Attack and Defense Around PowerShell Event Logging - NSFOCUS, Inc., a global network and cyber security leader, protects enterprises and carriers from advanced cyber attacks.

By NSFOCUS

Published: 2019-02-27 · Archived: 2026-04-05 15:44:22 UTC

0x00 Overview

PowerShell has been a focus of concern for network defense. The fileless PowerShell, featuring LotL and excellent ease of use, is widely used in various attack scenarios. In order to capture PowerShell-based attacks, an increasing number of security professionals tend to, through PowerShell event log analysis, extract attack records such as post-exploitation data for enterprise security monitoring, alerting, trackback, and forensics.

Thus, it can be seen that how to evade event logging has become an important phase in network defense. Keeping tabs on continuous improvements in security features in the PowerShell event viewer, attackers employ a variety of techniques and methods to corrupt data concerning the PowerShell logging tool itself and compromise the integrity of event logs. The vulnerability (**CVE-2018-8415**) patched by Microsoft in October 2018 is another means to evade the logging of the PowerShell event viewer. This document dwells upon security features of the

logging function of major versions of PowerShell, as well as attack means, ideas, and techniques against each version of the event viewer.

0x01 Introduction to PowerShell Attack and Defense

PowerShell is a powerful scripting language and shell framework mainly available for Windows-based computers, facilitating system management by administrators and likely to replace the default command prompt window on Windows operating systems. The PowerShell script, with good functions, is usually used for normal system management and security configuration. Unfortunately, these features are also understood by attackers who have translated them into attack features (see below), and therefore, they will definitely pose serious threats to enterprise networks.

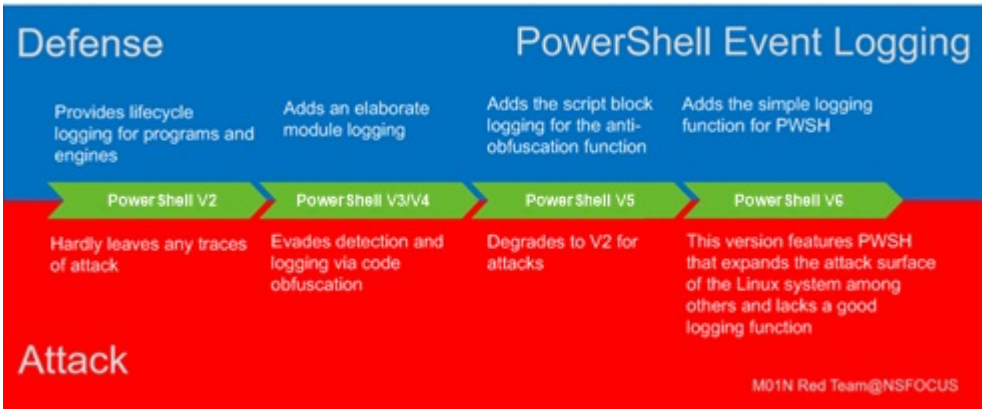
Main Attack Characteristics of PowerShell

- **Fileless anti-AV tool** to evade firewalls, various antivirus software, and intrusion prevention systems: With its fileless feature, PowerShell can be loaded from the memory and execute arbitrary code, without touching the hard disk.
- **LotL**: Attackers can easily reach the attack destination, while evading common attack detection systems and intrusion prevention systems. PowerShell comes with many Windows operating systems. It is highly unlikely that these built-in trusted tools could be detected and restricted by anti-malware software. Using such tools, attackers can effectively dodge common attack detection systems and intrusion prevention systems, without adding extra binaries.
- **Obfuscating code extremely easily**: Sharing characteristics of scripting languages, the flexible PowerShell, in conjunction with multiple obfuscation methods, can easily render traditional detection tools ineffective.
- **Excellent functionality and adaptability** to fit into various attack scenarios: PowerShell has a remote management mechanism built in for remote command execution. PowerShell supports WMI and .NET Framework, boasting incredible ease of use.

Since PowerShell was released in 2005, Microsoft has fixed a number of security issues in PowerShell during the 13-year course of fighting between defenders and offenders, making the PowerShell attack environment increasingly difficult to be exploited. One of important security measures Microsoft puts in place is the ScriptBlock logging function of PowerShell, which keeps full records of historical executions of PowerShell. Of course, this function contributes to attack trackback and forensics. However, there is a reciprocal relationship between offenders and defenders, who have been and will continue to be fighting a long battle against each other. The security defense research has been focusing on vulnerabilities in the PowerShell's logging function and the logging bypass method. In July 2018, a foreign security researcher nicknamed @Malwrologist spotted a flaw in the logging module of PowerShell, which allows attackers to truncate logs with null characters, causing the missing of important logs. Microsoft has fixed this issue (assigned **CVE-2018-8415**) in its patches released for this month.

PowerShell's Logging Function from the RT&BT Angle

Prior to analysis of this vulnerability, we will give a summary of PowerShell's logging function from the RT&BT angle. Now, we will look back the protection ideas and attack means of various PowerShell versions.



0x02 PowerShell V2—Initial Version of PowerShell

PowerShell V2 provides an event logging capability which assists the Blue Team (defender) with deduction and correlative analysis of attack events. Its simple logging function hardly records any traces of post-exploitation. For later versions, attackers, considering system compatibility, will try to downgrade later versions to V2 to evade logging.

PowerShell V2, as the initial version of this utility released by Microsoft, provides a basic event recording capability to make a brief record of events, but lacks a satisfying logging function. All the same, the default logging level of the old version can also provide sufficient evidence to show the way PowerShell is used. Besides, PowerShell V2 isolates remote handling from local activities and provides contextual information such as the session duration and related user accounts. All of these are sufficient to help members of the Blue Team with deduction and correlative analysis of attack events.

