

The No Hassle Guide to Event Query Language (EQL) for Threat Hunting

 varonis.com/blog/guide-no-hassle-eql-threat-hunting/

March 24,
2020



By



Andy Green

Updated: 4/5/2020

Did you ever have the urge to put together a few PowerShell scripts to parse and analyze the Sysmon event log in order to create your own threat analysis software? Nothing to be embarrassed about! But before you do anything rash, you should first read about the results of my own modest efforts in this area. If you're still convinced you want to take on this project, you'll quickly realize, as I did, how *hard* it is to develop real enterprise-level threat monitoring.

[Part I: Overview of EQL](#)

[Part II: Finding Threats With EQL's Join and Sequence](#)

[Part III: Advanced Threat Detection](#)

Thankfully, there's now open-source software, known as EQL, that takes care of the painful parts of this project involving parsing and organizing the Sysmon event data. With EQL, you can practically dive right in, avoiding the steep curve I went through herding PowerShell, Sysmon, JSON, data structure, and more into something useful.

In Part I, we'll quickly cover the basics of EQL, and then in Part II, we'll show how you can do fairly sophisticated behavior-based analysis to discover threats. Part III is an even deeper dive as we start exploring the Mitre Att&ck threat model with EQL.

EQL is a first step to getting a handle on cyber threats. But there's only so much that you do on your own! Watch the Varonis Incident Response team detect and manage real-world attacks. Register now for a live workshop.

Part I: Overview of EQL

EQL (pronounced equal) is a cybersecurity language designed by the folks at Endgame — love that name! — a cyber research and consulting company. While EQL was initially restricted for use within the company, it was released in 2018 as an open-source project on [Github](#) to nurture collaboration among security practitioners worldwide. Great idea! EQL also has potential as a pen-testing tool, which we'll explore in a future post.

The EQL core language is based on Python, there is an integration with Windows [Sysmon](#), and there are extensive [analytics](#). EQL benefits from its ability to match events, stack data, and perform analysis of aggregate data sets. In plain-speak, you can easily tap into a lot of process context that would usually require a complex query and coding for something like, "all the processes that performed network activity that are descended from "regsvr32.exe". It's also schema-independent and *OS-agnostic*, and so can be used with any data set or operating system (Linux, Windows).

The goal of EQL is to go beyond legacy reliance on Indicators of Compromise (IoCs) by using familiar shell-type syntax to craft queries for spotting interesting behaviors. By the way, the security analytics capabilities match up with the [Mitre ATT@CK](#) model. There's a lot more information on the EQL way of doing things in this very watchable [video](#) — skip to about the 6:50 mark.

EQL Ideas

Let's get into the basis in this initial dive. To simplify search, EQL thankfully drops the over-abundance of keywords found in PowerShell scripts in favor of a simpler, more practical function syntax. These functions can be used to perform math, and create more sophisticated expressions without entering long keyword-heavy strings. Yay!

Booleans

As you'd expect, EQL has boolean operators (and, or, not), the usual comparers (<,>, <=,>=,! =), and there's a case-insensitive wildcard search available via the asterisk character. For example, if you wanted to look up a "svchost" service process that doesn't have either -k in the command line, or services.exe as a parent process, you would write the query like this:

```
process where process_name == "svchost.exe" and (command_line != "* -k *" or parent_process_name != "services.exe")
```

Sequences

EQL sequences can be used to identify data points that share common attributes, such as a common process path and file path. These sequence queries can also be made time- and event-sensitive, for example, you can set a tracking point to end when a log-off event occurs, or after a set period of time has elapsed. This can be helpful to remove non-unique entries and reduce memory usage.

A generic sequence query looks like this:

```
sequence[event_type1(process, network, etc.) where condition1]
[event_type2 where condition2] ...
[event_typeN where conditionN]
```

Joins

Bless them for making database-like joins very simple. Joins can be used to match unordered events that share one, or several, user-defined properties. EQL's join can be thought of as a form of the sequences syntax, but without accounting for time constraints.

```
join by shared_field1, shared_field2, ...
[event_type1 where condition1]
[event_type2 where condition2] ...
[event_typeN where conditionN]
```

Pipes

Pipes can be used to conveniently filter and reduce the number of results from a data set and add more specificity to queries in post-processing. You can remove duplicate entries using the 'unique' pipe, can request the most (or least) common entries in a data set with the 'filter' command, or sort with, you guessed it, the 'sort' command.

