A Heuristic Approach to Signature Based Detection of PowerShell Obfuscation

Johnny Lu, Ryan Smith
Jet Propulsion Laboratory, Pasadena, CA, USA
California Institute of Technology
Corresponding Authors: johnny.lu@jpl.nasa.gov, ryan.smith@jpl.nasa.gov

Abstract*—The increased availability of Powershell on targeted systems has spurred the development of sophisticated toolkits and methodologies to facilitate attacks on Windows devices. Powershell provides a suitable environment for deploying malicious payloads and evading detection through a series of in-memory attacks without ever writing to disk. An entry point within the domain gives adversaries potential access to other hosts in the network to spread malware, especially throughout an enterprise environment. Our approach focuses on the proactive monitoring of Windows event logs forwarded to a centralized server to detect malicious Powershell usage and obfuscated commands in real time. By flagging certain signatures in the process command line, we demonstrate a detailed detection of malicious Powershell activity.

I. Introduction

Powershell is a versatile configuration management framework designed to automate system tasks as well as managing system processes in the Windows operating system. Powershell is built on the .NET framework and provides access to the Win32 API including core and system DLLs [1]. This presents an entry point for attackers to utilize Powershell as a standalone environment for deploying malicious payloads. The deployment of such a powerful tool also makes it ideal for cybercriminals to leverage Powershell as a template for an attack.

The usage of Powershell simplifies application and process management using lightweight scripts known as cmdlets [2]. By default, Powershell is installed in all versions of Windows 7, Windows Server 2008 and higher. This is one of the main reasons cybercriminals leverage Powershell as an attack framework. The minimal logs generated by default makes it difficult to detect under forensic analysis. System administrators generally trust the framework, often allowing malicious Powershell activity to pass through as regular traffic. This gives attackers the potential for misuse by utilizing Powershell in an attack.

II. Attack Vectors

Traditional security tools including network firewalls and antivirus software can be setup to prevent the execution of malicious scripts, but there are ways to evade traditional methods of detection. The Powershell library includes a function to encode commands before execution. This can be done by utilizing the post exploitation framework Empire to deploy payloads to target systems after initial network intrusion. Empire allows an attacker run executables or scripts completely in memory leaving few traces of activity on the target system [3]. Overall, Empire consists of a variety of modules that can be deployed on a target machine depending on the intentions.

Empire deploys an attack using a listener, agent, and stager. The listener is a session handler to control a compromised agent that reports back to the command and control server listening at a specified IP and port. The stager is the payload to deploy on a target machine giving the attacker full control of the target device.

In order for an attacker to execute Powershell commands on a target machine, the system must have the initial layer of security compromised. The attacker can utilize Powershell to execute an obfuscated launcher on the machine using the encoded command parameter to gain full control of the device using a reverse shell. Figure 1 depicts the launcher to be executed on a machine that will report to a remote server.

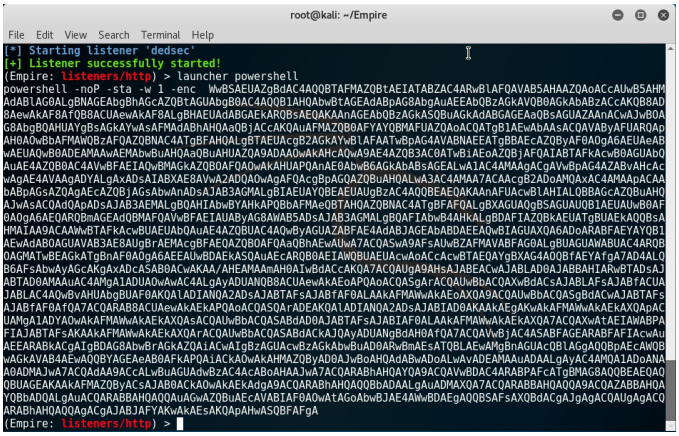

Fig 1. Command encoded in Base64

The encoded payload in its raw context is difficult for a user to interpret due to the nature of its obscurity. Specifically, the encoded command makes a request to a session handler on a specified port and executes commands that are sent back to the machine. This is performed using a listener, agent, and stager allowing an attacker to interact with the target machine.