Defense angle (from the perspective of Blue Team)

When any PowerShell commands or scripts are executed, whether locally or remotely, Windows can write events into three log files:

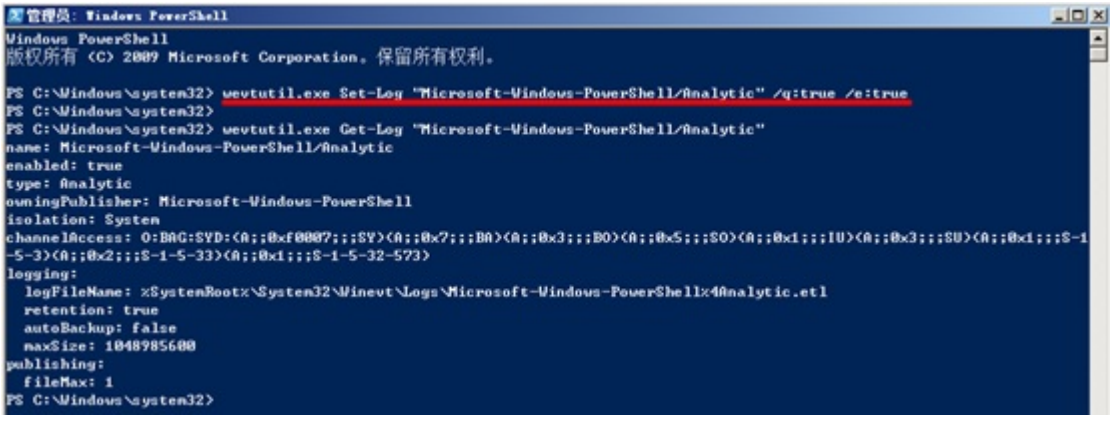
- Windows PowerShell.evtx
- Microsoft-Windows-PowerShell/Operational.evtx
- Microsoft-Windows-PowerShell/Analytic.etl

PowerShell implements remote handling via Windows Remote Management (WinRM): The following event log files can capture remote PowerShell activities.

- Microsoft-Windows-WinRM/Operational.evtx
- Microsoft-Windows-WinRM/Analytic.etl

Usually, PowerShell 2.0 can generate event logs indicating when the command or script execution starts or ends, what provider (show the type of function being in use) is loaded, and which user account performs activities, but fail to present detailed historical records of all executed commands or output. Analytic logs contain more information, helping us locate where certain errors occur. However, the analytic logging function, once enabled (disabled by default), will generate a great deal of record data in the production environment, hindering the actual analysis.

To view analytic logs, users can click **Show Analytics and Debug Logs** in the menu bar of the event viewer and select **Enable Log** in Microsoft-Windows-WinRM/Analytic or run the **wevtutil Set-Log** command to enable the logging function:

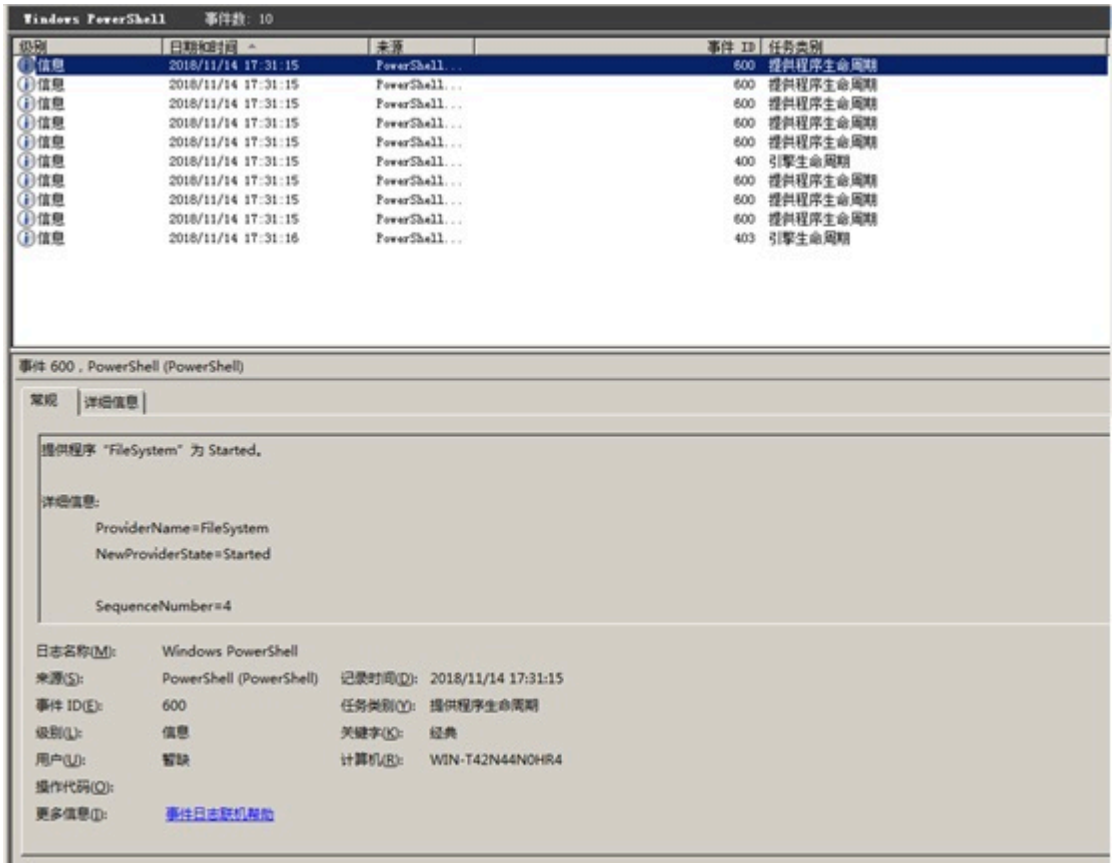


The following is a summary of important evidence captured by each event log file of PowerShell 2.0.

Windows PowerShell.evtx

Each time PowerShell executes a single command, whether it is a local or remote session, the following event logs (identified by event ID, i.e., EID) are generated:

- EID 400: The engine status is changed from None to Available. This event indicates the start of a PowerShell activity, whether local or remote.
- EID 600: indicates that providers such as WSMan start to perform a PowerShell activity on the system, for example, “Provider WSMan Is Started”.
- EID 403: The engine status is changed from Available to Stopped. This event records the completion of a PowerShell activity.



