A Pinch of Evasion, A Sprinkle of Tradecraft

@FortyNorthSec

Intro Slides

- Offensive Security Engineer@FortyNorth Security
- Tool Dev
  - Coeus
  - PersistAssist
- Pentester/Red Teamer + Red Team R&D/Researcher
- Situational Awareness Enthusiast
- BlackHat Instructor
- Whipping up culinary catastrophes since ‘14

fortynorth

What this talk is not

- No novel techniques, everything discussed is public research
- Nothing groundbreaking, earthshattering, world ending
- Mal Dev or In-depth evasion course
  - Though the WinAPI will be covered, we won't go into writing Proc Injectors or the like

What this talk is

- An attempt to lay a foundational knowledge base for general evasion operations
- How some of these techniques can be detected by defenders
  - Evading detections
- C2/tool agnostic. The intent is to teach techniques, not how to use a tool
- Develop better tradecraft and an OPSEC-considerate mindset

Also Disclaimer

- I am not responsible for any unauthorized/malicious use of the information shared over the course of this presentation.

Agenda

- Definitions
- What AV is and how they work
- Bypass techniques for AV
- WinAPI architecture and usage
- AMSI/ETW overview and bypasses
- What EDR is and how it works
- Hooking methods
- Bypassing Hooks
- Tradecraft and OPSEC considerations

Definitions

- AV –
- EDR –
- *DR –
  - Spoiler alert:
- WinAPI –
- AMSI –
- ETW –
- Tradecraft -

fortynorth

Definitions

- AV – AntiVirus
- EDR – Endpoint Detection and Response
- *DR – something something Detection and Response
  - Spoiler alert: they're all the same
- WinAPI – Windows API
- AMSI – AntiMalware Scanning Interface
- ETW – Event Tracing for Windows
- Tradecraft - "the techniques, methods and technologies used in modern espionage" - wikipedia (replace espionage w/ red team operations)

What is AV

- Detections mechanism
  - Signature
  - Behavioral/Heuristics
  - Reputation
- Signature
  - Searches for known bad hashes and strings
  - i.e "mimikatz"
- Behavioral/Heuristics
  - Files scanned have their operations scrutinized within a sandbox
- Reputation
  - "Hey I've never seen this before, bye Felicia"

Bypassing AV – Signatures

- Obfuscation, obfuscation, obfuscation
  - Can be as simple as XOR'ing code
- AES your strings!
  - No….seriously. malware analysis story time
- Base64 your shellcode a few times!
  - This bypassed defender at one point (and still might)
- Removing known bad strings
  - Names of known researchers
  - Rename libs and variables

Bypassing AV – Signatures

```
PS C:\> write-host "invoke-mimikatz"
At line:1 char:1
+ write-host "invoke-mimikatz"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\> write-host "mimikatz"
mimikatz
```

- Removing known bad strings
  - Names of known researchers
  - Rename libs and variables

fortynorth

Bypassing AV - Behavioral

- Some AVs won't bother if a file is too large
  - Pro tip: implant size should be ~600mb
  - NSA's Ackchyually: implant size should be under 150kb

- Sleep is a bit of a no-no, sandboxes will usually fast forward subjects
  - Instead, have the program perform some sort of operations

Bypassing AV - Reputation

- Leverage things the AV does recognize

- lolbas
  - MSBuild.exe
  - InstallUtil.exe

Bypassing AV – General Tips

- Compile your own code
  - sometimes, something as simple as recompiling code does the trick…thanks Avast
- Strip comments and obfuscate or remove strings that may trigger AV
  - https://github.com/ChrisTruncer/PenTestScripts/blob/master/CobaltScripts/removecomments.py
- Look into AV Bypass tools

Bypassing AV – DefenderCheck

- Basically running defender against your binary to check exactly what is being flagged where

```
PS C:\Users\Matt\Desktop> .\DefenderCheck.exe C:\Temp\mimikatz.exe
Target file size: 933528 bytes
Analyzing...

[!] Identified end of bad bytes at offset 0xA185B in the original file
File matched signature: "HackTool:Win64/Mikatz!dha"

00000000   00 5F 00 64 00 6F 00 4C  00 6F 00 63 00 61 00 6C   ·_·d·o·L·o·c·a·l
00000010   00 20 00 3B 00 20 00 22  00 25 00 73 00 22 00 20   · ·;· ·"·%·s·"·
00000020   00 6D 00 6F 00 64 00 75  00 6C 00 65 00 20 00 6E   ·m·o·d·u·l·e· ·n
00000030   00 6F 00 74 00 20 00 66  00 6F 00 75 00 6E 00 64   ·o·t· ·f·o·u·n·d
00000040   00 20 00 21 00 0A 00 00  00 00 00 00 00 0A 00 25   · ·!···********%
00000050   00 31 00 36 00 73 00 00  00 00 00 00 00 20 00 20   ·1·6·s******· ·
00000060   00 2D 00 20 00 20 00 25  00 73 00 00 00 20 00 20   ·-· · ·%·s··· ·
00000070   00 5B 00 25 00 73 00 5D  00 00 00 00 00 00 00 00   ·[·%·s·]********
00000080   00 00 00 00 00 45 00 52  00 52 00 4F 00 52 00 20   *****E·R·R·O·R·
00000090   00 6D 00 69 00 6D 00 69  00 6B 00 61 00 74 00 7A   ·m·i·m·i·k·a·t·z
000000A0   00 5F 00 64 00 6F 00 4C  00 6F 00 63 00 61 00 6C   ·_·d·o·L·o·c·a·l
000000B0   00 20 00 3B 00 20 00 22  00 25 00 73 00 22 00 20   · ·;· ·"·%·s·"·
000000C0   00 63 00 6F 00 6D 00 6D  00 61 00 6E 00 64 00 20   ·c·o·m·m·a·n·d·
000000D0   00 6F 00 66 00 20 00 22  00 25 00 73 00 22 00 20   ·o·f· ·"·%·s·"·
000000E0   00 6D 00 6F 00 64 00 75  00 6C 00 65 00 20 00 6E   ·m·o·d·u·l·e· ·n
000000F0   00 6F 00 74 00 20 00 66  00 6F 00 75 00 6E 00 64   ·o·t· ·f·o·u·n·d

PS C:\Users\Matt\Desktop>
```

https://github.com/matterpreter/DefenderCheck

Bypassing AV - DefenderCheck Internals

- Cool, we can use DefenderCheck…but how does it work?
  - Far simpler than you might think! - It, quite literally, runs defender against the target binary
- Step 1: segment the target binary
- Step 2: run defender against it
- Step 3: rinse and repeat

