

Hortonworks Cybersecurity Package

Run Book

(July 11, 2017)

Hortonworks Cybersecurity Package: Run Book

Copyright © 2012-2017 Hortonworks, Inc. Some rights reserved.

Hortonworks Cybersecurity Package (HCP) is a modern data application based on Apache Metron, powered by Apache Hadoop, Apache Storm, and related technologies.

HCP provides a framework and tools to enable greater efficiency in Security Operation Centers (SOCs) along with better and faster threat detection in real-time at massive scale. It provides ingestion, parsing and normalization of fully enriched, contextualized data, threat intelligence feeds, triage and machine learning based detection. It also provides end user near real-time dashboards.

Based on a strong foundation in the Hortonworks Data Platform (HDP) and Hortonworks DataFlow (HDF) stacks, HCP provides an integrated advanced platform for security analytics.

Please visit the [Hortonworks Data Platform](#) page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the [Support](#) or [Training](#) page. Feel free to [Contact Us](#) directly to discuss your specific needs.



Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 4.0 License.
<http://creativecommons.org/licenses/by-sa/4.0/legalcode>

Table of Contents

1. Overview	1
2. Adding a New Telemetry Data Source	2
2.1. Prerequisites	2
2.2. Stream Data into HCP	3
2.2.1. OPTIONAL: Install the Squid Package	4
2.2.2. Install NiFi	4
2.2.3. Create a NiFi Flow to Stream Events to HCP	4
2.3. Parse the Squid Data Source to HCP	10
2.3.1. Parse the Squid Telemetry Event	10
2.3.2. Verify That the Events Are Indexed	17
2.3.3. Create an Index Template	18
2.4. Add New Data Source to the Metron Dashboard	19
2.4.1. Configuring a New Data Source Index in the Metron Dashboard	19
2.4.2. Reviewing the New Data Source Data	21
3. Transform the Squid Message	22
4. Enriching Telemetry Events	25
4.1. Bulk Loading Enrichment Information	26
4.1.1. OPTIONAL: Create a Mock Enrichment Source	27
4.1.2. Configuring an Extractor Configuration File	27
4.1.3. Configuring Element-to-Enrichment Mapping	29
4.1.4. Running the Enrichment Loader	30
4.2. Mapping Fields to HBase Enrichments	31
4.3. OPTIONAL: Global Configuration	34
4.4. Verify That the Events Are Enriched	34
5. Enriching Threat Intelligence Information	36
5.1. OPTIONAL: Create a Mock Threat Intel Feed Source	36
5.2. Configuring an Extractor Configuration File	36
5.3. Configuring Element-to-Threat Intel Feed Mapping	38
5.4. Run the Threat Intelligence Loader	39
5.5. Mapping Fields to HBase Enrichments	40
5.6. Verify That the Threat Intel Events Are Enriched	43
6. Prioritizing Threat Intelligence	44
6.1. Prerequisites	44
6.2. Performing Threat Triage	44
6.3. Viewing Triaged or Scored Alerts	47
6.3.1. Threat Triage Alert Panel	47
6.3.2. HCP Metron Dashboard View of Alerts	47
7. Configuring Indexing	49
7.1. Default Configuration	49
7.2. Specifying Index Parameters	50
7.3. Turning off HDFS Writer	51
8. Setting Up a Profile	53
8.1. Installing Profiler	53
8.2. Creating a Profile	54
8.3. Configuring the Profiler	57
8.4. Starting the Profiler	58
8.5. Developing Profiles	58
8.6. Testing	59

List of Figures

2.1. Add Processor Dialog Box	5
2.2. New TailFile Processor	5
2.3. Configure Processor Dialog Box Settings Tab	6
2.4. Configure Processor Dialog Box Properties Tab	6
2.5. Configure Processor	7
2.6. Configure Properties	7
2.7. Create Connection Dialog Box	8
2.8. Data Flow	8
2.9. NiFi Operate Panel	9
2.10. New Sensor Panel	11
2.11. Grok Validator Panel with Sample Text	12
2.12. Grok Validator Panel with Statement Information	13
2.13. Grok Validator Panel with First Element and Test Results	14
2.14. Grok Validator Panel with Second Element and Test Results	15
2.15. mm_grok_validator_squid_complete.png	16
2.16. Elasticsearch	18
2.17. Ambari Task List	20
2.18. Configure an Index Pattern	20
2.19. Discover Tab with Squid Elements	21
3.1. Squid Schema Panel	22
3.2. New Schema Information Panel	23
3.3. Populated Schema Information Panel	24
4.1. New Schema Information Panel	33
4.2. Populated New Schema Information Panel	34
4.3. Elasticsearch	35
5.1. New Schema Information Panel	41
5.2. Populated New Schema Information Panel	42
5.3. Elasticsearch *** Needs updating ***	43
6.1. Threat Triage Rules Panel	45
6.2. Edit Rule Panel	46
6.3. Investigation Module Triaged Alert Panel	48
7.1. Management Module Advanced Panel	51

1. Overview

This guide is intended for Platform Engineers and others who are responsible for adding new telemetry data sources, enriching telemetry events, triaging threat intelligence information, and ensuring telemetry events are viewable in the user interface.

This guide walks you through how to add a specific new data telemetry: Squid proxy logs. Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. For more information on Squid, see squid-cache.org.

Unlike other HCP documentation, this guide provides detailed examples that are populated with information specific to the Squid data source.

Task	Description
Adding a New Telemetry Data Source [2]	This section describes how to add a telemetry data source to HCP.
Transform the Squid Message [22]	You can customize your sensor data to provide more meaningful data.
Enriching Telemetry Events [25]	After the raw security telemetry events have been parsed and normalized, the next step is to enrich the data elements of the normalized event.
Enriching Threat Intelligence Information [36]	You can enrich your threat intelligence information just like you enriched your telemetry information.
Prioritizing Threat Intelligence [44]	<p>Not all threat intelligence indicators are equal. Some require immediate response, while others can be dealt with or investigated as time and availability permits. As a result you need to triage and rank threats by severity.</p> <p>In HCP, you assign severity by associating possibly complex conditions with numeric scores.</p>
Configuring Indexing [49]	The indexing topology is a topology dedicated to taking the data from a topology that has been enriched and storing the data in one or more supported indices. More specifically, the enriched data is ingested into Kafka, written in an indexing batch or bolt with a specified size, and sent to one or more specified indices. The configuration is intended to configure the indexing used for a given sensor type (for example, snort).
Setting Up a Profile [53]	A profile describes the behavior of an entity on a network. An entity can be a server, user, subnet, or application. Once you generate a profile defining what normal behavior looks like, you can build models that identify anomalous behavior.

This guide assumes that you have met all of the HCP 1.2.2 prerequisites and successfully installed HCP 1.2.2.

2. Adding a New Telemetry Data Source

This section describes how to add a telemetry data source to HCP.

For our examples, we will use the Squid data source as our new telemetry data source.

To add the Squid telemetry data source, perform the following tasks:

Task	Description
Meet Prerequisites	Before you can add a new telemetry data source you must meet the user requirements listed in this section.
Stream Data Into HCP	The first step in adding a new data source telemetry is to stream all raw events from the telemetry data source into its own Kafka topic.
Parse the Squid Data Source to HCP	Parsers transform raw data (textual or raw bytes) into JSON messages suitable for downstream enrichment and indexing by HCP. You must create a parser for each new data source.
Add New Data Source to Dashboard	After a new data telemetry source has been added to HCP, you must add it to the Metron dashboard before you can create queries and filters for it and add telemetry panels displaying its data.

2.1. Prerequisites

Before you add a new telemetry device, you must perform the following actions:

- Install HDP and HDF, and then install HCP.

For information about installing HCP, see the [HCP Installation Guide](#).

- The following use case assumes the following user requirements:
 - Proxy events from Squid logs must be ingested in real-time.
 - Proxy logs must be parsed into a standardized JSON structure suitable for analysis by Metron.
 - In real-time, the Squid proxy events must be enriched so that the domain names contain the IP information.
 - In real-time, the IP within the proxy event must be checked against for threat intelligence feeds.
 - If there is a threat intelligence hit, an alert must be raised.
 - The SOC analyst must be able to view new telemetry events and alerts from Squid.
 - HCP must profile the Squid data and incorporate statistical features of the profile into the threat triage rules.
 - HCP must be able to deploy a machine learning model that derives additional insights from the stream.

- HCP must incorporate user's machine learning model into user's threat triage rule along with threat intel, static rules, and statistical rules.
- Set HCP values.

When you install HCP, you will set up several hosts. You will need the locations of these hosts, along with port numbers, and the Metron version. These values are listed below.

- KAFKA_HOST = The host where a Kafka broker is installed.
- ZOOKEEPER_HOST = The host where a ZooKeeper server is installed.
- PROBE_HOST = The host where your sensor, probes are installed. If don't have any sensors installed, pick the host where a Storm supervisor is running.
- SQUID_HOST = The host where you want to install SQUID. If you don't care, install SQUID on the PROBE_HOST.
- NIFI_HOST = Host where you will install NIFI. This should be the same host on which you installed Squid.
- HOST_WITH_ENRICHMENT_TAG = The host in your inventory hosts file that you put under the group "enrichment."
- SEARCH_HOST = The host where you have Elastic or Solr running. This is the host in your inventory hosts file that you put under the group "search". Pick one of the search hosts.
- SEARCH_HOST_PORT = The port of the search host where indexing is configured (for example, 9300).
- METRON_UI_HOST = The host where your Metron UI web application is running. This is the host in your inventory hosts file that you put under the group "web."
- METRON_VERSION = The release of the Metron binaries you are working with (for example, 0.2.0BETA-RC2).

2.2. Stream Data into HCP

The first step in adding a new data source telemetry is to stream all raw events from the telemetry data source into its own Kafka topic.

Complete the tasks provided in the following sections:

Task	Description
OPTIONAL: Install the Squid Package [4]	In this section we install the package for our Squid telemetry data source so we can generate data. If you have already installed your data source and have a log source, you can skip this section and move to the next one.
Install NiFi	If you haven't already done so, install NiFi.
Create a NiFi Flow to Stream Events to HCP	This section provides instructions to create a data flow to capture events from Squid and push them into HCP.

2.2.1. OPTIONAL: Install the Squid Package

In this section we install the package for our Squid telemetry data source so we can generate data. If you have already installed your data source and have a log source, you can skip this section and move to the next one.

1. ssh into \$SQUID_HOST,
2. Install and start Squid:

```
sudo yum install squid  
sudo service squid start
```

2.2.2. Install NiFi

If you haven't already done so, install NiFi. For information on installing NiFi, see [NiFi Command Line Installation](#).

NiFi is built to automate the flow of data between systems. As a result, NiFi works well to collect, ingest, and push data to HCP.



Important

NiFi cannot be installed on top of HDP, so you must install NiFi manually to use it with HCP.



Note

Ensure that the NiFi web application is using port 8089.

The port number is stored in `/usr/lib/nifi<version number>/conf/nifi.properties`.

2.2.3. Create a NiFi Flow to Stream Events to HCP

This section provides instructions to create a data flow to capture events from Squid and push them into HCP. For this task we will use NiFi to create two processors, one TailFile processor that will ingest Squid events and one PutKafka processor that will stream the information to Metron. When we create and define the PutKafka processor, Kafka will create a topic for the Squid data source. We'll then connect the two processors, generate some data in the Squid log, and watch the data flow from one processor to the other.


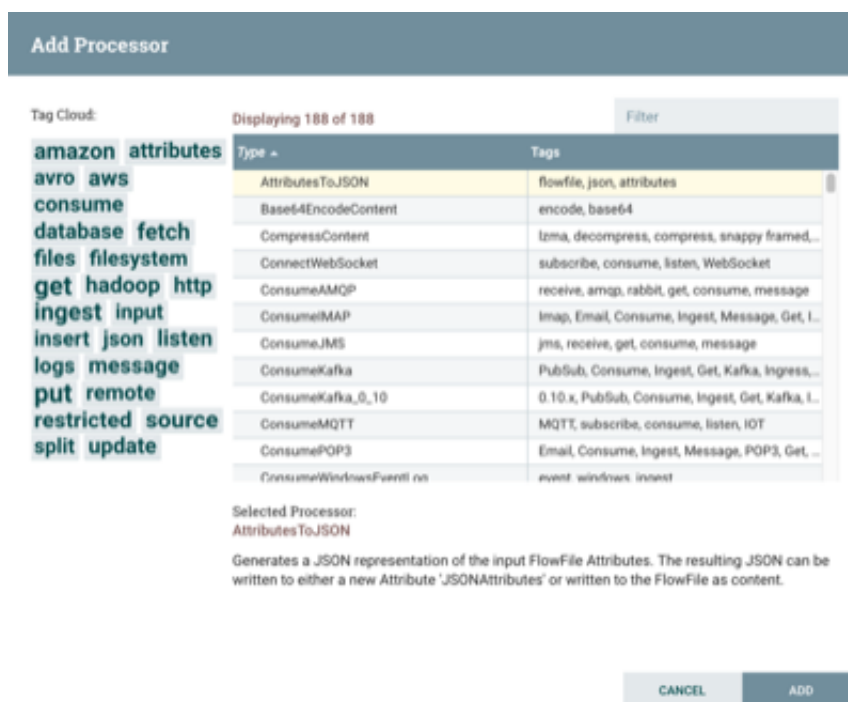
1. Drag the first icon on the toolbar  (the processor icon) to your workspace.
NiFi displays the Add Processor dialog box.

