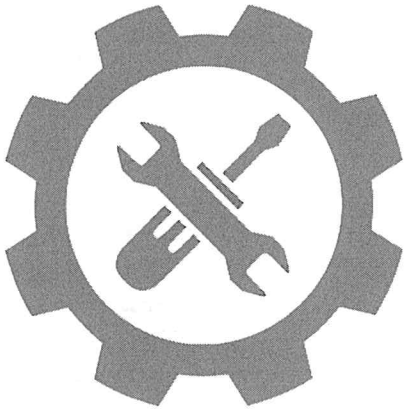
Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Exercise: Catching Persistence Using Autoruns & OSQuery



Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

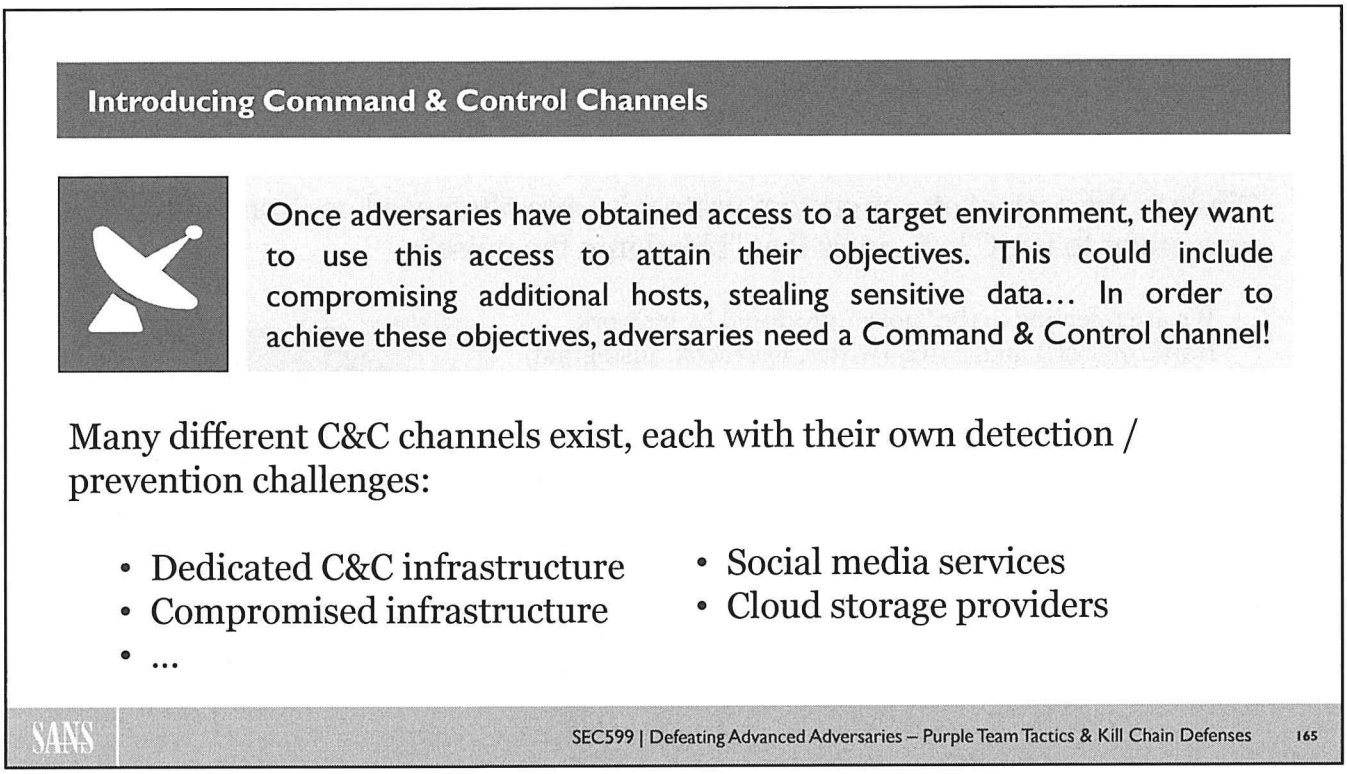
Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims



Introducing Command & Control Channels
Once adversaries have obtained access to a target environment, they want to USE this access to attain their objectives. This could include compromising additional hosts, stealing sensitive data, … In order to achieve these objectives, adversaries need a Command & Control channel!

For example, malware can spy on computers by capturing keystrokes, taking screenshots or screen recordings, recording audio from the built-in microphone, … If this malware would be implanted on all your corporate machines and automatically send all captured data to the adversaries, they would be overwhelmed with data and you would see a spike in outgoing traffic.

What (advanced) adversaries will do in this case is use the implanted malware to locate computers with interesting data and then activate the spying capabilities of said malware. This requires communication between the malware and the adversaries: This is done via so-called command & control channels. On one end, there is the compromised, corporate machine, and on the other end, the server under control of the adversary: The command & control server (C2).

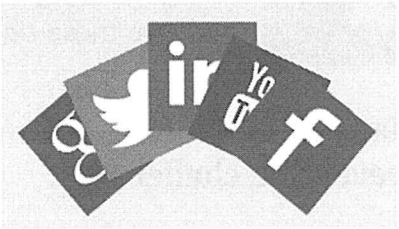
Many communications channels can be used as C2 channel. Malware authors started to use custom made, proprietary command & control channels over TCP connections, but later on, this evolved into using existing channels like DNS and HTTP. The reason is that these custom channels would be more easily detected and blocked in corporate networks.

© 2019 Erik Van Buggenhout & Stephen Sims

Detecting Command & Control Channels

While in the past, exotic or custom protocols were often used, malware authors now prefer to use C2 channels that "blend into the noise":

- We are referring to the "noise" produced by network traffic of social media like Twitter, Facebook, Instagram, even Dropbox, ...
- To be fair, a lot of modern web traffic genuinely looks malicious ☺
- CDN networks are all generating obfuscated, long, HTTP queries that look strange... This is not helping us!



We will illustrate this issue by zooming in one some of the most interesting C&C channels used by adversaries!

Detecting Command & Control Channels

To remain undetected when malware uses command & control channels, malware authors have started to adopt a strategy to "blend into the noise." By this, we mean that malware authors will use existing channels with a high degree of interaction: Messages that flow between clients and servers with high frequency.