Bypassing AV - DefenderCheck Internals

- What? Don't believe me?

```
while (true)
{
    if (debug) { Console.WriteLine("Testing {0} bytes", splitarray1.Length); }
    File.WriteAllBytes(testfilepath, splitarray1);
    string detectionStatus = Scan(testfilepath).ToString();
    if (detectionStatus.Equals("ThreatFound"))
    {
        if (debug) { Console.WriteLine("Threat found. Halfsplitting again..."); }
        byte[] temparray = HalfSplitter(splitarray1, lastgood);
        Array.Resize(ref splitarray1, temparray.Length);
        Array.Copy(temparray, splitarray1, temparray.Length);
    }
    else if (detectionStatus.Equals("NoThreatFound"))
    {
        if (debug) { Console.WriteLine("No threat found. Going up 50% of current size."); };
        lastgood = splitarray1.Length;
        byte[] temparray = Overshot(originalfilecontents, splitarray1.Length); //Create temp array with 1.5x more bytes
        Array.Resize(ref splitarray1, temparray.Length);
        Buffer.BlockCopy(temparray, 0, splitarray1, 0, temparray.Length);
    }
```

Bypassing AV - DefenderCheck

```
public static ScanResult Scan(string file, bool getsig = false)
{
    if (!File.Exists(file))
    {
        return ScanResult.FileNotFound;
    }

    var process = new Process();
    var mpcmdrun = new ProcessStartInfo(@"C:\Program Files\Windows Defender\MpCmdRun.exe")
    {
        Arguments = $"-Scan -ScanType 3 -File \"{file}\" -DisableRemediation -Trace -Level 0x10",
        CreateNoWindow = true,
        ErrorDialog = false,
        UseShellExecute = false,
        RedirectStandardOutput = true,
        WindowStyle = ProcessWindowStyle.Hidden
    };

    process.StartInfo = mpcmdrun;
    process.Start();
    process.WaitForExit(30000); //Wait 30s
```

WinAPI

- Abstracts various OS operations so programmers don't have to reinvent the wheel
  - idk about you, but I don’t want to have to rewrite file IO
- Accessible via a plethora of languages
  - For the purposes of this talk, we’ll stick to C# though these languages include C, C++, Rust, Go, Nim, PowerShell, Python, etc.
- tl;dr: Windows (or at least a portion of) is basically a ton of DLLs

WinAPI - MessageBoxA

MessageBoxA function (winuser.h)

Article • 02/09/2023 • 8 minutes to read

Feedback

Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

Syntax

C++ Copy

```cpp
int MessageBoxA(
  [in, optional] HWND   hWnd,
  [in, optional] LPCSTR lpText,
  [in, optional] LPCSTR lpCaption,
  [in]           UINT   uType
);
```

fortynorth

WinAPI - ReplaceTextW

ReplaceTextW function (commdlg.h)

Article • 02/09/2023 • 2 minutes to read

Feedback

Creates a system-defined modeless dialog box that lets the user specify a string to search for and a replacement string, as well as options to control the find and replace operations.

Syntax

C++ Copy

```
HWND ReplaceTextW(
  [in, out] LPFINDREPLACEW unnamedParam1
);
```

WinAPI - ImageList_ReadEx

ImageList_ReadEx function (commctrl.h)

Article • 10/13/2021 • 2 minutes to read

Feedback

Reads an image list from a stream, and returns an IImageList interface to the image list.

