ThreatQuotient

ELK Stack Guide

Version 2.0.0

Thursday, September 24, 2020

ThreatQuotient

11400 Commerce Park Dr., Suite 200

Reston, VA 20191

Support

Email:	support@threatq.com
Web:	Support.threatq.com
Phone:	703.574.9893

Warning and Disclaimer

Contents

Versioning

- Integration Version: 2.0.0
- ThreatQ Version: 4.24.0 or greater

Introduction

ELK is an extremely flexible document store and search platform that ingests files and data from a myriad of sources. In today's changing cybersecurity context, ELK is becoming a popular SIEM. Given ELK's flexibility in design and ingestion of data, a bi-directional integration can be performed in many ways.

This particular integration provides several key benefits, one of those is the maintainability and ease of installation for customers.

This integration uses the following integration points:

- Logstash Translation Filters
- Logstash Clone filters
- Elasticsearch threatq index
- Elasticsearch threatq-sightings indices
- Xpack Watcher Alert API

The integration works by installing the python wheel file onto the ELK server itself and provides 4 commands once installed. These commands:

- Synchronize data in a saved Indicator Search with the threatq index
- Get triggered alerts from the Xpack Watcher API and store them in ThreatQ
- Uses the same saved Indicator Searches to create files that can be used via Logstash's Translation filter
- Update the ThreatQ template in Elasticsearch, in the event the customer has run v1.x of the connector, but is now on v7.x of ELK

Uses the same saved Indicator Searches to create files that can be used via Logstash's Translation filter

Each level of the integration is optional, and customers can choose to integrate all three (recommended) or integrate only those pieces that fulfill their operational needs.

Preface

The purpose of this ThreatQuotient For ELK Stack installation guide is to provide the information necessary to implement the ThreatQuotient For ELK Stack.

Although it may be used as such, this document is not specifically intended to form a site reference guide.

It is assumed that the implementation engineer has experience installing and commissioning ThreatQuotient Apps and integrations covered within the document, as well as experience necessary to troubleshoot at a basic level

Audience

This document is intended for use by the following parties:

1. ELK system administrators & engineers
2. Security engineers

Assumptions

The following criteria is assumed to be in place and functional to allow the implementation of the ThreatQuotient into the managed estate:

- All ThreatQuotient equipment/instances are online and in service.
- All required firewall ports have been opened.

Destination	Port	Description
ThreatQ	443	Connections to ThreatQ via SSL
ELK Appliance	9200	Connections from where the integration is installed and Elasticsearch

- All equipment is powered from permanent power supplies
- A clock source of sufficient accuracy is connected to the network and the network is using it as the primary clock source.

If an Xpack license is available, then the alerts API can be used. If not, then the alerts API is unavailable.

This integration is only suitable for RHEL and RHEL variants (CentOS, Scientific Linux, etc). As such, only ELK servers of this type are supported. Contact support@threatq.com to see if another Linux variant is supported.

Passwords have not been provided in this document. Please contact your project team for this information, if required.

Prerequisites

Throughout this implementation document, there will be referrals to several files and directories, some of which will be symbolic, and others may change depend on specifics of the environmental setup.

Ensure all ThreatQ devices are set to the correct time (UTC is recommended), time zone and date, and using a clock source available to all.

To identify which time zone is closest to your present location, use the timedatectl command with the list-timezones command line option. For example, to list all available time zones in Europe, type:

```
timedatectl list-timezones | grep Europe

Europe/Amsterdam


Europe/Athens


Europe/Belgrade


Europe/Berlin
```

To change the time zone to UTC, type as root:

```
timedatectl set-timezone UTC
```

Recommended user accounts that are required for the installation

Example Username	System	Description
elk@<company>.com	ThreatQ	A dedicated user for the ELK integration is recommended. This user is where the indicator searches will be created and are the credentials to use during installation
threatq@<company>.com	ELK	If Xpack Security is being used, this would be a user that has access to read, write, create, and delete indices, as well as have access to the Watcher API. This is the user that ThreatQ will use to synchronize ThreatQ data in ELK

Installation

The ELK Stack integration is meant to be installed on the ELK Appliance and not the ThreatQ appliance.