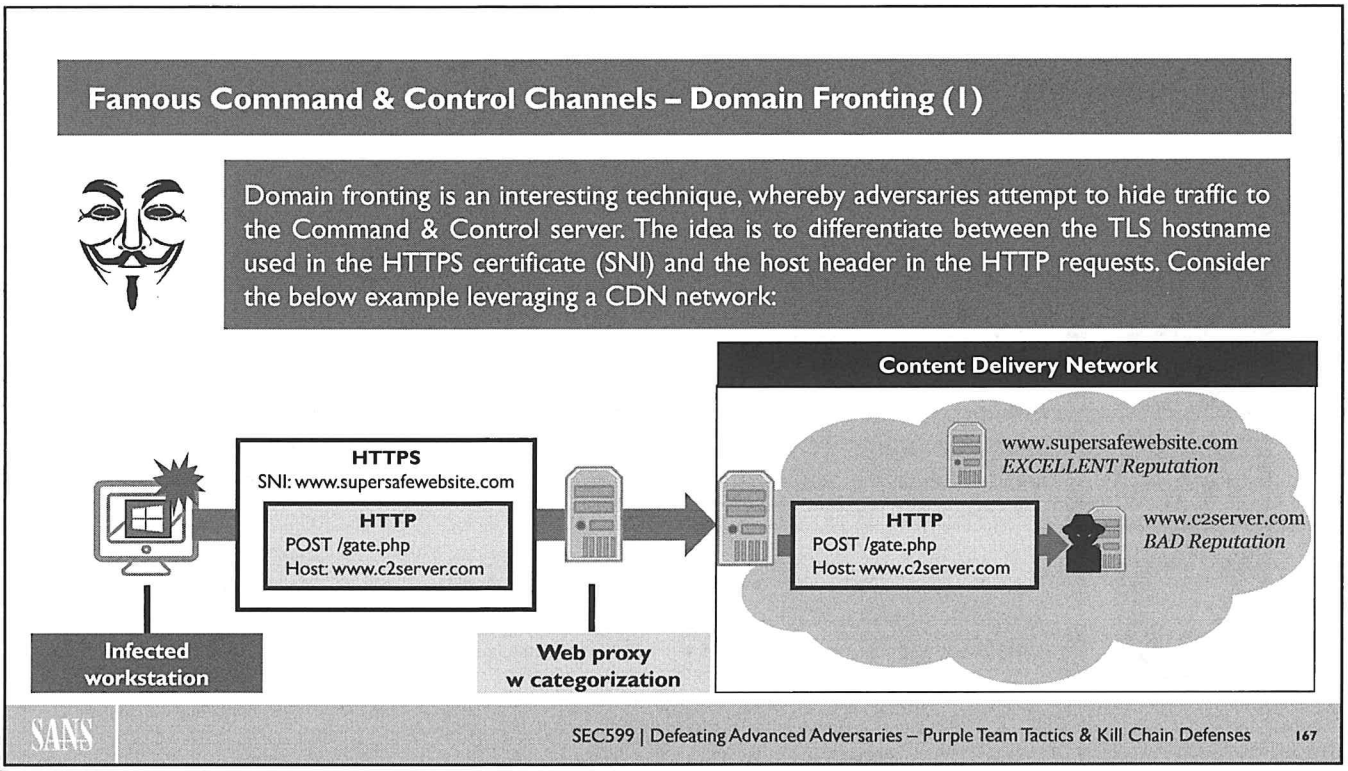
These types of channels can be found in social media applications. Social media produces messages at a high frequency, many corporate users use social media, also for business reasons (LinkedIn for example) and most social media applications have a public aspect. These factors explain why malware authors start to use social media channels as command & control channels: They can "blend into the noise."

Take, for example, malware that uses Twitter as a communication channel. The client (the malware) will have the credentials of a Twitter account and use this to read and write messages. These messages will go to the Twitter servers, like any other Twitter communication. The command & control server has another set of credentials for a Twitter account. Both accounts follow each other, and can thus communicate with each other, publicly or privately (via direct messages). If steganography is used to encode a command in a seemingly innocuous Twitter message, it becomes very hard to uncover these channels.

Twitter is just one example. There are many social media applications that are very popular with your corporate users, like Facebook, Instagram, Pinterest, Google+, Dropbox, … All these social media applications can be used as command & control channel.

In the next slides, we will discuss some famous C&C channels that were used in real attacks!

© 2019 Erik Van Buggenhout & Stephen Sims



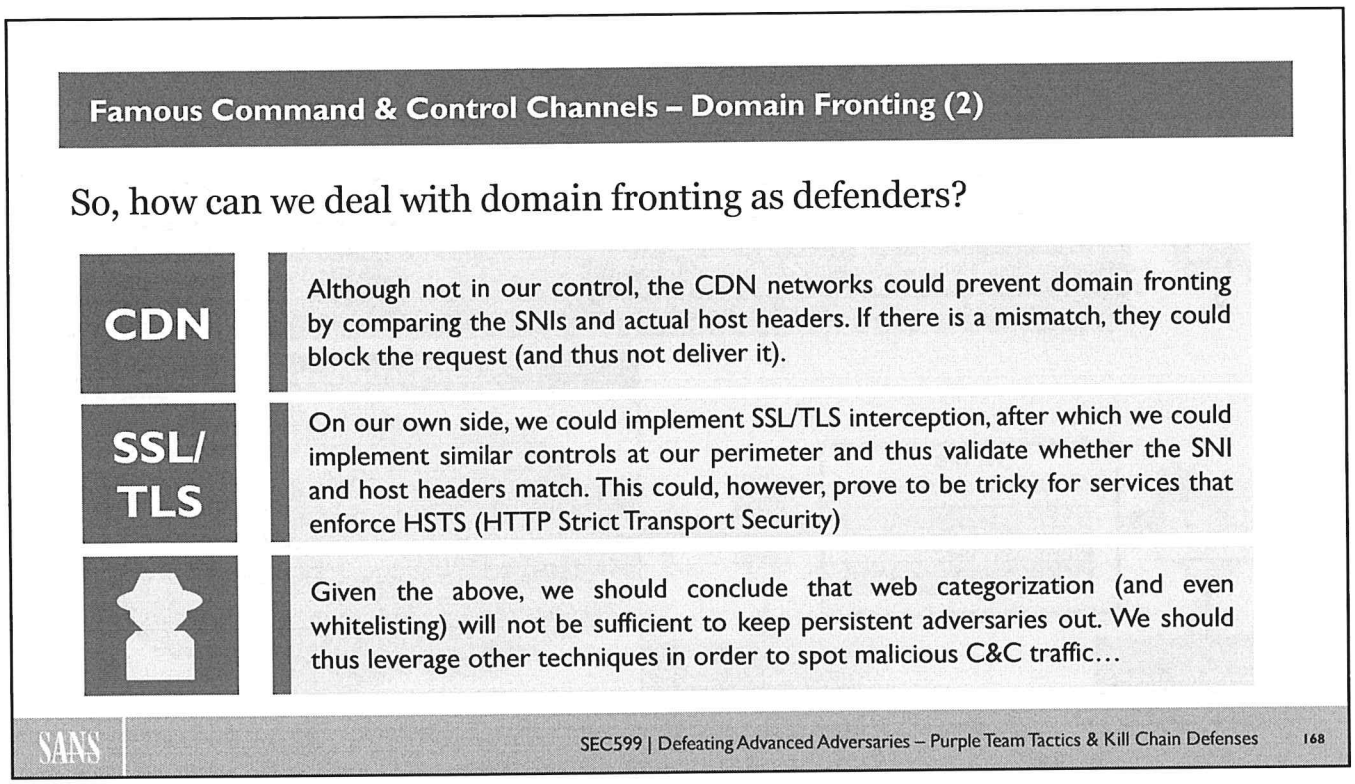
Famous Command & Control Channels – Domain Fronting (1)
Domain fronting is an interesting technique, whereby adversaries attempt to hide traffic to the Command & Control server. The idea is to differentiate between the TLS hostname used in the HTTPS certificate (SNI – Server Name Indication) and the host header in the actual HTTP requests. Consider the below example leveraging a CDN network:

- A workstation is infected with a malware sample that attempts to set up C&C connectivity over HTTPS;
- The initial DNS request is for a benign site (e.g. www.supersafewebsite.com);
- The benign site is also used to set up the HTTPS connection ("www.supersafewebsite.com" is used as the SNI);
- The actual malicious website is only used once the HTTPS connection is set up and is included in the HTTP host header of the HTTP request;
- The upstream server finally forwards the HTTP request to the malicious website.

This specific technique only works when the malicious and benign website are hosted on the same infrastructure, which is often the case in cloud network and CDNs. As of April 2018, Google no longer supports this, claiming it was never an intended / supported feature.

It's interesting to note that this technique is not only being used by adversaries, but also in attempts to bypass censorship (e.g. in authoritative regimes).