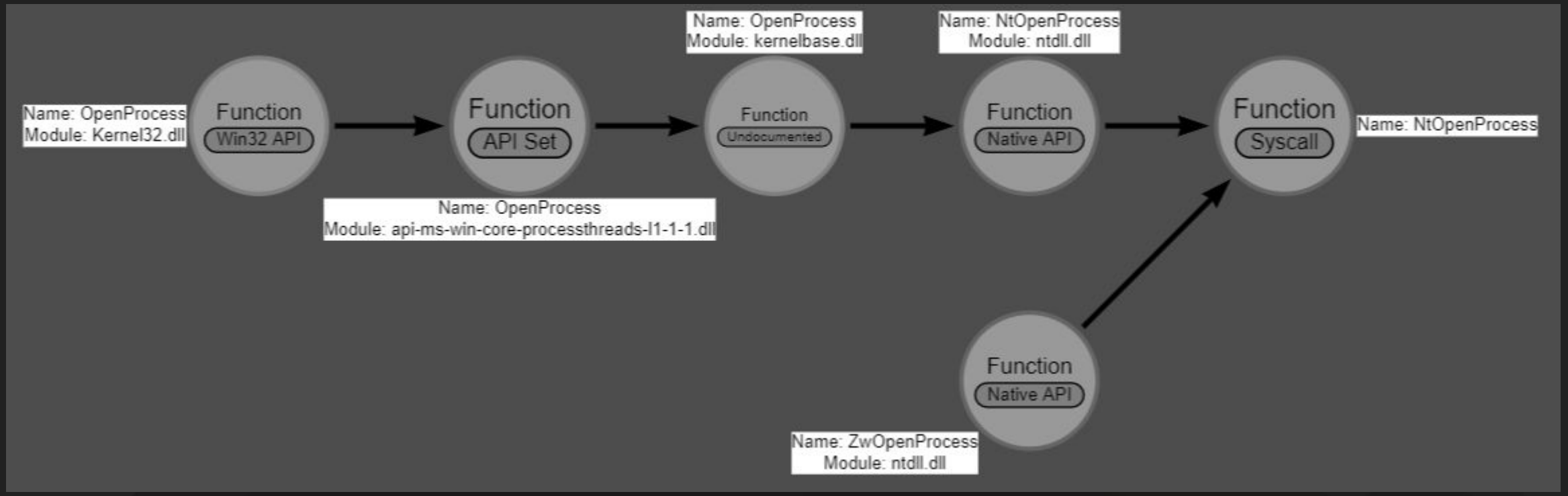
Syntax

C++ Copy

```
HRESULT ImageList_ReadEx(
  [in]  DWORD   dwFlags,
  [in]  IStream *pstm,
  [out] REFIID  riid,
  [out] PVOID   *ppv
);
```

