ThreatQuotient

ELK Stack Integration Guide

Version 2.1.1

October 23, 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 ... **4**
Introduction ... **5**
Prerequisites ... **6**
System Requirements ... 6
Timezone ... 6
Required User Accounts ... 7
Dependencies ... 7
Installation ... **9**
Configuration ... **12**
Usage ... **14**
tq-elk-sync-data ... 14
tq-elk-create-enrichment-files ... 16
tq-elk-get-alerts ... 17
tq-elk-update-template ... 18
Command Line Arguments ... 19
CRON ... **21**
Squid Proxy Hit Example ... **22**
Alert Parsing Configuration Example ... **23**
Logstash Filters Examples ... **25**
Translation Filters ... 25
Clone Filters ... 26
Change Log ... **27**

Versioning

- Current integration version: 2.1.1
- Supported on ThreatQ versions >= 4.24.0

There are two versions of this integration:

- Python 2 version
- Python 3 version

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.

- 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

The integration uses the same saved data collections 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.

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.

System Requirements

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. Otherwise, the alerts API will be 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.

Timezone

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
```

Required User Accounts

The following table contains the required users accounts for 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

Dependencies

The following dependencies are required in order to install the connector on the ELK server.

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

The following logstash plugins will also need to be installed.

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

Use the commands below to install these dependencies:

```
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
```

Installation

The connector can be installed from the ThreatQuotient repository with YUM credentials or offline via a .whl file.

Upgrading Users - Review the Change Log for updates to configuration parameters before updating. If there are changes to the configuration file (new/removed parameters), you must first delete the previous version's configuration file before proceeding with the install steps listed below. Failure to delete the previous configuration file will result in the connector failing.

1. Install the connector using one of the following methods:

   ThreatQ Repository

   a. Run the following command:

   ```
   pip install tq_conn_elk
   ```

   Offline via .whl file

   To install this connector from a wheel file, the wheel file (.whl) will need to be copied via SCP into your ThreatQ instance.

   a. Download the connector whl file with its dependencies:

   ```
   mkdir /tmp/tq_conn_elk
   pip download tq_conn_elk -d
   /tmp/tq_conn_elk/
   ```

   b. Archive the folder with the .whl files:

   ```
   tar -czvf tq_conn_elk.tgz /tmp/tq_conn_elk/
   ```

   c. Transfer all the whl files, the connector and all the dependencies, to the ThreatQ instance.

   d. Open the archive on ThreatQ:

   ```
   tar -xvf tq_conn_elk.tgz
   ```

   e. Install the connector on the ThreatQ instance.

> The example assumes that all the whl files are copied to /tmp/conn on the ThreatQ instance.

```
pip install /tmp/conn/tq_conn_elk-<version>-<python version>-none-any.whl --no-index --find-links /tmp/conn/
```

> Four drivers called tq-conn-elk-sync-data, tq-conn-elk-create-enrichment-files, tq-conn-elk-get-alerts, and tq-conn-elk-update-template will be installed. After installing with pip or setup.py, a script stub will appear in /usr/bin/tq-conn-elk-sync-data.

2. Once the application has been installed, a directory structure must be created for all configuration, logs and files, using the mkdir -p command. Use the commands below to create the required directories:

```
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
```

3. Perform an initial run using the following command:

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

Enter the following parameters when prompted:

PARAMETER	DESCRIPTION
ThreatQ Host	This is the host of the ThreatQ instance, either the IP Address or Hostname as resolvable by ThreatQ.
Client ID	This is the OAuth id that can be found at Settings Gear → User Management → API details within the user's details.
Email Address	This is the User in the ThreatQ System for integrations.
Password	The password for the above ThreatQ account.

PARAMETER	DESCRIPTION
Status	This is the default status for objects that are created by this Integration. It is common to set this to "Review", but Organization SOPs should be respected when setting this.

Example Output

