March,2020

Osquery

For Cyber Incident Response

LIFARS
your digital world, **secured**

Contents

OVERVIEW

In this case study we demonstrate the use of Osquery framework for incident response. We use Kolide Fleet as a front-end for endpoint and query management, and the popular ELK Stack to provide back-end storage, search capabilities and presentation of the acquired data to the analyst. We present a detailed step-by-step guide of the installation and configuration of all these tools, including the creation of deployment ready Osquery daemon installation package, that is to be distributed to the endpoints. Finally, operation of this setup is demonstrated while performing an incident response on endpoints infected with Dridex, quickly discovering IOCs (Indicator of Compromise) and a part of the malware persistence mechanism.

INTRODUCTION

The Osquery framework can be effectively deployed for cyber-security incident response, essentially performing a host intrusion detection role. The ability of querying for the presence of various artifacts within the operating system makes Osquery powerful tool for initial triage, as well as focused detection of particular IOCs.

Osquery is an operating system framework that allows administrators and cyber security personnel to obtain information about the operating system state of machines in their network, as if from a SQL database. While it is possible to make queries directly through the console of a single machine, it is the possibility of distributed deployment using the Osquery daemon, which provides us with endpoint visibility for monitoring and large-scale information-gathering purposes.

We start our work preparing the IR (Incident Response) server with an installation of the Kolide Fleet management tool. Since some of the information required for configuring the Osquery daemons might become known only after the incident responders arrive on-site, the necessity of deployable scalability requires us to be able to quickly create a pre-configured, self-sufficient and distributable installation package. We will demonstrate this process on a separate build machine, which will also have to be brought on-site. For data collection, indexing and presentation we employ the popular ELK stack, represented by Elasticsearch, Logstash and Kibana.

To demonstrate the usability of Osquery for incident response scenarios, we create a few general-purpose queries for initial triage of the machines in the network. Finally, we show the detection of IOCs of the Dridex banking malware.

INSTRUMENTATION SETUP

Our base system is Debian 10 x64 virtual machine with 10 GB RAM and 40 GB hard drive space. The clean installation of the operating system and all software components takes less than 10 GB.

KOLIDE FLEET INSTALLATION

The open source Kolide Fleet software for endpoint management and for running queries is our preferred tool. It allows us to store prepared queries, that can be further organized into query packs, simplifying the hunt for groups of IOCs. The prerequisites for Kolide Fleet are MariaDB and Redis for the backend storage databases:

```
> apt install mariadb-server -y
> mysql_secure_installation            # choose most secure options

> apt install redis-server -y
```

Then, we need to generate an SSL certificate, which will be used by the endpoints to connect to the fleet manager.

Note, that it is possible to use the server's IP for a certificate common name and it is for this reason, that the certificate needs to be generated each time we deploy a local server to a new incident.

```
> openssl genrsa -out osqkey.pem 4096
> openssl req -new -x509 -key osqkey.pem -out osqcert.pem -days 365 -subj
/CN=<server_ip>
```

The fleet configuration file must be created to point the fleet manager to the back-end resources and the generated SSL certificates. The jwt_key field will be populated in the next step.

```
> mousepad fleet.config
mysql:
  address: 127.0.0.1:3306
  database: osqir
  username: <user_name>
  password: <db_password>
redis:
  address: 127.0.0.1:6379
logging:
  json: true
auth:
  jwt_key:
server:
  cert: /home/<user_name>/osqcert.pem
  key: /home/<user_name>/osqkey.pem
  address: 0.0.0.0:443
```

We finalize Kolide Fleet setup by running the commands below.

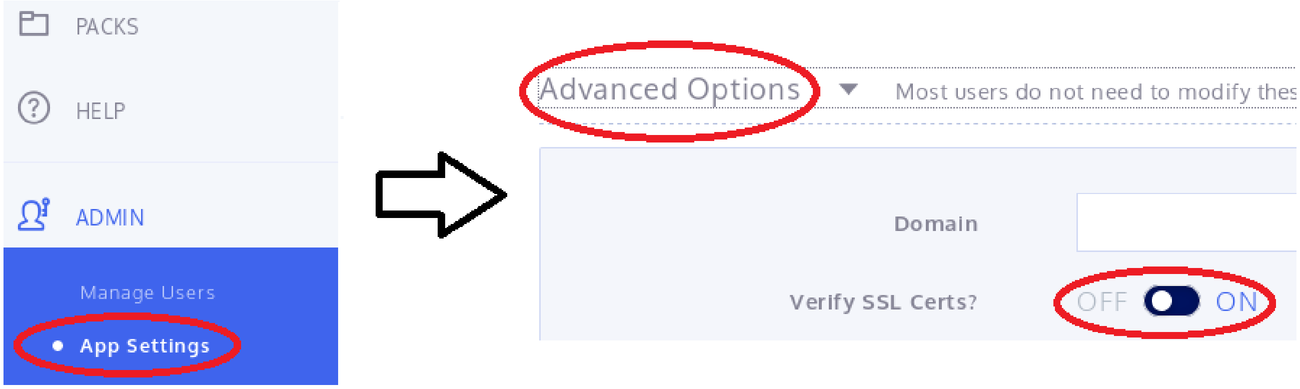
Note, that the second command will throw an error, with some additional info – copy the jwt_key value and paste it into the fleet.config file. Afterwards, test if everything went well by re-running the fleet serve command and visiting the fleet page at *hxxps://localhost*.

```
> fleet prepare db --config fleet.config
> fleet serve --config fleet.config
# throws error -> copy the shown key to fleet.config -> auth: jwt_key
# rerun serve command and test using browser https://<server_ip>
```

We can also run the fleet manager as a service.

```
> mousepad /etc/systemd/system/fleet.service
[Unit]
Description=kolide_fleet
After=network.target

[Service]
ExecStart=/usr/bin/fleet serve -c /home/<user_name>/fleet.config

[Install]
WantedBy=multi-user.target

> systemctl enable fleet.service
> systemctl start fleet.service
```

Finally, we need to disable SLL certificate verification in the fleet manager settings because we have a self-signed certificate.



Verify SSL Certs? OFF ON

ELK STACK INSTALLATION

ELK Stack or Elastic Stack is a highly customizable software suite with rich and powerful data collection, processing, searching and presentation functionality. It offers great flexibility in terms of available configuration options, allowing a wide variety of use-cases. It is important to match all used software component versions for compatibility reasons. At the time of this writing the latest version of ELK Stack is version 7.6.0 is the latest.

Note, that the free versions of these programs do not offer authentication settings, so consider setting the system firewall to block packets from non-local addresses to ports they are listening on – by default 9200 for Elasticsearch, 5601 for Kibana and 9600 for Logstash.

ELASTICSEARCH

The data store back-end is provided by the Elasticsearch, which is also the first component we will install. Java is a single prerequisite, but Debian 10 already ships with Java 11 installed.

```
>   wget   https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.6.0-
amd64.deb
> dpkg -i elasticsearch-7.6.0-amd64.deb
```

Let’s launch the Elasticsearch service and verify the successful installation using systemctl status.

```
> systemctl daemon-reload
> systemctl enable elasticsearch.service
> systemctl start elasticsearch.service
```

We must tell Elasticsearch which network address to listen on. Since we are deploying both the fleet manager and the ELK stack to a single machine, we use the _local_ value. Restart the service and we can move on to the next step.

```
> mousepad /etc/elasticsearch/elasticsearch.yml
# in the Network section uncomment and set
network.host: _local_
> systemctl restart elasticsearch.service
```

KIBANA

The K in ELK stack stands for Kibana - the user interface component responsible for data visualization, whether it is graphical or tabular. It also offers multiple interfaces for management of the particular ELK Stack deployment. We have chosen the .deb package as a means of installation:

```
> wget https://artifacts.elastic.co/downloads/kibana/kibana-7.6.0-amd64.deb
> dpkg -i kibana-7.6.0-amd64.deb
```

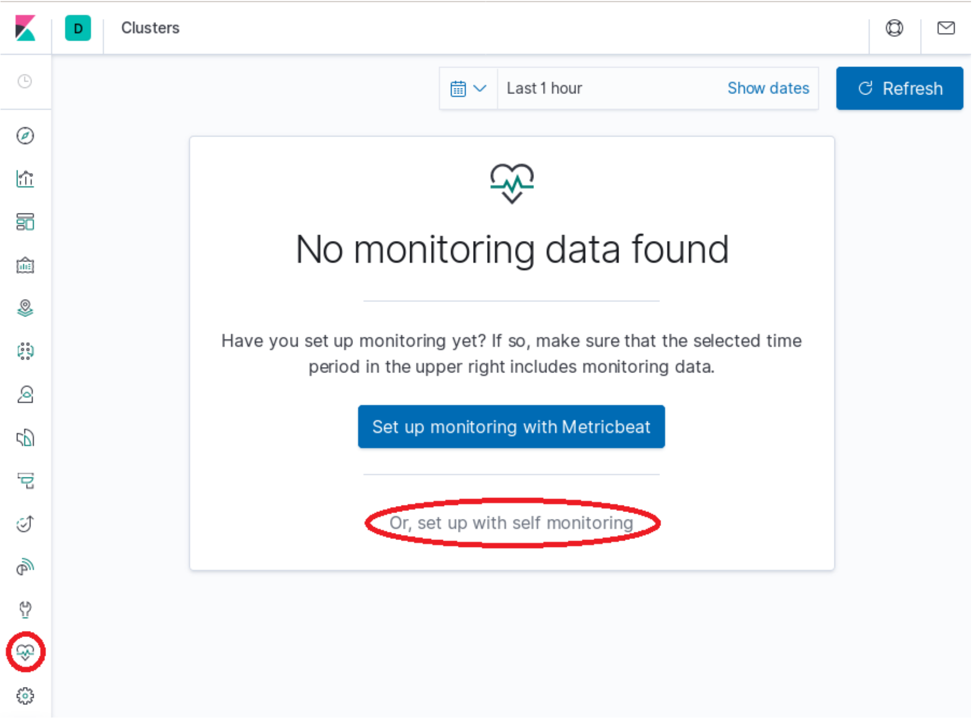
The Kibana service is then run and we can check if it started successfully with systemctl status.

```
> systemctl daemon-reload
> systemctl enable kibana.service
> systemctl start kibana.service
```

We are running all components of the ELK Stack on the same machine, so we set the configuration appropriately and restart the service.

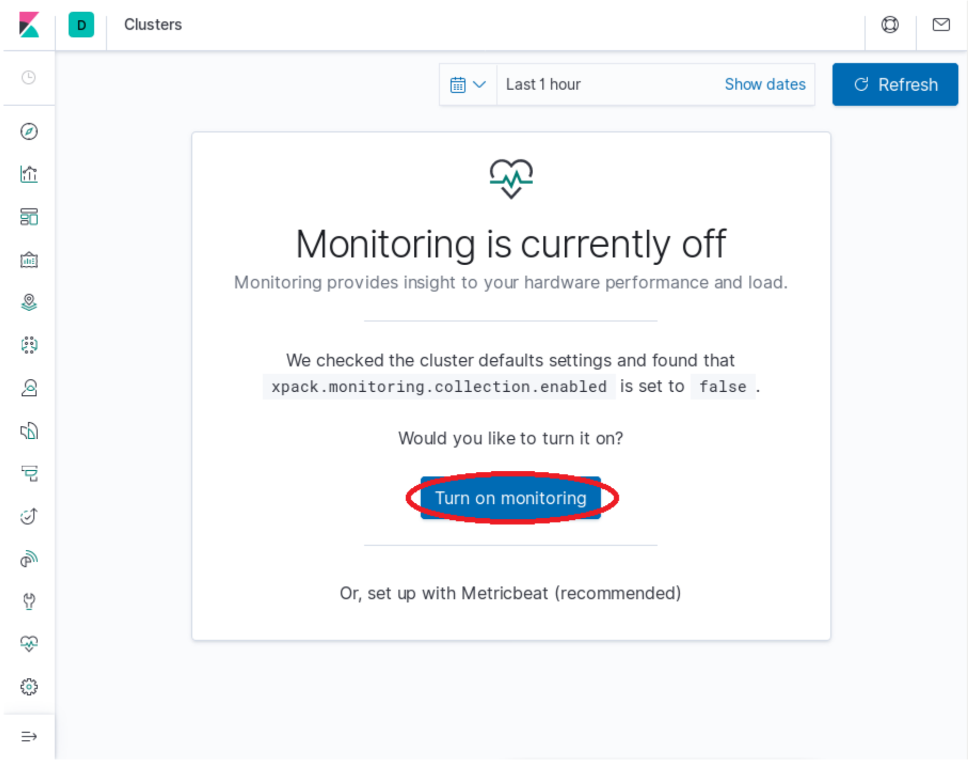
```
> mousepad /etc/kibana/kibana.yml
# uncomment and set
server.host = "localhost"

> systemctl restart kibana.service
```

We can visit the Kibana site at *localhost:5601*. For our use-case, we should enable the monitoring feature in the Stack Monitoring menu.


D
Clusters
Last 1 hour
Show dates
Refresh
Set up monitoring with Metricbeat
Or, set up with self monitoring

No monitoring data found

Have you set up monitoring yet? If so, make sure that the selected time period in the upper right includes monitoring data.



Monitoring is currently off

Monitoring provides insight to your hardware performance and load.

We checked the cluster defaults settings and found that `xpack.monitoring.collection.enabled` is set to `false` .

Would you like to turn it on?

LOGSTASH

The last component of our setup is Logstash, an immensely versatile software for collection of data from various sources. It also features scriptable data pre-processing and enrichment using its simple pipelined input – filter – output workflow. Results from scheduled Osquery query packs are located in */tmp/osquery_result*. Since we are deploying both the fleet manager and the ELK Stack on the same machine, we can import the data from the local filesystem directly into Logstash using the file input module.

