Hortonworks Cybersecurity Package

Administration

(April 13, 2017)

Hortonworks Cybersecurity Package: Administration

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.

Table of Contents

List of Figures

List of Tables

1. HCP Information Roadmap

This roadmap contains additional information on Hortonworks Cybersecurity Package (HCP) and Apache Metron.

Table 1.1. HCP Additional Information

Information type	Resources
Overview	• Apache Metron Website (Source: Apache wiki)
Installing	• Ambari Install Guide (Source: Hortonworks) • Command Line Install Guide (Source: Hortonworks) • Ambari Upgrade Guide (Source: Hortonworks) • Command Line Upgrade Guide (Source: Hortonworks)
Administering	• Apache Metron Documentation (Source: Apache wiki)
Developing	• Community Resources (Source: Apache wiki)
Reference	• About Metron (Source: Apache wiki)
Resources for contributors	• How to Contribute (Source: Apache wiki)
Hortonworks Community Connection	• Hortonworks Community Connection for Metron (Source: Hortonworks)

2. Introduction to Hortonworks CyberSecurity Suite

This guide is intended for Platform Engineers responsible for installing, configuring, and maintaining Hortonworks CyberSecurity Package (HCP) powered by Apache Metron. This guide is divided into three major sections:

- Configuring and Customizing
- Monitor and Management
- Concepts

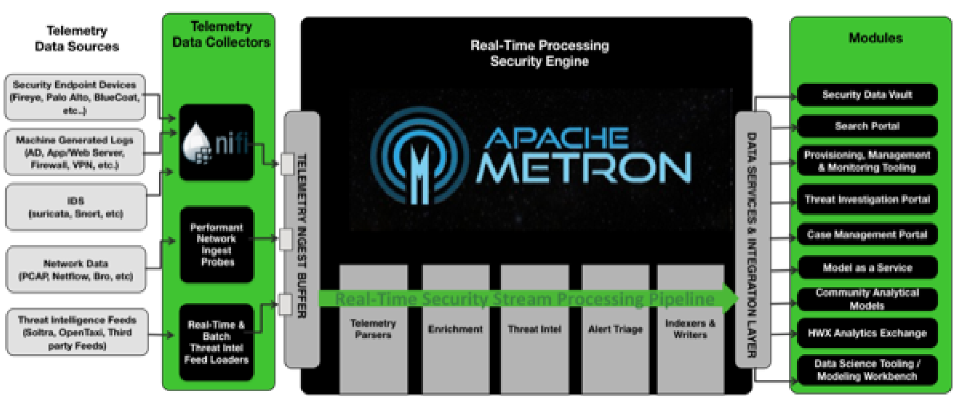
2.1. HCP Architecture

Hortonworks CyberSecurity Package (HCP) is a cybersecurity platform. It consists of the following components:

- Real-time Processing Security Engine [3]
- Telemetry Data Collectors [3]
- Data Services and Integration Layer [3]

Each of these components is described in the following sections.

Figure 2.1. HCP Architecture



The core of the HCP architecture is the Apache Metron real-time processing security engine. The data flow for HCP is performed in real-time and contains the following steps:

1. Information from telemetry data sources is ingested into Kafka topics. (Kafka is the telemetry event buffer.) A Kafka topic is created for every telemetry data source. This information is the raw telemetry data consisting of host logs, firewall logs, emails, and network data.
2. Once the information is ingested into Kafka topics, the data is parsed into a normalized JSON structure that Metron can read.

3. The information is then enriched with asset, geo, threat intelligence information, etc.
4. The information is then indexed, stored, and any resulting alerts are sent to the Metron dashboard as well as telemetry.

2.1.1. Real-time Processing Security Engine

The core of the HCP architecture is the Apache Metron real-time processing security engine. This component provides the ingest buffer to capture the raw events, and, in real time, parses the raw events, enriches the events with relevant contextual information, enriches the events with threat intelligence, and applies available models (such as triaging threats via the Stellar language), then writes the events to a searchable index as well as HDFS for after-the-fact analytics.

2.1.2. Telemetry Data Collectors

Telemetry data collectors push or stream the data source events into Apache Metron. HCP works with NiFi to push the majority of data sources into Apache Metron. For high-volume network data, HCP provides a performant network ingest probe. And for threat intelligence feeds, HCP supports a set of both streaming and batch loaders that enables you to push third-party intelligence feeds into Apache Metron.

2.1.3. Data Services and Integration Layer

This set of HCP modules provides different features for different SOC personas. HCP provides the following three modules for the integration layer:

Security data vault	Stores the data in HDFS.
Search portal	The Metron dashboard.
Provisioning, management, and monitoring tool	HCP provides a Management module that expedites provisioning and managing sensors. Other provisioning, management, and monitoring functions are supported through Ambari.

2.2. Understanding HCP Terminology

This section defines the key terminology associated with cybersecurity, Hadoop, and HCP:

Alert	Alerts provide information about currently security issues, vulnerabilities, and exploits.
Apache Kafka	Apache Kafka is a fast, scalable, durable, fault-tolerant publish-subscribe messaging system, that can be used for stream processing, messaging, website activity tracking, metrics collection and monitoring, log aggregation, and event sourcing.
Apache Storm	Apache Storm enables data-driven, automated activity by providing a real-time, scalable, fault-tolerant, highly available, distributed solution for streaming data.

Apache ZooKeeper	Apache ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
Cybersecurity	The protection of information systems from theft or damage to the hardware, software, and to the information on them, as well as from disruption or misdirection of the services they provide.
Enrichment data source	A data source containing additional information about telemetry ingested by HCP.
Enrichment bolt	The storm bolt that applies the enrichment to the telemetry.
Enrichment data loader	A streaming or a batch loader that stages the data from the enrichment source into HCP so that telemetry can be enriched in real-time with the information from the enrichment source
Forensic Investigator	Collects evidence on breach and attack incidents and prepares legal responses to breaches.
Parser	A storm bolt that transforms telemetry from its native format to a JSON that Metron is able to understand.
Security Data Scientist	Works with security data, performing data munging, visualization, plotting, exploration, feature engineering, model creation. Evaluates and monitors the correctness and currency of existing models
Security Operations Center (SOC)	A centralized unit that deals with cybersecurity issues for an organization by monitoring, assessing, and defending against cybersecurity attacks.
Security Platform Engineer	Installs, configures, and maintains security tools. Performs capacity planning and upgrades. Establishes best practices and reference architecture with respect to provisioning, managing, and using the security tools. Maintains the probes to collect data, load enrichment data, and manage threat feeds.
SOC Analyst	Responsible for monitoring security information and event management (SIEM) tools; searching for and investigating breaches and malware, and reviewing alerts; escalating alerts when appropriate; and following security playbooks.
SOC Investigator	Responsible for investigating more complicated or escalated alerts and breaches, such as Advanced Persistent Threats (APT). Hunts for malware attacks. Removes or quarantines the malware, breach, or infected system.

Telemetry data source	The source of telemetry data which can vary in level from low level (packet capture), intermediate level (deep packet analysis) or very high level (application logs).
Telemetry event	A single event in a stream of telemetry data. This can vary in level from low level (packet capture), intermediate level (deep packet analysis) or very high level (application logs).

3. Configuring and Customizing

One of the key design goals of Hortonworks Cybersecurity Package (HCP) powered by Apache Metron is that it should be easily extensible. HCP comes bundled with several telemetry data sources, enrichment topologies, and threat intelligence feeds. However, you might want to use HCP as a platform and build custom capabilities on top of it.

This chapter describes the following ways you can customize your HCP platform:

3.1. Adding a New Telemetry Data Source

This section describes how you add a new telemetry data source. Before HCP can process the information from a new telemetry data source, you must use one of the telemetry data collectors to ingest the information into the telemetry ingest buffer. Information moves from the data ingest buffer into the Metron real-time processing security engine, where it is parsed, enriched, triaged, and indexed. Finally, certain telemetry events can initiate alerts that can be assessed in the Metron dashboard.

To add a new telemetry data source, perform the following tasks:

1. Streaming Data into HCP [8]
2. Parsing a New Data Source to HCP [13]
3. Verifying That the Events Are Indexed [22]
4. For instructions on how to configure the Metron Dashboard to view the new data source telemetry events, see Hortonworks Cybersecurity User Guide.

The following sections provide steps for each task. You can perform these tasks by using the HCP Management module or CLI. Instructions are provided for both methods.

3.1.1. 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.

- Ensure that the new sensor is installed and set up.
- Ensure that NiFi or another telemetry data collection tool can feed the telemetry data source events into a Kafka topic.
- Determine your requirements.

  For example, you might decide that you need to meet the following requirements:

  - Proxy events from the data source 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, new data source 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 the new data source.

- 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.
  - NIFI_HOST = Host where you will install NIFI.
  - 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. (e.g., 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, HCP-1.1.0.0)

3.1.2. Streaming 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.

Note

Although HCP includes parsers for several data sources (for example, Bro, Snort, and YAF), you must still stream the raw data into HCP through a Kafka topic.

By default, the Snort parser is configured to use ZoneId.systemDefault() for the source `timeZone` for the incoming data and MM/dd/yy-HH:mm:ss.SSSSSS as the default `dateFormat`. Valid timezones are per Java's ZoneId.getAvailableZoneIds(). DateFormats should be valid per the options defined in https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html. Below is a sample configuration with the `dateFormat` and `timeZone` explicitly set in the parser config.

```
"parserConfig": {
"dateFormat" : "MM/dd/yy-HH:mm:ss.SSSSSS",
 "timeZone" : "America/New_York"
```

Note

When you install and configure Snort, you must configure Snort to include the year in the timestamp by modifying the `snort.conf` file as follows:

```
# Configure Snort to show year in timestamps
config show_year
```

This is important for the proper functioning of indexing and analytics.

Depending on the type of data you are streaming into HCP, you can use one of the following methods:

NiFi

This type of streaming method works for most types of data sources. For information on installing NiFi, see the NiFi documentation.

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.

Performant network ingestion probes	This type of streaming method is ideal for streaming high volume packet data. See Setting up pcap to View Your Raw Data for more information.
Real-time and batch threat intelligence feed loaders	This type of streaming method is used for real-time and batch threat intelligence feed loaders. For more information see Using Threat Intelligence Feeds.

3.1.2.1. Creating a NiFi Flow to Stream Events to HCP

This section provides instructions to create a flow to capture events from the new data source and push them into HCP.

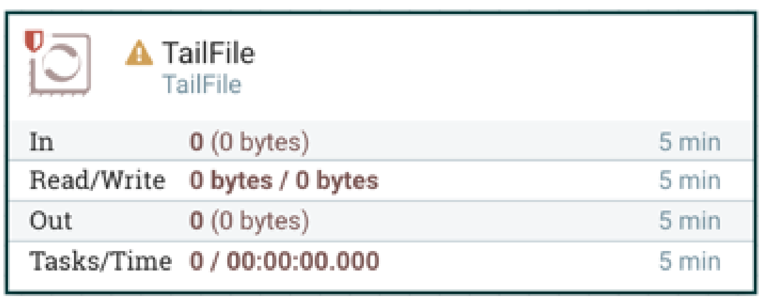
1. Drag the first icon on the toolbar (the processor icon) to your workspace.

   NiFi displays the Add Processor dialog box.

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

   NiFi displays a new TailFile processor.

