



Security: Solution Optimization for Modern Security Operations – Advanced Hunting

Scenarios and Usage Guide

Table of Contents

1	Introduction	2
2	Prerequisites	3
3	Observe APT29 data and brush up on the basics!.....	4
3.1.1	Where's the data?	4
3.1.2	Observing table structure	4
3.1.3	Query logs - limiting the output	4
3.1.4	Parameterize your query	6
3.1.5	Counting the total number of rows	7
3.1.6	Aggregation	7
3.1.7	Visualizing the data	8
4	Exercises	10
4.1	Filtering.....	10
4.1.1	Simple filtering.....	10
4.1.2	Using different predicates with 'where'	10
4.1.3	Using lookups with 'where'	11
4.2	Analyzing	12
4.2.1	Summarize by.....	12
4.2.2	Checking connections	12
4.2.3	Most recent services	13
4.3	Presenting.....	14
4.3.1	Charting logon intensity	14
4.3.2	What are the events?	14
4.4	Advanced KQL.....	15
4.4.1	Finding rare processes	15
4.5	Optimizing.....	16
5	APT29 incident investigation using advanced KQL	17

5.1	Attack Technique: User Execution	17
5.1.1	Using <i>parse</i> with regex, string functions and operators for building fields and filtering	17
5.1.2	Removing unnecessary fields and adding useful fields to the results	20
5.1.3	Finding top processes using top operator	21
5.1.4	Using render to visualize data	21
5.2	Attack Technique: Masquerading.....	23
5.2.1	Detecting the RTLO technique.....	23
5.3	Attack Technique: Uncommonly used ports	23
5.3.1	Checking the network destination with concatenating the values	23
5.4	Attack Technique: Command-line interface.....	24
5.4.1	Finding cmd.exe processes and their parent processes	24
5.5	Attack Technique: File and Directory Discovery.....	25
5.5.1	Joining several tables.....	25
5.6	Attack Techniques: Persistence, Exfiltration.....	27
5.6.1	Detecting PowerShell web connectivity with has_any	27
5.7	Attack Techniques: Elevation of Privileges.....	28
5.7.1	Using conditions in queries	28
5.8	Combining KQL with watchlists.....	29
5.8.1	Creating and editing a watchlist.....	29
5.8.2	Query the watchlist	32
5.8.3	Correlate logs data with business data	33
6	Microsoft Sentinel KQL Workbooks.....	35
6.1	Initializing workbooks	35
6.2	Working with Intro to KQL workbook.....	38
6.3	Working with Advanced KQL for Microsoft Sentinel workbook.....	40
7	Answers to the exercises	42
7.1	Filtering.....	42
7.1.1	Simple filtering.....	42

7.1.2	Using different predicates with 'where'	42
7.1.3	Using lookups with 'where'	42
7.2	Analyzing	43
7.2.1	Summarize by	43
7.2.2	Checking connections	43
7.2.3	Most recent services	43
7.3	Presenting	44
7.3.1	Charting logons intensity	44
7.3.2	What are the events?	44
7.4	Advanced KQL	45
7.4.1	Finding rare processes	45
7.5	Optimizing	46

1 Introduction

The scope of this tactical scenario is to build familiarity with KQL skills needed for advanced investigation, using the example of an APT29-style attack, following the guideline provided by MITRE ATT&CK Framework.

2 Prerequisites

All pre-requisites should have been prepared before the delivery started.

The Customer Commitment document containing setup details has been sent to the Customer immediately after the scoping call finished.

Note: for Advanced KQL, it's suggested that the setup script is run no more than 14 days before the engagement commences, so that the data is still within the "hot" lookup window.

3 Observe APT29 data and brush up on the basics!

3.1.1 Where's the data?

Run the following query:

```
union withsource=table *  
| summarize count() by table  
| sort by count_ desc
```

Analyze the output. Which table has the most entries?

3.1.2 Observing table structure

Run the following query:

```
fWindowsEvent  
| getschema
```

Note the data type for eventData column. What is it?

3.1.3 Query logs - limiting the output

Let's first define the timeframe for our first query.

Click on **the Time range** button and select Custom.

Microsoft Sentinel | Logs

Selected workspace: 'cybersoc'

The screenshot shows the Microsoft Sentinel Logs interface. At the top, there's a navigation bar with 'Microsoft Sentinel | Logs' and a 'Selected workspace: 'cybersoc'' indicator. Below this, there's a toolbar with buttons for 'UserExec...', 'New Que...', 'Uncomm...', and 'Run'. The 'Time range' dropdown is open, displaying a list of time intervals: 'Last 30 minutes', 'Last hour', 'Last 4 hours', 'Last 12 hours', 'Last 24 hours' (which is selected with a checkmark), 'Last 48 hours', 'Last 3 days', 'Last 7 days', 'Set in query', 'Custom' (highlighted by a mouse cursor), and 'Time range syntax'. On the left side, there's a sidebar with 'Tables', 'Queries', and 'Functions' tabs. Under 'Queries', there's a search bar and a 'Filter' button. Below that, there's a 'Collapse all' button and a list of queries: 'Graph of Running and Idle Node instances', 'Latest OnlineEndpoint Console Logs', 'List callers and their associated action in last 48 hours', and 'Plot compute cluster utilization'.

Set the timeframe as the last month up to the current day for now.

For our first query we will use the following:

```
fWindowsEvent
| where tolower(Channel)=='security'
| take 10
```

Note that we're limiting the output by specifying the number of results for the **take** operator.

Analyze the output. Most probably you'll recognize that we have Windows Security log entries.

Now change the query to the following.

```
fWindowsEvent
| where Channel == 'Microsoft-Windows-PowerShell/Operational'
| limit 20
```

Limit and take operators are essentially doing the same thing – returning the number of rows you specified in the query. Important notes about **limit** and **take**:

- Limit and take are mostly useful for tests:
- No sorting
- The results are randomly selected
 - It's not the same as **top**
- Default output limit is 30,000 records if not otherwise limited

3.1.4 Parameterize your query

Let's improve our query to avoid setting the timeframe in the GUI:

```
let start_time = datetime('2023-11-19');
let end_time = start_time + 3d;
fWindowsEvent
| where Channel == 'Microsoft-Windows-PowerShell/Operational'
and TimeGenerated > start_time and TimeGenerated < end_time
| take 10
```

As you can see, now you're defining the timeframe right in the query using **let** statements to define *variables*. Adjust **start_time** and **end_time** to correspond to the timeframe where you performed the data upload (remember dates are stored in UTC so you may need to adjust the date accordingly).

Let allows you to simply create variables you can use throughout the rest of the query. Using **let** and parameterizing your query is considered best practice, as it allows developing better-performing queries and encourages code reuse.

Note that any **let** statements must be run with the code referencing the variables every time – there isn't any persistent memory across runs.

3.1.5 Counting the total number of rows

Sometimes it's important to know the amount of data of a specific type, to be able to efficiently plan queries or limit results.

Let's check how many events we have coming from PowerShell Operational logs:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Microsoft-Windows-PowerShell/Operational'
| count
```

3.1.6 Aggregation

KQL has a **summarize** operator that aggregates the content of the input table.

Run the following query and analyze the results:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 4624 // Successfull logon
| summarize count() by Computer //Shows the number of successful logons per computer
```

3.1.7 Visualizing the data

KQL has a built-in **render** operator that tells the query engine that you want to show query results in a graphical way – for example, as an area chart, or bar chart.