© 2019 Erik Van Buggenhout & Stephen Sims



Famous Command & Control Channels – Domain Fronting (2)
So how can we deal with domain fronting as defenders?

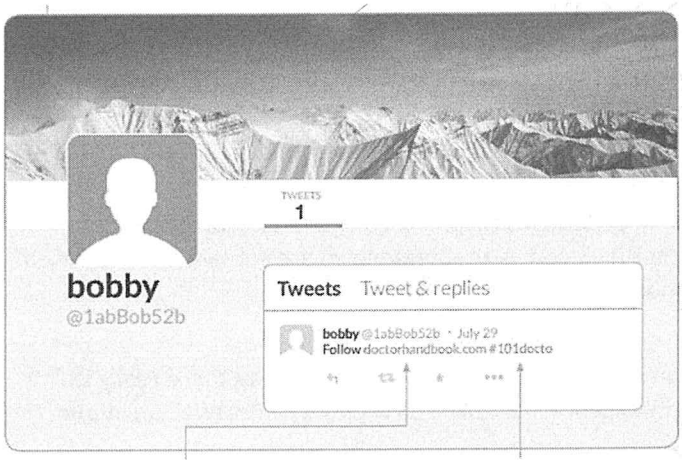
- Although not in our control, the CDN networks could prevent domain fronting by comparing the SNIs and actual host headers. If there is a mismatch, they could block the request (and thus not deliver it). This is, however, a tricky solution to implement, as it's out of the control of us as defenders;
- On our own side, we could implement SSL/TLS interception, after which we could implement similar controls at our perimeter and thus validate whether the SNI and host headers match. This could, however, prove to be tricky for services that enforce HSTS (HTTP Strict Transport Security);
- Given the above, we should conclude that web categorization (and even whitelisting) will not be sufficient to keep persistent adversaries out. We should thus leverage other techniques in order to spot malicious C&C traffic…

An excellent article detailing the origins of domain fronting can be found here:

https://www.bamsoftware.com/papers/fronting/

© 2019 Erik Van Buggenhout & Stephen Sims

Famous Command & Control Channels – Hammertoss Twitter & Steganography



Source: https://www2.fireeye.com/rs/848-DID-242/images/rpt-apt29-hammertoss.pdf

Hammertoss (APT-29) uses the following C&C strategy:

- Checks a different Twitter handle daily
- If handle is found, extract images from URLs in tweets
- The images include hidden data, inserted in the image using steganography and an encryption key
- The offset (in this case 101) and the decryption key (in this case doctor) are included in the tweet
- The data (typically commands) are extracted and executed

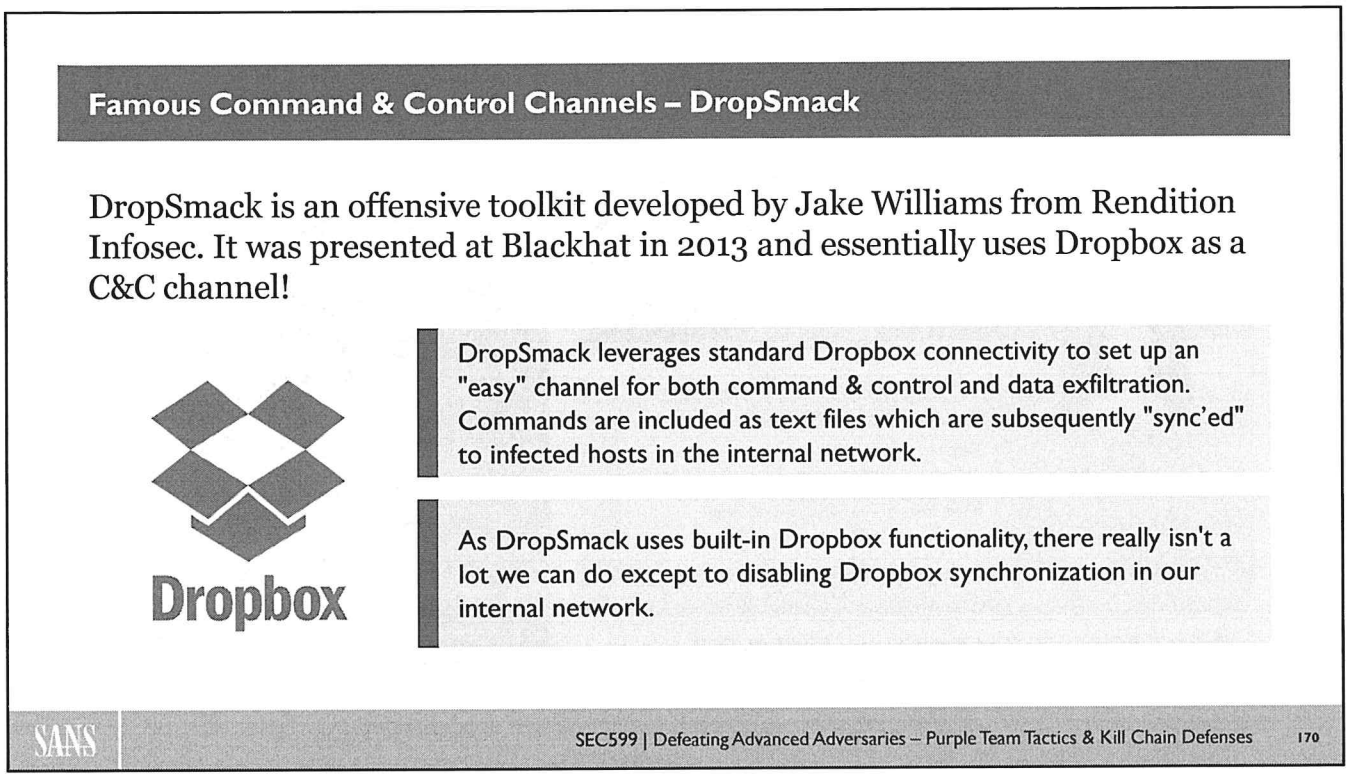
Famous Command & Control Channels – Hammertoss Twitter & Steganography

Another interesting example of a command & control channel is the use of steganography in Twitter images by the HAMMERTOS (by APT-29). In short, the following took place:

- Once installed, the HAMMERTOSS backdoor generates and looks for a specific Twitter handle (a different one every day). A built-in algorithm will generate a daily handle (e.g. 1abBob52b) and will subsequently check this twitter account.
- If the APT group has not registered the handle, the malware will take no action and wait for the next day.
- The APT group will register the protocol name and post tweets using the account. The tweets have the following format:
  - They include a link to an image
  - The image includes a hidden message (typically a command that is to be executed)
  - The message is hidden in the image using steganography (it is encrypted and placed in the image using a symmetric encryption key)
  - The offset (where the message is hidden) and the encryption key are included in the tweet
- Upon finding an image, the backdoor will download the image, extract the message and execute the requested command.