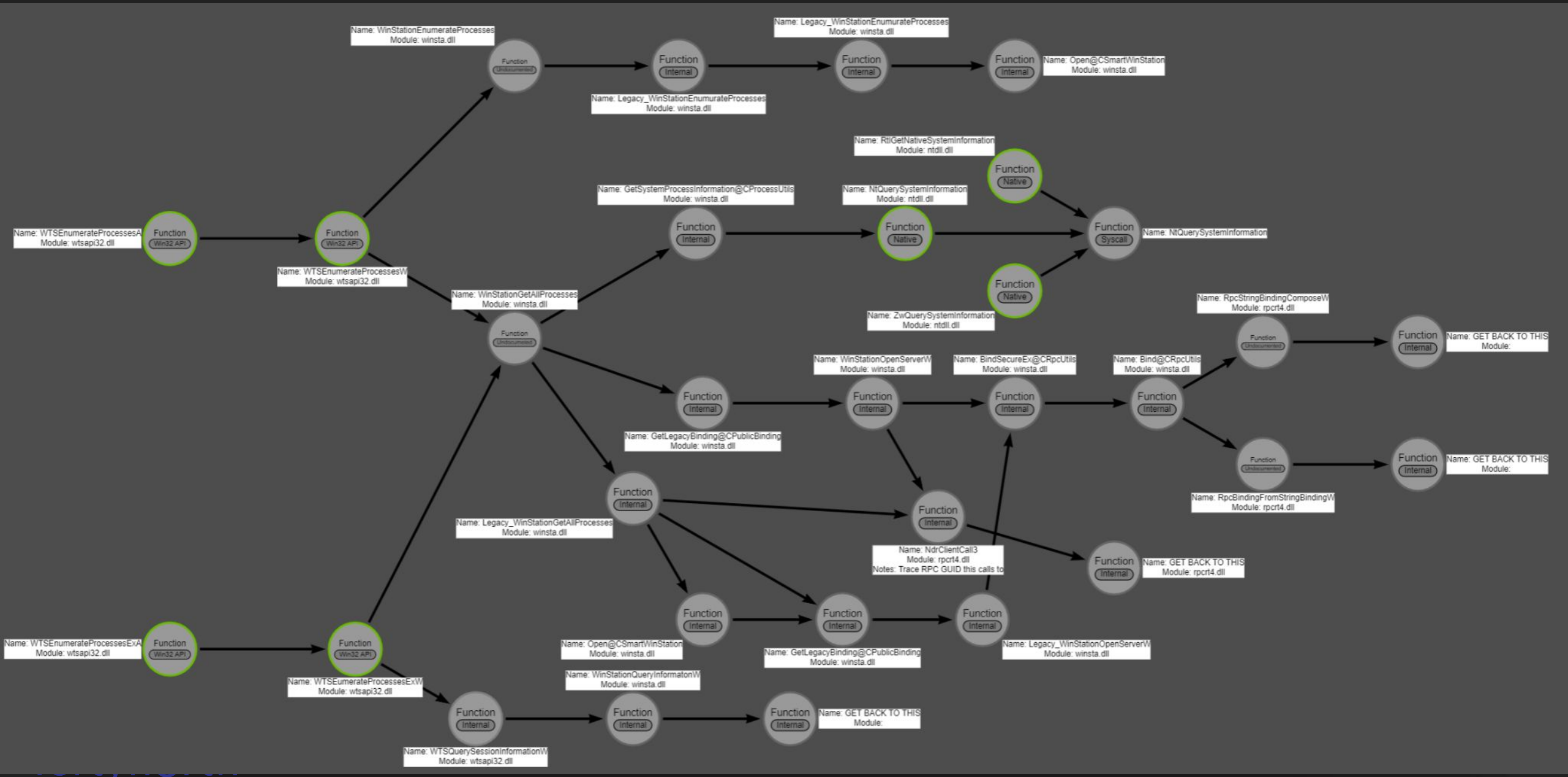
WinAPI

- Sometimes it's fairly simple/straightforward



WinAPI

- Other times….err not so much


Name: WTSEnumerateProcessesA
Module: wtsapi32.dll
Function
Win32 API
Function
Win32 API
Name: WTSEnumerateProcessesW
Module: wtsapi32.dll
Name: WinStationEnumerateProcesses
Module: winsta.dll
Function
Undocumented
Function
Internal
Name: Legacy_WinStationEnumurateProcesses
Module: winsta.dll
Name: Legacy_WinStationEnumurateProcesses
Module: winsta.dll
Function
Internal
Function
Internal
Name: Open@CSmartWinStation
Module: winsta.dll
Name: RtlGetNativeSystemInformation
Module: ntdll.dll
Function
Native
Name: GetSystemProcessInformation@CProcessUtils
Module: winsta.dll
Name: NtQuerySystemInformation
Module: ntdll.dll
Function
Internal
Function
Native
Function
Syscall
Name: NtQuerySystemInformation
Function
Native
Name: ZwQuerySystemInformation
Module: ntdll.dll
Name: WinStationGetAllProcesses
Module: winsta.dll
Function
Undocumented
Name: RpcStringBindingComposeW
Module: rpcrt4.dll
Function
Undocumented
Function
Internal
Name: GET BACK TO THIS
Module:
Name: WinStationOpenServerW
Module: winsta.dll
Name: BindSecureEx@CRpcUtils
Module: winsta.dll
Name: Bind@CRpcUtils
Module: winsta.dll
Function
Internal
Function
Internal
Function
Internal
Function
Internal
Name: GetLegacyBinding@CPublicBinding
Module: winsta.dll
Function
Undocumented
Function
Internal
Name: GET BACK TO THIS
Module:
Function
Internal
Name: RpcBindingFromStringBindingW
Module: rpcrt4.dll
Function
Internal
Name: Legacy_WinStationGetAllProcesses
Module: winsta.dll
Function
Internal
Name: NdrClientCall3
Module: rpcrt4.dll
Notes: Trace RPC GUID this calls to
Function
Internal
Name: GET BACK TO THIS
Module: rpcrt4.dll
Name: WTSEnumerateProcessesExA
Module: wtsapi32.dll
Function
Win32 API
Function
Win32 API
Name: WTSEumerateProcessesExW
Module: wtsapi32.dll
Function
Internal
Name: Open@CSmartWinStation
Module: winsta.dll
Function
Internal
Name: GetLegacyBinding@CPublicBinding
Module: winsta.dll
Function
Internal
Name: Legacy_WinStationOpenServerW
Module: winsta.dll
Name: WinStationQueryInformatonW
Module: winsta.dll
Function
Internal
Name: WTSQuerySessionInformationW
Module: wtsapi32.dll
Function
Internal
Function
Internal
Name: GET BACK TO THIS
Module:

WinAPI – Invoke The Platform

- "Platform Invoke"
- Most common way to leverage the WinAPI within C#, also the easiest to use...and detect
- Directly imports DLLs into program
  - Imports appear on IAT, more on this later

WinAPI – Ordinal Values

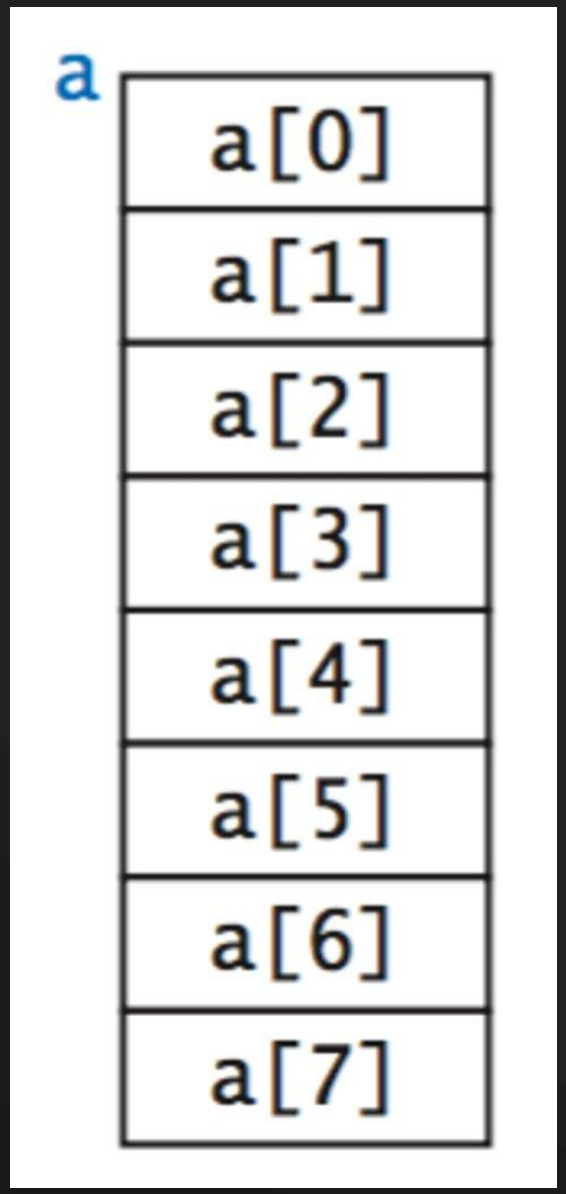
- Oldie but a goldie
  - Old school red teaming technique to reference API
- Method of referencing certain API without using the name
- These values can change from operating system/service pack
  - Meaning these will have to tailored to client machines
- https://fortynorthsecurity.com/blog/ordinal-values-and-c/

WinAPI – Ordinal Values

- Instead of referencing API by name, they are referenced by indices known as ordinal values

- Think of a list or a similar data structure, except this list points to WinAPI functions instead of values
  - Each element is accessed by its corresponding index