The HostName field is included in message details of events identified by EID 400 and EID 403. For a local activity, this filed is recorded as ConsoleHost (HostName = ConsoleHost); for a remote activity handled by PowerShell, HostName is recorded as ServerRemoteHost (HostName = ServerRemoteHost) on the system that is accessed.

Neither message records the user accounts associated with PowerShell activities. Viewing these events, analysts can determine how long a PowerShell session lasts and whether the session runs locally or remotely.

Microsoft-Windows-PowerShell/Operational.evtx

For PowerShell 2.0, this log file is not found to record any material information.

Microsoft-Windows-WinRM/Operational.evtx

This log file records the use of WinRM, including remote handling activities by PowerShell.

- EID 6: recorded when a remote handling activity is started on the client system, including the destination address to which the system is connected.
- EID 169: recorded when a remote handling activity is started on an accessed system, including the user name and authentication mechanism used to access WinRM.
- EID 142: If WinRM is disabled on the remote server, this event is recorded when the client attempts to initiate a remote shell connection.

Microsoft-Windows-PowerShell/Analytic.etl

As mentioned above, events can be captured only when analytic logging is enabled. Capturing such events is intended for troubleshooting rather than long-term security audits. When active, the log file records all security events relating to remote code execution under the following event IDs:

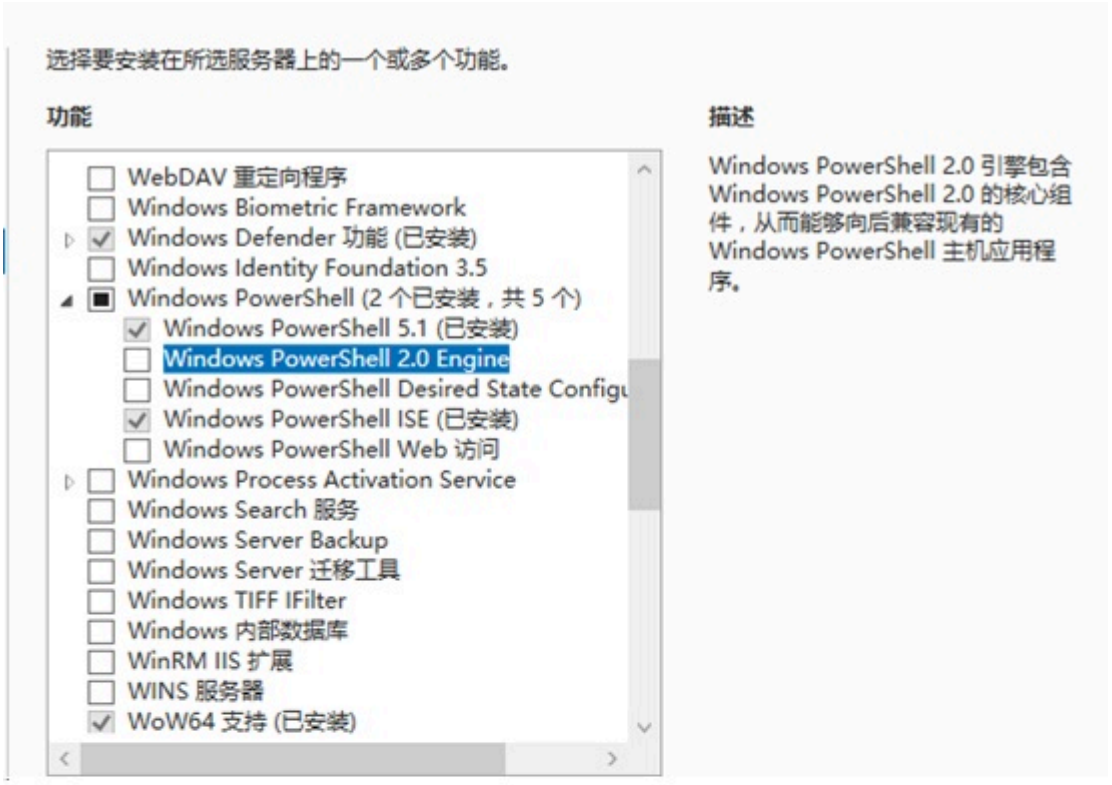
- EID 32850: records the user account authenticated for remote handling.
- EID 32867/32868: records each PowerShell input and output object exchanged during the remote handling of PowerShell, including protocol and version negotiation as well as command I/O. The objects are stored as XML-encoded hexadecimal strings in a field denoted “Payload data”. Such objects, because of the length, are often fragmented across multiple log messages.
- EID 142: If WinRM is disabled on the remote server, this event is recorded when the client attempts to initiate a remote shell connection.