For example, the 'count' pipe returns the total number of entries found matching the search query. The basic pipe structure is as follows:

```
process where true | count

// results look like// "count": 100, "key": totals"
```

Process Lineage

Process lineage is *automatically* tracked to simplify the discovery of vital information about a process, such as its origin and how long it has been live. Using this lineage data, EQL can really isolate results you need.

You can limit results to just process identifiers (PIDs) with a specific parentage, or remove PIDs based on *when* they became active. This helps avoid searching the same results over and over since you can quickly ignore PIDs that were active *before* you last conducted the

same search. To search for PowerShell entries that weren't launched by Windows Explorer, you'd enter this string:

```
process where process_name == "powershell.exe" and not descendant of [process where process_name == "explorer.exe"]
```

Hello World in EQL

This post is just a very quick introduction, and we'll explore EQL more thoroughly next time. Can an infosec blogger install and run a very trivial EQL query in about 30 minutes? The answer is yes.

I should have pointed out earlier that EQL works with JSON input. So if you want to analyze Sysmon output you need to convert it to JSON — which is easy to accomplish in [PowerShell](#) — and then it run through EQL. Fortunately, the Endgame gang has already published a few [datasets](#) on Github to work with.

Using one of the sets, I ran the EQL equivalent of “hello world” to count the number of Windows processes that start with “cmd”:

```
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$ ls *.json
example.json  rgsvr32.json
ubuntu@ip-172-31-1-94:~$ eql query -f rgsvr32.json 'process_name == "cmd*" | count'
{"count": 7, "key": "totals"}
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$
ubuntu@ip-172-31-1-94:~$ □
```

Victory! I installed EQL and got it to parse JSON input in about 30 minutes!

My point in this first section is to show that this is a very approachable language. In the next section, we'll explore how to leverage EQL to accomplish useful threat exploration tasks.

Part II: Finding Threats With EQL's Join and Sequence

Before we dive into EQL's powerful features, let's step back and examine what we're really trying to do. The larger point about EQL is that it is most emphatically not meant to search for specific malware names or other obvious indicators — plain-text keywords buried in files.

Since you feed in low-level Sysmon log files to EQL, you have very granular access to processes that are created, dlls loaded, as well as files read and written. With this information and real-world knowledge of threats — thank you, Mitre! — we can hunt for the *underlying activities* that won't show up in a legacy virus or signature scan.

The EQL gang has even put together a [mapping](#) of the Mitre Att&ck matrix into corresponding EQL statements. If you want to get ahead of the game, you can take a peek at their threat scenarios, but we'll return to it in Part III.

Working with the Sysmon Log

EQL is a query language, so what is it querying? Answer: A JSON-formatted file based on the Sysmon log.

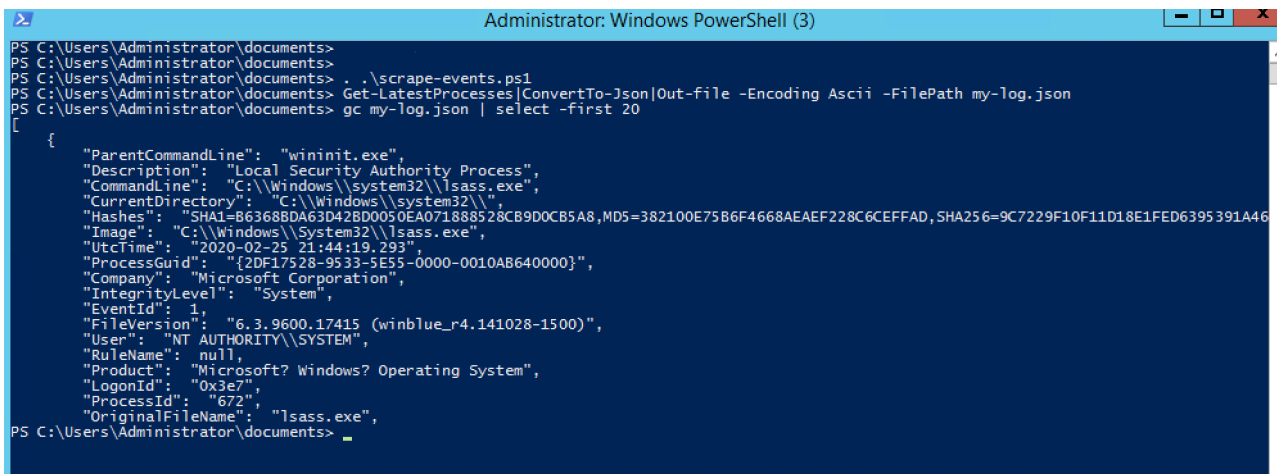
Fortunately, EQL has nice utility that will do the heavy lifting. You can go to my Github repository to download [scrape-events.ps1](#), and then run the Get-LatestProcesses function to do the conversion, like so:

```
Get-LatestProcesses | ConvertTo-Json | Out-File -Encoding ASCII -FilePath my-log.json
```