Let's run the following query:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Microsoft-Windows-PowerShell/Operational'
| summarize count() by bin(TimeGenerated, 5m)
| render barchart
```

As you can see, the query summarizes using the **bin** operator to group the data into 5-minute bins (buckets), then display it as a bar chart.

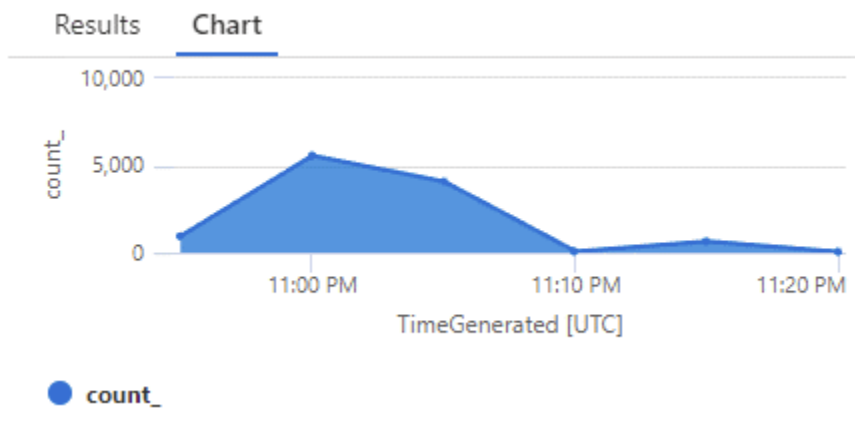
Note

The *bin* operator rounds data down to an integer multiple of a given bin size. It's frequently used with *summarize by*.

Now change the chart type to **areachart**.

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Microsoft-Windows-PowerShell/Operational'
| summarize count() by bin(TimeGenerated, 5m)
| render areachart
```

You should see something like the following as a result:



You can experiment with other visualizations if you want!

Visualizing data allows for quicker understanding of the data (like spikes of network activity to specific IP addresses or URLs).

4 Exercises

4.1 Filtering

4.1.1 Simple filtering

Run the following query (adjust timeframe before you run it):

```
fWindowsEvent  
| where Channel == 'Security'  
| where EventID == 4688
```

You've just found all events with ID 4688 (Process started)!

- Now adjust the query so you can list the events related to the **sdclt.exe** process starting.

The APT29 actor was found to be actively leveraging User Account Control bypass techniques, one of which is related to using the system utility sdclt.exe to run elevated processes.

When you're done, compare your resulting query with the answers at the end of the guide.

4.1.2 Using different predicates with 'where'

You're trying to find out whether Peter Beesly has logged on to any of the machines. You've built the following query:

```
fWindowsEvent  
| where Channel == 'Security'  
| where EventID == 4624  
| where EventData.TargetUserName == 'PBEESLY'
```

The query returned nothing... but you're not sure about the case of Peter's account name?

- Adjust the query so it works regardless of the case of the account name.
 - Avoid converting the username to upper/lower-case.

When you're done, you can compare your query with the answers at the end of the guide.

4.1.3 Using lookups with 'where'

Based on the attacker's profile, you're interested in file share access events, particularly in events that indicate access to two files through a file share: **psexesvc.exe** and **python.exe**.

You used the [Windows security auditing and monitoring reference](#) document to figure out the Event ID of the events created by this activity: 5145.

Now you want to query *fWindowsEvent* to get the 5145 events related to **psexesvc.exe** and **python.exe**.

You come up with this base query:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 5145
```

- How can you improve this query so that it returns only events related to the 2 executable files mentioned above?

When you're done, compare your resulting query with the answers at the end of the guide.

4.2 Analyzing

4.2.1 Summarize by

You decide to check what Windows file sharing usage is reported by the hosts. You've prepared the query below:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 5145
| where EventData.ShareName contains 'IPC'
| extend RelativeTargetName = tostring(EventData.RelativeTargetName)
| summarize count() by RelativeTargetName
```

You get some useful information from this, but now you want to

- refine your query to show only processes related to PSEXESVC,
- group the results by Computer and RelativeTargetName

When you finish, check your query with the answers at the end of the guide.

4.2.2 Checking connections

You want to examine what processes have been creating connections from the hosts.

Your query so far is as follows:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 5156
| extend App = tostring(EventData.Application)
| extend IP = tostring(EventData.DestAddress)
```


You've spotted two interesting-looking processes in the results – one is something like '**psexec**', and another one contains a substring like '**3aka3**' in the name.

Now you want to:

- See what connections were initiated by these 2 processes
 - Summarized by Computer and App names
- You would also prefer to have the IP address accompanied by a port in a single field in the results (e.g., *IP:Port* would be good!)
 - You would probably add a column and concatenate strings to do that...

When you're done, compare your resulting query with the answers at the end of the guide.

4.2.3 Most recent services

You decide to investigate the most recent services installed on each of the machines you've connected to the Microsoft Sentinel workspace.

Based on the [Windows security auditing and monitoring reference](#), when a new service is created an Event 4697 is generated.

- Build a query that will return the most recent service installation events for every computer you connected to Sentinel.
- Use *arg_max()* with *summarize by*.
- Note there are multiple approaches possible – you could summarize each service on each computer by installation date, or simply the latest service on each computer...

When you're done, compare your resulting query with the answers at the end of the guide.

4.3 Presenting

4.3.1 Charting logon intensity

You decided to check successful logon intensity on your machines.

- Visualize logon intensity (logons over time) as a time chart.
 - You remember that a successful logon causes Event ID 4624
- You don't want to include SYSTEM account activity
- You're not interested in Computer account logons
 - (those accounts ending with "\$" symbol)
- You want to use 1-minute intervals for your chart

How would you combine the query to get the time chart described above? Compare your resulting query with the answer at the end of the guide.

4.3.2 What are the events?

For cost optimization purposes, you want to:

- List of the top 10 Event IDs, by count
- You want to visualize these results as a pie chart

What should the resulting query look like?

You can compare your query with the answer at the end of the guide.

4.4 Advanced KQL

4.4.1 Finding rare processes

Run the following query:

```
let Sensitivity = 15;
fWindowsEvent
| where EventID == 4688
| extend ProcArray = split(EventData.NewProcessName, '\\')
// ProcArrayLength is Folder Depth
| extend ProcArrayLength = array_length(ProcArray)
| extend LastIndex = ProcArrayLength - 1
| extend Proc = ProcArray[LastIndex]
// ProcArray[0] is the proc's Drive
| extend DriveDepthProc = strcat(ProcArray[0], '-', ProcArrayLength, '-', Proc)
| summarize StartTimeUtc = min(TimeGenerated), EndTimeUtc = max(TimeGenerated), TimesSeen = count(), HostCount = dcount(Computer), Hosts = make_set(Computer), UserCount = dcount(tostring(EventData.SubjectUserName)), Users = make_set(tostring(EventData.SubjectUserName)) by DriveDepthProc
| where TimesSeen < Sensitivity
| extend timestamp = StartTimeUtc
```

You initially want to filter out three processes (and maybe more later).

- SearchProtocolHost.exe
- SearchFilterHost.exe
- MsMpEng.exe

Use **let** to define a *datatable* or a *dynamic* type, and filter out these three processes from the result.

You can compare your query with the answer at the end of the guide.

4.5 Optimizing

Run the query below:

```
fWindowsEvent
| extend ProcessId = toint(EventData.ProcessId)
| extend ParentProcessGuid = tostring(EventData.ParentProcessGuid)
| extend Image = tostring(EventData.Image)
| where Channel == "Microsoft-Windows-Sysmon/Operational"
| where EventID ==1
| join kind=innerunique (
fWindowsEvent
| extend ParentImage=tostring(EventData.ParentImage)
| extend ProcessGuid = tostring(EventData.ProcessGuid)
| where EventData.ParentImage endswith "services.exe"
) on $left.ParentProcessGuid == $right.ProcessGuid
|where Image endswith 'python.exe'
| distinct TimeGenerated, Image, ParentImage, Computer
```

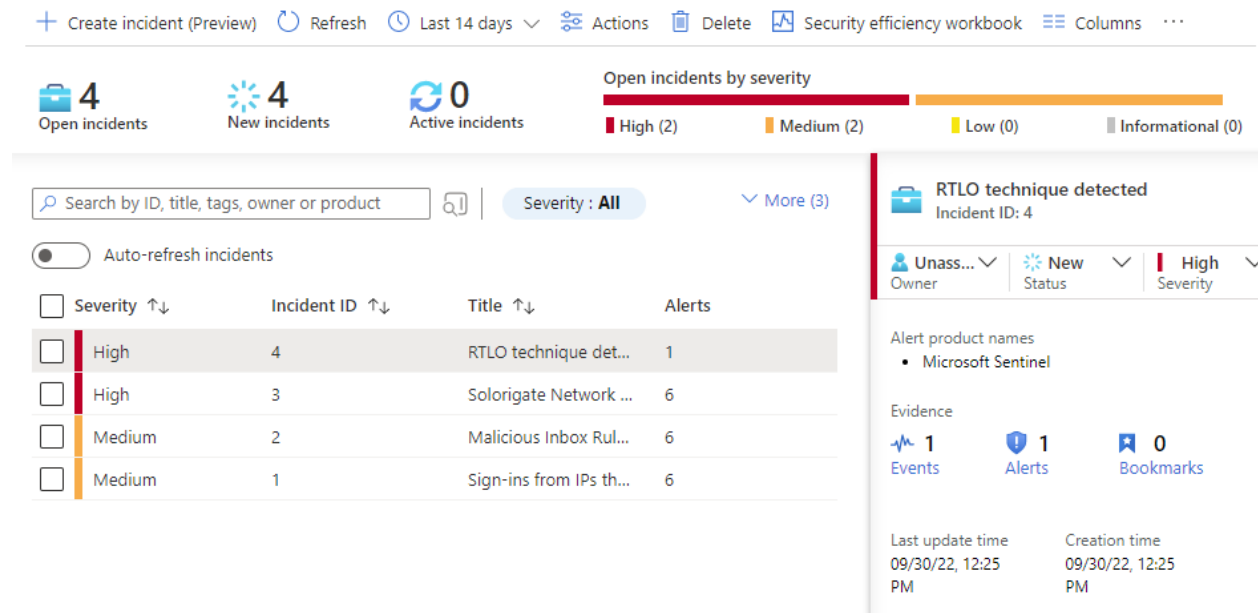
While this query should return some results and might work quickly (especially if you run it twice or more and have a very small volume of data) – it is far from being efficient!

- Try to improve the efficiency of the query

When you've finished you can compare your improved query version with the answer at the end of the guide.

5 APT29 incident investigation using advanced KQL

Shortly after you complete the pre-requisites deployment, you should see a new incident in Microsoft Sentinel Incidents blade.



You will use this incident as a starting point to complete the investigation and find artifacts in Sentinel logs using KQL.

Go section by section.

You will be working with Logs blade of Microsoft Sentinel. Go to Microsoft Sentinel and switch to the Logs blade for the start.

We will be going through all the queries one by one, investigating the details of the incident while learning the capabilities of KQL.

5.1 Attack Technique: User Execution

5.1.1 Using *parse* with regex, string functions and operators for building fields and filtering

Our initial incident referred to the execution of a malicious application, `cod.3aka3.scr`. Let's find out when the application was started and what machines were targeted.

The Security logs we're working with contain a dynamic field *EventData*. The Message field within the EventData JSON is just multiline text, and it contains important data we'd like to query.

Run this simple query and analyze the data structure and data by expanding the single row returned:

```
let start_time = datetime('2022-04-17');
let end_time = start_time + 3d;
fWindowsEvent
| where TimeGenerated > start_time and TimeGenerated<end_time
| where EventID == 4688 // Process started
| take 1
```

MandatoryLabel	S-1-16-8192
	A new process has been created.
	Creator Subject: Security ID: S-1-5-21-1830255721-3727074217-2423397540-1107 Account Name: pbeesly Account Domain: DMEVALS Logon ID: 0x3731F3
	Target Subject: Security ID: S-1-0-0 Account Name: - Account Domain: - Logon ID: 0x0
Message	Process Information: New Process ID: 0x214c New Process Name: C:\ProgramData\victim\cod.3aka3.scr Token Elevation Type: %%1938 Mandatory Label: S-1-16-8192 Creator Process ID: 0x1158 Creator Process Name: C:\Windows\explorer.exe Process Command Line: "C:\ProgramData\victim\cod.3aka3.scr" /S Token Elevation Type indicates the type of token that was assigned to the new process in accordance with User Account Control policy.

In this case, most of the fields from the Message are available as separate fields in the EventData.

But even if they weren't, KQL provides a **parse** operator that can help us to easily extract values from multiline text.

Run the following query:

```
let start_time = datetime('2023-11-19');
let end_time = start_time + 3d;
fWindowsEvent
| where TimeGenerated < end_time and TimeGenerated > start_time
| where tolower(Channel)=='security'
| parse kind = regex EventData.Message with * '\tCreator Process
Name:\t'CreatorProcessName'\r\n' *
| parse kind = regex EventData.Message with * '\tNew Process
Name:\t'NewProcessName'\r\n' *
| where CreatorProcessName contains 'ExplOreR.EXE' and NewProcessName
endswith '.scr'
```

Pay attention to the lines with *parse* operator.

What they are trying to do is extract 2 matches in *EventData.Message* of every event, where the line contains "Creator Process Name" or "New Process Name" with a corresponding value after ":". Note that the query also incorporates the tab symbol ("\t") and new line symbol ("\n") to provide more precise matching.

The result of work of the parse operator (given it was successful) will be stored in new fields – **CreatorProcessName** and **NewProcessName**. These 2 fields will be added to the result set.

Since we're interested in specific results that would allow us to see whether any process with "3aka3" substring has been started from "explorer.exe", we're going to filter on these two criteria. We can use the **contains** operator for this.

Note:

contains and **endswith** operators are *case-insensitive*, so we might go creative in writing explorer.exe process name.

There're case-sensitive variants of these operators – **contains_cs** and **endswith_cs** that generally works faster than their case-insensitive prototypes

Run the query and analyze the output. You should see a single event when explorer.exe started a new process with quite an unusual name. Note that the **CreatorProcessName** and **NewProcessName** fields have been added to the resulting columns.

Now let's simplify the query.

Add filtering by Event ID 4688 (Process Created) in an appropriate place (recall the query optimization guidance) and replace all the *regex* statements with `EventData.<Field>` (as the `EventData` fields were helpfully extracted for us ahead of time).

5.1.2 Removing unnecessary fields and adding useful fields to the results

In the results of the previous query, we had several useful fields displayed, but we also have some fields we didn't need. Let's change the query a bit to add useful information and remove noise.

Note:

Sometimes the additional fields will have useful context (and might allow entity mapping or other insights by the analyst), so particularly when writing analytics rule detection code, it's important not to over-simplify the output.

Hiding work-product fields (those created as a side-effect while producing output) is typically a good practice.

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where tolower(Channel)=='security'
| where EventID==4688
| extend CreatorProcessName = tostring(EventData.ParentProcessName)
| extend NewProcessName = tostring(EventData.NewProcessName)
| where CreatorProcessName contains 'ExploreR.EXE' and NewProcessName
contains '3aka3'
| extend NewProcessId = EventData.NewProcessId
| project-away EventData, Provider, Channel, Task, Type, TenantId
```

Here we added some fields with **extend** operator and removed unnecessary fields with **project-away** operator.