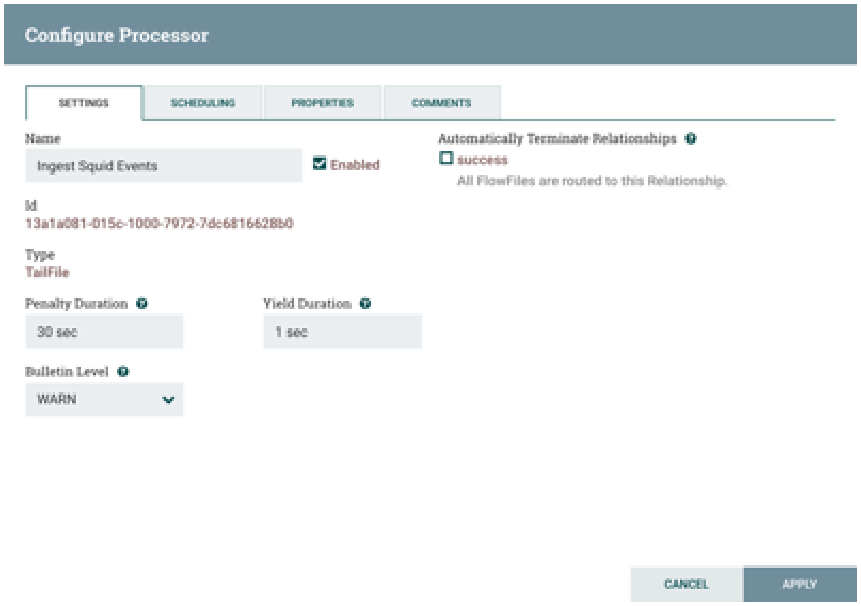
   Figure 3.1. New TailFile Processor

3. Right-click the processor icon and select **Configure** to display the Configure Processor dialog box.

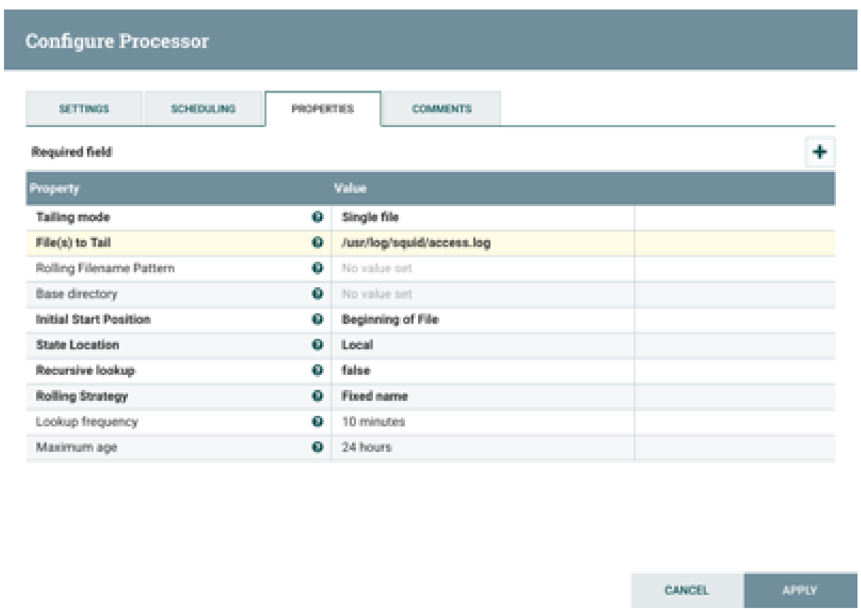
   - In the Settings tab, change the name to `Ingest $DATASOURCE Events`.

Figure 3.2. Configure Processor Dialog Box Settings Tab



- In the **Properties** tab, enter the path to the datasource file in the **Value** column for the **File(s) to Tail** property:

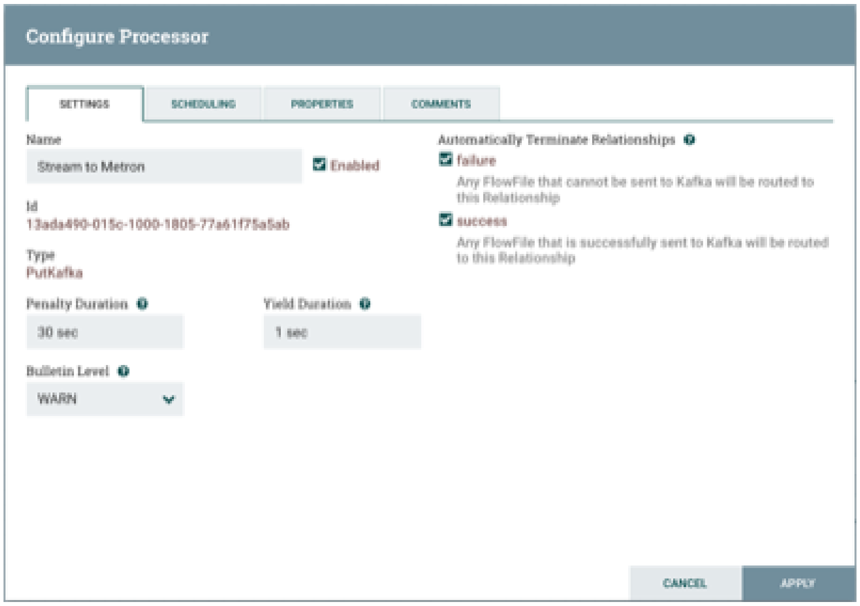
Figure 3.3. Configure Processor Dialog Box Properties Tab



4. Add another processor by dragging the Processor icon to the main window.
5. Select the **PutKafka** type of processor and click **Add**.
6. Right-click the processor and select **Configure**.

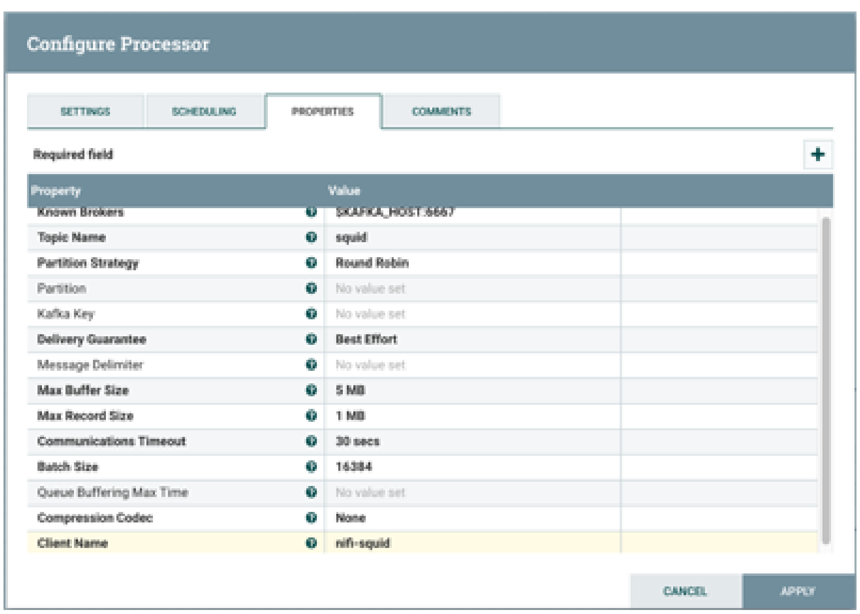
7. In the Settings tab, change the name to **`Stream to Metron`** and then click the relationship check boxes for failure and success.

Figure 3.4. Configure Processor Settings Tab



8. In the Properties tab, set the following three properties:
   - Known Brokers: $KAFKA_HOST:6667
   - Topic Name: $DATAPROCESSOR
   - Client Name: nifi-$DATAPROCESSOR

Figure 3.5. Configure Processor Properties Tab



9. Create a connection by dragging the arrow from the Ingest $DATAPROCESSOR Events processor to the Stream to Metron processor.

   NiFi displays a Create Connection dialog box.

   ### Figure 3.6. Create Connection Dialog Box

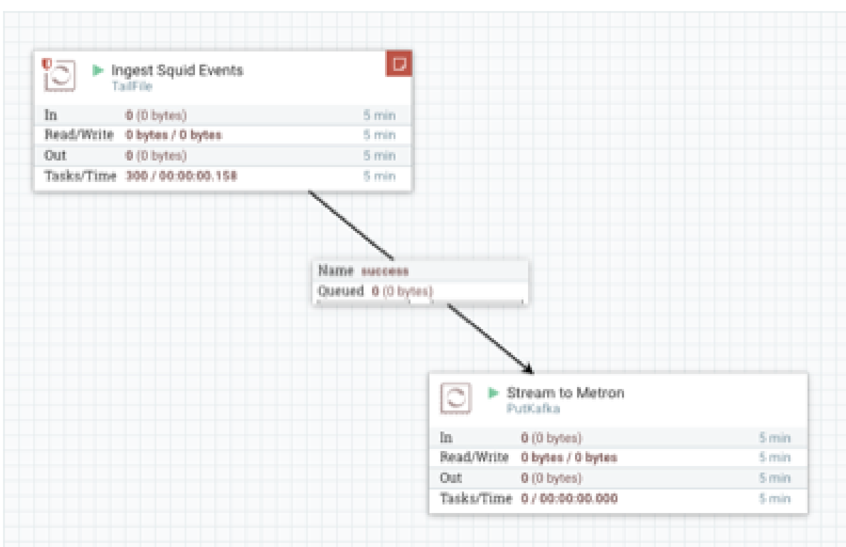
   

10. Click **Add** to accept the default settings for the connection.

11. Press the Shift key and draw a box around both parsers to select the entire flow; then click the play button (green arrow).

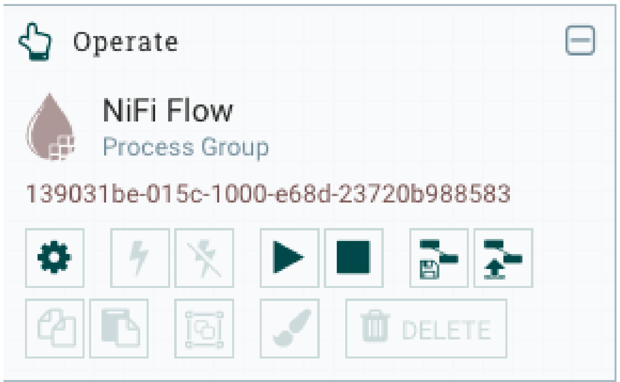
    You should see all of the processor icons turn into green arrows.

    ### Figure 3.7. NiFi Data Flow

12. Click (Start button in the Operate panel.

Figure 3.8. Operate Panel



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

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

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

15. After about five minutes, you should see a new Elastic Search index called $DATAPROCESSOR_index* in the Elastic Admin UI.

For more information about creating a NiFi data flow, see the NiFi documentation.

3.1.3. Parsing a New 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.

You can transform the field output in the JSON messages into information and formats to make the output more useful. For example, you can change the timestamp field output from GMT to your timezone.

You must make two decisions before you parse a new data source:

- Type of parser you will use for your data source

  For more information about which parser to use, see Parsers.

  HCP supports two types of parsers: Java and general purpose:

  - General Purpose - HCP supports two general purpose parsers: Grok and CSV. These parsers are ideal for structured or semi structured logs that are well understood and telemetries with lower volumes of traffic.

  - A Java parser is appropriate for a telemetry type that is complex to parse, with high volumes of traffic.

- How you will parse the new data source

HCP enables you to parse a new data source and transform data fields using the HCP Management module or the command line interface. Both methods are described in the following sections:

- Using the Management Module [14]
- CLI Method [18]

3.1.3.1. Using the Management Module

This section explains how to use the HCP Management module to parse a new data source and transform data fields.

Although HCP supports both Java and general purpose parsers, the following workflow uses the general purpose parser, Grok.

1. Determine the format of the new data source's log entries, so you can parse them:

   a. Look at the different log files that can be created and determine which log file needs to be parsed:

   ```
   sudo su -
   cd /var/log/$NEW_DATASOURCE
   ls
   ```

   The file you want is typically the `access.log`, but your data source might use a different name.

   b. Generate entries for the log that needs to be parsed so that you can see the format of the entries.

   For example:

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

2. Create a Kafka topic for the new data source:

   a. Log in to $KAFKA_HOST as root.

   b. Create a Kafka topic named the same as the new data source:

   ```
   /usr/hdp/current/kafka-broker/bin/kafka-topics.sh
   --zookeeper $ZOOKEEPER_HOST:2181 --create --topic $NEW_DATASOURCE
   --partitions 1 --replication-factor 1
   ```

   c. List all of the Kafka topics, to ensure that the new topic exists:

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

3. Create a Grok statement file that defines the Grok expression for the log type you identified in Step 1.

   Refer to the Grok documentation for additional details.

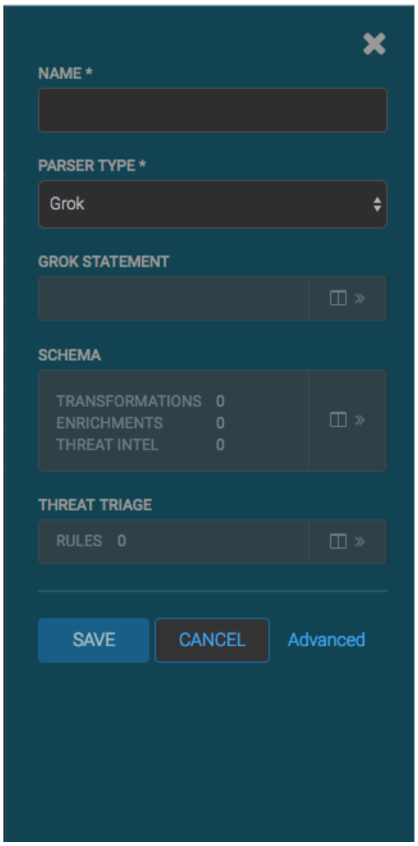
4. Launch the HCP Management module by entering `$METRON_MANAGEMENT_UI_HOST:4200` in a webStreami browser.

