Microsoft

Security Rockstar Hands-on Lab with Azure Sentinel

Technical Workshop

In this lab, attendees will learn how to use Azure Sentinel to visualize their customer's security estate, and help them achieve actionable results from their visualized logging and auditing data. This Technical Workshop will provide hands-on experience using Azure Sentinel. You will learn how to react to indicators of attack using data connectors, analytics and Azure Playbook automation. Additionally, you'll proactively hunt adversaries using Workbooks, hunting KQL queries, and Jupyter Notebooks to ensure your customers have the best chance of early detection and actionable sets of data.

Introduction

Objectives

After completing this lab, you will be able to:

- Deploy Azure Sentinel as a platform for visualizing, investigating, and alerting on your customer's security big data
- Confidently use Sentinel to render your customer's visualized data truly actionable
- Respond to security incidents and Indicators of Compromise discovered with Sentinel
- Proactively hunt for misconfigurations and Indicators of Compromise with hunting queries

Prerequisites

Before working on this lab, the following helps:

- Working knowledge of SIEM and SOAR technology
- Familiarity with common attack scenarios and techniques
- Familiarity with Azure Playbooks and automation
- Awareness of Kusto Query Language (KQL)

Overview of the Technical Workshop

In this technical workshop, attendees will learn the importance of effectively representing big data to tell an actionable and persuasive security story. Attendees will use new technologies to accomplish data analysis that wasn't readily available previously. Specifically, attendees will use Azure Sentinel in two distinct ways.

First, attendees use Data Connectors to Connect and Aggregate data, Kusto queries to filter data and to create Analytics (Alerts and Incidents), and Playbook automation to efficiently respond.

Then, attendees will switch gears to learn a Proactive posture; attendees will use Sentinel to hunt for adversaries using Workbooks, Hunting KQL queries, and Jupyter Notebooks.

Scenario

In this technical workshop, attendees will analyze log data from a Pre-populated Sentinel workspace. You will not need (and won't be able to use) your own Azure tenant or subscription. User names and passwords will be provided throughout. Please log on exactly as detailed, including the number that you have been assigned. **Failure to do so will result in potential problems for other lab participants.**

The Sentinel workspace(s) that you will be working with have been configured with multiple data sources, including Office 365, Azure Active Directory (AD) Sign-in logs, Azure AD Audit logs, and Windows Security Event logs. All Data connectors have been configured to forward all relevant data to the sample Sentinel workspace(s) you'll be working with. Several vulnerabilities have also been configured in these data source environments.

Lab technology

In this technical workshop, you will not need to access any IaaS or any other machines directly. Instead, you will use your own computer to access a preconfigured lab in Azure.

User accounts

To access the azure lab, you will navigate to: https://portal.azure.com.

You will then logon with the following account:

AdminXX@sentinellab.xyz, where "XX" is the number you have been assigned at the start of class.

For example, if I was assigned number 47, my username would be user47@sentinellab.xyz

Username: **AdminXX@sentinellab.xyz**

Password: **Assigned by instructor**

This is the account you will use except where otherwise specified (as in Exercise 2).

Important Note: Because data changes rapidly, throughout this lab your results may be slightly different than the examples given throughout this lab guide. The guide is meant to be illustrative; slight differences in data are not a problem.

Important Note 2: All exercises in this lab should be completed using an InPrivate (Edge) or Incognito (Chrome) browser session. This will ensure that your Microsoft or other credentials do not interfere.

Exercise 1: Review Azure Sentinel Data Connectors

In this exercise you will be reviewing the data connectors of Azure Sentinel to confirm that data is already being collected in the Sentinel Workspace. Data connectors are critical to a setup of Azure Sentinel so that data can flow into the workspace. To save time, these data connectors have been configured in advance.

Task 1: Log on to Azure and Navigate to Sentinel

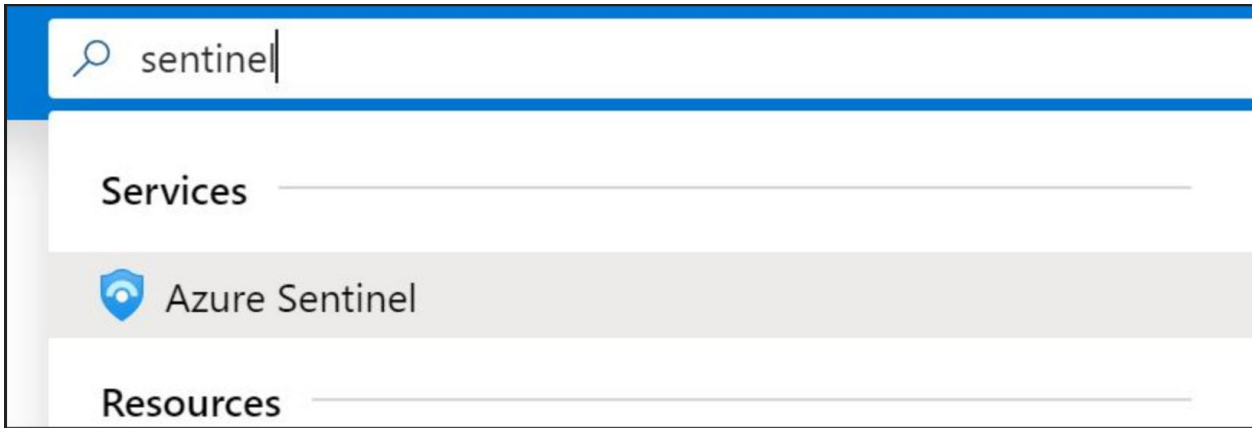
In this task, you will log into the Azure portal and navigate to Sentinel.

1. In a browser, navigate to **https://portal.azure.com**.
2. Enter the account information:

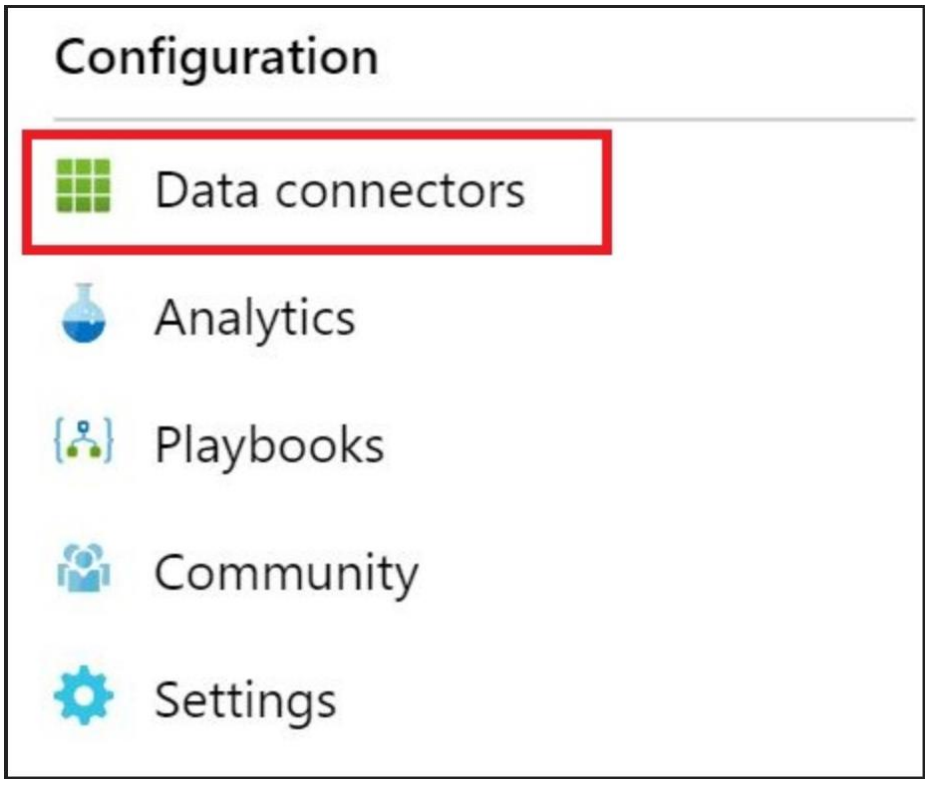
Username: **AdminXX@sentinellab.xyz** (where "XX" is the number you have been assigned at the start of class)

Password: **Assigned by instructor**

3. This brings you to the Azure Home. In the box at the top, search for **Sentinel**.
4. In the results, click on **Azure Sentinel**



5. Click **briandel**, which is the name of the Sentinel Workspace.
6. In the Navigation bar, select Data Connectors



7. Locate and click on the data connector **Azure Active Directory.**
8. Click **Open connector page**
9. Confirm the status of the connector as **Connected** on the left side of the page.

10. Confirm that there has been data received within the last 30 minutes.

Exercise 2: Setup Alerts (Analytics) and Respond to Incidents in Azure Sentinel

In this section we will configure Alerts in Azure Sentinel to generate Incidents for our security team. During the review of these incidents we will build a Runbook to simplify the handling of these alerts.

Task 1: Log on to the My Apps portal as a Standard User

In this task you will logon to the My Apps portal as a standard user to confirm the accounts functionality. Afterwards, you will send bad passwords to the account to simulate an unauthorized user attempting to gain access.

1. In a new browser session, navigate to **https://myapps.microsoft.com**
2. Enter the account information:

Username: **standardXX@sentinellab.xyz** (where "XX" is the number you have been assigned at the start of class)

Password: Provided by instructor

3. Confirm you have access to the My Apps portal.
4. Close all instances of your browser and Navigate back to **https://myapps.microsoft.com**
5. This time, when logging on as standardxx@sentinellab.xyz type in an incorrect password
6. Repeat this invalid logon attempt 5 times

Task 2: Log on to Azure and Navigate to Sentinel

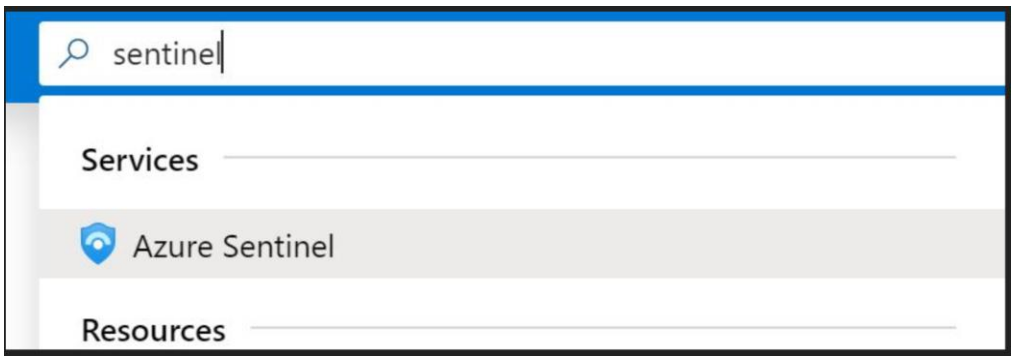
In this task you will log into the Azure portal and navigate to Sentinel.

1. In a browser, navigate to **https://portal.azure.com**.
2. In the window that opens, click **Use another account**.
3. Enter the account information:

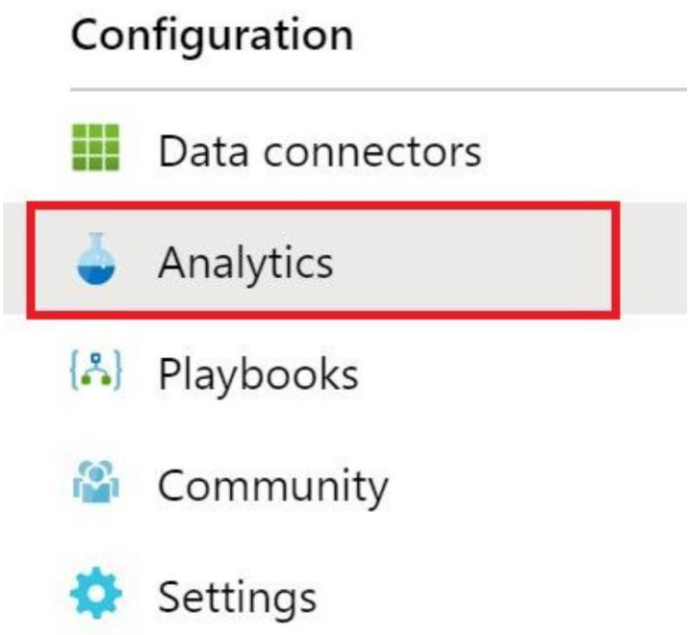
Username: **AdminXX@sentinellab.xyz** (where "XX" is the number you have been assigned at the start of class)

Password: Provided by instructor

4. This brings you to the Azure Home. In the box at the top, search for **Sentinel**.
5. In the results, click on **Azure Sentinel**



6. Click **briandel**, which is the name of the Sentinel Workspace.
7. Within the navigation bar, click **Analytics**



Task 2: Create Analytics Rule for Failed Logon

1. Click **Create** and choose **Scheduled query rule** from the flyout menu
2. In the Name box type standardXX – Failed Logon (where "XX" is the number you have been assigned at the start of class)
3. Leave the **Description** box blank
4. For Tactics, Choose **Initial Access**
5. Select Alert Severity **Medium**
6. Set Status to **Enabled**
7. Click **Next: Set rule logic**

8. In **Rule query** copy the following query (substitute "XX" with your assigned number)

```
SigninLogs
| where Status.errorCode == 50126 // Invalid Username or password
| where UserPrincipalName contains "standardXX"
| project TimeGenerated, Status.failureReason, UserPrincipalName,
Status.errorCode, UserId, IPAddress
| sort by TimeGenerated desc
```

9. Under Entity Mapping, map the following entities
    - Account -> UserprincipalName (Click **Add**)
    - IP address -> IPAddress (Click **Add**)
10. Set the Query scheduling to:
    - Run query every: **5 minutes**
    - Lookup data from the last: **30 minutes**
11. Set the Alert Threshold to 'Generate alert when number of query results' to **Is greater than** and the threshold to **0**
12. Under Suppression, Set Stop running query after alert is generated to **On** and Stop running query for: **30 minutes**
13. At the bottom, click **Next : Automated response**.
14. Do not change anything on the next screen and at the bottom click **Next: Review**.
15. Click **Create**.

Task 3: Create Analytics Rule for Admin Group Change

16. Click **Create** and choose **Scheduled query rule** from the flyout menu

In the Name box type AdminXX – Admin Group Change (where "XX" is the number you have been assigned at the start of class)

1. Leave **Description** blank
2. For Tactics, choose **Persistence** and **Privilege Escalation**
3. Select Alert Severity **Medium**
4. Set status to **Enabled**.
5. At the bottom, click **Next: Set rule logic**.
6. In **Rule query** copy the following query

```
let accttypes = dynamic(['Domain Admins', 'Enterprise Admins', 'Schema Admins',
'Administrators' , 'Account Operators' , 'Backup Operators' , 'Print Operators' ,
```

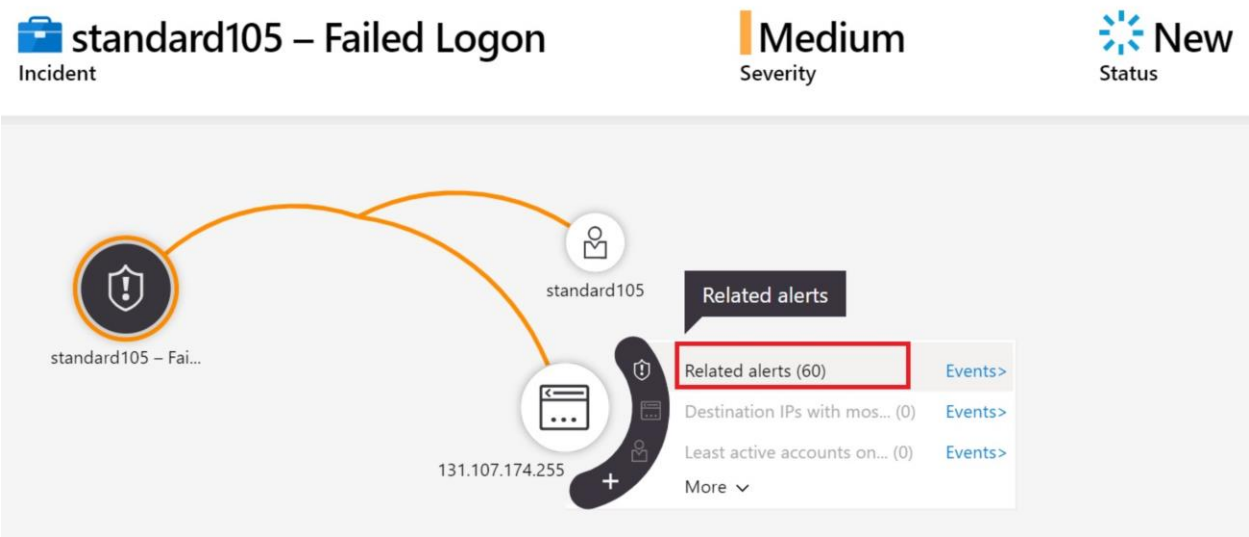
```
'Server Operators' , 'Domain Controllers' , 'Read-only Domain Controllers' ,
'Group Policy Creator Owners' , 'Cryptographic Operators']);
SecurityEvent
| where EventID in (4728, 4729, 4732, 4733, 4756, 4757)
| extend ChangeType = case(EventID in (4729, 4733, 4757), "Member Removed",
"Member Added")
| parse EventData with *'"TargetUserName">'TargetUserName'<'*
| where TargetUserName in (accttypes)
| project ChangeType, ImpactedGroup=TargetUserName, ImpactedAccount=MemberName,
ChangeMaker=Account, DC=Computer
```

7. Under Entity Mapping, map the following entities
    - Account -> ImpactedAccount (Click **Add**)
    - Host -> ImpactedGroup (Click **Add**)
8. Set the Run query every to **30 minutes**
9. Set the Lookup data from the last to **1 hour**
10. Set the Alert Threshold to 'Generate alert when number of query results' to **Is greater than** and the threshold to **0**
11. Set Stop running query after alert is generated to **On** and Stop running query for: **8 hours**
12. Click **Next: Automated response**
13. Leave the following page unchanged and click **Next: Review**
14. Click **Create**

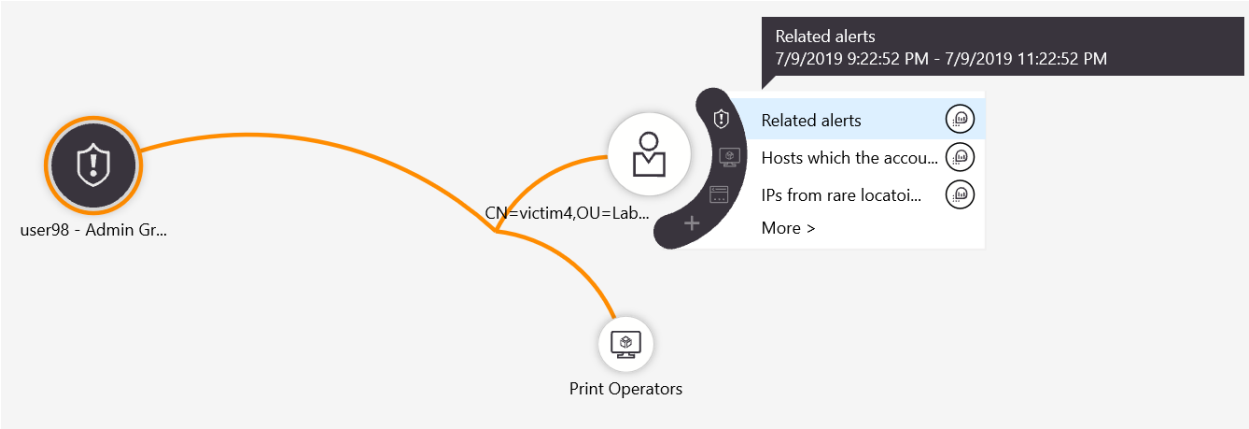
Task 4: Review your Incidents

1. Navigate back to **Incidents**
2. Locate your incident standardXX – Failed Logon (where "XX" is the number you have been assigned at the start of class)
3. Click your Incident. Then Click the **Investigate** button on the lower right-hand side
4. Within the Investigation window hover over the IP Address node and select **Related Alerts** to bring in additional alert data associated to that IP address

    Note: Data will vary based on the other events occurring in the lab. The IP address will likely be different and the number of related alerts will depend on other users in the lab. This is how **Investigation** within Sentinel is intended to function.