5.1.3 Finding top processes using top operator

Let's run the following query:

```
let start_time = ago(14d);
let RunProcesses = fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 4688
| extend CreatorProcessName = tostring(EventData.ParentProcessName)
| extend NewProcessName = tostring(EventData.NewProcessName);
// Find the 20 processes that were run the most
RunProcesses
| summarize count() by NewProcessName
| top 20 by count_
```

Here we use the **top** operator in combination with **summarize count() by** to make an aggregation of the number of processes started, displaying the top 20.

5.1.4 Using render to visualize data

Run the following query, which is based on the previous one.

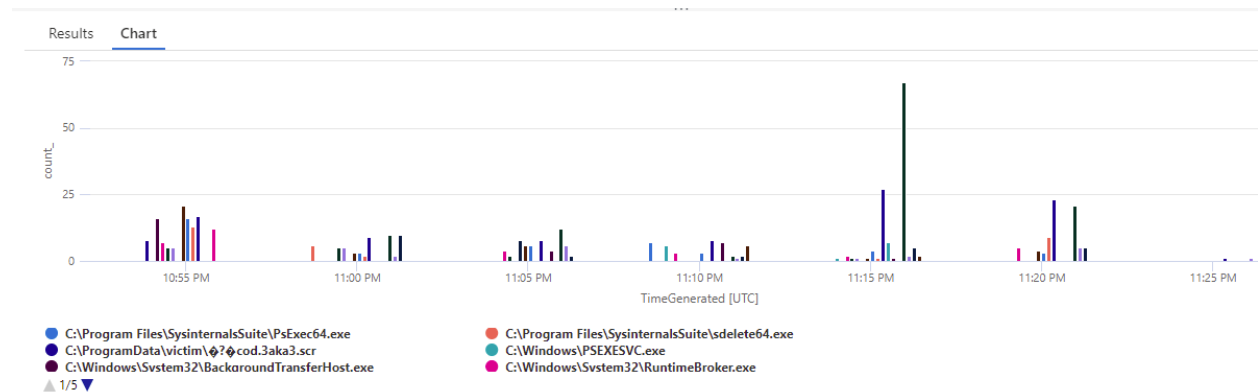
```
let start_time = ago(14d);
let RunProcesses = fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 4688
| extend CreatorProcessName = tostring(EventData.ParentProcessName)
| extend NewProcessName = tostring(EventData.NewProcessName);
let Top20Processes =
RunProcesses
| summarize count() by NewProcessName
| top 20 by count_;
// Create a time chart of these 20 processes - hour by hour
```

```
RunProcesses
```

```
| where NewProcessName in (Top20Processes)
| summarize count_() by bin (TimeGenerated, 5m), NewProcessName
| render columnchart kind = unstacked
```

Note that we defined 2 datatable variables using let statements – *RunProcesses* and *Top20Processes*, building one on the results from the other!

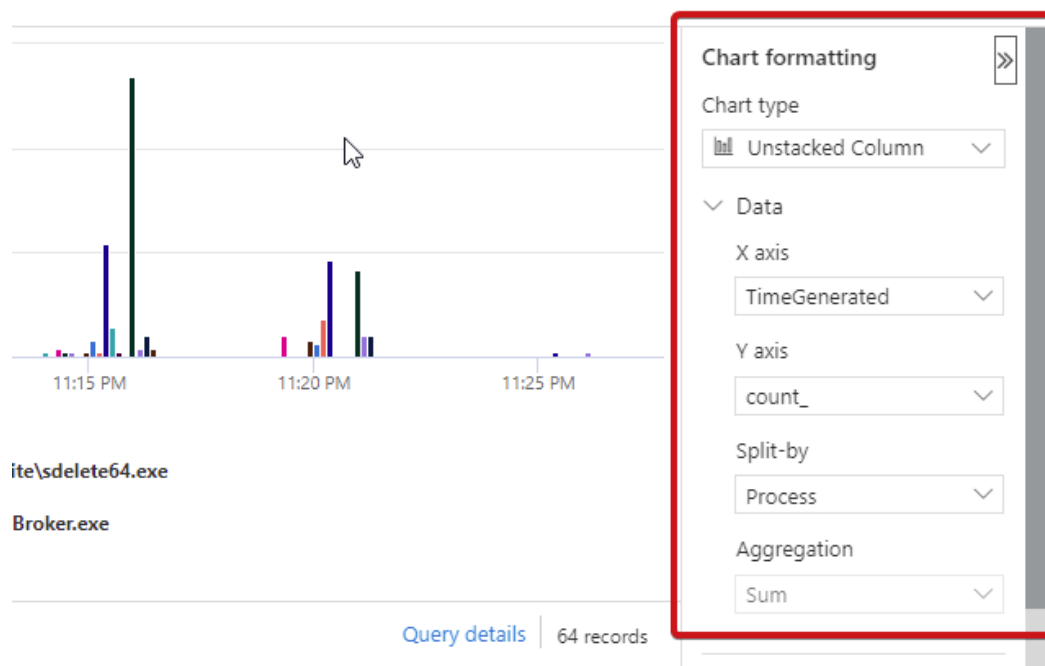
After running the query, you should get results like the picture below:



- Identify the time interval when python.exe was run.

You can click on the processes names in the legend to remove the processes you're not very interested in from the chart.

Now open the chart formatting panel from the right and experiment with the settings.



5.2 Attack Technique: Masquerading

5.2.1 Detecting the RTLO technique

Run the query below:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 4688
| extend NewProcessName = tostring(EventData.NewProcessName)
| where NewProcessName contains '@'
```

Here we're simply doing a search by a character we've seen in the "3aka3.scr" file. This file uses the Right-To-Left-Override (RTLO) technique to trick a user into thinking they're opening a .doc file (cod. read right-to-left) while they're really opening a .scr file (a screensaver executable, which is, essentially, a renamed .exe).

You can read more on RTLO [here](#) if you're interested.

5.3 Attack Technique: Uncommonly used ports

5.3.1 Checking the network destination with concatenating the values

Run the following query:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 5156
| extend AppName = tostring (EventData.Application)
| where AppName contains '3aka3'
| extend DestAddress = tostring (EventData.DestAddress)
| extend DestPort = tostring (EventData.DestPort)
| extend Destination = strcat(DestAddress,':',DestPort)
| project TimeGenerated, Computer, AppName, Destination
```

Here we're using the string concatenation function **strcat** to build a more appealing Destination field which contains the IP address accompanied with the port (1234:80)

5.4 Attack Technique: Command-line interface

5.4.1 Finding cmd.exe processes and their parent processes

Run the following query:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 4688
| extend NewProcessName = tostring(EventData.NewProcessName)
| where NewProcessName endswith '\\cmd.exe'
| extend CreatorProcessName = tostring(EventData.ParentProcessName)
| project TimeGenerated, Computer, CreatorProcessName, NewProcessName
```

Note that we added the escaped “\” symbol in cmd.exe path to ensure we get cmd.exe and not dsregcmd.exe-related rows.

Tip: we could have also used the @ symbol – i.e. **endswith @'\\cmd.exe'** - to disable escaping for that string.

5.5 Attack Technique: File and Directory Discovery

5.5.1 Joining several tables

The query below is targeted to find the instances of PowerShell initiated from our well-known "3aka3" executable and executing cmdlets containing "childitem" substring for files and directories discovery.

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == "Microsoft-Windows-PowerShell/Operational"
| where EventID == 4104
| where EventData.Message contains "childitem"
| extend ExecutionProcessId=toint(EventData.ExecutionProcessId)
| extend ScriptBlock = tostring(EventData.ScriptBlockText)
| join kind=innerunique (
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == "Microsoft-Windows-Sysmon/Operational"
| where EventID ==1
| extend ProcessId = toint(EventData.ProcessId)
| extend ParentProcessId = toint(EventData.ParentProcessId)
) on $left.ExecutionProcessId ==$right.ProcessId
| join kind=innerunique (
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == "Microsoft-Windows-Sysmon/Operational"
| where EventID ==1
| where EventData.ParentImage contains "3aka3"
| extend CreatorProcess=tostring(EventData.ParentImage)
| extend ProcessId = toint(EventData.ProcessId)
) on $left.ParentProcessId == $right.ProcessId
| distinct Computer, ScriptBlock, CreatorProcess
```

Review the results.