Powershell version 5.0 includes enhanced logging visibility with the addition of module logging, script block logging, and transcription. This allows for the recording of portions of scripts, some de-obfuscated code, blocks of code as they are being executed, and keeps a record of user, session, timestamp, and metadata for each command [4]. Figure 2 is shown to represents Figure 1 in its de-encoded form. The underlying components of the command can be understood from specific keywords in the string. Specifically, the underlying functions executed has the target machine open a

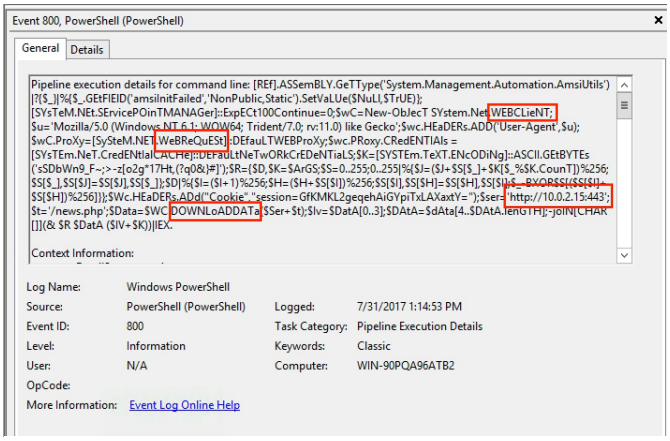

Fig 2. Decoded Base64 command

web client to make request to the listed IP address 10.0.2.15 at port 443 and downloads the data it receives to be executed on the machine. Ultimately, this presents a difficult process to detect malicious commands running without first reversing the encoded command.

Variations of functions can be used to facilitate common attacks using Powershell as a template. For instance, there are more than 100,000 possible variations of the EncodedCommand parameter alone [5]. Many of these tools used for obfuscation are freely available on the web as public tools.

Another obfuscation technique involves a combination of splitting a string, reversing or encrypting the pieces, adding escape characters, and merging the parts together by concatenation or using the format operator. The command in

USING SPECIAL CHARACTERS

When used within quotation marks, the escape character indicates a special character that provides instructions to the command parser.

The following special characters are recognized by PowerShell:

Escape Sequence	Special Character
`0	Null
`a	Alert
`b	Backspace
`f	Form feed
`n	New line
`r	Carriage return
`t	Horizontal tab
`v	Vertical tab

Fig 3. Recognized Powershell escape characters [8]

Figure 4 has the system open a webclient, download a string and execute the string as a command. One of them being Invoke-Obfuscation, a Powershell command and script obfuscator developed by security researcher Daniel Bohannon at Mandiant. The tool performs string-level obfuscations using various encoding and encrypting methods including ASCII, hexadecimal, octal, binary, and SecureString [6]. Moreover, the purposes of the tool aids other researchers by simulating obfuscated commands in order to test detection capabilities within the network.

Invoke-Obfuscation presents a magnitude of possible randomly generated obfuscations. These string-generated commands can also be encapsulated under layers of obfuscation, making it difficult to reverse. One method is by using escape characters which are ignored by the command line if not recognized when parsed by Windows Powershell. Figure 3 shows a list of the recognized special escaped characters. This type of obfuscation has been seen in the wild for instance, by re-writing the argument /i:http to /i:^h^t^t^p to break signature-based detection [7].

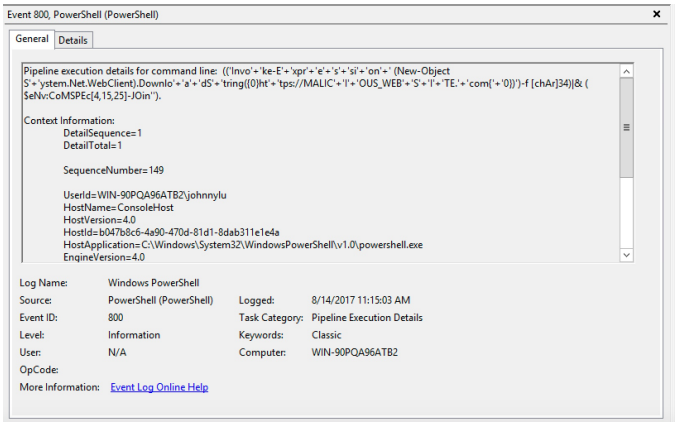

Fig. 4 Obfuscated command invocation

The instance in Figure 4 shows a basic use case of executing obfuscated commands on a remote system. Moreover, commands can be obfuscated into layers, making it much more difficult to reverse engineer or understand the code and its objectives in an attack. Figure 5 shows the command in