Or if you're a big fan of our Sysmon threat hunting [post](#) and already have installed my home-brewed module, you can run this as well

```
get-sysmonlogs | ConvertToJson | Out-file -Encoding Ascii my-log.json.
```

If you don't have Sysmon, you can borrow some pre-cooked customized logs that can be found in EQL's [repository](#).



```
Administrator: Windows PowerShell (3)
PS C:\Users\Administrator\documents>
PS C:\Users\Administrator\documents>
PS C:\Users\Administrator\documents> . .\scrape-events.ps1
PS C:\Users\Administrator\documents> Get-LatestProcesses | ConvertTo-Json | Out-File -Encoding Ascii -FilePath my-log.json
PS C:\Users\Administrator\documents> gc my-log.json | select -first 20
[
  {
    "ParentCommandLine": "wininit.exe",
    "Description": "Local Security Authority Process",
    "CommandLine": "C:\\Windows\\system32\\lsass.exe",
    "CurrentDirectory": "C:\\Windows\\system32\\",
    "Hashes": "SHA1=B63688DA63D428D0050EA071888528CB9D0C85A8,MD5=382100E75B6F4668AEAEF228C6CEFFAD,SHA256=9C7229F10F11D18E1FED6395391A46",
    "Image": "C:\\Windows\\System32\\lsass.exe",
    "UtcTime": "2020-02-25 21:44:19.293",
    "ProcessGuid": "{2DF17528-9533-5E55-0000-0010AB640000}",
    "Company": "Microsoft Corporation",
    "IntegrityLevel": "System",
    "EventId": 1,
    "FileVersion": "6.3.9600.17415 (winblue_r4.141028-1500)",
    "User": "NT AUTHORITY\\SYSTEM",
    "RuleName": null,
    "Product": "Microsoft? Windows? Operating System",
    "LogonId": "0x3e7",
    "ProcessId": "672",
    "OriginalFileName": "lsass.exe",
  }
]
```

A pipeline for converting a Sysmon log into JSON format. You'll need EQL's [eventscraper.ps1](#).

Anyway, with the JSON log you're ready to work with EQL. The installation for Python-based EQL is [straightforward](#) on a Linux platform, and then once the EQL command interpreter is up, import the JSON log with the "input" command:

```

Downloads — ubuntu@ip-172-31-1-94: ~/eqlib — ssh -i andy-key.pem ubuntu@ec2-35-163-159-22.us-west-2.compute.amazonaws.com — 144x30
ubuntu@ip-172-31-1-94:~/eqlib$
ubuntu@ip-172-31-1-94:~/eqlib$
ubuntu@ip-172-31-1-94:~/eqlib$ ls
build          docs          eql_search.py  my-log.json    normalized-T1117-AtomicRed-regsrvr32.json  requirements_rtd.txt  tests
CONTRIBUTING.md  eql.engine    LICENSE        my-own-log.json  __pycache__   setup.cfg         utils
data           eqllib       Makefile       my-own-log.json.1  qt.py         setup.py
dist          eqllib.egg-info  MANIFEST.in   my-sysmon-data.json  README.md     sysmon-rta.json
ubuntu@ip-172-31-1-94:~/eqlib$ eql

=====
EQL SHELL
=====
type help to view more commands
eql> input my-own-log.json
Using file my-own-log.json with 971 events
eql> schema
{'process': {'Cline': 'string',
             'Cluster': 'string',
             'EdgeCnt': 'number',
             'Key': 'string',
             'PKey': 'string',
             'User': 'string',
             'Visited': 'number',
             'Weight': 'number',
             'event_type': 'string',
             'parent_process_name': 'string',
             'pid': 'string',
             'ppid': 'string',
             'process_name': 'string',
             'timestamp': 'number'}}
eql>