An excellent analysis of the HAMMERTOSS backdoor was done by FireEye analysts and can be found here: https://www2.fireeye.com/rs/848-DID-242/images/rpt-apt29-hammertoss.pdf

© 2019 Erik Van Buggenhout & Stephen Sims

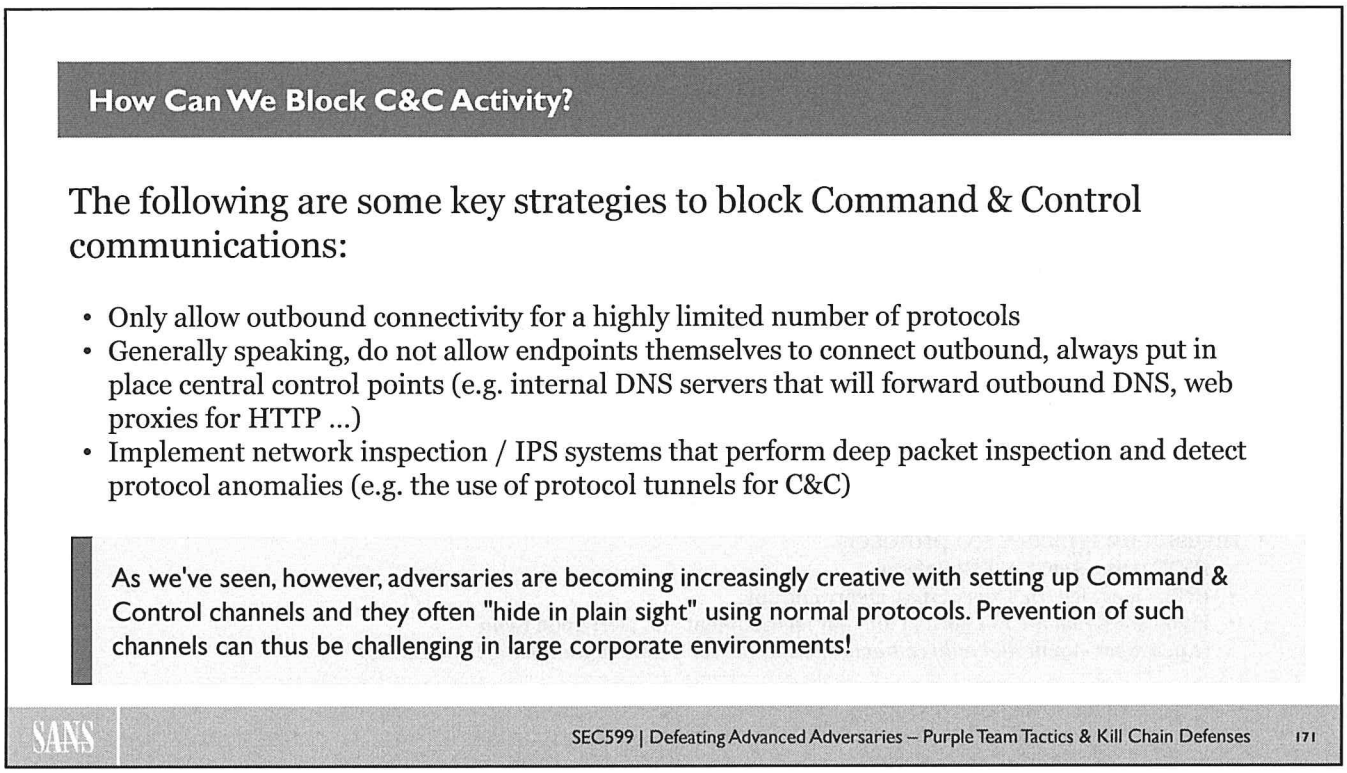


Famous Command & Control Channels – DropSmack
DropSmack is an offensive toolkit developed by Jake Williams from Rendition Infosec. It was presented at Blackhat in 2013 and essentially uses Dropbox as a C&C channel!

- DropSmack leverages standard Dropbox connectivity to set up an "easy" channel for both command & control and data exfiltration. Commands are included as text files which are subsequently "sync'ed" to infected hosts in the internal network. It can, of course, also be used in the other direction to "sync" files out of the network…
- As DropSmack uses built-in Dropbox functionality, there really isn't a lot we can do except to disabling Dropbox synchronization in our internal network.

It's important to note that this is not really "a vulnerability" in Dropbox. This type of technique could be used against the vast majority of cloud-based file sharing services. This opens the discussion whether we should allow cloud-based file sharing within the enterprise. Given the rise of cloud within enterprises, this will be a hot discussion topic. ☺

© 2019 Erik Van Buggenhout & Stephen Sims



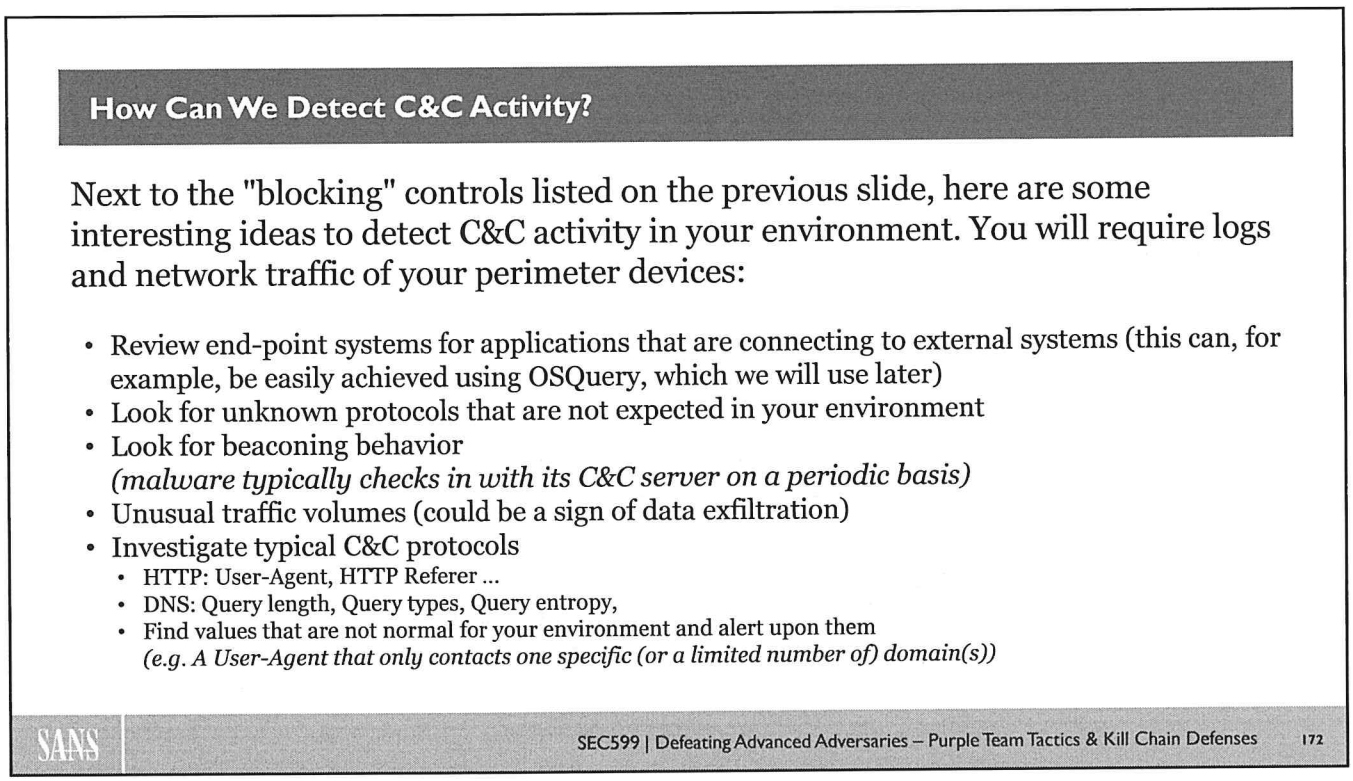
How Can We Block C&C Activity?