5. You may explore other available options in the investigation graph. However, depending upon the data available at the time, some options in the graph will be grayed out.
6. Navigate back to Azure Sentinel -> Incidents and locate your incident AdminXX – Admin Group Change
7. Click on your incident and Click the **Investigate** button
8. Within the investigation window you can see the User that was added to the group, and the Group that they were added to.
9. Hover over the User account victim4 and click **Related alerts** to see additional alerts related to this Admin group change involving this account.

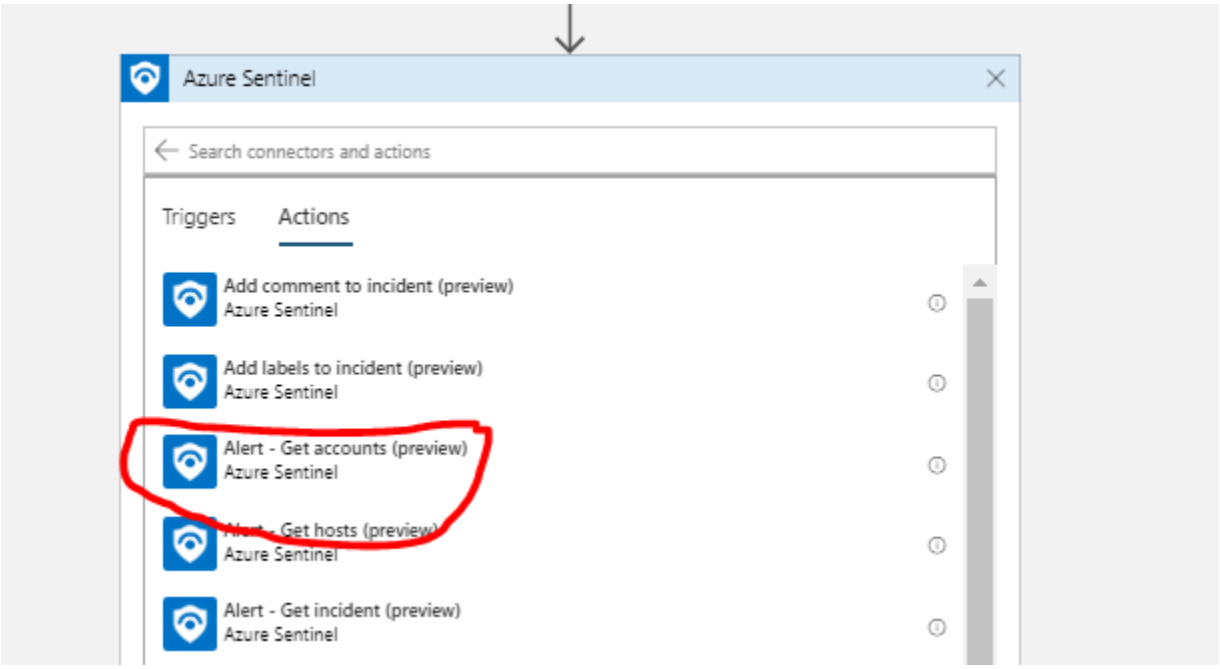


Task 5: Create a playbook to respond to your incident

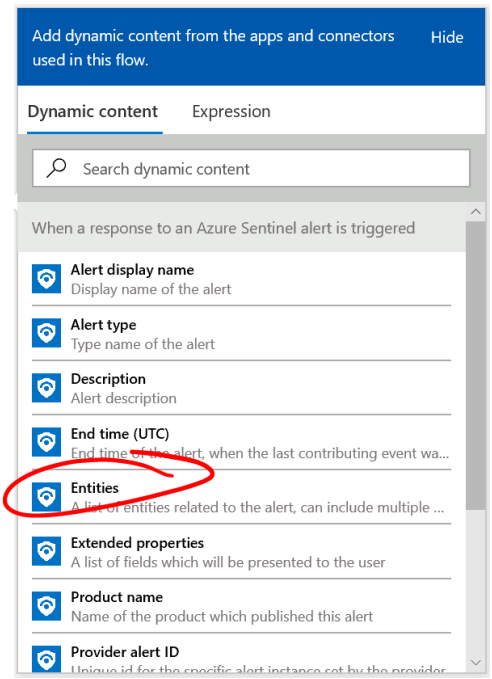
In this playbook we will be adding a user into a group based on a incident. This group is already linked to a conditional access policy which will block user logons to all cloud applications.

1. In the Azure Portal, search for **Logic Apps** in the Search Bar at the top.
2. Click the **Add** button.
3. Under Resource Group, Select **OI-Default-East-US**

4. In the Logic App Name field, type standardXX-Block-With-Conditional-Access-Policy (where "XX" is the number you have been assigned)
5. Select Location **East US**
6. Leave Log Analytics turned **Off**
7. Click **Review + Create**
8. On the next screen, click **Create**.
9. Wait for the Logic App to deploy.
10. On the *Your Deployment is complete* screen, click **Go to resource**.
11. In the Logic Apps Designer, Click **Blank Logic App**
12. In the text box **Search connectors and triggers** type **Sentinel** and click on the icon for **Azure Sentinel.**
13. Select **When a response to an Azure Sentinel alert is triggered**
14. Click **New Step**
15. Search for **Sentinel** and click on the **Azure Sentinel** icon
16. Click **Alert - Get Accounts (preview)**

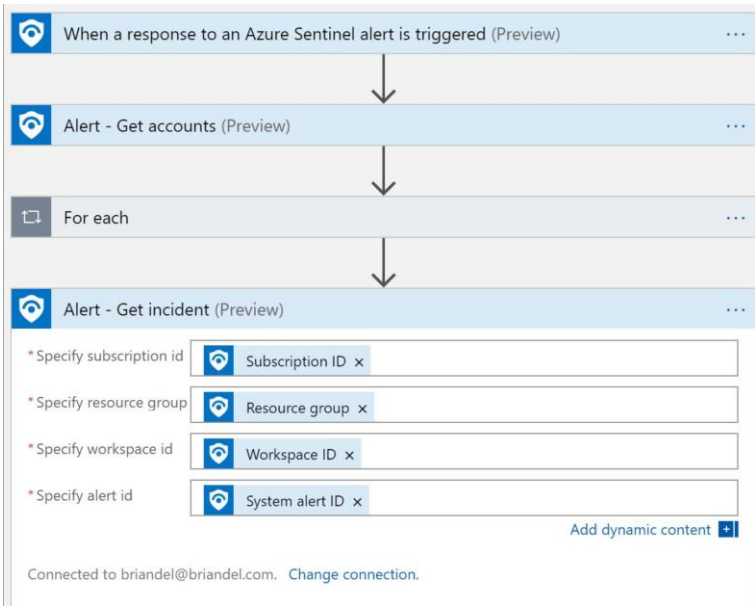


17. Click the text box **Entities List** and from the flyout, select **Entities**

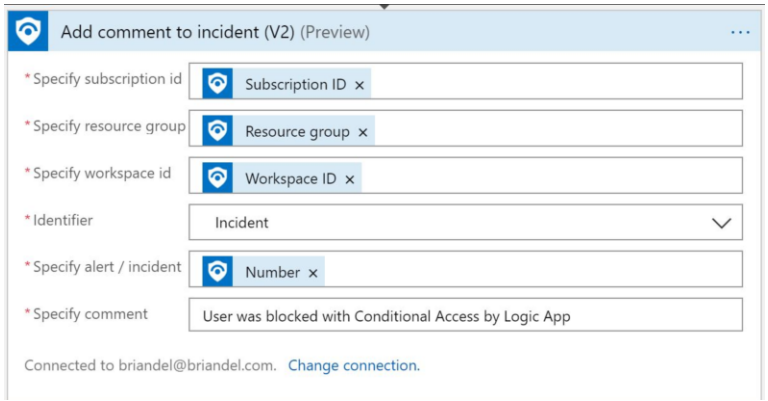


18. Click **New Step** and search for and select **Control.**
19. **C**lick on **For each**
20. In the **Select an output from previous steps** click in the text box and click **Accounts** in the flyout
21. Click **Add an action**
22. Search for and select **Azure AD**
23. Select **Get user**
24. Click in the **User Id or Principal Name** field
25. From the flyout click **Accounts Name**
26. In the flyout, click **Expression.**
27. In the **fx** box type **string('@')** and click **Ok.**
28. From the flyout, click **Accounts UPN Suffix**
29. Next, click **Add an action**
30. Search for and click on Azure AD
31. Select **Add user to group**
32. Under Group ID, enter **4aa42844-7b1a-410f-b4eb-aacdc3e98b92**
33. Under User ID, select **Id** from the flyout
34. At the bottom of the page, click **New Step**
35. Search for Azure Sentinel and select **Alert – Get incident (preview)**
36. Click Specify Subscription Id and select **Subscription ID** in the flyout
37. Click Specify resource group and select **Resource group** from the flyout

38. Click Specify workspace id and select **Workspace ID** in the flyout
39. Click Specify alert id and select **System alert ID** in the flyout



40. Click New Step, search for Azure Sentinel and select **Add comment to incident (V2) (preview)**
41. Click Specify Subscription Id and select **Subscription ID** in the flyout
42. Click Specify resource group and select **Resource group** from the flyout
43. Click Specify workspace id and select **Workspace ID** in the flyout
44. Click Identifier and select **Incident** from the drop down menu
45. Click Specify alert / incident and click **Number** from the flyout
46. Click Specify comment and type in **"User was blocked with Conditional Access by Logic App"**

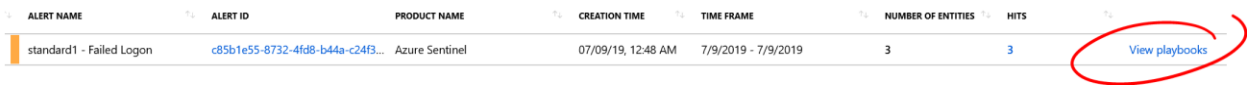


47. Click **Save**

Task 6: Use your new playbook to respond to a incident

1. Navigate back to **Sentinel -> Incidents** in the Azure Portal
2. Click on your incident standardXX – Failed Logon

3. Click **View full details**
4. Under the Incident, click **View playbooks**

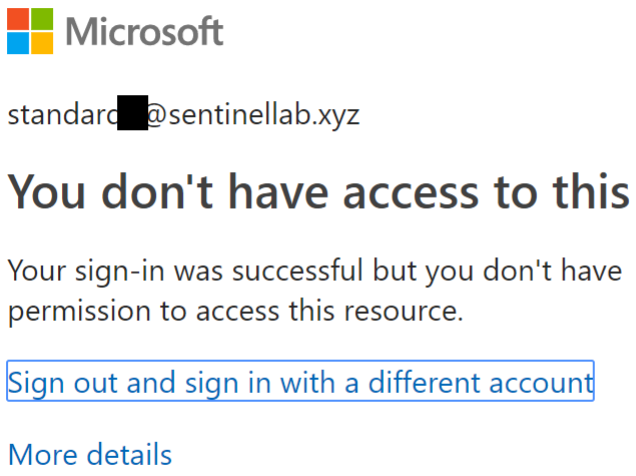


5. From the list of Playbooks locate **standardXX-Block-With-Conditional-Access-Policy** and click **Run**
6. Close all browser windows and open a new browser
7. Navigate to **https://myapps.microsoft.com**
8. Enter the account information:

Username: **standardXX@sentinellab.xyz** (where "XX" is the number you have been assigned at the start of class)

Password: assigned by instructor

9. You will now see that access to the my apps portal has been blocked due to conditional access



Task 7: (Optional) Review Additional Logic Apps Controls

Navigate back to your Logic App and review other available controls to automate a response to your Sentinel Incident.

Exercise 3: Proactively Investigate Potential Threats, Misconfigurations, and Suspicious Activities Visually

In this exercise, you will learn how to access Azure Sentinel and how to use a Workbook to view data visually. You will also learn how to examine the underlying Kusto queries, and to change them.

For this lab, Data connectors have already been added and appropriately configured as necessary.

Task 1: Log on to Azure and Navigate to Sentinel

1. In a browser, navigate to **https://portal.azure.com**.
2. Enter the account information:

Username: **AdminXX@sentinellab.xyz** (where "XX" is the number you have been assigned at the start of class)

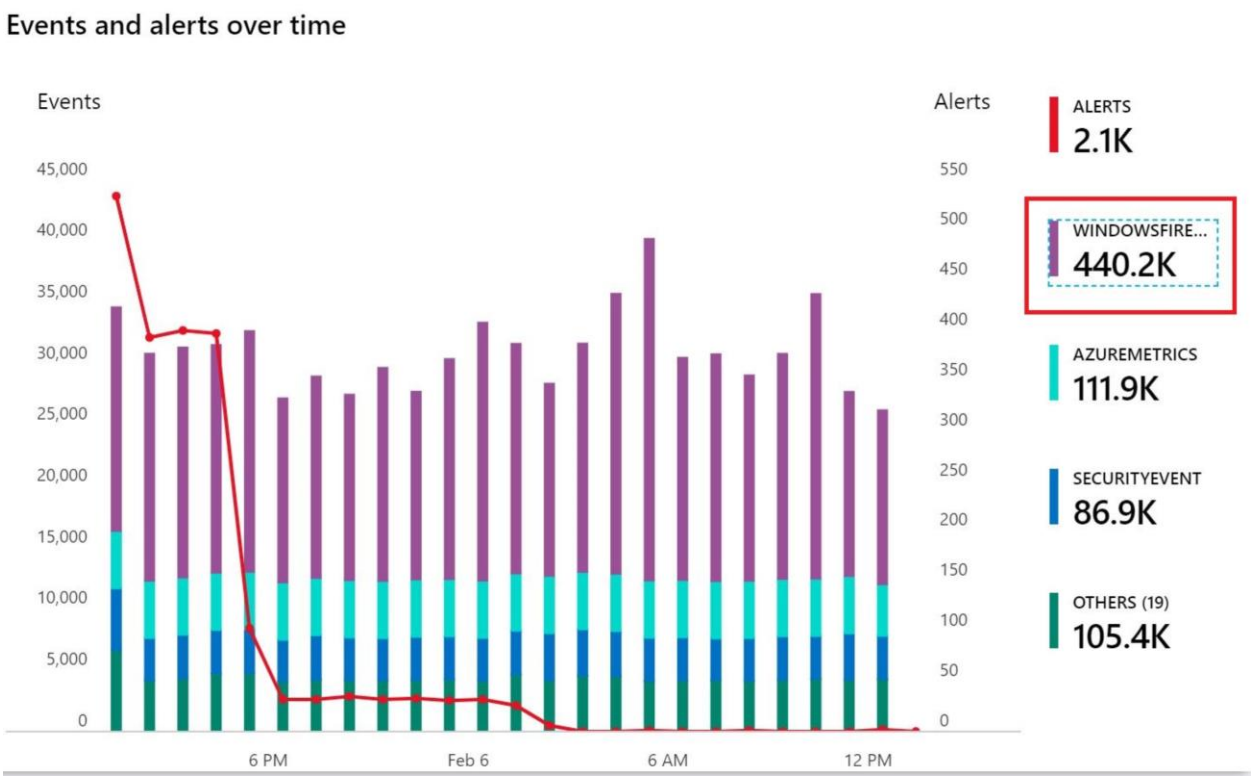
Password: supplied by your instructor

3. This brings you to the Azure Home. In the box at the top, search for **Sentinel**.
4. In the results, click on **Azure Sentinel**

5. Click **briandel**, which is the name of the Sentinel Workspace.
6. Take a moment to examine the Sentinel **Overview** page. This page gives you a bird's eye view of your estate's data.