```

Input the raw JSON log, and check the schema just to see that it recognizes field names.

At a more technical level, you have to make sure that core field names are available to EQL, and so you should study the EQL schema for various types of Sysmon events, which can be found [here](#).

I'm focusing on "process" events for this guide, so you need a few fields (pid, ppid, timestamp) to accomplish some useful threat hunting. To get the full value of EQL, you'll also need to tweak the timestamp field so it represents *numeric ticks*. Sorry, but you'll have to add a PowerShell statement to do this as part of a pipeline (see the PS datetime object and the ticks property), and I'll leave that as an exercise.

Basic Threat Hunting With Join

EQL lets you experiment by making it easy to test your (educated and well-researched) hunches. An ordinary user who is working directly with PowerShell or the legacy cmd shell might be a tip off of that this account is engaging in unusual activities.

You can enter directly into the EQL command interpreter the following:

```
search process where process_name == "PowerShell.exe" or process_name == "cmd.exe"
```

Unfortunately, EQL will start spewing out JSON after you run this command, which may be tolerable for a small log file. The way to tame the rawl JSON is to use EQL's table statement, and format the output into columns:

```

y": "cmd.exe", "User": "NT AUTHORITY\\SYSTEM", "Visited": 0, "Weight": 0, "event_type": "process", "parent_process_name
: "cmd.exe", "pid": "1324", "ppid": "1312", "process_name": "reg.exe", "timestamp": 637189329391320000}
{"Cline": "\"C:\\Program Files\\Amazon\\Ec2ConfigService\\Scripts\\AWSAcpiSpcrReader.exe\"", "Cluster": "", "EdgeCnt":
, "Key": "AWSAcpiS.exe", "PKey": "cmd.exe", "User": "NT AUTHORITY\\SYSTEM", "Visited": 0, "Weight": 0, "event_type": "p
rocess", "parent_process_name": "cmd.exe", "pid": "2812", "ppid": "2800", "process_name": "AWSAcpiS.exe", "timestamp": 6
7172895386070000}
{"Cline": "C:\\Windows\\system32\\reg.exe query hklm\\software\\microsoft\\windows\\softwareinventorylogging /v collec
tionstate /reg:64", "Cluster": "", "EdgeCnt": 0, "Key": "reg.exe", "PKey": "cmd.exe", "User": "NT AUTHORITY\\SYSTEM", "V
isited": 0, "Weight": 0, "event_type": "process", "parent_process_name": "cmd.exe", "pid": "2680", "ppid": "2696", "proc
ss_name": "reg.exe", "timestamp": 637187603615050000}
{"Cline": "wmic.exe nic GET Name,NetConnectionID ", "Cluster": "", "EdgeCnt": 0, "Key": "wmic.exe", "PKey": "cmd.exe",
"User": "NT AUTHORITY\\SYSTEM", "Visited": 0, "Weight": 0, "event_type": "process", "parent_process_name": "cmd.exe", "
id": "1384", "ppid": "1148", "process_name": "wmic.exe", "timestamp": 637187603020840000}
{"Cline": "C:\\Windows\\system32\\cmd.exe /c C:\\Windows\\system32\\reg.exe query hklm\\software\\microsoft\\windows\\s
oftwareinventorylogging /v collectionstate /reg:64", "Cluster": "", "EdgeCnt": 0, "Key": "Cmd.Exe", "PKey": "cmd.exe", "
ser": "NT AUTHORITY\\SYSTEM", "Visited": 0, "Weight": 0, "event_type": "process", "parent_process_name": "cmd.exe", "pi
d": "2984", "ppid": "2960", "process_name": "Cmd.Exe", "timestamp": 637176350342060000}
{"Cline": "C:\\Windows\\system32\\cmd.exe /c \"\"C:\\Program Files (x86)\\GUESS\\guessallgui.bat\" \"\", \"Cluster\": \"\",
EdgeCnt\": 0, \"Key\": \"Cmd.Exe\", \"PKey\": \"explorer.exe\", \"User\": \"CORP\\Administrator\", \"Visited\": 0, \"Weight\": 0, \"event
type\": \"process\", \"parent_process_name\": \"explorer.exe\", \"pid\": \"3592\", \"ppid\": \"2332\", \"process_name\": \"Cmd.Exe\", \"tim
stamp\": 637183491512580000}
64 results found
eql> table process_name, parent_process_name, User, Pid
=====
process_name  parent_process_name  User                Pid
=====
Cmd.Exe       powershell.exe      NT AUTHORITY\\SYSTEM
FIND.EXE      cmd.exe              NT AUTHORITY\\SYSTEM
reg.exe       cmd.exe              NT AUTHORITY\\SYSTEM
Cmd.Exe       svchost.exe          NT AUTHORITY\\SYSTEM
FIND.EXE      cmd.exe              NT AUTHORITY\\SYSTEM
Cmd.Exe       powershell.exe      CORP\\Administrator
Cmd.Exe       svchost.exe          NT AUTHORITY\\SYSTEM
reg.exe       cmd.exe              NT AUTHORITY\\SYSTEM
Cmd.Exe       svchost.exe          NT AUTHORITY\\SYSTEM

