CERT-EU

FOR THE EU INSTITUTIONS, BODIES AND AGENCIES

CERT-EU Security Whitepaper 2019-001

PowerShell – Cybersecurity Perspective

Ciprian-Nicolae BOLDEA, Krzysztof SOCHA

ver. **1.0**
July 19, 2019

TLP: WHITE

Contents

1 Summary

In the last years we have seen an increasing use of PowerShell for malicious purposes. This was mainly caused by its powerfulness and lack of means to counter this kind of usage. On the other hand PowerShell also evolved, providing currently also more means for defenders.

The aim of this document is to present PowerShell from a cybersecurity perspective. Described are also controls that can be implemented in the prevention and detection of cyberattacks using PowerShell.

2 Background

PowerShell is an automation platform and a scripting language for Windows, which aims at simplifying the system management. Consisting of a command-line shell with associated scripting language and built on the .NET Framework. PowerShell provides rich objects and a massive set of built-in functionality. PowerShell provides full access to system functions like Windows Management Instrumentation (WMI) and Component Object Model (COM) objects. It is a powerful tool used for Windows environments control [1, 2].

Initially a Windows component only, PowerShell (PS) was made open-source and cross-platform on the 18th of August 2016 [3].

Being a powerful tool, installed by default on most Windows computers, it has begun to be used more and more often in the attacks. Other contributing factors:

- lack of proper logging in the initial versions,
- lack of monitoring of PS events in many organizations,
- use of obfuscation that makes static signatures ineffective,
- ability to do code injection without dropping malicious code to disk,
- remote access capabilities,
- since PowerShell is used in Windows administration, attacks might be misperceived as regular tasks.

Malicious PowerShell scripts are predominantly used as downloaders (such as Office macros) during the incursion phase. The second most common use is during the lateral movement phase, allowing a threat actor to execute code on a remote computer when spreading inside the network. PowerShell can also download and execute commands directly in memory, making it hard for forensics experts to trace the infection [4].

3 Important Features for Cybersecurity

3.1 Execution Policies

PowerShell execution policies allow to determine the conditions under which PowerShell loads configuration files and runs scripts.

Execution policy can be set for the local computer, for the current user or even for a particular session. Group Policy setting can be used to set execution policy for computers and users. Execution policies for the local computer and current user are stored in the registry. The execution policy for a particular session is stored only in memory and is lost when the session is closed.The execution policy is not a security system that restricts user actions. For example, users can easily

circumvent a policy by typing the script contents at the command line when they cannot run a script. Instead, the execution policy helps users to set basic rules and prevents them from violating them unintentionally [18].

3.2 PowerShell Versions

There are 5 major versions of PowerShell. More details about PowerShell versions installed by default on each version of Windows can be found in Table 1 in [4]. The most popular versions encountered are however versions 2 and 5.

Version 2 was the first very popular version of PowerShell. It is installed by default on Windows 7 and Windows 2008 R2, but it is also present in newer versions of Windows for compatibility reasons. Because of the poor logging, there are some attacks targeting the usage of the PowerShell Version 2. In Windows 10 the integrated Version 2 can be disabled in optional features [5].

Currently, the latest PowerShell version is 5. This version adds additional capabilities with the constrained language, the logging, and brings also some improvements regarding Just Enough Administration [8].

Note: there is also PowerShell core version. By contrast to regular PowerShell, this is multiplaform, but limited in functionalities. The PowerShell Core version 6 was launched in August 2018 [17].

3.3 PowerShell Language Modes

PowerShell can be used in different language modes. In `FullLanguageMode` an attacker can load at his will COM objects/libraries/classes into PowerShell session. `RestrictedLanguageMode` and `ConstrainedLanguageMode` (starting with version 3) can be used to limit the language elements that are permitted in the session [13][1] `ConstrainedLanguageMode` cuts this ability down massively and breaks nearly all attacking frameworks. PowerShell version 5 can be used to enforce this either with Applocker in *Allow* mode or with DeviceGuard using UMCI (User Mode Code Integrity). At the same time, allowed scripts will still keep working using the `FullLanguageMode` [5].