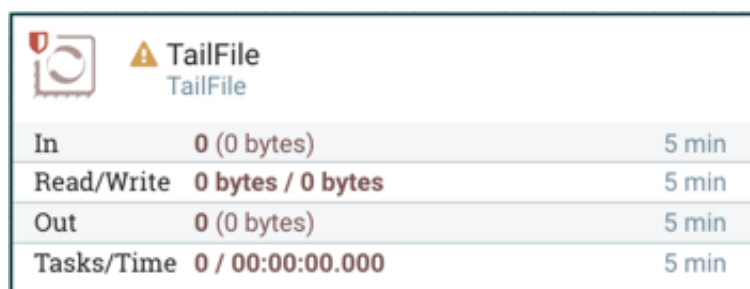
Figure 2.1. Add Processor Dialog Box



2. Select the TailFile type of processor and click **Add**.

NiFi displays a new TailFile processor.

Figure 2.2. New TailFile Processor



3. Right-click the processor icon and select **Configure** to display the **Configure Processor** dialog box.
 - In the **Settings** tab, change the name to **Ingest Squid Events**.

Figure 2.3. Configure Processor Dialog Box Settings Tab

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Name: Ingest Squid Events ☒ Enabled ☐ success

Id: 13a1a081-015c-1000-7972-7dc6816628b0

Type: TailFile

Penalty Duration: 30 sec Yield Duration: 1 sec

Bulletin Level: WARN

Automatically Terminate Relationships: success
All FlowFiles are routed to this Relationship.

CANCEL APPLY

- In the **Properties** tab, enter the path to the squid `access.log` file in the **Value** column for the **File(s) to Tail** property.

Figure 2.4. Configure Processor Dialog Box Properties Tab

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field +

Property	Value
Tailing mode	Single file
File(s) to Tail	/usr/log/squid/access.log
Rolling Filename Pattern	No value set
Base directory	No value set
Initial Start Position	Beginning of File
State Location	Local
Recursive lookup	false
Rolling Strategy	Fixed name
Lookup frequency	10 minutes
Maximum age	24 hours

CANCEL APPLY

4. Click **Apply** to save your changes and dismiss the **Configure Processor** dialog box.
5. Add another processor by dragging the **Processor** icon to the main window.
6. Select the **PutKafka** type of processor and click **Add**.
7. Right-click the processor and select **Configure**.

8. In the Settings tab, change the name to **Stream to Metron** and then select the Automatically Terminate Relationships check boxes for **Failure** and **Success**.

Figure 2.5. Configure Processor

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Name: Stream to Metron ☒ Enabled

Id: 13ada490-015c-1000-1805-77a61f75a5ab

Type: PutKafka

Penalty Duration: 30 sec Yield Duration: 1 sec

Bulletin Level: WARN

Automatically Terminate Relationships

☒ failure
Any FlowFile that cannot be sent to Kafka will be routed to this Relationship

☒ success
Any FlowFile that is successfully sent to Kafka will be routed to this Relationship

CANCEL APPLY

9. In the Properties tab, set the following three properties:

- Known Brokers: \$KAFKA_HOST:6667
- Topic Name: squid
- Client Name: nifi-squid

Figure 2.6. Configure Properties

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value
Known Brokers	\$KAFKA_HOST:6667
Topic Name	squid
Partition Strategy	Round Robin
Partition	No value set
Kafka Key	No value set
Delivery Guarantee	Best Effort
Message Delimiter	No value set
Max Buffer Size	5 MB
Max Record Size	1 MB
Communications Timeout	30 secs
Batch Size	16384
Queue Buffering Max Time	No value set
Compression Codec	None
Client Name	nifi-squid

CANCEL APPLY

10. Click **Apply** to save your changes and dismiss the **Configure Processor** dialog box.
11. Create a connection by dragging the arrow from the Ingest Squid Events processor to the Stream to Metron processor.

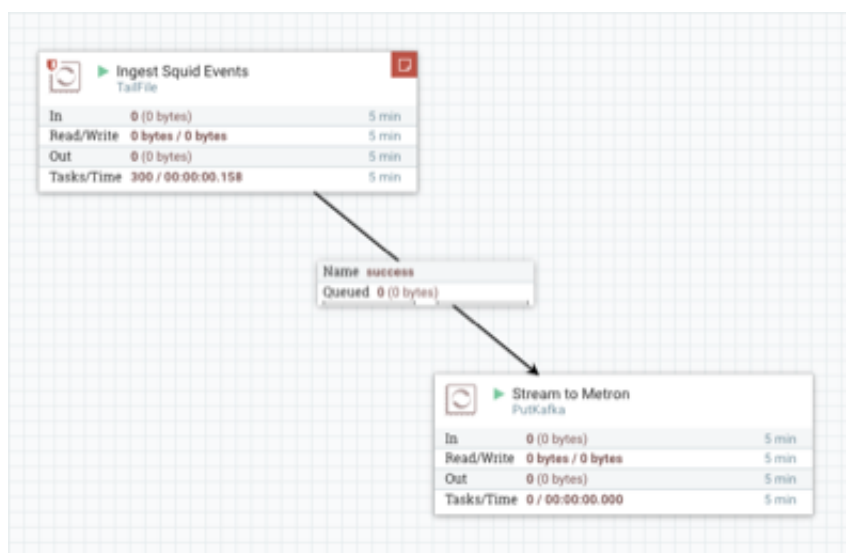
NiFi displays a **Create Connection** dialog box.

Figure 2.7. Create Connection Dialog Box



12. Click **Add** to accept the default settings for the connection.
13. Press the Shift key and draw a box around both parsers to select the entire flow.

Figure 2.8. Data Flow




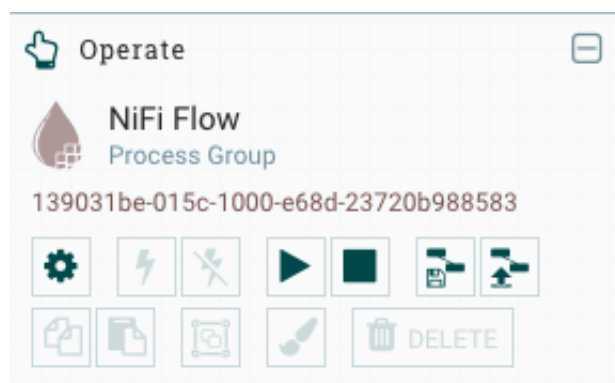
14. Click  (Start button) in the **Operate** panel.

Figure 2.9. NiFi Operate Panel



15. Generate some data using the new data processor client.

- a. Use ssh to access the host for the new data source.
- b. With Squid started, look at the different log files that get created:

```
sudo su -  
cd /var/log/squid  
ls
```

The file you want for Squid is the `access.log`, but another data source might use a different name.

- c. Generate entries for the log so you can see the format of the entries.

```
squidclient -h 127.0.0.1 "http://www.atmape.ru"
```

You will see the following data in the `access.log` file.

```
1481143984.330 1111 127.0.0.1 TCP_MISS/301 714 GET http://www.atmape.  
ru/ - DIRECT/212.109.217.71 text/html
```

- d. Using the Squid log entries, you can determine the format of the log entries is:

```
timestamp | time elapsed | remotehost | code/status | bytes | method |  
URL rfc931 peerstatus/peerhost | type
```

You should see metrics on the processor of data being pushed into Metron.

16. Look at the Storm UI for the parser topology and you should see tuples coming in.

- Go to Ambari UI.
- From the **Quick Links** pull-down menu in the upper center of the window, select **Storm UI**.

17. Before leaving this section, run the following commands to fill the `access.log` with data. You'll need this data when you enrich the telemetry data.

```
squidclient -h 127.0.0.1 "http://www.aliexpress.com/af/shoes.html?ltype=wholesale&d=y&origin=n&isViewCP=y&catId=0&initiative_id=SB_20160622082445&SearchText=shoes"
squidclient -h 127.0.0.1 "http://www.help.landl.co.uk/domains-c40986/transfer-domains-c79878"
squidclient -h 127.0.0.1 "http://www.pravda.ru/science/"
squidclient -h 127.0.0.1 "http://www.brightsideofthesun.com/2016/6/25/12027078/anatomy-of-a-deal-phoenix-suns-pick-bender-chriss"
squidclient -h 127.0.0.1 "https://www.microsoftstore.com/store/msusa/en_US/pdp/Microsoft-Band-2-Charging-Stand/productID.329506400"
squidclient -h 127.0.0.1 "https://tfl.gov.uk/plan-a-journey/"
squidclient -h 127.0.0.1 "https://www.facebook.com/Africa-Bike-Week-1550200608567001/"
squidclient -h 127.0.0.1 "http://www.ebay.com/itm/02-Infiniti-QX4-Rear-spoiler-Air-deflector-Nissan-Pathfinder-/172240020293?fits=Make%3AInfiniti%7CModel%3AQX4&hash=item281a4e2345:g:iMkAAOSwoBtW4Iwx&vxp=mtr"
squidclient -h 127.0.0.1 "http://www.recruit.jp/corporate/english/company/index.html"
squidclient -h 127.0.0.1 "http://www.lada.ru/en/cars/4x4/3dv/about.html"
squidclient -h 127.0.0.1 "http://www.help.landl.co.uk/domains-c40986/transfer-domains-c79878"
squidclient -h 127.0.0.1 "http://www.aliexpress.com/af/shoes.html?ltype=wholesale&d=y&origin=n&isViewCP=y&catId=0&initiative_id=SB_20160622082445&SearchText=shoes"
```

For more information about creating a NiFi data flow, see the [NiFi documentation](#).

2.3. Parse the Squid Data Source to HCP

Parsers transform raw data (textual or raw bytes) into JSON messages suitable for downstream enrichment and indexing by HCP. There is one parser for each data source and the information is piped to the Enrichment/Threat Intelligence topology.

The following sections guide you through how to add the Squid telemetry to HCP. The following procedures use the Management module UI whenever possible. If you would like to see these steps performed using the CLI, see the [HCP Administration Guide](#).

Task	Description
Parse the Squid Telemetry Event [10]	This section uses the Management module to create a parser for the new data source.
Verify that the Events are Indexed	After you finish parsing your new data source, you should verify that the data source events are indexed and the output matches any Stellar transformation functions you used.
Create an Index Template	To work with a new data source data in the Metron dashboard, you need to ensure that the data is landing in the search index (Elasticsearch) with the correct data types. You can achieve this by defining an index template.

2.3.1. Parse the Squid Telemetry Event


1. Launch the HCP Management module:
 - a. From the Ambari Dashboard panel, click **Metron**.
 - b. Make sure you have the **Summary** tab selected.

- c. Select the Metron Management UI from the Summary list.

The **Management UI** tool should display in a separate browser tab.

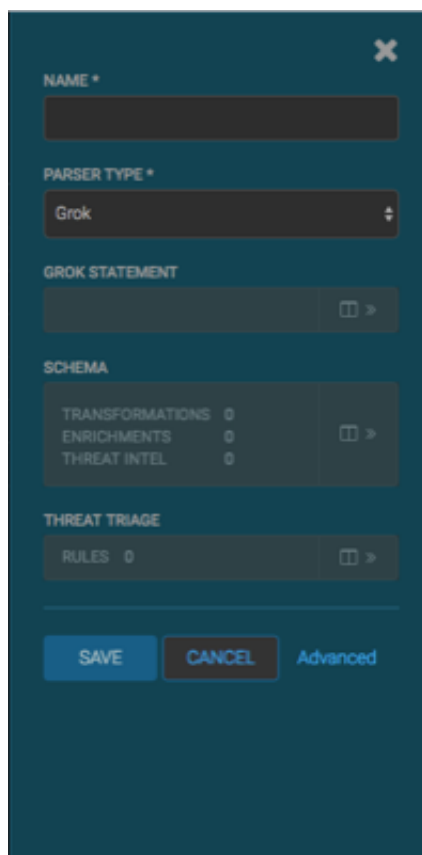
Alternatively, you can launch the module from
\$METRON_MANAGEMENT_UI_HOST:4200 in a browser.

2. Click **Sensors** on the left side of the window, under **Operations**.

3. Click  (the add button) in the lower right corner of the screen.

The Management module displays a panel used to create the new sensor.

Figure 2.10. New Sensor Panel



4. In the **NAME** field, enter the name of the sensor.

For our example, we use the name **squid**.

If a Kafka topic already exists for the sensor name, the module displays a message similar to **Kafka Topic Exists. Emitting** and displays the name of the **Parser Type**. You can skip to the next section, [Verify Events are Indexed](#).

If no matching Kafka topic is found, the module displays **No Matching Kafka Topic**.

5. In the **Parser Type** field, choose the type of parser for the new sensor.

If you chose a Grok parser type, the module prompts for a Grok statement.

6. If no Kafka topic exists for squid, create a Grok statement for the new parser:

a. In the Grok Statement box, click the  (expand window button) to display the Grok Validator panel.