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

6. 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 3.9. New Sensor Panel



7. In the **NAME** field, enter the name of the new sensor.

   If a Kafka topic already exists for the sensor name, the module displays a message similar to **Kafka Topic Exists. Emitting**. If no matching Kafka topic is found, the module displays **No Matching Kafka Topic**.

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

   If you chose a Grok parser type and no Kafka type is detected, the module prompts for a Grok Statement.

9. If no Kafka topic exists for your sensor, create a Kafka topic for the sensor.

   a. In the Kafka Topic text box, click the arrow to display the Configure Kafka Topic dialog box

b. Enter the partition and replication factor for the Kafka type associated with the new sensor, and then click Save.

10. Enter a Grok statement for the new parser:

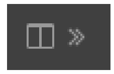
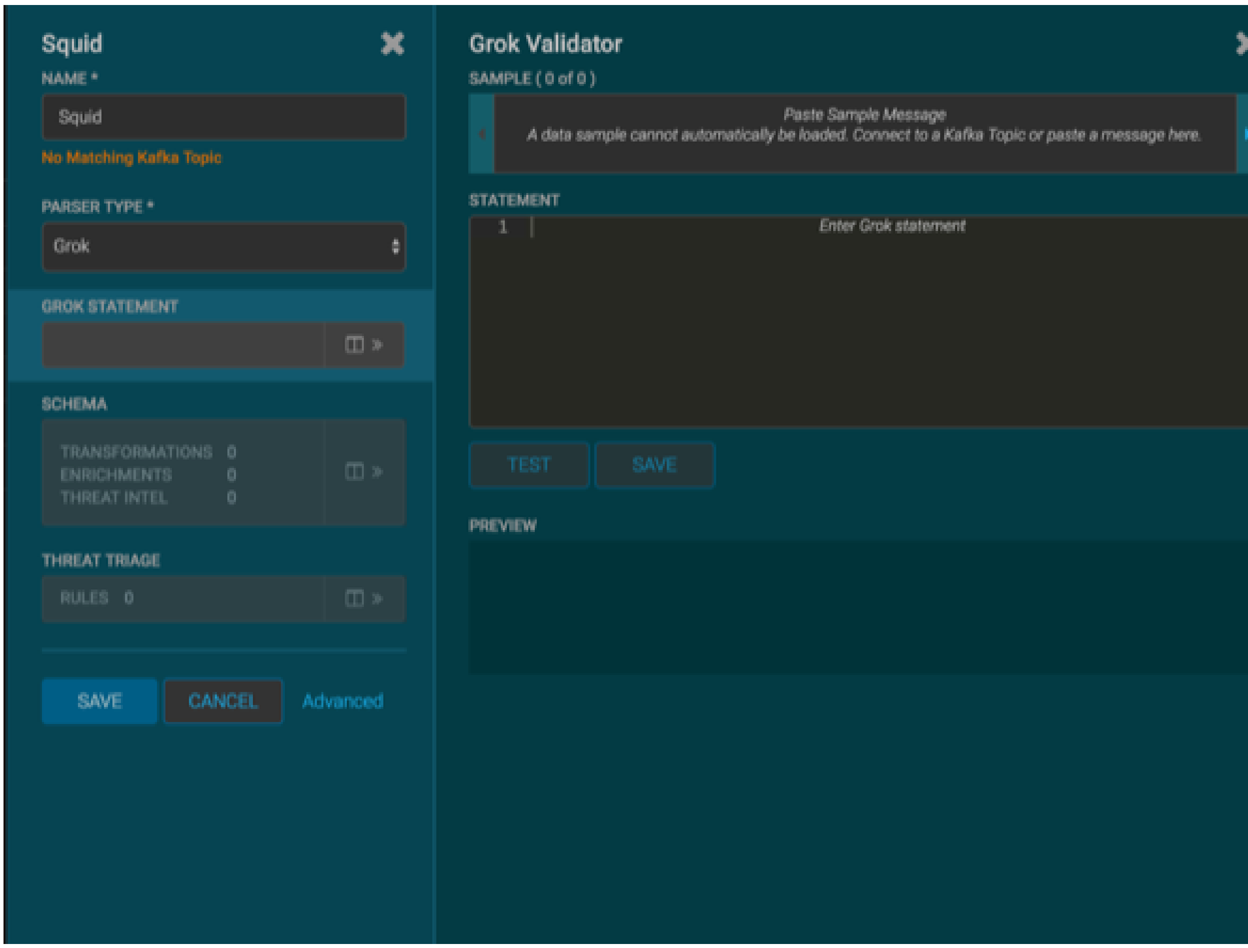
a. In the Grok Statement box, click the  (expand window button) to display the Grok Validator panel.

Figure 3.10. Grok Validator Panel



b. In the **SAMPLE** text field, enter a sample log entry for the data source.

c. In the **STATEMENT** text field, enter the Grok statement you created for the data source, and then click **TEST**.

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

The validator displays the results of the 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.

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.

d. Click **SAVE** to save the Grok statement for the sensor.

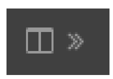
11. Click **SAVE** to save the sensor information and add it to the list of Sensors.

    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.

12. Add your transformation information:

Note

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

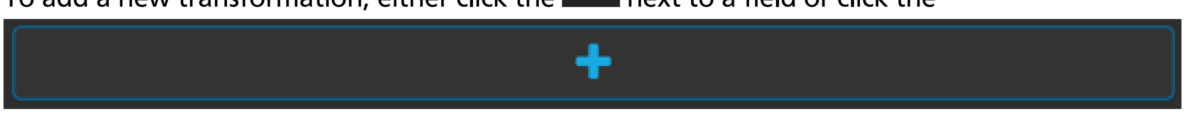
a. In the Schema box, click  (expand window button).

The Management module 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.

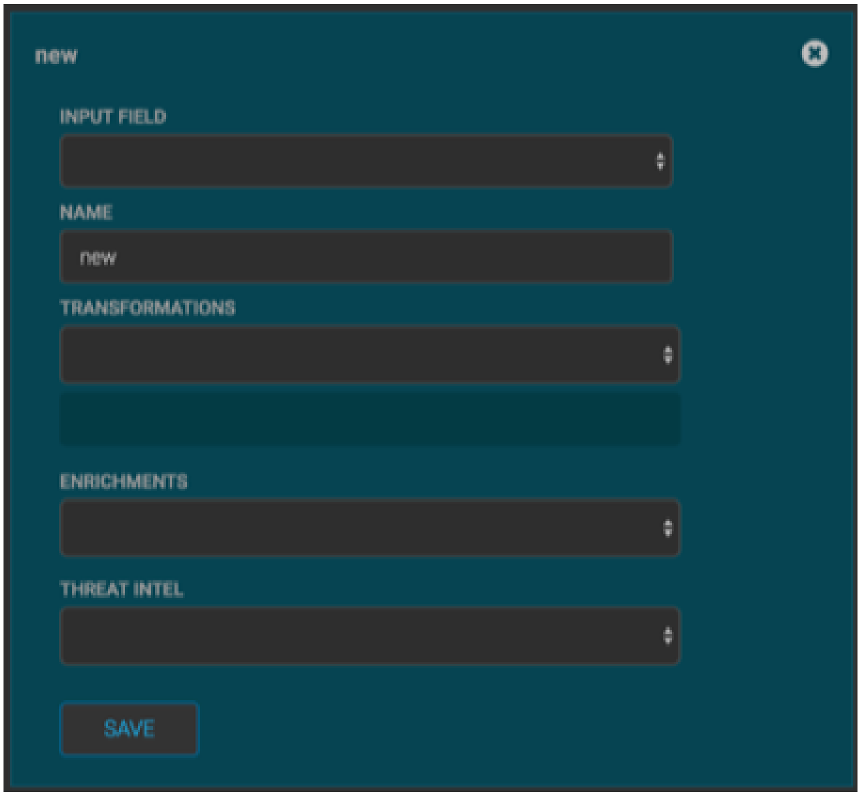
You can use the right and left arrow buttons in the Sample field to view the parsed version of each sample message available from the sensor.

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.

b. 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 3.11. New Schema Information Panel



c. In the dialog box, 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.

d. 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 the "x" next to the field.

e. You can also suppress fields with the transformation feature by clicking (suppress icon).

   This icon prevents the field from being displayed, but it does not remove the field entirely.

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

3.1.3.2. CLI Method

This section shows you how to use the Grok parser to parse a new data source using the CLI.

1. Determine the format of the new data source's log entries, so that you can parse them:

   a. Use ssh to access the host for the new data source.

   b. Look at the different log files that can be created and determine which log file needs to be parsed. This is typically the access.log, but your data source might use a different name.

   ```
   sudo su -
   cd /var/log/$NEW_DATASOURCE
   ls
   ```

   c. Generate entries for the log that needs to be parsed so you can see the format of the entries.

   For example:

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

2. Create a Kafka topic for the new data source:

   a. Log in to $KAFKA_HOST as root.

   b. Create a Kafka topic named the same as the new data source:

   ```
   /usr/hdp/current/kafka-broker/bin/kafka-topics.sh
   --zookeeper $ZOOKEEPER_HOST:2181 --create --topic $NEW_DATASOURCE
   --partitions 1 --replication-factor 1
   ```

   c. List all of the Kafka topics, to ensure that the new topic exists:

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

3. Create a Grok statement.

   a. Define the Grok expression for the log type you identified in Step 1 by creating a Grok statement file.

   Refer to the Grok documentation for additional details.

   b. Validate the Grok pattern to make sure it is valid.

   You can use a tool such as Grok Constructor to validate your Grok pattern.

   c. Save the Grok pattern and load it into Hadoop Distributed File System (HDFS) in a named location:

      i. Create a local file for the new data source:

      ```
      touch /tmp/$DATASOURCE
      ```

      ii. Open $DATASOURCE and add the Grok pattern defined in Step 3b:

      ```
      vi /tmp/$DATASOURCE
      ```

iii. Put the $DATASOURCE file into the HDFS directory where Metron stores its Grok parsers.

Existing Grok parsers that ship with HCP are staged under `/apps/metron/patterns`:

```
su - hdfs
hadoop fs -rmr /apps/metron/patterns/$DATASOURCE
hdfs dfs -put /tmp/$DATASOURCE /apps/metron/patterns/
```

4. Define a parser configuration for the Metron Parsing Topology.

After the Grok pattern is staged in HDFS, you must define a parser configuration for the Metron Parsing Topology. The Metron Parsing Topology (also known as the Normalizing Topology) is designed to take a sensor input (in its native format) and turn it into a Metron JSON Object. For more information about the Metron parsing topology, see Parsers.

a. ssh as root into host with HCP installed.

b. Create a $DATASOURCE parser configuration file at `$METRON_HOME/config/zookeeper/parsers/$DATASOURCE.json`:

For example:

```
{
"parserClassName": "org.apache.metron.parsers.GrokParser",
"sensorTopic": "$DATASOURCE",
"parserConfig": {
"grokPath": "/apps/metron/patterns/$DATASOURCE",
"patternLabel": "$DATASOURCE_DELIMITED",
"timestampField": "timestamp"
},
"fieldTransformations" : [
   {
     "transformation" : "STELLAR"
     ,"output" : [ "full_hostname", "domain_without_subdomains" ]
     ,"config" : {
                  "full_hostname" : "URL_TO_HOST(url)"
                  ,"domain_without_subdomains" :
 "DOMAIN_REMOVE_SUBDOMAINS(full_hostname)"
                  }
    }
   ]
}
```

Where:

parserClassName	The name of the parser's class that is in the jar file.
sensorTopic	The Kafka topic on which the telemetry is being streamed.
parserConfig	The configuration file.
grokPath	The path for the Grok statement.

patternLabel	The top-level pattern of the Grok file.
fieldTransformations	An array of complex objects representing the transformations to be done on the message generated from the parser before writing out to the Kafka topic. In this example, the Grok parser is designed to extract the URL, but the only information that you need is the domain (or even the domain without subdomains). To obtain this, you can use the Stellar Field Transformation (under the fieldTransformations element). The Stellar Field Transformation allows you to use the Stellar DSL (Domain Specific Language) to define extra transformations to be performed on the messages flowing through the topology. For more information on using the fieldTransformations element in the parser configuration, see Parsers.

c. Use the following script to upload configurations to Apache ZooKeeper:

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

Note

You might receive the following warning messages when you execute the previous command. You can safely ignore these warning messages.

```
log4j:WARN No appenders could be found for logger (org.apache.
curator.framework.imps.CuratorFrameworkImpl).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.
html#noconfig for more info.
```

5. Deploy the new parser topology to the cluster:

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

   b. Deploy the new parser topology:

```
$METRON_HOME/bin/start_parser_topology.sh -k $KAFKA_HOST:6667 -z
 $ZOOKEEPER_HOST:2181 -s $DATASOURCE
```

   c. Use the Apache Storm UI to ensure that the new topology is listed and that it has no errors.