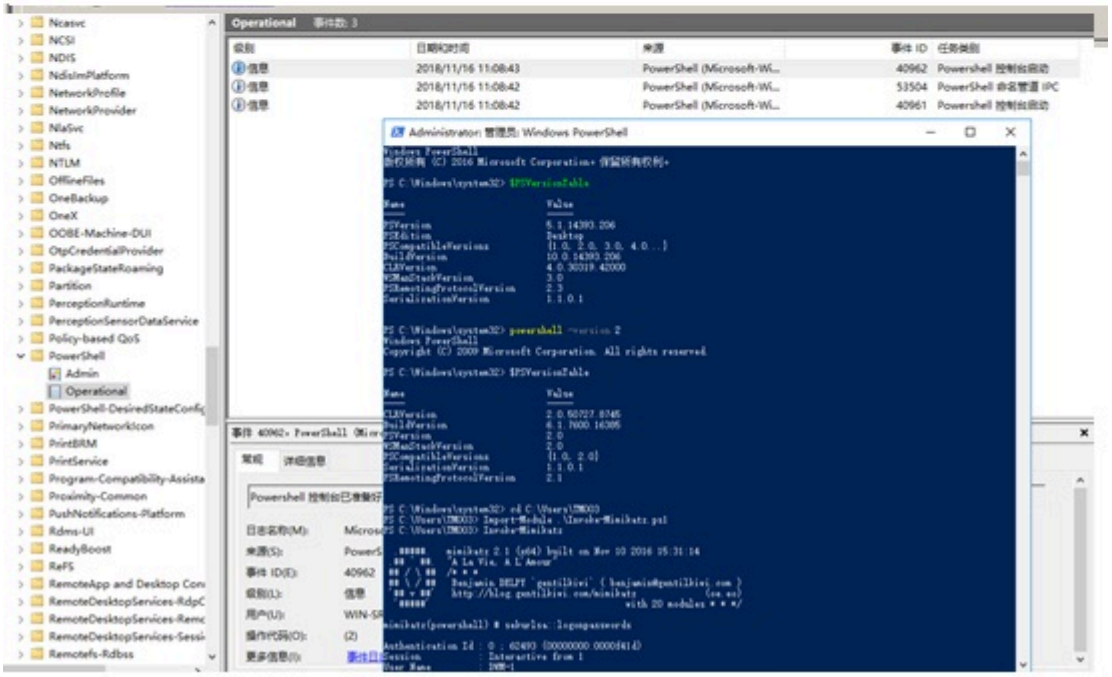
Microsoft-Windows-WinRM/Analytic.etl

Similar to PowerShell Analytic logging, WinRM Analytic logging is not enabled by default. Once configured, it generates a great number of events which are encoded once again and difficult to analyze.

Attack angle (from the perspective of Red Team):

Due to the incompleteness of logs in earlier versions, various post-exploitation activities concerning PowerShell are nearly traceless. Even in later versions, as PowerShell 2.0 can be enabled on the system, attackers, by taking advantage of forward compatibility, often run the **powershell -version 2** command to switch the PowerShell command line to PowerShell 2.0 to evade the logging function. This is analogous to “downgrade attack”.





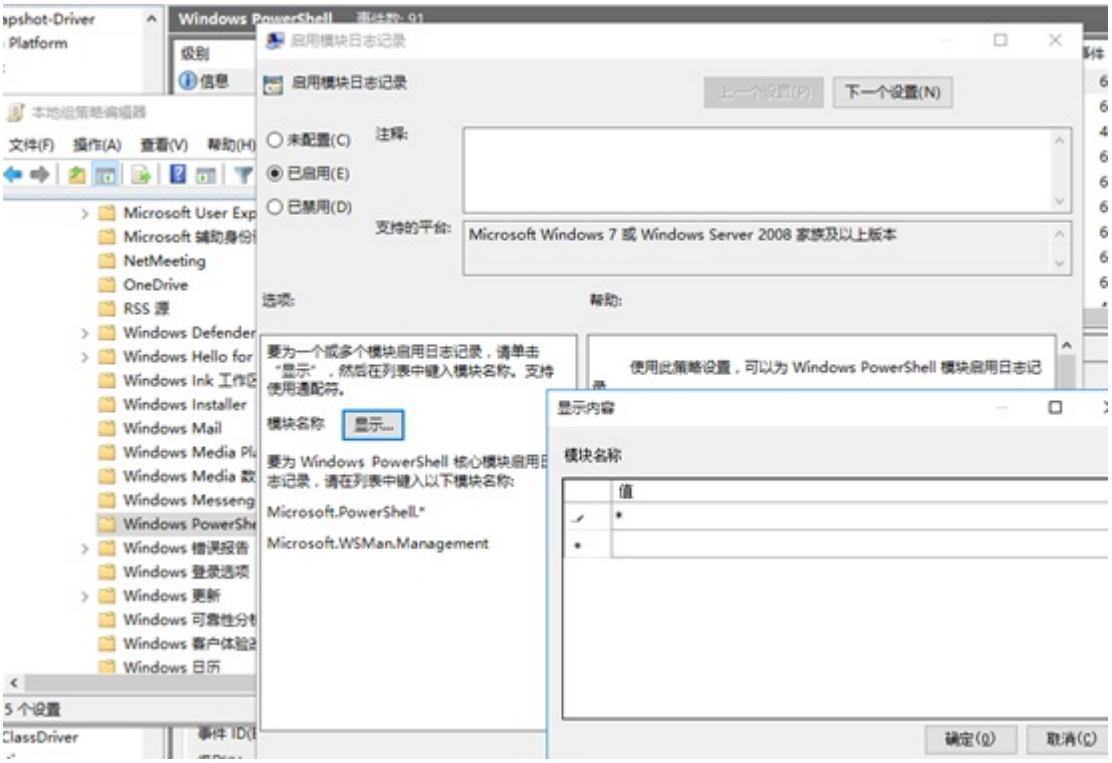
0x03 PowerShell V3/V4 Featuring Comprehensive Logging

Compared with earlier versions, PowerShell V3/V4 provides a more comprehensive logging function. At this time, attackers turn to obfuscation means to obscure logs so as to escape identification and detection.

Windows PowerShell 3.0 improves the logging and tracing support for commands and modules, with support for Event Tracing in Windows (ETW) logs, an editable LogPipelineExecutionDetails property of modules, and the