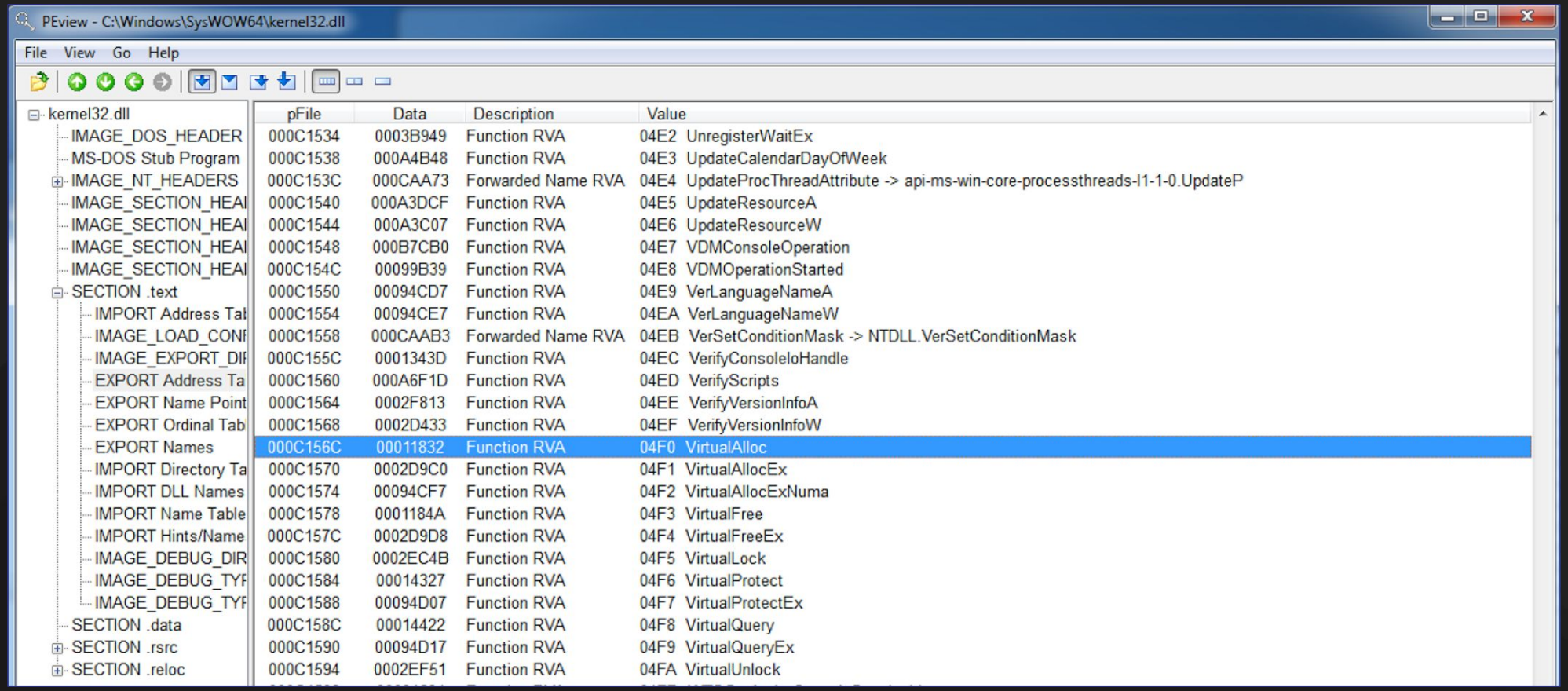
```
[DllImport("kernel32.dll", EntryPoint = "#1114", CharSet = CharSet.Unicode,
    SetLastError = true, CallingConvention = CallingConvention.StdCall)]
public static extern IntPtr ChrisQueueUserAPC(IntPtr pfnAPC, IntPtr hThread,
    IntPtr dwData);
```



fortynorth

WinAPI – Ordinal Values

- Many ways to obtain ordinal values
  - Debuggers (I.e IDA), PEView, etc.



fortynorth

WinAPI Aside – Delegates

- Delegates - "schematic" methods
  - Can be thought of as blueprints for a custom method

```
[UnmanagedFunctionPointer(CallingConvention.StdCall)]
public delegate Boolean GetFileTime(IntPtr hFile, ref FileStructs.FILETIME lpCreationTime, ref FileStructs.FILETIME lpLastAccessTime,
    ref FileStructs.FILETIME lpLastWriteTime);
```

WinAPI – Invoke The Dynamic

- "Dynamic Invoke"
- Alternative method of API invocation
  - Create function prototype and make delegate
  - "Cast" pointer as delegate
  - Use as any other method

```
[UnmanagedFunctionPointer(CallingConvention.StdCall)]
public delegate Boolean GetFileTime(IntPtr hFile, ref FileStructs.FILETIME lpCreationTime, ref FileStructs.FILETIME lpLastAccessTime,
    ref FileStructs.FILETIME lpLastWriteTime);
```

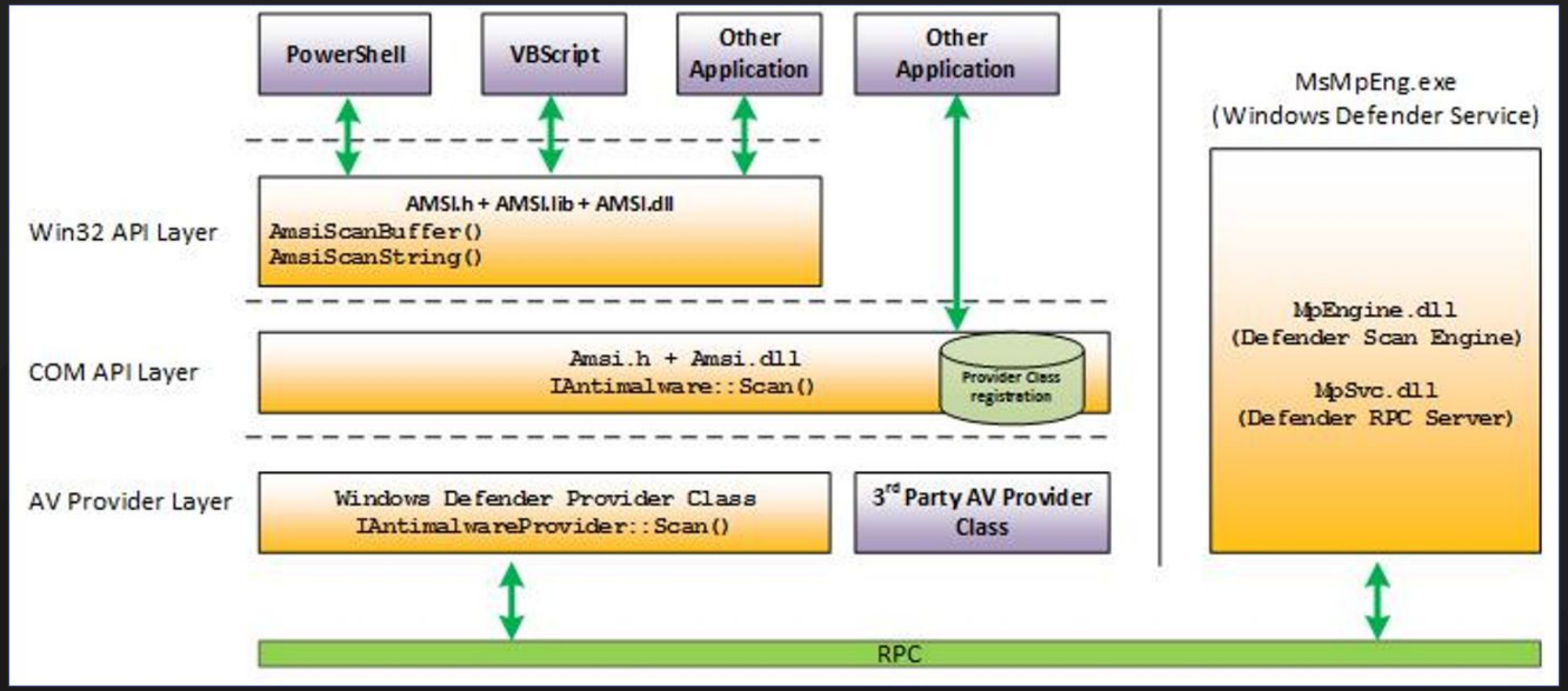
```
ptr = API.TSS.GetLibraryAddress("kernel32.dll", "SetFileTime");
var SetFileTime = GetDelegateForFunctionPointer(ptr, typeof(API.Delegates.SetFileTime)) as API.Delegates.SetFileTime;
```