Run Time range: Custom Save Share + New alert rule Export P

```
10 ) on $left.ExecutionProcessId == $right.ProcessId
11 | join kind=innerunique ... (
12 WindowsEvent
13 | where Channel == "Microsoft-Windows-Sysmon/Operational" and EventID == 1 and EventData.Par
14 | extend CreatorProcess=tostring(EventData.ParentImage)
15 | extend ProcessId = toint(EventData.ProcessId)
16 ) on $left.ParentProcessId == $right.ProcessId
17 | distinct Computer, ScriptBlock, CreatorProcess
```

Results Chart Add bookmark

<input type="checkbox"/> Computer	ScriptBlock	CreatorProcess
<input type="checkbox"/> SCRANTON.dmevals.local	\$env:APPDATA;\$files=ChildItem	C:\ProgramData\victim\cod.3aka3.scr
Computer	SCRANTON.dmevals.local	
ScriptBlock	\$env:APPDATA;\$files=ChildItem -Path \$env:USERPROFILE\ -Include *.doc,*.xps,*.xls,*.ppt,*.pps,*.wps,*.wp *.jpg,*.txt,*.lnk -Recurse -ErrorAction SilentlyContinue Select -ExpandProperty FullName; Compress-Ar	
CreatorProcess	C:\ProgramData\victim\cod.3aka3.scr	

Several things to note about this query:

1. First, it's doing a *join* of PowerShell-related events (containing "childitem" substring) on Sysmon's "Process started" events. The result of this join is identifying the Parent ProcessId of the PowerShell.exe that was executing "childitem" cmdlet.
2. The second step is joining on the gathered ParentProcessId to the same Sysmon data table, but in this case with filtering on ParentImage name. ("3aka3.scr")
3. There are multiple types of joins. We're using *kind=innerunique* to ensure that even if we duplicated some log entries (for example by running our pre-requisites import script a couple of times) we won't get multiple duplicated entries.
Read more about different [join flavors](#).
4. Very often the joining is performed between different tables (like SysLog and SecurityEvent), but joining the table "on" itself can also be handy in many cases.
5. The last operator **distinct** ensures that only unique rows will be returned.

But wait! It seems we forgot to add two fields to the resulting set: *TimeGenerated* and *AccountName*. Try to add these 2 columns yourself.

You end up with something like this:

Results	Chart	Add bookmark			
<input type="checkbox"/>	TimeGenerated [UTC]	AccountName	ScriptBlock	CreatorProcess	Computer
<input type="checkbox"/>	> 4/18/2022, 10:56:17.000 PM	pbeesly	\$env:APPDATA;\$files=ChildItem...	C:\ProgramData\victim\??.co...	SCRANTON.dmevals.local

5.6 Attack Techniques: Persistence, Exfiltration

5.6.1 Detecting PowerShell web connectivity with `has_any`

Run the following query:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where EventData.ScriptBlockText has_any("WebClient",
    "DownloadFile",
    "DownloadData",
    "DownloadString",
    "WebRequest",
    "Shellcode",
    "http",
    "https")
| project TimeGenerated, Computer, EventData.ScriptBlockText
```

With **has_any** we're quickly testing `EventData.ScriptBlockText` against some of words that might be related to PowerShell cmdlets communicating with external URLs/IP addresses.

Note that this is a very broad query and will pick up many legitimate script blocks as well.

5.7 Attack Techniques: Elevation of Privileges

5.7.1 Using conditions in queries

Run the following query:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where EventID==4688
| where Channel == "Security"
| where EventData.ParentProcessName contains "sdclt.exe"
| extend TokenTypeRaw=tostring(EventData.TokenElevationType)
| extend TokenType = iff(TokenTypeRaw == "%1937","Elevated",iff(TokenTypeRaw
== "%1936","Full","Normal"))
| project TimeGenerated, Computer, TokenType, EventData.Message
```

Nothing new here, except for *iff*-related statement. We use the **iff** function to build the value of *TokenType* so it looks prettier than a numeric code. This demonstrates a single-layer nested **iff** statement.

But in cases where we have more possible values it might be more convenient to use **case** instead of **iff**.

- Change our query so it uses **case** instead of **iff**.

You can see how **case** might be easier for future query use/updates.

5.8 Combining KQL with watchlists

5.8.1 Creating and editing a watchlist

You did some KQL-based investigation against prepared data in this lab.

In real cases, sometimes you would like to correlate log data with arbitrary data like some ad-hoc extractions from an HR system, an accounting system, or another enrichment source. How can KQL tackle this problem?

One of the possible ways is to employ Microsoft Sentinel *watchlists* and use KQL to correlate data from those watchlists with log data. Watchlists can be easily updated by SOC staff or applications as circumstances change.

First, let's prepare an extraction of data from a business system - let's say, an HR system.

1. Create a new csv file with this content:

```
accountname,ismanager
ashum,0
christiek,1
andyk,0
jsnow,1
pbeesly,1
trevorg,0
```

As you can see there are two fields in this csv file:

- accountname – the logon name of an employee.
- ismanager – a mark whether a specific user has managerial job position.

2. Save the file with the name **Employees.csv**.
3. Logon to Microsoft Sentinel and switch to the Watchlists blade.
4. Click the **Add New** button and start creating a new watchlist.
5. Provide basic information – Name and Alias – as **Employees** for both fields.

×

Watchlist wizard

...

Create new watchlist

edback

list ↗

atchlist selecte
tchlist for more

General

Source

Review and create

Name *

Employees ✓

Description

Alias *

Employees ✓

Next: Source >

6. Click **Next**.

7. Upload the CSV file you created.

Ensure there's no error and you can see the file preview section with both columns displayed correctly for all each accountname you have.

General

Source

Review and create

Source type

Local file

File type

CSV file with a header (.csv)

Number of lines before row with headings *

0

Upload file *

employees.csv

Drag and drop the files or Browse for files

SearchKey *

accountname

The SearchKey is used to optimize query performance when using watchlists for joins with other data. For example, enable a column with IP addresses to be the designated SearchKey field, then use this field to join in other event tables by IP address. [Learn more and get examples about SearchKey](#)

Reset

File preview | First 50 rows and first 5 columns

accountname	ismanager
ashum	0
christiek	1
andyk	0
jsnow	1
pbeesly	1
trevorg	0

Previous

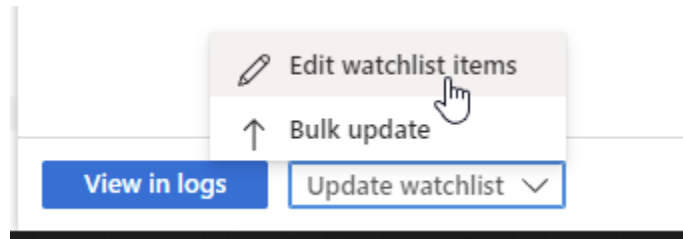
Next: Review and create >

8. Select SearchKey as accountname and click **Next: Review and Create**, then **Create** to finish the creation of the watchlist.

It should be quickly added and listed in the Portal.

You created the initial watchlist, but – oops! - we've forgotten to add some additional managers that are not in the HR system. Let's add them manually.