Task 2: Navigate the Overview Page and Understand Basic Kusto Queries

1. Look closely at the pane entitled **Events and alerts over time**. This pane provides a bar graph representing alerts, Windows Firewall Events, Security Events, Performance data, and other data that has been recently gathered.



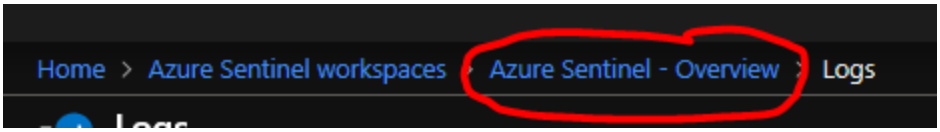
2. Click on **WindowsFirewall** on the Right-hand side.

3. You may encounter a **Welcome to Log Analytics** screen. If so, click **Get Started.** If not, simply continue with the next step.
4. This opens up the **Logs** window and also demonstrates a critical point to understand. Azure Log Analytics (and the Kusto Query Language) forms a large portion of the platform upon which Azure Sentinel has been built.
5. Once the query ("WindowsFirewall") finishes, examine the data.
6. Click one of the chevrons on the left to inspect a data element as in the example below (your data will be different).

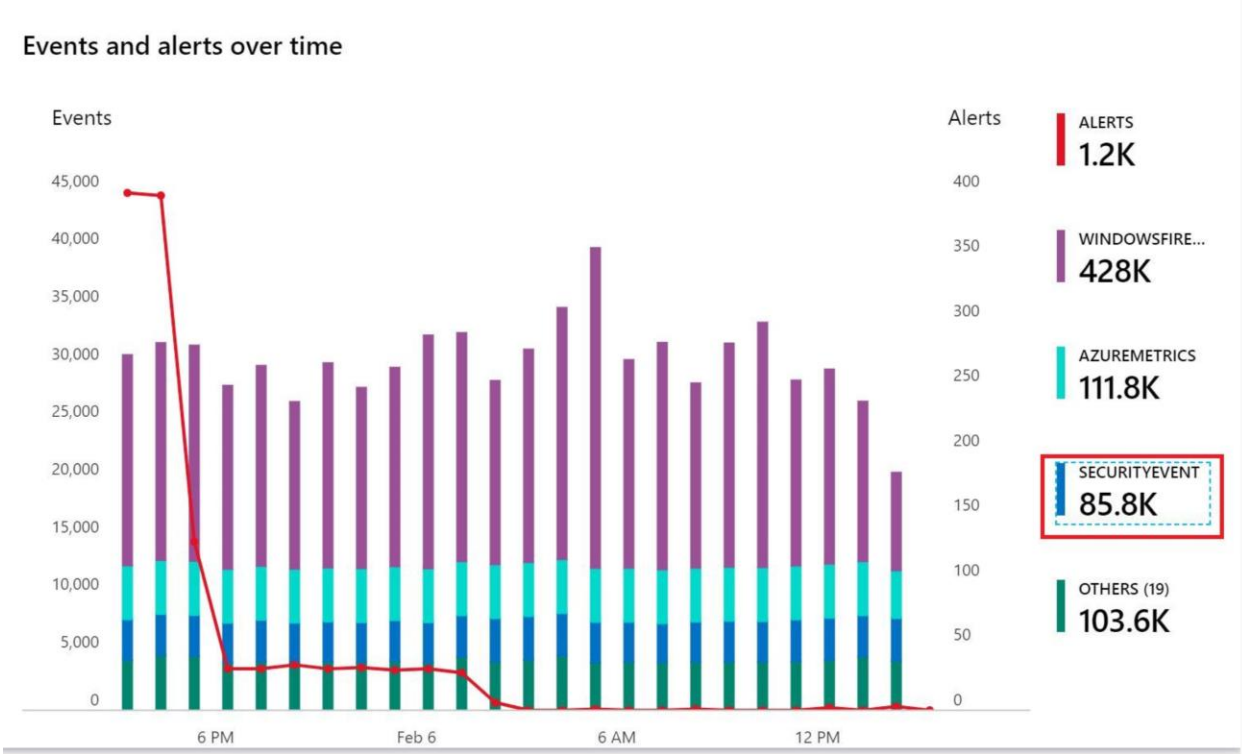
2019-07-05T15:20:58.000	DC04.ad.briandel.ca	SEND	ALLOW	UDP

TenantId	6ba2759c-1c00-4aa0-88e8-138379ea383c
SourceSystem	OpsManager
Computer	DC04.ad.briandel.ca
TimeGenerated [UTC]	2019-07-05T15:20:58Z
CommunicationDirection	SEND
FirewallAction	ALLOW
Protocol	UDP
SourceIP	192.168.2.4
DestinationIP	10.0.0.4
RemoteIP	10.0.0.4
SourcePort	59,516
FullDestinationAddress	10.0.0.4:53
DestinationPort	53
MG	00000000-0000-0000-0000-000000000001
TimeCollected [UTC]	2019-07-05T15:58:18.827Z
ManagementGroupName	AOI-6ba2759c-1c00-4aa0-88e8-138379ea383c
Type	WindowsFirewall

7. In keeping with this simple query, this piece of data in the example above tells a simple story. In the example here, the machine DC04.ad.briandel.ca sent (through the Firewall) a UDP packet to 10.0.0.4 over Port 59,121. In itself, it is not terribly interesting. But in aggregate or as part of an investigation, Kusto queries like this can provide critical data as we shall see.
8. Return to the Sentinel Overview page by clicking **Azure Sentinel – Overview** at the top.



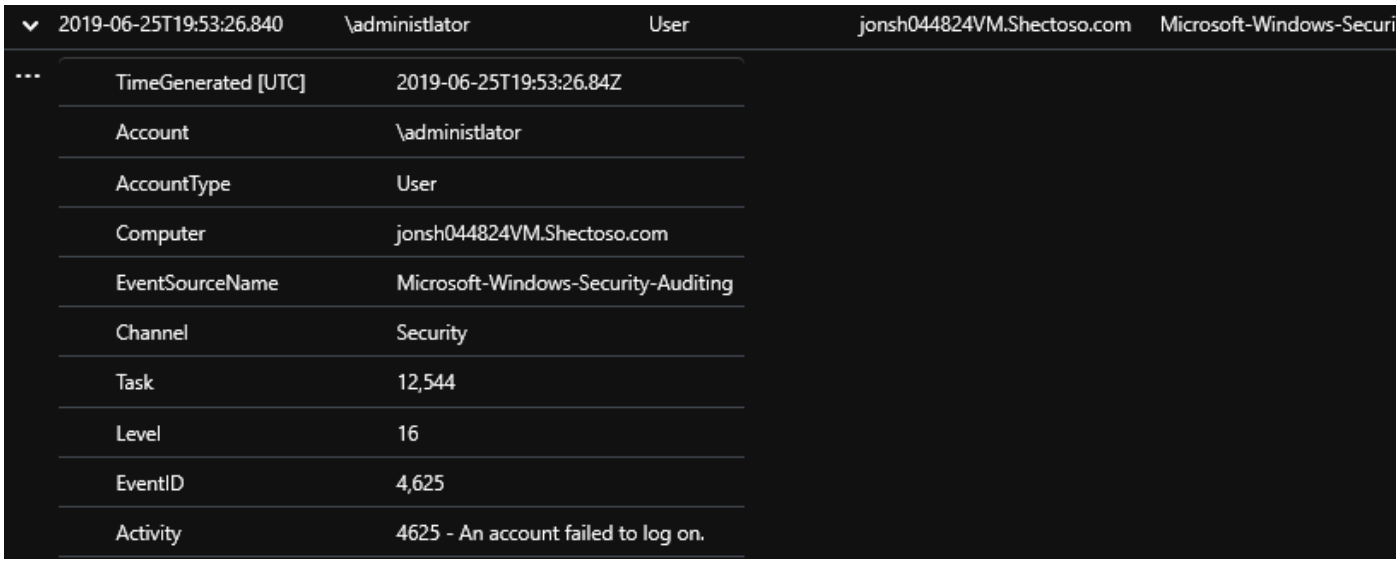
9. Click one **Security Event** in the Events and alerts over time pane.



10. This query is slightly more complex and will look something like this (the date and time fields will be different):

```
union SecurityEvent
| where TimeGenerated >= datetime(2019-10-03T06:24:39.208Z) and TimeGenerated < datetime(2019-10-03T06:24:39.208Z) + 1h
```

11. This query is asking for a SecurityEvent (from a Windows security log) during a specified date and time.
12. Click on one of the chevrons, preferably for an **AccountType** of **User** and not a built-in account like NT AUTHORITY.



In this example above, the administrator account has failed to log on. This could be very telling as an early indicator of attack. Why is someone logging on (and failing) using the administrator account? Your data will look different than the example above.

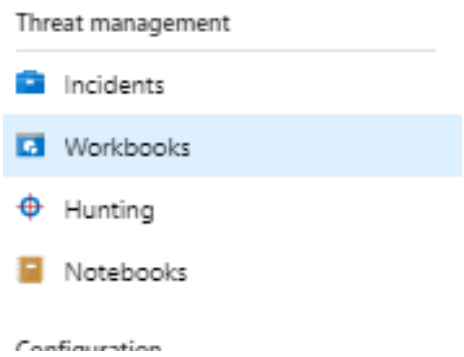
13. Click **Azure Sentinel – Overview** to return to the Sentinel Overview page.

Task 3: Utilize Workbooks to Proactively Investigate Potential Threats, Misconfigurations, and Suspicious Activities

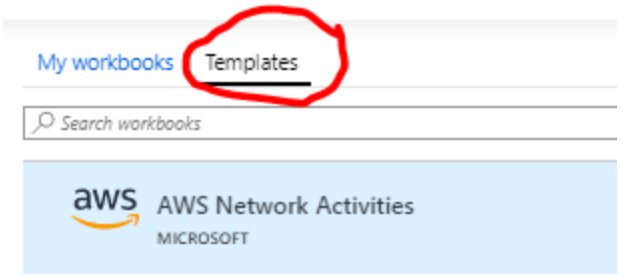
Step 1: Learn to navigate the Workbooks frame

In this task, you will learn to proactively investigate using Sentinel Workbook. Sentinel Workbooks provide visualized data from your connected data sources to help render them proactively actionable.

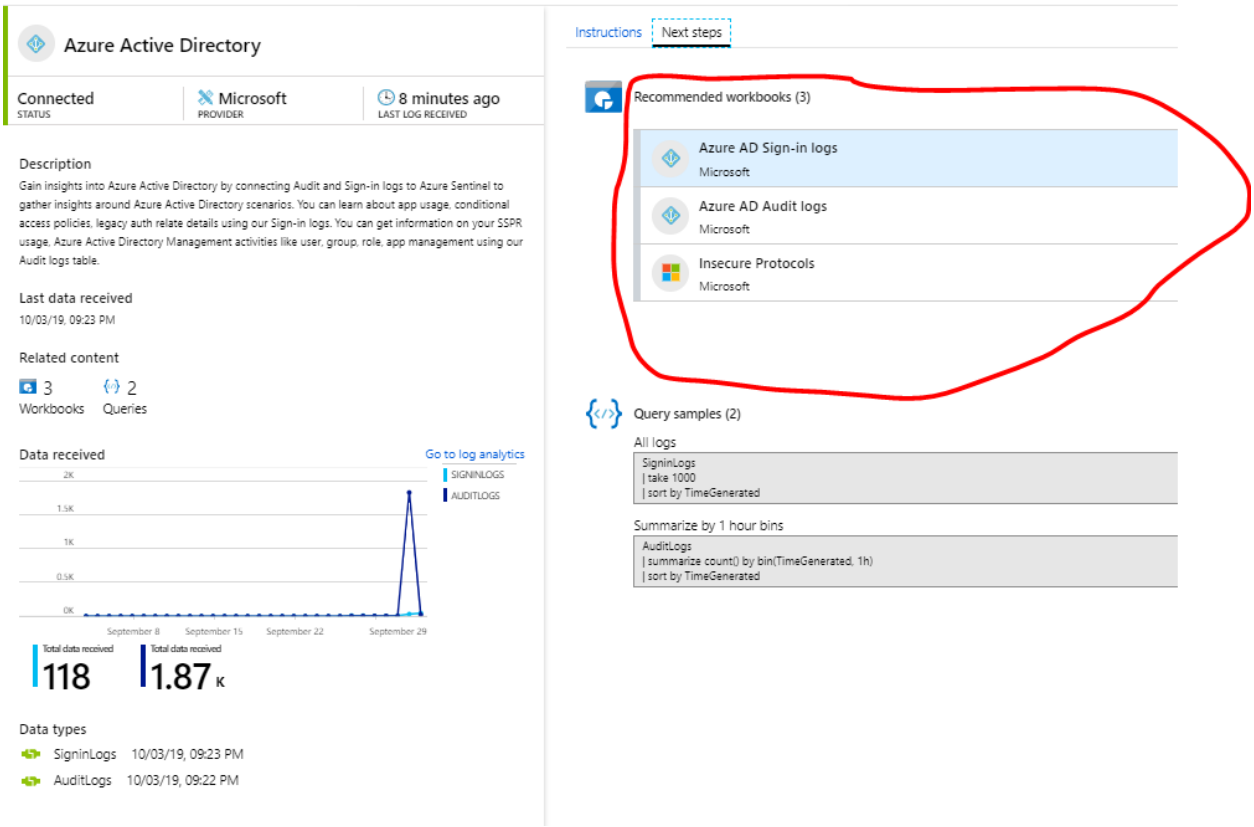
1. From the Sentinel Overview page, click on **Workbooks**.



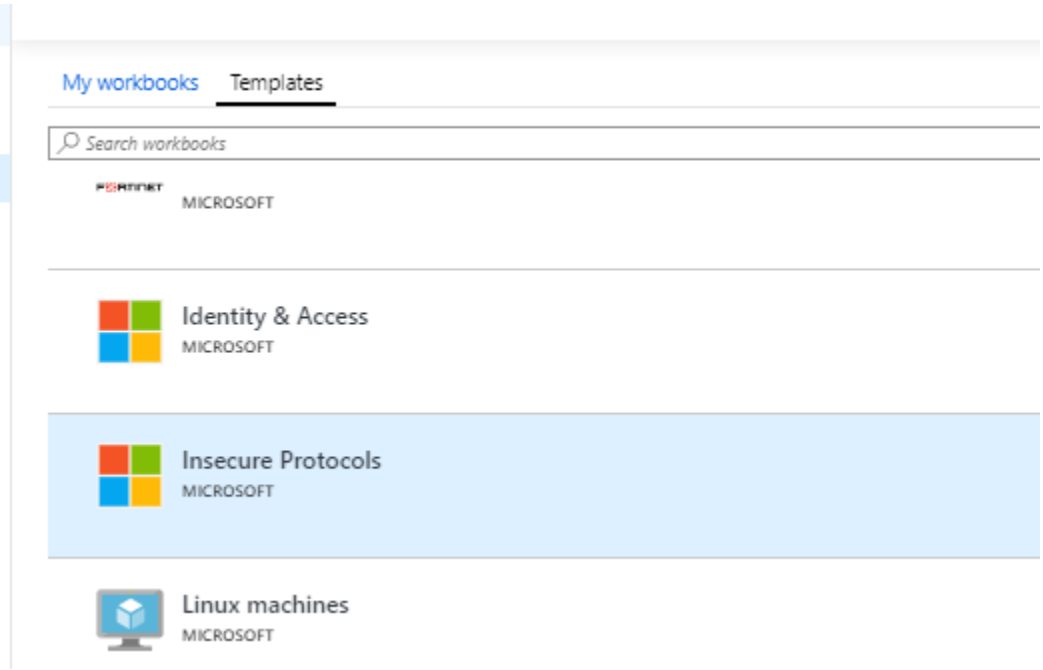
2. On the next blade that appears, click **Templates**.



3. Take a moment to browse the list of built-in Workbooks such as **Azure Activity**, **Azure AD Sign-in logs**, **Office 365**, and third-party Workbooks for products from vendors like Symantec and Palo Alto. It is important to understand that **Workbooks** are typically used with **Data connectors**. In fact, when you add a Data connector, you typically receive a **Recommended Workbook**, as in this example:

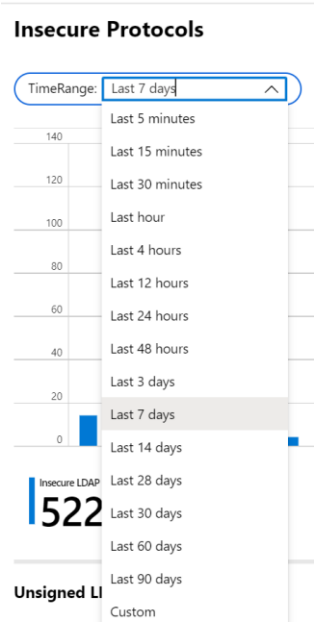


4. Click on **Insecure Protocols**. In the lower right-hand corner, click **View template** in order to bring up the workbook.



5. The **Insecure Protocols** workspace visually represents Security log data from on-premise or IaaS Active Directory Domain Controllers as well as Legacy Authentications taking place against Azure AD. Specifically, it represents data detailing which insecure protocols are currently in use and their data flows throughout the estate.