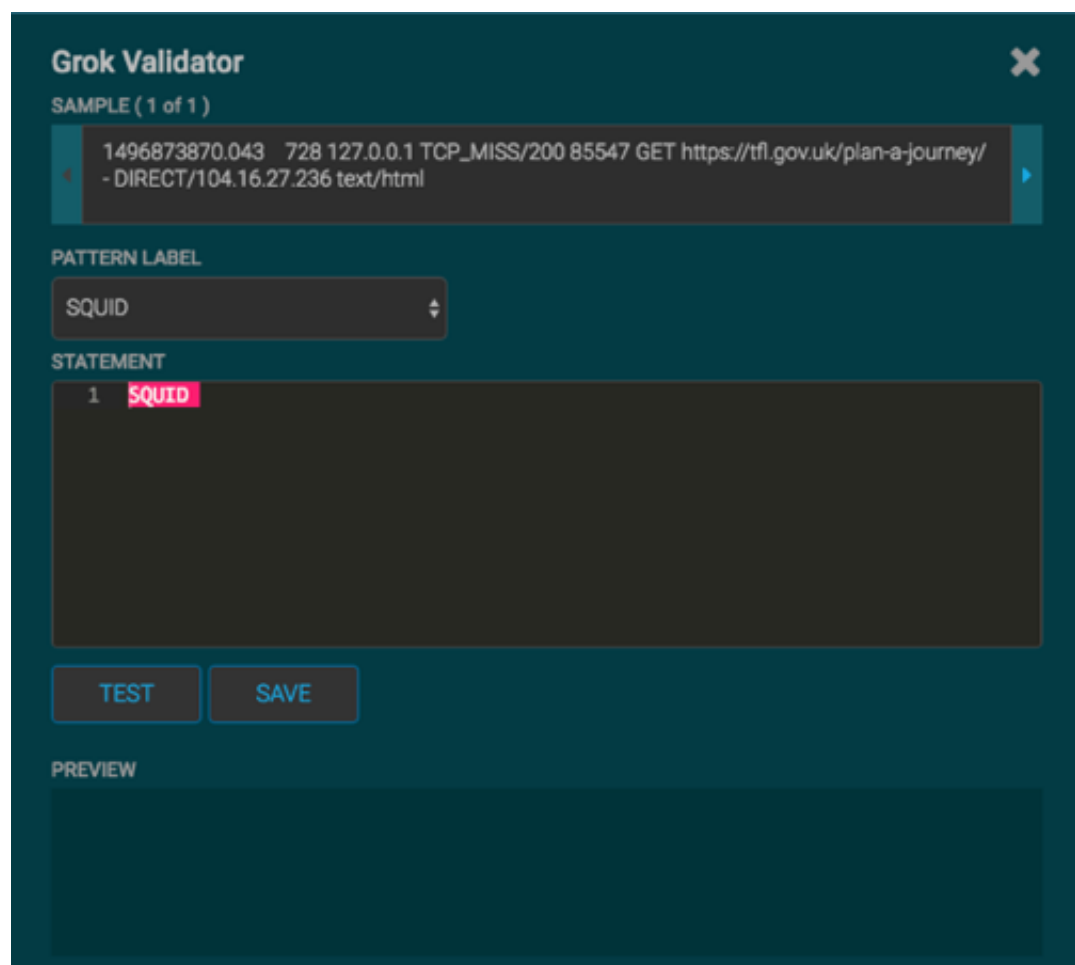
b. Choose or enter a name for the Grok statement in the **PATTERN LABEL** field.

For our example, we chose SQUID.

c. If there is no data in the **SAMPLE** text field, enter a sample log entry for the data source.

To create sample log entries, see [Step 15](#).

Figure 2.11. Grok Validator Panel with Sample Text



d. Refer to the format of the log entries you determined in [Step 11d](#).

For example, the log entry format for Squid is:

```
timestamp | time elapsed | remotehost | code/status | bytes | method |
URL rfc931 peerstatus/peerhost | type
```

- e. In the **STATEMENT** field, enter the first element in the sensor log file.

For Squid, the first element in the sensor log file is `timestamp` which is a number, so we enter `%{NUMBER:timestamp}`.



Note

The Management module automatically completes partial words in your Grok statement as you enter them.

Figure 2.12. Grok Validator Panel with Statement Information

Grok Validator

SAMPLE (1 of 1)

```
1496873870.043 728 127.0.0.1 TCP_MISS/200 85547 GET https://tfl.gov.uk/plan-a-journey/-DIRECT/104.16.27.236 text/html
```

PATTERN LABEL

SQUID

STATEMENT

```
1 SQUID %{NUMBER:timestamp}
```

TEST SAVE

- f. Click **TEST**.

If the validator finds an error, it displays the error information. If the validation succeeds, it displays the valid mapping in the **PREVIEW** field.

Because you entered the timestamp element, the validator parses the timestamp correctly and leaves the rest of the information as random data.

Figure 2.13. Grok Validator Panel with First Element and Test Results

The screenshot shows the Grok Validator interface. At the top, it says "Grok Validator" with a close button. Below that, it says "SAMPLE (1 of 1)". The sample log entry is: "1496873870.043 728 127.0.0.1 TCP_MISS/200 85547 GET https://tfl.gov.uk/plan-a-journey/- DIRECT/104.16.27.236 text/html". Below the sample, there is a "PATTERN LABEL" dropdown menu set to "SQUID_DELIMITED". Below that, there is a "STATEMENT" text area containing "1 SQUID_DELIMITED %{NUMBER:timestamp}". At the bottom, there are "TEST" and "SAVE" buttons. Below the buttons, there is a "PREVIEW" section showing the original string and the extracted timestamp.

Grok Validator

SAMPLE (1 of 1)

1496873870.043 728 127.0.0.1 TCP_MISS/200 85547 GET https://tfl.gov.uk/plan-a-journey/- DIRECT/104.16.27.236 text/html

PATTERN LABEL

SQUID_DELIMITED

STATEMENT

1 SQUID_DELIMITED %{NUMBER:timestamp}

TEST SAVE

PREVIEW

original_string	1496873870.043 728 127.0.0.1 TCP_MISS/200 85547 GET https://tfl.gov.uk/plan-a-journey/- DIRECT/104.16.27.236 text/html
timestamp	1496873870.043

g. Refer to the log entry format and enter the second element.

For Squid, the second element is `elapsed`, so we enter `%{INT:elapsed}` and click **TEST** again.

Figure 2.14. Grok Validator Panel with Second Element and Test Results

The screenshot shows the Grok Validator interface. At the top, there's a title bar 'Grok Validator' with a close button. Below it, a 'SAMPLE (1 of 1)' section displays a log entry: '1496873870.043 728 127.0.0.1 TCP_MISS/200 85547 GET https://tfl.gov.uk/plan-a-journey/-DIRECT/104.16.27.236 text/html'. Below the sample, there's a 'PATTERN LABEL' dropdown menu set to 'SQUID_DELIMITED'. Underneath is a 'STATEMENT' text area containing the Grok pattern: '1 SQUID_DELIMITED %{NUMBER:timestamp} %{INT:elapsed}'. At the bottom, there are 'TEST' and 'SAVE' buttons. Below these buttons is a 'PREVIEW' section showing the results of the test: 'elapsed' is 85547, 'original_string' is the full log entry, and 'timestamp' is 200.

PREVIEW	
elapsed	85547
original_string	1496873870.043 728 127.0.0.1 TCP_MISS/200 85547 GET https://tfl.gov.uk/plan-a-journey/-DIRECT/104.16.27.236 text/html
timestamp	200

- h. Continue to build and test the Grok statement until you have entries for each element in the log entry.

Figure 2.15. mm_grok_validator_squid_complete.png

The screenshot displays a web-based interface for configuring a Grok pattern. At the top, there is a 'Paste Sample Message' section with a message: 'A data sample cannot automatically be loaded. Connect to a Kafka Topic or paste a message here.' Below this, the 'PATTERN LABEL' is set to 'SQUID_DELIMITED'. The 'STATEMENT' section contains a multi-line Grok pattern:

```
1 SQUID_DELIMITED %{NUMBER:timestamp}[%0-9]*%{INT:elapsed} %{IP
:ip_src_addr} %{WORD:action} %{NUMBER:code} %{NUMBER:bytes} %{WORD
:method} %{NOTSPACE:url}[%0-9]*(%{IP:ip_dst_addr})?
```

 The pattern is syntax-highlighted. Below the statement, there are 'TEST' and 'SAVE' buttons. At the bottom, there is a 'PREVIEW' section which is currently empty.



Note

You should perform the Grok validation using several different sensor log entries to ensure that the Grok statement is valid for all sensor logs. To display additional sensor log entries, click the forward or backward arrow icon on the side of the **SAMPLE** text box.

- i. When your Grok statement is complete and valid, click **SAVE** to save the Grok statement for the sensor.

7. Click **SAVE** to save the sensor information and add it to the list of Sensors.

This new data source processor topology ingests from the \$Kafka topic and then parses the event with the HCP Grok framework using the Grok pattern. The result is a standard JSON Metron structure that then is added to the "enrichment" Kafka topic for further processing.

8. Test that a Kafka topic has been created for the Squid parser:

- a. Navigate to the following directory:

```
/usr/hdp/current/kafka-broker/bin
```

- b. List all of the Kafka topics:

```
./kafka-topics.sh --zookeeper localhost:2181 --list
```

You should see the following list of Kafka topics:

- bro
- enrichments
- ubdexubg
- snort
- squid

2.3.2. Verify That the Events Are Indexed

After you finish adding your new data source, you should verify that the data source events are indexed and the output matches any Stellar transformation functions you used.

By convention, the index where the new messages are indexed is called `squid_index_[timestamp]` and the document type is `squid_doc`.

Use the Elasticsearch Head plug-in to verify that the messages were indexed correctly:

1. Log in to \$SEARCH_HOST host:

```
ssh into Host $SEARCH_HOST
```

2. Install the head plug-in:

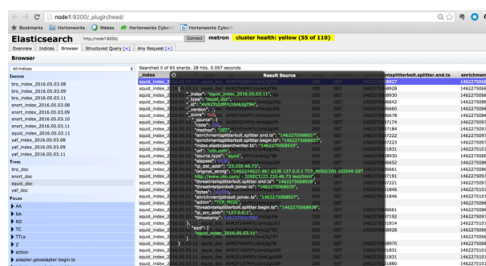
```
/usr/share/elasticsearch/bin/plugin install mobz/elasticsearch-head/1.x
```

3. Navigate to ElasticSearch Head UI: `http://$SEARCH_HOST:9200/_plugin/head/`.

4. Click the Browser tab and select the squid document in the left panel; then select one of the sample docs.

You should see something like the following:

Figure 2.16. Elasticsearch



5. Review the output to ensure it reflects the Stellar transformation functions you used.

2.3.3. Create an Index Template

To work with a new data source data in the Metron dashboard, you need to ensure that the data is landing in the search index (Elasticsearch) with the correct data types. You can achieve this by defining an index template.



Note

You will need to update the Index template after you add or change enrichments for a data source.

1. Run the following command to create an index template for the new data source.

The following is an example of an index template for a new sensor called 'sensor1'.

```
curl -XPOST $SEARCH_HOST:$SEARCH_PORT/_template/$DATASOURCE_index -d '{
  "template": "sensor1_index*",
  "mappings": {
    "sensor1_doc": {
      "_timestamp": {
        "enabled": true
      },
      "properties": {
        "timestamp": {
          "type": "date",
          "format": "epoch_millis"
        },
        "ip_src_addr": {
          "type": "ip"
        },
        "ip_src_port": {
          "type": "integer"
        },
        "ip_dst_addr": {
          "type": "ip"
        },
        "ip_dst_port": {
          "type": "integer"
        }
      }
    }
  }
}
```

```
}'
```

The example assumes the following:

- The template applies to any indices that are named `sensor1_index*`.
- The index has one document type that must be named `sensor1_doc`.
- The index contains timestamps.
- The properties section defines the types of each field. This example defines the five common fields that most sensors contain.
- Additional fields can be added following the five that are already defined.

By default, Elasticsearch will attempt to analyze all fields of type string. This means that Elasticsearch will tokenize the string and perform additional processing to enable free-form text search. In many cases, you want to treat each of the string fields as enumerations. This is why most fields in the index template are ``not_analyzed``.

2. An index template will only apply for indices that are created after the template is created. Delete the existing indices for the new data source so that new ones can be generated with the index template.

```
curl -XDELETE $SEARCH_HOST:9200/$DATASOURCE*
```

3. Wait for the new data source index to be re-created.

This might take a minute or two based on how fast the new data source data is being consumed in your environment.

```
curl -XGET $SEARCH_HOST:9200/$DATASOURCE*
```

2.4. Add New Data Source to the Metron Dashboard

After a new data telemetry source has been added to HCP, you will need to add it to the Metron dashboard before you can create queries and filters for it and add telemetry panels displaying its data. Complete the steps in the following sections to add a new telemetry source to the Metron dashboard:

Task	Description
Configuring a New Data Source Index in the Metron Dashboard [19]	Now that you have an index for the new data source with all of the right data types, you need to tell the Metron dashboard about this index.
Reviewing the New Data Source Data [21]	Now that the Metron dashboard is aware of the new data source index, you can look at the data.

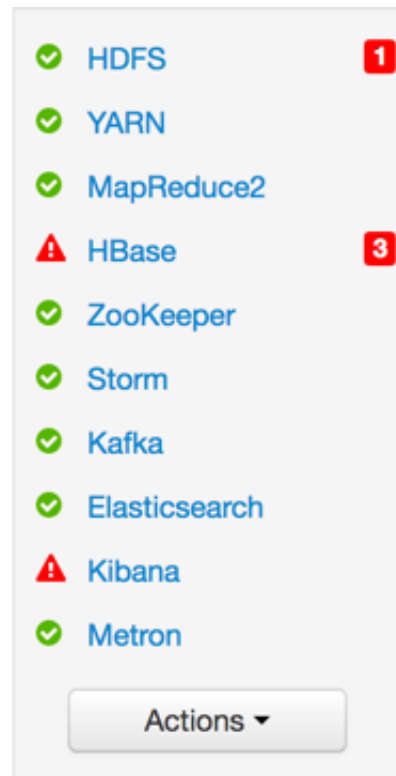
2.4.1. Configuring a New Data Source Index in the Metron Dashboard

Now that you have an index for the new data source with all of the right data types, you need to tell the Metron dashboard about this index.