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

3.1.4. Verifying 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 $DATASOURCE_index_[timestamp] and the document type is $DATASOURCE_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:

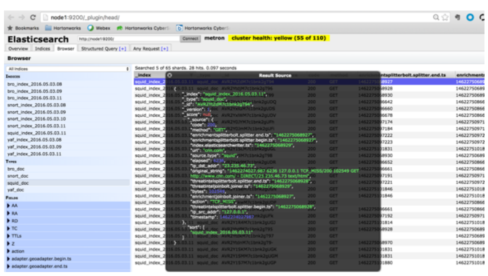
   ```
   /usr/share/elasticsearch/bin/plugin install mobz/elasticsearch-head/1.x
   ```

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

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

   You should see something like the following:

   Figure 3.12. Elasticsearch With Index Information

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

3.2. 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). Examples of enrichments are GEO where an external IP address is enriched with GeoIP information (lat/long coordinates + City/State/Country) and HOST enrichment where an IP gets enriched with Host details (for example, IP corresponds to Host X which is part of a web server farm for an e-commerce application). This information makes the data more useful and relevant, which assists the SOC analyst and SOC investigator in researching alerts. Threat intelligence is another type of enrichment. For information about threat intelligence see Using Threat Intelligence.

HCP provides the following 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.

One of the features of the enrichment topology is that it groups messages together by the HBase key. An advantage of grouping messages together is that whenever you execute a Stellar function, you can add a caching layer, thus decreasing the need to do a call to HBase for every event.

Prior to enabling an enrichment capability within HCP, the enrichment store (which for HCP is primarily HBase) must be loaded with enrichment data. Enrichment data can either be bulk loaded from the local file system, HDFS, or be streamed into the enrichment store via the parser framework. 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.

Each enrichment bolt can enrich a specific field/tag within a Metron message. When a bolt recognizes that it is able to enrich a field, it reaches into the enrichment store, pulls out the enrichment, and tags the message with the enrichment. The enrichment is then stored within the bolt's in-memory cache. HCP uses the underlying Storm routing capabilities to make sure that similar enrichment values are sent to the appropriate bolts that already have these values cached in-memory.

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

- Asset
- GeoIP
- User

To configure an enrichment source, complete the following steps:

- Prerequisites [7]
- Bulk Loading Enrichment Information [24]
  - Configuring an Extractor Configuration File [26]
  - Configuring Element-to-Enrichment Mapping [29]
  - Running the Enrichment Loader [29]
  - Mapping Fields to HBase Enrichments [30]
- Streaming Enrichment Information [34]

For more information about the Metron enrichment framework, see Enrichment Framework.

3.2.1. Bulk Loading Enrichment Information

Enrichment data can either be bulk loaded from HDFS or be streamed into enrichment store via pluggable loading framework. This section provides the steps to bulk load enrichment data.

You can bulk load enrichment information from the following sources:

- Taxii Loader
- HDFS via MapReduce
- Flat File Ingestion

Taxii Loader

The shell script `$METRON_HOME/bin/threatintel_taxii_load.sh` can be used to poll a Taxii server for STIX documents and ingest them into HBase.

It is quite common for this Taxii server to be an aggregation server such as Soltra Edge.

In addition to the Enrichment and Extractor configs described in the following sections, this loader requires a configuration file describing the connection information to the Taxii server. The following is an example of a configuration file:

```
{
   "endpoint" : "http://localhost:8282/taxii-discovery-service"
  ,"type" : "DISCOVER"
  ,"collection" : "guest.Abuse_ch"
  ,"table" : "threat_intel"
  ,"columnFamily" : "cf"
  ,"allowedIndicatorTypes" : [ "domainname:FQDN", "address:IPV_4_ADDR" ]
}
```

where:

endpoint	The URL of the endpoint.
type	`POLL` or `DISCOVER` depending on the endpoint.
collection	The Taxii collection to ingest.
table	The HBase table to import into.
columnFamily	The column family to import into.
allowedIndicatorTypes	An array of acceptable threat intelligence types (see the "Enrichment Type Name" column of the Stix table above for the possibilities).

The parameters for the utility are as follows:

Short Code	Long Code	Is Required?	Description
-h		No	Generate the help screen/ set of options

Short Code	Long Code	Is Required?	Description
-e	–extractor_config	Yes	JSON document describing the extractor for this input data source
-c	–taxii_connection_config	Yes	The JSON config file to configure the connection
-p	–time_between_polls	No	The time between polling the Taxii server in milliseconds. (default: 1 hour)
-b	–begin_time	No	Start time to poll the Taxii server (all data from that point will be gathered in the first pull). The format for the date is yyyy-MM-dd HH:mm:ss
-l	–log4j	No	The Log4j properties to load
-n	–enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

N

HDFS via MapReduce

The shell script `$METRON_HOME/bin/threatintel_bulk_load.sh` will kick off a MapReduce job to load data staged in HDFS into an HBase table.

Note

Despite what the naming might suggest, this utility works for enrichment as well as threat intel due to the underlying infrastructure being the same.

The parameters for the utility are as follows:

Short Code	Long Code	Is 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	–table	Yes	The HBase table to import into
-f	–column_family	Yes	The HBase table column family to import into
-i	–input	Yes	The input data location on HDFS
-n	–enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

CSV File

The shell script `$METRON_HOME/bin/flatfile_loader.sh` will read data from local disk and load the enrichment or threat intel data into an HBase table.

One special thing to note here is that 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 `BY_LINE` and `org.apache.metron.dataloads.extractor.inputformat.WholeFileFormat`

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.

The parameters for the utility are as follows:

Short Code	Long Code	Is 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 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.

3.2.1.1. 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.
2. Determine the schema of the enrichment source.
3. Create an extractor configuration file called `extractor_config_temp.json` 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:

   `value_transform`
   Transforms fields defined in the `columns` mapping with Stellar transformations. New keys introduced in the transform are added to the key metadata. For example:

   ```
   "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 `indicator` 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 `domain`, 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. To access properties that reside in the global configuration file, provide a ZooKeeper quorum via the `zk_quorum` property. If the global configuration looks like `"global_property" : "metron-ftw"`, enter the following to expand the `value_transform`:

```
"value_transform" : {
    "domain" : "DOMAIN_REMOVE_TLD(domain)",
     "a-new-prop" : "global_property"
 },
```

   The resulting value data will look like the following:

Indicator	Type	Value
google	top_domains	{ "rank" : "1", "domain" : "google", "a-new-prop" : "metron-ftw" }
yahoo	top_domains	{ "rank" : "2", "domain" : "yahoo", "a-new-prop" : "metron-ftw" }

6. Remove any non-ASCII invisible characters that might have been included when you cut 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. It is up the user to keep this configuration file for future use, if needed.

3.2.1.2. Configuring Element-to-Enrichment Mapping

Configure which element of a tuple should be enriched with which enrichment type.

This configuration is stored in ZooKeeper.

1. Log in as root user to the host that has Metron installed.
2. Cut and paste the following into a file called `enrichment_config_temp.json`, being sure to customize $ZOOKEEPER_HOST and $DATASOURCE to your specific values, where $DATASOURCE refers to the name of the datasource that is used to bulk load the enrichment:

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

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

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

3.2.1.3. 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 and store the enrichment configuration in ZooKeeper.

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

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

   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 a client for your particular data source to execute requests.

3.2.1.4. 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 perform this step using either the HCP Management module or the CLI. Both of these methods are described in the following subsections.

3.2.1.4.1. Management Module Method

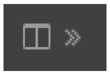
Now that you have parsed the data source, you can refine the parser output in three ways:

- Transformations
- Enrichments
- Threat Intel

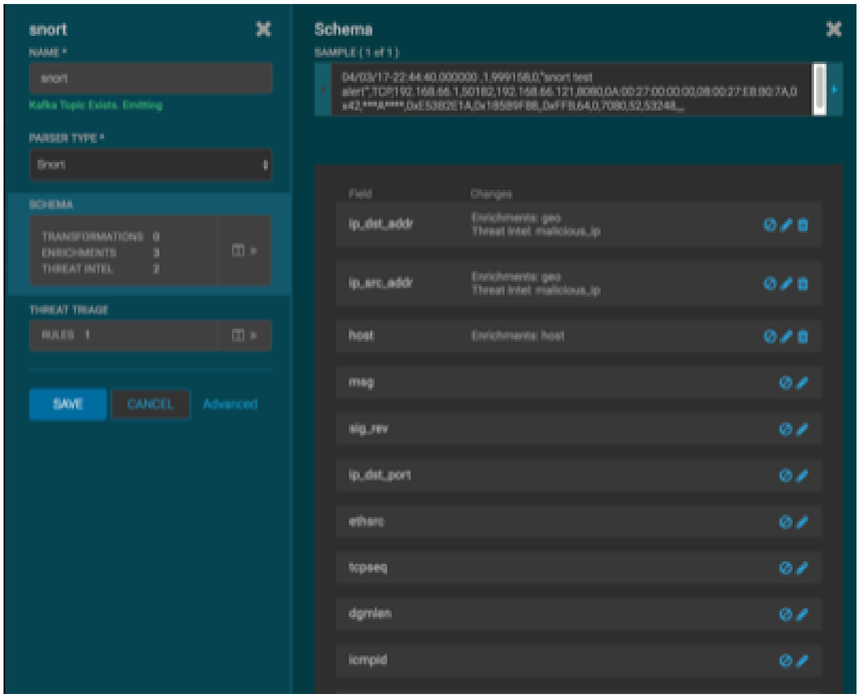
Each of the parser outputs is added or modified in the **Schema** field. To modify any of the parser outputs, complete the following steps:

> **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.

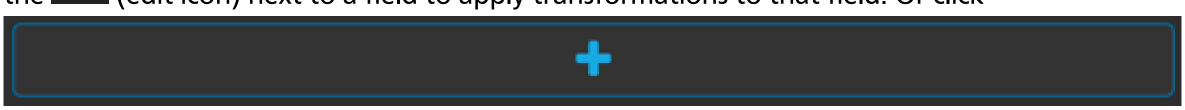
   The Management Module displays the sensor panel for the new sensor.
3. In the Schema box, click  (expand window button).

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



The 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.

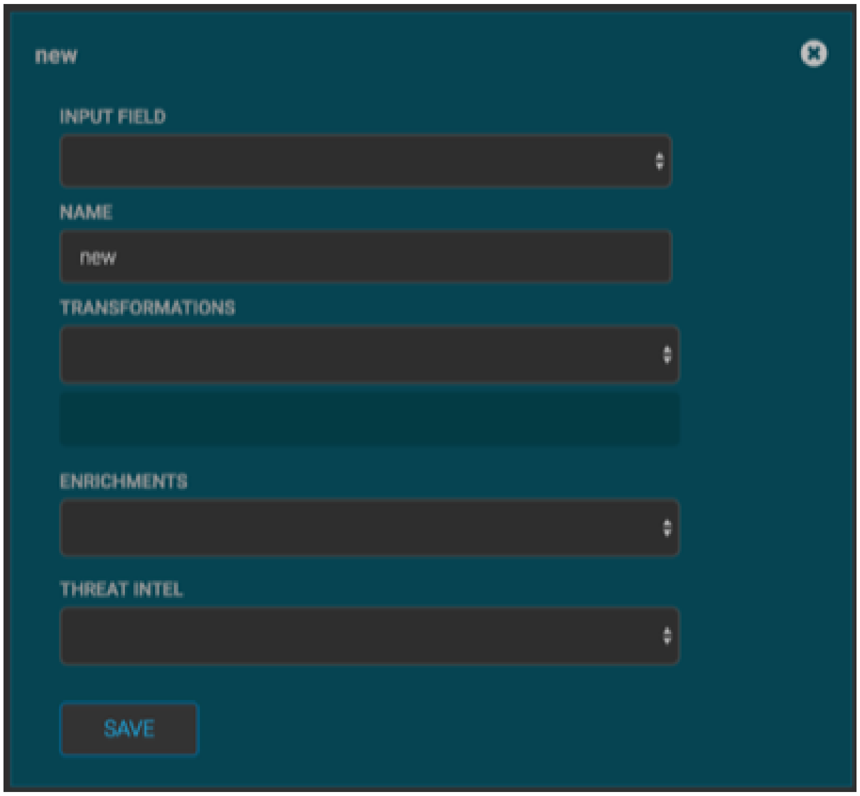
You can use the right and left arrow buttons in the Sample field to view the parsed version of each sample message available from the sensor.

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 3.13. New Schema Information Panel



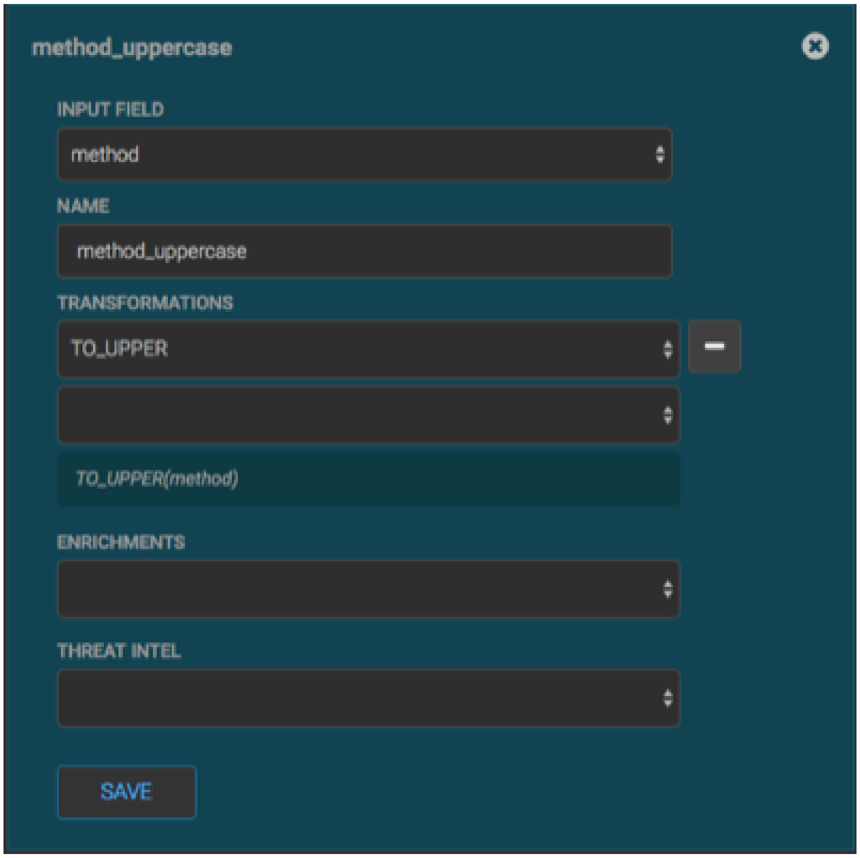
5. In the dialog box, enter the name of the new field in the **NAME** field, choose an input field from the **INPUT FIELD** box, and choose your transformation from the **TRANSFORMATIONS** field or enrichment from the **ENRICHMENTS** field.

   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 3.14. Populated New Schema Information Panel



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.

3.2.1.4.2. CLI Method

1. Edit the new data source enrichment configuration at `$METRON_HOME/config/zookeeper/enrichments/$DATASOURCE` to associate the `ip_src_addr` with the user enrichment.

   For example:

```
{
  "index" : "squid",
  "batchSize" : 1,
  "enrichment" : {
    "fieldMap" : {
      "hbaseEnrichment" : [ "ip_src_addr" ]
    },
```

```
    "fieldToTypeMap" : {
      "ip_src_addr" : [ "whois" ]
    },
    "config" : { }
  },
  "threatIntel" : {
    "fieldMap" : { },
    "fieldToTypeMap" : { },
    "config" : { },
    "triageConfig" : {
      "riskLevelRules" : { },
      "aggregator" : "MAX",
      "aggregationConfig" : { }
    }
  },
  "configuration" : { }
}
```

2. Push this configuration to ZooKeeper:

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

After you have finished enriching the telemetry events, ensure that the enriched data is displaying on the Metron dashboard. For instructions on adding a new telemetry data source to the Metron Dashboard, see Adding a New Data Source.

3.2.2. Streaming Enrichment Information

Streaming intelligence feeds are incorporated slightly differently than data from a flat CSV file. This section describes how to define a streaming source.

1. Because we are defining a streaming source, we need to define a parser topology to handle the streaming data. In order to do that, we will need to create a file in `$METRONHOME/zookeeper/parsers/user.json`.

   Define a parser topology to handle the streaming data:

```
touch $METRONHOME/zookeeper/parsers/user.json
```

2. Populate the file the parser topology definition. For example:

```
{
 "parserClassName" : "org.apache.metron.parsers.csv.CSVParser"
 ,"writerClassName" : "org.apache.metron.writer.hbase.
SimpleHbaseEnrichmentWriter"
 ,"sensorTopic":"user"
 ,"parserConfig":
 {
    "shew.table" : "enrichment"
   ,"shew.cf" : "t"
   ,"shew.keyColumns" : "ip"
   ,"shew.enrichmentType" : "user"
   ,"columns" : {
      "user" : 0
     ,"ip" : 1
                }
 }
```

```
}
```

This file fully defines the input structure and how that data can be used in enrichment.

3. Push this configuration file to ZooKeeper and start the parser topology by running the following:

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --zookeeper
 $ZOOKEEPER_HOST:2181 --replication-factor 1 --partitions 1 --topic user
$METRON_HOME/bin/zk_load_configs.sh -m PUSH -z $ZOOKEEPER_HOST:2181
 $METRON_HOME/zookeeper
$METRON_HOME/bin/start_parser_topology.sh -s user -z $ZOOKEEPER_HOST:2181 -k
 $KAKFA_HOST:6667
```

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

4. Edit the new data source enrichment configuration at `$METRON_HOME/config/zookeeper/enrichments/$DATASOURCE` to associate the `ip_src_addr` with the user enrichment.

For example:

```
{
  "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. Push this configuration to ZooKeeper:

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

3.3. 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 stored in ZooKeeper and on disk at `$METRON_HOME/config/zookeeper/indexing`.

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`).

3.3.1. 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:

```
{
}
```

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

3.3.2. Specifying Index Parameters

You can to specify the parameters for the writers rather than using the default values. You can use either the Management Module or the CLI to specify writer parameters:

- Specifying index parameters using the Management Module
- Specifying index parameters using the CLI

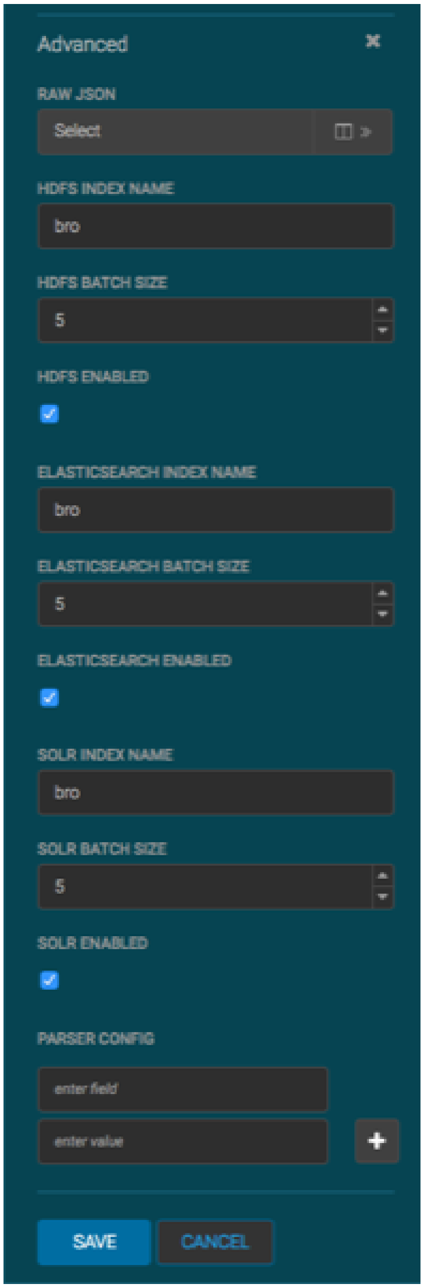
3.3.2.1. Specifying Index Parameters using the Management Module

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 3.15. Management Module Advanced Panel



3. Enter index configuration information for your sensor.

4. Click **Save** to save your changes and push your configuration to ZooKeeper.

3.3.2.2. Specifying Index Parameters Using the CLI

To specify the parameters for the writers rather than using the default values, you can use the following syntax in the Indexing Configuration file, located at `$METRON_HOME/config/zookeeper/indexing`.

1. Create the Indexing Configuration file at `$METRON_HOME/config/zookeeper/indexing`.

   ```
   touch /$METRON_HOME/config/zookeeper/indexing/index.json
   ```

2. Populate the `$sensor_name.json` file with index configuration information for each of your sensors, using syntax similar to the following:

   ```
   {
      "elasticsearch": {
         "index": "foo",
         "batchSize" : 100,
         "enabled" : true
       },
      "hdfs": {
         "index": "foo",
         "batchSize": 1,
         "enabled" : true
       }
   }
   ```

   This syntax specifies the following parameter values:

   - Elasticsearch writer or index
     - index name of "foo"
     - batch size of 100
     - enabled
   - HDFS writer or index
     - index name of "foo"
     - batch size of 1
     - enabled

3. Push the configuration to ZooKeeper:

   ```
    /usr/metron/$METRON_VERSION/bin/zk_load_configs.sh --mode PUSH -i /usr/
   metron/$METRON_VERSION/config/zookeeper -z $ZOOKEEPER_HOST:2181
   ```

3.3.3. Turning Off HDFS Writer

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

Create or modify

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

```
      "batchSize": 100,
      "enabled" : false
    }
}
```

3.4. Using Threat Intelligence Feeds

The threat intelligence topology takes a normalized JSON message and cross references it against threat intelligence, tags it with alerts if appropriate, runs the results against the scoring component of machine learning models where appropriate, and stores the telemetry in a data store. This section provides the following steps for using threat intelligence feeds:

- Prerequisites [40]
- Bulk Loading Enrichment Information [24]
  - Configuring an Extractor Configuration File [43]
  - Configure Mapping for the Intelligence Feed [45]
  - Running the Threat Intel Loader [46]
  - Mapping Fields to HBase Threat Intel [47]
- Creating a Streaming Threat Intel Feed Source [50]

Threat intelligence topologies perform the following tasks:

- Mark messages as threats based on data in external data stores
- Mark threat alerts with a numeric triage level based on a set of Stellar rules

3.4.1. Prerequisites

Perform the following tasks before configuring threat intelligence feeds:

1. Choose your threat intelligence sources.
2. **Recommended but not required:** Install a threat intelligence feed aggregator, such as SoltraEdge.

3.4.2. Bulk Loading Threat Intelligence Information

You can bulk load threat intelligence information from the following sources:

- Taxii Loader
- HDFS via MapReduce
- Flat File Ingestion (CVS)

Taxii Loader

The shell script `$METRON_HOME/bin/threatintel_taxii_load.sh` can be used to poll a Taxii server for STIX documents and ingest them into HBase.

It is quite common for this Taxii server to be an aggregation server such as Soltra Edge.

In addition to the Enrichment and Extractor configs described in the following sections, this loader requires a configuration file describing the connection information to the Taxii server. The following is an example of a configuration file:

```
{
   "endpoint" : "http://localhost:8282/taxii-discovery-service"
  ,"type" : "DISCOVER"
  ,"collection" : "guest.Abuse_ch"
  ,"table" : "threat_intel"
  ,"columnFamily" : "cf"
  ,"allowedIndicatorTypes" : [ "domainname:FQDN", "address:IPV_4_ADDR" ]
}
```

where:

endpoint	The URL of the endpoint
type	`POLL` or `DISCOVER` depending on the endpoint.
collection	The Taxii collection to ingest
table	The HBase table to import into
columnFamily	The column family to import into
allowedIndicatorTypes	an array of acceptable threat intel types (see the "Enrichment Type Name" column of the Stix table above for the possibilities).

The parameters for the utility are as follows:

Short Code	Long Code	Is 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
-c	–taxii_connection_config	Yes	The JSON config file to configure the connection
-p	–time_between_polls	No	The time between polling the Taxii server in milliseconds. (default: 1 hour)
-b	–begin_time	No	Start time to poll the Taxii server (all data from that point will be gathered in the first pull). The format for the date is yyyy-MM-dd HH:mm:ss
-l	–log4j	No	The Log4j Properties to load
-n	–enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

HDFS via MapReduce

The shell script `$METRON_HOME/bin/threatintel_bulk_load.sh` will kick off a MapReduce job to load data staged in HDFS into an HBase table.

Note

Despite what the naming might suggest, this utility works for enrichment as well as threat intel due to the underlying infrastructure being the same.

The parameters for the utility are as follows:

Short Code	Long Code	Is 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	–table	Yes	The HBase table to import into
-f	–column_family	Yes	The HBase table column family to import into
-i	–input	Yes	The input data location on HDFS
-n	–enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

CSV File

The shell script `$METRON_HOME/bin/flatfile_loader.sh` will read data from local disk and load the enrichment or threat intel data into an HBase table.

One special thing to note here is that 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 `BY_LINE` and `org.apache.metron.dataloads.extractor.inputformat.WholeFile`

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.

The parameters for the utility are as follows:

Short Code	Long Code	Is 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

Short Code	Long Code	Is Required?	Description
-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 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.

3.4.3. Configuring an Extractor Configuration File

After you have a threat intelligence feed source, you must configure an extractor configuration file that describes the source.

1. Log in as root user to the host on which Metron is installed.
2. Create a file called `extractor_config_temp.json` and add the following content:

```
{
"config" : {
   "columns" : {
      "domain" : 0
      ,"source" : 1
   }
   ,"indicator_column" : "domain"
   ,"type" : "zeusList"
   ,"separator" : ","
  }
  ,"extractor" : "CSV"
}
```

3. 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:

`value_transform`

Transforms fields defined in the `columns` mapping with Stellar transformations. New keys introduced in the transform are added to the key metadata. For example:

```
"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 `indicator` 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 `domain`, 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" }

4. To access properties that reside in the global configuration file, provide a ZooKeeper quorum via the `zk_quorum` property. If the global configuration looks like

`"global_property" : "metron-ftw"`, enter the following to expand the `value_transform`:

```
"value_transform" : {
    "domain" : "DOMAIN_REMOVE_TLD(domain)",
     "a-new-prop" : "global_property"
 },
```

The resulting value data will look like the following:

Indicator	Type	Value
google	top_domains	{ "rank" : "1", "domain" : "google", "a-new-prop" : "metron-ftw" }
yahoo	top_domains	{ "rank" : "2", "domain" : "yahoo", "a-new-prop" : "metron-ftw" }

5. Remove any non-ASCII characters:

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

6. Configure the mapping for the element-to-threat intelligence feed.

   This step configures which element of a tuple to cross-reference with which threat intelligence feed. This configuration is stored in ZooKeeper.