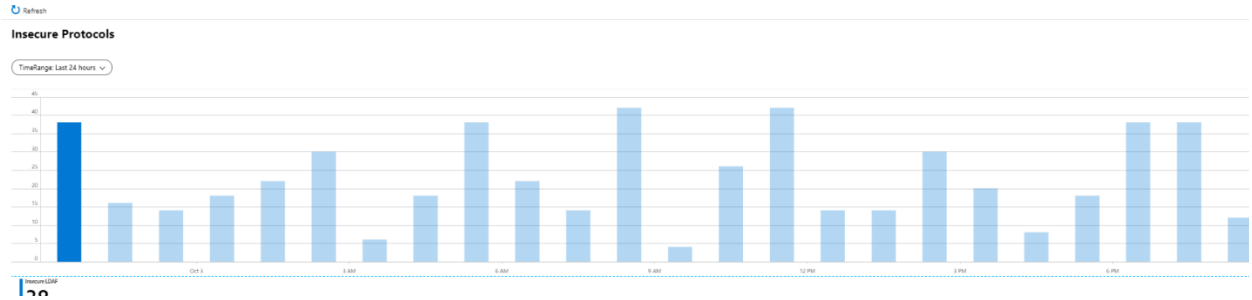
6. In the upper left-hand corner, set the TimeRange to last 7 days:



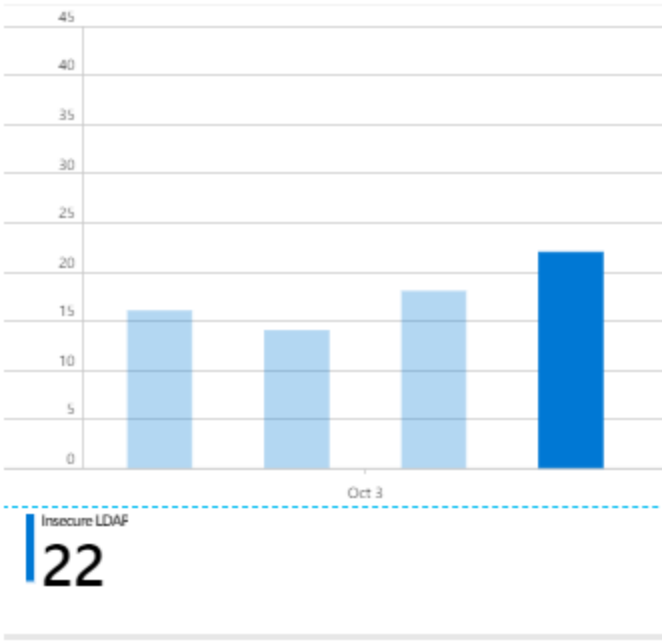
7. Take a moment to browse the workbook. This workbook addresses the problem of organizations that wish to remove insecure protocols (like NTLMv1, SMB1, wDigest and weak Kerberos ciphers) but are unable to do so for fear of breaking critical business systems. They need to remediate the sources before they can disable the protocols. And for that they need data visualization.

Step 2: Dive deeper in the Insecure Protocols Workbook to Investigate Security Data

1. Let's take a look at the overview graph titled **Insecure Protocols**.



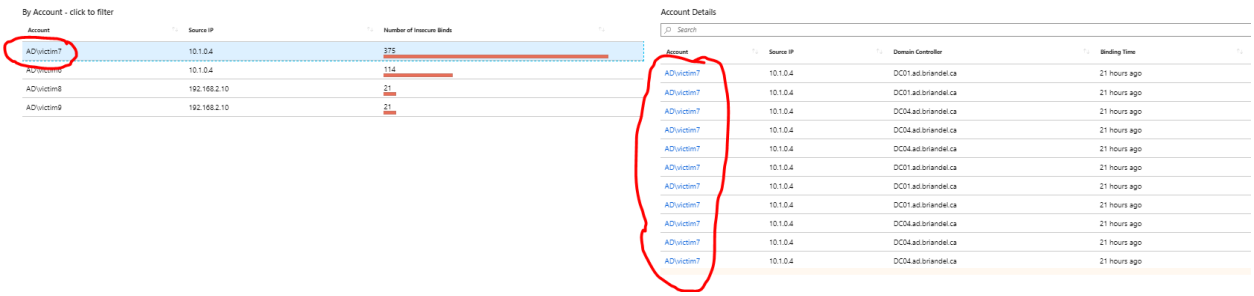
2. On the bar chart, we can immediately see something important. Out of all the Insecure Protocols that could be running in this estate, only Insecure LDAP shows up. That tell us that, so long as this trend continues, we could safely turn off other protocols such as NTLMv1 and SMB1. Click on various bars to conform that 100% of the Insecure Protocols in the environment are Insecure LDAP.



3. We can also drop down to the Unsigned LDAP section of the page (directly below the Overview) for more insightful data representations.

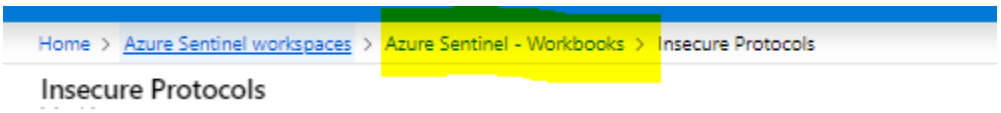
Step 3: Understand and Manipulate the Insecure Protocol Queries Behind the Workbook

1. On the **Insecure Protocols** workbook, scroll down to **Unsigned LDAP**. This was the most active Insecure Protocol in the estate, so it makes sense to examine it closely.
2. Note that by clicking on an account on the left, you may filter the Account Details to the right.

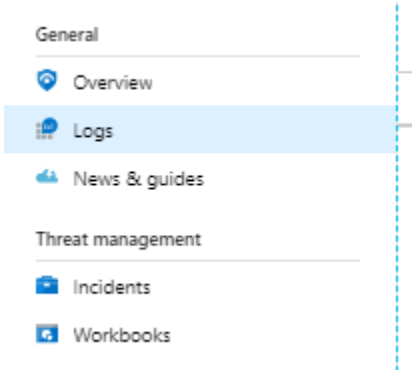


Step 4: Delve deeper in the Data Represented in the Workbook

1. Sometimes we will need to assist our customers in delving deeper into a particular data set or sets. In order to do that, we will often want to drill down closer to the actual data collected by Sentinel (Log Analytics on the back end).
2. Click on the **Azure Sentinel - Workbooks** at the top of the page



3. Click on **Logs**

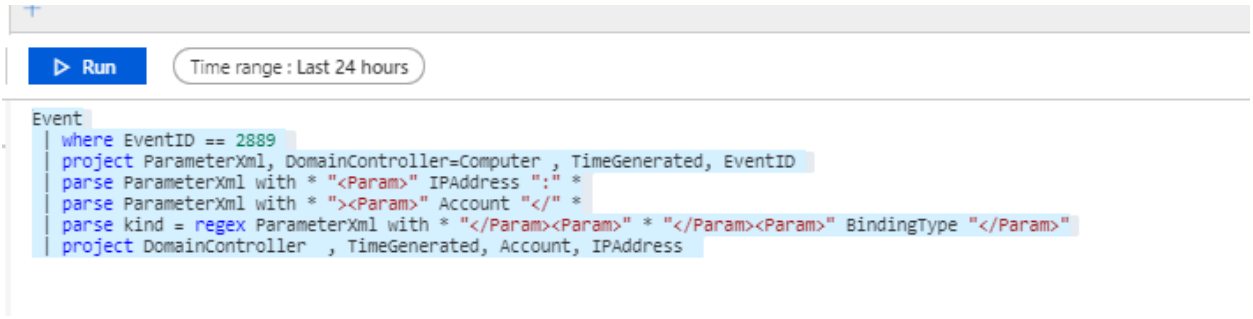


4. Let's delve deeper into Unsigned LDAP Traffic using a query that is doing some extractions as well as RegEx matching to put the data in an acceptable format.
5. For example, maybe we need to know the Domain Controller being targeted by the Insecure LDAP binds. We can easily do that in the following steps.
6. In the query window, copy the following query:

```
Event
 | where EventID == 2889
 | project ParameterXml, DomainController=Computer , TimeGenerated, EventID
 | parse ParameterXml with * "<Param>" IPAddress ":" *
```

```
 | parse ParameterXml with * "><Param>" Account "</" *
 | parse kind = regex ParameterXml with * "</Param><Param>" * "</Param><Param>"
BindingType "</Param>"
 | project DomainController  , TimeGenerated, Account, IPAddress
```

And click **Run**



In the output, Click one of the chevrons to the left side of the screen in order to examine one of these logs.

7. This type of data can help fill in the missing pieces to our story. Depending on which log you clicked, you can gain the missing information. In the example below, we see that victim7 made an Insecure LDAP bind from IP 10.1.0.4 and against DC04. Your data will look different.

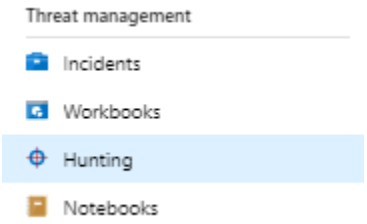
Exercise 4: Proactively Investigate Potential Threats, Misconfigurations, and Suspicious Activities with KQL Queries

Task 1: Navigate to Sentinel Hunting and Run all queries

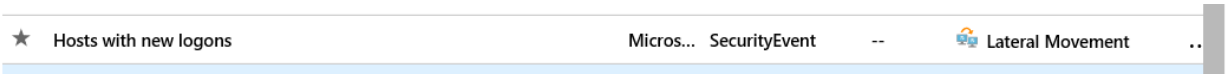
As in Task 1, we will be proactively hunting for Indicators of Attack in Sentinel. In that task we were doing so visually through the use of workbooks. Here, we will use built-in KQL queries to identify signs of potential problems.

1. From the Azure Sentinel homepage, click on **Hunting**.

2. In the pane that opens, click on Hunting.



3. Examine the Queries in the middle pane. These are built-in KQL queries provided to help you hunt for adversaries in your environment. Pay attention to the description and also special attention to the *Tactics* column. You may learn more about a particular tactic by mousing over it for a description. This column helps you understand which part of the attacker kill chain you may insert yourself in with the query. For example, by examining *Hosts with new logons*, you may be able to detect *Lateral Movement* in the environment.



4. By running all the built-in queries, you have a quick indicator of specific areas of concern. These may form a ready-made map of where your time should be spent hunting in any given point in time.

5. Click **Run all queries**.

6. Look at the *Results* column. Note that several of these queries have returned results. These require further investigation. You will want to help your customers pay special investigative attention to the results of queries like *New processes observed in last 24 hours*, and *uncommon processes – bottom 5%*. These can be indicators of compromise that are often missed.

Task 2. Run Several Single Hunting Queries and Examine the Output in Detail

In this Task, we will run several single queries to hunt for indicators of compromise.

Step 1: Run the *New processes observed* Query and Examine the Output

1. Click on the query *New processes observed in the last 24 hours*.
2. This brings up information about this Hunting query on the right side of the screen.

New processes observed in last 24 hours

Microsoft	238	SecurityEvent
Provider	Results	Data Source

DESCRIPTION

These new processes could be benign new programs installed on hosts; however, especially in normally stable environments, these new processes could provide an indication of an unauthorized/malicious binary that has been installed and run. Reviewing the wider context of the logon sessions in which these binaries ran can provide a good starting point for identifying possible attacks.

CREATED TIME

2/15/2019

QUERY

```
let ProcessCreationEvents=() {
let processEvents=SecurityEvent
| where EventID==4688
| where TimeGenerated >= ago(30d)
| project TimeGenerated, ComputerName=Computer,AccountNa
processEvents};
```

View query results >

TACTICS

Execution

The execution tactic represents techniques that result in execution of adversary-controlled code on a local or remote system.

read more on mitre.com

Run Query | View Results

3. Optional: for background information, you may click on *read more on mitre.com*. This is an excellent knowledge base of adversary tactics and techniques that have been documented in the wild.
4. Click **Run Query** and then click **View Results**. This launches the *Logs* page, pre-populates the KQL query, and displays the results. Your data will be different that the example below.

HostCount	FileName
1	C:\Program Files\Microsoft Monitoring Agent\Agent\Health Service State\Monitoring Host Temporary Files 3\10
1	C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_1.297.467.0.exe
1	C:\Windows\System32\PING.EXE
2	C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_1.297.466.0.exe
1	C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_1.297.471.0.exe
1	C:\Windows\System32\notepad.exe
1	C:\Windows\System32\Dism.exe
1	C:\Windows\Temp\F8A847DF-5867-4579-B7C2-D7D81B1D00E7\DismHost.exe
1	C:\Windows\System32\wbem\unsecapp.exe
1	C:\Windows\ImmersiveControlPanel\SystemSettings.exe
1	C:\Windows\System32\SystemSettingsAdminFlows.exe
1	C:\Windows\System32\LocationNotificationWindows.exe
2	C:\Windows\System32\gpupdate.exe
1	C:\Windows\System32\CredentialUIBroker.exe
1	C:\Windows\Temp\276785E4-287A-4BDA-BF98-A9DED1117730\DismHost.exe
1	C:\Windows\System32\TokenBrokerCookies.exe
2	C:\Windows\System32\SettingSyncHost.exe
1	C:\Windows\System32\PickerHost.exe
1	C:\Windows\System32\unregmp2.exe

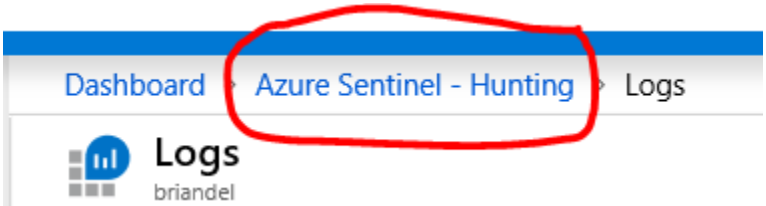
5. Briefly examine the table of results. Notice that it features *HostCount* and *FileName* columns of information. From the results, we can see that there have been a number of new processes launched across the estate in the last 24 hours.
6. This may be a sign of normal, benign programs being run. However, in normally stable environments, these new processes could provide an indication of an unauthorized/malicious binary that has been installed and

run. Look closely at the results as shown below.

Drag a column header and drop it here to group by that column

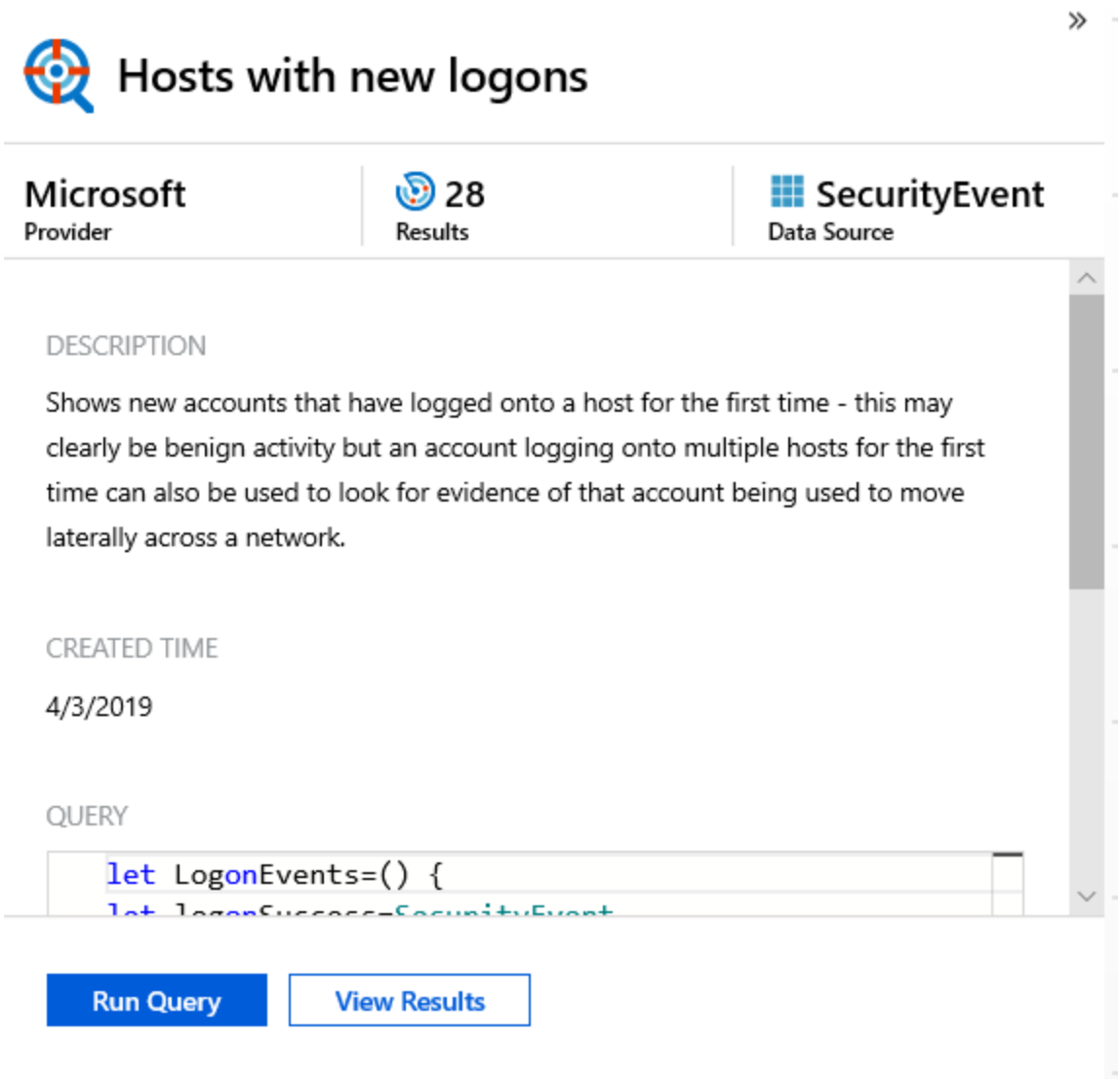
HostCount	FileName
1	C:\Program Files\Microsoft Monitoring Agent\Agent\Health Service State\Monitoring Host Temporary Files 36\437\pmfexe.exe
1	C:\Scripts\Rename\902262316.exe
1	C:\Windows\System32\wscript.exe
1	C:\Scripts\Rename\1731236076.exe
1	C:\Scripts\Rename\2059605237.exe
1	C:\Scripts\Rename\593164151.exe
1	C:\Windows\System32\nslookup.exe
1	C:\Scripts\Rename\1144691243.exe
1	C:\Scripts\Rename\295513871.exe
1	C:\Scripts\Rename\969440667.exe
1	C:\Scripts\Rename\215221449.exe
1	C:\Scripts\Rename\544185390.exe
1	C:\Scripts\Rename\863796238.exe
1	C:\Scripts\Rename\1025880745.exe
1	C:\Scripts\Rename\887051908.exe
1	C:\Scripts\Rename\1731803675.exe
5	C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_1.299.883.0.exe
1	C:\Scripts\Rename\1308385559.exe
1	C:\Scripts\Rename\1176072566.exe
1	C:\Scripts\Rename\1430479002.exe
1	C:\Scripts\Rename\1276535405.exe
1	C:\Scripts\Rename\2009150354.exe

7. Right away we might notice some suspicious activity. For purposes of our lab, let's focus on the highlighted process above and the ones like it. These are processes we do not normally recognize – and they may very well be nefarious. Certainly, we need to investigate further.
8. Note that we have identified a protentially dangerous process that otherwise would have gone unnoticed in the environment. This can be an early indicator of compromise – and early detection is key to being able to create a meaningful response.
9. Click on **Azure Sentinel – Hunting** at the top to return to he Hunting section.