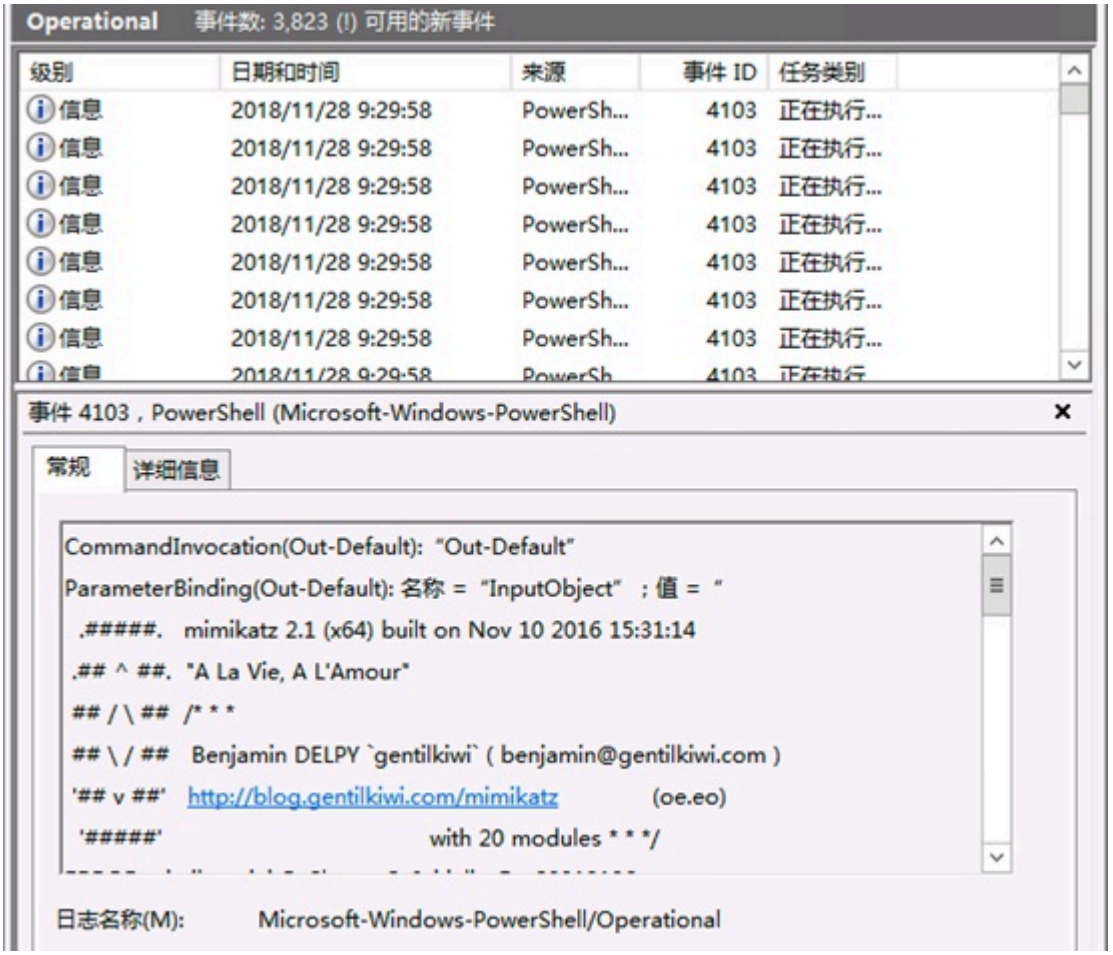
“Turn on Module Logging” Group Policy setting. PowerShell module logging has been available since PowerShell V3 and will log all events to EID 4103.

PowerShell module logging can be configured to record all activities of each PowerShell module, covering single PowerShell commands, imported modules, and remote management. The module logging function can be enabled by configuring GPO settings.



Alternately, setting the following registry values will have the same effect:

- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging → EnableModuleLogging = 1
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging \ModuleNames → * = *



The module logging records the CommandInvocation type and ParameterBlinding content involved during PowerShell script or command execution, covering the execution process and input/output contents. The addition of the logging function makes it possible to almost keep complete records of PowerShell execution logs, greatly facilitating log analysis and alert monitoring.

In view of the attack-defense development, after this version is released, attackers also consider other methods to escape the logging function, for example, employing quite a few obfuscation algorithms for obscuring data.

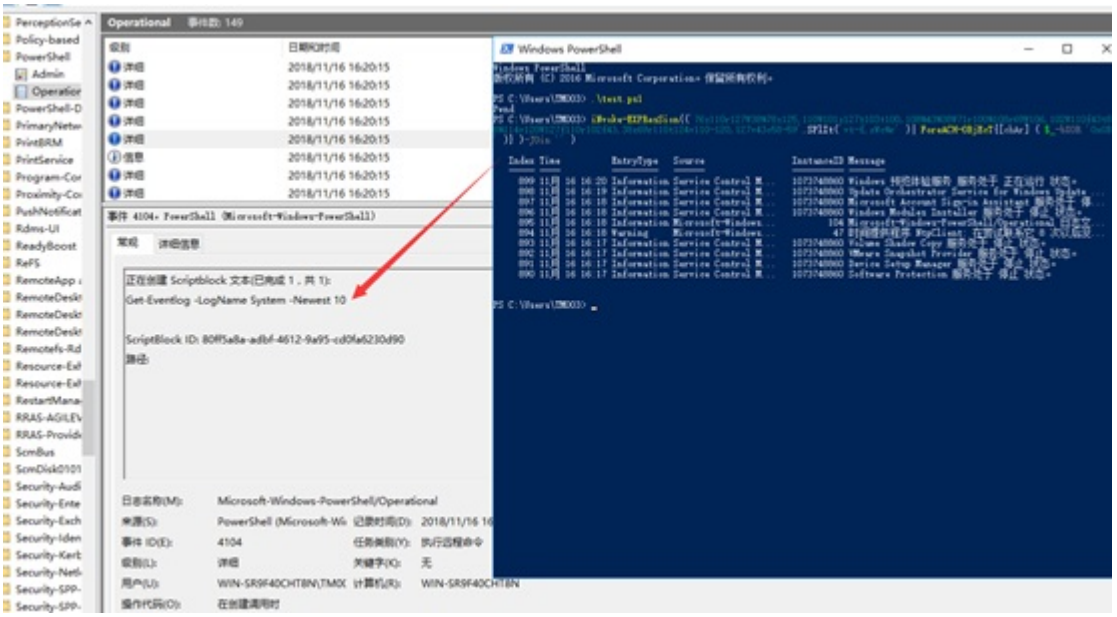
0x04 PowerShell v5 Capable of Deobfuscation

PowerShell V5 adds the CLM and ScriptBlock logging functions, thus capable of deobfuscating PowerShell code and recording event logs to effectively fight against previous attack means. At this time, the attack thinking lays great emphasis on how to downgrade to PowerShell V2.

As the PowerShell attack technology matures, attackers have carried out a lot of code obfuscations to evade protection and logging. However, it is difficult to discover or confirm what these code is executed for, prior to code execution. This makes attack detection and forensics a trickier job. For this reason, Microsoft has added the log dumping and ScriptBlock logging functions to PowerShell V5.0 and later and logs all events to EID 4104. The ScriptBlock logging gives the capability of recording de-obfuscated PowerShell code.