The following are some key strategies to block Command & Control communications:

- Only allow outbound connectivity for a highly limited number of protocols
- Generally speaking, do not allow endpoints themselves to connect outbound; always put in place central control points (e.g. internal DNS servers that will forward outbound DNS, web proxies for HTTP …)
- Implement network inspection / IPS systems that perform deep packet inspection and detect protocol anomalies (e.g. the use of protocol tunnels for C&C)
- Configure network devices to check that the type of traffic matches the port. For example, we only allow HTTP on port 80, we block SSH over port 80 but allow it over port 22.

As we've seen, however, adversaries are becoming increasingly creative with setting up Command & Control channels, and they often "hide in plain sight" using normal protocols. Prevention of such channels can thus be challenging in large corporate environments! Let's investigate how we can try detecting the use of Command & Controls!

© 2019 Erik Van Buggenhout & Stephen Sims



How Can We Detect C&C Activity?
Next to the "blocking" controls listed on the previous slide, here are some interesting ideas to detect C&C activity in your environment. In any case, if we want to start detecting Command & Control activity, we will need to start performing network monitoring and collecting logs from our perimeter devices. Additionally, it might be worth reviewing end-points for applications that are connecting to external systems (this can, for example, be easily achieved using OSQuery, which we will use tomorrow)

Furthermore, look for unknown protocols that are not expected in your environment. This is fairly easy, as exotic protocols will raise alerts and are easy to spot.

Implement protocol specific gateways (such as outbound web proxies) that can analyze the protocol and analyze protocol fields and settings. Highly popular protocols used for C&C connectivity include HTTP and DNS. In HTTP traffic, interesting fields to analyze include, for example, the User-Agent and the "Referer" request headers. For DNS, we can look at the query length, query types and entropy.

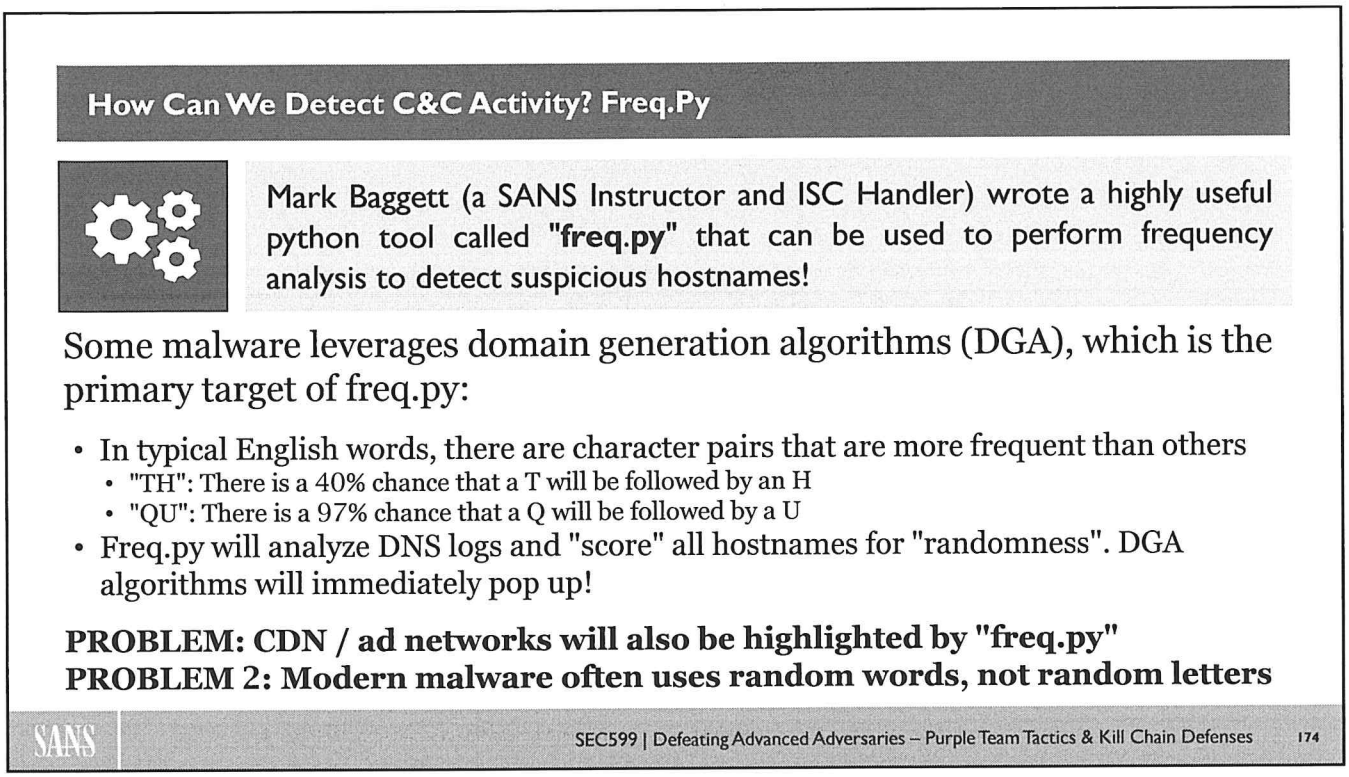
Even advanced adversaries will not always invent new protocols but reuse existing command & control protocols. By configuring IDS and IPS systems with up-to-date rules that detect various known command & control channels, we increase our chances to detect known command & control protocols. Properly configuring proxies and firewalls to only allow known and trusted protocols and websites, we can further reduce the use of command & control channels.

Anomaly-based network detection systems can help us detect new, unknown command & control channels. These detection systems come with advanced network protection devices and are sometimes running on dedicated computers to provide full processing power. Detecting beaconing is one such application. By restricting the initiation of network connections to devices inside our network, the command & control server cannot connect directly to the compromised machine. It is the compromised machine that has to initiate the connection on a regular basis to check if the command & control server has instructions. This is called beaconing, and by analyzing the frequency of connections, regular beaconing can be detected.

© 2019 Erik Van Buggenhout & Stephen Sims

One of the problems we face is that more and more our adversaries will use encrypted command & control channels (like SSL/TLS), which drastically reduces our opportunities to inspect traffic. TLS interception can help us here.

© 2019 Erik Van Buggenhout & Stephen Sims



How Can We Detect C&C Activity? Freq.py

Mark Baggett (a SANS Instructor and ISC Handler) wrote a highly useful python tool called "freq.py" that can be used to perform frequency analysis to detect suspicious hostnames! Some malware leverages domain generation algorithms (DGA), an interesting example is the Zeus banking trojan.

Malware families that use DGA are the primary detection target of freq.py… How does it work?

- In typical English words, there are character pairs that are more frequent than others:
  - "TH": There is a 40% chance that a T will be followed by an H
  - "QU": There is a 97% chance that a Q will be followed by a U
- Freq.py will analyze DNS logs and "score" all hostnames for "randomness". DGA algorithms will immediately pop up!

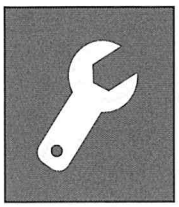
The development and usage of freq.py is described in detail in Mark Baggett's SANS ISC post: https://isc.sans.edu/forums/diary/Detecting+Random+Finding+Algorithmically+chosen+DNS+names+DGA/19893/

It's important to note that this approach has the following limitations:

1. First of all, CDN / ad networks will typically use randomly generated (sub-)domain names as well, which will also be highlighted by "freq.py". This could be solved by whitelisting certain domains from the list (then again, you might whitelist actual malicious domains)
2. Secondly, modern malware that relies on DGA sometimes uses a new approach: Instead of using random letter, they use random words to generate domain names. This will defeat freq.py, as it's looking for uncommon characters pairs, not uncommon combinations of words…

© 2019 Erik Van Buggenhout & Stephen Sims

How Can We Detect C&C Activity? RITA



RITA (Real Intelligence Threat Analytics) is a project first started by Black Hills Information Security, which is now being further maintained by Offensive CounterMeasures. You can find its source code on GitHub!