AMSI

- AntiMalware Scanning Interface
- Microsoft published malware scanning driver, used by various 3rd party AV vendors
- Scans PowerShell, C#, VBA, VBScript, JScript, XLM, COM, MSI, and more

AMSI

- Amsi.dll contains various scanning functions used to scan potentially malicious contents



fortynorth

Bypassing AMSI – Forcing an Error

- Straight up break AMSI
- Can't scan anything if its broken!
- An option, but maybe not the best option

Bypassing AMSI - Patching

- Interfacing with the AMSI API to tamper with AMSI scanflow
- Usually involves modifying opcodes to return clean results on scans

Bypassing AMSI - AMSITrigger

- DefenderCheck, but for AMSI
- Returns line numbers of strings that may be flagged as malicious

```
C:\Users\rythmstick>AmsiTrigger.exe -i=c:\tools\AMSI\InvokeMimiktz.ps1 -f=2
(1)     "Invoke-Mimikatz"
(6)     "Invoke-ReflectivePEInjection"
(12)    "Invoke-Mimikatz"
(22)    "in memory using PowerShell. Can be used to dump credentials without writing anything to disk"
(44)    "Invoke-Mimikatz"
(49)    "Invoke-Mimikatz"
(54)    "Invoke-Mimikatz"
(57)    "Invoke-ReflectivePEInjection"
(58)    "Invoke-ReflectivePEInjection"
(58)    "Invoke-ReflectivePEInjection"
(535)   "Add-Member NoteProperty -Name VirtualProtect -Value $VirtualProtect"
(560)   "Add-Member -MemberType NoteProperty -Name WriteProcessMemory -Value $WriteProcessMemory"
(1008)  ".CreateRemoteThread.Invoke($ProcessHandle, [IntPtr]::Zero, [UIntPtr][UInt64]0xFFFF, $StartAddress, $ArgumentPtr,
```

fortynorth

source: https://www.rythmstick.net/posts/amsitrigger/

Bypassing AMSI – amsi.fail

- Azure function used to generate PowerShell oneliners to disable AMSI using the following techniques
  - Matt Graeber's reflection method
  - Rastamouse's AmsiScanBuffer patching method
  - Matt Graeber's Reflection method w/ WMF5 autologging bypass
  - Forcing an error – Commonly achieved by setting AmsiUtils to null or 0
- Allows for the generation of both cleartext and encoded one-liners
- Generated at runtime so no two bypasses share the same signature, bypassing signature-based detection

Bypassing AMSI – amsi.fail

AMSI.fail
~Flangvik~

AMSI GitHub

What is AMSI.fail?

AMSI.fail generates obfuscated PowerShell snippets that break or disable AMSI for the current process. The snippets are randomly selected from a small pool of techniques/variations before being obfuscated. Every snippet is obfuscated at runtime/request so that no generated output share the same signatures.

Generate

Generate Encoded

ETW

- Event Tracing for Windows
- Sysmon uses this along with many AVs/EDRs for telemetry
- There is an event for ETW tampering....kinda

Bypassing ETW - Patching

- Similar to AMSI patching
  - GetProcAddress to fetch ETWEventWrite
  - Allow writing perms
  - Replace value in ETWEventWrite
  - Set permissions back

By now you hopefully see the flaw in relying on ETW for indicators of malicious activity. Let's make some modifications to our unmanaged .NET loader by adding in the ability to patch out the **ntdll!EtwEventWrite** call.