Note, that if we had separate machines for fleet management and the ELK Stack, we would have to install Filebeat on the fleet management machine in order to forward the data to Logstash on the ELK machine. Logstash installation is identical to other components:

```
> wget https://artifacts.elastic.co/downloads/logstash/logstash-7.6.0.deb
> dpkg -i logstash-7.6.0-amd64.deb
```

Run the service and optionally check its status to see if everything works.

```
> systemctl daemon-reload
> systemctl enable logstash.service
> systemctl start logstash.service
```

Next, we install the file input module and Elasticsearch output module:

```
> /usr/share/logstash/bin/logstash-plugin install logstash-input-file
> /usr/share/logstash/bin/logstash-plugin install logstash-output-elasticsearch
```

Logstash is configured through the */etc/logstash/logstash.yml* file. The *path.config* field points to a directory, where we will create and store Logstash pipeline configurations and thanks to *config.reload.automatic* set to true, all changes to the pipeline configurations will be reloaded automatically. Restart the service.

```
> mousepad /etc/logstash/logstash.yml
# uncomment and set
path.config: /etc/logstash/conf.d
config.reload.automatic: true

> systemctl restart logstash.service
```

Now, we configure a basic Logstash pipeline. We start by creating a .conf file in the */etc/logstash/conf.d* folder. We point the file input section to the file with the query results and direct the output to the Elasticsearch instance. We will configure the filter section later.

```
> mousepad /etc/logstash/conf.d/<logstash_pipe_filename>.conf
input {
  file {
    path => "/tmp/osquery_result"
  }
}

# filter {
# }

output {
  # stdout { codec => rubydebug }
  elasticsearch {
     hosts => ["localhost:9200"]
  }
}
```

The commented-out line in the output section can help in debugging the filter configuration by routing the output to console, we will address this a bit later because we have no Osquery endpoints and no queries scheduled yet.

OSQUERY INSTALLER PACKAGE BUILD

In order to simplify the deployment of Osquery to endpoints at a larger scale, it is advantageous to create an install-and-forget installer package with custom prepared configuration files for the Osquery daemons being deployed. We want the daemons to automatically enroll in the fleet without any further interaction other than pushing the installation package to endpoints via software management or in the worst case, double-clicking the installer. We perform the build on a Windows 10 x64 machine, and we start by an entirely optional step of installing Chocolatey package manager, which makes the installation of other tools easier. We need an administrator PowerShell and we also have to restart PowerShell whenever there are changes to environment variables.

```
> Set-ExecutionPolicy Bypass -Scope Process -Force; iex((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

Restart PowerShell and install Git and WiX Toolset:

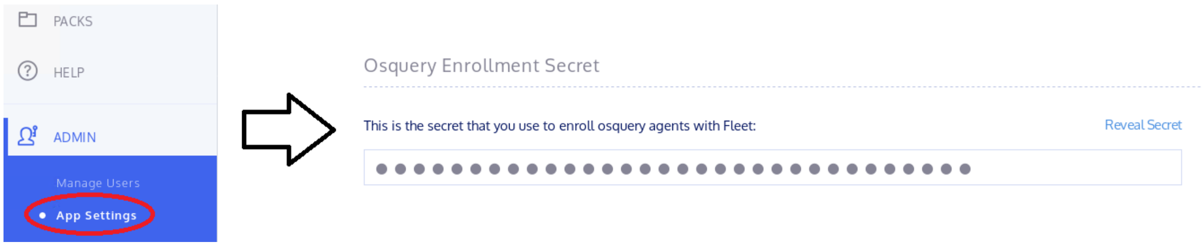
```
> choco install git -y
> choco install wixtoolset -y
```

Close PowerShell and add the WiX Toolset bin path to the PATH variable manually, the default location is *C:|Program Files (x86)|WiX Toolset v3.11|bin*. Let’s continue in PowerShell:

```
> git clone https://github.com/facebook/osquery.git
> cd .\osquery\
> git checkout tags/<latest_version_number>
```

The latest version number of Osquery can be found on its Github repository under the Releases tab. Further, need to download and install the pre-built Windows package from *https://osquery.io/downloads*. This allows us to build the .msi installer only, without having to compile the Osquery binaries first. Now we can create our build folder and necessary configuration for the Osquery daemon. The first file will contain the enrollment secret string, which the endpoint daemons will use for registering with our Kolide Fleet server.

```
> mkdir build\msi
> cd .\build\msi\

> notepad.exe <enrollment_secret_filename>
# Copy the enrolment secret string from Kolide Fleet setting page
```



Osquery Enrollment Secret

This is the secret that you use to enroll osquery agents with Fleet:

●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●

Next is a configuration file containing flags for the Osquery daemon. Replace the bracketed strings with appropriate values for your case. The SSL certificate we have generated previously must be copied over from the fleet server.

```
> notepad.exe <flags_filename>
 --tls_hostname=<fleet_hostname_or_ip>
 --tls_server_certs=C:\Program Files\osquery\<server_ssl_cert_filename>
 --host_identifier=hostname
 --enroll_tls_endpoint=/api/v1/osquery/enroll
 --config_plugin=tls
 --config_tls_endpoint=/api/v1/osquery/config
 --config_tls_refresh=10
 --disable_distributed=false
 --distributed_plugin=tls
 --distributed_interval=10
 --distributed_tls_max_attempts=3
 --distributed_tls_read_endpoint=/api/v1/osquery/distributed/read
 --distributed_tls_write_endpoint=/api/v1/osquery/distributed/write
 --logger_plugin=tls
 --logger_tls_endpoint=/api/v1/osquery/log
 --logger_tls_period=10
 --enroll_secret_path=C:\Program Files\osquery\<enrollment_secret_filename>
```

We can now run the *make_windows_package.ps1* PowerShell script provided in the Git package to create a .wxs configuration file for WiX tools. The script may or may not create a .msi file. While running the script, we found that the file did not work properly, due to an error in the generated .wxs file, this can be fixed manually. To resolve this issue, look around line number 128 and correct the line responsible for including the default *certs.pem* file to following:

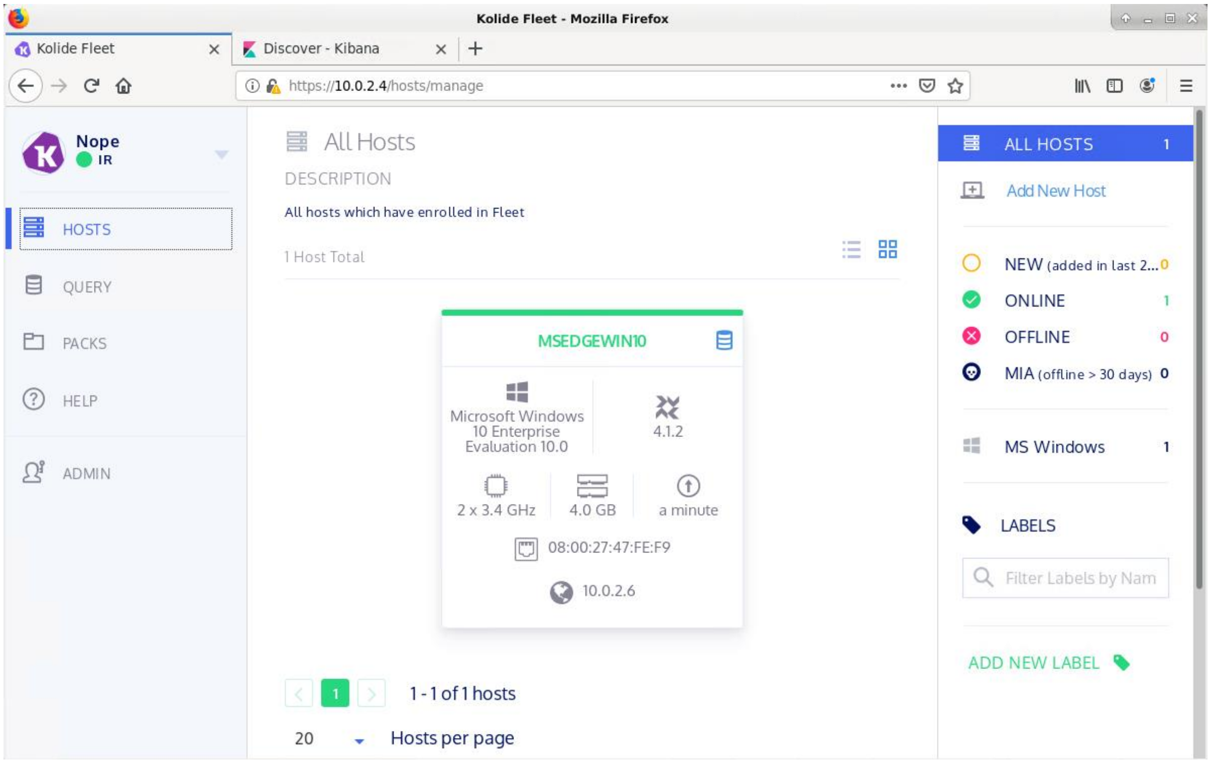
```
<File Id='cert_0' Name='certs.pem'
Source='<root_folder>\osquery\tools\deployment\..\..\tools\deployment\certs.pem'/>
```

Finish with generating a .wixobj file and building the final .msi package:

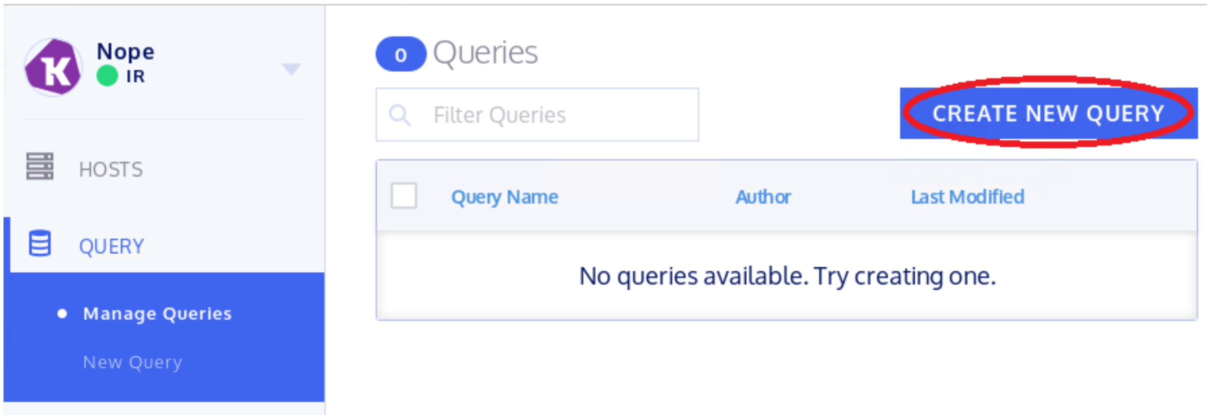
```
> candle.exe .\osquery.wxs
> light.exe -ext WiXUtilExtension .\osquery.wixobj -o .\osquery.msi
```

INSTRUMENTATION TEST

To test our configuration, we will deploy the built package to a Windows 10 x64 test endpoint on the same network as the server. For this purpose, we created an internal network in Virtualbox to keep the network isolated because of later tests with actual malware. Once we run our Osquery .msi package on the endpoint, we should see the endpoint appear in Kolide Fleet dashboard on our server:

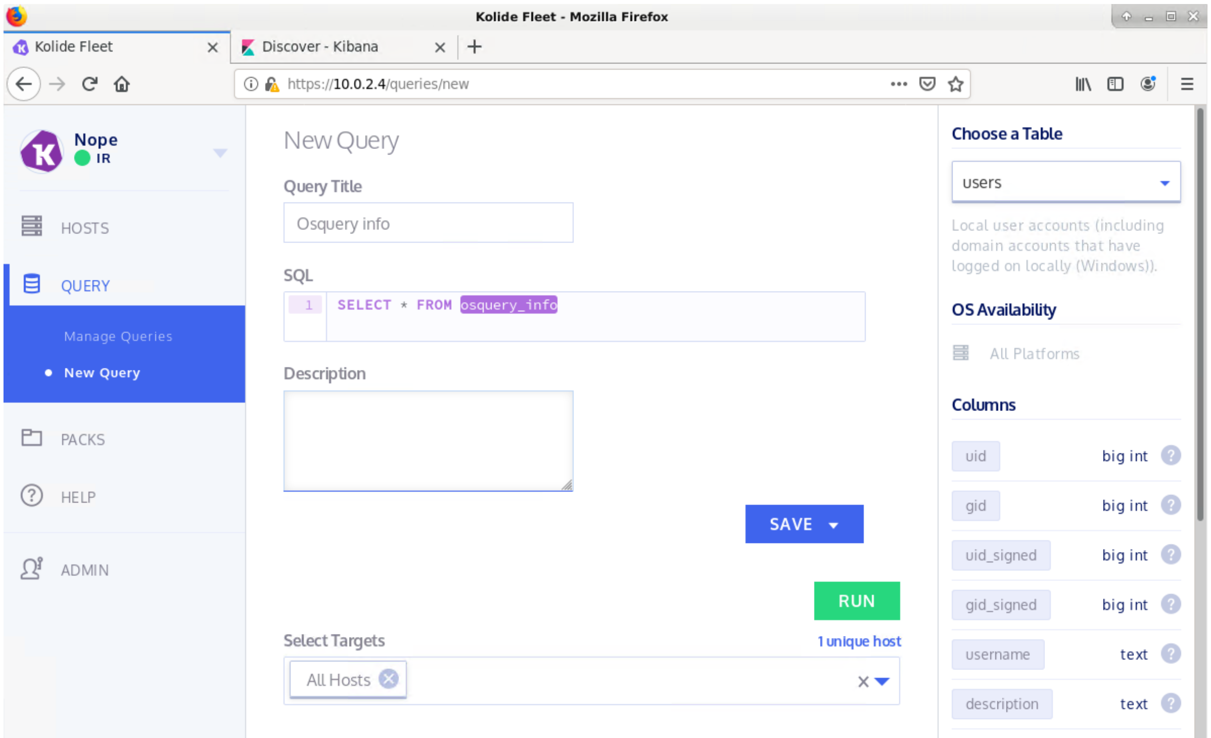


Let’s create a test query, which will return information about the daemon itself:



Fill in the title, then query itself in SQL syntax, add some description and assign the hosts on which the query should be run.

Note the help panel on the right, which will come in handy when dealing with more complex queries.



New Query

Query Title: Osquery info

SQL