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

   b. Cut and paste the following file into a file called `enrichment_config_temp.json`":

```
{
     "zkQuorum" : "$ZOOKEEPER_HOST:2181"
    ,"sensorToFieldList" : {
     "$DATASOURCE" : {
          "type" : "THREAT_INTEL"
         ,"fieldToEnrichmentTypes" : {
               "domain_without_subdomains" : [ "zeusList" ]
          }
     }
   }
}
```

   c. Remove the non-ASCII characters:

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

3.4.4. Configure Mapping for the Intelligence Feed

1. Configure the mapping for the element-to-threat intelligence feed.

   This step configures which element of a tuple to cross-reference with which threat intelligence feed. This configuration is stored in ZooKeeper.

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

b. Cut and paste the following file into a file called `enrichment_config_temp.json`":

```
{
     "zkQuorum" : "$ZOOKEEPER_HOST:2181"
    ,"sensorToFieldList" : {
     "$DATASOURCE" : {
          "type" : "THREAT_INTEL"
         ,"fieldToEnrichmentTypes" : {
               "domain_without_subdomains" : [ "zeusList" ]
          }
     }
   }
}
```

c. Remove the non-ASCII characters:

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

3.4.5. Running the Threat Intel Loader

After you have defined the threat intelligence source, threat intelligence extractor, and threat intelligence mapping configuration, run the loader to move the data from the threat intelligence source to the Metron threat intelligence store and to store the enrichment configuration in ZooKeeper.

1. Log in to $HOST_WITH_ENRICHMENT_TAG as root.
2. Run the loader:

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

   This command adds the threat intelligence data into 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 `threatintel`.
3. Verify that the logs were properly ingested into HBase:

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

   You should see a configuration for the sensor that looks something like the following:

Figure 3.16. Threat Intel Configuration

4. Generate some data to populate the Metron Dashboard.

3.4.6. Mapping Fields to HBase Threat Intel

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

You can perform this step using either the HCP Management module or the CLI. Both of these methods are described in the following subsections.

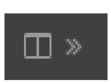
3.4.6.1. Management Module Method

Defining the threat intel topology is very similar to defining the transformation and enrichment topology.

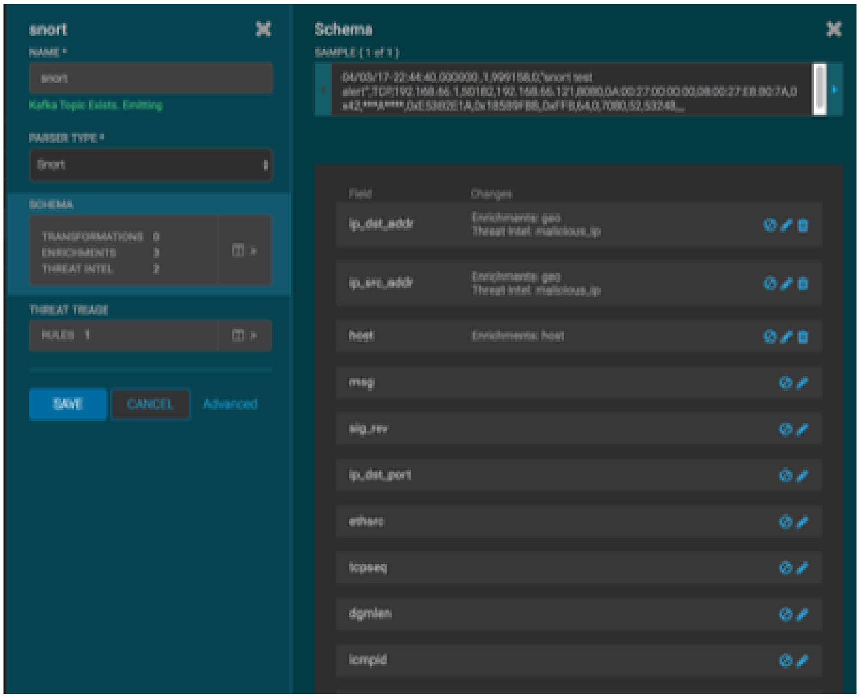
Each of the parser outputs is added or modified in the **Schema** field. To modify any of the parser outputs, complete the following steps:

> **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.

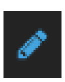
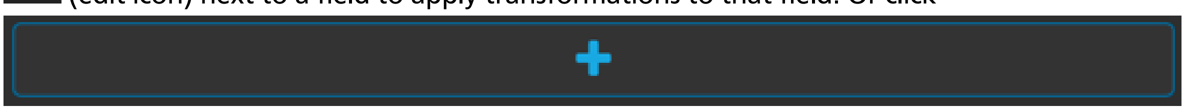
   The Management module displays the sensor panel for the new sensor.
3. In the Schema box, click  (expand window button).

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



The 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 threat intel against these parsed messages.

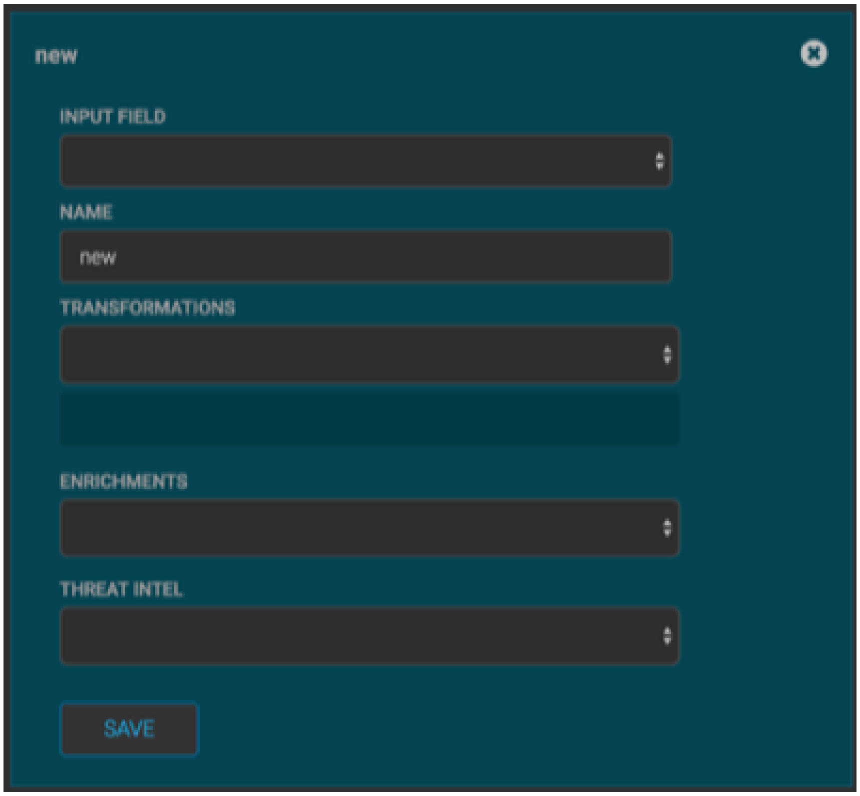
You can use the right and left arrow buttons in the Sample field to view the parsed version of each sample message available from the sensor.

4. You can apply threat intel 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 choose to create and transform a new field, rather than transforming an existing field.

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

Figure 3.17. New Schema Information Panel



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 **THREAT INTEL** field .

6. Click **SAVE** to save your changes.

7. You can suppress fields from 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.

3.4.6.2. CLI Method

1. Edit the new data source threat intel configuration at `$METRON_HOME/config/zookeeper/enrichments/$DATASOURCE` to associate the `ip_src_addr` with the user enrichment.

   For example:

```
{
  "index" : "squid",
  "batchSize" : 1,
  "enrichment" : {
```

```
    "fieldMap" : {
      "hbaseEnrichment" : [ "ip_src_addr" ]
    },
    "fieldToTypeMap" : {
      "ip_src_addr" : [ "whois" ]
    },
    "config" : { }
  },
  "threatIntel" : {
    "fieldMap" : { },
    "fieldToTypeMap" : { },
    "config" : { },
    "triageConfig" : {
      "riskLevelRules" : { },
      "aggregator" : "MAX",
      "aggregationConfig" : { }
    }
  },
  "configuration" : { }
}
```

2. Push this configuration to ZooKeeper:

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

After you have finished enriching the telemetry events, ensure that the enriched data is displaying on the Metron dashboard. For instructions on adding a new telemetry data source to the Metron Dashboard, see Adding a New Data Source.

3.4.7. Creating a Streaming Threat Intel Feed Source

Streaming intelligence feeds and are incorporated slightly differently than data from a flat CSV file. This section describes how to define a streaming source.

Because we are defining a streaming source, we need to define a parser topology to handle the streaming data. In order to do that, we will need to create a file in $METRONHOME/zookeeper/parsers/user.json.

1. Define a parser topology to handle the streaming data:

```
touch $METRONHOME/zookeeper/parsers/user.json
```

2. Populate the file the parser topology definition. For example:

```
{
 "parserClassName" : "org.apache.metron.parsers.csv.CSVParser"
 ,"writerClassName" : "org.apache.metron.writer.hbase.
SimpleHbaseEnrichmentWriter"
 ,"sensorTopic":"user"
 ,"parserConfig":
 {
    "shew.table" : "enrichment"
   ,"shew.cf" : "t"
   ,"shew.keyColumns" : "ip"
   ,"shew.enrichmentType" : "user"
   ,"columns" : {
```

```
      "user" : 0
     ,"ip" : 1
                }
 }
}
```

This file fully defines the input structure and how that data can be used in enrichment.

3. Push this configuration file to ZooKeeper and start the parser topology by running the following:

   ```
   /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --zookeeper
    $ZOOKEEPER_HOST:2181 --replication-factor 1 --partitions 1 --topic user
   $METRON_HOME/bin/zk_load_configs.sh -m PUSH -z $ZOOKEEPER_HOST:2181
    $METRON_HOME/zookeeper
   $METRON_HOME/bin/start_parser_topology.sh -s user -z $ZOOKEEPER_HOST:2181 -k
    $KAFKA_HOST:6667
   ```

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

4. Edit the new data source enrichment configuration at `$METRON_HOME/config/zookeeper/enrichments/$DATASOURCE` to associate the `ip_src_addr` with the user enrichment.