Step 2: Run the *Hosts with new logons* Query and Examine the Output

1. Click on the query *Hosts with new logons*.
2. This brings up information about this Hunting query on the right side of the screen.



3. Notice the description.
4. Scroll down to view the query and the tactics.

QUERY

```
    | union logonSuccess
    };
    LogonEvents
    | where TimeGenerated >= ago(30d)
    | where ActionType == 'Logon'
    | summarize count() by ComputerName, AccountName
```

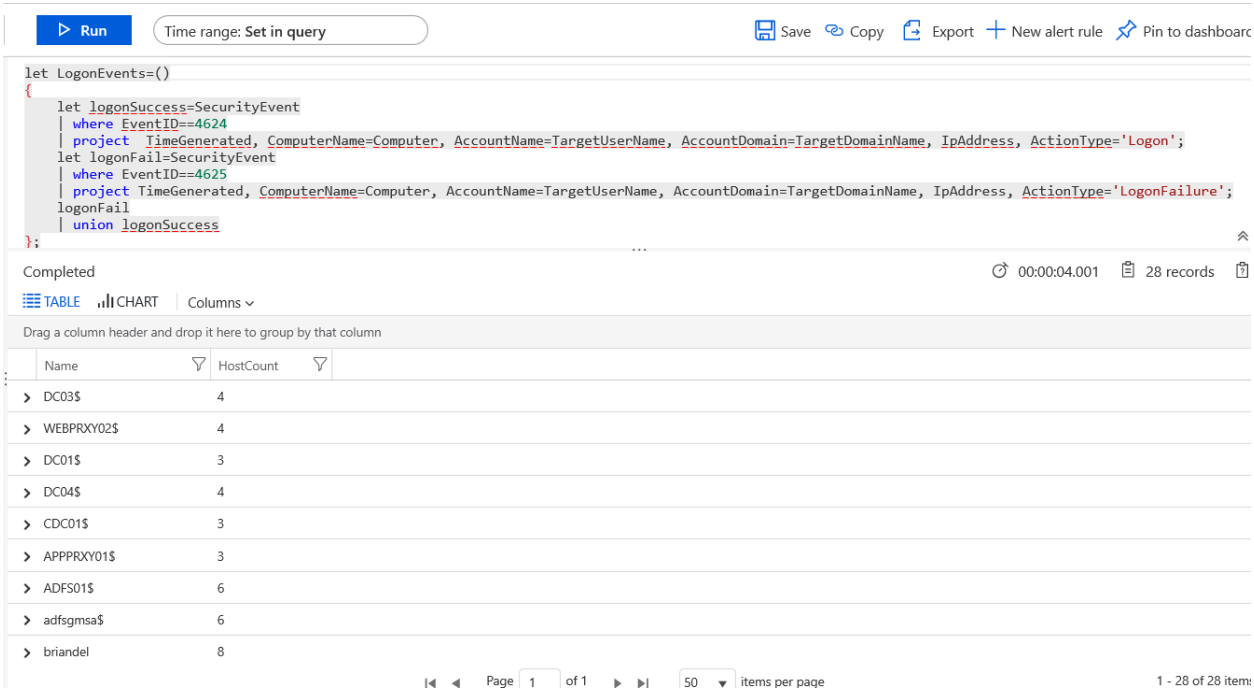
View query results >

TACTICS

Lateral Movement

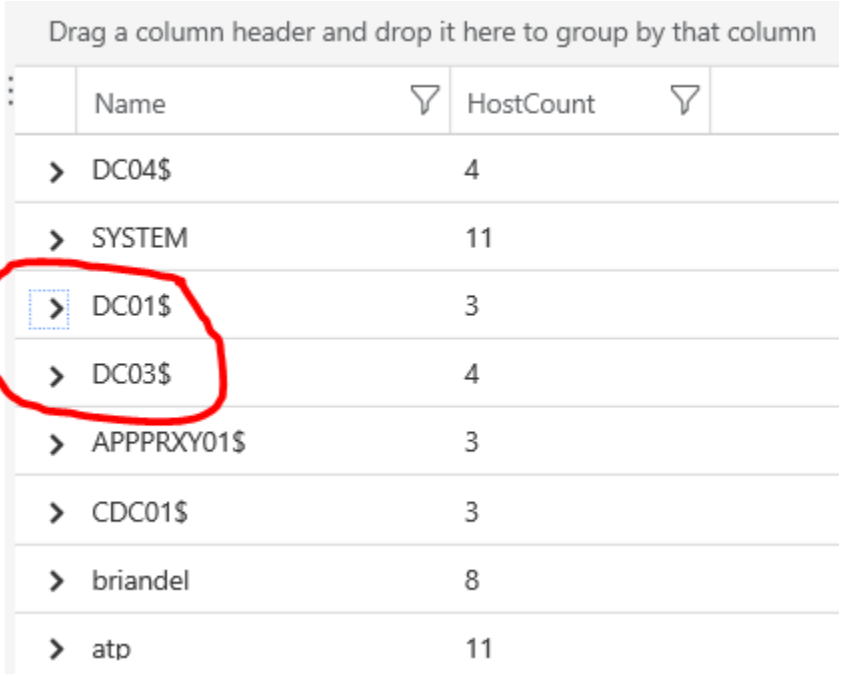
Lateral movement consists of techniques that enable an adversary to access and control remote systems on a network.
read more on mitre.com

5. Optional: for background information, you may click on *read more on mitre.com*. This is a knowledge base of adversary tactics and techniques that have been documented in the wild.
6. Click **Run Query** and then click **View Results**. This launches the *Logs* page, pre-populates the KQL query, and displays the results.



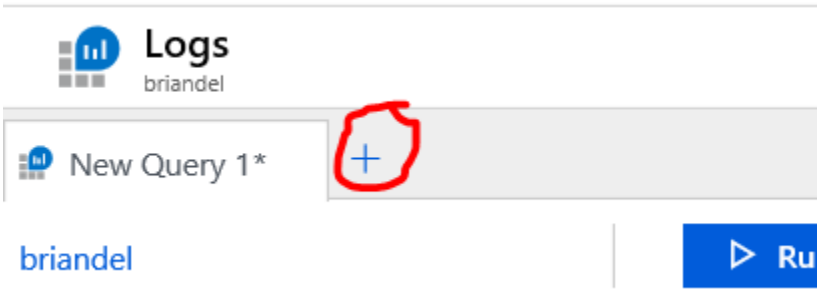
7. Briefly examine the query. The query is asking for successful and failed logon events, creating a union, setting some time parameters, joining several tables, and setting some output parameters.

8. Briefly examine the table of results. Notice that it features the *projected* attributes *Name* and *Hostcount*. From the results, we can see that a subset of computers have indeed received recent (new) logon activity.
9. Your output may look slightly different, but right away we might notice some suspicious activity. Multiple domain controllers (DCs) have received new logons. This bears further investigation. New logons to DCs are typically rare; DCs also represent high-value attack targets.



Task 3: Craft and Execute a Hunting Query to Investigate the New DC Logons

1. In this Task, we want to investigate the accounts that have logged onto the DCs in the last day.
2. Click the **Plus** (+) sign next to *New Query 1*.

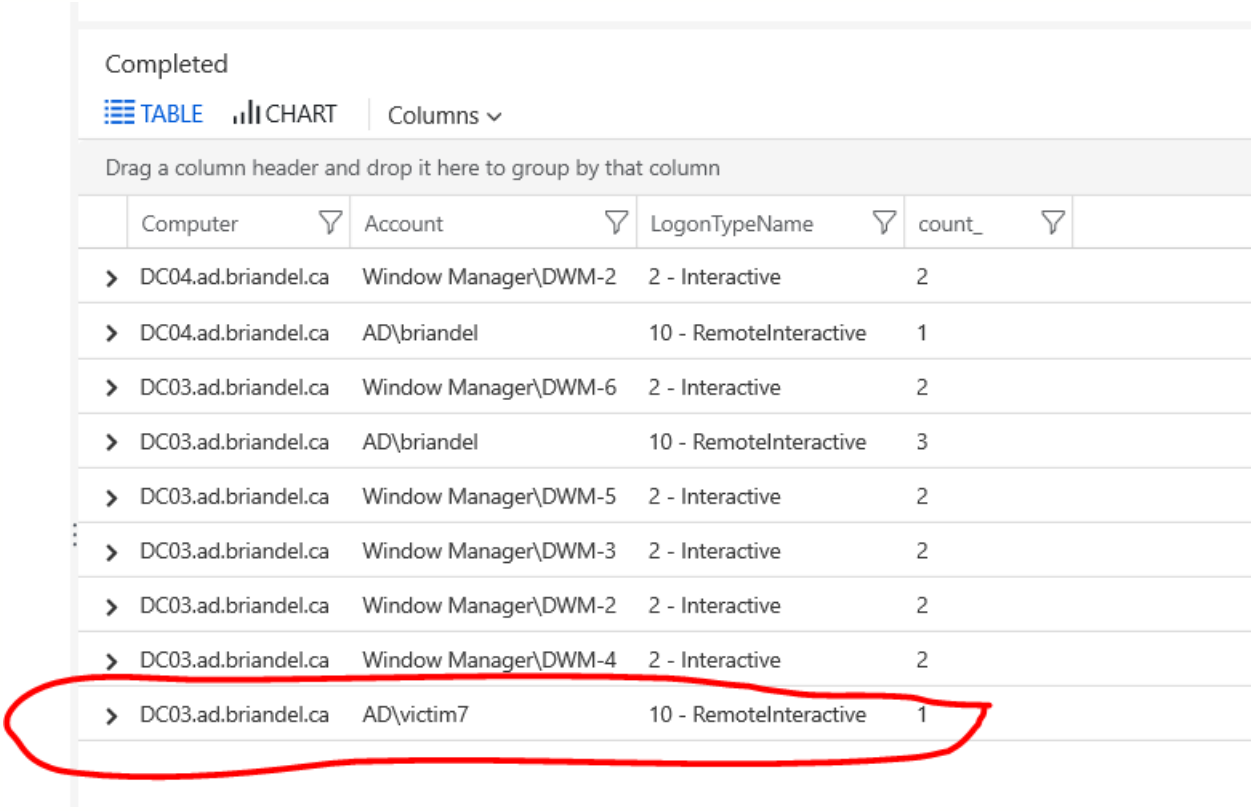


3. In the new query window, copy the text of this query to investigate the accounts that have logged onto the DCs:

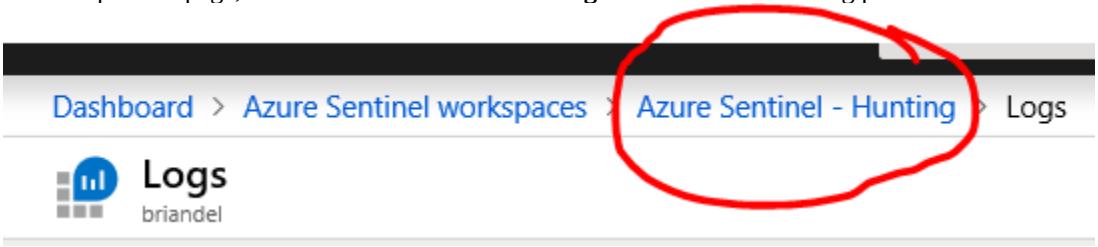
```
SecurityEvent
| where EventID == 4624
```

```
| where TimeGenerated >= ago(3d)
| where Computer startswith "DC01" or Computer startswith "DC03" or Computer startswith "DC04"
| where Account !endswith "$" and Account != @"NT AUTHORITY\LOCAL SERVICE" and Account !=
@"NT AUTHORITY\SYSTEM"
| summarize count() by Computer, Account, LogonTypeName
```

4. Examine the output. Here you would hunt for any Accounts that you would not expect to log onto DCs. For example, an account that did not belong to a member of the IT staff (and specifically an Active Directory administrator) would throw a red flag to take action. You could also additionally hunt for discrepancies in logon times, for example, by modifying the query in various ways.\
5. In the query results, do you see any accounts that you would flag with your customer? In the example below, we see an account called *victim7* having logged into DC03. That is definitely an indicator of compromise. We would need, in this case, to take action against both the account and potentially inside of Active Directory. We could even be in an incident response type of scenario. Your data will differ slightly from the example shown below.



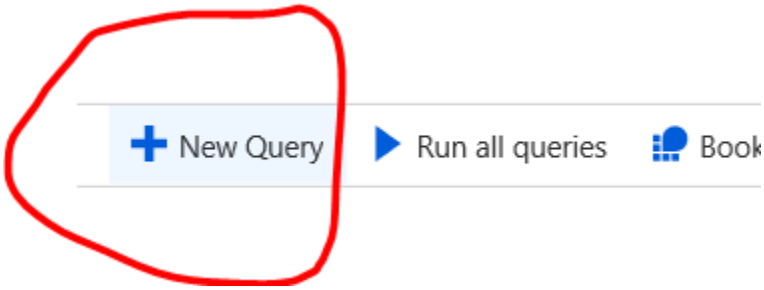
6. At the top of the page, click **Azure Sentinel – Hunting** to return to the Hunting pane.



Task 4: Craft and Execute a Unique Hunting Query to Investigate a Distinct Security Concern

In this Task, you will pick up where the Insecure Protocols Workbook left off. You will craft a query to hunt for the use of Legacy Protocols against Azure AD. One source of legacy protocols is often, but not exclusively, the use of mail browsers accessing Office 365/Exchange Online. In reality, you could make this a workbook, or an extension of the Insecure Protocols workbook. But in the interest of time, here you will craft this activity as a Hunting Query.

1. From within the Hunting section of the portal, Click on the **+ New Query** button



2. On the *Create custom query* page, Fill in the following information

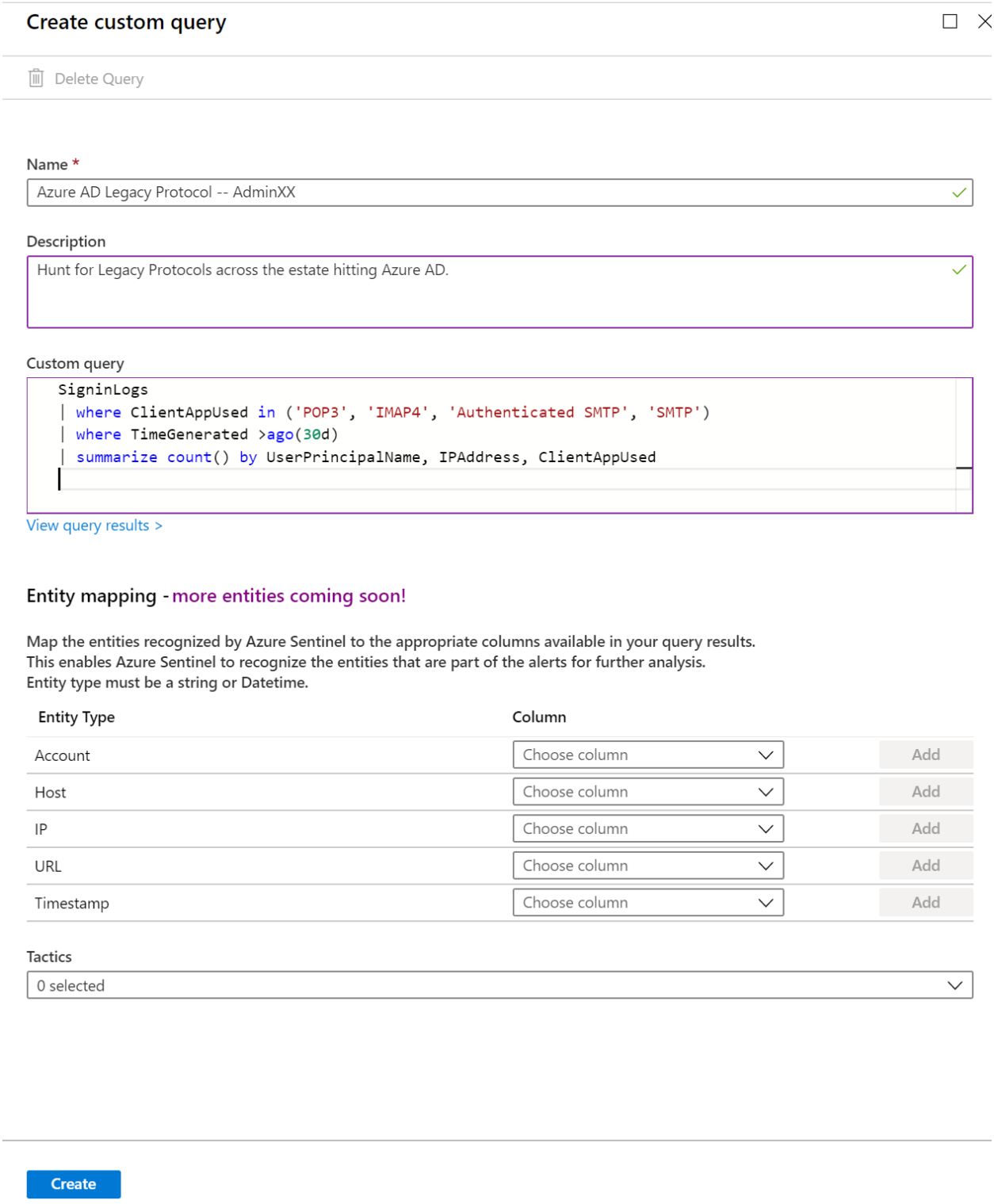
   Name: Azure AD Legacy Protocol – Admin[Number]

   *[Where Admin1 is the name of your logon. For Example, if my logon is Admin 43, then the Name field would read **Azure AD Legacy Protocol – Admin43**.]*

   Description: Hunt for Legacy Protocols across the estate hitting Azure AD.