1. Select the watchlist you've just created and click on Edit watchlist items in the Update watchlist button popup menu.



2. Add a couple more managers to the list, like in the screenshot.

Edit watchlist items

Employees | SearchKey field: accountname

Refresh + Add new Save Delete Columns

<input type="checkbox"/> accountname	ismanager
<input type="checkbox"/> scooper	1
<input type="checkbox"/> bkripke	1
<input type="checkbox"/> ashum	0
<input type="checkbox"/> christiek	1
<input type="checkbox"/> andyk	0
<input type="checkbox"/> jsnow	1
<input type="checkbox"/> pbeesly	1
<input type="checkbox"/> trevorg	0

3. Click **Save** and confirm the addition of 2 new records.

Our watchlist is ready to be correlated against our log data.

5.8.2 Query the watchlist

In Microsoft Sentinel switch to Logs blade and run the following query:

```
_GetWatchlist('Employees')
```

Note: There is a short delay for the watchlist data be visible in the workspace. Please allow a few minutes for the data to appear.

You should soon see the results from the watchlist you uploaded along with the records you added manually:

```
1 _GetWatchlist('Employees')
```

Results Chart Add bookmark						
<input type="checkbox"/>	LastUpdatedTimeUTC [UTC]	_DTItemId	SearchKey	accountname	ismanager	
<input type="checkbox"/>	> 5/10/2022, 11:52:28.716 AM	8e82a3dd-925b-4a0e-a98b-4f2a73f84d7a	jsnow	jsnow	1	
<input type="checkbox"/>	> 5/10/2022, 11:56:07.195 AM	19ee1785-b400-48c4-9828-dabbdf30ea4	scooper	scooper	1	
<input type="checkbox"/>	> 5/10/2022, 11:52:28.716 AM	9b95e23c-f08e-405d-ba36-73f14236bca2	andyk	andyk	0	
<input type="checkbox"/>	> 5/10/2022, 11:52:28.716 AM	3ec72b6a-1b1a-42ed-9ab5-eed1408a33db	pbeesly	pbeesly	1	
<input type="checkbox"/>	> 5/10/2022, 11:56:07.195 AM	e1120377-d239-4114-8798-ac87c51e3569	bkripke	bkripke	1	
<input type="checkbox"/>	> 5/10/2022, 11:52:28.716 AM	df56e813-b5aa-47eb-8b40-3de887afeddc	christiek	christiek	1	
<input type="checkbox"/>	> 5/10/2022, 11:52:28.716 AM	eb4b425e-ac85-471e-9dbb-f905dc81a28b	ashum	ashum	0	
<input type="checkbox"/>	> 5/10/2022, 11:52:28.716 AM	bb19d7d3-5776-4d73-9253-d435dc33c3a7	trevorg	trevorg	0	

Now let's do some filtering against the watchlist. Change the query so it looks like this:

```
_GetWatchlist('Employees')  
| where ismanager==1
```

As you can see, you can work with watchlists in the same way you work with tables.

5.8.3 Correlate logs data with business data

Our scenario: we want to find out whether any of the managers of our organization has been logging on to one of the computers (SCRANTON) that seem compromised.

Review and run the following query:

```
let start_time = ago(14d);
let LogonTypes = datatable(LogonType:int, FriendlyLogonType:string)
[
    2, "Interactive",
    3, "Network",
    4, "Batch",
    5, "Service",
    7, "Unlock",
    10, "Remote Interactive",
    11, "Cached Interactive",
];
let Managers = _GetWatchlist('Employees')
| where toint(ismanager) == 1
| project accountname;
fWindowsEvent
| where TimeGenerated > start_time
| where EventID == 4624
| where Computer=='SCRANTON.dmevals.local'
| extend LogonType=toint(EventData.LogonType)
| where EventData.LogonType!=3
| extend UserName = tostring(EventData.TargetUserName)
| extend SID = tostring(EventData.TargetUserSid)
| where UserName in (Managers)
| lookup kind=leftouter LogonTypes on LogonType
| project TimeGenerated, UserName, FriendlyLogonType, Computer
```

Many of the concepts in this query should already be familiar to you. Some topics that are worth mentioning:

- We're defining a datatable *LogonTypes* that is later used for quick joining using the *lookup* operator. This essentially lets us avoid defining joining criteria because the joining field names (LogonType) are the same. As a result, we get all the corresponding values from the lookup table automatically.
- We load the data from our Employees watchlist into a variable filtering out non-managerial staff, and then join the table with our main data set.
- We're interested in EventID 4624 (successful logon) and we're not interested in Network Logons (Type 3).
- We're also filtering out unnecessary fields.

Review the results:

```
1 let start_time = datetime('2022-04-17');
2 let end_time = start_time + 3d;
3 let LogonTypes = datatable(LogonType:int, FriendlyLogonType:string)
4 [
5     2, "Interactive",
6     3, "Network",
7     4, "Batch",
8     5, "Service",
9     7, "Unlock",
10    10, "Remote Interactive",
11    11, "Cached Interactive",
12 ];
13 let Managers = .GetWatchlist('Employees')
14 |> where toint(ismanager) == 1
15 |> project accountname;
16 WindowsEvent
17 |> parse kind = regex EventData.Message with * 'New Logon:\r\n\tSecurity ID:\t\t' SID:'\r\n\tAccount Name:\t\t' UserName:'\r\n'.*
18 |> extend LogonType=toint(EventData.LogonType)
19 |> where EventID == 4624 and EventData.LogonType!=3 and Computer=='SCRANTON.dmevals.local'
20 and TimeGenerated > start_time and TimeGenerated<end_time
21 and UserName in (Managers)
22 |> lookup kind=leftouter LogonTypes on LogonType
23 |> project TimeGenerated, UserName, FriendlyLogonType, Computer
24
25
```

Results			
<input type="checkbox"/>	TimeGenerated [UTC]	UserName	FriendlyLogonType
<input type="checkbox"/>	4/18/2022, 11:20:46.000 PM	pbeesly	Remote Interactive

Now remove the “**where**” filter related to LogonType and run the query again. Any additional results?

Modify the query so it also displays the authentication protocol (Authentication Package) the user utilized for the logons.

Have you been successful with that? *Congrats!* If not - ask your trainer for hints/assistance!

6 Microsoft Sentinel KQL Workbooks

Microsoft Sentinel Content Hub includes a content pack with two workbooks that have basic and advanced level KQL content for new and existing users looking to learn KQL.

6.1 Initializing workbooks

Open Microsoft Sentinel page, go to Content Hub and add the *KQL Training* content pack.

This installs two templates into the Workbooks blade. Once installed, go to Workbooks, then Templates, then search for "KQL". We are interested in two workbooks:

- Intro to KQL. Introductory level workbook that you can use for brushing up your KQL knowledge
- Advanced KQL for Microsoft Sentinel. Advanced-level KQL queries and examples that you might use for improving your KQL skills and as a source of ideas for own queries.

The screenshot shows the Microsoft Sentinel 'Workbooks' page. The left sidebar contains navigation links under 'General' (Overview, Logs, News & guides, Search (Preview)) and 'Threat management' (Incidents, Workbooks, Hunting, Notebooks, Entity behavior). The 'Workbooks' link is highlighted. The main area shows a search bar with 'kql' entered. Below the search bar, two workbooks are listed: 'Advanced KQL for Microsoft Sentinel' and 'Intro to KQL', both from the 'MICROSOFT SENTINEL COMMUNITY'. Above the list, there are statistics: 3 Saved workbooks, 126 Templates, and 1 Update. The 'Templates' tab is selected.