```

EQL's table command helps prettify raw JSON output.

Perhaps, there is some suspicious activity buried in here though it's not unusual for Administrators to be mucking around with command shells.

You'd probably want to add criteria that's based on more than just running a command shell: a command shell *followed* by "whoami.exe" and "findstr.exe" should trigger alarm bells. And then if EQL finds a match, you'd want to organize all the output under user name, rather than seeing it spewed out in random order.

This leads nicely to EQL's "join" command, which I touched on in part I. It's the familiar join from the database world, but instead applied to JSON output. I can craft the following command:

```

search join by User [process where process_name == "cmd.exe"]
[process where process_name == "whoami.exe"]
[process where process_name == "findstr.exe"]

```

In effect, you're searching for *any* user who's run all three of these commands:

```
Downloads — ubuntu@ip-172-31-1-94: ~/eqlib — ssh -i andy-key.pem ubuntu@ec2-35-163-159-22.us-west-2.compute.amaz...
CORP\monty whoami.exe
CORP\monty hostname.exe
=====
6 results found
eql> search
[ ..> join by User [process where process_name == "cmd.exe"]
[ ..> [process where process_name == "whoami.exe"]
[ ..> [process where process_name == "findstr.exe"]
[ ..>
[ ..>
=====
User          process_name
=====
CORP\monty    Cmd.Exe
CORP\monty    whoami.exe
CORP\monty    FINDSTR.EXE
=====
3 results found
eql> █
```

What's going on with Mr. Burns?

Let's Sequence It!

The join commands lets me quickly see that employee Monty is a possible suspect. There may be a benign explanation for his use of the cmd shell. What would really make the case would be if this sequence of commands were run within a *short time interval*.

You can see what I'm getting at, right? Maybe Monty ran a cmd shell six months ago and then for whatever reason was searching his files using findstr, and then three months ago, he was bored, brought up a shell, and entered "whoami.exe". Strange, but isn't he a bit like you and me, as the song goes?

If the cmd shell, findstr, and whomai were all run within, say, 10 minutes of each other, then you can more confidently say that Monty's behavior is far more consistent with an account that's been hacked.

That's where EQL's "sequence" comes into play. It's essentially a join with a time parameter. So I'd run it like this:

```
search sequence by User with maxspan=10m [process where process_name == "cmd.exe"]
[process where process_name == "whoami.exe"]
[process where process_name == "findstr.exe"]
```

If I find that Monty's activities fit this EQL sequence search, then I'm about ready to say "got ya". Though I'd probably want to do more analysis.