```
SELECT * FROM osquery_info
```

Description

Select Targets: All Hosts

Choose a Table: users

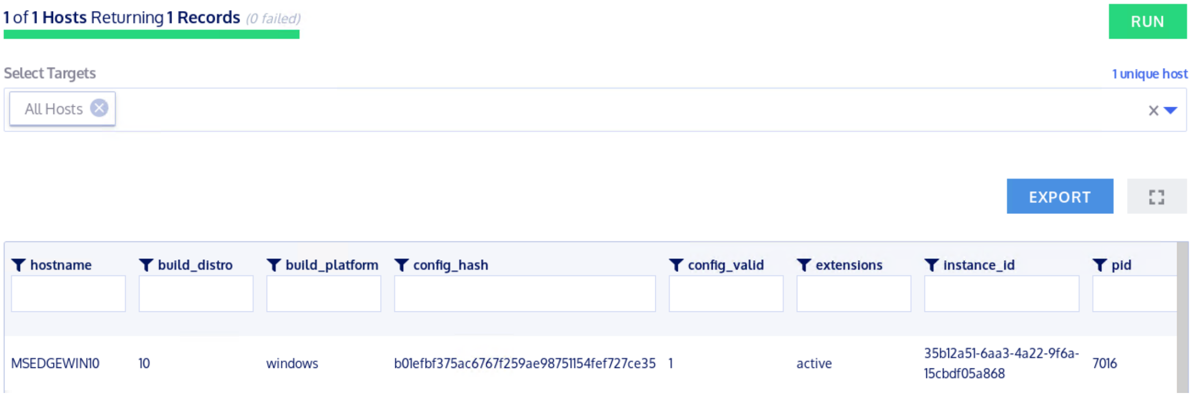
Local user accounts (including domain accounts that have logged on locally (Windows)).

OS Availability: All Platforms

Columns

uid	big int
gid	big int
uid_signed	big int
gid_signed	big int
username	text
description	text

We save the query and run it, to see if everything is working. We should see the output in a few seconds:

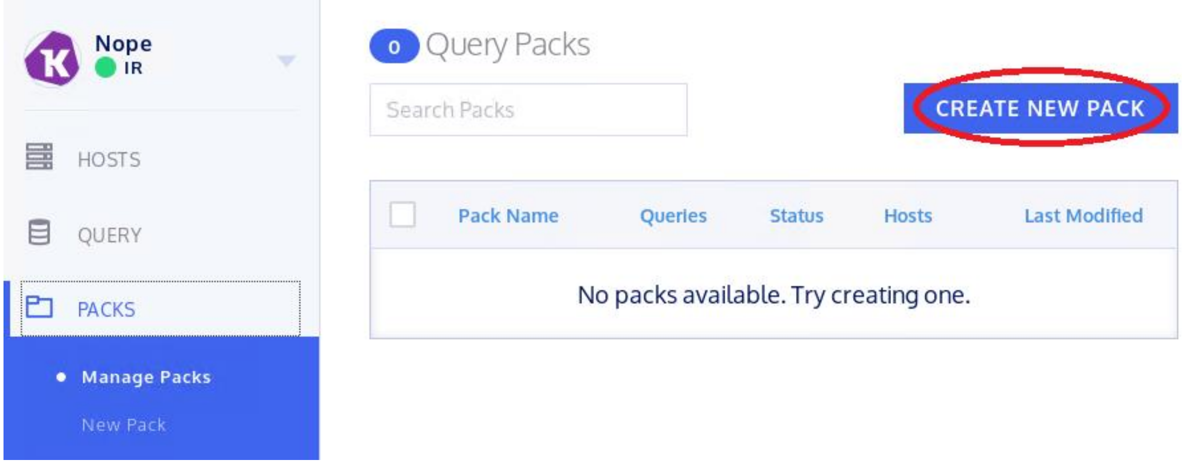


1 of 1 Hosts Returning 1 Records (0 failed)

Select Targets: All Hosts

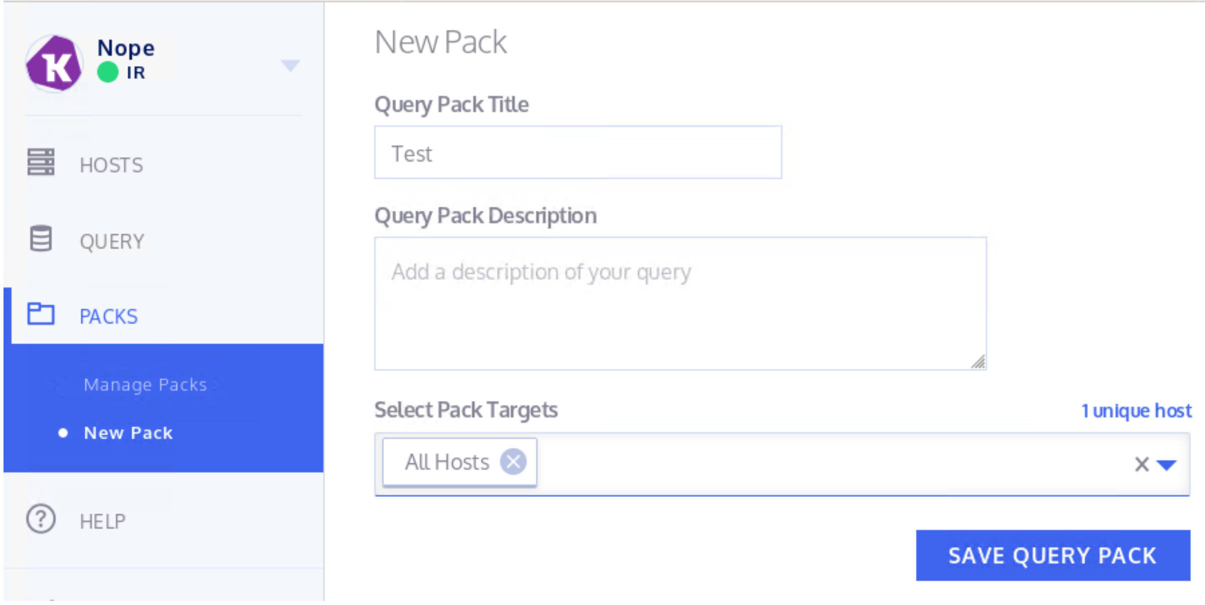
hostname	build_distro	build_platform	config_hash	config_valid	extensions	instance_id	pid
MSEDGEWIN10	10	windows	b01efbf375ac6767f259ae98751154fef727ce35	1	active	35b12a51-6aa3-4a22-9f6a-15cbdf05a868	7016

These queries provide output in the web interface only, and although we will be using them for tailored querying and for query testing, we need to schedule a query pack to automatically run a set of queries at scheduled intervals:



No packs available. Try creating one.

Set the title, target hosts and save the pack:

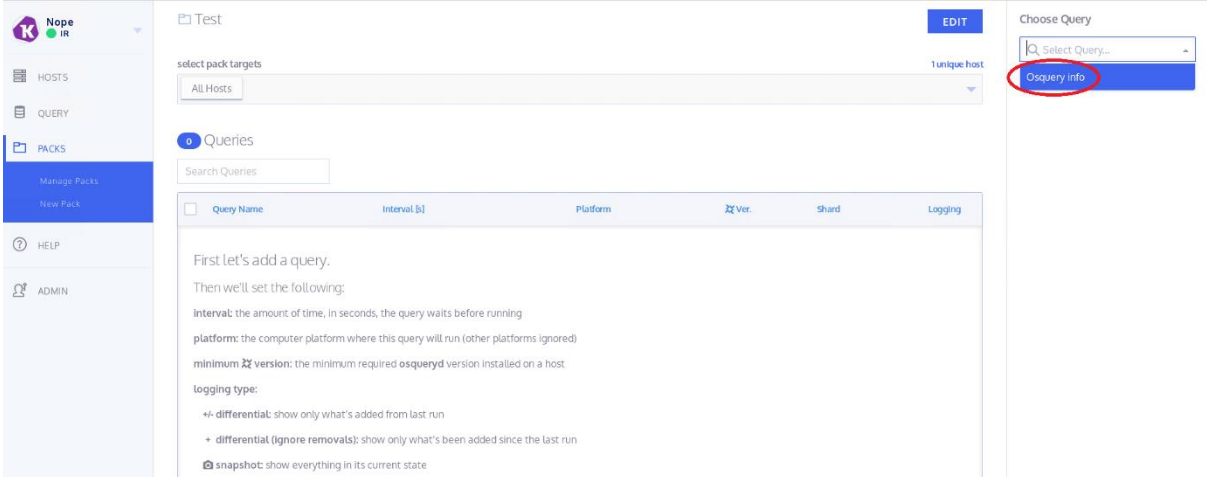


Query Pack Title: Test

Query Pack Description: Add a description of your query

Select Pack Targets: All Hosts — 1 unique host

Select the prepared Osquery info query from the dropdown list:



Test

select pack targets: All Hosts — 1 unique host

0 Queries

Search Queries

Query Name	Interval [s]	Platform	Ver.	Shard	Logging

First let's add a query.

Then we'll set the following:

interval: the amount of time, in seconds, the query waits before running

platform: the computer platform where this query will run (other platforms ignored)

minimum version: the minimum required **osqueryd** version installed on a host

logging type:

+/- **differential:** show only what's added from last run

+ **differential (ignore removals):** show only what's been added since the last run

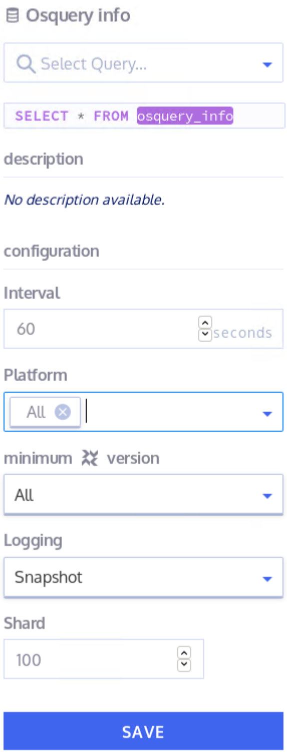
snapshot: show everything in its current state

Choose Query

Select Query...

Osquery info

In the boxes that appear we fill the query interval, target platform, Osquery daemon version, type of data to capture and shard. The shard is the percentage of hosts that will be queried. Snapshot logging will always capture the entire state of the queried information as opposed to differential logging, which only captures changes in respect to a previous state. After saving the settings the query will appear in the list in the center of the screen. We can also check that the query pack is enabled to run in the query pack menu.



Osquery info

Select Query...