Microsoft Sentinel | Workbooks ...
Selected workspace: 'cybersoc'

Search (Ctrl+/) « Refresh + Add workbook

General

- Overview
- Logs
- News & guides
- Search (Preview)

Threat management

- Incidents
- Workbooks
- Hunting
- Notebooks
- Entity behavior

3 Saved workbooks 126 Templates 1 Updates

My workbooks Templates

Search: kql

Workbook name ↑↓

- Advanced KQL for Microsoft Sentinel
MICROSOFT SENTINEL COMMUNITY
- Intro to KQL
MICROSOFT SENTINEL COMMUNITY

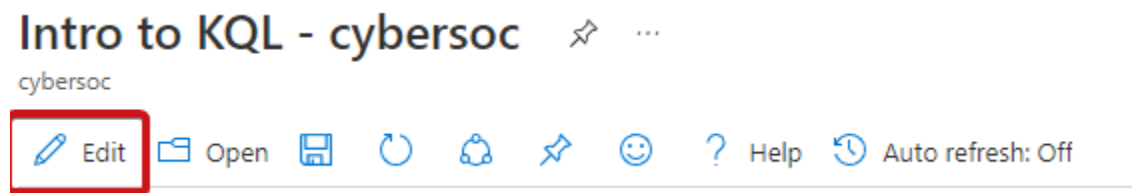
Install (save) both workbooks.

Note:

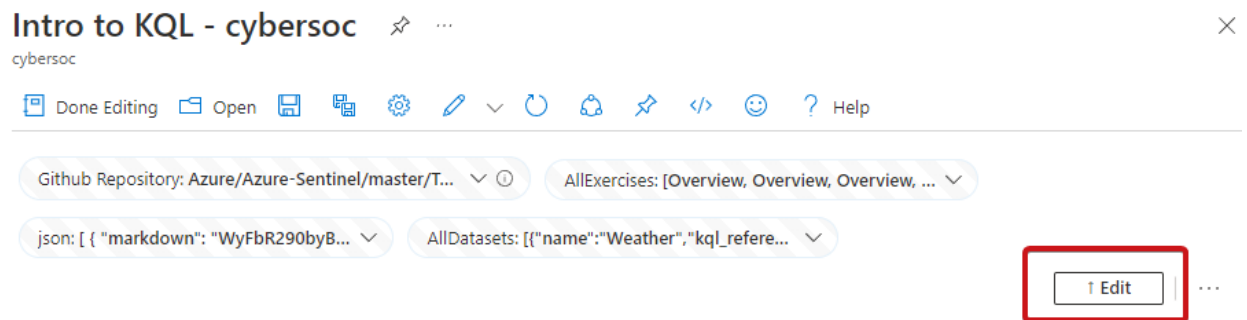
These workbooks use external sources (like github.com). Access to these external data sources are blocked by default, and you may have to explicitly allow access to them.

To unblock the external data sources:

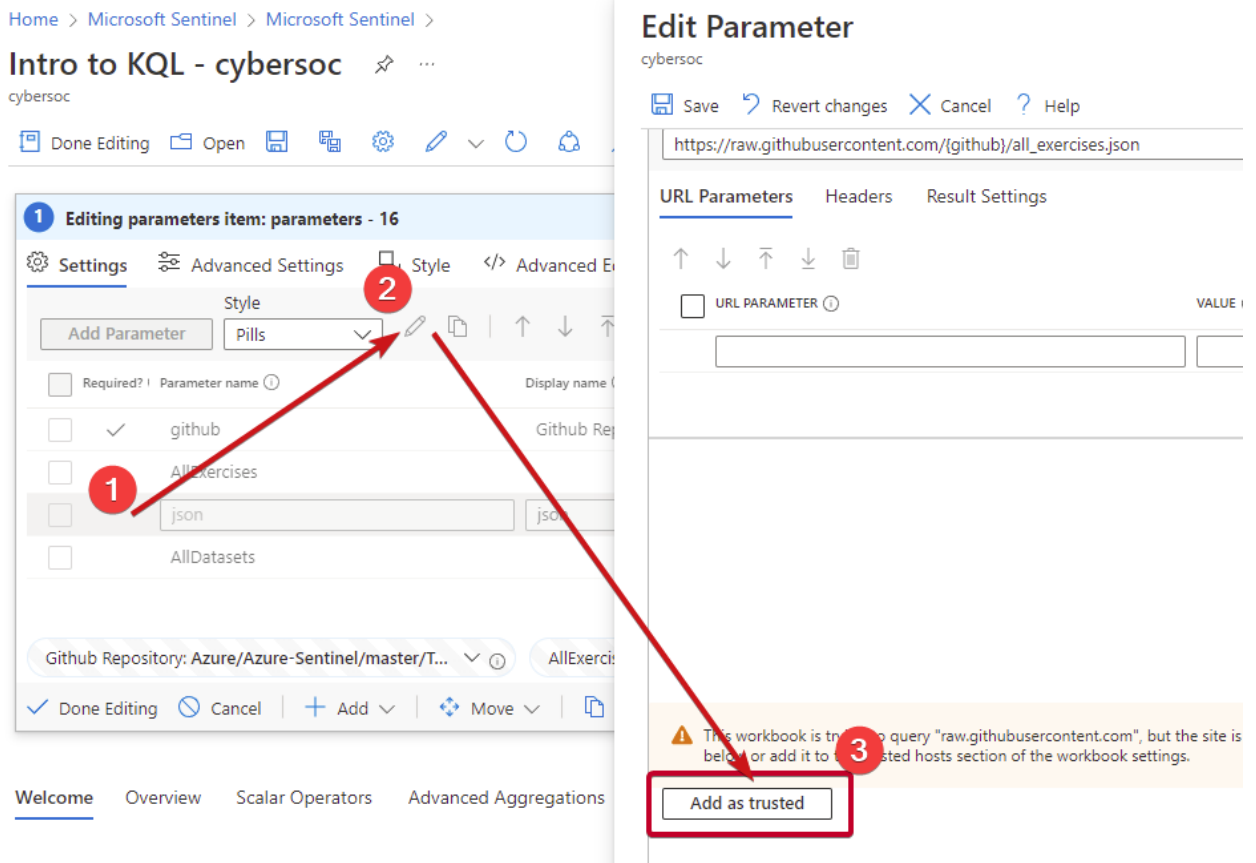
1. Open the workbook (click **View saved workbook** button)
2. Click **Edit** button:



3. Click on **Edit** button beneath the top section:



4. In Setting section select parameters, then click on the pencil button. In the new window, click on **Run Query** button.



6. Check other parameters to ensure that all external data sources (if there are anything else in addition to GitHub) are marked as trusted.
7. When you're done Save the changes and finish editing.

6.2 Working with Intro to KQL workbook

This workbook has been developed to assist new and existing users learn and grow in the Kusto Query Language (KQL). The goal of this workbook is to introduce the most commonly used KQL operators that are relevant to Microsoft Sentinel. By the end of the workbook, your knowledge will be at a 200 level.

This workbook is comprised of multiple tabs. Each tab contains several key items:

- Operator: choose an operator to study.
- Exercise: choose an exercise to practice.
- Data type: corresponds to the data table that is being used in the exercise.
- Answer: decide if you would like to see the answer.
- Summary: details about the operator that has been selected.
- Example: samples of how a real query would look like with the selected operator.
- When to use: advice around when the selected operator is used with Microsoft Sentinel.

The exercise area is made up of 6 main items:

- Question: selected exercise to perform.
- Answer space: location where you will enter your answer.
- Expected answer: the expected answer that you are attempting to achieve.
- Your answer: the results from the query you have written.
- Answer Checker: lists if the answer you have entered is correct or not.

To have practice with the workbook:

- Select a tab to navigate.
- Choose an operator to practice.
- Select an exercise to attempt.
- Enter your answer and confirm if it is correct. If not, reference documentation and content until correct.
- Move on to another operator or attempt other exercises for that operator.