It is possible to install this onto the ThreatQ platform specifically, but a provision would need to be made for a mechanism to move the created translation files from the ThreatQ instance to the ELK Logstash server(s).

Prerequisites

To install this connector on the ELK server, there are dependencies that we will need to install.

- epel-release
- gcc
- python-devel
- openssl-devel

There is also sveral logstash plugins that will also need to be installed.

- logstash-filter-kv
- logstash-filter-translate
- logstash-filter-prune
- logstash-filter-clone

To install all of these, please refer to the commands below:

```
sudo yum install -y epel-release gcc python-devel openssl-devel

sudo yum install -y python-pip

/usr/share/logstash/bin/logstash-plugin install logstash-filter-kv

/usr/share/logstash/bin/logstash-plugin install logstash-filter-translate

/usr/share/logstash/bin/logstash-plugin install logstash-filter-prune

/usr/share/logstash/bin/logstash-plugin install logstash-filter-clone
```

Install Methods

The ELK connector can be installed using the following methods:

- via ThreatQ Repository
- via .whl File

via ThreatQ Repository

1. Run the following command:

```
pip install -I

https://<USERNAME>:<PASSWORD>@extensions.threatq.com/threatq/integrations tq-

conn-elk
```

via .whl File

1. Run the following command:

```
pip install tq_conn_elk-2.0.0-py2-none-any.whl
```

Post Install

After installing the connector, you will need to create the directories that will be used in the application. See the example below:

```
mkdir -p /opt/tq-integrations/elk

mkdir -p /opt/tq-integrations/elk/config

mkdir -p /opt/tq-integrations/elk/logs

mkdir -p /opt/logstash_filters

mkdir -p /optlogstash_inputs
```

First Run

After installing the connector, you will need to run it for the first time, manually. This will install the connector into the ThreatQ UI so it can be configured.

```
tq-elk-sync-data -c /opt/tq-integrations/elk/config/ -ll /opt/tq-

integrations/elk/logs/ -v3
```

After issuing the above command, you will be asked the following questions to initialize the integration:

Prompt	Description
ThreatQ Host	Hostname or IP Address of the ThreatQ server. *If this is a hostname, it must be resolvable on the installation point.*
Client ID	This is the OAuth Management value found in **User bubble** → **My Account** → API Credentials
E-Mail Address	This is the e-mail address of the ThreatQ user for this integration
Password	This is the password for the above ThreatQ user
Status	This is the default status of newly created IoCs from this integration

```
$> tq-elk-sync-data -c /opt/tq-integrations/elk/config/ -ll /opt/tq-

integrations/elk/logs/ -v 3


ThreatQ Host: <ThreatQ Host IP or Hostnme>


Client ID: <ClientID>


E-Mail Address: <EMAIL ADDRESS>


Password: <PASSWORD>


Status: Active


Connector configured.  Set information in UI.
```

Configuration

To configure the connector, navigate to the ThreatQ Instance.

> ***Note:*** *ThreatQuotient does not issue API keys for third-party vendors. Contact the specific vendor to obtain API keys and other feed-related credentials.*

To configure the connector:

1. Click on the **Settings** icon and select **Incoming Feeds**.
2. Locate the connector under the **Lab** tab.
3. Click on the **Feed Settings** link for the connector.
4. Under the Connection tab, enter the following configuration parameters:

Parameter	Description
Elasticsearch Hosts	This is a comma separated list of Elasticsearch hosts for use with this integration. These hosts should be hostnames or IP Addresses that can be resolved/reached from the Installation Point. These should be in the form hostname:port or ip address:port. For instance, if your Elasticsearch node is at 192.168.0.2 and running the default port of 9200, then 192.168.0.2:9200 should be entered.
Elasticsearch Username	If Xpack Security is being used, this supports HTTP Basic Auth style authentication. This would be the Elasticsearch User that should be used to synchronize data between the systems. If Xpack Security is disabled, leave this blank.
Elasticsearch Password	If the above Username is provided, then the password must also be provided. This can be left blank if Xpack Security is disabled.
Saved Threat Library Searches	This is a comma separated list of saved Threat Library Searches (**Threat Library** → **Indicators**) that should be used as the export to synchronize data with.