3.4 Signing

Signing allows setting a trust level in the scripts. The signing certificates can be created by internal PKI environment and give even the developers security to not accidentally modify and execute scripts. In combination with DeviceGuard or Applocker this can be used as a prerequisite for executing scripts. Only signed scripts would be then executed in the `FullLanguageMode`.

3.5 Securing Privileged Access

Just Enough Administration (JEA) feature in PowerShell 5 allows specific users to perform specific administrative tasks on servers without giving them administrator rights [10]. JEA is

[1] Alternative would be configuration of ExecutionPolicy and AppLocker/DeviceGuard to limit the `FullLanguageMode` [5].

based on Windows PowerShell-constrained run spaces, a technology used at Microsoft to help secure administrative tasks [14].

3.6 PowerShell Logging

PowerShell logging evolved in successive versions.

- In version 2, through Transcription, it has the ability to record the content of a PowerShell session.
- Module Logging introduced in version 3 capture execution details.
- With Deep Script Block Logging in version 5 logging is done at the base level of executable code in PowerShell. It represents a command typed interactively in the PowerShell console, supplied through the command line, wrapped in a function or script. By default Deep Script Block Logging is recording any suspicious PS scripts.

A more comprehensive description of PowerShell logging can be found in [11].

3.7 Transcription

In the first PowerShell versions Transcription was difficult to setup. It was supported only in the interactive PowerShell console. It is improved in version 5. Now it allows the logging of PowerShell commands and the output of those commands also from scripts and remote sessions[15].

3.8 Ways of Running PowerShell

PowerShell can be run in various ways:

- from PowerShell console – the most common one,
- from within a C#/.NET applications [21],
- from command-line,
- from macros in MS Office documents,
- from various scripts including VBScripts [22].

3.9 PowerShell for Remote Administration

Being encrypted, requiring admin rights, but verbose in logging, Microsoft recommends usage of newer PowerShell versions for remote administration [5, 9].

4 Prevention and Detection of PowerShell Attacks

4.1 Execution Policies Bypass

By default, Microsoft restricts PowerShell scripts with execution policies. These were not designed as a security feature, but rather to prevent users from accidentally executing scripts. In fact, there are several ways to bypass the execution policy [19]. All internal legitimately used PowerShell scripts should be signed and all unsigned scripts should be blocked through the execution policy. While there are simple ways to bypass the execution policy, enabling it makes

infection more difficult. The security team should be able to monitor for any attempt to bypass the execution policy and follow up on it. Details about monitoring can be found in [20].

4.2 PowerShell Downgrade Attacks

With the advent of PowerShell v5 with new security features, old versions of PowerShell become more attractive for attackers.

There are two ways to perform a PowerShell downgrade attack:

1. **Command-Line Version Parameter** `PowerShell -Version 2 -Command <...>` (or any of the `-Version` abbreviations). `PowerShell.exe` itself is just a simple native application that hosts the CLR, and the `-Version` switch tells PowerShell which version of the PowerShell assemblies to load.
2. **Applications Compiled using V2 Reference Assemblies** When an application is compiled in C# to leverage the PowerShell engine, it might link against the PowerShell v2 reference assemblies during development. Windows will use the PowerShell v2 engine (if available), when the application runs. Otherwise, PowerShell's type forwarding will run the application using the currently installed PowerShell engine.

Detection for this specific attack can be done with Event Log. `Windows PowerShell event ID 400`. This event includes the field `Engine Version`.

Prevention for downgrade attacks is to be done with with AppLocker / file auditing. When CLR loads PowerShell assemblies, it will first load the managed assemblies from the GAC (if they are available). It will also load the native images that contain pre-jitted code if the assemblies are NGEN'd (which they are). These can either be audited or blocked directly.