   Custom query:

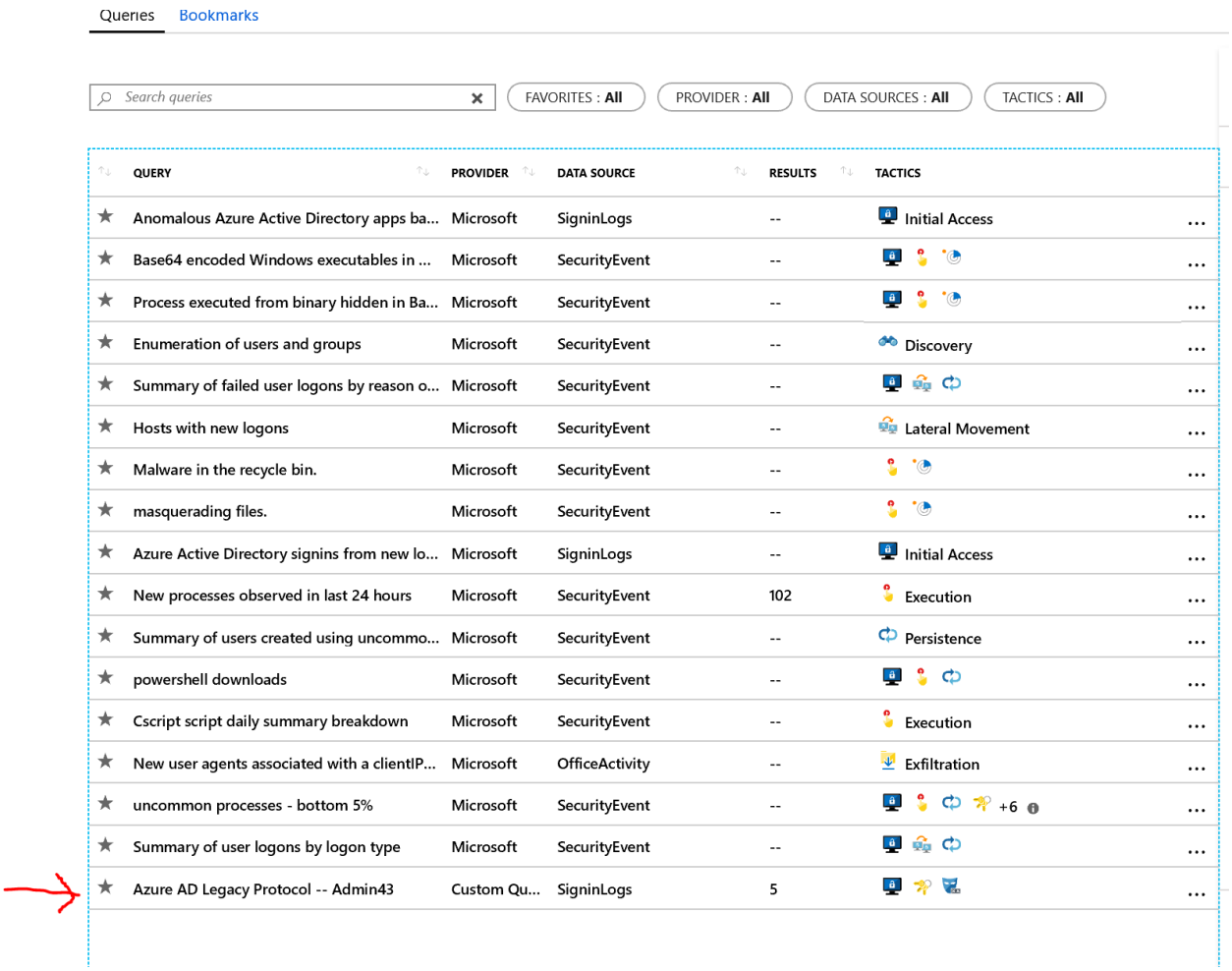
```
SigninLogs
| where ClientAppUsed in ('POP3', 'IMAP4', 'Authenticated SMTP', 'SMTP')
| where TimeGenerated >ago(30d)
| summarize count() by UserPrincipalName, IPAddress, ClientAppUsed
```

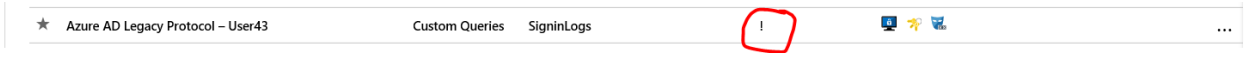


In this example, my user name is User43.

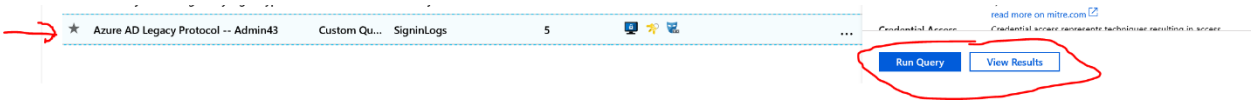
3. Leave the **Entity mapping** fields blank.

4. Add the Tactics that you think best match the attacker tactics you are hunting for. For example, **Initial Access, Privilege Escalation** and **Credential Access** are all risks of using Legacy Protocols.
5. Click **Create**.
6. On the main Hunting page, examine your new query.



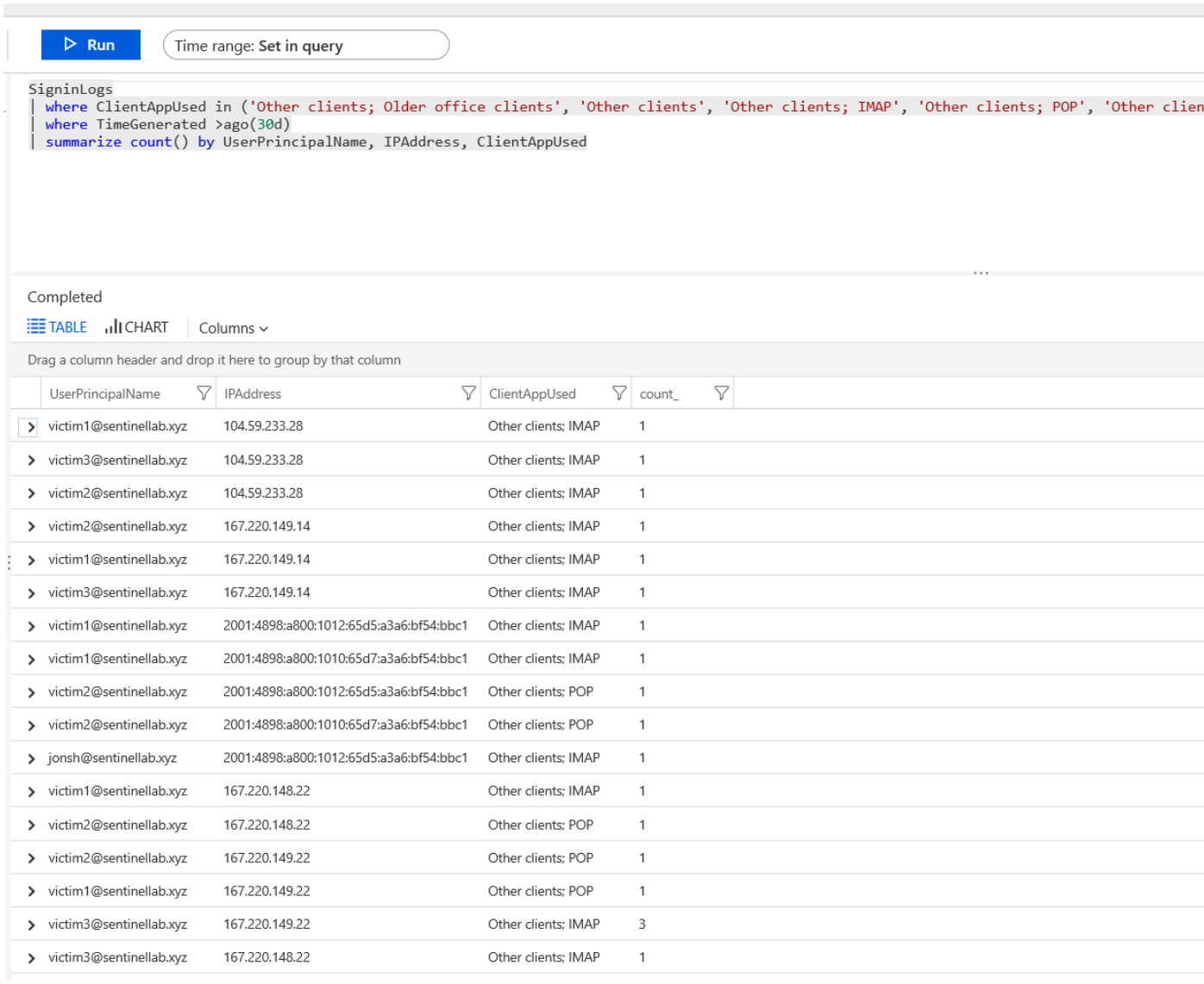
7. Click **Run all queries** to include your new query in the overall Hunting experience.
8. Note: If you receive an exclamation point for the result, do not worry. This is an intermittent known issue and should be ignored for now.
9. 

10. Click on your new query, click **Run Query** and then click **View Results**.

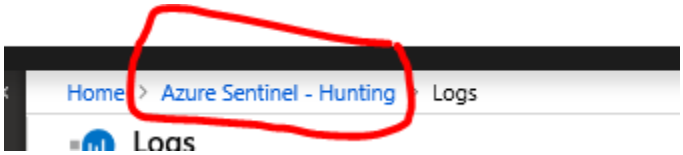


11. On the following page, examine the output. You are now able to Hunt for Users and IP Addresses that are using Legacy Protocols to access Azure AD; you also know the actual protocol used

(IMAP, POP, etc.). Finally, the advantage of creating a *Hunting Query* is that *anyone* in your company can now easily reuse your work to hunt for misconfigurations or indicators of attack.



12. At the top, click **Azure Sentinel – Hunting**.



Exercise 5 (optional): Proactively Investigate a Threat using a Defined Set of Operations with a Jupyter Notebook

Important Note: The following section on Jupyter Notebooks may be optionally completed if you have time. Bear in mind that the technology is marked as Preview and may therefore be buggy and/or challenging to work with.

If experience an issue running the Jupyter Notebooks, this is probably due to the Preview status of the feature. In this case, we recommend you read through the remainder of the lab without executing the steps. This will aid in understanding how Juipyter Notebooks are intended to work.

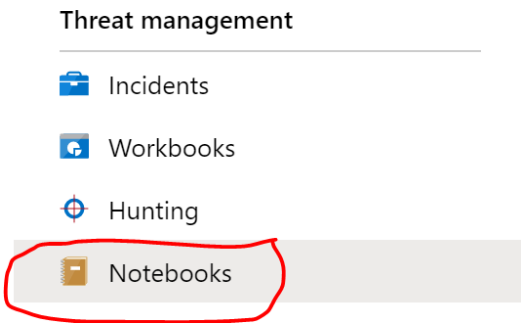
Task 1: Launch Notebooks, clone the Sentinel Notebooks from GitHub, and launch the Appropriate Notebook

1. **Background** (from https://docs.microsoft.com/en-us/azure/sentinel/notebooks) - *The foundation of Azure Sentinel is the data store; it combines high performance querying, dynamic schema, and scales to massive data volumes. The Azure Sentinel portal and all Azure Sentinel tools use a common API to access this data store. The same API is also available for external tools such as Jupyter notebooks and Python. While many common tasks can be carried out in the portal, Jupyter extends the scope of what you can do with this data. It combines full programmability with a huge collection of libraries for machine learning, visualization, and data analysis. These attributes make Jupyter a compelling tool for security investigation and hunting.*

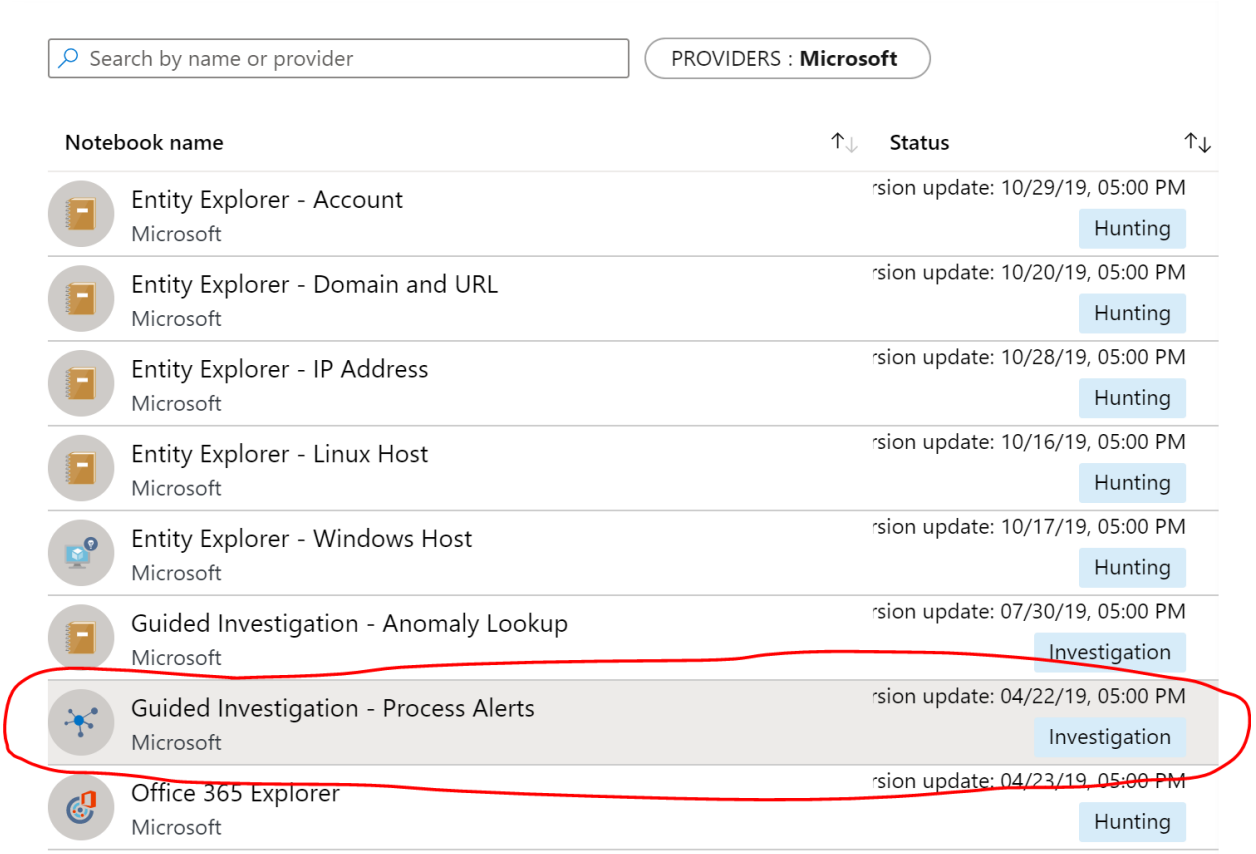
   In this Exercise, we will delve deeper into one of the query results that we saw earlier. Recall in Exercise two that we saw an anomalous (and unexpected) login from one of the lab *victim* accounts. In real life, SOC analysts at our customers would need to delve further into this alert that we proactively discovered while doing our Hunting work.

   Conceptually, Jupyter notebooks pick up where Hunting queries leave off. As a general rule, Hunting queries are useful when surveying a large swatch of an organization's estate. Typically, this is an excellent starting point. When we need to delve further to investigate a larger set of processes or a perhaps multiple entities, we will often turn to Jupyter notebooks. As a rule of thumb, the average person can remember around seven queries. For anything more complex, you'll want to set up a repeatable, detailed process in a Jupyter notebook.