Welcome Overview Scalar Operators **Advanced Aggregations** Dataset Operators External Data Strings (Coming Soon) Anomalies (Coming Soon) Misc. (Coming Soon)

Select Section: Arg_min Exercise: ArgMinEx1 Dataset: Weather Show Documentation: Yes No Show Answer: Yes No

Arg_min - Exercise: ArgMinEx1

[Documentation](#)

<https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/arg-min-aggregation>

Summary:

The arg_min operator finds a row in a group that minimizes a specified expression and returns the value of the expression to return.
Example: summarize [(NameExprToMinimize, NameExprToReturn [, ...])=] arg_min (ExprToMinimize, | ExprToReturn [, ...])

ExprToMinimize: Expression that will be used for aggregation calculation.
ExprToReturn: Expression that will be used for returning the value when ExprToMinimize is minimum. Expression to return may be a wildcard (*) to return all columns of the input table.

Example:

```
SecurityEvent | summarize arg_min(Computer, EventID)  
SignInLogs | summarize arg_min(TimeGenerated, UserDisplayName)
```

When to use:

arg_min is best used when looking to return the highest or most recent log for a specified item. The most common use for this operator is to find the most recent log ingested from a source. This can be useful when health checks, and finding most recent activity on entities or events.

Question: Return the records from each Location with the lowest High temperature for that Location ordered by High from lowest to highest

Put your answer here

6.3 Working with Advanced KQL for Microsoft Sentinel workbook

Working with Advanced KQL for Microsoft Sentinel interactive workbook is designed to help you improve KQL proficiency by taking a use-case driven approach based on:

- Grouping KQL operators/commands by Category for easy navigation.
- Listing possible tasks a user would perform with KQL in Microsoft Sentinel. Each task includes KQL operators used, sample queries and use cases.
Compiling a list of existing content found in Microsoft Sentinel (Analytics Rules, Hunting Queries, Workbooks etc.) to provide additional references specific to the KQL operators you want to learn.
- Allowing you to execute the sample queries on the fly with your own environment (having APT29 sample data ingested) or "LA Demo" - a public demo environment. Try the sample KQL statements in real time without the need to navigate away from the Workbook.

How to use this Workbook:


- Begin by select a Category that suits the outcomes you would like to achieve with KQL. Alternatively, you can refer to "Table of Contents" for an overview of what is covered under each Category.
- After clicking on a Category, you will be presented a list of tasks under "I want to". Select a task that suits your requirement.
- Review the Operators, sample queries, sample use cases and reference resources found in Microsoft Sentinel accordingly.
- Try the queries in your own environment or "LA Demo" using the "Try it" buttons.

Note:

Try it in your environment button will only be visible when the **Workspace** parameter at the top of the workbook is populated.

LA Demo environment might not have all the tables or data specified for each sample queries and use cases.

Advanced KQL for Microsoft Sentinel

 [Click here for Overview](#)

Workspace: CyberSOC

Category

- Aggregation
- Converting value
- Correlation
- Dealing with Array values
- Dealing with Datetime
- Dealing with IP address
- Dealing with Fields
- Filter
- Anomalies
- Functions (e.g. parser, parameterized)
- Parsing
- Using Watchlist Data

7 Answers to the exercises

7.1 Filtering

7.1.1 Simple filtering

You should get something like this query:

```
fWindowsEvent
| where Channel == 'Security'
| where EventID == 4688
| where EventData.NewProcessName contains "sdclt.exe"
```

7.1.2 Using different predicates with 'where'

Your query might look like this:

```
fWindowsEvent
| where Channel == 'Security'
| where EventID == 4624
| where EventData.TargetUserName =~ 'PBEESLY'
```

7.1.3 Using lookups with 'where'

Your query might look like this:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 5145
| extend ProcName=toString(EventData.RelativeTargetName)
| where EventData.RelativeTargetName has_any ('psexesvc.exe', 'python.exe')
```

7.2 Analyzing

7.2.1 Summarize by

Your resulting query might look like this one:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 5145
| where EventData.ShareName contains 'IPC'
| extend RelativeTargetName = tostring(EventData.RelativeTargetName)
| where RelativeTargetName has 'psexesvc'
| summarize count() by Computer, RelativeTargetName
```

7.2.2 Checking connections

The resulting query should look like:

```
let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 5156
| extend App = tostring(EventData.Application)
| where App contains "psexec" or App contains '3aka3'
| extend IP = tostring(EventData.DestAddress)
| extend Port = tostring(EventData.DestPort)
| summarize count() by Computer, App, IpPort=strcat(IP,":",Port)
```

7.2.3 Most recent services

The resulting query might look like:

```

let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 4697
| extend ServiceName = tostring(EventData.ServiceName)
| summarize arg_max(TimeGenerated, *) by ServiceName, Computer

```

7.3 Presenting

7.3.1 Charting logons intensity

Your query might look like this one:

```

let start_time = ago(14d);
fWindowsEvent
| where TimeGenerated > start_time
| where Channel == 'Security'
| where EventID == 4624
| where EventData.TargetUserName !endswith '$' and EventData.TargetUserName != 'SYSTEM'
| summarize count() by bin(TimeGenerated, 1m), Computer
| render timechart

```

7.3.2 What are the events?

Your resulting query might look like this:

```

fWindowsEvent
| where Channel == 'Security'
| summarize count() by EventID
| top 10 by count_
| extend EventID = tostring(EventID)
| render piechart

```


7.4 Advanced KQL

7.4.1 Finding rare processes

Your resulting query might look like this:

```
let Allowlist = dynamic (['SearchProtocolHost.exe',
'SearchFilterHost.exe', 'MsMpEng.exe']);

let Sensitivity = 15;

fWindowsEvent
| where EventID == 4688
| extend ProcArray = split(EventData.NewProcessName, '\\')
// ProcArrayLength is Folder Depth
| extend ProcArrayLength = array_length(ProcArray)
| extend LastIndex = ProcArrayLength - 1
| extend Proc = ProcArray[LastIndex]
| where Proc !in (Allowlist)
// ProcArray[0] is the proc's Drive
| extend DriveDepthProc = strcat(ProcArray[0], '-', ProcArrayLength, '-',
Proc)
| summarize StartTimeUtc = min(TimeGenerated), EndTimeUtc =
max(TimeGenerated), TimesSeen = count(), HostCount = dcount(Computer), Hosts =
make_set(Computer), UserCount = dcount(tostring(EventData.SubjectUserName)),
Users = make_set(tostring(EventData.SubjectUserName)) by DriveDepthProc
| where TimesSeen < Sensitivity
| extend timestamp = StartTimeUtc
```

7.5 Optimizing

Your resulting query might look like this one:

```
let start_time = datetime('2022-04-17');
let end_time = start_time + 3d;
fWindowsEvent
| where TimeGenerated < end_time and TimeGenerated > start_time
| where EventID ==1
| where Channel == "Microsoft-Windows-Sysmon/Operational"
| extend Image = tostring(EventData.Image)
| where Image endswith 'python.exe'
| extend ParentProcessGuid = tostring(EventData.ParentProcessGuid)
| join kind=innerunique (
fWindowsEvent
| where TimeGenerated < end_time and TimeGenerated > start_time
| where EventID ==1
| where Channel == "Microsoft-Windows-Sysmon/Operational"
| where EventData.ParentImage endswith "services.exe"
| extend ParentImage=tostring(EventData.ParentImage)
| extend ProcessGuid = tostring(EventData.ProcessGuid)
) on $left.ParentProcessGuid == $right.ProcessGuid
| distinct TimeGenerated, Image, ParentImage, Computer
```

The key changes to the initial query:

- **let** defines the timeframe;
- We explicitly filter on timeframe for both sides of the **join** operator;
- **ProcessId** is not a part of the resulting column set, we can simply drop it;
- Filtering is done early for both tables participating in **join**;
- We do **extend** only after we filtered out unnecessary events.