1. Launch the Metron dashboard if you have not already done so:

1. From Ambari, click Kibana in the list of quick tasks.

Figure 2.17. Ambari Task List



2. Select **Metron UI** from the Quick Links menu in the top center of the window.

2. Click the **Settings** tab on the Metron dashboard.

3. Make sure you have the **Indices** tab selected, then click **+Add New**.

Kibana displays the **Configure an index pattern** window. Use the index pattern window to identify your telemetry source.

Figure 2.18. Configure an Index Pattern

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

☒ Index contains time-based events
☐ Use event times to create index names [DEPRECATED]

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

logstash-*

☐ Do not expand index pattern when searching (Not recommended)

By default, searches against any time-based index pattern that contains a wildcard will automatically be expanded to query only the indices that contain data within the currently selected time range.

Searching against the index pattern logstash-* will actually query elasticsearch for the specific matching indices (e.g. logstash-2015.12.21) that fall within the current time range.

Unable to fetch mapping. Do you have indices matching the pattern?

- In most cases the name of the index pattern will match the sensor name. For example, the 'bro' sensor has an index pattern of 'bro-*'.

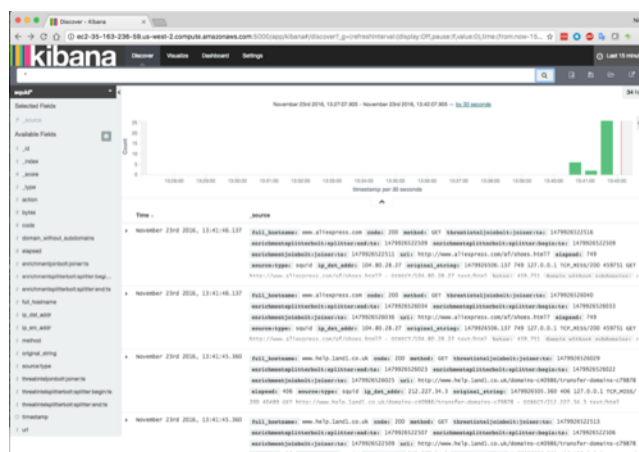
- If your data telemetry source does contain time-based events, leave the check box as is. Most of your data telemetry sources will contain time-based events.

- If you would like this new index pattern to be the default, click the Green Star icon (★).

Now that the Metron dashboard is aware of the new data source index, you can look at the data.

1. Display the Metron dashboard.
2. Click on the **Discover** tab and then choose the newly created data source index pattern.
3. Click any of the fields in the left column to see a representation of the variety of data for that specific field.
4. Click the Right Facing Arrow icon next to a specific record in the center of the window (the **Document** table) to expand the record and display the available data.

Figure 2.19. Discover Tab with Squid Elements



3. Transform the Squid Message

You can customize your sensor data to provide more meaningful data. For example, you can choose to transform a url to provide the domain name of the outbound connection or the IP address. To do this, you need to add transformation information:



Note

Your sensor must be running and producing data before you can add transformation information.

1.



In the Management module, click (edit button) for your sensor.

The Management module displays the schema panel.

Figure 3.1. Squid Schema Panel

The screenshot shows the 'Squid' schema panel with a dark teal background. At the top, there's a title 'Squid' and a close button (X). Below the title, there's a 'NAME *' field with the value 'Squid'. Underneath, it says 'No Matching Kafka Topic' in orange. Then, there's a 'PARSER TYPE *' dropdown menu with 'Grok' selected. Below that is a 'GROK STATEMENT' field with a small icon and a right arrow. The 'SCHEMA' section contains a table with three rows: 'TRANSFORMATIONS 0', 'ENRICHMENTS 0', and 'THREAT INTEL 0'. To the right of the table is a small icon and a right arrow. Below the schema section is a 'THREAT TRIAGE' section with a 'RULES 0' field and a small icon and a right arrow. At the bottom, there are three buttons: 'SAVE' (blue), 'CANCEL' (dark grey), and 'Advanced' (light blue).


2. In the Schema box, click  (expand window button).

The Management module displays the Schema panel and populates it with message, field, and value information.

The Sample field, at the top of the panel, displays a parsed version of a sample message from the sensor. The Management module will test your transformations against this parsed message.

You can use the right and left arrow buttons in the Sample field to view the parsed version of each sample message available from the sensor.

You can apply transformations to an existing field or create a new field. Typically users choose to create and transform a new field, rather than transforming an existing field.

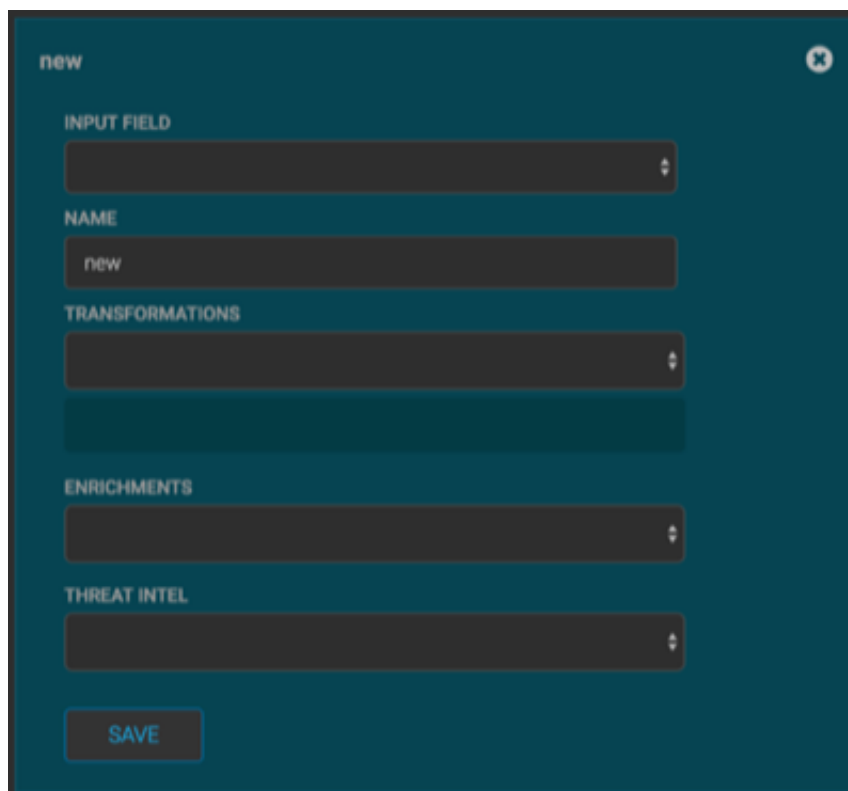
3. To add a new transformation, either click the  next to a field or click the



(plus sign) at the bottom of the **Schema** panel.

The module displays a new dialog box for your transformations.

Figure 3.2. New Schema Information Panel



4. Choose the field you want to transform from the **INPUT FIELD** box, enter the name of the new field in the **NAME** field, and then choose a function with the appropriate parameters in the **TRANSFORMATIONS** box. You can apply more than transformation to the input field.

Figure 3.3. Populated Schema Information Panel

ip_dst_addr_copy

INPUT FIELD

ip_dst_addr

NAME

ip_dst_addr_copy

TRANSFORMATIONS

DOMAIN_REMOVE_SUBDOMAINS

DOMAIN_REMOVE_TLD

DOMAIN_REMOVE_TLD(DOMAIN_REMOVE_SUBDOMAINS(ip_dst_addr))

ENRICHMENTS

THREAT INTEL

SAVE

5. Click **SAVE** to save your additions.

The Management module populates the Transforms field with the number of transformations applied to the sensor.

If you change your mind and want to remove a transformation, click "-" next to the field.

6. Click **SAVE** in the parser panel to save the transformation information.

4. Enriching Telemetry Events

After the raw security telemetry events have been parsed and normalized, the next step is to enrich the data elements of the normalized event.

Enrichments add external data from data stores (such as HBase). HCP uses a combination of HBase, Storm, and the telemetry messages in json format to enrich the data in real time to make it relevant and consumable. You can use this enriched information immediately rather than needing to hunt in different silos for the relevant information.

HCP supports two types of configurations: global and sensor specific. The sensor specific configuration configures the individual enrichments and threat intelligence enrichments for a given sensor type (for example, squid). This section describes sensor specific configurations. For more information on global configuration, see [Global Configuration](#).

HCP provides two types of enrichment:

- Telemetry events
- Threat intelligence information

For information about threat intelligence see [Enriching Threat Intelligence Information](#).

HCP provides the following telemetry enrichment sources but you can add your own enrichment sources to suit your needs:

- Asset
- GeoIP
- User



Note

The telemetry data sources for which HCP includes parsers (for example, Bro, Snort, and YAF) already include enrichment topologies. These topologies will become effective when you start the data sources in HCP.

Prior to enabling an enrichment capability within HCP, the enrichment store (which for HCP is primarily HBase) must be loaded with enrichment data. The dataload utilities convert raw data sources to a primitive key (type, indicator) and value and place it in HBase.

HCP supports three types of enrichment loaders:

- Bulk load from HDFS via MapReduce
- Taxii Loader
- Flat File ingestion

For simplicity's sake, we use the bulk loader to load enrichments:

Task	Description
Bulk Loading Enrichment Information	Bulk loading is used to load information that does not change frequently.
Mapping Fields to HBase Enrichments	Now that you have data flowing into the HBase table, you need to ensure that the enrichment topology can be used to enrich the data flowing past.
OPTIONAL:Global Configuration	Global enrichments are applied to all data sources as opposed to other enrichments that are applied at the field level to individual sensors. This type of enrichment can save you time by applying common enrichments to all of your sensors.
Verify Events are Enriched	After you finish enriching your new data source, you should verify that the output matches your enrichment information.

4.1. Bulk Loading Enrichment Information

Bulk loading is used to load information that does not change frequently. For example, bulk loading is ideal for loading from an asset database on a daily or even weekly basis because you don't typically change the number of assets on a network very often.

Enrichment data can be bulk loaded from the local file system, HDFS. The enrichment loader transforms the enrichment into a JSON format that is understandable to Metron. The loading framework has additional capabilities for aging data out of the enrichment stores based on time. Once the stores are loaded, an enrichment bolt that can interact with the enrichment store can be incorporated into the enrichment topology.

You can bulk load enrichment information from the following sources:

- CSV Flat File Ingestion
- HDFS via MapReduce
- Taxii Loader

For our example, we will use CSV flat file ingestion. For more information about Taxii Loader, see [Bulk Loading Threat Intelligence Information](#). For more information about HDFS via MapReduce, see [Bulk Loading Threat Intelligence Information](#)

Bulk loading information consists of the following major steps:

Task	Description
OPTIONAL: Create a Mock Enrichment Source [27]	For our runbook demonstration we create a mock enrichment source. In your production environment you will want to use a genuine enrichment source.
Configuring an Extractor Configuration File [27]	The extractor configuration file is used to bulk load the enrichment store into HBase.
Configuring Element-to-Enrichment Mapping [29]	This section configures what element of a tuple should be enriched with what enrichment type.
Running the Enrichment Loader [30]	After the enrichment source and enrichment configuration are defined, you must run the loader to move the data from the enrichment source to the HCP enrichment store (HBase) and store the enrichment configuration in ZooKeeper.

4.1.1. OPTIONAL: Create a Mock Enrichment Source

For our runbook demonstration, we create a mock enrichment source. In your production environment you will want to use a genuine enrichment source.

To create a mock enrichment source, complete the following steps:

1. As root user, log into \$HOST_WITH_ENRICHMENT_TAG.

```
sudo -s $HOST_WITH_ENRICHMENT_TAG
```

2. Copy and paste the following data into a file called `whois_ref.csv` in `$METRON_HOME/config`. This CSV file represents our enrichment source.

```
google.com, "Google Inc.", "US", "Dns Admin",874306800000
work.net, "", "US", "PERFECT PRIVACY, LLC",788706000000
capitalone.com, "Capital One Services, Inc.", "US", "Domain Manager",
795081600000
cisco.com, "Cisco Technology Inc.", "US", "Info Sec",547988400000
cnn.com, "Turner Broadcasting System, Inc.", "US", "Domain Name Manager",
748695600000
news.com, "CBS Interactive Inc.", "US", "Domain Admin",833353200000
nba.com, "NBA Media Ventures, LLC", "US", "C/O Domain Administrator",
786027600000
espn.com, "ESPN, Inc.", "US", "ESPN, Inc.",781268400000
pravda.com, "Internet Invest, Ltd. dba Imena.ua", "UA", "Whois privacy
protection service",806583600000
hortonworks.com, "Hortonworks, Inc.", "US", "Domain Administrator",
1303427404000
microsoft.com, "Microsoft Corporation", "US", "Domain Administrator",
673156800000
yahoo.com, "Yahoo! Inc.", "US", "Domain Administrator",790416000000
rackspace.com, "Rackspace US, Inc.", "US", "Domain Admin",903092400000
landl.co.uk, "1 & 1 Internet Ltd", "UK", "Domain Admin",943315200000
```

Make sure you don't have an empty newline character as the last line of the CSV file, as that will result in a null pointer exception.