This is a good point to end part II, and you should try some of your own searches on JSON-ized Sysmon log data. Check back next week for part III, where we'll go deeper into EQL and also explore some of the Mitre Att&ck threat scripts.

Part III: Advanced Threat Detection With EQL

With the above as background, we're now ready for more realistic (and complicated) queries. But before we delve into this, it's worth the time to take up some practical matters with EQL.

It is a very flexible language, and in fact, you can try EQL on any JSON-formatted data. It'll work fine. What makes process data special is the relationship between parent and child, which is captured in the pid and ppid fields in the Sysmon log.

In order to play along with EQL, there are few fields in the JSON-formatted file that should have the required names that you look at [here](#) in the EQL documentation under the schema definitions.

Let me bring to your attention two particular fields since it's not obvious what they map into from the Sysmon record. They are unique_pid and unique_ppid. If you look at the Sysmon log (below) you'll see the fields ProcessGuid and ParentProcessGuid.

Friendly View
 XML View

RuleName	
UtcTime	2019-12-16 14:22:28.297
ProcessGuid	{2DF17528-9324-5DF7-0000-0010884A0D00}
ProcessId	1628
Image	C:\Windows\System32\cmd.exe
FileVersion	6.3.9600.17415 (winblue_r4.141028-1500)
Description	Windows Command Processor
Product	Microsoft® Windows® Operating System
Company	Microsoft Corporation
OriginalFileName	Cmd.Exe
CommandLine	cmd.exe /Q /c cd 1 > \\127.0.0.1 \ADMIN\$_1576506144.45 2>&1
CurrentDirectory	C:\
User	CORP\lex
LogonGuid	{2DF17528-9323-5DF7-0000-0020D0440D00}
LogonId	0xd44d0
TerminalSessionId	0
IntegrityLevel	High
Hashes	SHA1=7C3D7281E1151FE4127923F4B4C3CD36
ParentProcessGuid	{2DF17528-9324-5DF7-0000-001089450D00}
ParentProcessId	4772
ParentImage	C:\Windows\System32\wbem\WmiPrvSE.exe
ParentCommandLine	C:\Windows\system32 \wbem\wmiprvse.exe -secured - Embedding

Copy
Close

ProcessGuid corresponds to EQL's unique_pid, and ParentProcessGuid to parent_unique_pid.

You guessed it, they map into the aforementioned pair of fields. And you'll have to keep in mind to use these names in your PowerShell code.

One more piece of business. As I mentioned in the previous section, for EQL's sequence, you'll need a "timestamp" field. And that is found in the "UtcTime" field in the Sysmon record. There's a little bit of PowerShell to convert this universal time into ticks, which EQL then uses to quickly calculate seconds, minutes, or hours duration:

```
$t= [datetime] $_.UtcTime  
$obj| add-Member -MemberType NoteProperty -Name timestamp -Value $t.Ticks
```

You'll need something like the above buried in the iterator that's going through the objects from the Sysmon log. Note: I wasn't completely consistent with my field names in my own experiment, so please forgive my use of Cline instead of the standard "command_line" below.

Exploring Sysmon With EQL's Descendant Search

In the previous section, we began using EQL to look for possible threats. The idea was to search for the use of cmd.exe or whoami.exe or hostname.exe in Sysmon with the knowledge that ordinary employees would likely not be entering these command and therefore this could be a sign of a threat.

After making my field names more consistent with EQL's schema, I can now leverage the "descendant of" and "child of" EQL keywords for a cooler kind of search.

What I really want to do is find all processes that are descended from a cmd.exe shell. Why do that? Because as we pointed out early, the use of a cmd shell is unusual for average workers, but very normal for malware or a hacker. With the "descendant of" EQL query, I can not only find the child processes but also anything these processes themselves launched. The whole shebang!