As script code needs to be de-obfuscated prior to execution, the ScriptBlock logging function records the actual code before it is passed to the PowerShell engine for execution. Therefore, many centralized log systems hardly

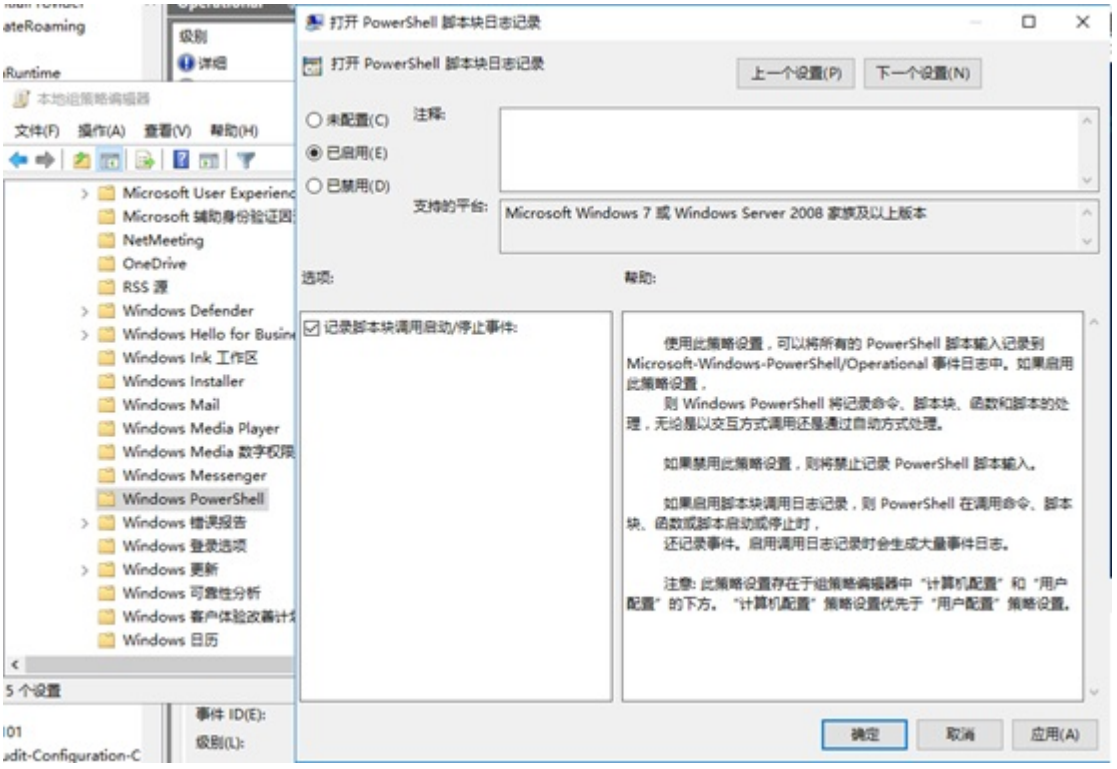
report an alert when capturing suspicious logs. Of course, in my opinion, such alerts are valuable for sample analysis and emergency forensics.



To enable ScriptBlock logging, users can run PowerShell V5 with administrative privileges and execute the following commands:

Install-Module -Name scriptblockloggingnalyzer -Scope CurrentUser
set-SBLLogSize -MaxSizeMB 1000
Enalbe-SBL

Alternatively, users can enable this function and record script file invocation information by configuring GPO settings:



Certainly, users can enable this function by modifying the following registry key:

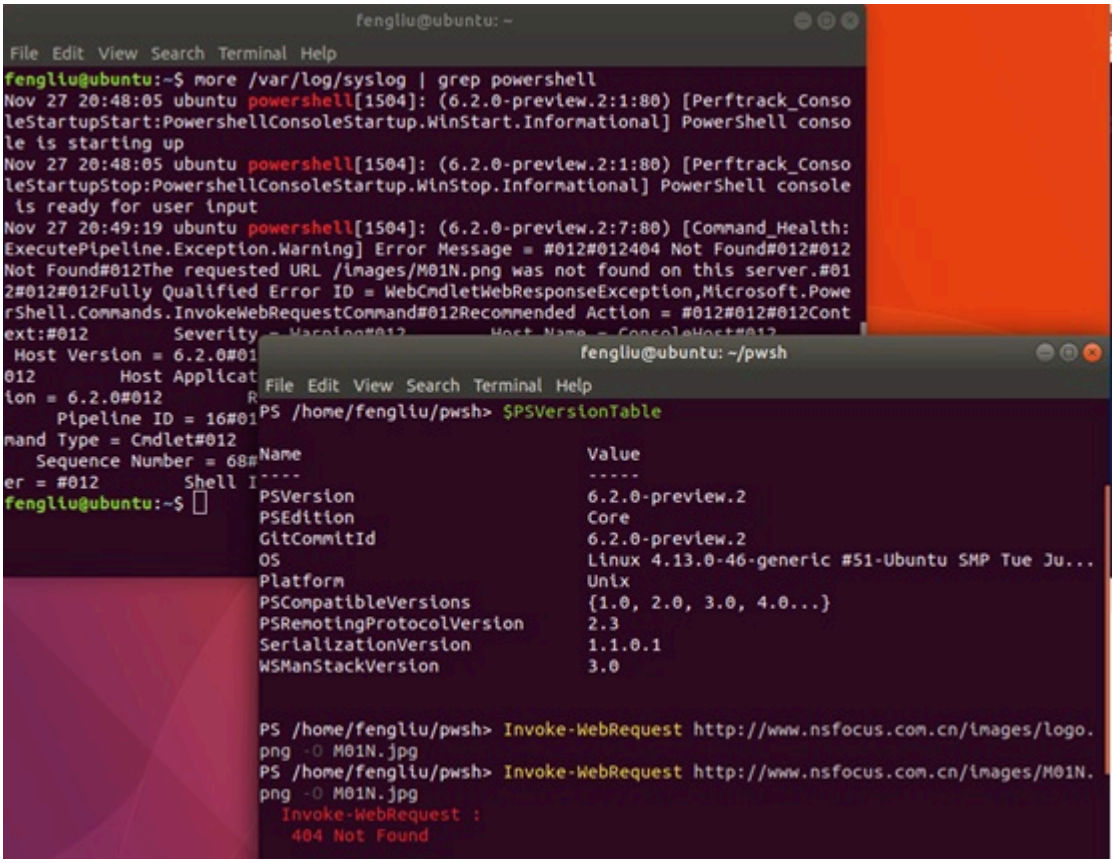
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging → EnableScriptBlockLogging = 1

PowerShell 5.0 supports Windows 7/2008 R2 and later. Though a number of enhanced logging functions in PowerShell 5.0 are reversely ported to PowerShell 4.0, we recommend that PowerSehll 5.0 be installed on each Windows platform. PowerShell 5.0 provides functions unavailable in V4.0, such as generating logs for suspicious script blocks.