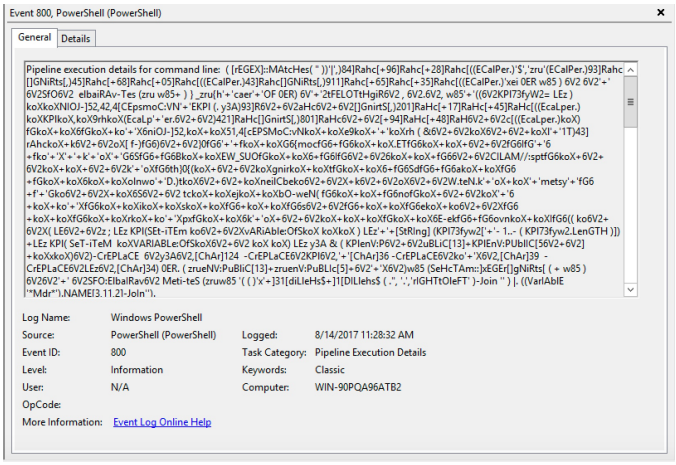

Fig. 5 Command under 3 layers of obfuscation

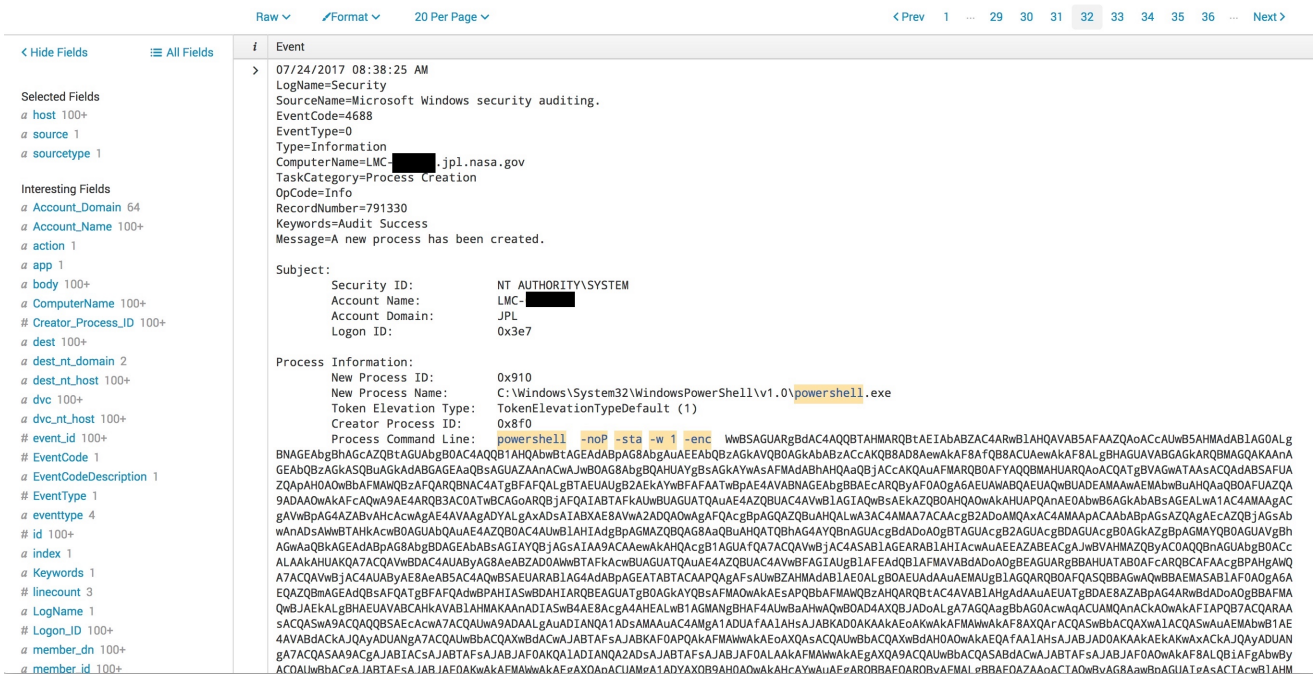

Fig. 6 Detected Powershell Event Log

Figure 4 under three layers of obfuscation that achieves the same end result. Based on the details of the command pipeline, it is difficult to determine the goal of what the command is trying to accomplish. Obfuscation is almost never used legitimately by typical users. Nonetheless, there is definitely potential for an attacker to obfuscate in order to hide their intentions.