With AppLocker or Device Guard the most robust solution is to block earlier versions of the PowerShell engines by version. Both native images and MSIL assemblies should be blocked.

More details on the PowerShell Downgrade Attacks can be found in [5].

4.3 Migration to Windows 10

Alongside with default PowerShell 5 installation, Windows 10 comes along with many security features:

- **Anti-Malware Scan Interface** (AMSI) is an interface which can be used by every antivirus scanner to evaluate if a script is potentially harmful. AMSI enables all of the scripting engines (PowerShell, VBScript, and JScript) to request analysis of dynamic content, from a script file, typed commands at the command line, and even code downloaded and executed in memory. When code is delivered to the PowerShell *engine* (`System.Management.Automation.dll`), it is sent to the AMSI for anti-malware checks. Windows Defender supports by default AMSI on Windows 10 [5].
- **Device Guard** is a combination of enterprise-related hardware and software security features that, when configured together, will lock a device down so that it can only run trusted applications that you define in your code integrity policies.
- **Credential Guard**. The hashes are stored in a separate container on which the attacker cannot read on.
- **Protected Event Logging** allows participating applications (PowerShell) encrypt sensitive data as they write it to the event log. For example PowerShell scripts may contain credentials or other sensitive data [15].

4.4 Advanced Threat Protection

Windows Defender Advanced Threat Protection (ATP) is a (paid) post-breach detection, investigation, and response tool from Microsoft. ATP combines sensors built-in to the operating system with a powerful security cloud service enabling Security Operations to detect, investigate, contain, and respond to advanced attacks against their network.

4.5 Advanced Threat Analytics

Advanced Threat Analytics (ATA) help protect organization from advanced attacks. The key methods to detect attacks are:

- Behavioral Analytics – Learning the normal patterns of users and the devices they use. Patterns outside the normal will be flagged such as using different devices or working different/longer hours.
- Detection for known malicious attacks and security issues – Known attacks such as pass-the-ticket, pass-the-hash, brute force and so on.

4.6 Detection of attacks

For the detection of attacks involving Powershell, it is recommend to establish a baseline of normal activity in an environment. Deviations from the baseline may serve as an indication of attacker activity. It is recommend that organizations formulate a PowerShell monitoring strategy by first assessing and enumerating [23]:

- **Which servers/server groups are administered via PowerShell remoting. Are there local PowerShell script executed. The same for workstations/end-user systems.**
- **Which domain accounts use PowerShell remoting. What are the source hostnames from which these users would administer systems.**
- **What are the names and common directories used for legitimate PowerShell scripts within the environment. Are the legitimate scripts used by the organization digitally signed.**
- **Are any systems configured to automatically load and execute PowerShell scripts for maintenance or administration purposes.**

Once complete, administrators can rely on centralized Windows event log forwarding and collection (for at-scale monitoring) or local event log analysis (for targeted forensics and investigations) to identify signs of anomalous PowerShell usage. This effort will require filtering and tuning – that's where having a baseline, or even monitoring a subset of known-good systems with common configuration for a period of time, can help.

Instruction on how to enable and configure Module, Script Block, and Transcription logging in Windows PowerShell can be found in [24].

In Annex B are presented examples of log sources and events to be monitored.

5 Conclusions

PowerShell is a tool that evolved, and by using the latest version with the proper configuration (logging, monitoring, disable other versions) many attack paths would be blocked.

Migration to recent Windows version and implementing other recommended security measures will also improve security posture related to PowerShell.

6 Annex A – Definitions

AMSI - Anti-Malware Scan Interface

CLR - Common Language Runtime is the virtual machine component of Microsoft's .NET framework that manages the execution of .NET programs

GAC - Global Assembly Cache is a machine wide repository for .Net Assemblies. PowerShell GAC provides several PowerShell commands to view and modify the GAC. PowerShell GAC works standalone and does not depend on tools like `gacutils.exe`. PowerShell GAC uses the documented native GAC API, so it does not depend on any GAC internals like changing folder structures.