```
tq-elk-sync-data -c /opt/tq-integrations/elk/config/ -ll /opt/tq-integrations/elk/logs/ -v 3
ThreatQ Host: <ThreatQ Host IP or Hostname>
Client ID: <ClientID>
E-Mail Address: <EMAIL ADDRESS>
Password: <PASSWORD>
Status: Review
Connector configured. Set information in UI
```

You will still need to configure and then enable the connector.

Configuration

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

To configure the integration:

1. Navigate to your integrations management page in ThreatQ.
2. Select the **Labs** option from the *Category* dropdown (optional).

   > If you are installing the connector for the first time, it will be located under the **Disabled** tab.

3. Click on the integration to open its details page.
4. Enter the following parameter under the **Configuration** tab:

PARAMETER	DESCRIPTION
Elasticsearch Host	The IP Address or Hostname with port or your ELK server. 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	The username to use to authenticate with your ELK server. > 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.

PARAMETER	DESCRIPTION
Elasticsearch Password	The password used to authenticate with your ELK server. This can be left blank if Xpack Security is disabled.
Saved Threat Library Searches	The data collection name(s) that are created in ThreatQ to import Custom IoC's to ElasticSearch.
Attribute Filter	The list of attributes to be sent over to ElasticSearch. If left blank, all attributes will be sent over. 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 Configuration	A YAML configuration block for describing how to parse Indicators of Compromise out of hits returned by an Xpack Watcher Alert. Required only for the `tq-conn-elk-get-alerts` driver. See the Alert Parsing Configuration Example for more details.
Elasticsearch SSL CA certificate location	The location to the Elasticsearch certificate when the certificate doesn't pass SSL verification.

5. Review the **Settings** configuration, make any changes if needed, and click on **Save**.
6. Click on the toggle switch, located above the *Additional Information* section, to enable it.

Usage

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

tq-elk-sync-data

Run the following command to sycn threat intelligence data from ThreatQ to ELK:

```
<> tq-elk-sync-data -c /opt/tq-integrations/elk/config/ -ll /opt/tq-integrations/logs/ -v3
```

Example

```
<> 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 ~]#
```

tq-elk-create-enrichment-files

You can create the enrichment files using the following command:

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

Example

```
< > 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
```

tq-elk-get-alerts

You can get alerts from the ELK system by running the following command:

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

Example

```
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
```

tq-elk-update-template

You can run the following command if you have used an older version of the connector on ELK 7.x or have upgraded your ELK from version 6.0 that was previously running the connector.

```
tq-elk-update-template -c /opt/tq-integrations/elk/config/ -ll /opt/tq-integrations/elk/logs/ -v3
```

Example

```
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
```

Command Line Arguments

This connector supports the following custom command line arguments:

ARGUMENT	DESCRIPTION
-h, --help	Shows this help message and exits.
-n NAME, --name NAME	This sets the name for this connector. In some cases, it is useful to have multiple connectors of the same type executing against a single TQ instance. For example, the Syslog Exporter can be run against multiple target and multiple exports, each with their own name and configuration
-d, --no-differential	If exports are used in this connector, this will turn 'off' the differential flag for the execution. This allows debugging and testing to be done on export endpoints without having to rebuild the exports after the test. THIS SHOULD NEVER BE USED IN PRODUCTION.
-ll LOGLOCATION, --loglocation LOGLOCATION	Sets the logging location for the connector. The location should exist and be writable by the current. A special value of 'stdout' means to log to the console (this happens by default).
-c CONFIG, --config CONFIG	This is the location of the configuration file for the connector. This location must be readable and writable by the current user. If no config file path is given, the current directory will be used. This file is also where some information from each run of the connector may be put (last run time, private oauth, etc.)
-v {1,2,3}, --verbosity {1,2,3}	This is the logging verbosity level where **3** means everything. The default is **1** (Warning).
--external-proxy, -ep	This enables a proxy to be used to contac the internet for the data required by this connector. This specifies an internet facing proxy, NOT a proxy to the TQ instance.

ARGUMENT	DESCRIPTION
--enrichment-dest ENRICHMENT_DEST	Enrichment File Destination that can be referenced by logstash filters

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.

In the example below, the command will execute the connector every two hours.

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:

   Every 2 Hours Example

   ```
   0 */2 * * * tq-conn-elk-sync-data -c /path/to/config/directory/ -ll /path/to/log/
   directory/ -ll /path/to/config/directory/ -v3
   ```

4. Save and exit CRON.

Squid Proxy Hit Example

This is an example of a Squid Proxy match. The text contained within has been redacted and should only serve 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-*"
     ],
```

Alert Parsing Configuration Example

This is an example of Xpack Alerting. The text contained within has been redacted and should only serve 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 the Squid Proxy Hit Example.

Viewing the parsing configuration, there are three main components:

COMPONENT	DESCRIPTION
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).
icos	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

COMPONENT	DESCRIPTION
	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.

Logstash Filters Examples

This is an example of Logstash Filters. The text contained within has been redacted and should only serve 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.

```
 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 2.1.1**
  - Removed cryptography package from setup.py
  - Added version 4.0.2 to the configparser requirement in setup.py
- **Version 2.1.0**
  - Added functionality to allow the program to be run in both Python 2 and Python 3
- **Version 2.0.0**
  - Updated 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
- **Version 1.0.0**
  - Initial Release