Parameter	Description
Attribute Filter	This is a comma separated list of Attribute Names, which are attributes you want sent to ELK. If left blank, all related attributes will be imported to Elasticsearch.
Alert Parsing Config	See the Alert Parsing Configuration section
Elasticsearch SSL CA Certificate Location	This is the location to the Elasticsearch SSL certificate.

5. Click on **Save Changes**.
6. Click on the toggle switch to the left of the connector name to enable the connector.

Executing the Connector

Once the connector is installed and configured, you can now successfully run 4 commands. These commands are as follows:

- `tq-elk-sync-data`
- `tq-elk-create-enrichment-files`
- `tq-elk-get-alerts`
- `tq-elk-update-template`

To sync threat intelligence from ThreatQ to ELK run the following command:

```
tq-elk-sync-data -c /opt/tq-integrations/elk/config/ -ll /opt/tq-
integrations/logs/ -v3
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver DEBUG: Private Connection
Established
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Established Connection to
ElasticSearch
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Established ThreatQ Index
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Starting to compare search
active for synchronization
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Compared 1000 indicators
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Compared 2000 indicators
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed Comparing a total
of 27711 indicators
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Beginning Synchronization
of 27711 IoCs
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed Synchronization
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed execution of the
ELK Stack Connector in xx seconds
[root@ELK ~]#
```

Then you can run the following command to create the enrichment files:

```
tq-elk-create-enrichment-files -c /opt/tq-integrations/elk/config/ -ll
/opt/tq-integrations/logs/ --enrichment-dest /opt/logstash_filters/ -v3

xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver DEBUG: Private Connection
Established

xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver WARNING: No Enrichment
Destination Given, using current directory

xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Starting to download data
for search active

xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: 1000 IoC written to
tranlsation files

xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: 2000 IoC written to
tranlsation files

xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed execution of the
ELK Stack Connector in xx seconds
```

Then run the following command to get alerts from the ELK system.

```
tq-elk-get-alerts -c /opt/tq-integrations/elk/config/ -ll /opt/tq-
integrations/logs/ -v3
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver DEBUG: Private Connection
Established
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Established Connection to
ELK Stack
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Getting threatq-squid-
sightings alerts
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed getting threatq-
squid-sightings alerts
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed execution of the
ELK Stack Connector in xx seconds
```

Finally you can run the next command if you've run the old connector on ELK 7.x or have upgraded your ELK from v6.0 and ran the connector previously against this system

```
tq-elk-update-template -c /opt/tq-integrations/elk/config/ -ll /opt/tq-
integrations/elk/logs/ -v3
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver DEBUG: Private Connection
Established
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Successfully updated the
threatq template in Elasticsearch.
xxxx-xx-xx xx:xx:xx - tq_conn_elk.tq_driver INFO: Completed execution of the
ELK Stack Connector in xx seconds
```

Kibana Index Setup

Threatq* Index

After the `tq-elk-sync-data` command has been run, we can now create the index pattern for the ThreatQ data in Kibana.

1. From Kibana, navigate to Management → Index patterns
2. Search for **threatq***
3. Click Next step
4. Choose the configuration for Time Filter field name (default of created_at is fine)
5. Click **Create index pattern**

CRON

Automatic CRON configuration has been removed from this script. To run this script on a recurring basis use CRON or some other jobs scheduler. The argument in the CRON script must specify the config and log locations.

Add an entry to your Linux crontab to execute the connector at a recurring interval. Depending on how quickly you need updates, this can be run multiple times a day (no more than once an hour) or a few times a week.

Threat Intelligence data changes frequently, as new data enters the system and old data stales out, these changes should be exported to ELK.

To accomplish this, one of the three commands are executed via cron or some other scheduling system to repeatedly refresh the data. These commands are described below:

- `tq-elk-sync-data`: This synchronizes data between saved Indicator Searches and the threatq index in Elasticsearch
- `tq-elk-get-alerts`: This synchronized triggered alerts that have their internal conditions met with ThreatQ as Events of type Sighting.
- `tq-elk-create-enrichment-files`: This produces a set of files, based off of the saved Indicator Searches, to be used with Logstash Filters.