4.1.2. Configuring an Extractor Configuration File

The extractor configuration file is used to bulk load the enrichment store into HBase.

Complete the following steps to configure the extractor configuration file:

1. Log in as root to the host on which Metron is installed.

```
sudo -s $METRON_HOME
```

2. Determine the schema of the enrichment source.

The schema of our mock enrichment source is `domain|owner|registeredCountry|registeredTimestamp`.

3. Create an extractor configuration file called `extractor_config.json` at `$METRON_HOME/config` and populate it with the enrichment source schema.

For example:

```
{
  "config" : {
    "columns" : {
      "domain" : 0
      , "owner" : 1
      , "home_country" : 2
      , "registrar" : 3
      , "domain_created_timestamp" : 4
    }
    , "indicator_column" : "domain"
    , "type" : "whois"
    , "separator" : ","
  }
  , "extractor" : "CSV"
}
```

4. You can transform and filter the enrichment data as it is loaded into HBase by using Stellar extractor properties in the extractor configuration file. HCP supports the following Stellar extractor properties:

Extractor Property	Description	Example
value_transform	Transforms fields defined in the columns mapping with Stellar transformations. New keys introduced in the transform are added to the key metadata.	"value_transform" : { "domain" : "DOMAIN_REMOVE_TLD(domain)" }
value_filter	Allows additional filtering with Stellar predicates based on results from the value transformations. In the following example, records whose domain property is empty after removing the TLD are omitted.	"value_filter" : "LENGTH(domain) > 0", "indicator_column" : "domain",
indicator_transform	Transforms the indicator column independent of the value transformations. You can refer to the original indicator value by using <code>indicator</code> as the variable name, as shown in the following example. In addition, if you prefer to piggyback your transformations, you can refer to the variable <code>domain</code> , which allows your indicator transforms to inherit transformations done to this value during the value transformations.	"indicator_transform" : { "indicator" : "DOMAIN_REMOVE_TLD(indicator)" }
indicator_filter	Allows additional filtering with Stellar predicates based on results from the value transformations. In the following example, records whose indicator value is empty after removing the TLD are omitted.	"indicator_filter" : "LENGTH(indicator) > 0", "type" : "top_domains",

If you include all of the supported Stellar extractor properties in the extractor configuration file, it will look similar to the following:

```
{
  "config" : {
    "zk_quorum" : "$ZOOKEEPER_HOST:2181",
    "columns" : {
      "rank" : 0,
```



```
"domain" : 1
},
"value_transform" : {
  "domain" : "DOMAIN_REMOVE_TLD(domain)"
},
"value_filter" : "LENGTH(domain) > 0",
"indicator_column" : "domain",
"indicator_transform" : {
  "indicator" : "DOMAIN_REMOVE_TLD(indicator)"
},
"indicator_filter" : "LENGTH(indicator) > 0",
"type" : "top_domains",
"separator" : ",",
},
"extractor" : "CSV"
}
```

Running a file import with the above data and extractor configuration will result in the following two extracted data records:

Indicator	Type	Value
google	top_domains	{ "rank": "1", "domain": "google" }
yahoo	top_domains	{ "rank": "2", "domain": "yahoo" }

5. Remove any non-ASCII invisible characters that might have been included if you copy and pasted:

```
iconv -c -f utf-8 -t ascii extractor_config_temp.json -o extractor_config.json
```



Note

The `extractor_config.json` file is not stored anywhere by the loader. This file is used once by the bulk loader to parse the enrichment dataset. If you would like to keep a copy of this file, be sure to save a copy to another location.

4.1.3. Configuring Element-to-Enrichment Mapping

We now need to configure what element of a tuple should be enriched with what enrichment type.

This configuration is stored in ZooKeeper.

1. Log in as root user to the host that has Metron installed.

```
sudo -s $METRON_HOME
```

2. Copy and paste the following into a file called `enrichment_config_temp.json` at `$METRON_HOME/config`.

```
{
  "zkQuorum" : "$ZOOKEEPER_HOST:2181"
, "sensorToFieldList" : {
  "squid" : {
    "type" : "ENRICHMENT"
  , "fieldToEnrichmentTypes" : {
    "domain_without_subdomains" : [ "whois" ]
  }
}
```

```
}  
}  
}
```

3. Remove any non-ASCII invisible characters that might have been included if you copy and pasted:

```
iconv -c -f utf-8 -t ascii enrichment_config_temp.json -o enrichment_config.  
json
```

4.1.4. Running the Enrichment Loader

After the enrichment source and enrichment configuration are defined, you must run the loader to move the data from the enrichment source to the HCP enrichment store (HBase) and store the enrichment configuration in ZooKeeper.



Note

There is a special configuration parameter to the Extractor config that is only considered during this loader:

inputFormatHandler	This specifies how to consider the data. The two implementations are <code>BY_LINE</code> and <code>org.apache.metron.dataloads.extractor.inputformat.</code>
--------------------	---

The default is `BY_LINE`, which makes sense for a list of CSVs where each line indicates a unit of information which can be imported. However, if you are importing a set of STIX documents, then you want each document to be considered as input to the Extractor.

1. Use the loader to move the enrichment source to the enrichment store in ZooKeeper.

Perform the following from the location containing your extractor and enrichment configuration files and your enrichment source. In our example, this information is located at `$METRON_HOME/config`.

```
$METRON_HOME/bin/flatfile_loader.sh -n enrichment_config.json -i whois_ref.  
csv -t enrichment -c t -e  
extractor_config.json
```

The parameters for the utility are as follows:

Short Code	Long Code	Required	Description
-h		No	Generate the help screen/set of options
-e	-extractor_config	Yes	JSON document describing the extractor for this input data source
-t	-hbase_table	Yes	The HBase table to import into
-c	-hbase_cf	Yes	The HBase table column family to import into
-i	-input	Yes	The input data location on local disk. If this is a

Short Code	Long Code	Required	Description
			file, then that file will be loaded. If this is a directory, then the files will be loaded recursively under that directory.
-l	-log4j	No	The log4j properties file to load
-n	-enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

HCP loads the enrichment data into Apache HBase and establishes a ZooKeeper mapping. The data is extracted using the extractor and configuration defined in the `extractor_config.json` file and populated into an HBase table called `enrichment`.

2. Verify that the logs were properly ingested into HBase:

```
hbase shell
scan 'enrichment'
```

3. Verify that the ZooKeeper enrichment tag was properly populated:

```
$METRON_HOME/bin/zk_load_configs.sh -m DUMP -z $ZOOKEEPER_HOST:2181
```

4. Generate some data by using the Squid client to execute requests.

- a. Use ssh to access the host for Squid.

- b. Start Squid and navigate to `/var/log/squid`:

```
sudo service squid start
sudo su -
cd /var/log/squid
```

- c. Generate some data by entering the following:

```
squidclient http://www.cnn.com
```

4.2. Mapping Fields to HBase Enrichments

Now that you have data flowing into the HBase table, you need to ensure that the enrichment topology can be used to enrich the data flowing past.

You can refine the parser output in three ways:

- Transformations
- Enrichments
- Threat Intel

Each of the parser outputs is added or modified in the **Schema** field. To modify any of the parser outputs, complete the following steps:



Note

To load sample data from your sensor, the sensor must be running and producing data.

1. Display the Management module UI.
2. Select the new sensor from the list of sensors on the main window.

3.

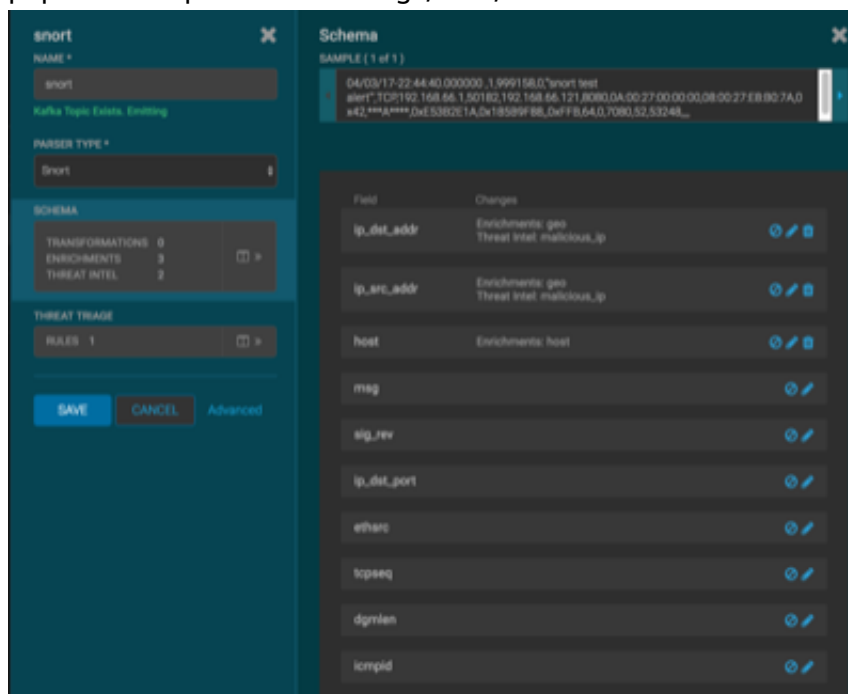
Click the pencil icon in the list of tool icons  for the new sensor.

The Management Module displays the sensor panel for the new sensor.

4.

In the Schema box, click  (expand window button).

The Management module displays a second panel and populates the panel with message, field, and value information.



The Sample field, at the top of the panel, displays a parsed version of a sample message from the sensor. The Management module will test your transformations against these parsed messages.

You can use the right and left arrow buttons in the Sample field to view the parsed version of each sample message available from the sensor.

5. You can apply transformations to an existing field or create a new field. Click



the  (edit icon) next to a field to apply transformations to that field. Or click

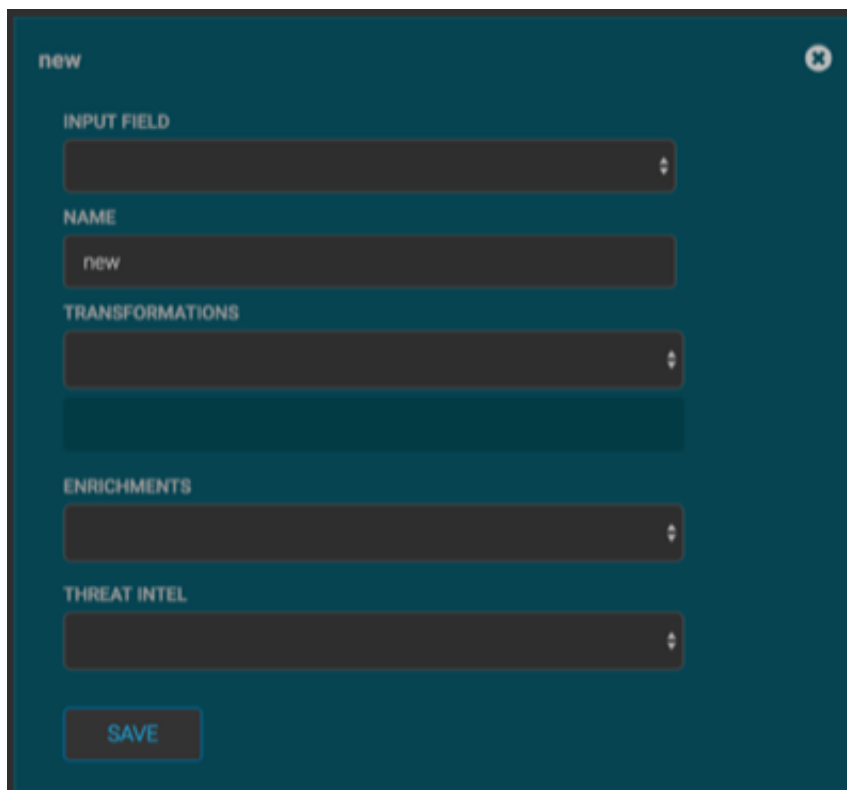


(plus sign) at the bottom of the Schema panel to create new fields.

Typically users store transformations in a new field rather than overriding existing fields.

For both options, the Management module expands the panel with a dialog box containing fields in which you can enter field information.

Figure 4.1. New Schema Information Panel



6. In the dialog box, enter the name of the new field in the **NAME** field, choose an input field from the **INPUT FIELD** box, and choose your transformation from the **TRANSFORMATIONS** field or enrichment from the **ENRICHMENTS** field.


For example, to create a new field showing the lower case version of the method field, do the following:

- Enter method-uppercase in the **NAME** field.
- Choose `method` from the **INPUT FIELD**.
- Choose `TO_UPPER` in the **TRANSFORMATIONS** field.

Your new schema information panel should look like this:

Figure 4.2. Populated New Schema Information Panel

The screenshot shows a configuration panel for a schema named 'method_uppercase'. It includes sections for input fields, names, transformations, enrichments, and threat intelligence. The 'TRANSFORMATIONS' section is currently set to 'TO_UPPER', and the preview shows 'TO_UPPER(method)'. A 'SAVE' button is located at the bottom left of the panel.

7. Click **SAVE** to save your changes.
8. You can suppress fields from showing in the Index by clicking  (suppress icon).
9. Click **SAVE** to save the changed information.

The Management module updates the Schema field with the number of changes applied to the sensor.