III. Methods of Detection

In this section, we focus on detecting malicious Powershell activity at the source when executed in the command line. This is mainly because once Powershell is invoked, an attacker leaves few traces of activity during an intrusion by running scripts entirely in memory [9]. To detect malicious Powershell at its source, we query specific arguments in the command and flag certain keywords that seem suspicious such as encoded commands. Improving the detection of such arguments can be done by actively monitoring log data and flagging indicators of obfuscation. Some keywords we were monitoring for included: -Enc, -sta, -NoP, -NoL, -Win Hidden, -NonI, -EP Bypass, reverse, split, replace, concat, and the -f format operator. Even so, in the previous section we found multiple ways of re-writing a command to achieve the same result.

In order to scale with the processing of log data and events, we used Splunk ES, a SIEM (Security Information and Event Management) to manage data aggregation in a centralized server. The server processes the raw machine generated data into human readable results and analyzes the data to deliver real time indexing, graphs, reports, alerts, and dashboards [10]. This provides us with the capabilities to detect common patterns or anomalies in the data.

The Splunk application Search and Reporting allows us to filter specifically Windows logs using the index wineventlog and the event code 4688 signifying the creation of a new process. We narrowed down the events to commands invoking powershell.exe and the argument "–enc" including variants such as "-EncodedCommand," "-EC," and "-en." With alerts in place, we executed the base64 encoded payload and we were able to detect its usage on the network. Referring back to

2017-08-14 14:13:57	ryansmit	[redacted]	DOMAIN ADMIN QUERY	success	C:\Windows\System32\net.exe	C:\Windows\System32\net1.exe	C:\WINDOWS\syst
2017-08-14 14:13:57	ryansmit		DOMAIN ADMIN QUERY	success	C:\Windows\System32\cmd.exe	C:\Windows\System32\net.exe	net group "Domain
2017-08-14 13:30:48			PSHELL-ENC	success		C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	powershell -noP -s
2017-08-14 13:30:32	ryansmit		PSEXEC	success	C:\Windows\System32\cmd.exe	C:\Users\ryansmit\Desktop\PSTools\PsExec.exe	psexec \\128.149.
2017-08-14 13:25:31	ryansmit		PSEXEC	success	C:\Windows\System32\cmd.exe	C:\Users\ryansmit\Desktop\PSTools\PsExec.exe	psexec \\128.149.
2017-08-14 13:25:26	ryansmit		PSEXEC	success	C:\Windows\System32\cmd.exe	C:\Users\ryansmit\Desktop\PSTools\PsExec.exe	psexec \\128.149.
2017-08-14 13:23:54	ryansmit		PSEXEC	success	C:\Windows\System32\cmd.exe	C:\Users\ryansmit\Desktop\PSTools\PsExec.exe	psexec \\128.149.
2017-08-14 12:47:50			PSHELL-ENC	success		C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	powershell -noP -s
2017-08-14 12:47:33	ryansmit		PSEXEC	success	C:\Windows\System32\cmd.exe	C:\Users\ryansmit\Desktop\PSTools\PsExec.exe	psexec \\128.149.
2017-08-14 12:43:11			PSHELL-ENC	success		C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	powershell -noP -s

Fig. 7 Dashboard of PSExec and Powershell encoded activity

the entire log displayed the user who executed the command including the name of the device shown in Figure 6.

Our team created a dashboard to keep track of suspicious Powershell activity including the usage of Powershell for lateral movement within the domain illustrated in Figure 7. With our active indexers in place, we search for known vulnerabilities from the PSExec and WMIC processes that allows executing programs on remote systems. PsExec simplifies the setup and installation of client software on remote systems and gives users full interactivity of console applications, while WMIC is a built in Windows application that allows users to run batch scripts on remote systems from the console [11], [12]. These processes have legitimate uses for system administrators to manage systems on the network, but they also provide a potential tool for attackers to spread malware. If given access to a domain administrator account, an attacker can utilize lateral movement modules in Empire to infect other systems the admin has access to. This makes it paramount to track malicious Powershell at its source before self-replicating throughout a domain of hosts.