   For example:

   ```
   {
     "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. Push this configuration to ZooKeeper:

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

3.5. 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:

3.5.1. Prerequisites

Before you can prioritize a threat intelligence enrichment, you must ensure that the enrichment is working properly

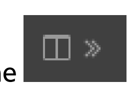
3.5.2. Performing Threat Triage Using the Management Module

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.

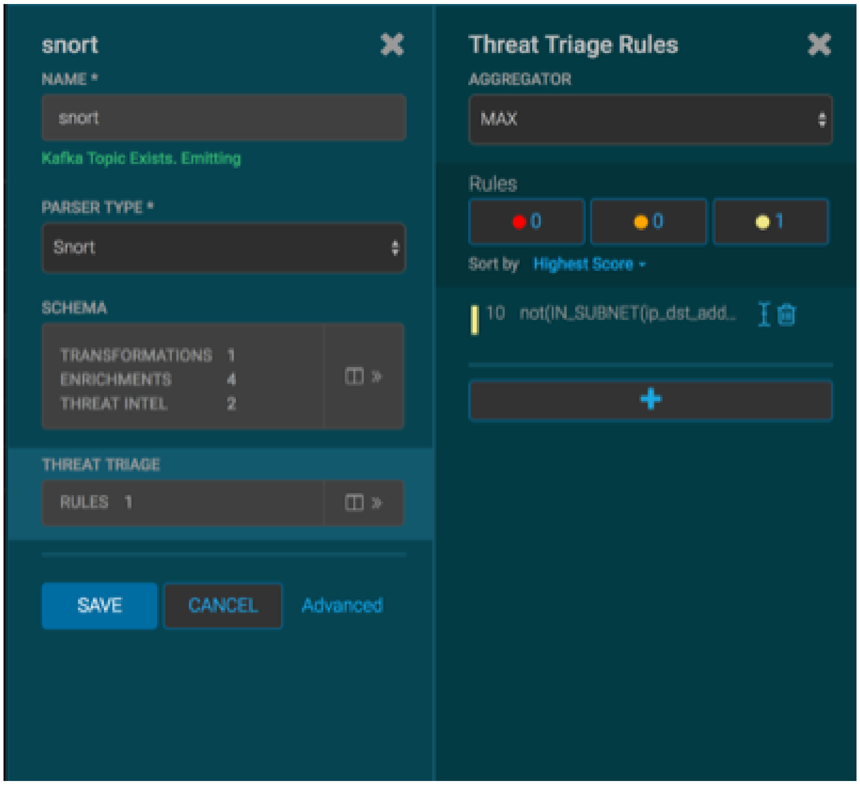
Rule 2 If the URL ends with neither .com nor .net, then imagine that you want to receive an alert score of 10.

To create these rules, complete the following steps:

1. On the sensor panel, in the Threat Triage field, click the  icon (expand window).

   The module displays the Threat Triage Rules panel.

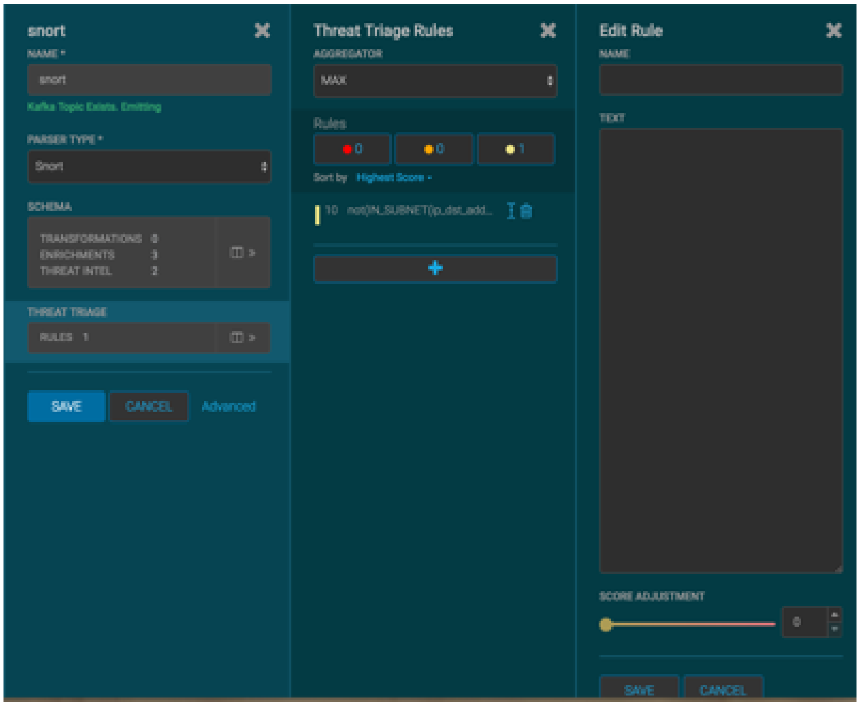
Figure 3.18. Threat Triage Rules Panel



2. Click the + button to add a rule.

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

Figure 3.19. Edit Rule Panel



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

   For example:

   ```
   Exists(IsAlert)
   ```

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

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

7. 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.

8. 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.

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

3.5.3. Performing Threat Triage Using the CLI

The perform threat triage using the CLI, you must complete the following steps:

- Creating the Threat Triage Rule Configuration [55]
- Uploading the Threat Triage Configuration to ZooKeeper [57]
- Viewing Triaged or Scored Alerts [58]

3.5.3.1. Creating the Threat Triage Rule Configuration

The goal of threat triage is to prioritize the alerts that pose the greatest threat and need urgent attention. To create a threat triage rule configuration, you must first define your rules. Each rule has a predicate to determine whether or not the rule applies. The threat score from each applied rule is aggregated into a single threat triage score that is used to prioritize high risk threats.

Following are some examples:

Rule 1 — If a threat intelligence enrichment type zeusList 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.

Rule 3 — For each message, the triage score is the maximum score across all conditions.

These example rules become the following example configuration:

```
"triageConfig" : {
   "riskLevelRules" : [
{
"name" : "zeusList is alerted"
"comment" : "Threat intelligence enrichment type zeusList is alerted."
"rule": "exists(threatintels.hbaseThreatIntel.domain_without_subdomains.
zeusList)"
"score" : 5
}
{
"name" : "Does not end with .com or .net"
"comment" : "The URL ends with neither .com nor .net."
"rule": "not(ENDS_WITH(domain_without_subdomains, '.com') or
 ENDS_WITH(domain_without_subdomains, '.net'))" : 10
"score" : 10
}
]
      ,"aggregator" : "MAX"
```

```
       ,”aggregationConfig” : { }
}
```

You can use the 'reason' field to generate a message explaining why a rule fired. One or more rules may fire when triaging a threat. Having detailed, contextual information about the environment when a rule fired can greatly assist actioning the alert. For example:

Rule 1 For hostname, the value exceeds threshold of value-threshold, receive an alert score of 10.

This example rule becomes the following example configuration:

```
“triageConfig” : {
   “riskLevelRules” : [
      {
      “name” : “Abnormal Value"
      “comment” : “The value has exceeded the threshold",
      "reason": "FORMAT('For '%s' the value '%d' exceeds threshold of '%d',
 hostname, value, value_threshold)"
      “rule”: "value > value_threshold”,
      “score” : 10
      }
   ],
   “aggregator” : “MAX”,
   ”aggregationConfig” : { }
}
```

If the value threshold is exceeded, Threat Triage will generate a message similar to the following:

```
"threat.triage.score": 10.0,
"threat.triage.rules.0.name": "Abnormal Value",
"threat.triage.rules.0.comment": "The value has exceeded the threshold",
"threat.triage.rules.0.score": 10.0,
"threat.triage.rules.0.reason": "For '10.0.0.1' the value '101' exceeds
 threshold of '42'"
```

where

riskLevelRules — This is a list of rules (represented as Stellar expressions) associated with scores with optional names and comments.

- name — The name of the threat triage rule.
- comment — A comment describing the rule.
- reason — An optional Stellar expression that when executed results in a custom message describing why the rule fired.
- rule — The rule, represented as a Stellar statement.
- score — Associated threat triage score for the rule.

aggregator — An aggregation function that takes all non-zero scores representing the matching queries from `riskLevelRules` and aggregates them into a single score.

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.

3.5.3.2. Uploading the Threat Triage Configuration to ZooKeeper

To apply this example triage configuration, you must modify the configuration for the new sensor in the enrichment topology.

1. Log in as root user to the host on which Metron is installed.
2. Modify `$METRON_HOME/config/zookeeper/sensors/$DATASOURCE.json`.

   Because the configuration in ZooKeeper might be out of sync with the configuration on disk, ensure that they are in sync by downloading the ZooKeeper configuration first:

   ```
   $METRON_HOME/bin/zk_load_configs.sh -m PULL -z $ZOOKEEPER_HOST:2181 -f -o
    $METRON_HOME/config/zookeeper
   ```

3. Validate that the enrichment configuration for the data source exists:

   ```
   cat $METRON_HOME/config/zookeeper/enrichments/$DATASOURCE.json
   ```

4. In the `$METRON_HOME/config/zookeeper/enrichments/$DATASOURCE.json` file, add the following to the `triageConfig` section in the threat intelligence section.

   For example:

   ```
   "threatIntel" : {
       "fieldMap" : {
         "hbaseThreatIntel" : [ "domain_without_subdomains" ]
       },
       "fieldToTypeMap" : {
         "domain_without_subdomains" : [ "zeusList" ]
       },
       "config" : { },
       "triageConfig" : {
         "riskLevelRules" : {
            "exists(threatintels.hbaseThreatIntel.domain_without_subdomains.
   zeusList)" : 5
                 , "not(ENDS_WITH(domain_without_subdomains, '.com') or
    ENDS_WITH(domain_without_subdomains, '.net'))" : 10
                              }
           ,"aggregator" : "MAX"
           ,"aggregationConfig" : { }
                         }
   ```

```
                }
  }
```

5. Ensure that the aggregator field indicates MAX.

6. Push the configuration back to ZooKeeper:

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

3.5.3.3. 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.

An alert in the indexing topic in Kafka will 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) #
##########################################################
```

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 3.20. Investigation Module Triaged Alert Panel

3.6. Setting Up Global Configuration

Global configurations are applied to all data sources as opposed to other configurations that are applied to a specific sensor. In other words, every message from every sensor is validated against global configuration rules. The format of the global enrichment is a JSON string-to-object map that is stored in ZooKeeper. For example:

```
{
  "es.clustername": "metron",
  "es.ip": "node1",
  "es.port": "9300",
  "es.date.format": "yyyy.MM.dd.HH",
  "fieldValidations" : [
              {
                "input" : [ "ip_src_addr", "ip_dst_addr" ],
                "validation" : "IP",
                "config" : {
                    "type" : "IPV4"
                           }
```

```
                }
                        ]
}
```

where

es.ip
: A single or collection of elastic search master nodes. They may be specified via the widely accepted `hostname:port` syntax. If a port is not specified, then a separate global property `es.port` is required:

  Example: `es.ip` : [“10.0.0.1:1234”, “10.0.0.2:1234”]

  Example: `es.ip` : “10.0.0.1” (thus requiring `es.port` to be specified as well)

  Example: `es.ip` : “10.0.0.1:1234” (thus not requiring `es.port` to be specified)

es.port
: The port of the elastic search master node. This is not strictly required if the port is specified in the `es.ip global` property as described above. It is expected that this be an integer or a string representation of an integer.

  Example: `es.port` : “1234"

  Example: `es.port` : 1234

es.clustername
: The elastic search cluster name to which you want to write.

  Example: `es.clustername` : “metron” (providing your ES cluster is configured to have metron be a valid cluster name)

es.date.format
: We shard the indices first by sensor and then by date. This provides the granularity time-wise that we shard.

  Example: `es.date.format` : “yyyy.MM.dd.HH” (this would shard by hour creating, for example, a Bro shard of bro_2016.01.01.01, bro_2016.01.01.02, etc.)

  Example: `es.date.format` : “yyyy.MM.dd” (this would shard by day, creating, for example, a Bro shard of bro_2016.01.01, bro_2016.01.02, etc.)

3.6.1. Parser Field Validations

Inside the global configuration, there is a validation framework that enables you to construct validation rules that cross all sensors. This is done in the form of validation plugins where assertions about fields or whole messages can be made.

The format for this is a `fieldValidations` field inside of global config. This is associated with an array of field validation objects structured using the following fields:

input
: An array of input fields or a single field. If this is omitted, then the whole messages is passed to the validator.

config	A String to Object map for validation configuration. This is optional if the validation function requires no configuration.
validation	The validation function to be used. This is one of the following:

STELLAR	Execute a Stellar Language statement. Expects the query string in the `condition` field of the config.
IP	Validates that the input fields are an IP address. By default, if no configuration is set, it assumes IPV4, but you can specify the type by passing in type with either `IPV6` or `IPV4` or by passing in a list `[IPV4,IPV6]` in which case the input(s) will be validated against both.
DOMAIN	Validates that the fields are all domains.
EMAIL	Validates that the fields are all email addresses.
URL	Validates that the fields are all URLs.
DATE	Validates that the fields are a date. Expects `format` in the config.
INTEGER	Validates that the fields are an integer. String representation of an integer is allowed.
REGEX_MATCH	Validates that the fields match a regex. Expects `pattern` in the config.
NOT_EMPTY	Validates that the fields exist and are not empty (after trimming.)