4.3. OPTIONAL: Global Configuration

Global enrichments are applied to all data sources as opposed to other enrichments that are applied at the field level to individual sensors. This type of enrichment can save you time by applying common enrichments to all of your sensors. For more information on configuring global configurations, see [Global Configuration](#).

4.4. Verify That the Events Are Enriched

After you finish enriching your new data source, you should verify that the output matches your enrichment information.

By convention, the index where the new messages are indexed is called `squid_index_[timestamp]` and the document type is `squid_doc`.

Use the Elasticsearch Head plug-in to verify that the messages display your enrichment information:

1. Log in to `$SEARCH_HOST` host:

```
ssh into Host $SEARCH_HOST
```

2. Install the head plug-in:

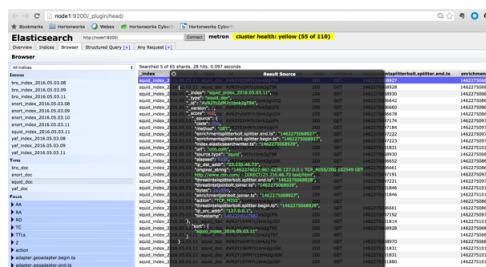
```
/usr/share/elasticsearch/bin/plugin install mobz/elasticsearch-head/1.x
```

3. Navigate to ElasticSearch Head UI: `http://$SEARCH_HOST:9200/_plugin/head/`.

4. Click the Browser tab and select the squid document in the left panel; then select one of the sample docs.

You should see something like the following:

Figure 4.3. Elasticsearch



5. Review the output to ensure it reflects the enrichment functions you used.

5. Enriching Threat Intelligence Information

You can enrich your threat intelligence information just like you enriched your telemetry information.

You can choose to skip this section and come back to it later if you don't want to enrich your threat intelligence information at this time.

Metron provides an extensible framework to plug in threat intel sources. Each threat intel source has two components: an enrichment data source and an enrichment bolt. The threat intelligence feeds are loaded into a threat intelligence store similar to how the enrichment feeds are loaded. The keys are loaded in a key-value format. The key is the indicator and the value is the JSON formatted description of what the indicator is.

We recommend using a threat feed aggregator such as [Soltra](#) to dedup and normalize the feeds via STIX/Taxii. Metron provides an adapter that is able to read Soltra-produced STIX/Taxii feeds and stream them into HBase, which is the preferred data store to back high-speed threat intel lookups on HCP. HCP additionally provides a flat file and STIX bulk loader that can normalize, dedup, and bulk load or poll threat intel data into HBase even without the use of a threat feed aggregator.

5.1. OPTIONAL: Create a Mock Threat Intel Feed Source

Metron is designed to work with STIX/Taxii threat feeds, but can also be bulk loaded with threat data from a CSV file. In this example, we will explore the CSV example. The same loader framework that is used for enrichment here is used for threat intelligence. Similar to enrichments, we need to set up a data.csv file, the extractor config JSON, and the enrichment config JSON.

For this example, we will use a Zeus malware tracker list located here: <https://zeustracker.abuse.ch/blocklist.php?download=domainblocklist>.

1. Log into the \$HOST_WITH_ENRICHMENT_TAG as root.
2. Copy the contents from the Zeus malware tracker list link to a file called domainblocklist.csv.

```
curl https://zeustracker.abuse.ch/blocklist.php?download=domainblocklist | grep -v "^#" | grep -v "^$" | grep -v "^https" | awk '{print $1",abuse.ch"}' > domainblocklist.csv
```

5.2. Configuring an Extractor Configuration File

Complete the following steps to configure the extractor configuration file:

1. Log in as root to the host on which Metron is installed.


```
sudo -s $METRON_HOME
```

2. Determine the schema of the enrichment source.

The schema of our mock enrichment source is domain|owner|registeredCountry|registeredTimestamp.

3. Create an extractor configuration file called `extractor_config_temp.json` at `$METRON_HOME/config` and populate it with the threat intel source schema.

HCP supports a subset of STIX messages for importation:

STIX Type	Specific Type	Enrichment Type Name
Address	IPV_4_ADDR	address:IPV_4_ADDR
Address	IPV_6_ADDR	address:IPV_6_ADDR
Address	E_MAIL	address:E_MAIL
Address	MAC	address:MAC
Domain	FQDN	domain:FQDN
Hostname		hostname

The following example configures the STIX extractor to load from a series of STIX files, however we only want to bring in IPv4 addresses from the set of all possible addresses. Note that if no categories are specified for import, all are assumed. Also, only address and domain types allow filtering via `stix_address_categories` and `stix_domain_categories` config parameters.

For example:

```
{
  "config" : {
    "stix_address_categories" : "IPV_4_ADDR"
  },
  "extractor" : "STIX"
}
```

4. Remove any non-ASCII invisible characters that might have been included if you copy and pasted:

```
iconv -c -f utf-8 -t ascii extractor_config_temp.json -o extractor_config.json
```

5. OPTIONAL: You also have the ability to transform and threat intel data using Stellar as it is loaded into HBase. This feature is available to all extractor types.

As an example, we will be providing a CSV list of top domains as an enrichment and filtering the value metadata, as well as the indicator column, with Stellar expressions.

```
{
  "config" : {
    "zk_quorum" : "node1:2181",
    "columns" : {
      "rank" : 0,
      "domain" : 1
    },
    "value_transform" : {
```

```

        "domain" : "DOMAIN_REMOVE_TLD(domain)"
    },
    "value_filter" : "LENGTH(domain) > 0",
    "indicator_column" : "domain",
    "indicator_transform" : {
        "indicator" : "DOMAIN_REMOVE_TLD(indicator)"
    },
    "indicator_filter" : "LENGTH(indicator) > 0",
    "type" : "top_domains",
    "separator" : ",",
    },
    "extractor" : "CSV"
}

```

5.3. Configuring Element-to-Threat Intel Feed Mapping

We now need to configure what element of a tuple should be enriched with what enrichment type.

This configuration is stored in ZooKeeper.

1. Log in as root user to the host that has Metron installed.

```
sudo -s $METRON_HOME
```

2. Copy and paste the following into a file called `enrichment_config_temp.json` at `$METRON_HOME/config`.

```

{
  "zkQuorum" : "localhost:2181"
  , "sensorToFieldList" : {
    "bro" : {
      "type" : "THREAT_INTEL"
      , "fieldToEnrichmentTypes" : {
        "ip_src_addr" : [ "malicious_ip" ]
        , "ip_dst_addr" : [ "malicious_ip" ]
      }
    }
  }
}

```

You must specify the following:

- The zookeeper quorum which holds the cluster configuration
- The mapping between the fields in the enriched documents and the enrichment types.

This configuration allows the ingestion tools to update ZooKeeper post-ingestion so that the enrichment topology can take advantage immediately of the new type.

3. Remove any non-ASCII invisible characters that might have been included if you copy and pasted:

```
iconv -c -f utf-8 -t ascii enrichment_config_temp.json -o enrichment_config.json
```

5.4. Run the Threat Intelligence Loader

Now that you have the threat intel source, threat intel extractor, and threat intel mapping config defined, you can run the loader to move the data from the threat intel source to the Metron threat intel Store and store the enrichment config in ZooKeeper.

1. Log into the \$HOST_WITH_ENRICHMENT_TAG as root.

2. Run the loader:

```
/usr/metron/$METRON_RELEASE/bin/flatfile_loader.sh -n enrichment_config.json  
-i domainblocklist.csv -t threatintel -c t -e extractor_config.json
```

The previous command adds the threat intel data into HBase and establishes a ZooKeeper mapping. The data is populated into an HBase table called threatintel.

3. To verify that the logs were properly ingested into HBase, run the following command:

```
hbase shell  
scan 'threatintel'
```

4. Now check if the ZooKeeper enrichment tag was properly populated:

```
/usr/metron/$METRON_RELEASE/bin/zk_load_configs.sh -m DUMP -z  
$ZOOKEEPER_HOST:2181
```

You should see a configuration for the Squid sensor something like the following

```
{  
  "index" : "squid",  
  "batchSize" : 1,  
  "enrichment" : {  
    "fieldMap" : {  
      "hbaseEnrichment" : [ "ip_src_addr" ]  
    },  
    "fieldToTypeMap" : {  
      "ip_src_addr" : [ "user" ]  
    },  
    "config" : { }  
  },  
  "threatIntel" : {  
    "fieldMap" : { },  
    "fieldToTypeMap" : { },  
    "config" : { },  
    "triageConfig" : {  
      "riskLevelRules" : { },  
      "aggregator" : "MAX",  
      "aggregationConfig" : { }  
    }  
  },  
  "configuration" : { }  
}
```

5. Generate some data by using the Squid client to execute http requests.

```
squidclient http://www.actdhaka.com
```

5.5. Mapping Fields to HBase Enrichments

Now that you have data flowing into the HBase table, you need to ensure that the enrichment topology can be used to enrich the data flowing past.

You can refine the parser output in three ways:

- Transformations
- Enrichments
- Threat Intel

Each of the parser outputs is added or modified in the **Schema** field. To modify any of the parser outputs, complete the following steps:



Note

To load sample data from your sensor, the sensor must be running and producing data.

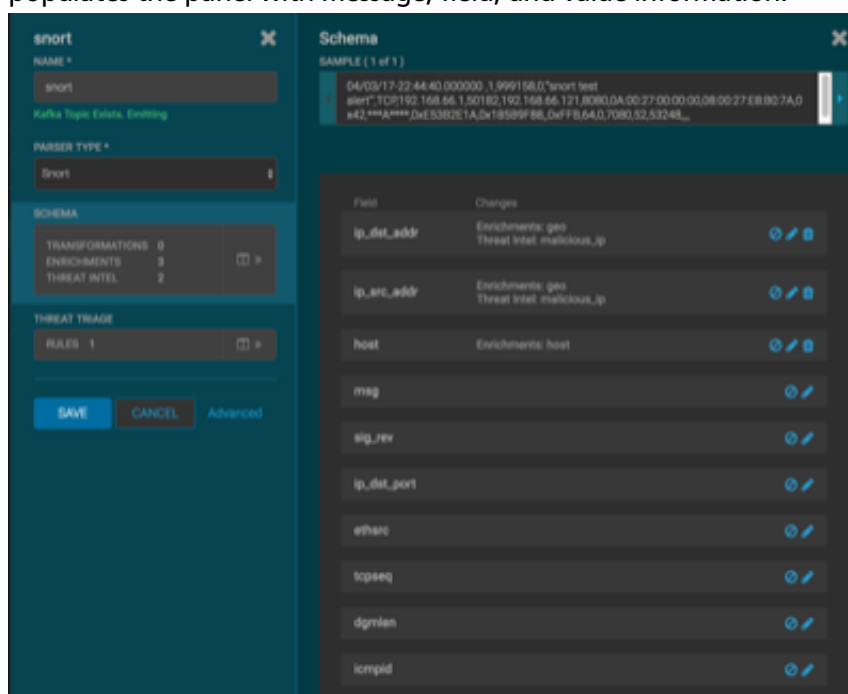
1. Select the new sensor from the list of sensors on the main window.

2. Click the pencil icon in the list of tool icons  for the new sensor.

The Management Module displays the sensor panel for the new sensor.

3. In the Schema box, click  (expand window button).

The Management module displays a second panel and populates the panel with message, field, and value information.



The screenshot shows the Management Module interface. On the left, the 'snort' sensor configuration panel is visible, showing fields for NAME, KAFKA TOPIC, PARSER TYPE, SCHEMA, and THREAT TRIAGE. The SCHEMA section lists TRANSFORMATIONS (0), ENRICHMENTS (3), and THREAT INTEL (2). The THREAT TRIAGE section shows RULES (1). At the bottom are buttons for SAVE, CANCEL, and Advanced.

On the right, the 'Schema' panel is expanded, showing a list of fields and their corresponding changes. The fields are:

Field	Changes	Actions
ip_dst_addr	Enrichments: geo Threat Intel: malicious_ip	
ip_src_addr	Enrichments: geo Threat Intel: malicious_ip	
host	Enrichments: host	
msg		
sig_rev		
ip_dst_port		
ethsrc		
tcpseq		
dgmten		
icmpid		

The Sample field, at the top of the panel, displays a parsed version of a sample message from the sensor. The Management module will test your transformations against these parsed messages.

You can use the right and left arrow buttons in the Sample field to view the parsed version of each sample message available from the sensor.

4. You can apply transformations to an existing field or create a new field. Click



the (edit icon) next to a field to apply transformations to that field. Or click



(plus sign) at the bottom of the Schema panel to create new fields.

Typically users store transformations in a new field rather than overriding existing fields.

For both options, the Management module expands the panel with a dialog box containing fields in which you can enter field information.

Figure 5.1. New Schema Information Panel

The dialog box is titled "new" and contains the following fields:

- INPUT FIELD**: A dropdown menu.
- NAME**: A text input field containing the text "new".
- TRANSFORMATIONS**: A dropdown menu.
- ENRICHMENTS**: A dropdown menu.
- THREAT INTEL**: A dropdown menu.
- SAVE**: A button at the bottom.

5. In the dialog box, enter the name of the new field in the **NAME** field, choose an input field from the **INPUT FIELD** box, and choose your transformation from the **TRANSFORMATIONS** field or enrichment from the **ENRICHMENTS** field.


For example, to create a new field showing the lower case version of the method field, do the following:

- Enter method-uppercase in the **NAME** field.
- Choose `method` from the **INPUT FIELD**.
- Choose `TO_UPPER` in the **TRANSFORMATIONS** field.