Playing the part of a defender. I used the following EQL to search though my Sysmon data:

```
Search process where descendant of [process where process_name == "Cmd.exe"] or process where process_name == "Cmd.exe"| sort ppid
```

And then organized my results into pretty columns with this:

```
table User,pid,ppid, Cline
```

Here's a section of the output that is particularly revealing of threats:

CORP\monty	2188	2280	Cmd.Exe	cmd.exe /Q /c cd 1> \\127.0.0.1\ADMIN\$_1584973379.51 2>&1
CORP\monty	3752	2280	Cmd.Exe	cmd.exe /Q /c cd \ 1> \\127.0.0.1\ADMIN\$_1584973379.51 2>&1
CORP\monty	2652	2280	Cmd.Exe	cmd.exe /Q /c cd 1> \\127.0.0.1\ADMIN\$_1584973379.51 2>&1
CORP\monty	2436	2280	Cmd.Exe	cmd.exe /Q /c cd .. 1> \\127.0.0.1\ADMIN\$_1584973379.51 2>&1
CORP\monty	2072	2280	Cmd.Exe	cmd.exe /Q /c cd 1> \\127.0.0.1\ADMIN\$_1584973379.51 2>&1
CORP\Administrator	2520	2472	WerFault.exe	"C:\Windows\system32\WerFault.exe" -k -lcq
NT AUTHORITY\SYSTEM	2612	2600	AWSAcpiS.exe	"C:\Program Files\Amazon\Ec2ConfigService\Scripts\AWSAcpiSpcrReader.exe"
Window Manager\DWM-3	84	2612	dwm.exe	"dwm.exe"
CORP\Administrator	724	2620	rdpclip.exe	rdpclip
NT AUTHORITY\SYSTEM	2696	2640	Cmd.Exe	C:\Windows\system32\cmd.exe /c C:\Windows\system32\reg.exe query hklm\software\micro
collectionstate /reg:64				
NT AUTHORITY\SYSTEM	2704	2648	Cmd.Exe	C:\Windows\system32\cmd.exe /c C:\Windows\system32\reg.exe query hklm\software\micro
collectionstate /reg:64				
NT AUTHORITY\SYSTEM	3180	2732	NgenTask.exe	"C:\Windows\Microsoft.NET\Framework64\v4.0.30319\NgenTask.exe" /RuntimeWide /StopEve
CORP\Administrator	4296	2808	rdpclip.exe	rdpclip
CORP\Administrator	1528	2808	rdpclip.exe	rdpclip
NT AUTHORITY\SYSTEM	2584	2876	CVTRES.EXE	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\cvtres.exe /NOLOGO /READONLY /MACHIN
" c:\Windows\Temp\q2ypkz13\CSCEE3CFB995D44D009469DE8B86E5C449.TMP"				
NT AUTHORITY\SYSTEM	2900	2876	Cmd.Exe	C:\Windows\system32\cmd.exe /c C:\Windows\system32\reg.exe query hklm\software\micro
collectionstate /reg:64				
NT AUTHORITY\SYSTEM	2908	2900	reg.exe	C:\Windows\system32\reg.exe query hklm\software\microsoft\windows\softwareinventory
NT AUTHORITY\SYSTEM	2936	2912	Cmd.Exe	C:\Windows\system32\cmd.exe /c C:\Windows\system32\reg.exe query hklm\software\micro
collectionstate /reg:64				
CORP\Administrator	1596	2964	rdpclip.exe	rdpclip
NT AUTHORITY\SYSTEM	3004	2980	Cmd.Exe	C:\Windows\system32\cmd.exe /c C:\Windows\system32\reg.exe query hklm\software\micro
collectionstate /reg:64				
NT AUTHORITY\SYSTEM	3284	3368	NgenTask.exe	"C:\Windows\Microsoft.NET\Framework64\v4.0.30319\NgenTask.exe" /RuntimeWide /StopEve
NT AUTHORITY\SYSTEM	3584	3400	Cmd.Exe	cmd.exe
NT AUTHORITY\SYSTEM	2432	3584	whoami.exe	whoami
NT AUTHORITY\SYSTEM	2424	3584	FIND.EXE	find password *
NT AUTHORITY\SYSTEM	2316	3584	hostname.exe	hostname
NT AUTHORITY\SYSTEM	3200	3584	FIND.EXE	find
NT AUTHORITY\SYSTEM	1960	3584	FIND.EXE	find "password" *
NT AUTHORITY\SYSTEM	1760	3584	whoami.exe	whoami

The process events in the red boxes require further analysis, ASAP!