```
SELECT * FROM osquery_info
```

description

No description available.

configuration

Interval: 60 seconds

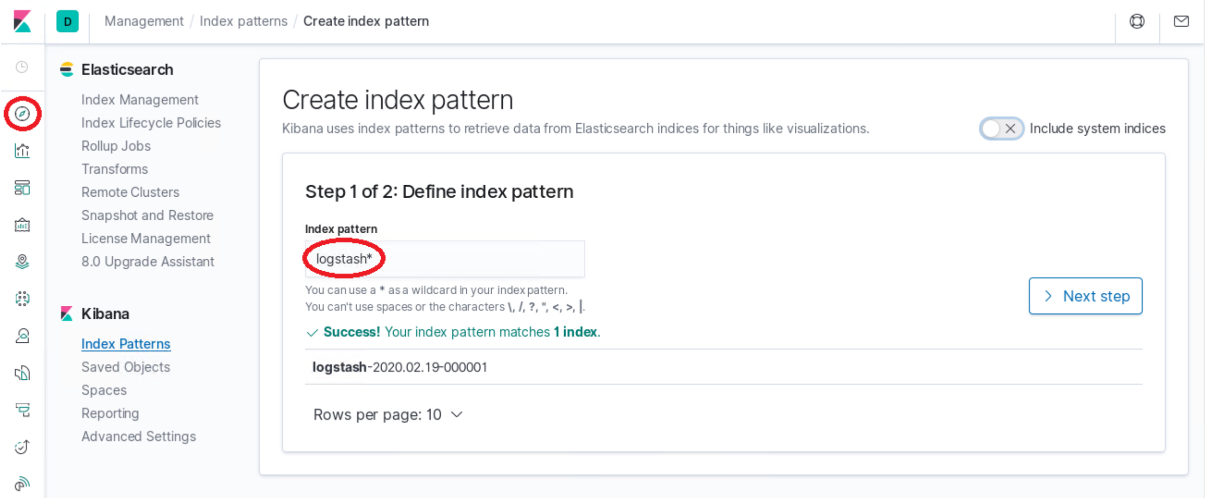
Platform: All

minimum version: All

Logging: Snapshot

Shard: 100

Query packs are an effective way to keep the queries organized, for example we could create a query pack for a particular type of malware and enable it when we want to check for the presence of its IOCs, to which the queries from that pack will be tailored. The response data will be written into the */tmp/osquery_result* file mentioned before. At this point, we have the Osquery part of our instrumentation working and now it is time to bring the ELK stack into play. Let's move to the Discover tab on the Kibana portal, where we will be prompted to create a new index pattern. We should already see our Logstash index in the table so we can enter *logstash** into the Index pattern textbox:



Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Step 1 of 2: Define index pattern

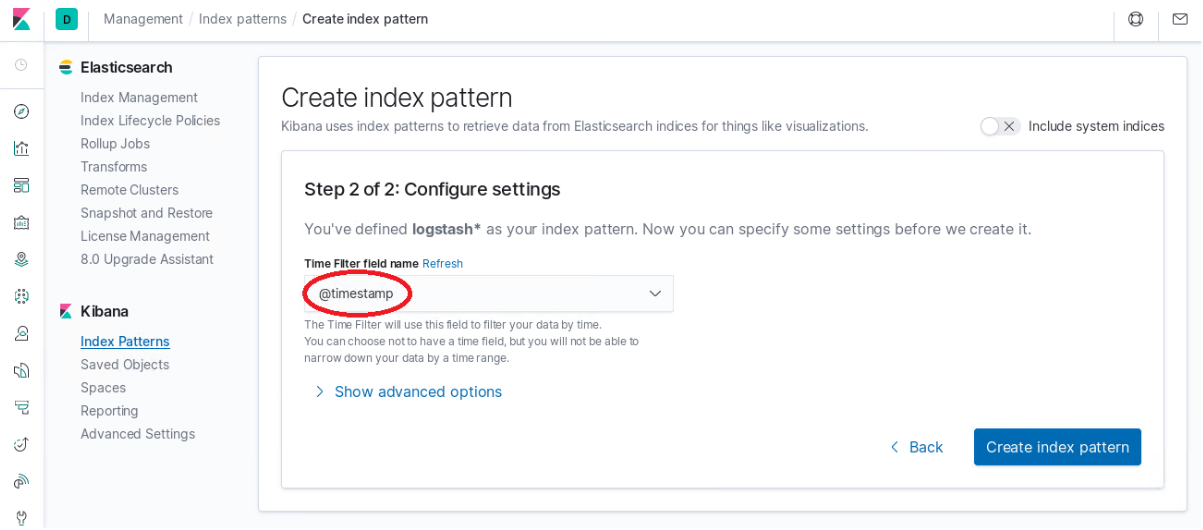
Index pattern: logstash*

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

✓ **Success!** Your index pattern matches **1 index**.

logstash-2020.02.19-000001

On the next page, select "@timestamp" from the Filter field dropdown list and click Create index pattern:



Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

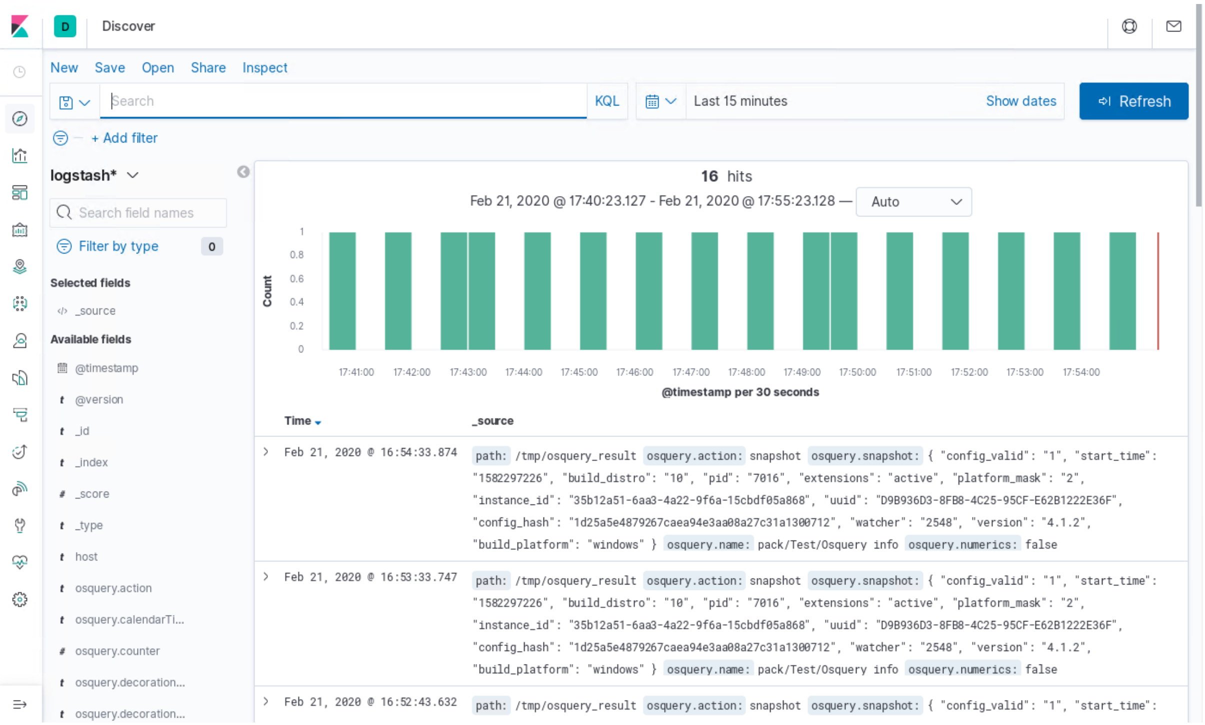
Step 2 of 2: Configure settings

You've defined **logstash*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh: @timestamp

The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

When we move to the Discover tab, we should see incoming data according to the scheduled query pack:



```
Time                          _source
Feb 21, 2020 @ 16:54:33.874   path: /tmp/osquery_result osquery.action: snapshot osquery.snapshot: { "config_valid": "1", "start_time":
                              "1582297226", "build_distro": "10", "pid": "7016", "extensions": "active", "platform_mask": "2",
                              "instance_id": "35b12a51-6aa3-4a22-9f6a-15cbdf05a868", "uuid": "D9B936D3-8FB8-4C25-95CF-E62B1222E36F",
                              "config_hash": "1d25a5e4879267caea94e3aa08a27c31a1300712", "watcher": "2548", "version": "4.1.2",
                              "build_platform": "windows" } osquery.name: pack/Test/Osquery info osquery.numerics: false
Feb 21, 2020 @ 16:53:33.747   path: /tmp/osquery_result osquery.action: snapshot osquery.snapshot: { "config_valid": "1", "start_time":
                              "1582297226", "build_distro": "10", "pid": "7016", "extensions": "active", "platform_mask": "2",
                              "instance_id": "35b12a51-6aa3-4a22-9f6a-15cbdf05a868", "uuid": "D9B936D3-8FB8-4C25-95CF-E62B1222E36F",
                              "config_hash": "1d25a5e4879267caea94e3aa08a27c31a1300712", "watcher": "2548", "version": "4.1.2",
                              "build_platform": "windows" } osquery.name: pack/Test/Osquery info osquery.numerics: false
Feb 21, 2020 @ 16:52:43.632   path: /tmp/osquery_result osquery.action: snapshot osquery.snapshot: { "config_valid": "1", "start_time":
```

TROUBLESHOOTING

If you see a screen like the one above, awesome! In case you do not, we can troubleshoot the building blocks of our setup from the beginning:

Is the endpoint correctly enrolled in the fleet manager? It must be visible in the Kolide Fleet > Hosts screen. If not, check the configuration file containing the flags for the Osquery daemon. If the problem persists, stop the Osquery service and run the daemon directly from PowerShell to see what error is reported.

Do we get a response when a single query is ran from the Kolide Fleet interface?

Is the query pack configuration correct? Check if the host number for that query pack is not zero.

Does the */tmp/osquery_result* file contain data? If yes, the problem is somewhere in the ELK Stack configuration.

Is the Logstash service and the Logstash pipeline configured correctly? Stop the Logstash service and run the command below to see Logstash messages. Also, uncomment the line in the pipeline config file we created that deals with writing the data into stdout. Consider using the *--config.test_and_exit* flag, which performs a dry run with the current configuration.

```
> systemctl stop logstash.service
> /usr/share/logstash/bin/logstash --path.settings=/etc/logstash
```

Provided you have query packs scheduled to run from the Kolide Fleet interface, you can also use these commands when scripting a new pipeline filter immediately see the output and any possible errors.

Still nothing? Move on to the Elasticsearch by disabling its service and running it from console. Check the configuration file as well, especially the network settings. The same procedure can be done with Kibana, if necessary.

QUERY PROCESSING

At this point, we have our tools in a working state and we can focus on tuning the system for our incident response and threat hunting purposes. We will be adding new queries, scheduling new query packs and script a general-purpose Logstash pipeline filter to process their results. To make things more realistic, we enroll more endpoints into our fleet with Windows 10, 8.1 and 7, to also demonstrate Osquery compatibility.

If we look at the Kibana > Discover page, we will see that all the interesting data from Osquery are located in the *message* field and are currently non-indexable and thus not searchable. To solve this, let's open the Logstash pipeline configuration file and insert the text below into the filter section:

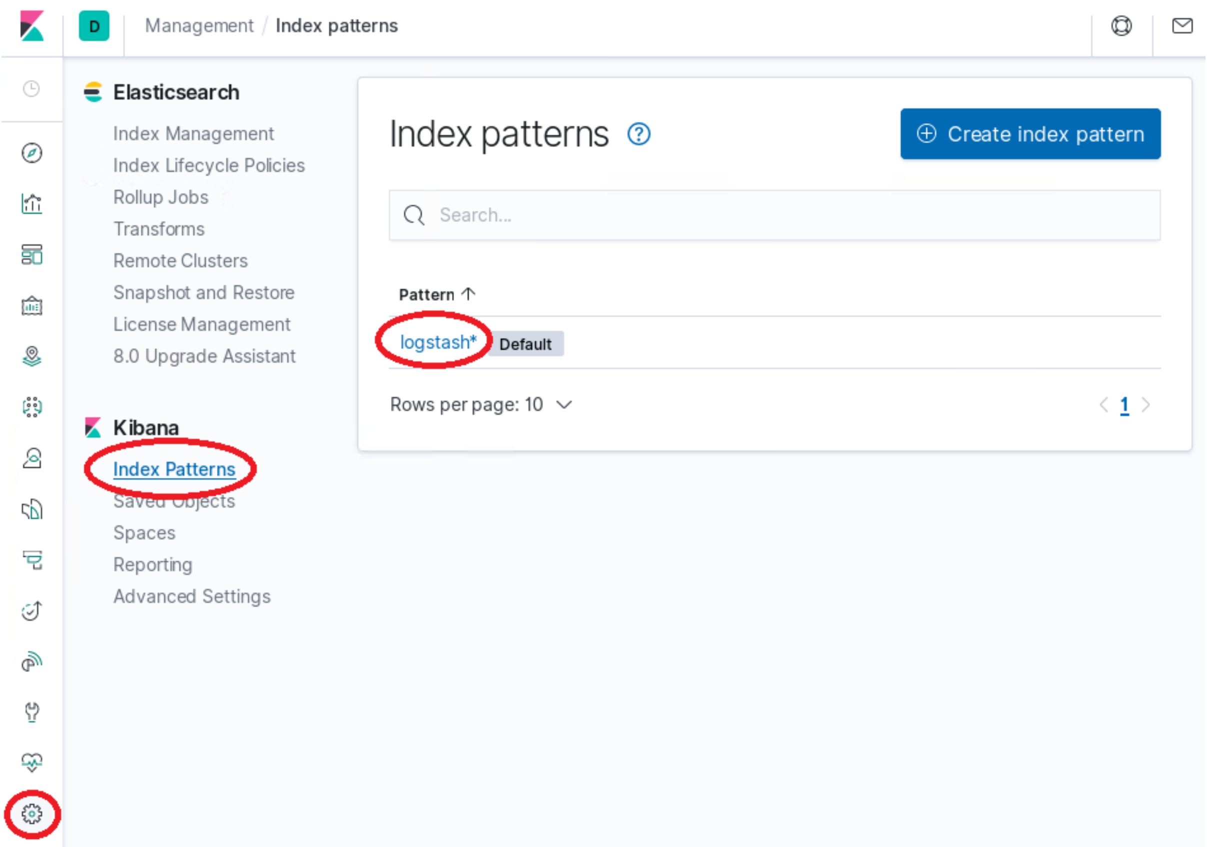
```
> mousepad /etc/logstash/conf.d/<logstash_pipe_filename>.conf
...
filter {
  json {
    source => "message"
    target => "osquery"
    remove_field => ["message"]
  }
  split {
    field => "[osquery][snapshot]"
    target => "[osquery][response]"
    remove_field => ["[osquery][snapshot]"]
  }
}
...
```

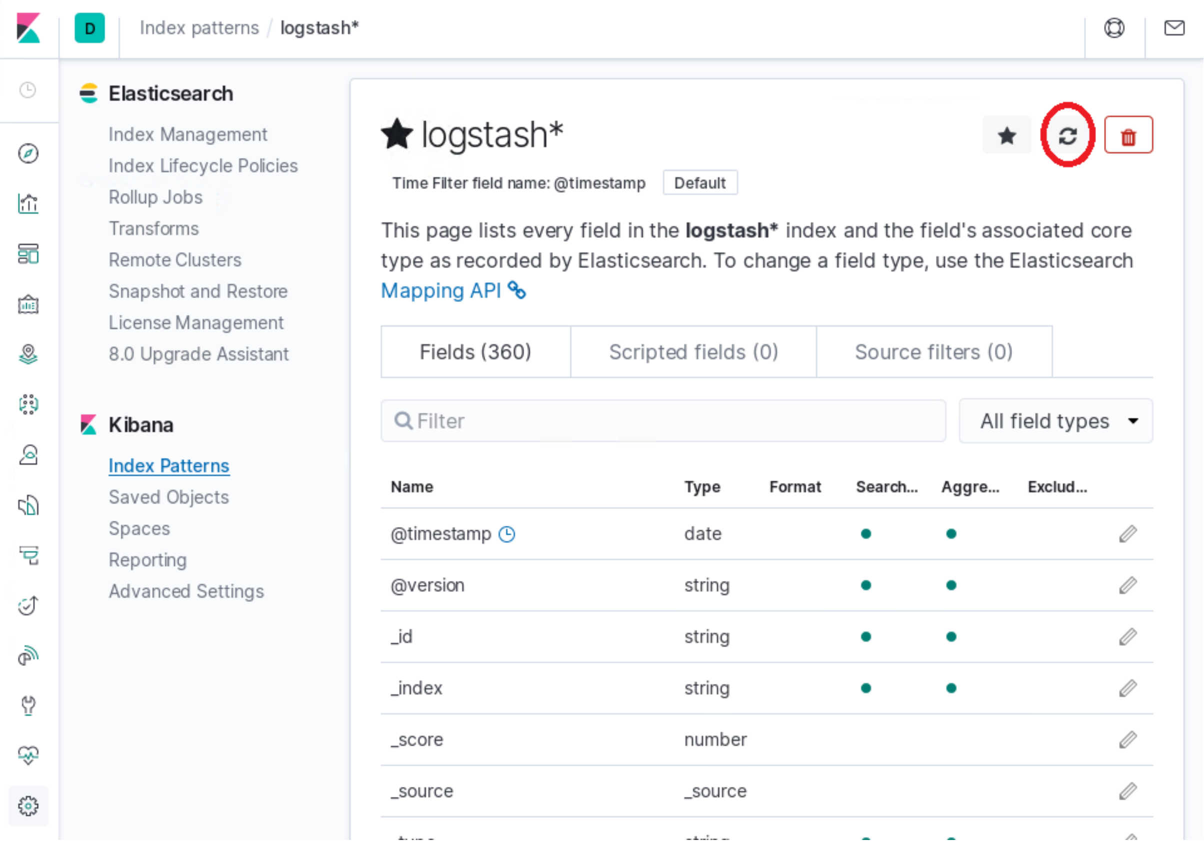
This runs a json parser against the *message* field, unpacks its depth-1 contents under a new *osquery* field and deletes the *message* field. For differential querying, the *osquery.action* will have different values, such as *added* or *removed* and in which case the data will at this point be located under *osquery.columns.<subkey>*. On the other hand, for queries scheduled to take entire snapshots, the data will now be under a single *osquery.snapshot* field, which we have to further split into multiple fields with names *osquery.<name>.<subkey>*. By choosing the *<name>* key to be *columns*, we effectively merge identically named keys from differential and snapshot querying, which is necessary if we want to be able to search the data by keys from both types of acquisition. Take care to choose the type of querying which makes the most sense for your situation.

It will happen in some situations, that the field merging described in the previous paragraph can put together data with different context, creating clutter and noise. Thus, we can improve the searchability of our data by taking the *osquery.name*, which contains the name of the query pack and query itself. We will create two new fields during the Logstash filter phase - *query_pack* and *query_name*, which can later be used to narrow down our search or exclude some queries from the search output in Kibana. The grok Logstash filter module can be used:

```
> mousepad /etc/logstash/conf.d/<logstash_pipe_filename>.conf
...
filter {
  ...
  grok {
    match => { "[osquery][name]" =>
             ".*/%{WORD:[osquery][query_pack]}/%{WORD:[osquery][query_name]}" }
  }
}
...
```

Lastly, it is important to refresh the index pattern to include the newly created fields and make them searchable. This is done through the Kibana Management tab:





★ logstash*

Time Filter field name: @timestamp Default

This page lists every field in the **logstash*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch Mapping API

Name	Type	Format	Search...	Aggre...	Exclud...
@timestamp	date		●	●	
@version	string		●	●	
_id	string		●	●	
_index	string		●	●	
_score	number				
_source	_source				
_type	string		●	●	

INCIDENT RESPONSE METHODOLOGY USING OSQUERY AND ELK STACK

Incident response and threat hunting are deeply involved multi-step processes requiring different approaches for each stage. In general, after arriving on-site we want to collect a broad spectrum of data and search for anything out of the ordinary.

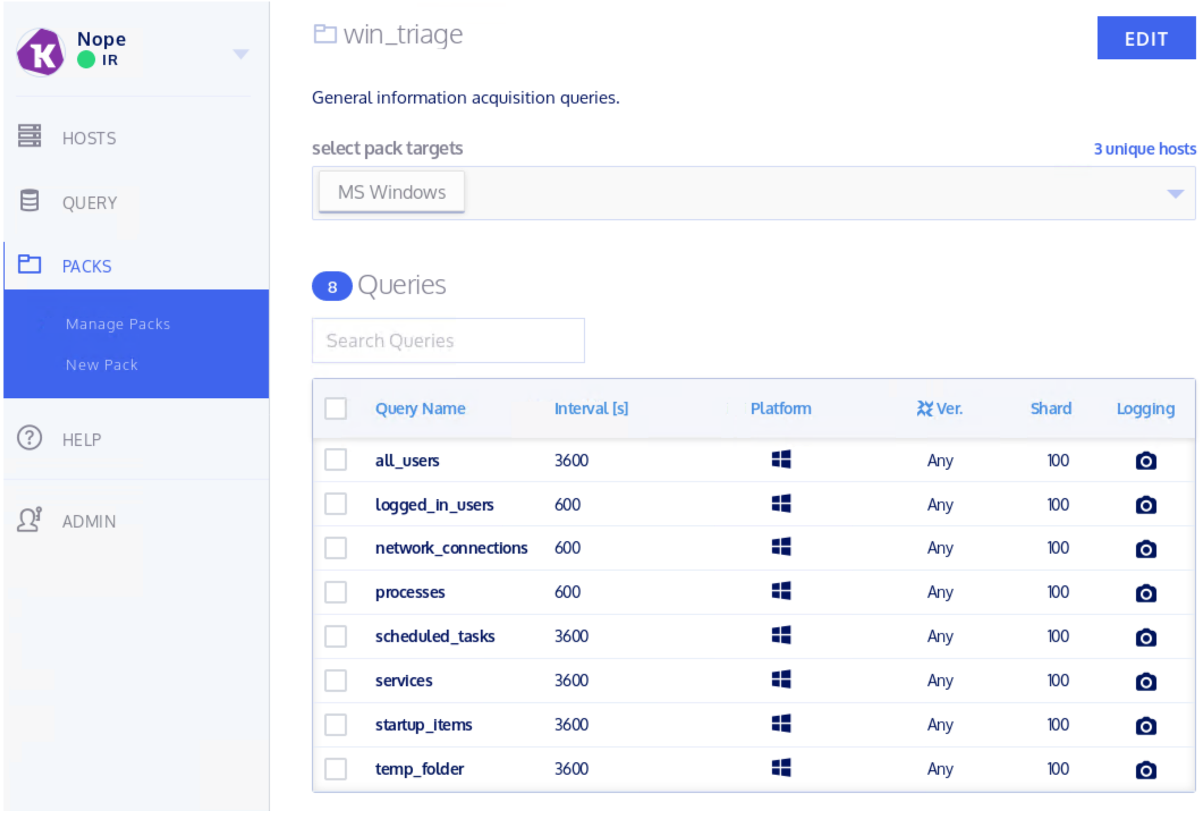
In the second step, we tune our data acquisition to uncover more in-depth information about the suspicious items.

For the initial triage phase, we prepare a set of simple and general Osquery queries for Windows endpoints to acquire information about users, startup items, scheduled tasks, services, running processes, active network connections and suspicious files in user Temp folders:

```
SELECT users.username, users.type FROM users;

SELECT user, tty, type, time FROM logged_in_users;

SELECT name, action, path, state, datetime(last_run_time,'unixepoch') AS last_run_time
FROM scheduled_tasks;
```

```
SELECT name, path, source, status FROM startup_items;

SELECT name, path, status FROM services;

SELECT processes.name, processes.path, processes.cmdline, hash.sha256 FROM processes
JOIN hash WHERE hash.path LIKE processes.path;