`Ngen.exe` - Native Image Generator is a tool that improves the performance of managed applications. `Ngen.exe` creates native images, which are files containing compiled processor-specific machine code, and installs them into the native image cache on the local computer. The runtime can use native images from the cache instead of using the JIT compiler to compile the original assembly.

JEA - Just Enough Administration

JIT - just-in-time

MSIL - Microsoft Intermediate Language is a language used as the output of a number of compilers

PS - PowerShell

7 Annex B – Log Sources and Events for the Detection of PowerShell Attacks

7.1 Relevant Windows Events

7.1.1 Windows PowerShell event log entries indicating the start and stop of PowerShell activity

- **Event ID 400:** "Engine state is changed from None to Available", upon the start of any local or remote PowerShell activity.
- **Event ID 600:** referencing "WSMan" (e.g. "Provider WSMan Is Started"), indicating the onset of PowerShell remoting activity on both source and destination systems.
- **Event ID 403:** "Engine state is changed from Available to Stopped", upon the end of the PowerShell activity.
- **Event ID 40961:** "PowerShell console is starting up"
- **Event ID 4100:** "Error Message = File (path to)test.ps1 cannot be loaded. . . "

7.1.2 System event log entries indicating a configuration change to the Windows Remote Management service

- **Event ID 7040:** "The start type of the Windows Remote Management (WS-Management) service was changed from [disabled / demand start] to auto start." – recorded when PowerShell remoting is enabled.
- **Event ID 10148:** "The WinRM service is listening for WS-Management requests" – recorded upon reboot on systems where remoting has been enabled.

7.1.3 WinRM Operational event log entries indicating authentication prior to PowerShell remoting on an accessed system

- **Event ID 169:** "User [DOMAIN\Account] authenticated successfully using [authentication_protocol]"

7.1.4 Security event log entries indicating the execution of the PowerShell console or interpreter

- **Event ID 4688:** "A new process has been created" – includes account name, domain, and executable name in the event message.

7.1.5 AppLocker event log entries recording the local execution of PowerShell scripts

It is recommended to enable AppLocker in audit mode across an environment for this specific purpose. Upon script execution in audit mode, the AppLocker MSI and Script Event Log may record: - **Event ID 8006:** "[script_path] was allowed to run but would have been prevented from running if the AppLocker policy were enforced" - **Event ID 8005:** "[script_path] was allowed to run"

7.2 Splunk Queries for Sysmon Events Examples [25]

7.2.1 Query 1 – Powershell Started from Suspicious Processes

Objective: Monitor browser and MS Office processes for cmd or PowerShell as child process on workstations. Other suspicious executables like `cscript.exe`, `wscript.exe`, `rundll32.exe`, etc., might be added to the rule below.

Query:

```
sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational" eventcode=1
  parentimage="C:\\Program Files (x86)\\Microsoft Office\\Office14\\WINWORD.EXE" OR
  parentimage="C:\\Program Files (x86)\\Microsoft Office\\Office14\\OUTLOOK.EXE" OR
  parentimage="C:\\Program Files (x86)\\Microsoft Office\\Office14\\EXCEL.EXE" OR
  parentimage="C:\\Program Files (x86)\\Microsoft Office\\Office14\\POWERPNT.EXE" OR
  parentimage="C:\\Program Files (x86)\\Internet Explorer\\iexplore.exe" OR
  parentimage="C:\\Program Files\\Internet Explorer\\iexplore.exe" OR
  parentimage="\"C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe\"" OR
  parentimage="C:\\Program Files (x86)\\Google\\Chrome\\Application\\chrome.exe" AND
```

```
  (image="C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe" OR
  image="C:\\Windows\\System32\\cmd.exe" OR
  image="C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe")
```

Comments: It is very common in modern malware to exploit a vulnerability in Office macros or Flash player and start a command prompt. Microsoft discovered a 0-day based in Flash based on this behavior[2]. Next Generation Endpoint Protection Products are using this behavior. For example Carbon Black detects Locky[3]

7.2.2 Query 2 – Detecting Long PowerShell Command

Objective: Detect long PowerShell commands that most probably will include obfuscated malicious code by length of command.