Your new schema information panel should look like this:

Figure 5.2. Populated New Schema Information Panel

The screenshot shows a configuration panel for a new schema field named 'method_uppercase'. The panel is organized into several sections: 'INPUT FIELD' with a dropdown menu currently showing 'method'; 'NAME' with a text input field containing 'method_uppercase'; 'TRANSFORMATIONS' with a dropdown menu showing 'TO_UPPER', a minus button to the right, and a preview box below showing 'TO_UPPER(method)'; 'ENRICHMENTS' with an empty dropdown menu; and 'THREAT INTEL' with an empty dropdown menu. At the bottom left of the panel is a 'SAVE' button. The panel has a dark teal background and a title bar at the top with the name 'method_uppercase' and a close icon.

6. Click **SAVE** to save your changes.
7. You can suppress fields from showing in the Index by clicking  (suppress icon).
8. Click **SAVE** to save the changed information.

The Management module updates the Schema field with the number of changes applied to the sensor.

5.6. Verify That the Threat Intel Events Are Enriched

*** Is this correct? ***

After you finish enriching your new data source, you should verify that the output matches your enrichment information.

By convention, the index where the new messages are indexed is called `squid_index_[timestamp]` and the document type is `squid_doc`.

Use the Elasticsearch Head plug-in to verify that the messages display your enrichment information:

1. Log in to `$SEARCH_HOST` host:

```
ssh into Host $SEARCH_HOST
```

2. Install the head plug-in:

If you are connected to the internet, you can use the following command:

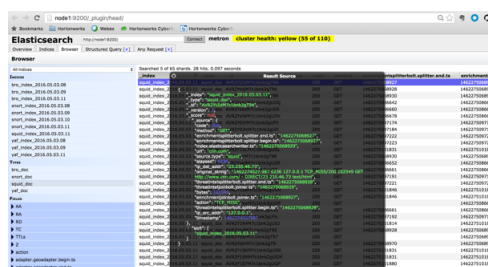
```
/usr/share/elasticsearch/bin/plugin install mobz/elasticsearch-head/1.x
```

If you are not connected to the internet, complete the following steps:

- a. Download the tar.gz by hand.
 - b. untar into the `plugins` directory.
 - c. At this point, the URL `http://:9200/_plugin/head/` should work (don't forget the trailing `/`).
3. Navigate to Elasticsearch Head UI: `http://$SEARCH_HOST:9200/_plugin/head/`.
 4. Click the Browser tab and select the squid document in the left panel; then select one of the sample docs.

You should see something like the following:

Figure 5.3. Elasticsearch * Needs updating *****



5. Review the output to ensure it reflects the enrichment functions you used.

6. Prioritizing Threat Intelligence

Not all threat intelligence indicators are equal. Some require immediate response, while others can be dealt with or investigated as time and availability permits. As a result you need to triage and rank threats by severity.

In HCP, you assign severity by associating possibly complex conditions with numeric scores. Then, for each message, you use a configurable aggregation function to evaluate the set of conditions and to aggregate the set of numbers for matching conditions. This aggregated score is added to the message in the `threat.triage.level` field. For more information about Stellar and threat triage configurations, see [Using Stellar to Set up Threat Triage Configurations](#).

This section details the steps to understand and create severity rules, configure them in ZooKeeper, and view the resulting alerts in the HCP Investigation module:

- [Prerequisites \[44\]](#)
- [Performing Threat Triage \[44\]](#)
- [Viewing Triaged or Scored Alerts \[47\]](#)

6.1. Prerequisites




Before you can prioritize a threat intelligence enrichment, you must ensure that the enrichment is working properly. See [Enriching Telemetry Events](#) and [Enriching Threat Intelligence Information](#) for more information.

6.2. Performing Threat Triage

To create a threat triage rule configuration, you must first define your rules. These rules identify the conditions in the data source data flow and associate alert scores with those conditions. Following are some examples:

- Rule 1 If a threat intelligence enrichment type is alerted, imagine that you want to receive an alert score of 5.
- Rule 2 If the URL ends with neither .com nor .net, then imagine that you want to receive an alert score of 10.

To create these rules, complete the following steps:

1.  Click the  (edit button) for your sensor.
2. In the Threat Triage field, click the  icon (expand window).

The module displays the Threat Triage Rules panel.

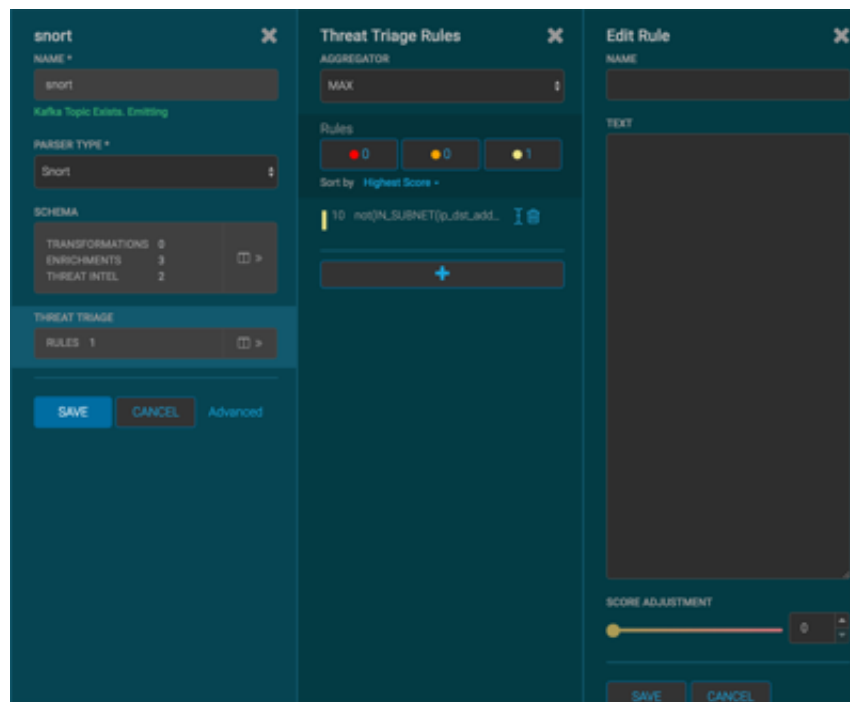
Figure 6.1. Threat Triage Rules Panel

The screenshot displays the 'Threat Triage Rules' configuration interface. On the left, the 'snort' parser configuration is visible, including fields for NAME, PARSER TYPE, and a SCHEMA table listing TRANSFORMATIONS (1), ENRICHMENTS (4), and THREAT INTEL (2). The right panel shows the 'Threat Triage Rules' configuration, including an AGGREGATOR dropdown set to 'MAX', a Rules section with three colored buttons (red 0, yellow 0, green 1), and a Sort by dropdown set to 'Highest Score'. Below the Rules section is a list of rules, currently showing one rule: '10 not(IN_SUBNET(ip_dst_add...)' with edit and delete icons. At the bottom of the left section are 'SAVE', 'CANCEL', and 'Advanced' buttons.

3. Click the + button to add a rule.

The module displays the **Edit Rule** panel.

Figure 6.2. Edit Rule Panel



4. Assign a name to the new rule by entering the name in the NAME field.
5. In the Text field, enter the syntax for the new rule.

For example:

```
Exists(IsAlert)
```

6. Use the **SCORE ADJUSTMENT** slider to choose the threat score for the rule.
7. Click **SAVE** to save the new rule.

The new rule is listed in the Threat Triage Rules panel.

8. Choose how you want to aggregate your rules by choosing a value from the Aggregator menu.

You can choose between:

- | | |
|---------------|--|
| MAX | The maximum of all of the associated values for matching queries. |
| MIN | The minimum of all of the associated values for matching queries. |
| MEAN | the mean of all of the associated values for matching queries. |
| POSITIVE_MEAN | The mean of the positive associated values for the matching queries. |

9. You can use the **Rules** section and the **Sort by** pull down menu below the **Rules** section to filter how threat triages display.

For example, to display only high levels alerts, click the box containing the red indicator. To sort the high level alerts from highest to lowest, choose **Highest Score** from the **Sort by** pull down menu.

10. Click **SAVE** on the Sensor panel to save your changes.

6.3. Viewing Triaged or Scored Alerts

You can view triaged alerts in the indexing topic in Kafka or in the triaged alert panel in the HCP Metron dashboard.

- [Threat Triage Alert Panel \[47\]](#)
- [HCP Metron Dashboard View of Alerts \[47\]](#)

6.3.1. Threat Triage Alert Panel

To view the Threat Triage Alert Panel in Kafka, complete the following steps. An alert in the indexing topic in Kafka will appear similar to the following:

1. List the Kafka topics to find the threat triage alert panel:

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper
$ZOOKEEPER_HOST:2181 --list
```

2. View the threat triage alert Kafka topic:

```
cd $METRON_HOME/bin/.stellar
THREAT_TRIAGE_PRINT(conf)
```

The topic should appear similar to the following:

```
> THREAT_TRIAGE_PRINT(conf)
#####
# Name                               # Comment # Triage Rule
#                               # Score # Reason
#
#####
# Abnormal DNS Port #               # source.type == "bro" and protocol == "dns" and
ip_dst_port != 53 # 10             # FORMAT("Abnormal DNS Port: expected: 53, found:
%s:%d", ip_dst_addr, ip_dst_port) #
#####
```

6.3.2. HCP Metron Dashboard View of Alerts

The following figure shows you an example of a triaged alert panel in the HCP Metron dashboard. For URLs from cnn.com, no threat alert is shown, so no triage level is set. Notice the lack of a threat.triage.level field:

Figure 6.3. Investigation Module Triaged Alert Panel



Threat Alerts

Time	sourceType	threat-triage-level	full_hostname	ip_src_addr	ip_dst_addr
* June 29th 2016, 17:14:30.483	squld	5	www.actthaka.com	127.0.0.1	198.50.239.7
* June 29th 2016, 17:14:29.196	squld	5	www.actthaka.com	127.0.0.1	198.50.239.7
* June 29th 2016, 17:14:28.025	squld	5	www.actthaka.com	127.0.0.1	198.50.239.7

7. Configuring Indexing

The indexing topology is a topology dedicated to taking the data from a topology that has been enriched and storing the data in one or more supported indices. More specifically, the enriched data is ingested into Kafka, written in an indexing batch or bolt with a specified size, and sent to one or more specified indices. The configuration is intended to configure the indexing used for a given sensor type (for example, snort).

Currently, HCP supports the following indices:

- Elasticsearch
- Solr
- HDFS under /apps/metron/enrichment/indexed

Depending on how you start the indexing topology, it can have HDFS and either elasticsearch or SOLR writers running.

Just like the Global Configuration file, the Indexing Configuration file format is a JSON file stored in ZooKeeper and on disk at \$METRON_HOME/config/zookeeper/ indexing.



Note

Errors during indexing are sent to a Kafka queue called index_errors

Within the sensor-specific configuration, you can configure the individual writers. The parameters currently supported are:

index	The name of the index to write to (defaulted to the name of the sensor).
batchSize	The size of the batch that is written to the indices at once (defaulted to 1).
enabled	Whether the index or writer is enabled (default true).

7.1. Default Configuration

If you do not configure the individual writers, the sensor-specific configuration will use the default values. You can choose to use this default configuration by either not creating the Indexing Configuration file or by entering the following in the file. You can name the file anything you like, for example index_config.json, but it must be located at \$METRON_HOME/config/zookeeper/indexing.

```
{  
}
```


If a writer configuration is unspecified, then a warning is indicated in the Storm console. For example, WARNING: Default and (likely) unoptimized writer config used for hdfs writer and sensor squid. You can ignore this warning message if you intend to use the default configuration.

This default configuration uses the following configuration:

- elasticsearch writer
 - index name the same as the sensor
 - batch size of 1
 - enabled
- hdfs writer
 - index name the same as the sensor
 - batch size of 1
 - enabled

7.2. Specifying Index Parameters

You can to specify the parameters for the writers rather than using the default values using the HCP Management Module..

1. Edit your sensor by clicking  (the edit button) next your sensor in the Management Module.
2. Click the **Advanced** button next to **Save** and **Cancel**.

The Management Module expands the panel to display the Advanced fields.

Figure 7.1. Management Module Advanced Panel

The screenshot shows the 'Advanced' configuration panel for a sensor. It contains several sections for configuring different data sources:

- RAW JSON**: A 'Select' button with a right-pointing arrow icon.
- HDFS INDEX NAME**: A text input field containing 'squid' and an information icon.
- HDFS BATCH SIZE**: A numeric input field set to '5' with up/down arrows.
- HDFS ENABLED**: A checked checkbox.
- ELASTICSEARCH INDEX NAME**: A text input field containing 'squid'.
- ELASTICSEARCH BATCH SIZE**: A numeric input field set to '5' with up/down arrows.
- ELASTICSEARCH ENABLED**: A checked checkbox.
- SOLR INDEX NAME**: An empty text input field.
- SOLR BATCH SIZE**: A numeric input field set to '1' with up/down arrows.
- SOLR ENABLED**: A checked checkbox.
- PARSER CONFIG**: A text input field containing 'grokPath'.

At the bottom, there is a preview of the configuration path: '/patterns/squid'.

3. Enter index configuration information for your sensor.
4. Click **Save** to save your changes and push your configuration to ZooKeeper.

7.3. Turning off HDFS Writer

You can also turn off the HDFS index or writer using the following syntax in the index.json file.