In our results, we were able to narrow down Powershell activity corresponding to the system and user when we deployed our implementation to production. Figure 8 depicts the detected frequency of Powershell encoding and PSExec activity over the course of a 30 day period. Furthermore, anytime our automated search heads detect suspicious activity, it will fire off alerts and we will be able to undergo appropriate measures to isolate and further investigate the device.

Figure 8 Frequency of detected user and system PS activity

User Activity

	User	Activity	Sparkline	Count
1	FNTUBE-WF-P1$	PSHELL-ENC		2
2	LMC-056800VM10$	PSHELL-ENC		1
3	LMC-060505$	PSHELL-ENC		4
4	LMC-063912$	PSHELL-ENC		8
5	clong	PSEXEC		6
6	jaustin	PSEXEC		19
7	jmorello	PSEXEC		9
8	ryansmit	ADMIN		14
9	ryansmit	PSEXEC		15
10	ryansmit	PSHELL-ENC		2
11	ryansmit	WMIC		2

(a)

System Activity

	System	activity	Sparkline	Count
1	ITS-TS	PSHELL-ENC		5
2	JAUSTIN-DT	PSEXEC		19
3	LMC-042002	PSHELL-ENC		31
4	LMC-043191	PSEXEC		6
5	LMC-045096	PSHELL-ENC		146
6	LMC-052247	PSEXEC		9
7	LMC-056800VM10	ADMIN		18
8	LMC-056800VM10	PSEXEC		17
9	LMC-056800VM10	PSHELL-ENC		6
10	LMC-056800VM10	WMIC		7
11	LMC-057066	MIMIKATZ		1
12	LMC-060505	PSHELL-ENC		4
13	LMC-063912	PSHELL-ENC		8
14	chps-prod-sql2	PSHELL-ENC		2
15	fntube-wf-p1.jpl.nasa.gov	PSHELL-ENC		2

(b)

IV. Concluding Remarks

Powershell can be used as a standalone tool to perform a variety of tasks on a Windows system but also presents a security flaw with potential for misuse. An attacker can encode or obfuscate payloads to execute malicious commands and evade traditional methods of detection. By actively monitoring for certain keywords, we can flag potential indicators of obfuscation. This is done by scaling our log collection to a central server for processing. At this time, we are capable of detecting some of the basic forms of Powershell attacks and obfuscations. Our next approach will be moving on to detect relatively higher levels of obfuscation.

References

[1] “API Sets for Universal Windows Platform (UWP) apps.” API Sets for Universal Windows Platform (UWP) apps (Windows), Microsoft.

[2] “Writing a Windows Powershell Cmdlet.” Cmdlet Overview, Microsoft Developer Network. Microsoft.

[3] “The Increased Use of Powershell Attacks.” The Increased Use of Powershell Attacks, Symantec.

[4] Dunwoody, Matthew. “Greater Visibility Through PowerShell Logging « Threat Research Blog.” FireEye, FireEye, 11 Feb. 2016.

[5] White, Jeff. “Pulling Back the Curtains on EncodedCommand PowerShell Attacks.” Palo Alto Networks, Unit 42, 10 Mar. 2017.

[6] Bohannon, Daniel. Invoke-Obfuscation: PowerShell obFUsk8tion Techniques & How To (Try To) D""e`Tec`T 'Th' 'Em'. Mandiant.

[7] Bohannon, Daniel. Obfuscation in the Wild: Targeted Attackers Lead the Way in Evasion Techniques « Threat Research Blog. FireEye, 30 June 2017.

[8] Jofre, Juanpablo. “About Escape Characters.” about_Escape_Characters | Microsoft Docs, Microsoft, 9 June 2017. Accessed 14 Aug. 2017.

[9] Kazanciyan, Ryan. Hastings, Matt. “Investigating Powershell Attacks.” FireEye. Black Hat USA, 2014.

[10] “Splunk Software as a SIEM.” Improve your security posture by using Splunk as your SIEM, Splunk. 2016.

[11] Russinovich, Mark. “PsExec v2.2.” PsExec - Windows Sysinternals | Microsoft Docs, Microsoft, 29 June 2016.

[12] Wilansky, Ethan. “WMIC - Take Command-Line Control over WMI.” Microsoft Developer Network, Microsoft, Mar. 2002.