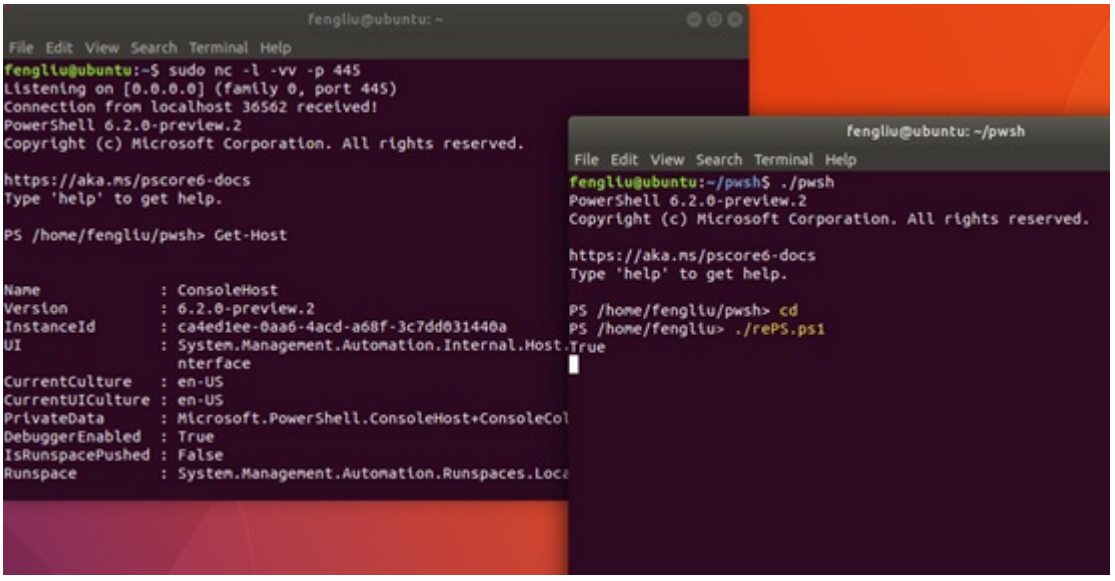
0x05 PowerShell V6 Providing PWSH—A New Attack Surface

Out of functional requirements, PowerShell V6 supports more operating systems and also exposes a new attack surface—PWSH.

As PowerShell is installed with PWSH on Linux and macOS among other operating systems, logging is an indispensable part, for the sake of security. PowerShell uses the os-log API on a local host to log in to Apple’s uniform logging system. On Linux, PowerShell uses the syslog-based logging function which Microsoft has elevated to a logging solution almost available across the platform.



Attack thinking (from the perspective of Red Team): The addition of PowerShell to other systems, though facilitating management by administrators, undoubtedly expands the attack surface of these systems. Besides, the logging function is inadequate in the latest version currently available. During certain relevant tests, I find two points: 1. Logs are generated once a PowerShell execution error is reported. 2. If PowerShell runs properly without an error, only two syslog logs are generated: “PowerShell console is starting up” and “PowerShell console is ready for user input”. For instance, creating a simple reverse shell is another method and this reverse PWSH operation is yet logged.



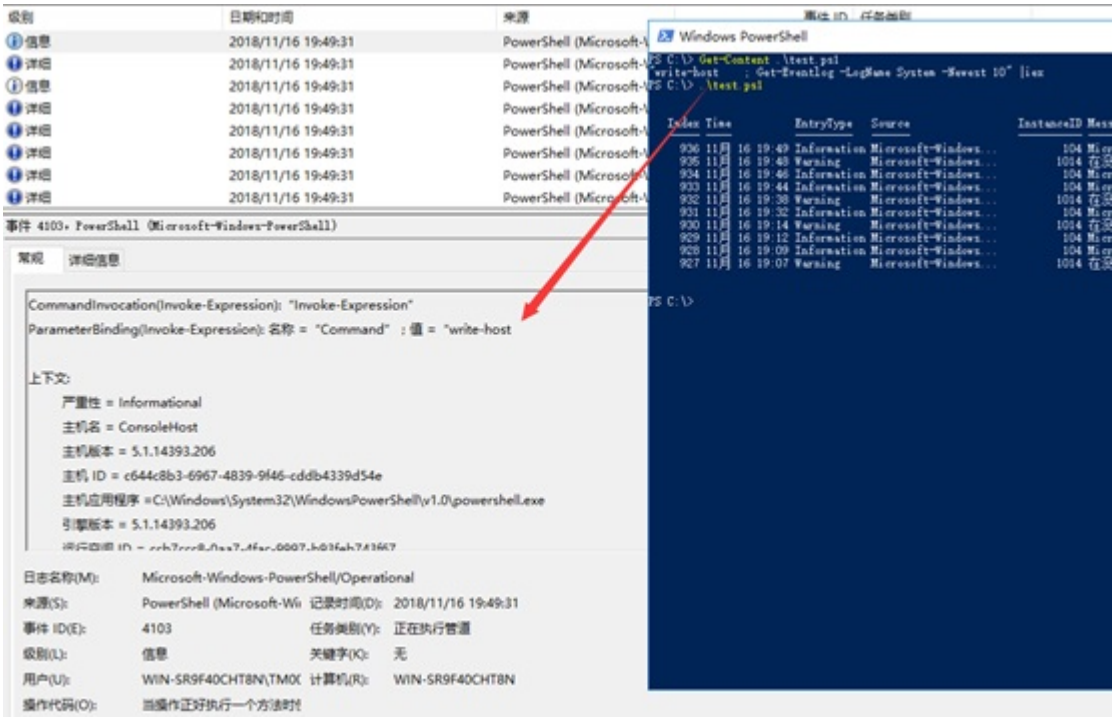
0x06 Logging Bypass Vulnerability (CVE-2018-8415)