```
{
```

```
"elasticsearch": {  
  "index": "foo",  
  "enabled" : true  
},  
"hdfs": {  
  "index": "foo",  
  "batchSize": 100,  
  "enabled" : false  
}  
}
```


8. Setting Up a Profile

A profile describes the behavior of an entity on a network. An entity can be a server, user, subnet, or application. Once you generate a profile defining what normal behavior looks like, you can build models that identify anomalous behavior.

- [Installing Profiler \[53\]](#)
- [Creating a Profile \[54\]](#)
- [Configuring the Profiler \[57\]](#)
- [Starting the Profiler \[58\]](#)
- [Developing Profiles \[58\]](#)
- [Testing \[59\]](#)

8.1. Installing Profiler

Follow these instructions to install the Profiler. This assumes that HCP has already been installed and validated.

1. Build the Metron RPMs (see Building the [RPMs](#)).

You may have already built the Metron RPMs when core Metron was installed.

```
$ find metron-deployment/ -name "metron-profiler*.rpm"
metron-deployment/packaging/docker/rpm-docker/RPMS/noarch/metron-
profiler-0.4.1-201707131420.noarch.rpm
```

2. Copy the Profiler RPM to the installation host.

The installation host must be the same host on which HCP was installed. Depending on how you installed HCP, the Profiler RPM might have already been copied to this host with the other HCP RPMs.

```
[root@$METRON_HOME ~]# find /localrepo/ -name "metron-profiler*.rpm"
/localrepo/metron-profiler-0.4.0-201707112313.noarch.rpm
```

3. Install the RPM.

```
[root@$METRON_HOME ~]# rpm -ivh metron-profiler-*.noarch.rpm
Preparing...                               #####
[100%]                                     #####
  1:metron-profiler                         #####
[100%]                                     #####
```

```
[root@$METRON_HOME ~]# rpm -ql metron-profiler
/usr/metron
/usr/metron/0.4.1
/usr/metron/0.4.1/bin
/usr/metron/0.4.1/bin/start_profiler_topology.sh
/usr/metron/0.4.1/config
/usr/metron/0.4.1/config/profiler.properties
/usr/metron/0.4.1/flux
/usr/metron/0.4.1/flux/profiler
/usr/metron/0.4.1/flux/profiler/remote.yaml
/usr/metron/0.4.1/lib
/usr/metron/0.4.1/lib/metron-profiler-0.4.0-uber.jar
```

4. Create a table within HBase that will store the profile data. By default, the table is named `profiler` with a column family `P`. The table name and column family must match the Profiler's configuration (see [Configuring the Profiler](#)).

```
$ /usr/hdp/current/hbase-client/bin/hbase shell
hbase(main):001:0> create 'profiler', 'P'
```

5. Edit the configuration file located at `$METRON_HOME/config/profiler.properties`.

```
kafka.zk=node1:2181
kafka.broker=node1:6667
```

Change `kafka.zk` to refer to ZooKeeper in your environment.

Change `kafka.broker` to refer to a Kafka Broker in your environment.

6. Start the Profiler topology.

```
$ cd $METRON_HOME
$ bin/start_profiler_topology.sh
```

At this point the Profiler is running and consuming telemetry messages. We have not defined any profiles yet, so it is not doing anything very useful. The next section walks you through the steps to create your very first "Hello, World!" profile.

8.2. Creating a Profile

To create a profile, complete the following steps:

1. Create a table within HBase that will store the profile data.

The table name and column family must match the Profiler's configuration.

```
$ /usr/hdp/current/hbase-client/bin/hbase shell
hbase(main):001:0> create 'profiler', 'P'
```

2. Define the profile in a file located at `$METRON_HOME/config/zookeeper/profiler.json`.

The following example JSON creates a profile that simply counts the number of messages per `ip_src_addr` during each sampling interval.

```
{
  "profiles": [
    {
      "profile": "test",
      "foreach": "ip_src_addr",
      "init": { "count": 0 },
      "update": { "count": "count + 1" },
      "result": "count"
    }
  ]
}
```

Table 2.1. Profile Elements

Name	Description	
profile	Required	A unique name identifying the profile. The field is treated as a string.
foreach	Required	<p>A separate profile is maintained 'for each' of these. This is effectively the entity that the profile is describing. The field is expected to contain a Stellar expression whose result is the entity name.</p> <p>For example, if <code>ip_src_addr</code> then a separate profile would be maintained for each unique IP source address in the data; 10.0.0.1, 10.0.0.2, etc.</p>
onlyif	Optional	An expression that determines if a message should be applied to the profile. A Stellar expression that returns a Boolean is expected. A message is only applied to a profile if this expression is true. This allows a profile to filter the messages that get applied to it.
groupBy	Optional	<p>One or more Stellar expressions used to group the profile measurements when persisted. This is intended to sort the Profile data to allow for a contiguous scan when accessing subsets of the data.</p> <p>The 'groupBy' expressions can refer to any field within <code>aorg.apache.metron.profiler.ProfileMeasurements</code>. A common use case would be grouping by day of week. This allows a contiguous scan to access all profile data for Mondays only. Using the following definition would achieve this.</p> <pre>"groupBy": ["DAY_OF_WEEK()"]</pre>
init	Optional	One or more expressions executed at the start of a window period. A map is expected where the key is the variable name and the value is a Stellar expression. The map can contain 0 or more variables/expressions. At the start of each window period the expression

		is executed once and stored in a variable with the given name. <pre>"init": { "var1": "0", "var2": "1" }</pre>
update	Required	One or more expressions executed when a message is applied to the profile. A map is expected where the key is the variable name and the value is a Stellar expression. The map can include 0 or more variables/expressions. When each message is applied to the profile, the expression is executed and stored in a variable with the given name. <pre>"update": { "var1": "var1 + 1", "var2": "var2 + 1" }</pre>
result	Required	A Stellar expression that is executed when the window period expires. The expression is expected to summarize the messages that were applied to the profile over the window period. The expression must result in a numeric value such as a Double, Long, Float, Short, or Integer. For more advanced use cases, a profile can generate two types of results. A profile can define one or both of these result types at the same time. <ul style="list-style-type: none"> • profile: A required expression that defines a value that is persisted for later retrieval. • triage: An optional expression that defines values that are accessible within the Threat Triage process.
expires	Optional	A numeric value that defines how many days the profile data is retained. After this time, the data expires and is no longer accessible. If no value is defined, the data does not expire.

3. Upload the profile definition to ZooKeeper:

```
$ cd /$METRON_HOME/
$ bin/zk_load_configs.sh -m PUSH -i config/zookeeper/ -z
$ZOOKEEPER_HOST:2181
```

4. Start the Profiler topology:

```
$ bin/start_profiler_topology.sh
```

5. Ensure that test messages are being sent to the Profiler's input topic in Kafka.

The Profiler will consume messages from the `inputTopic` defined in the Profiler's configuration.

6. Check the HBase table to validate that the Profiler is writing the profile.

Remember that the Profiler is flushing the profile every 15 minutes. You will need to wait at least this long to start seeing profile data in HBase.

```
$ /usr/hdp/current/hbase-client/bin/hbase shell
hbase(main):001:0> count 'profiler'
*** Output Information ***
```

7. Use the Profiler Client to read the profile data.

The following example `PROFILE_GET` command reads data written by the sample profile given above, if 10.0.0.1 is one of the input values for `ip_src_addr`. For more information on using the API client, refer to [Accessing Profiles](#).

```
$ bin/stellar -z $ZOOKEEPER_HOST:2181

[Stellar]>>> PROFILE_GET( "test", "10.0.0.1", PROFILE_FIXED(30, "MINUTES"))
[451, 448]
```

8.3. Configuring the Profiler

The Profiler is installed in the HCP install and runs as an independent Storm topology. The configuration for the Profiler topology is stored in ZooKeeper at `/metron/topology/profiler`. These properties also exist in the default installation of HCP at `$METRON_HOME/config/zookeeper/profiler.json`. The profiler values can be changed on disk and then uploaded to ZooKeeper using `$METRON_HOME/bin/zk_load_configs.sh`.

You might need to work with your Platform Engineer to modify or tune the Profiler values.



Note

The Profiler can persist any serializable object, not just numeric values.

Settings.	Description
profiler.workers	The number of worker processes to create for the topology.
profiler.executors	The number of executors to spawn per component.
profiler.input.topic	The name of the Kafka topic from which to consume data.
profiler.output.topic	The name of the Kafka topic to which profile data is written. Only used with profiles that use the triage result field.
profiler.period.duration	The duration of each profile period. This value should be define along with <code>profiler.period.duration.units</code> .
profiler.period.duration.units	The units used to specify the profile period duration. This value should be defined along with <code>profiler.period.duration</code> .
profiler.ttl	If a message has not been applied to a Profile in this period of time, the Profile will be forgotten and its resources will be cleaned up. This value should be defined along with <code>profiler.ttl.units</code> .

profiler.ttl.units	The units used to specify the <code>profiler.ttl</code> .
profiler.hbase.salt.divisor	A salt is prepended to the row key to help prevent hotspotting. This constant is used to generate the salt. Ideally, this constant should be roughly equal to the number of nodes in the HBase cluster.
profiler.hbase.table	The name of the HBase table that profiles are written to.
profiler.hbase.column.family	The column family used to store profiles.
profiler.hbase.batch	The number of puts that are written in a single batch.
profiler.hbase.flush.interval.seconds	The maximum number of seconds between batch writes to HBase.

8.4. Starting the Profiler

After altering the configuration, start the profiler using the following command:

```
$METRON_HOME/bin/start_profiler_topology.sh
```

8.5. Developing Profiles

Troubleshooting issues when programming against a live stream of data can be difficult. The Stellar REPL (an interactive top level or language shell) is a powerful tool to help work out the kinds of enrichments and transformations that are needed. The Stellar REPL can also be used to help when developing profiles for the Profiler.

Follow these steps in the Stellar REPL to see how it can be used to help create profiles.

1. Take a first pass at defining your profile.

As an example, in the editor copy/paste the basic "Hello, World" profile below.

```
[Stellar]>>> conf := SHELL_EDIT()
[Stellar]>>> conf
{
  "profiles": [
    {
      "profile": "hello-world",
      "onlyif": "exists(ip_src_addr)",
      "foreach": "ip_src_addr",
      "init": { "count": "0" },
      "update": { "count": "count + 1" },
      "result": "count"
    }
  ]
}
```

2. Initialize the Profiler.

```
[Stellar]>>> profiler := PROFILER_INIT(conf)
[Stellar]>>> profiler
org.apache.metron.profiler.StandAloneProfiler@4f8ef473
```

3. Create a message to simulate the type of telemetry that you expect to be profiled.

As an example, in the editor copy/paste the JSON below.

```
[Stellar]>>> message := SHELL_EDIT()
[Stellar]>>> message
{
  "ip_src_addr": "10.0.0.1",
  "protocol": "HTTPS",
  "length": "10",
  "bytes_in": "234"
}
```

4. Apply some telemetry messages to your profiles. The following applies the same message 3 times.

```
[Stellar]>>> PROFILER_APPLY(message, profiler)
org.apache.metron.profiler.StandAloneProfiler@4f8ef473

[Stellar]>>> PROFILER_APPLY(message, profiler)
org.apache.metron.profiler.StandAloneProfiler@4f8ef473

[Stellar]>>> PROFILER_APPLY(message, profiler)
org.apache.metron.profiler.StandAloneProfiler@4f8ef473
```

5. Flush the Profiler to see what has been calculated.

A flush is what occurs at the end of each 15 minute period in the Profiler. The result is a list of profile measurements. Each measurement is a map containing detailed information about the profile data that has been generated.

```
[Stellar]>>> values := PROFILER_FLUSH(profiler)
[Stellar]>>> values
[{period={duration=900000, period=1669628, start=1502665200000, end=
1502666100000},
  profile=hello-world, groups=[], value=3, entity=10.0.0.1}]
```

This profile counts the number of messages by IP source address. Notice that the value is '3' for the entity '10.0.0.1' as we applied 3 messages with an 'ip_src_addr' of '10.0.0.1'. There will always be one measurement for each [profile, entity] pair.

6. If you are unhappy with the data that has been generated, then 'wash, rinse and repeat' this process. After you are satisfied with the data being generated by the profile, then follow the [Getting Started](#) guide to use the profile against your live, streaming data in a Metron cluster.

8.6. Testing

To validate that everything is working, login to the server hosting Metron. We use the Stellar Shell to replicate the execution environment of Stellar running in a Storm topology, like Metron's Parser or Enrichment topology. Replace 'node1:2181' with the URL to a ZooKeeper Broker.

```
[root@node1 0.3.0]# bin/stellar -z $ZOOKEEPER_HOST:2181
Stellar, Go!
Please note that functions are loading lazily in the background and will be
unavailable until loaded fully.
{es.clustername=metron, es.ip=node1, es.port=9300, es.date.format=yyyy.MM.dd.
HH}
```

```
[Stellar]>>> ?PROFILE_GET
Functions loaded, you may refer to functions now...
PROFILE_GET
Description: Retrieves a series of values from a stored profile.

Arguments:
  profile - The name of the profile.
  entity - The name of the entity.
  durationAgo - How long ago should values be retrieved from?
  units - The units of 'durationAgo'.
  groups - Optional - The groups used to sort the profile.

Returns: The profile measurements.

[Stellar]>>> PROFILE_GET('test','192.168.138.158', 1, 'HOURS') [12078.0, 8921.0, 12131.0]
```

The client API call above has retrieved the past hour of the 'test' profile for the entity '192.168.138.158'.