RITA is an open-source framework for network traffic analysis. It was designed to ingest Bro / Zeek (network security monitor) logs and has the following interesting analysis options:

- Search for signs of **beaconing behavior** in and out of your network.
- Search for signs of **DNS-based covert channels.**
- Search for **suspicious, long URLs** that often indicate malware.

RITA supports a few other items, but the above are most relevant for detecting C&C traffic!

How Can We Detect C&C Activity? RITA

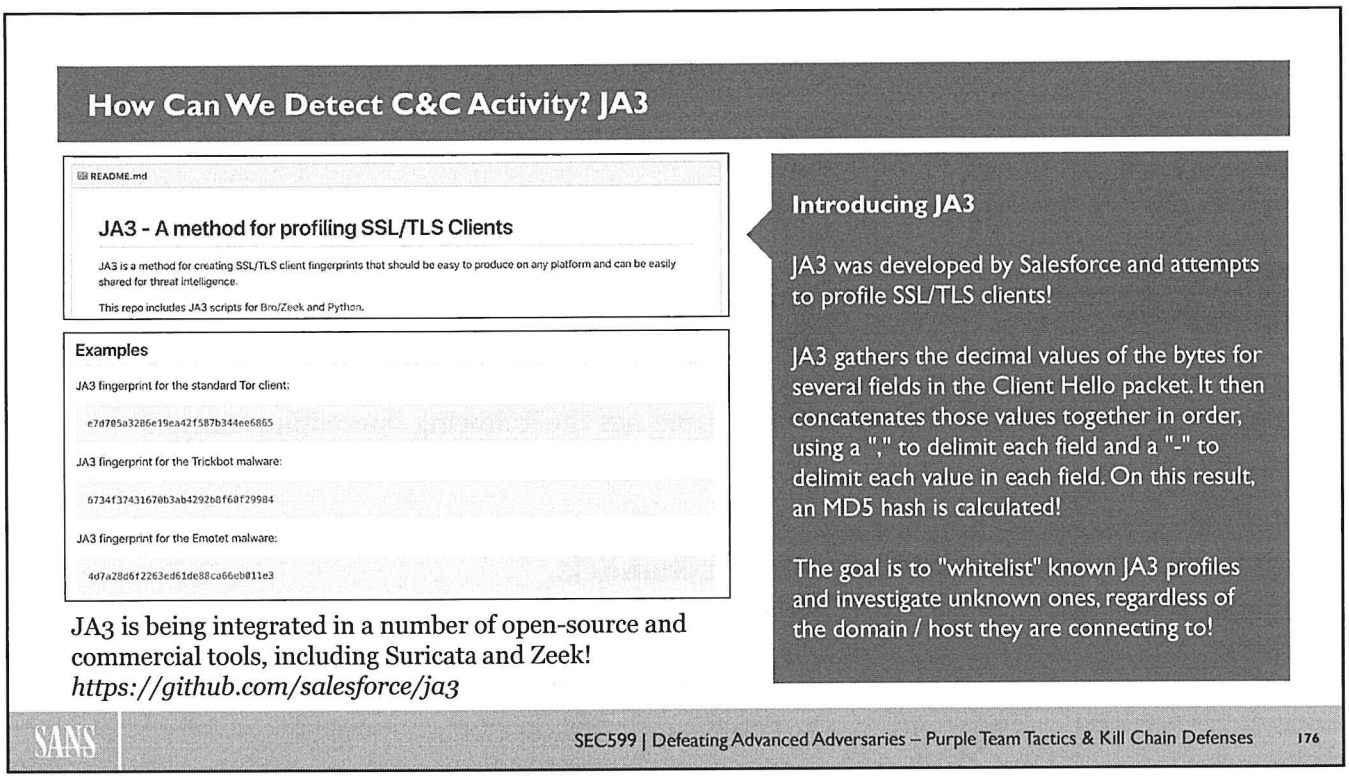
RITA (Real Intelligence Threat Analytics) is a project first started by Black Hills Information Security, which is now being further maintained by Offensive CounterMeasures. You can find its source code on the following GitHub page: https://github.com/ocmdev/rita

RITA is an open-source framework for network traffic analysis. It was designed to ingest Bro / Zeek (network security monitor) logs and has the following interesting analysis options:

- Search for signs of beaconing behavior in and out of your network;
- Search for signs of DNS based covert channels;
- Search for suspicious, long, URL's that are often indicative of malware;

RITA does support other analysis types as well, but the ones listed above are most relevant to detect Command & Control traffic. Please refer to RITA's GitHub page for additional information, including for example install, configuration and usage guidelines.

© 2019 Erik Van Buggenhout & Stephen Sims



How Can We Detect C&C Activity? JA3

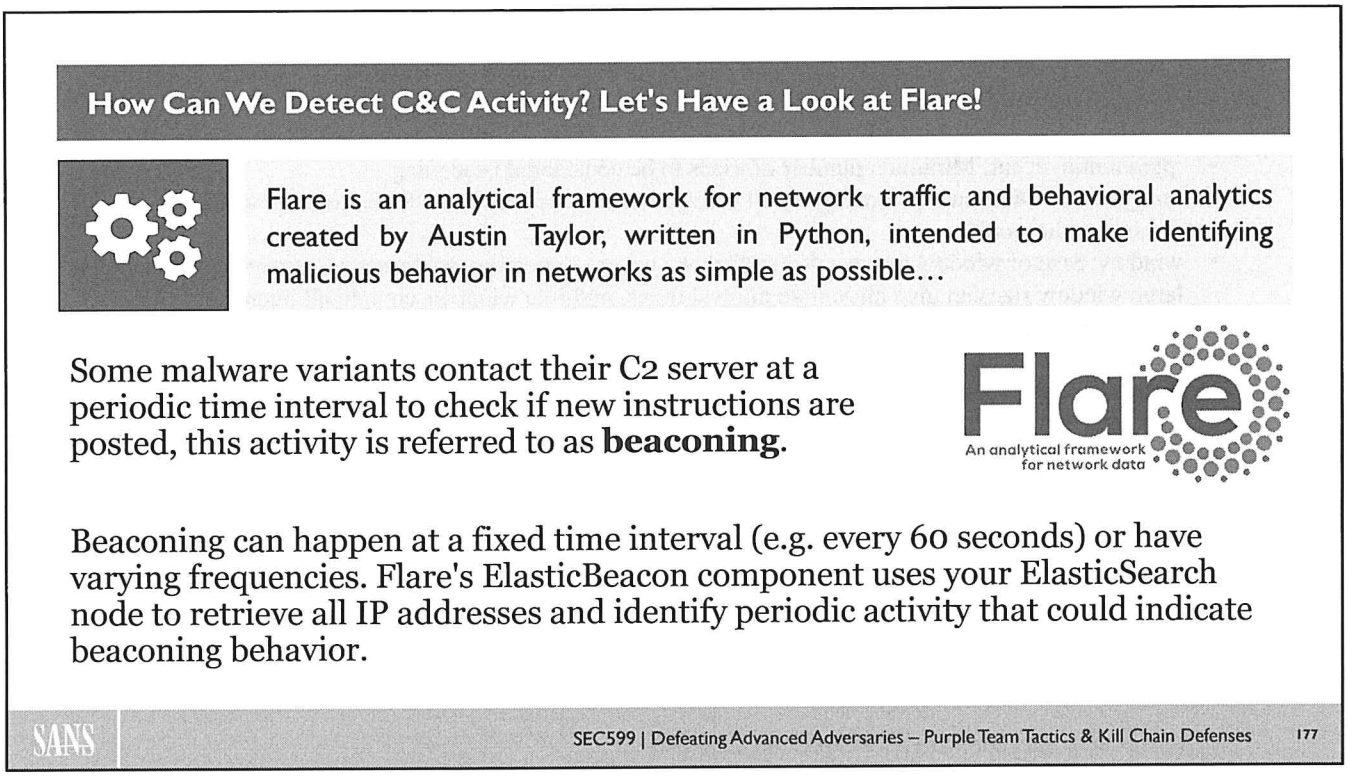
As we discussed previously, trying to detect malicious traffic by using known IOCs (known bads) is a losing strategy! What if we could, however, detect malicious encrypted traffic by fingerprinting the SSL/TLS client that is generating it in the first place? This is exactly what JA3 hopes to achieve!