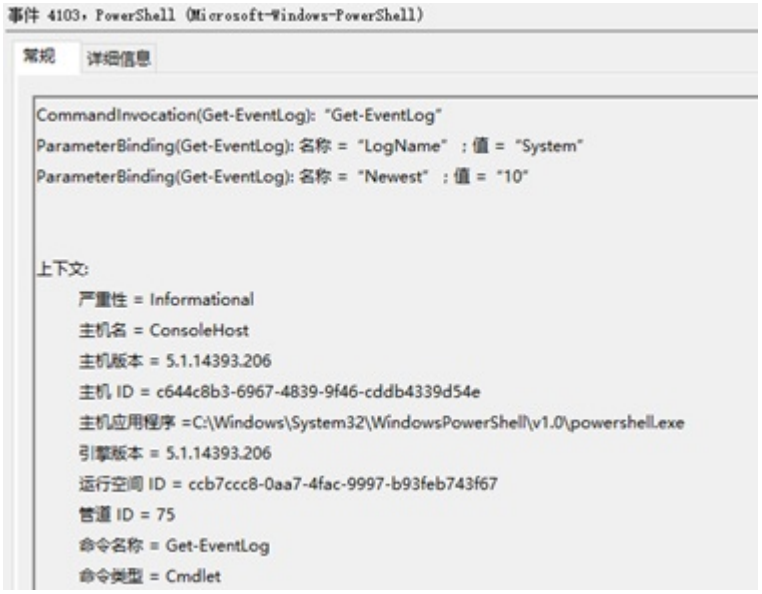
A tampering vulnerability exists in PowerShell that could allow an attacker to execute unlogged code. To exploit this vulnerability, an attacker would need to log on to the affected system and run a specially crafted application. The security update addresses the vulnerability by correcting log management of special characters.

Microsoft describes and rates this vulnerability as “important” (yet to reach the severe level). Taking advantage of this vulnerability, attackers, via crafted code, could bypass the ScriptBlock logging function that is mentioned above. According to the patch indicated on the GitHub website, this vulnerability affects all PowerShell Core versions (including PWSH) and the patch remediation solution only replaces \u0000 with \u2400 in Unicode. From the comments included in the patch shown in the following figure, we can speculate about how the vulnerability works, which, to put it simply, is that truncating logs with null characters leads ScriptBlock logging to stop recording commands.

```
1356 1358                         {
1357 1359                             textToLog = encodedContent;
     1360 +                           wasEncoded = true;
1358 1361                         }
1359 1362                     }
1360 1363                 }
1361 1364             }
1362 1365
     1366 +         if(!wasEncoded)
     1367 +         {
     1368 +             // The logging mechanism(s) cannot handle null and rendering may not be able to handle
     1369 +             // null as we have the string defined as a null terminated string in the manifest.
     1370 +             // So, replace null characters with the Unicode `SYMBOL FOR NULL`
     1371 +             // We don't just remove the characters to preserve the fact that a null character was there.
     1372 +
     1373 +             textToLog = textToLog.Replace('\u0000','\u2400');
     1374 +         }
     1375 +
1363 1376             if (scriptBlock._scriptBlockData.HasSuspiciousContent)
1364 1377             {
1365 1378                 PSEtwLog.LogOperationalWarning(PSEventId.ScriptBlock_Compile_Detail, PSOpcode.Create, PSTask.ExecuteCo
```

The vulnerability discoverer nicknamed @Malwrologist revealed this issue on his own Twitter profile as early as July 2018. Following the discoverer’s thinking, we have reproduced this vulnerability and found that because of the null character restriction, this vulnerability can only be triggered during script execution. Due to inner restrictions of the command-line environment, it is impossible to exploit this vulnerability using a single command. Of course, EID 4103 event logs cannot be truncated as intended during the execution of multiple spliced commands, with key-value pairs in the logs still left.





Attack thinking (from the perspective of Red Team): Though key-value pairs are left after exploitation of the vulnerability, attack script code, in actual attack scenarios, features a complicated execution logic for implementing related functions. Moreover, as EID 4103 logs lack the deobfuscation capability, generalizing script functions and attack intentions from a large amount of data incurs a high analysis cost. Clearly, this vulnerability is still very valuable for launching attacks.

0x07 Conclusion

As a matter of fact, PowerShell has been widely exploited for carrying out attack activities of varied scales. For instance, it can be seen in downloaders, horizontal scaling of the intranet, privilege maintaining system backdoor, and even attack events initiated by APT organizations such as MuddyWater and FruityArmor. Predictably, PowerShell will still be a hot technology in the coming years. Therefore, as important data support for alert

monitoring in this regard, PowerShell event logs must be given to full play. It is recommended that enterprise users keep the PowerShell event viewer updated to the latest version all the time and enable ScriptBlock logging for better defense.

NSFOCUS Fu Ying Labs will keep abreast of the latest attack and defense technologies and threat risks. All people interested in various attack and defense technologies are welcome to communicate with us.

0x08 References

https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/

https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8415

https://github.com/PowerShell/PowerShell/pull/8253

https://twitter.com/DissectMalware/status/1016462916059631616

About NSFOCUS Fu Ying Labs

NSFOCUS Fu Ying Labs focuses on security threat research and monitoring technologies, covering threat identification, tracing, and capture technologies as well as threat actor identification technologies.

By doing research in botnet threats, anti-DDoS, web confrontation, threats of exploitation of vulnerabilities in popular service systems, ID authentication threats, digital asset threats, threats from the underground industry, and emerging threats, we have a good grasp of threats in the live network so as to identify risks, mitigate harms done by threats, and provide decision-making support for defense against threats.