SELECT                processes.name,                process_open_sockets.local_port,
process_open_sockets.remote_address  FROM  process_open_sockets  JOIN  processes  USING
(pid);
SELECT path, sha256 FROM hash WHERE path LIKE 'C:\Users\%\AppData\Local\Temp\%%';
```

We put all these into a new Windows triage query pack in the Kolide Fleet interface, as shown previously.

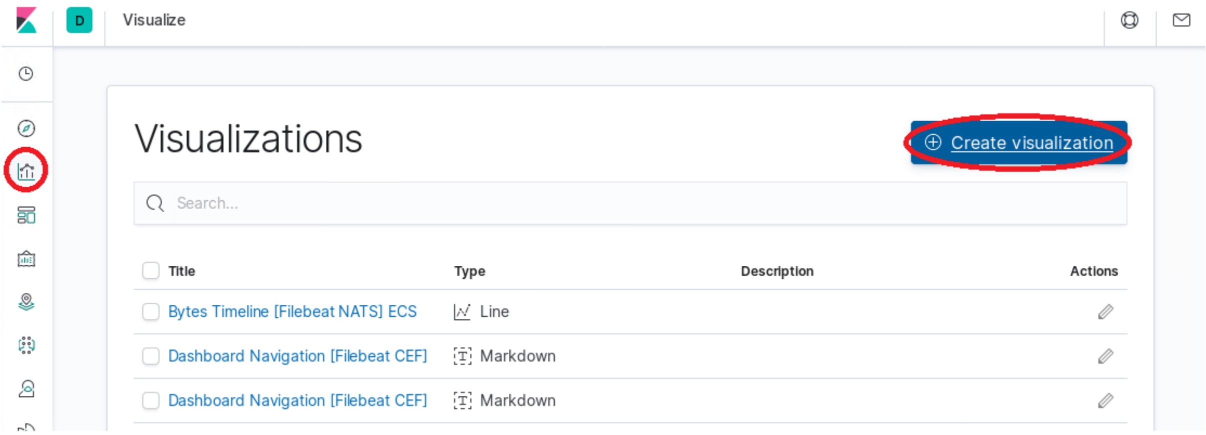


win_triage

General information acquisition queries.

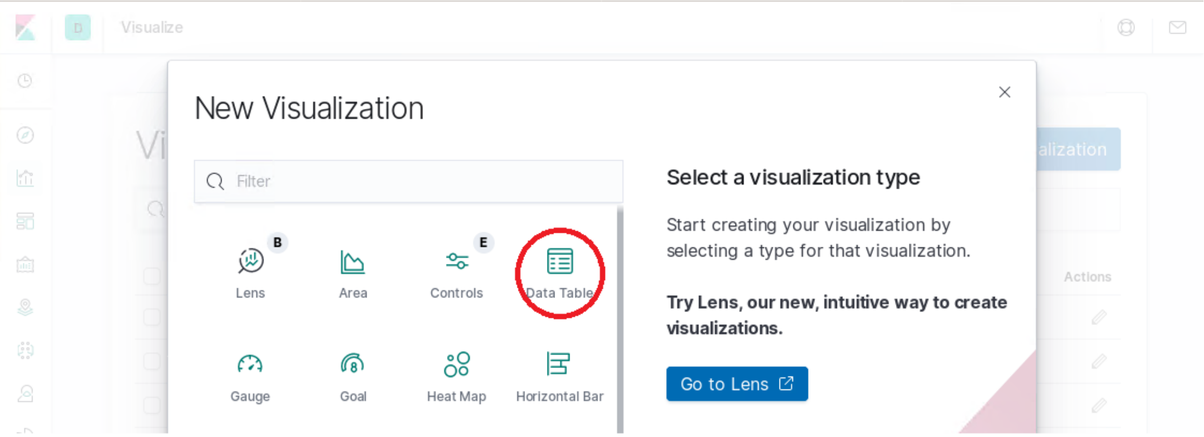
□	Query Name	Interval [s]	Platform	Ver.	Shard	Logging
□	all_users	3600	⊞	Any	100	📷
□	logged_in_users	600	⊞	Any	100	📷
□	network_connections	600	⊞	Any	100	📷
□	processes	600	⊞	Any	100	📷
□	scheduled_tasks	3600	⊞	Any	100	📷
□	services	3600	⊞	Any	100	📷
□	startup_items	3600	⊞	Any	100	📷
□	temp_folder	3600	⊞	Any	100	📷

We should see the data appearing in the Kibana Discover tab, however, we will soon realize that not all of the data is presented in a usable form, for example a list of unique endpoints that are running a binary with a particular hash. This is where Kibana visualizations come to our aid. In general, visualizations are graphical elements behaving according to pre-configured filters and search queries, that make it much quicker and easier to obtain readable and actionable data about the incident. We will mostly be using the Data Table visualization, but the setup principles are very similar for any other visualization type. Let us start with creating a Data Table visualization displaying scheduled tasks.



Visualizations

Title	Type	Description	Actions
Bytes Timeline [Filebeat NATS] ECS	Line		
Dashboard Navigation [Filebeat CEF]	Markdown		
Dashboard Navigation [Filebeat CEF]	Markdown		



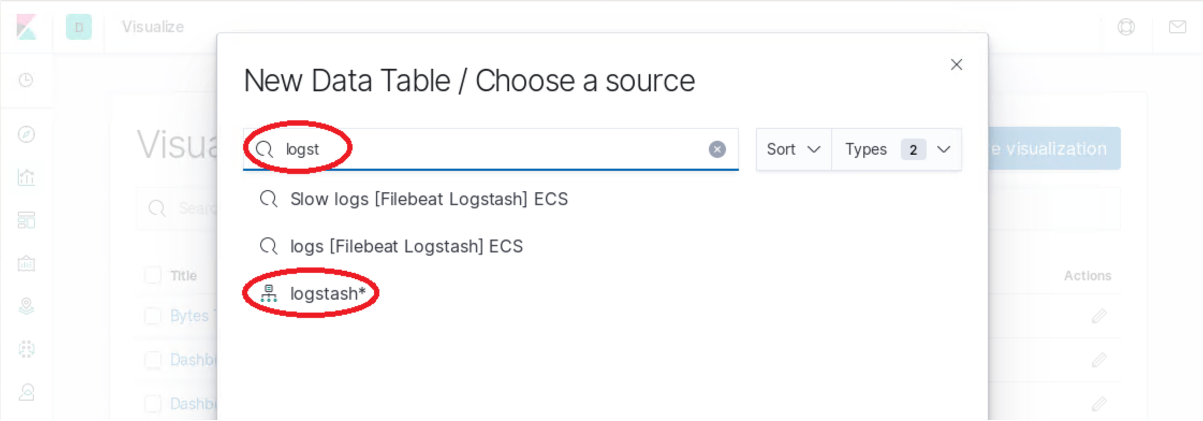
New Visualization

Select a visualization type

Start creating your visualization by selecting a type for that visualization.

Try Lens, our new, intuitive way to create visualizations.

Enter the name of the index pattern as the data source.



New Data Table / Choose a source

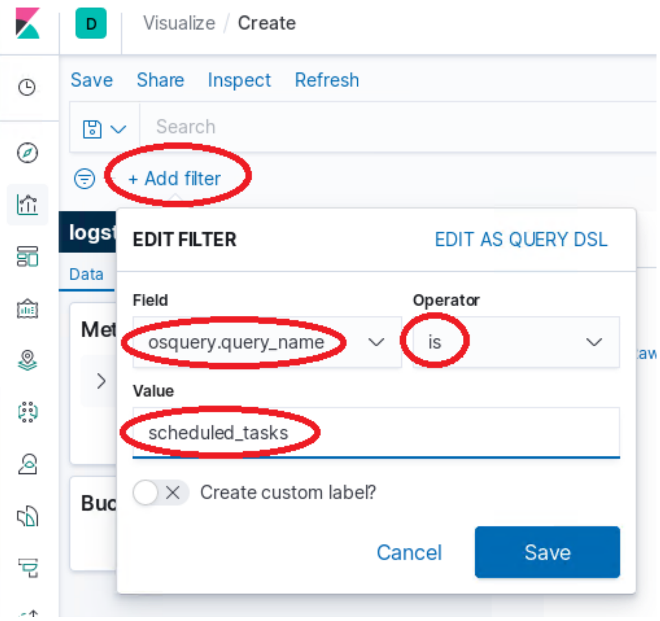
logst

Slow logs [Filebeat Logstash] ECS

logs [Filebeat Logstash] ECS

logstash*

We will be presented with an empty data table. We must configure three things – filter, metrics and buckets. Since we want this table to contain only the scheduled tasks data, we set the filter to include only the data returned by that particular query and we will use the *osquery.query_name* field, that we have created in the Logstash pipeline.



EDIT FILTER

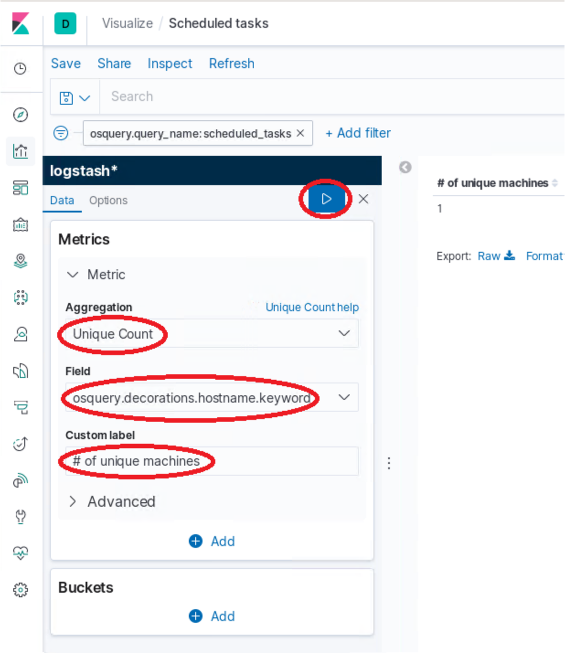
Field: osquery.query_name

Operator: is

Value: scheduled_tasks

Create custom label?

Then, we set the table metric, which in most cases will be the count of unique endpoints. We can create the *Apply changes* button to see the effect.



osquery.query_name: scheduled_tasks

logstash*

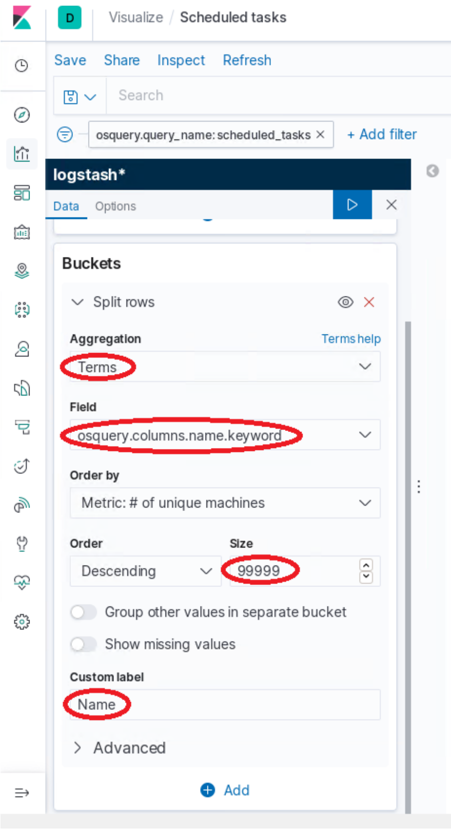
Aggregation: Unique Count

Field: osquery.decorations.hostname.keyword

Custom label: # of unique machines

of unique machines
1

Finally, we need to add the actual data to the display, i.e. a column that will display the name of the scheduled task. We do that by adding a "split rows" bucket, which will be aggregated by *Terms* on the *osquery.columns.name.keyword* field. Thanks to the filter we applied at the top, the *name* field from other queries will not scramble the data for this table. We want to be able to see all available rows, so we set the *Size* to a large number.



Aggregation: Terms

Field: osquery.columns.name.keyword

Order by: Metric: # of unique machines

Order: Descending

Size: 99999

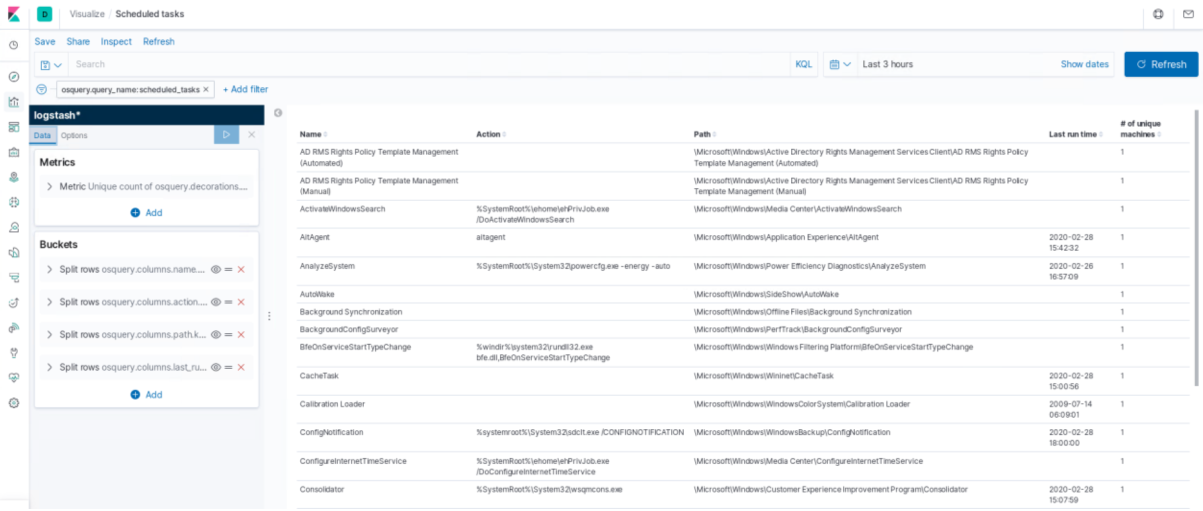
○ Group other values in separate bucket

○ Show missing values

Custom label: Name

After applying the settings, the new column will be added to the table. After creating a new bucket for every data column of the original query that we want to display, we will have the final look of the table.

Note that we did not display the *state* column in the table, and we are leaving the refinement to the analyst using the main search bar. We can also configure the number of displayed rows in the *Options* tab.



Metrics

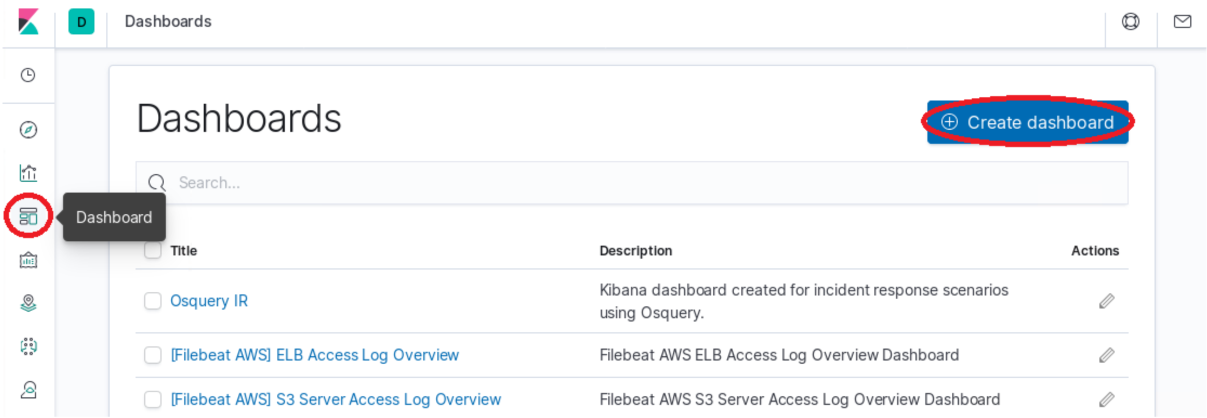
- Metric Unique count of osquery.decorations....

Buckets

- Split rows osquery.columns.name....
- Split rows osquery.columns.action....
- Split rows osquery.columns.path.k...
- Split rows osquery.columns.last_ru...

Name	Action	Path	Last run time	# of unique machines
AD RMS Rights Policy Template Management (Automated)		\Microsoft\Windows\Active Directory Rights Management Services Client\AD RMS Rights Policy Template Management (Automated)		1
AD RMS Rights Policy Template Management (Manual)		\Microsoft\Windows\Active Directory Rights Management Services Client\AD RMS Rights Policy Template Management (Manual)		1
ActivateWindowsSearch	%SystemRoot%\ehome\ehPrivJob.exe /DoActivateWindowsSearch	\Microsoft\Windows\Media Center\ActivateWindowsSearch		1
AitAgent	aitagent	\Microsoft\Windows\Application Experience\AitAgent	2020-02-28 15:42:32	1
AnalyzeSystem	%SystemRoot%\System32\powercfg.exe -energy -auto	\Microsoft\Windows\Power Efficiency Diagnostics\AnalyzeSystem	2020-02-26 16:57:09	1
AutoWake		\Microsoft\Windows\SideShow\AutoWake		1
Background Synchronization		\Microsoft\Windows\Offline Files\Background Synchronization		1
BackgroundConfigSurveyor		\Microsoft\Windows\PerfTrack\BackgroundConfigSurveyor		1
BfeOnServiceStartTypeChange	%windir%\system32\rundll32.exe bfe.dll,BfeOnServiceStartTypeChange	\Microsoft\Windows\Windows Filtering Platform\BfeOnServiceStartTypeChange		1
CacheTask		\Microsoft\Windows\Wininet\CacheTask	2020-02-28 15:00:56	1
Calibration Loader		\Microsoft\Windows\WindowsColorSystem\Calibration Loader	2009-07-14 06:09:01	1
ConfigNotification	%systemroot%\System32\sdclt.exe /CONFIGNOTIFICATION	\Microsoft\Windows\WindowsBackup\ConfigNotification	2020-02-28 18:00:00	1
ConfigureInternetTimeService	%SystemRoot%\ehome\ehPrivJob.exe /DoConfigureInternetTimeService	\Microsoft\Windows\Media Center\ConfigureInternetTimeService		1
Consolidator	%SystemRoot%\System32\wsqmcons.exe	\Microsoft\Windows\Customer Experience Improvement Program\Consolidator	2020-02-28 15:07:59	1

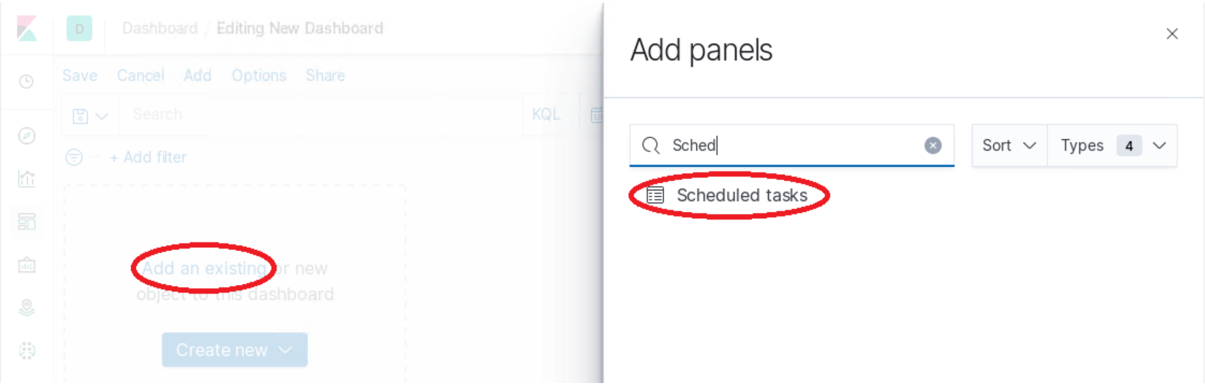
Do not forget to save the created visualization in the top left corner and make note of the title. Now we can move to the Kibana Dashboard tab and start creating a new dashboard, which will be the screen the analyst will work out of.



Dashboards

Title	Description	Actions
Osquery IR	Kibana dashboard created for incident response scenarios using Osquery.	
[Filebeat AWS] ELB Access Log Overview	Filebeat AWS ELB Access Log Overview Dashboard	
[Filebeat AWS] S3 Server Access Log Overview	Filebeat AWS S3 Server Access Log Overview Dashboard	

We want to add the created *Scheduled tasks* visualization:

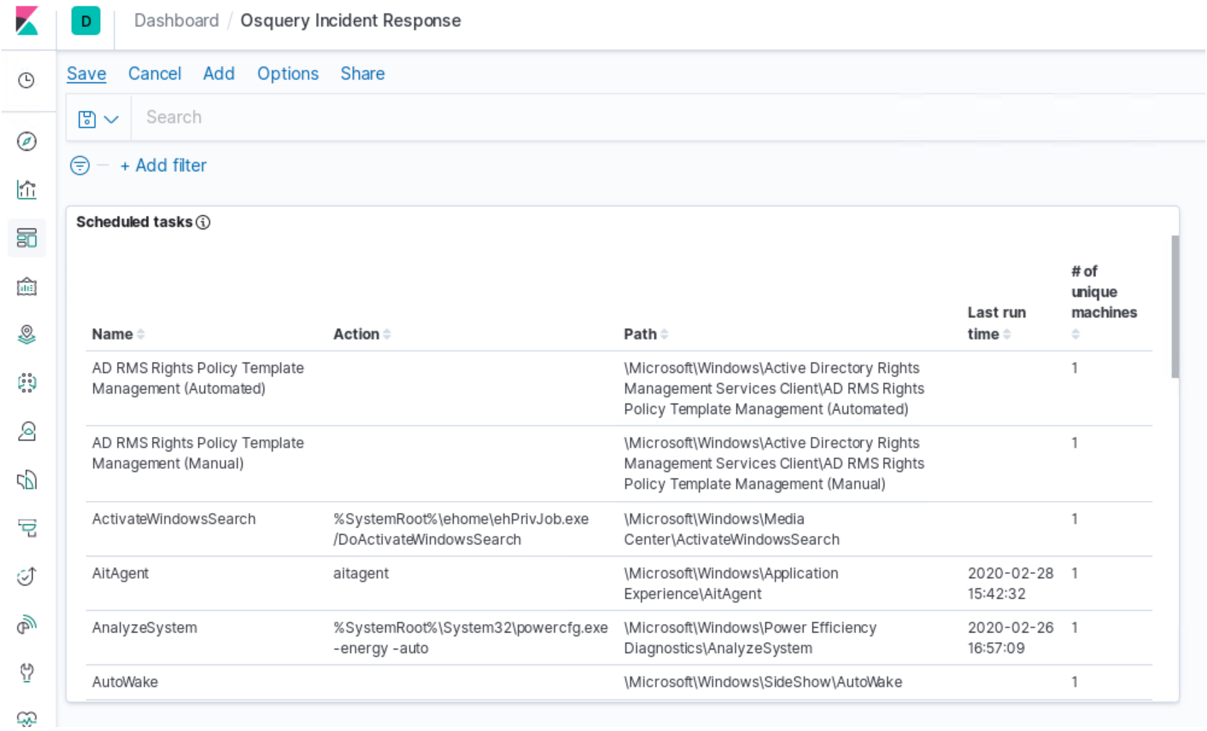

D
Dashboard / Editing New Dashboard
Save
Cancel
Add
Options
Share
Search
KQL
+ Add filter
Add an existing or new object to this dashboard
Create new
Sort
Types 4

Add panels

Sched

Scheduled tasks

And there it is! We save the dashboard and go create visualizations for the other queries. We also create some helper tables, such as list of all endpoints. Get creative in thinking of new ways of correlating and presenting the data :)

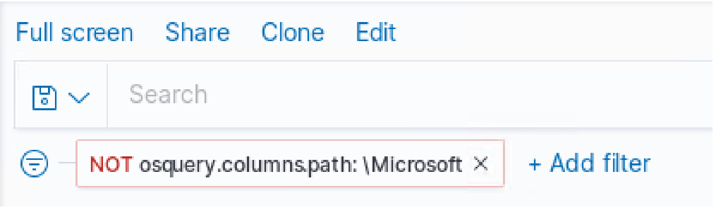

Dashboard / Osquery Incident Response
Save Cancel Add Options Share
Search
+ Add filter

Scheduled tasks

Name	Action	Path	Last run time	# of unique machines
AD RMS Rights Policy Template Management (Automated)		\Microsoft\Windows\Active Directory Rights Management Services Client\AD RMS Rights Policy Template Management (Automated)		1
AD RMS Rights Policy Template Management (Manual)		\Microsoft\Windows\Active Directory Rights Management Services Client\AD RMS Rights Policy Template Management (Manual)		1
ActivateWindowsSearch	%SystemRoot%\ehome\ehPrivJob.exe /DoActivateWindowsSearch	\Microsoft\Windows\Media Center\ActivateWindowsSearch		1
AitAgent	aitagent	\Microsoft\Windows\Application Experience\AitAgent	2020-02-28 15:42:32	1
AnalyzeSystem	%SystemRoot%\System32\powercfg.exe -energy -auto	\Microsoft\Windows\Power Efficiency Diagnostics\AnalyzeSystem	2020-02-26 16:57:09	1
AutoWake		\Microsoft\Windows\SideShow\AutoWake		1

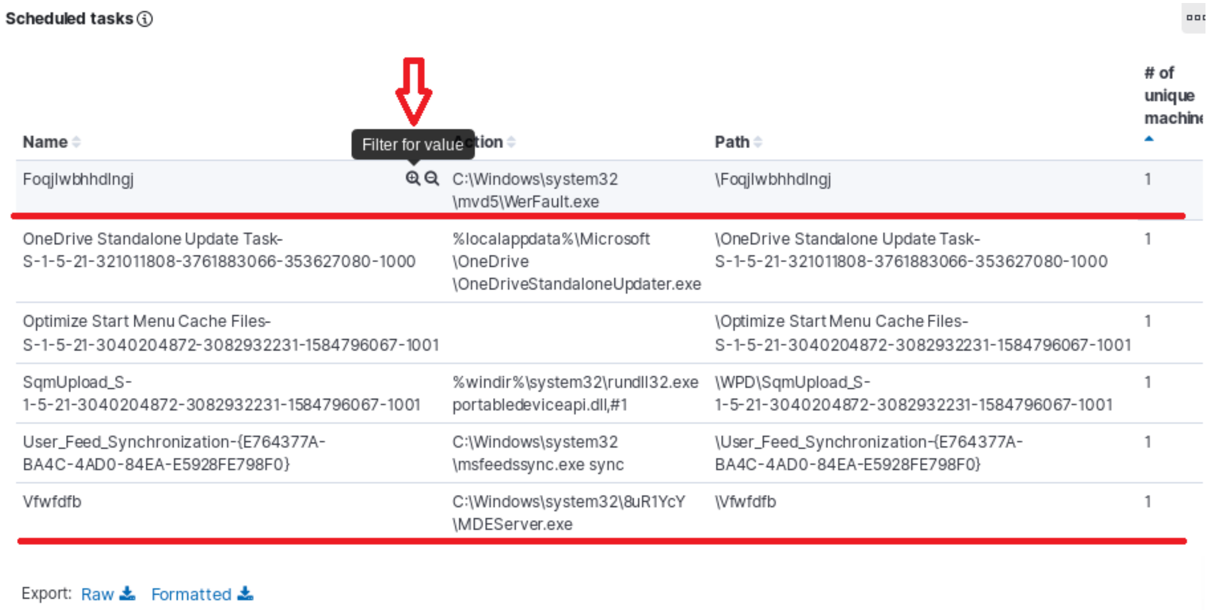
INCIDENT TRIAGE

We have chosen the Dridex banking malware as an example to demonstrate how we can use Osquery to detect a malware compromise in our network. We run the Dridex sample on one of our victim machines and after letting the Osquery collect the data, we can start looking at the tables in our Dashboard. Let's start with, *Scheduled tasks*. The *|Microsoft* tasks seem to be legitimate, so we can filter them out:



NOT osquery.columns.path: \Microsoft

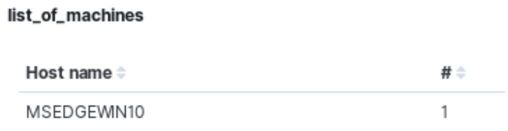
This immediately reduces the number of entries significantly. And we see that there is something fishy…



Scheduled tasks ⓘ

Name	Action	Path	# of unique machine
Foqjlwbhhdlngj	C:\Windows\system32 \mvd5\WerFault.exe	\Foqjlwbhhdlngj	1
OneDrive Standalone Update Task- S-1-5-21-321011808-3761883066-353627080-1000	%localappdata%\Microsoft \OneDrive \OneDriveStandaloneUpdater.exe	\OneDrive Standalone Update Task- S-1-5-21-321011808-3761883066-353627080-1000	1
Optimize Start Menu Cache Files- S-1-5-21-3040204872-3082932231-1584796067-1001		\Optimize Start Menu Cache Files- S-1-5-21-3040204872-3082932231-1584796067-1001	1
SqmUpload_S- 1-5-21-3040204872-3082932231-1584796067-1001	%windir%\system32\rundll32.exe portabledeviceapi.dll,#1	\WPD\SqmUpload_S- 1-5-21-3040204872-3082932231-1584796067-1001	1
User_Feed_Synchronization-{E764377A- BA4C-4AD0-84EA-E5928FE798F0}	C:\Windows\system32 \msfeedssync.exe sync	\User_Feed_Synchronization-{E764377A- BA4C-4AD0-84EA-E5928FE798F0}	1
Vfwfdfb	C:\Windows\system32\8uR1YcY \MDEServer.exe	\Vfwfdfb	1

Two entries do not look normal. In order to find which machine a particular entry belongs to; we click the small *Filter for value* button and take a look at the table with a list of endpoints:

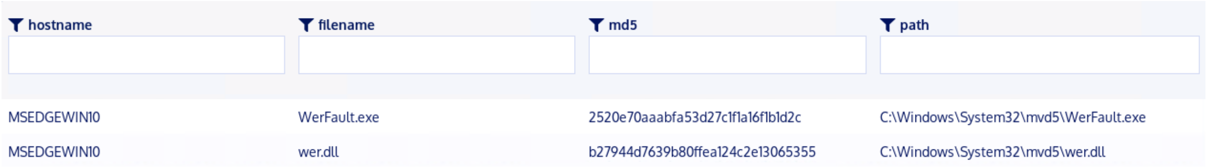


list_of_machines

Host name	#
MSEDGEWIN10	1

And we see that it is our Windows 10 endpoint. We now delete the generate the filter and replace it with one filtering data for the *MSEDGEWIN10* machine. However, looking at the process table, we see that there is no process running from the suspicious location that we found in the scheduled tasks table, which probably means that it has already finished execution. We need to pull additional data from the endpoint. To do this, we switch to Kolide Fleet and create a new query, which will show us the contents of the suspicious folder and calculates hashes of anything found within:

```
SELECT file.path, file.filename, hash.md5 FROM file JOIN hash WHERE file.path LIKE
“C:\Windows\system32\mvd5\%%” AND hash.path LIKE file.path;
```

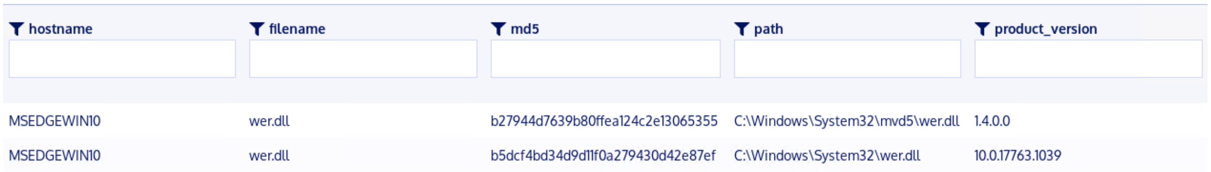


hostname	filename	md5	path
MSEDGEWIN10	WerFault.exe	2520e70aaabfa53d27c1f1a16f1b1d2c	C:\Windows\System32\mvd5\WerFault.exe
MSEDGEWIN10	wer.dll	b27944d7639b80ffea124c2e13065355	C:\Windows\System32\mvd5\wer.dll

We can upload the hashes to Virustotal or another AV API and most likely see, that the hash is not yet present in the Virustotal database. This adds to our suspicion, because the names of the files are those of legitimate OS files, which are supposed to be located only in the System32 folder. If we upload the DLL to the Virustotal, we see that we indeed have a problem with the number of AVs complaining about Dridex. The rogue wer.dll masquerades as a legitimate operating system .dll file, which can normally be found in the System32 folder. We can also see, that the .exe file present in the folder is just a copy of a legitimate and signed Windows executable. Both these claims can be proven by running a query, such as:

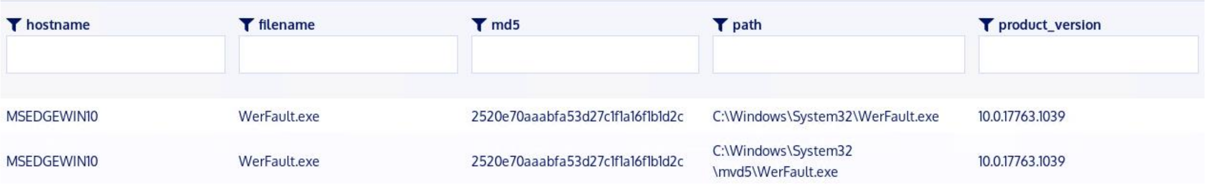
```
SELECT file.path, file.filename, file.product_version, hash.md5 FROM file JOIN hash
WHERE (file.path='C:\Windows\System32\wer.dll' OR
file.path='C:\Windows\System32\mvd5\wer.dll') AND hash.path LIKE file.path;
```

Which will give us the result:



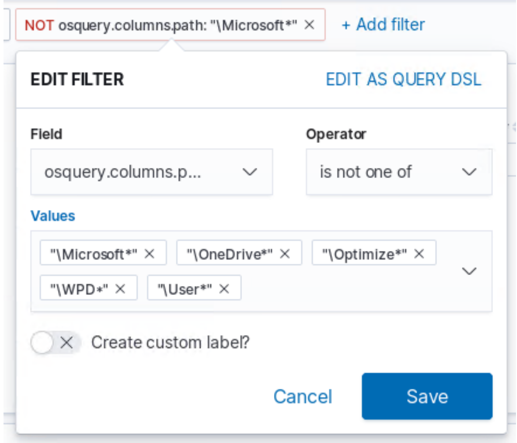
hostname	filename	md5	path	product_version
MSEDGEWIN10	wer.dll	b27944d7639b80ffea124c2e13065355	C:\Windows\System32\mvd5\wer.dll	1.4.0.0
MSEDGEWIN10	wer.dll	b5dcf4bd34d9d11f0a279430d42e87ef	C:\Windows\System32\wer.dll	10.0.17763.1039

We can also see that the DLL in the System32 folder has no detections on Virustotal and is signed by Microsoft. We can do the same check for the EXE file, which will tell us that the suspicious EXE file is just a copy of a legitimate OS file:



hostname	filename	md5	path	product_version
MSEDGEWIN10	WerFault.exe	2520e70aaabfa53d27c1f1a16f1b1d2c	C:\Windows\System32\WerFault.exe	10.0.17763.1039
MSEDGEWIN10	WerFault.exe	2520e70aaabfa53d27c1f1a16f1b1d2c	C:\Windows\System32 \mvd5\WerFault.exe	10.0.17763.1039

We can also see from our Kibana Dashboard that the Windows 10 endpoint is most likely not the only one compromised, as there is one more suspicious scheduled task. In order to confirm the list of compromised devices, we use the Kibana filter to filter out the other legitimate scheduled tasks, until all the clutter is gone:



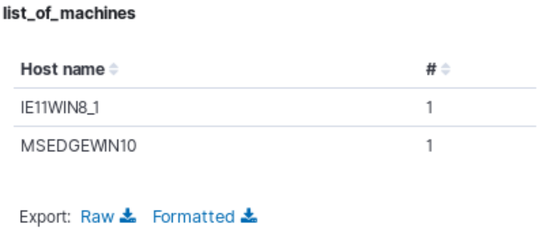
NOT osquery.columns.path: "\Microsoft*"

Field: osquery.columns.p...

Operator: is not one of

Values: "\Microsoft*" "\OneDrive*" "\Optimize*" "\WPD*" "\User*"

After that, the *Scheduled tasks* table contains only the malicious entries, showing paths to the malware folders, which we can export to a file. At the same time, the "List of machines" table shows only the infected endpoints, which can now be isolated from the network and their condition remediated.



list_of_machines

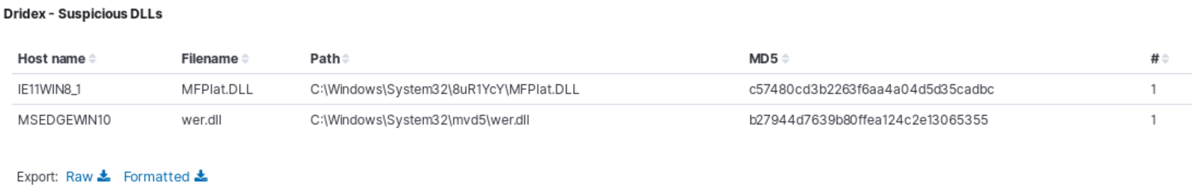
Host name	#
IE11WIN8_1	1
MSEDGEWIN10	1

Alternatively, we can build a new Osquery query to draw some more data. We cannot know what randomly generated folders with the malicious DLL there will be on the endpoints, so we will have to perform a wildcard search. This means getting all DLLs from all subfolders of System32, which is likely to be a large dataset, so expect long execution times. We know that the malicious DLLs are located in a level 1 subfolder of System32, so we can limit the recursion depth with an appropriate use of the wildcards. In case of a large network, consider scheduling the same query in multiple packs, each targeting only a smaller subset of the endpoints. Since we already know from Virustotal that we are dealing with Dridex and that recent samples are polymorphic, we cannot match for the found DLL hash in the query. We have to get all the DLLs along with their hashes, optionally filter the data clutter as previously, export the remaining suspicious hashes and put them into Virustotal or another behavioral analysis tool. So, our query will be:

```
SELECT file.filename, file.path, hash.md5 FROM file JOIN hash WHERE file.path LIKE
“C:\Windows\System32\%\%.dll” AND hash.path LIKE file.path;
```

Testing the query through the Kolide Fleet on our three endpoints yields 775 results. We save the query and create a new pack. We schedule the query to run not too often. After we see the data from this query incoming in the Kibana Discovery tab we create a new Data Table visualization and add it into our incident response Dashboard. Since the new query might have induced new data columns, we must refresh the Kibana index pattern in order to see the new columns. If needed, we could create a new Dashboard just for Dridex IOCs.

If we had used Logstash pipeline to forward the MD5 hashed to an AV API and then used another pipeline to collect the scan results for every DLL, we could filter just those entries that have a non-zero number of elements. Because we do not have this luxury, we must use the filters on the “path” column to weed out known legitimate system folders. It might be worthwhile to save the filter string, so we do not have to write it the next time. After we are done, we have the list of the malicious DLLs along with their hashes and infected host names:



Dridex - Suspicious DLLs

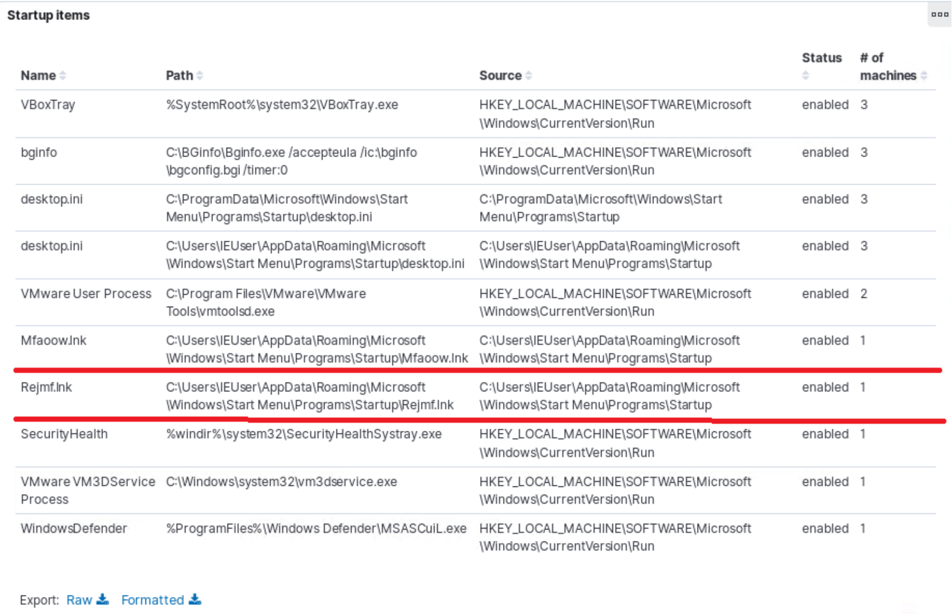
Host name	Filename	Path	MD5	#
IE11WIN8_1	MFPlat.DLL	C:\Windows\System32\8uR1YcY\MFPlat.DLL	c57480cd3b2263f6aa4a04d5d35cadbc	1
MSEDGEWIN10	wer.dll	C:\Windows\System32\mvd5\wer.dll	b27944d7639b80ffea124c2e13065355	1

We could still broaden this activity by searching for all DLL and EXE files in the first-level subfolders and searching for their namesake in the System32 folder using our Osquery-fu:

```
SELECT f1.filename, h2.sha256, 'Yes' as MATCH
FROM (SELECT f.filename, f.path, h.sha256 FROM file f JOIN hash h ON f.path=h.path
WHERE f.path like 'C:\Users\%\AppData\Roaming\%\%' AND f.filename LIKE '%.exe') as f1,
(SELECT f.filename, f.path, h.sha256 FROM file f JOIN hash h ON f.path=h.path WHERE
f.path = 'C:\Windows\System32\dpapimig.exe') as h2
WHERE CASE WHEN f1.filename = h2.filename THEN 1 ELSE 0 END;
```

But we will not do that here so let’s just say that it would not give us any new results.

Now let’s try and delete the scheduled task from the machine using the administrative tools. Also delete the malicious files… Looks like we wo… Oh, they just reappeared! This means there is another mechanism keeping the persistence. Let’s take a look at the *Startup items* table:

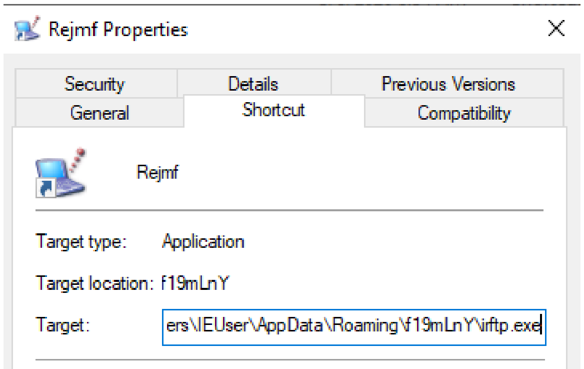


Startup items

Name	Path	Source	Status	# of machines
VBoxTray	%SystemRoot%\system32\VBoxTray.exe	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	enabled	3
bginfo	C:\BGinfo\Bginfo.exe /accepteula /ic:\bginfo\bgconfig.bgi /timer:0	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	enabled	3
desktop.ini	C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\desktop.ini	C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup	enabled	3
desktop.ini	C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\desktop.ini	C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup	enabled	3
VMware User Process	C:\Program Files\VMware\VMware Tools\vmtoolsd.exe	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	enabled	2
Mfaoow.lnk	C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Mfaoow.lnk	C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup	enabled	1
Rejmf.lnk	C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Rejmf.lnk	C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup	enabled	1
SecurityHealth	%windir%\system32\SecurityHealthSystray.exe	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	enabled	1
VMware VM3DService Process	C:\Windows\system32\vm3dservice.exe	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	enabled	1
WindowsDefender	%ProgramFiles%\Windows Defender\MSASCuiL.exe	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	enabled	1

Export: Raw Formatted

There are two items that look out of the ordinary. Since we have direct access to the infected machines, we can take a look at the target of one of these suspicious LNK files:



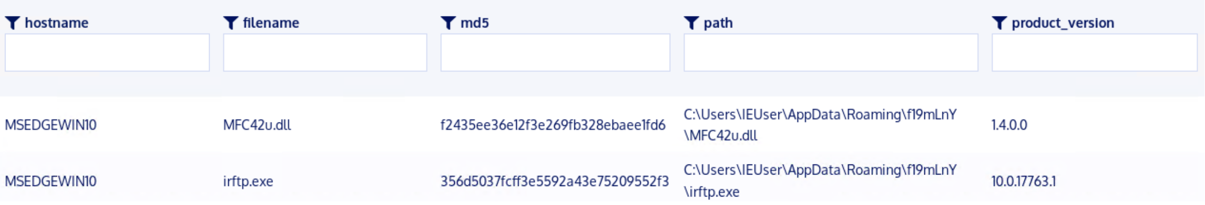
Rejmf

Target type: Application

Target location: f19mLnY

Target: ers\IEUser\AppData\Roaming\f19mLnY\irftp.exe

We will find another suspicious randomly named folder *C:\Users\IEUser\AppData\Roaming\f19mLnY*, which we could enumerate for its contents:



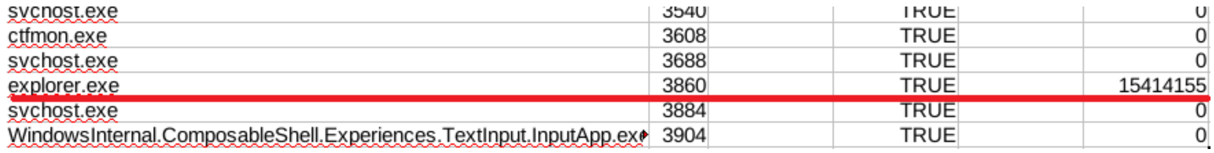
hostname	filename	md5	path	product_version
MSEDGEWIN10	MFC42u.dll	f2435ee36e12f3e269fb328ebaee1fd6	C:\Users\IEUser\AppData\Roaming\f19mLnY\MFC42u.dll	1.4.0.0
MSEDGEWIN10	irftp.exe	356d5037fcff3e5592a43e75209552f3	C:\Users\IEUser\AppData\Roaming\f19mLnY\irftp.exe	10.0.17763.1

And we see another pair of malicious DLL and out-of-place operating system executable. As expected from a polymorphic malware, the hash of the DLL will be different from the one found in the System32 subfolder. We have also made an important finding about the malware – It abuses the way how Windows searches for DLLs when an EXE is run – it looks into the current directory first. Also, deleting the files will only make them reappear again.

What is the mechanism re-creating the malicious files we found? Unfortunately, the process open_files and file_events Osquery tables are not supported for Windows. Time to get creative! Obviously, the process has to be writing to the hard drive. We know the exact size of the data being written. And the *processes* Osquery table contains a *disk_bytes_written* field! This means we can take a snapshot of this value before deleting the files and again right after they reappear. Then, we will calculate the difference between the two value of every process and look for a value matching, or being close to, the total size of the files. Before we continue, there is an important point to make – Osquery is blind to anything happening between the snapshots. If the persistence if maintained by some repeating short-duration process, we will not see it. Let's hope it is a continuously running process. We are still manually querying the Windows 10 endpoint from the Kolide Fleet interface and our query will be:

```
SELECT pid, name, disk_bytes_written FROM processes;
```

To make the calculated difference stand out more, we will delete all four files (two DLL, two EXE) five times in a row before taking the second snapshot. The sum size of the four files we will be deleting is 3.325.952 bytes. We make sure to be subtracting the values for the same PID as the number of processes can change, the non-matching processes must be filtered out. After performing the two queries as described and calculating the differences, we are looking for an outstanding value of five times the above-mentioned size, so approx. 15 MB in total.



svchost.exe	3540	TRUE	0
ctfmon.exe	3608	TRUE	0
svchost.exe	3688	TRUE	0
explorer.exe	3860	TRUE	15414155
svchost.exe	3884	TRUE	0
WindowsInternal.ComposableShell.Experiences.TextInput.InputApp.exe	3904	TRUE	0

And it looks like the culprit is none other than *explorer.exe* itself! The total value of written bytes is almost an exact match to what we were expecting, with no other process coming close. This means that the persistence maintaining mechanism is injected in the explorer process. We have learned quite a lot about how the Dridex sample operates using just Osquery and the ELK Stack! At this point, we could secure the malware samples and hand them over to our malware analyst buddies.

SUMMARY

From the setup and deployment of the instrumentation to finding the IOCs – we have demonstrated how can incidents responders expand their toolset with Osquery coupled to ELK Stack. Kolide Fleet was deployed for managing our endpoints and for creating, running and scheduling Osquery queries. Then we deployed the Elasticsearch, Logstash, Kibana trio and explained the configuration steps. Special focus was on the configuration of the Logstash pipeline, serving as the interface between the Osquery output and Elasticsearch input. We have also covered in detail how to prepare the Kibana interface for actual use, which we have demonstrated by performing an incident response of Dridex infected machines. Using a few Kibana visualizations a some Osquery queries a number of IOCs were identified – Suspicious scheduled tasks, startup items and malicious DLLs. We even delved into the initial behavioral analysis of the malware sample by uncovering its persistence mechanism – injecting a thread into the explorer.exe process. The tools presented in this case study can be a valuable addition to the incident team arsenal and what is even better – their capabilities can be further improved and expanded.