To set up the cron job. Follow the instructions below.

1. Log into your ThreatQ host via a CLI terminal session.
2. Enter the following command:

```
crontab -e
```

This will enable the editing of the crontab, using vi.

Depending on how often you wish the cronjob to run, you will need to adjust the time to suit the environment.

3. Enter the commands below:

This example shows ever 10, 15, and 20 minutes past the hour

```
10 * * * * tq-elk-sync-data -c /opt/tq-integrations/elk/config/ -ll /opt/tq-
integrations/logs/ -v3

15 * * * * tq-elk-get-alerts -c / opt/tq-integrations/elk/config/ -ll /
opt/tq-integrations/logs/ -v3

20 * * * * tq-elk-create-enrichment-files -c / opt/tq-
integrations/elk/config/ -ll / opt/tq-integrations/logs/ --enrichment-dest
/opt/logstash_filters/ -v3
```

4. Save and exit cron.

Command Line Arguments

This connector supports the following custom command line arguments:

Argument	Description
`--enrichment-dest`	Enrichment file destination that can be referenced by logstash filters.
`-n --name [connector name]`	This argument allows you to install the connector with a custom name This argument is mainly used when you want to install multiple instances of the connector. For instance, if you have multiple saved searches that you want exported. You can simply setup a new instance of this connector with a new name, for a new saved search

Appendix A: Squid Proxy Hit Example

This is an example of a Squid Proxy match. The text contained within has been redacted and should **only** server as a reference.

```
{

  "_index": ".watcher-history-7-2018.08.14",

  "_type": "doc",

  "_id": "threatq-squid-sightings_6d57dded-f167-43c2-ab62-c39a8726f327-2018-08-14T12:41:57.487Z",

  "_score": 1,

  "_source": {

    "watch_id": "threatq-squid-sightings",

    "node": "2QNng82MStyeJKKfyu8JgA",

    "state": "executed",

    "status": {

      "state": {

        "active": true,

        "timestamp": "2018-08-13T17:31:40.161Z"

      },

      "last_checked": "2018-08-14T12:41:57.487Z",

      "last_met_condition": "2018-08-14T12:41:57.487Z",

      "actions": {
        "my-logging-action": {

          "ack": {

            "timestamp": "2018-08-14T12:41:57.487Z",

            "state": "ackable"

          },

          "last_execution": {

            "timestamp": "2018-08-14T12:41:57.487Z",

            "successful": true

          },

          "last_successful_execution": {

            "timestamp": "2018-08-14T12:41:57.487Z",

            "successful": true

          }

        }

      },

      "execution_state": "executed",

      "version": -1

    },

    "trigger_event": {
```

```
    "type": "schedule",

    "triggered_time": "2018-08-14T12:41:57.486Z",

    "schedule": {

      "scheduled_time": "2018-08-14T12:41:57.363Z"

    }

  },

  "input": {

    "search": {

      "request": {

        "search_type": "query_then_fetch",

        "indices": [

          "squid-access-*"

        ],
```

Appendix B: Alert Parsing Configuration Example

This is an example of Xpack Alerting. The text contained within has been redacted and should **only** server as a reference.

XPack Alerting is extremely flexible, and as such, the parsing method for this extremely flexible parsing must also be flexible. To accommodate this flexibility, the **Alert Parsing Configuration** field has been added. In this field, the **Watcher ID** is given as the main key, and then the hits, the IoC description, and the IoC Attributes description is given so that hits can be processed properly.

As an example. It is assumed we have a Squid Proxy server and are monitoring the logs for matches against the ThreatQ data via the translation filters (refer to Logstash Filters Section). This allows us to use the following parsing configuration:

```
---

  threatq-squid-sightings:

    hits: "result.input.payload.hits.hits"

    attributes:

      _source.status_code: "ELK Squid Status Code"

      _source.path: "ELK Original File Path"

      _source.@timestamp: "ELK Seen at"

      _soruce.hierarchy_code: "SQUID Hierarchy Code"

      _id: "ELK Event ID"
```

```
iocs:

  _source.server: "IP Address"

  _source.url: "URL"
```

A corresponding match would appear like the text found in **Error! Reference source not found.**:

Viewing the parsing configuration, there are three main components:

- **Hits**: This is the path in the watcher output to the hits that should be recorded as IoCs. In this case, the hits are results.input.payload.hits.hits, but this will vary for any watcher configuration. The best way to do develop this for your installation is to simulate a hit via the Watcher configuration page in Elk (Management→ Watcher)
- **iocs**: These are the individual Indicators of Compromise that caused the trigger. In the case above we say that the field _source.server should contain an IP Address and that _source.url should contain a URL. When this alert is parsed, those two fields will be used to create 2 IoCs in ThreatQ and link them to triggered event.
- **attributes:** These are attributes in the data stream that should be parsed and recorded for each IoC. The form is path: "Attribute Name" and the value will be whatever is in that path. If a path doesn't exist, it will be skipped.

Appendix C: Logstash Filters Example

This is an example of Logstash Filters. The text contained within has been redacted and should **only** server as a reference.

Logstash provides a way to parse and pre-analyze logs, then storing them in several places, including the Elasticsearch stores. Since Logstash can do pre-analysis work, it is possible to enrich data in the log stream and store that data as part of the original data. To do this, a Logstash Translation filter is used.

It is also possible to store sightings of ThreatQ data directly. To do this, after the data has been enriched via the Translation filter, a Clone filter is used to split the data, clean it, and store it in a second sightings index, to be analyzed separately.

Translation Filters

Included below is an example configuration for a Logstash Translation filter. This can be configured in a variety of ways, but the below configuration is the expected configuration for items downstream.

Figure 1: Translation Filter

```
translate {

            dictionary_path => "/opt/logstash_filters/ThreatQ Translation File -- IP Address.yml"

            field => server

            destination => threatq_data

            add_field => { "threatq_match_field" => "server"}

            add_tag => ["threatq_match"]

        }

kv {
```

```
            allow_duplicate_values => false

            field_split => "|"

            target => "threatq_match_data"

            remove_field => "threatq_data"

            source => "threatq_data"

        }
```

Here, the files generated by the `tq-elk-create-enrichment-files` command are placed in the `/opt/logstash_filters` directory. In this location, an IP Address specific file is used to check the field `server` from the upstream parsing.

If that IP Address is in the `ThreatQ Translation File - IP Address.yml` file, it will grab pertinent information, add it to the `threatq_data`

field.

The `kv` filter is then used to parse that stored information into its subsidiary parts.

These are then added to the `threatq_match_data` sub-object, which contains the following information:

- ThreatQ ID
- Link back to ThreatQ for that IoC
- Sources
- Adversaries
- Score
- Indicator Type

This information is then added into the stream so that sightings of ThreatQ data can easily be found.

Clone Filters

As the number of sightings of ThreatQ data rises, containing a footprint of these in their own index would be beneficial for analysis and roll up reporting. To do this, a `clone` filter can be used. The following configuration will show a clone filter that is copies the data, cleans the parsed event information, and keeps only the relevant ThreatQ data.

This will speed up analysis of Sightings greatly.

```
if "threatq_match" in [tags] {

            clone {

                clones => [ "threatq_sighting" ]

                add_tag => [ "threatq_match_clone" ]

            }

            if [type] == "threatq_sighting" {

                prune {

                   whitelist_names => ["^message$", "threatq_match_data", "threatq_match_field",

"@timestamp", "^timestamp$", "^type$", "^tag$"]

                }

            }

        }
```

In the above example, this takes data that has been matched to ThreatQ data in one of the Translation Filter Files, it then clones this and then strips it of nearly all of the original data (except the message field) and then sends it upstream.

To store this data, an output for this type of event ({{threatq_sighting}}) would have to be added. It should look like the following:

```
if [type] == "threatq_sighting" {

        elasticsearch {

            hosts => ["localhost:9200"]

            index => "threatq-sightings-%{+YYYY.MM.dd}"

        }
```

Once complete, the data will fill the threatq-sightings index.

Change Log

Version	Details
2.0.0	• Update for ELK 7.x • Added Attribute Filter • Added template update command • Migrated to Threat Library Search • Added SSL cert location for elasticsearch Added the search names to the index
1.0.0	Initial Release