2. From the menu on the left-hand side or the page you are on, click **Notebooks**.



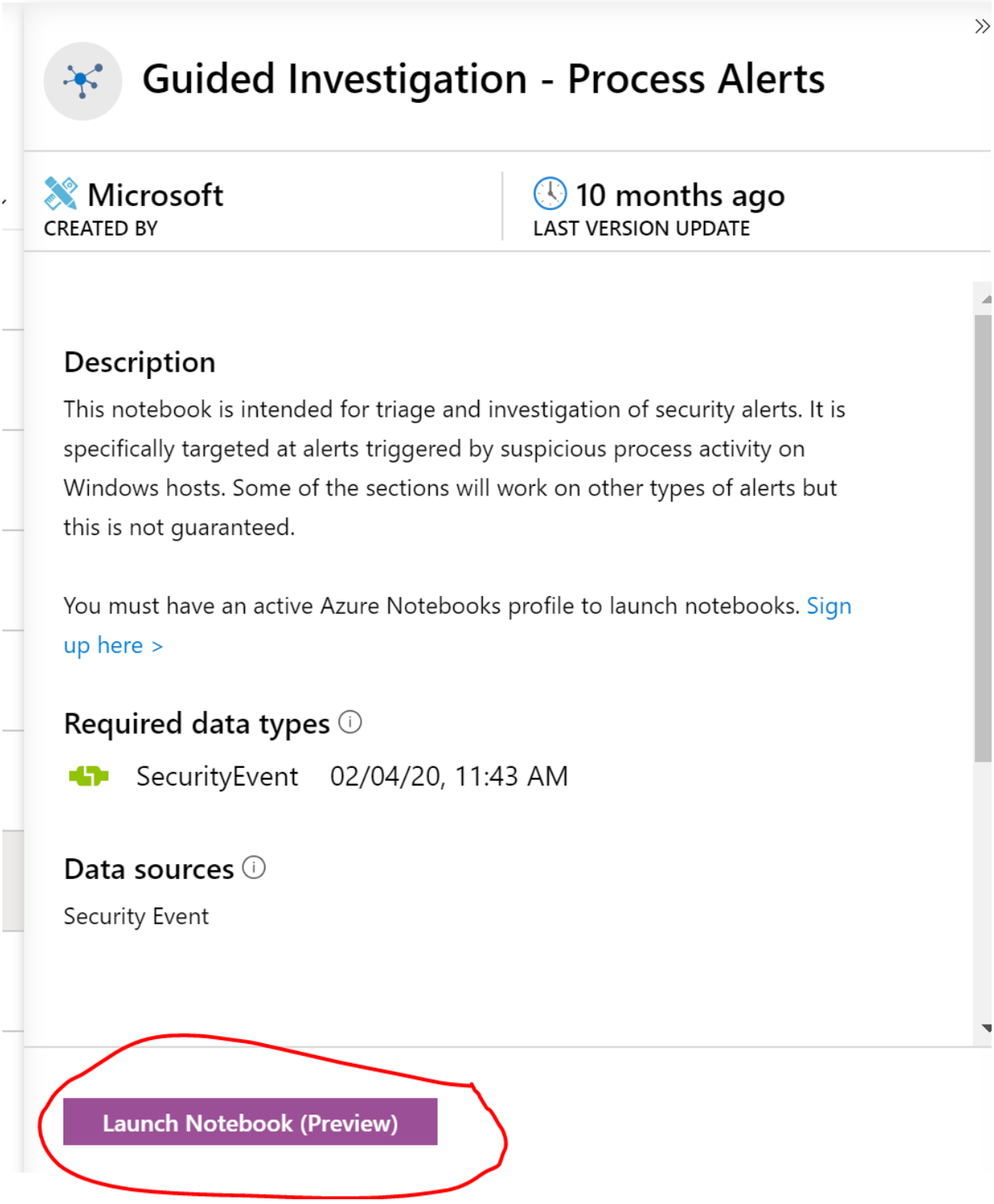
3. Click on the Notebook name **Guided Investigation – Process Alerts**



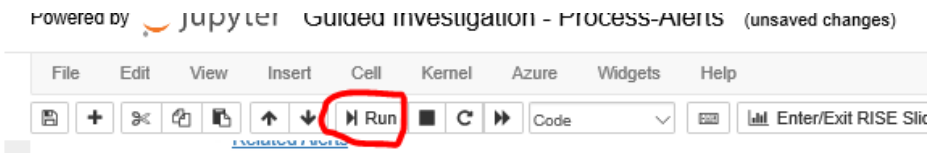
4. In the right-hand pane, click **Launch Notebooks (Preview)**

LEARN MORE

To launch, create your Azure Notebooks profile



5. This opens up a new tab. For Authentication, choose your Admin account in Sentinel lab from **Pick an account**. You may or may not be prompted for authentication.
6. The next page clones the repository from GitHub and, after a few moments, brings you into the Jupyter Notebook. Wait a moment while the Notebook finishes starting.
7. Take a moment to familiarize yourself with the controls. Typically throughout this exercise, you will click on a box of code and then click the **Run** button at the top. For example, to Install Packages, I would click the block of code below the *Install Packages* header. A box would appear around the block of code and then I would click **Run**.
8. This is an example only:



Contents

Setup

1. Make sure that you have installed packages specified in the setup (uncomment the l
2. There are some manual steps up to selecting the alert ID. After this most of the note
3. Major sections should be executable independently (e.g. Alert Command line and H

Install Packages

The first time this cell runs for a new Azure Notebooks project or local Python environme

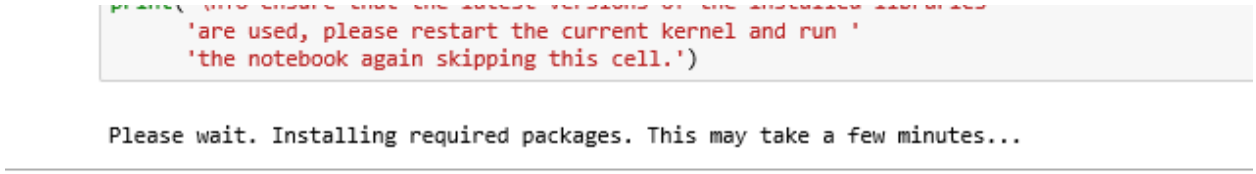
If you see any import failures (`ImportError`) in the notebook, please re-run this cell an

Note you may see some warnings about package incompatibility with certain packages.

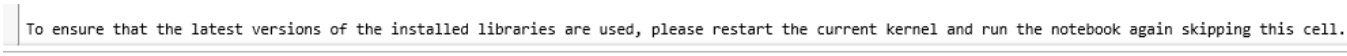
In [ ]:
```
import sys
import warnings

warnings.filterwarnings("ignore",category=DeprecationWarning)

MIN_REQ_PYTHON = (3,6)
if sys.version_info < MIN_REQ_PYTHON:
    print('Check the Kernel->Change Kernel menu and ensure that Python
    print('or later is selected as the active kernel.')
    sys.exit("Python %s.%s or later is required.\n" % MIN_REQ_PYTHON)

# Package Installs - try to avoid if they are already installed
try:
    import msticpy.sectools as sectools
    import Kqlmagic
    print('If you answer "n" this cell will exit with an error in order
    print('This error can safely be ignored.')
    resp = input('msticpy and Kqlmagic packages are already loaded. Do
    if resp.strip().lower() != 'y':
        sys.exit('pip install aborted - you may skip this error and con
    else:
        print('After installation has completed, restart the current ke
              'the notebook again skipping this cell.')
except ImportError:
    pass

print('\nPlease wait. Installing required packages. This may take a few
!pip install git+https://github.com/microsoft/msticpy --upgrade --user
!pip install Kqlmagic --no-cache-dir --upgrade --user

print('\nTo ensure that the latest versions of the installed libraries
      'are used, please restart the current kernel and run '
      'the notebook again skipping this cell.')
```

Task 2: Launch Prerequisite Processes and Authenticate in the Process-Alerts Notebook

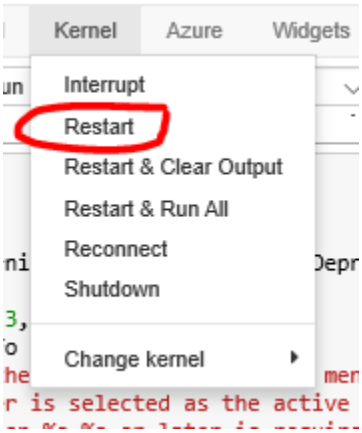
1. Click on that box of code (the one below *Install Packages* and then click **Run**.
2. Note the status message that appears:



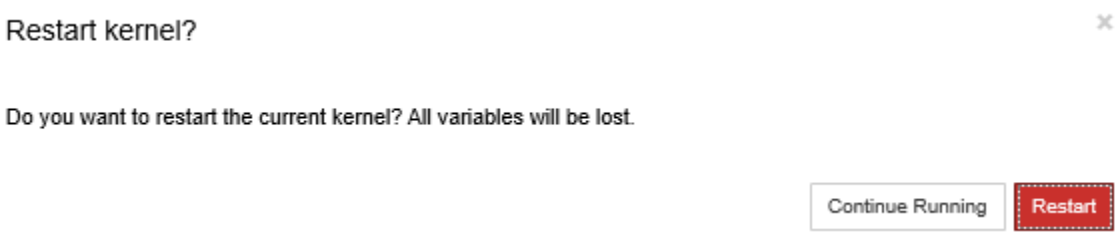
3. Wait for the code to finish. At the bottom of the message box, notice the following instructions:



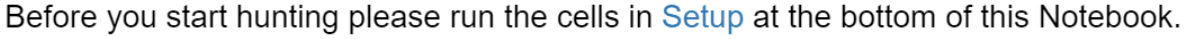
4. At the top, click the **Kernel** menu and then click **Restart**.



5. On the Pop-up that appears, click **Restart.**



6. Typically restarting the Kernel is a fast process. You may not even notice the status messages in the upper right of your screen. This is not a problem.
7. Read through the paragraph entitled **Hunting Hypothesis.** Once you understand the basic hypothesis, pay special attention to this disclaimer:



At the time of this writing, the hyperlink does not appear to work.

8. Manually scroll down to the bottom of the Jupyter Notebook to the section entitled **Setup Cell.**
9. Click on **Setup cell** and from the menu at the top choose **Insert** and **Insert Cell Above**.

In the cell that appears, type the following: !pip install msticpy --upgrade --user

10. Click on the cell you created and close **Run.**
11. Then click on **Setup cell** and click **Run.**
12. Navigate back up to **Initialization** and click **Run.**
13. Scroll down to *Get WorkspaceId***.** Click in the first block of code and click **Run.** The second block of code should then highlight. Click **Run** again. You should see the following output:

Read Workspace configuration from local config.json for workspace **briandel**

TENANT_ID: 020cd98f-1002-45b7-90ff-69fc68bdd027

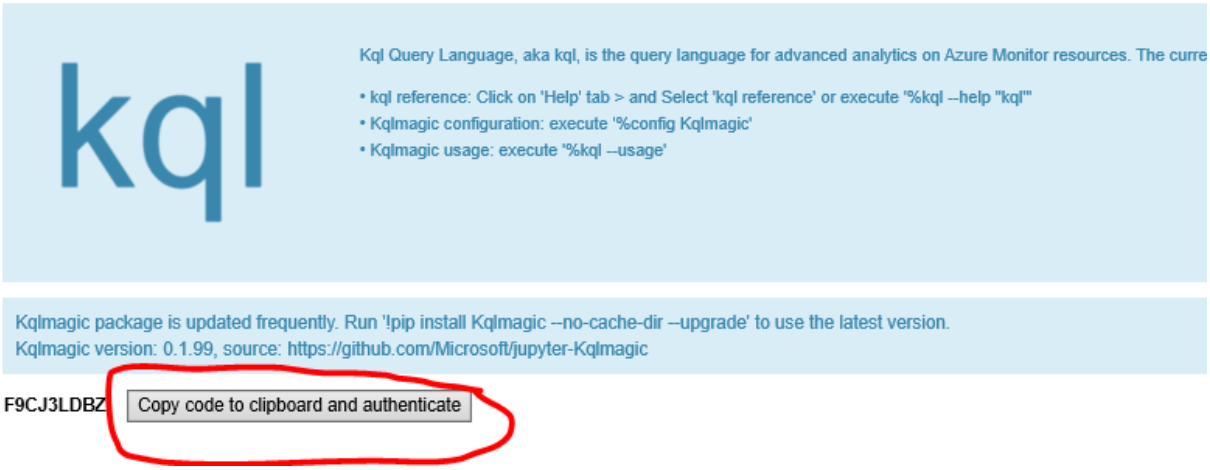
SUBSCRIPTION_ID: 397cdfbc-d326-412f-acdd-f3a40db4aaee

RESOURCE_GROUP: OI-Default-East-US

WORKSPACE_ID: 6ba2759c-1c00-4aa0-88e8-138379ea383c

WORKSPACE_NAME: briandel

14. Scroll down to *Authenticate to Log Analytics*, click the codeblock and click **Run**.
15. Wait until the following button appears:



16. When it does, click that button. In the screen that pops up, paste the code from your clipboard and click **Next**.
17. In the *Pick an account* window, choose the **AdminXX@sentinellab.xyz** account.

Task 3: Proactively Investigate Process Alerts by Leveraging the Power of the Jupyter Notebook

Step 1: Query for the Top Azure Sentinel Alerts in the Notebook

1. Click the code below *Get Alerts List* and click **Run.** Do not change any of the query time boundaries. Then click the code block below the query time boundaries and click **Run**.

```
alert_q_times = mas.QueryTime(units='day', max_before=20, before=5, max_after=1)
alert_q_times.display()
```

Set query time boundaries

Origin Date 7/8/2019 Time (24hr) 18:21:00.827095

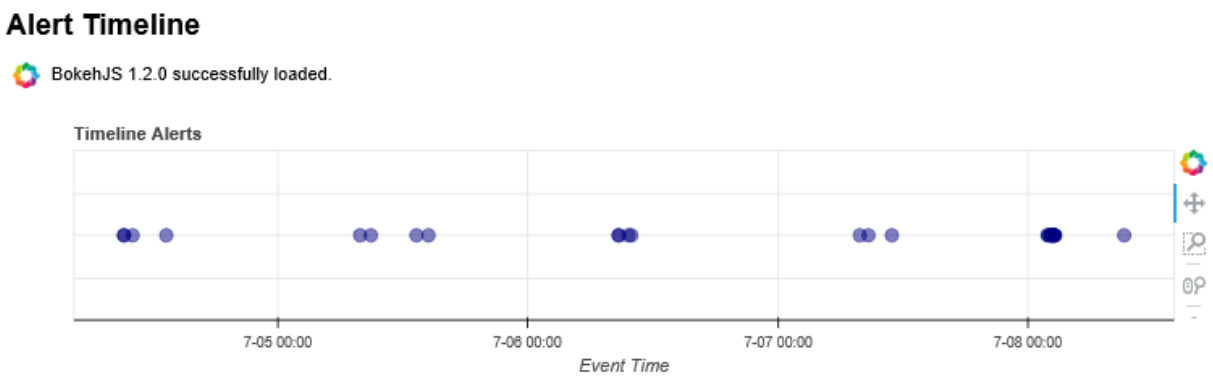
Time Range (day):

Query start time (UTC): 2019-06-07 18:21:00.827095

Query end time (UTC) : 2019-07-09 18:21:00.827095

```
alert_counts = qry.list_alerts_counts(provs=[alert_q_times])
alert_list = qry.list_alerts(provs=[alert_q_times])
print(len(alert_counts), ' distinct alert types')
print(len(alert_list), ' distinct alerts')
display(HTML('<h2>Alert Timeline</h2>'))
nbdisp.display_timeline(data=alert_list, source_columns = ['AlertName', 'CompromisedEntity'], title='Alerts', height=200)
display(HTML('<h2>Top alerts</h2>'))
alert_counts.head(20) # remove '.head(20)'' to see the full list grouped by AlertName
```

2. Examine the output in the graph entitled *Alert Timeline*. Here, a narrative begins to emerge. By mousing over the points on the timeline, we can see a number of alerts that have been collected from across the estate. For example, we might see potential reconnaissance activities like Administrative group enumeration, or potential Indicators of Attack like failed logons.



3. Examine the output below the label *Top alerts*. These are largely the same alerts as we saw in the graph, but in table format. One piece of information here is that we can gain an *alertCount*. For example, we see that *Local Administrators group members were enumerated* occurred 16 times in our data set below.

Top alerts

8]:

	AlertName	alertCount	firstAlert	lastAlert
0	Local Administrators group members were enumerated	16	2019-07-04 09:14:18	2019-07-08 09:12:52
1	Invalid Logon	8	2019-07-08 01:51:19	2019-07-08 02:30:21
2	Invalid Logon 3	3	2019-07-08 02:23:00	2019-07-08 02:33:03
3	Invalid Logon 2	1	2019-07-08 02:13:39	2019-07-08 02:13:39
4	Invalid Logon 4	1	2019-07-08 02:35:03	2019-07-08 02:35:03

Step 2: Choose Alerts and Intensify the Investigation

1. Click the code box under *Choose Alert to Investigate* and click **Run.**

```
In [ ]: get_alert = None
        alert_select = mas.AlertSelector(alerts=alert_list, action=nbdisp.display_alert)
        alert_select.display()
```

2. Examine the output noting that multiple alerts have returned.

```
2019-07-06 09:55:12  Local Administrators group members were enumerated (DC01) [id:2518398938879499999_
2019-07-07 07:50:25  Local Administrators group members were enumerated (DC04) [id:2518398149748769999_
2019-07-07 08:40:03  Local Administrators group members were enumerated (DC03) [id:2518398119966969999_
2019-07-07 10:55:12  Local Administrators group members were enumerated (DC01) [id:2518398038872129999_
2019-07-08 00:46:14  Invalid Logon () [id:e2c0e9c3-ff7c-439c-82ef-b5b9466772f4]
2019-07-08 01:40:15  Invalid Logon () [id:c04a793b-1596-4ca6-bd88-2b7c78271570]
2019-07-08 01:50:15  Invalid Logon () [id:d1502756-d7e7-4559-8326-a50270e7f860]
2019-07-08 01:55:15  Invalid Logon () [id:c1ca54aa-2563-41d5-9057-17f4224654e4]
2019-07-08 01:58:33  Invalid Logon 2 () [id:708d4274-b045-44ea-b22e-5932e81c4601]
2019-07-08 02:00:15  Invalid Logon () [id:695fb32c-ec31-4606-a13b-475c66c16e0d]
```

3. Click on one of the **Local Administrators group members were enumerated** alerts.
4. Examine the data that appears in the output window. This is *very useful* data detailing when the alert took place, on which entity, by whom, and using which protocol.
5. Our investigation is proceeding and there is still more data to gather.
6. Skip the section *Or paste in an alert ID and fetch it*.
7. Run the code block below *Extract properties and entities from Alert.* Note that the output is parsed into a concise and useful format in the table entitled *ExtendedProperies.*

ExtendedProperties:

	0
Compromised Host	DC03
User Name	AD\atp
Account Session Id	0x1d9827cea
Suspicious Process	-
Suspicious Process Id	0x0
vmname	DC03.ad.briandel.ca
Enumerated Group Domain name	Builtin
Enumerated Group Name	Administrators
Enumerating User Domain name	AD
Enumerating User Logon ID	0x1d9827cea
Enumerating User Name	atp
resourceType	Non-Azure Resource
ReportingSystem	Azure
OccuringDatacenter	Unknown