JA3 was developed by Salesforce and attempts to profile SSL/TLS clients! JA3 gathers the decimal values of the bytes for the following fields in the Client Hello packet; SSL Version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve Formats. It then concatenates those values together in order, using a "," to delimit each field and a "-" to delimit each value in each field.

On this result, an MD5 hash is calculated! The goal is to "whitelist" known JA3 profiles and investigate unknown ones, regardless of the domain / host they are connecting to!

JA3 is being integrated in a number of open-source and commercial tools, including Suricata & Zeek. It's hosted and developed at: https://github.com/salesforce/ja3! On the GitHub page, they host a long list of known JA3 fingerprints, which could theoretically be whitelisted!

© 2019 Erik Van Buggenhout & Stephen Sims



How Can We Detect C&C Activity? Let's have a look at Flare!
Flare is an analytical framework for network traffic and behavioral analytics created by Austin Taylor, written in Python, intended to make identifying malicious behavior in networks as simple as possible. You can find the source code for Flare on the following GitHub page: https://github.com/austin-taylor/flare.

As we discussed earlier in the C2 chapter, malware often contacts a C2 server to get instructions on what actions they need to execute. Some malware variants contact these C2 servers at a periodic time interval; this activity is referred to as beaconing.

Beaconing can occur at fixed time intervals or have varying frequencies:

- Fixed time intervals: Malware reaches out to its C2 server at a fixed time interval (hard interval) such as every 60 seconds or each hour on the hour;
- Varying frequencies: Malware reaches out to its C2 server at varying frequencies; this could be done by a random timer embedded into the malware or by looking for specific user actions to initiate the communication (e.g. communicate every time the user unlocks the workstation).

In order for the ElasticBeacon function of Flare to work, it needs to have access to an ElasticSearch that contains flow data (created by Bro or Suricata). Flare runs on your local machine and can be given access to your ElasticSearch node using the following command:

ssh -NfL 9200:localhost:9200 elasticsearch-user@10.10.10.80, where 10.10.10.80 is the IP address of your ElasticSearch node.

In order to test the connection before using Flare, you can execute "*curl localhost:9200*" which should show you the ElasticSearch information.

© 2019 Erik Van Buggenhout & Stephen Sims

Now that the connection is ready, you can start configuring the .ini file in the configs directory; this file contains settings to connect to your ElasticSearch and parameters to fine-tune beaconing detection. The items to configure in the .ini file are explained as follows:

- :param min_occur: Minimum number of triads to be considered beaconing
- min_percent: Minimum percentage of all connection attempts that must fall within the window to be considered beaconing
- window: Size of window in seconds in which we group connections to determine percentage, using a large window size can give inaccurate interval times, multiple windows contain all interesting packets, so the first window to match is the interval
- threads: Number of cores to use
- period: Number of hours to locate beacons for
- min_interval: Minimum interval between events to consider for beaconing behavior
- es_host: IP Address of elasticsearch host (default is localhost)
- es_timeout: Sets timeout to 480 seconds
- kibana_version: 4 or 5 (query will depend on version)

Once all settings in the .ini file have been configured, you can execute Flare from the command line using following command:

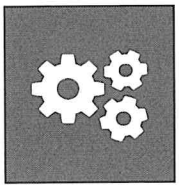
flare_beacon --group --whois --focus_outbound -c configs/elasticsearch.ini -csv beacons.csv

The options used in this command are explained as follows:

- --group: Groups the results for better visibility
- --whois: Performs whois lookup on IP addresses
- --focus_outbound: Filters out multicast, private and broadcast addresses from destination IPs
- -c: The .ini config file for Flare to use
- --csv: Output results to CSV (HTML is also possible using --html, however, for further analysis, CSV files are easier to use)

© 2019 Erik Van Buggenhout & Stephen Sims

How Can We Detect C&C Activity? Flare (Cont.)



Next to beaconing detection, Flare also has a few other interesting options:

- Check domains against the Alexa domain list
- Perform WHOIS lookups
- DGA prediction

Flare will provide the following output fields per domain, which will allow you to determine whether malicious beaconing is happening or not (legitimate programs often show signs of beaconing as well, for example, Dropbox):

- **bytes_toserver**: Total sum of bytes sent from source IP address to server
- **dest_degree**: Number of distinct source IP addresses that communicate to this server
- **occurrences**: Number of sessions between source IP and server identified as beaconing
- **percent**: Percent of traffic between source IP and server that is considered to be beaconing
- **interval**: Intervals between each beacon in seconds

How Can We Detect C&C Activity? Flare (Cont.)
Flare will provide the following output fields per domain, which will allow you to determine whether malicious beaconing is happening or not (legitimate programs often show signs of beaconing as well, for example, Dropbox).

- **bytes_toserver**: Total sum of bytes sent from source IP address to server
- **dest_degree**: Number of distinct source IP addresses that communicate to this server
- **occurrences**: Number of sessions between source IP and server identified as beaconing
- **percent**: Percent of traffic between source IP and server that is considered to be beaconing
- **interval**: Intervals between each beacon in seconds

By filtering out known good servers, you can bring down the number of interesting results, which you then subsequently can look up in your ELK stack for further in-depth analysis.

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

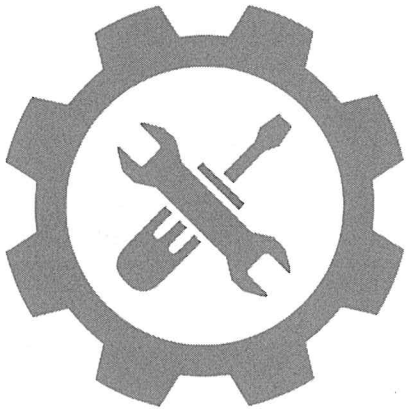
Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Exercise - Detecting C&C Channels Using Suricata, JA3, & RITA



Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims

Course Roadmap

- Day 1: Introduction & Reconnaissance
- Day 2: Payload Delivery & Execution
- **Day 3: Exploitation, Persistence and Command & Control**
- Day 4: Lateral Movement
- Day 5: Action on Objectives, Threat Hunting & Incident Response
- Day 6: APT Defender Capstone