The output redirections for user Monty are suspicious, as well as the cluster of whoami.exe, find.exe, and hostname.exe commands for the System user.

As the defender, I want to dig a little deeper and look at a process with pid of 3584, the parent process and then perhaps its parent, to learn more about the second group of incriminating commands:

```

eql>
eql> search process where pid == '3584'
=====
User          pid  ppid  process_name  Cline
=====
NT AUTHORITY\SYSTEM  3584  3400  Cmd.Exe       cmd.exe

1 result found
eql>
eql> search process where pid == '3400'
=====
User          pid  ppid  process_name  Cline
=====
NT AUTHORITY\SYSTEM  3400  668  FHlpDBWj.exe  C:\Windows\FHlpDBWj.exe

1 result found
eql>

```

It all leads back to a cryptically named process in the Windows folder. #gotcha

You'll notice in the above screenshot of my EQL session that a strangely named process in the C:\Windows started the whole chain. Those of you've been reading the [blog](#) know that this is a sign of a remote [psexec](#).

One more piece of confirming evidence is to see if the whoami.exe and find.exe processes have been run in a very short period of time, say 5 minutes. If so, that would really make the case for a threat. From the previous section we know about the EQL join's maxspan option. You can see the results of my query below:

```

eql> search
..> sequence by ppid with maxspan=15m [process where process_name == "find.exe"]
..> [process where process_name == "whoami.exe"]
..>
..>
=====
User                pid  ppid  process_name  Cline
=====
NT AUTHORITY\SYSTEM 1960 3584  FIND.EXE      find "password" *
NT AUTHORITY\SYSTEM 1760 3584  whoami.exe    whoami
=====
2 results found
eql>

```

EQL's maxspan option matches the search criteria when it all falls within a time interval. And that's useful for spotting behaviors!

EQL's Pre-Cooked Mitre Att&ack Queries

The far larger point in the above EQL exploration is that we're looking for *behaviors*. I'm not searching for a specific malware name or scanning for some plain text keywords. When the attackers are living of the land this kind of approach is not very successful. As I've been trying to show, searching for the underlying activities, which can't be hidden because they're captured in the Sysmon log, is a far more productive method.

However, you or I can't possibly know the underlying activities and behaviors that go along with all the threats out there! Thankfully, the Mitre folks have their Att&ack Matrix, and the EQL team has worked out queries that correspond with their classifications. And you can find all this delicious information here.

I tried their WMI Execution with Command Line Redirection, but suitably modified because I didn't have all the Sysmon log information that's required—file audit data 'cause we know that's a CPU killer, and also spews out too many log entries for my modest experiments.

You can see the results of my modified WMI threat hunting query below:

```

Downloads — ubuntu@ip-172-31-1-94: ~/eqllib — ssh -i andy-key.pem ubuntu@ec2-35-163-159-22.us-west-2.compute.amazonaws.com — 133x22
eql>
eql> input my-log.json
Using file my-log.json with 1001 events
eql> table pid,process_name, parent_process_name, Cline
eql> search
..> process where process_name == "cmd.exe" and Cline == "*>*" and parent_process_name == "WmiPrvSe.exe"
..>
..>
=====
pid  process_name  parent_process_name  Cline
=====
664  Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c hostname 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
3076 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd documents 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
3632 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd documents 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
2720 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd .. 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
2836 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd administrator 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
4012 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c dir 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
3620 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
1292 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd documents 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
4028 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd .. 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
4076 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd documents 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1
2992 Cmd.Exe       WmiPrvSe.exe        cmd.exe /Q /c cd 1> \\127.0.0.1\ADMIN$\__1584973379.51 2>&1

```

Borrowed EQL's query for spotting a WMI threat. No surprises here since I used Inpacket's wmiexec in previous experiments. Good work EQL!

In the next section, we'll look at more of the Att&ack framework, take care of a few loose ends, and then conclude with a big picture view of threat hunting. Check back again later next week!