Query:

```
sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational" eventcode=1
  image="C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe" OR
  image="C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe" | eval
  c_length=len(commandline) | where c_length>1000
```

7.2.3 Query 3 – Search for Suspicious Strings

Objective: This query searches for the presence of Invoke-Expression or IEX or Download strings

Query:

```
sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational" eventcode=1   powershell.exe
   Invoke* OR IEX   OR Download* | table _time, computername, processid, image,commandline,
  parentprocessid,parentimage,parentcommandline
```

Comments: Since Powershell is more than `powershell.exe` [13][4] even if `Powershell.exe` is blacklisted (not easy since it is normally used for administrative tasks) a more detailed analysis for PowerShell logs is needed. Good references for configuring PowerShell logging from FireEye [14][5]and for analyzing PowerShell logs from Carbon Black[15] and from Australian Department of Defence[6]

8 Annex C – References

[1] https://msdn.microsoft.com/en-us/PowerShell/mt173057.aspx

[2] https://en.wikipedia.org/wiki/PowerShell

[2] https://goo.gl/lvwKMw
[3] https://goo.gl/LRwHPf
[4] http://www.adsecurity.org/?p=2604
[5] https://goo.gl/zIEV5i
[6] http://goo.gl/RrJGbX

[3] https://arstechnica.com/information-technology/2016/08/PowerShell-is-microsofts-latest-open-source-release-coming-to-linux-os-x/

[4] https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-PowerShell-in-attacks-16-en.pdf

[5] https://blogs.msdn.microsoft.com/daviddasneves/2017/05/25/PowerShell-security-at-enterprise-customers/

[6] https://www.fireeye.com/content/dam/fireeye-www/global/en/solutions/pdfs/wp-lazanciyan-investigating-PowerShell-attacks.pdf

[7] http://www.leeholmes.com/blog/2017/03/17/detecting-and-preventing-PowerShell-downgrade-attacks/

[8] https://msdn.microsoft.com/en-us/library/dn896648.aspx

[9] https://blogs.technet.microsoft.com/ashleymcglone/2016/06/29/whos-afraid-of-PowerShell-security/

[10] https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/securing-privileged-access

[11] https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html

[12] https://docs.microsoft.com/en-us/windows/device-security/device-guard/device-guard-deployment-guide

[13] https://docs.microsoft.com/en-gb/powershell/module/microsoft.powershell.core/about/about_language_modes?view=powershell-6

[14] https://msdn.microsoft.com/en-us/library/dn896648.aspx

[15] https://blogs.msdn.microsoft.com/PowerShell/2015/06/09/PowerShell-the-blue-team/

[16] https://www.microsoft.com/en-us/windowsforbusiness/windows-atp

[17] https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-core-60?view=powershell-6

[18] https://docs.microsoft.com/en-gb/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-6&viewFallbackFrom=powershell-Microsoft.PowerShell.Core

[19] https://blog.netspi.com/15-ways-to-bypass-the-powershell-execution-policy/

[20] https://static1.squarespace.com/static/552092d5e4b0661088167e5c/t/5760096ecf80a129e0b17634/1465911664070/Windows+PowerShell+Logging+Cheat+Sheet+ver+June+2016+v2.pdf

[21] https://blogs.msdn.microsoft.com/kebab/2014/04/28/executing-powershell-scripts-from-c/

[22] https://blogs.technet.microsoft.com/heyscriptingguy/2012/07/18/how-to-use-vbscript-to-run-a-powershell-script/

[23] https://www.powershellmagazine.com/2014/07/16/investigating-powershell-attacks/

[24] https://www.rootusers.com/enable-and-configure-module-script-block-and-transcription-logging-in-windows-powershell/

[25] https://securitylogs.org/2016/05/07/sysmon-logs-at-scale/