For this example we will target x86. Let's dig out that **EtwEventWrite** function to see what we're working with. If we follow the function disassembly we find that the return is completed via a **ret 14h** opcode:

```
779f2459 33cc          xor     ecx, esp
779f245b e8501a0100    call    ntdll!__security_check_cookie (77
779f2460 8be5          mov     esp, ebp
779f2462 5d            pop     ebp
779f2463 c21400        ret     14h
```

To neuter this function we will use the same **ret 14h** opcode bytes of **c21400** and apply them to the beginning of the function:

```
// Get the EventWrite function
void *eventWrite = GetProcAddress(LoadLibraryA("ntdll"), "EtwEventWrite");

// Allow writing to page
VirtualProtect(eventWrite, 4, PAGE_EXECUTE_READWRITE, &oldProt);

// Patch with "ret 14" on x86
memcpy(eventWrite, "\xc2\x14\x00\x00", 4);

// Return memory to original protection
VirtualProtect(eventWrite, 4, oldProt, &oldOldProt);
```

fortynorth

https://blog.xpnsec.com/hiding-your-dotnet-etw/

What is EDR

- Endpoint Detection and Response
- Malware. Not really, but mostly yes

Hooking - IAT

- Import Address Table

- Lists all static imports of a binary
  - Detects things like P/Invoke as the DLLs are statically imported

Bypassing IAT Hooking

- IAT only displays statically imported API
- If only statically imported APIs are logged, don't statically import API :)
- Various languages used for offensive operations allow for the ability to dynamically fetch and execute API calls. Or use something like syscalls*.

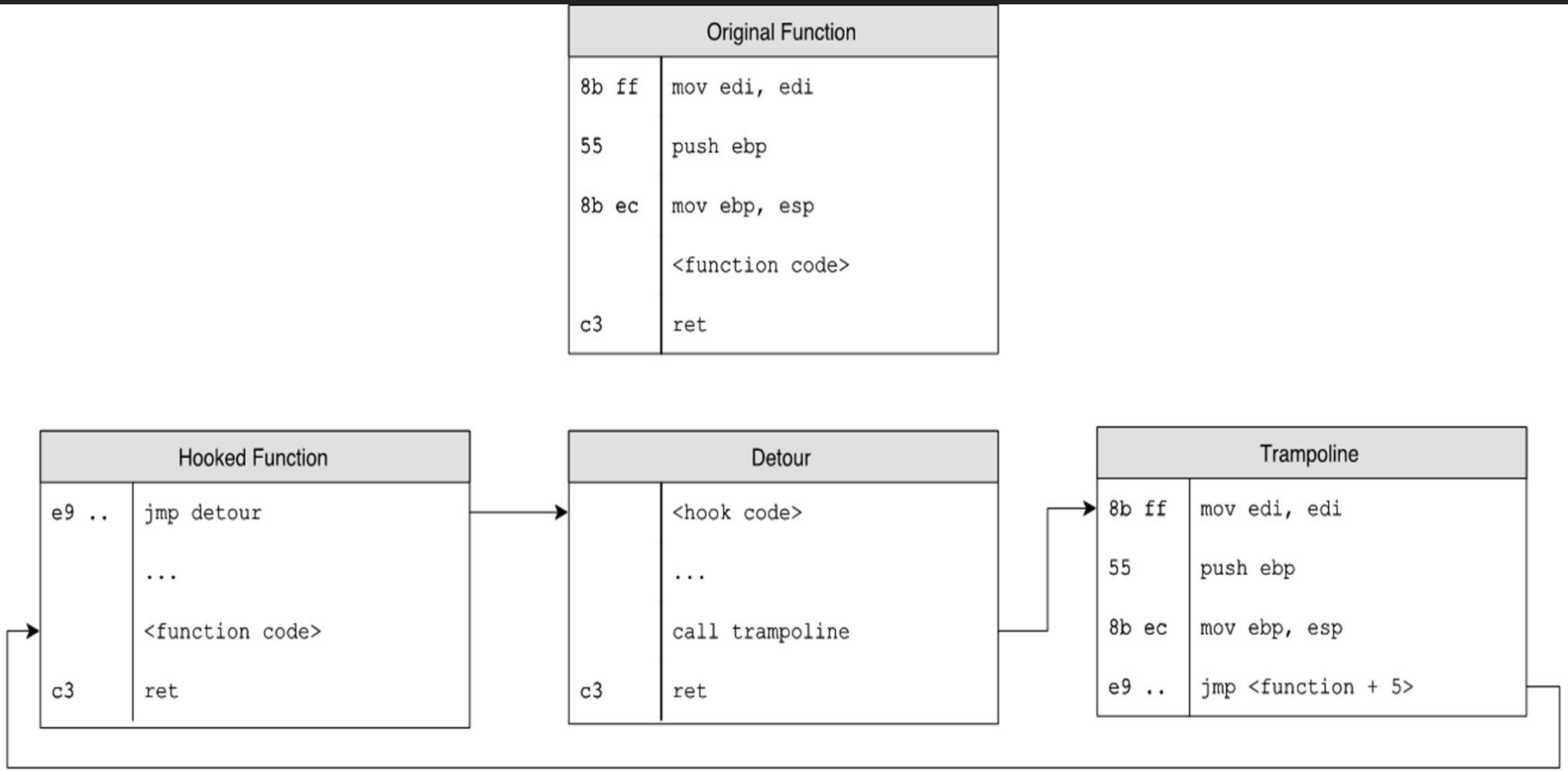
fortynorth

*syscalls are out of the scope of this presentation

Hooking – API (How EDRs Detect)

- Most modern EDR vendors perform what's called "Hooking"

- These hooks make it so that everytime one of these API are called, it's sent to the EDR for investigation



fortynorth

Source: https://blogs.blackberry.com/en/2017/02/universal-unhooking-blinding-security-software

Bypassing API Hooking

- Using lower level API that aren't hooked
- Using system calls
- HellsGate/HalosGate
  - HellsGate - dynamically fetch syscalls by reading PEB
  - HalosGate - unhooks API calls
- Loading a fresh copy of a DLL and using that!