8. The *Entity counts* table may also be useful in determining the extent of the event in question.

Entity counts:
host: 2, account: 2, file: 1, process: 1, hostlogonsession: 1


```
{ 'DnsDomain': 'AD.BRIANDEL.CA',
  'HostName': 'DC03',
  'NTDomain': 'AD',
  'NetBiosName': 'DC03',
  'OMSAgentID': 'a99f70d7-0924-4333-906f-38a444866daf',
  'Type': 'host'}
{ 'Host': { 'DnsDomain': 'AD.BRIANDEL.CA',
            'HostName': 'DC03',
            'NTDomain': 'AD',
            'NetBiosName': 'DC03',
            'OMSAgentID': 'a99f70d7-0924-4333-906f-38a444866daf',
            'Type': 'host'},
  'IsDomainJoined': True,
  'LogonId': '0x1d9827cea',
  'NTDomain': 'AD',
  'Name': 'atp',
  'Sid': 'S-1-5-21-565363340-1337343146-2447627351-24668',
  'Type': 'account'}
{'FullPath': 'None-', 'Name': '-', 'Type': 'file'}
{ 'Account': { 'Host': { 'DnsDomain': 'AD.BRIANDEL.CA',
                         'HostName': 'DC03',
                         'NTDomain': 'AD',
                         'NetBiosName': 'DC03',
                         'OMSAgentID': 'a99f70d7-0924-4333-906f-38a444866daf',
                         'Type': 'host'},
               'IsDomainJoined': True,
               'LogonId': '0x1d9827cea',
               'NTDomain': 'AD',
               'Name': 'atp',
               'Sid': 'S-1-5-21-565363340-1337343146-2447627351-24668',
               'Type': 'account'},
  'ImageFile': {'FullPath': 'None-', 'Name': '-', 'Type': 'file'},
  'ProcessId': '0x0',
  'Type': 'process'}
{ 'Host': { 'DnsDomain': 'AD.BRIANDEL.CA',
            'HostName': 'DC03',
            'NTDomain': 'AD',
            'NetBiosName': 'DC03',
            'OMSAgentID': 'a99f70d7-0924-4333-906f-38a444866daf',
            'Type': 'host'},
```

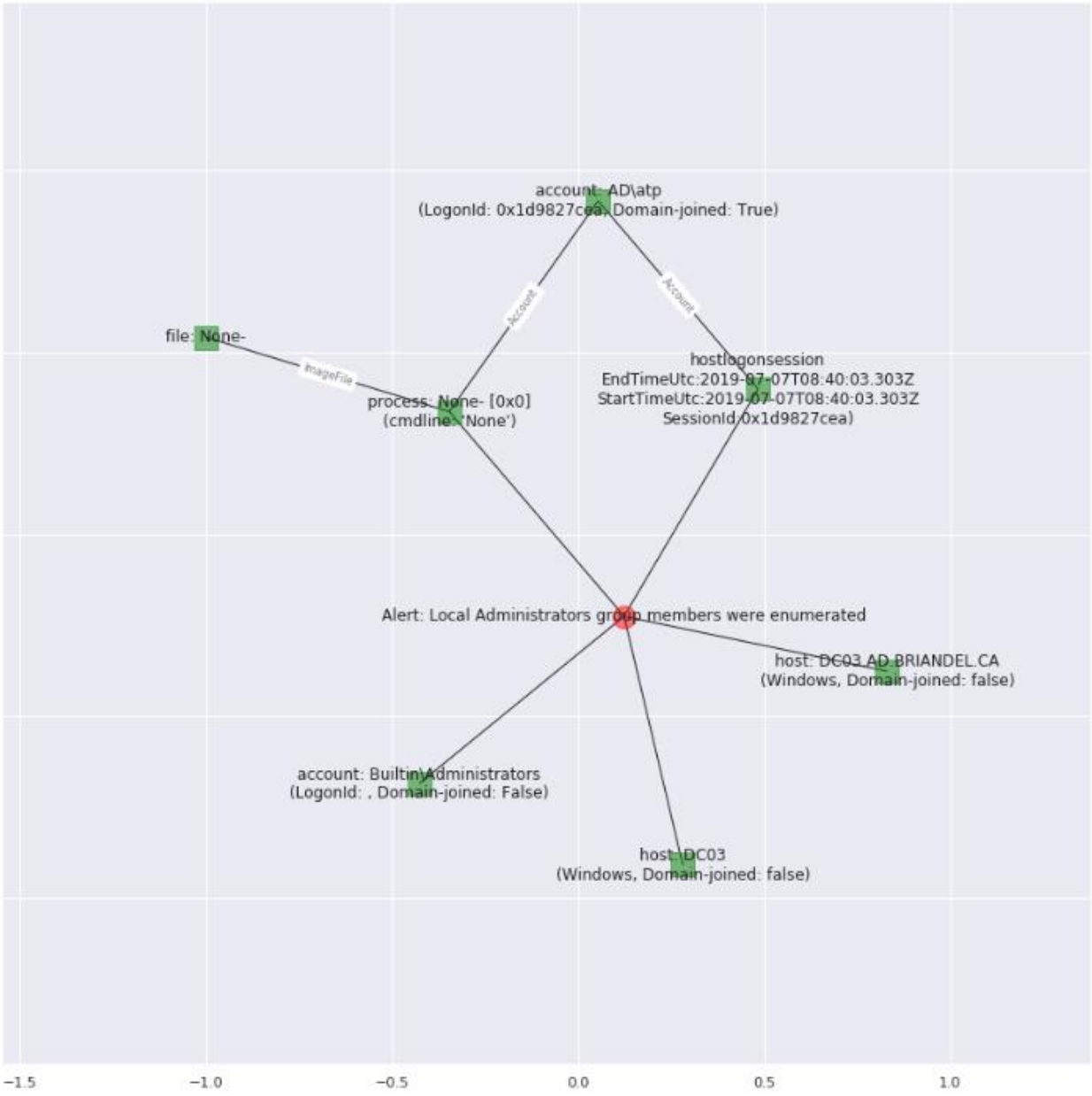

Step 3: Determine the Extent of the Alert with Entity Graphing

1. Run the block of code in the *Plot using Netwrkx/Matplotlib* section.

Plot using Networkx/Matplotlib

```
: # Draw the graph using Networkx/Matplotlib
  %matplotlib inline
  alertentity_graph = mas.create_alert_graph(security_alert)
  nbdisp.draw_alert_entity_graph(alertentity_graph, width=15)
```

2. Examine the output, which takes the form of a very telling graph.



3. The narrative of our alert really emerges here. At the center of the graph (the red dot), we see our original alert in question. Hanging off of it, we see all of the related entities like the account used, process (if applicable), host, the actual group that was interrogated, and the hostlogonsession times. We now have a good idea of where we should take action as we move from the *detect* to the *respond* phase of our work.

Step 4: Establish if there are Related Alerts to Consider in this Investigation

1. Run the first block of code under the *Related Alerts* section.

```
# set the origin time to the time of our alert
query_times = mas.QueryTime(units='day', origin_time=security_alert.TimeGenerated,
                            max_before=28, max_after=1, before=5)
query_times.display()
```

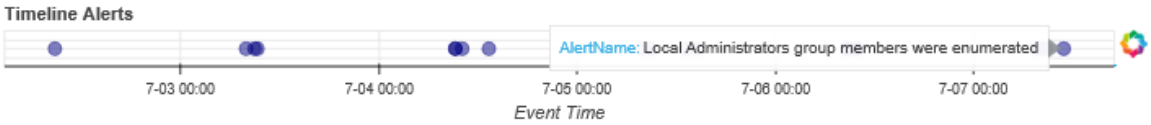
2. Leave the query time boundaries at their default.

3. Run the next block of code.

```
if not security_alert.primary_host:
    print('Related alerts is not yet supported for alerts that are not host-based')
    related_alerts = None
else:
    related_alerts = qry.list_related_alerts(provs=[query_times, security_alert])


    if related_alerts is not None and not related_alerts.empty:
        host_alert_items = related_alerts\
            .query('host_match == @True')[['AlertType', 'StartTimeUtc']]\
            .groupby('AlertType').StartTimeUtc.agg('count').to_dict()
        acct_alert_items = related_alerts\
            .query('acct_match == @True')[['AlertType', 'StartTimeUtc']]\
            .groupby('AlertType').StartTimeUtc.agg('count').to_dict()
        proc_alert_items = related_alerts\
            .query('proc_match == @True')[['AlertType', 'StartTimeUtc']]\
            .groupby('AlertType').StartTimeUtc.agg('count').to_dict()

        def print_related_alerts(alertDict, entityType, entityName):
            if len(alertDict) > 0:
                print('Found {} different alert types related to this {} (\'{}\')'
                      .format(len(alertDict), entityType, entityName))
                for (k,v) in alertDict.items():
                    print('    {}, Count of alerts: {}'.format(k, v))
            else:
                print('No alerts for {} entity \'{}\''.format(entityType, entityName))

        print_related_alerts(host_alert_items, 'host', security_alert.hostname)
        print_related_alerts(acct_alert_items, 'account',
                             security_alert.primary_account.qualified_name
                             if security_alert.primary_account
                             else None)
        print_related_alerts(proc_alert_items, 'process',
                             security_alert.primary_process.ProcessFilePath
                             if security_alert.primary_process
                             else None)
        nbdisp.display_timeline(data=related_alerts, source_columns = ['AlertName'], title='Alerts', height=100)
    else:
        display(Markdown('No related alerts found.'))
```

4. Examine the output in the first graph

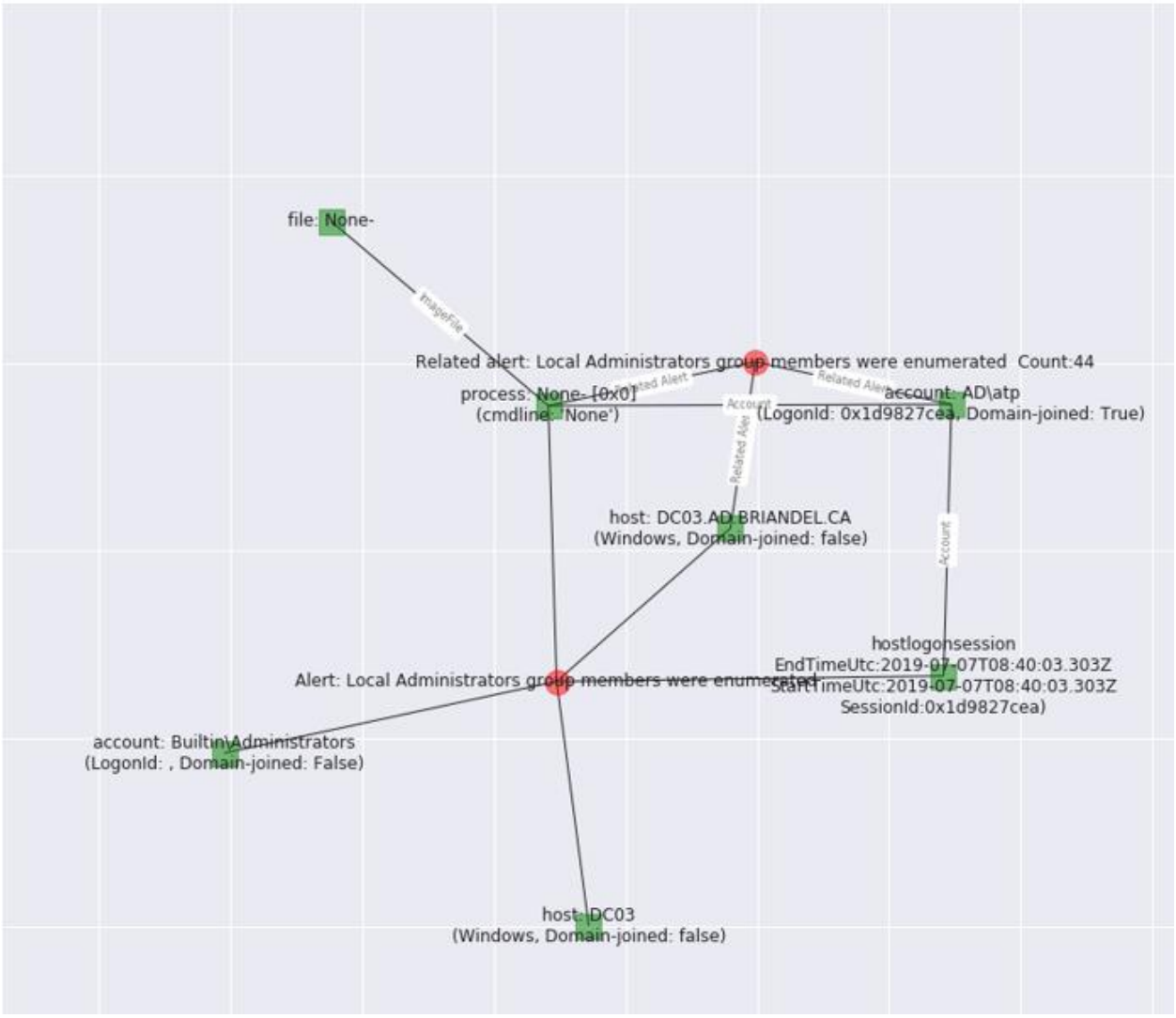
5. Mousing over the points on the graph tells us that there were, in fact, *multiple* enumerations of Administrative groups that took place during the timeframe of this data set. The occurrence that we are investigating is not the only one.



6. Run the block of code under *Show these related alerts on a graph*

```
# Draw a graph of this (add to entity graph)
%matplotlib notebook
%matplotlib inline

if related_alerts is not None and not related_alerts.empty:
    rel_alert_graph = mas.add_related_alerts(related_alerts=related_alerts,
                                             alertgraph=alertentity_graph)
    nbdisp.draw_alert_entity_graph(rel_alert_graph, width=15)
else:
    display(Markdown('No related alerts found.'))
```

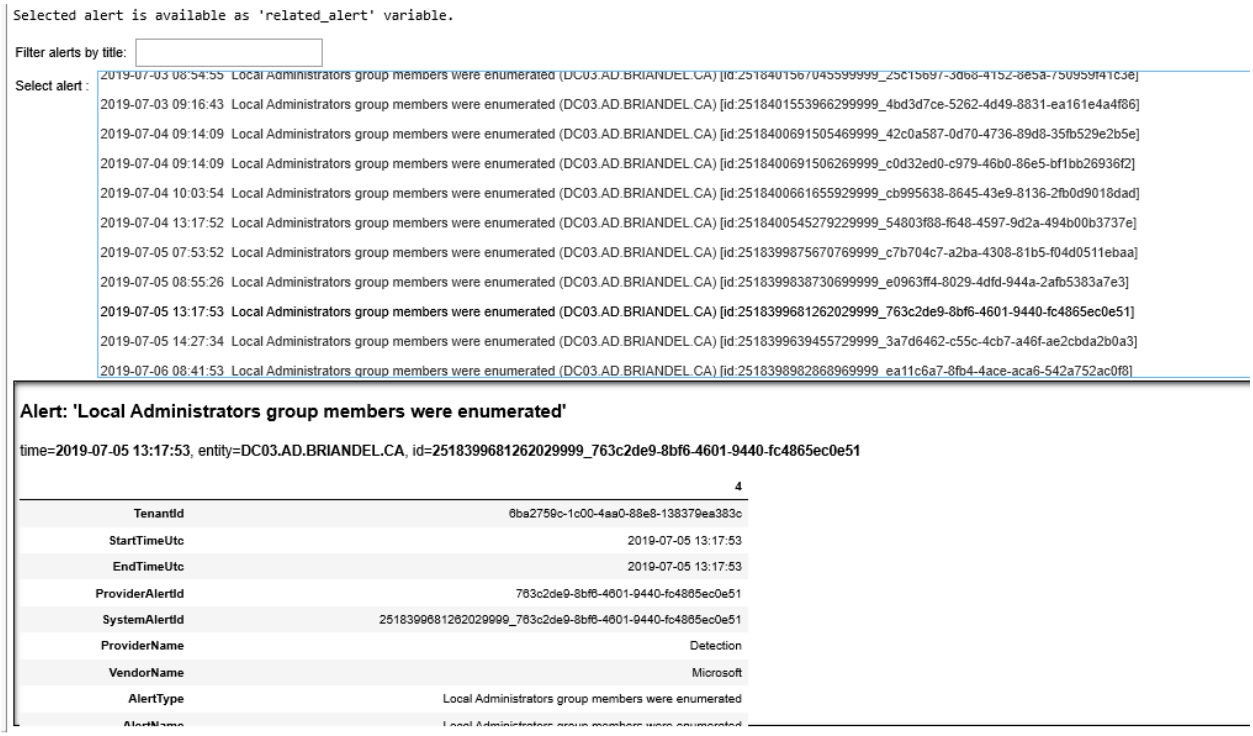
7. Examine the output. We see that, in fact, there were *many* times that the administrative group membership was enumerated; all appear to have used an account or two to accomplish the fact. We now have a very good idea of where our problem exists in terms of the account which has likely been compromised.



8. Run the code under *Brows List of Related Alerts*

```
def disp_full_alert(alert):
    global related_alert
    related_alert = mas.SecurityAlert(alert)
    nbdisp.display_alert(related_alert, show_entities=True)

if related_alerts is not None and not related_alerts.empty:
    related_alerts['CompromisedEntity'] = related_alerts['Computer']
    print('Selected alert is available as \'related_alert\' variable.')
    rel_alert_select = mas.AlertSelector(alerts=related_alerts, action=disp_full_alert)
    rel_alert_select.display()
else:
    display(Markdown('No related alerts found.'))
```

9. Examine the output. The final step in our investigation gives us a truly birds' eye view of the extent of this alert activity. We have a list of each and every time the action took place. By clicking on one of the events, we can also gain detailed data about it in the table below entitled *Alert: 'Local Administrators group members were enumerated'*



10. That ends this particular investigation. The remainder of the Notebook deals with specific Processes, which does not really apply to our particular event. If the event were Process driven, we might utilize those steps to gain additional insight.

Step 5: Optional/Time Permitting – Explore a Different Alert that has Processes Available to Explore

1. Go back to **Multiple Domain Accounts Queried** and explore the notebook for this alert.
2. Note that it will enable specific Processes that you interrogate in the second portion of the Notebook.