SEC599.3

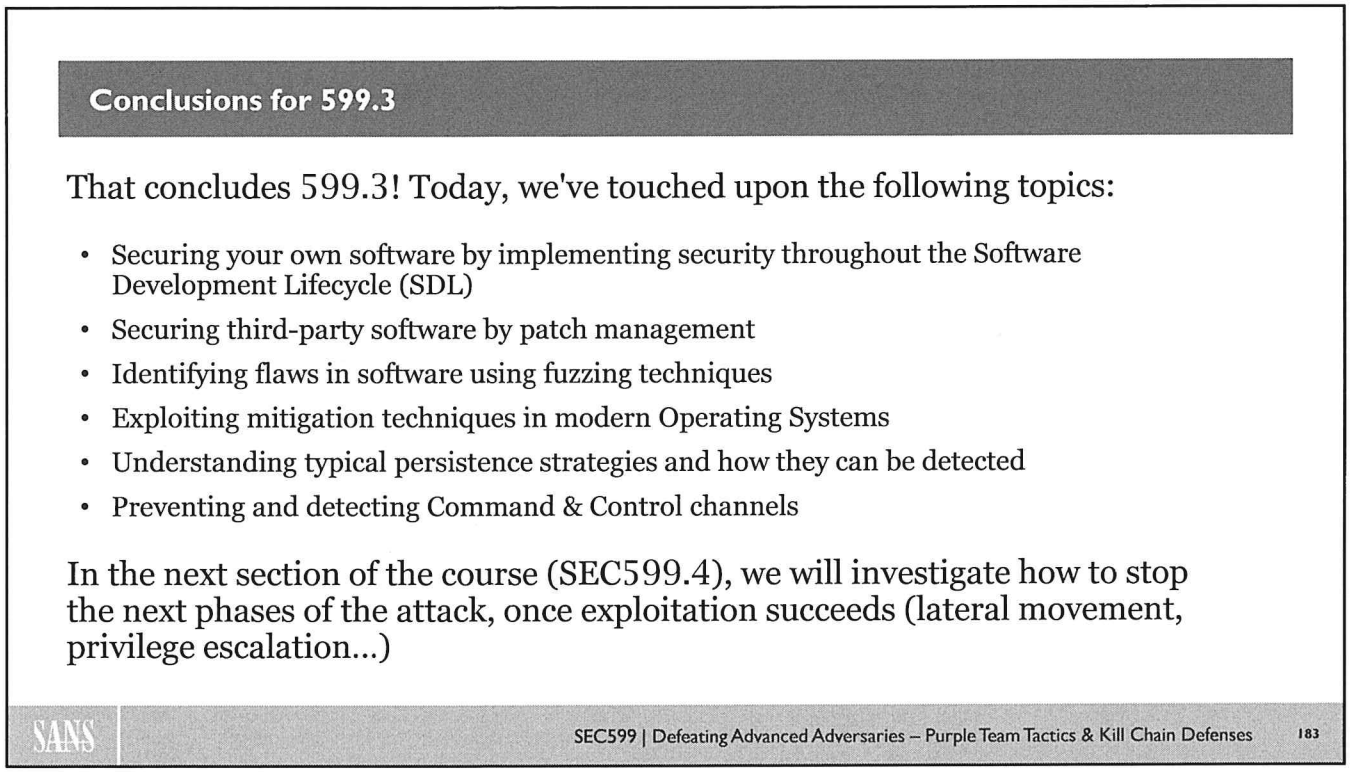
Protecting applications from exploitation
- Software Development Lifecycle (SDL) & Threat Modeling
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation using Compile-Time Controls
- Exploit Mitigation Techniques – ExploitGuard, EMET & others
- Exercise: Exploit Mitigation using ExploitGuard

Avoiding installation
- Typical persistence strategies
- How do adversaries achieve persistence?
- Exercise: Catching persistence using Autoruns & OSQuery

Foiling Command & Control
- Detecting Command & Control channels
- Exercise: Detecting C&C channels using Suricata, JA3, & RITA

This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims



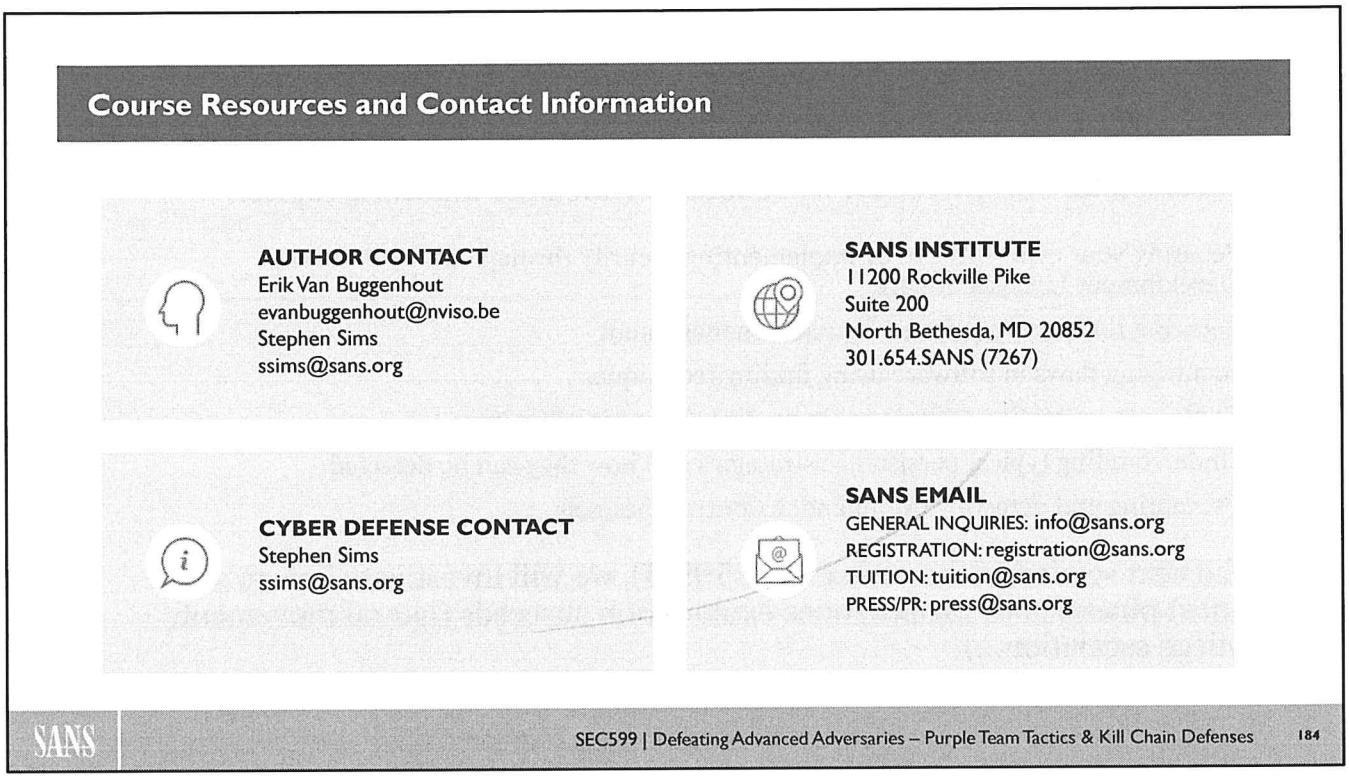
Conclusions for 599.3

So long, 599.3! Today, we looked into how initial exploitation by adversaries can be prevented and detected. Among others, we touched upon the following topics:

- Securing your own software by implementing security throughout the Software Development Lifecycle (SDL)
- Securing third-party software by patch management
- Identifying flaws in software using fuzzing techniques
- Exploiting mitigation techniques in modern Operating Systems
- Understanding typical persistence strategies and how they can be detected
- Preventing and detecting Command & Control channels

In the next section of the course (SEC599.4), we will investigate how to stop the next phases of the attack, once exploitation succeeds (lateral movement, privilege escalation…)

© 2019 Erik Van Buggenhout & Stephen Sims



This page intentionally left blank.

© 2019 Erik Van Buggenhout & Stephen Sims