General Tradecraft Considerations

- Living in the proper context
  - I.E using runtimebroker.exe to use execute-assembly
- Living in mem is best, but what if dropping files to disk is a must?
  - Enter timestomping
- Named pipe naming conventions
  - *cough* *cough* chrome named pipe conventions *cough*

```
------        12/31/1600   7:00 PM              1 chrome.sync.3048.13520.469780030
------        12/31/1600   7:00 PM              1 warp_service
------        12/31/1600   7:00 PM              2 crashpad_14904_OSYCSFXJYCCVYLHD
------        12/31/1600   7:00 PM              1 mojo.14904.4836.18300522837948067728
------        12/31/1600   7:00 PM              1 mojo.14904.4836.11952947583381587058
------        12/31/1600   7:00 PM              1 mojo.14904.4836.15912202897419572893
```

Named Pipes? I don't name pipe

- IPC method
  - InterProcessCommunication
- ephemeral windows mechanism used to send data between processes/applications
- Generate Sysmon Event IDs 17 - 18

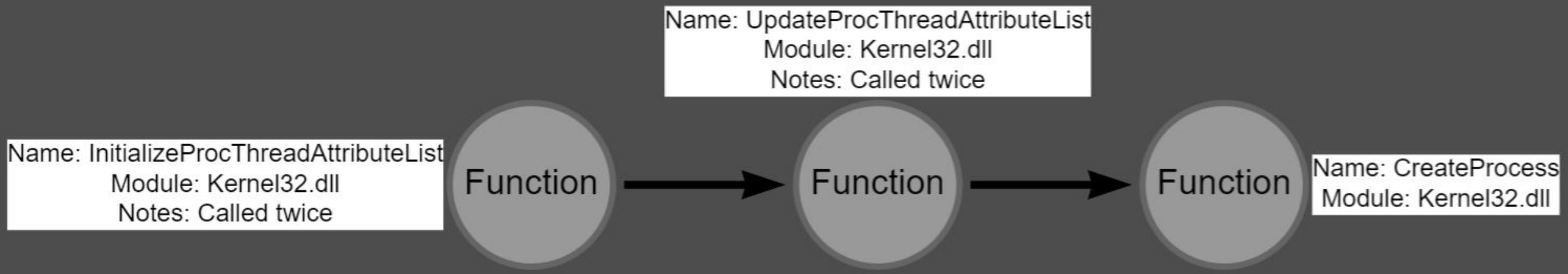
Tradecraft – Log Hunting/Manipulating

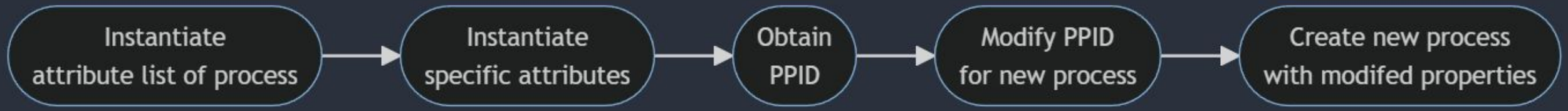
- Have read access to logs? Leverage this!
- AV/EDRs have exclude paths
  - Find these and live within them to evade these products
- Some vendors also list processed not monitored
  - Identify and inject into these processes where possible
- https://www.blackhillsinfosec.com/windows-event-logs-for-red-teams/

Tradecraft – PPID Spoofing

- PPID – Parent Process ID
- Modify the parent-child process creation to make it appear as if a process was spawned by an arbitrary process
- I.e instead of WINWORD.exe -> cmd.exe, try visualstudio.exe -> cmd.exe

Tradecraft -PPID Spoofing

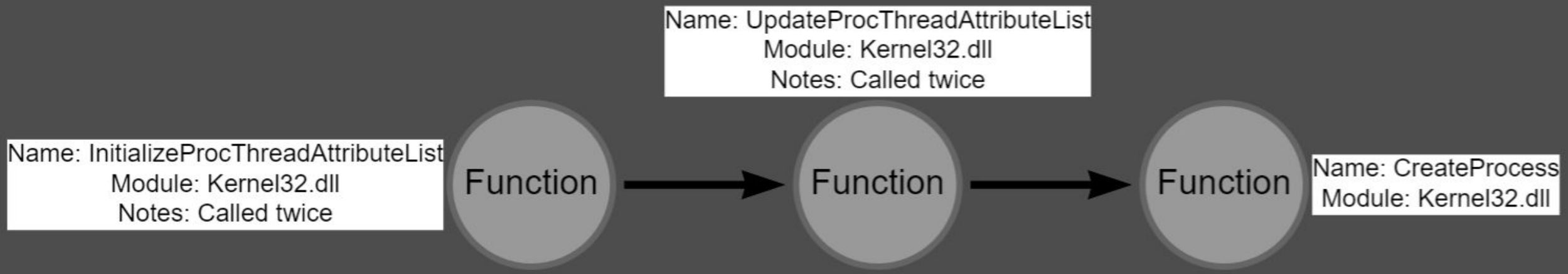




Tradecraft -BlockDLLs

- Some AV/EDR vendors will attempt to load their DLLs
  - Mainly for process investigation
- By modifying certain attributes of a process, we can make it so that only microsoft signed DLLs could be injected into our processes
- YMMV considering various EDR products are MS signed
- Looks similar to PPID spoofing, that's because it is…as far as process modification goes

Tradecraft - BlockDLLs



Tradecraft – Fork&Run vs. Inline

- Fork&Run – creates new proc, uses named pipe to retrieve output
  - "sacrificial" proc created, forensically dirty

- Inline – stays in the same process
  - Stay inline where possible
  - https://github.com/anthemtotheego/InlineExecute-Assembly
  - TEST YOUR CODE, one screw up and RIP your beacon

Pro Tip [-1]

- If all else fails, there's always the tried and true RPG-7

vx-underground
@vxunderground

We have seen many tweets recently about silly malware concepts like "syscalls", "unhooking", or "obfuscation".

Here is our #1 #RedTeamTip to avoid EDRs. Use an RPG-7 to obliterate the computer. The EDR cannot detect your malware if the computer is not operational

Socials/QnA

- Twitter: https://twitter.com/Gr1mmie
- Mastodon: https://infosec.exchange/@Gr1mmie
- Blog: https://grimmie.net/
- Github: https://github.com/Gr1mmie