For example, the following validation validates that `ip_src_addr` is an IPv4 address via the IP validator:

```
"fieldValidations" : [
              {
                "input" : [ "ip_src_addr" ],
                "validation" : "IP",
                "config" : {
                    "type" : "IPV4"
                           }
              }
                       ]
```

You can also create a validation using Stellar. The following validation uses Stellar to validate the same thing as the previous example:

```
"fieldValidations" : [
              {
                "validation" : "STELLAR",
                "config" : {
                    "condition" : "IS_IP(ip_src_addr, 'IPV4')"
                           }
              }
                       ]
```

3.7. Configuring the Profiler

A profile describes the behavior of an entity on a network. This feature is typically used by a data scientist and you should coordinate with the data scientist determine if they will need your assistance with customizing the Profiler values. For more information on configuring the Profiler, see Configuring the Profiler.

3.7.1. 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 `$METRON_HOME/config/zookeeper/profiler.json`. The values can be changed on disk and then uploaded to ZooKeeper using `$METRON_HOME/bin/zk_load_configs.sh`.

Note

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

Table 3.1. Profiler Properties

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.period.duration	The duration of each profile period. This value should be define along with `profiler.period.duration.units`.
profiler.period.duration.units	The units used to specify the profile period duration. This value should be defined along with `profiler.period.duration`.
profiler.hbase.salt.divisor	A salt is prepended to the row key to help prevent hotspotting. This constant is used to generate the sale. 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.

3.8. Creating 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. This can be achieved 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'.

   - 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 is expected to contain 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.

```
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"
        }
      }
    }
  }
}
```

2. 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`.

3. 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 $SEARCH_HOST:9200/$DATSOURCE*
```

4. 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*
```

3.9. Configuring the Metron Dashboard to View the New Data Source Telemetry Events

After HCP is configured to parse, index, and persist telemetry events and NiFi is pushing data to HCP, you can view streaming telemetry data in the Metron Dashboard. See HCP User Guide for information about configuring the Metron Dashboard.

3.10. Setting up pcap to View Your Raw Data

The pcap data source creates a Storm topology that can rapidly ingest raw data directly into HDFS from Kafka. As a result, you can store all of your cybersecurity data in its raw form in HDFS and review or query it at a later date. HCP supports two pcap components:

- The pycapa tool aimed at low-volume packet capture

  Pycapa is a open-source Python-based probe created by Cisco.

- The Data Plane Development Kit (DPDK) based tool aimed at high-volume packet capture.

  DPDK is a set of data plane libraries and network interface controller drivers for fast packet processing.

The rest of this chapter provides or points to instructions for setting up pycapa and DPDK and starting pcap:

- Setting up pycapa [63]
- Setting up DPDK [64]
- Starting pcap [64]

3.10.1. Setting up pycapa

You can set up pycapa by completing the following steps. This installation assumes the following environment variables:

```
PYCAPA_HOME=/opt/pycapa
PYTHON27_HOME =/opt/rh/python27/root
```

1. Install the following packages:

```
epel-release
```

```
centos-release-scl
"@Development tools"
python27
python27-scldevel
python27-python-virtualenv
libpcap-devel
libselinux-python
```

For example:

```
yum -y install epel-release centos-release-scl
yum -y install "@Development tools" python27 python27-scldevel python27-
python-virtualenv libpcap-devel libselinux-python
```

2. Set up the following directory:

```
mkdir $PYCAPA_HOME && chmod 755 $PYCAPA_HOME
```

3. Create the following virtual environment:

```
export LD_LIBRARY_PATH="/opt/rh/python27/root/usr/lib64"
${PYTHON27_HOME}/usr/bin/virtualenv pycapa-venv
```

4. Copy `incubator-metron/metron-sensors/pycapa` from the Metron source tree into `$PYCAPA_HOME` on the node on which you would like to install pycapa.

5. Build pycapa:

```
cd ${PYCAPA_HOME}/pycapa
activate the virtualenv
source ${PYCAPA_HOME}/pycapa-venv/bin/activate
pip install -r requirements.txt
python setup.py install
```

6. Start the pycapa packet capture producer:

```
cd ${PYCAPA_HOME}/pycapa-venv/bin
pycapa --producer --topic pcap -i $ETH_INTERFACE -k $KAFKA_HOST:6667
```

3.10.2. Setting up DPDK

To set up DPDK, see http://dpdk.org/doc/guides/linux_gsg/index.html.

3.10.3. Starting pcap

To start pcap, HCP provides a utility script. This script takes no arguments and is very simple to run. Complete the following steps to start pcap:

1. Log into the host on which you are running Metron.

2. If you are running HCP on an Ambari-managed cluster, perform the following steps. If you are running a VM or a cluster that is not managed by Ambari, skip to Step 3.

   a. Update the `$METRON_HOME/config/pcap.properties` by changing `kafka.zk` to the appropriate server.

      You can retrieve the appropriate server information from Ambari in **Kafka service** > **Configs** > **Kafka Broker** > **zookeeper.connect**.

b. On the HDFS host, create `/apps/metron/pcap`, change its ownership to metron:hadoop, and change its permissions to 775.

```
hdfs dfs -mkdir /apps/metron/pcap
hdfs dfs -chown metron:hadoop /apps/metron/pcap
hdfs dfs -chmod 755 /apps/metron/pcap
```

c. Create a Metron user's home directory on HDFS and change its ownership to the Metron user.

```
hdfs dfs -mkdir /user/metron
hdfs dfs -chown metron:hadoop /user/metron
hdfs dfs -chmod 755 /user/metron
```

d. Create a pcap topic in Kafka.

i. Switch to metron user:

```
su - metron
```

ii. Create a Kafka topic named pcap:

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh \
--zookeeper $ZOOKEEPER_HOST:2181 \
--create \
--topic pcap \
--partitions 1 \
--replication-factor 1
```

iii. List all of the Kafka topics, to ensure that the new pcap topic exists:

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

3. Start the pcap topology:

```
$METRON_HOME/bin/start_pcap_topology.sh
```

If HCP is installed on an Ambari-managed cluster, proceed the previous command with `su - metron`.

4. Check the Storm topology to ensure that packets are being captured.

After Storm has captured a sufficient number of packets, you can check to ensure it is creating files on HDFS:

```
hadoop fs -ls /apps/metron/pcap
```

3.11. Troubleshooting Parsers

This section provides some troubleshooting solutions for parser issues.

3.11.1. Storm is Not Receiving Data From a New Data Source

1. Ensure that your Grok parser statement is valid.

   You can use GrokConstructor to test your parser statement.

   If you need to modify your Grok parser statement, you must kill the topology for your new data source in the Storm UI and then resubmit your data source.

   a. Log into HOST $HOST_WITH_ENRICHMENT_TAG as root.

   b. Deploy the new parser topology:

   ```
   $METRON_HOME/bin/start_parser_topology.sh -k $KAFKA_HOST:6667 -z
    $ZOOKEEPER_HOST:2181 -s $DATASOURCE
   ```

   c. Go to the Storm UI. You should now see the new topology. Ensure that the topology has no errors.

2. Ensure that the Kafka topic you created for your new data source is receiving data.

3. Check your NiFi configuration to ensure that data is flowing between the Kafka topic for your new data source into HCP.

3.11.2. Determining Which Events Are Not Being Processed

Events that are not processed end up in a dead letter queue. There are two types of events. One, where the event could not be parsed at all. Two, where the event was parsed, but failed validation

4. Monitor and Management

Hortonworks Cybersecurity Package (HCP) powered by Apache Metron provides a number of options for monitoring and managing your system. Before you perform any of these monitoring and management tasks, we suggest that you become familiar with HCP data throughput by referring to Understanding Throughput.

The rest of this chapter provides detailed instructions on performing the following monitoring and management tasks:

4.1. Understanding Throughput

The data flow for HCP is performed in real-time and contains the following steps:

1. Information from telemetry data sources is ingested into Kafka files through the telemetry event buffer. This information is the raw telemetry data consisting of host logs, firewall logs, emails, and network data. Depending on the type of data you are streaming into HCP, you can use one of the following telemetry data collectors to ingest the data:

NiFi	This type of streaming method works for most types of telemetry data sources. See the NiFi documentation for more information,
Performant network ingestion probes	This type of streaming method is ideal for streaming high volume packet data. See Using pcap to View Your Raw Data for more information.
Real-time and batch threat intelligence feed loaders	This type of streaming method is used for real-time and batch threat intelligence feed loaders.

2. Once the information is ingested into Kafka files, the data is parsed into a normalized JSON structure that HCP can read. This information is parsed using a Java or general purpose parser and then it is uploaded to ZooKeeper. A Kafka file containing the parser information is created for every telemetry data source.

3. The information is then enriched with asset, geo, and threat intelligence information.

4. The information is then indexed, stored, and any resulting alerts are sent to the Metron dashboard.

4.2. Updating ZooKeeper

ZooKeeper configurations should be stored on disk in the following structure starting at `$METRON_HOME/config/zookeeper`:

global.json	The global config
sensors	The subdirectory containing sensor enrichment configuration JSON (for example, snort.json, bro.json)

By default, this directory as deployed by the ansible infrastructure is at `$METRON_HOME/config/zookeeper`.

While the configs are stored on disk, they must be loaded into ZooKeeper to be used. To this end, there is a utility program to assist in this called `$METRON_HOME/bin/zk_load_config.sh`.

This has the following options:

`-f,--force`	Force operation
`-h,--help`	Generate Help screen
`-i,--input_dir <DIR>`	The input directory containing the configuration files named like "$source.json"
`-m,--mode <MODE>`	The mode of operation: DUMP, PULL, PUSH
`-o,--output_dir <DIR>`	The output directory which will store the JSON configuration from ZooKeeper
`-z,--zk_quorum <host:port,[host:port]*>`	ZooKeeper Quorum URL (zk1:port,zk2:port,...)

Usage examples:

- To dump the existing configs from ZooKeeper on the single node vagrant machine:

```
$METRON_HOME/bin/zk_load_configs.sh -z node1:2181 -m DUMP
```

- To push the configs into ZooKeeper on the single node vagrant machine:

```
$METRON_HOME/bin/zk_load_configs.sh -z node1:2181 -m PUSH -i
$METRON_HOME/config/zookeeper
```

- To pull the configs from ZooKeeper to the single node vagrant machine disk:

```
$METRON_HOME/bin/zk_load_configs.sh -z node1:2181 -m PULL -o
$METRON_HOME/config/zookeeper -f
```

4.3. Managing Sensors

You can manage your sensors and associated topologies using either the HCP Management module or the Storm UI. The following procedures use the HCP Management module to manage sensors. For information on using Storm to manage sensors, see the Storm documentation.

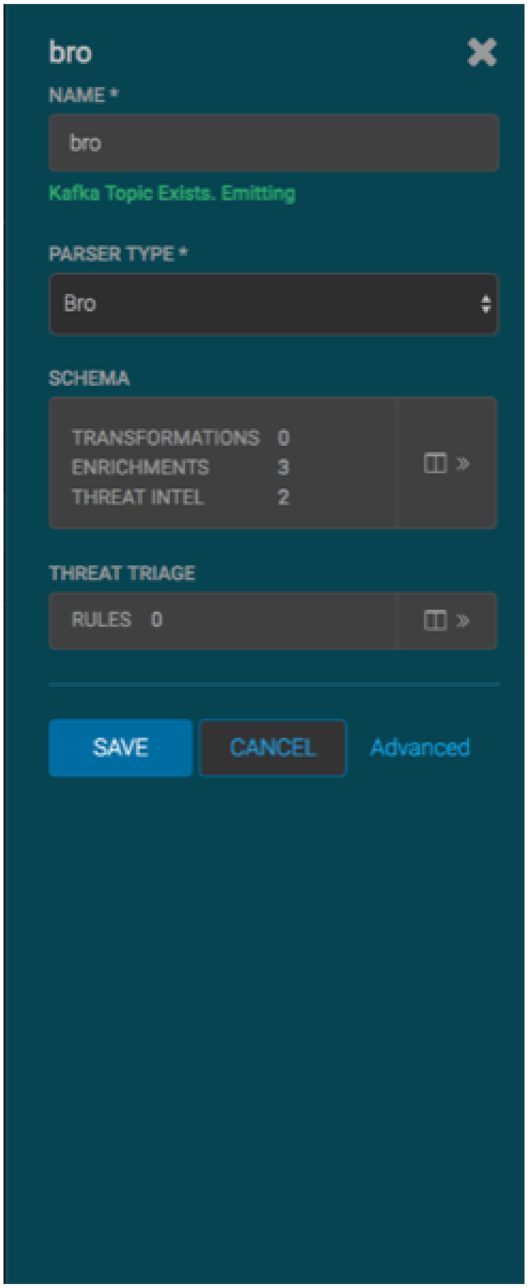
4.3.1. Modifying a Sensor

You can modify any sensor listed in HCP Management module.

1. Select **Sensors** in the **Operations** panel on the left side of the window, and then click  (edit button) for the sensor you want to modify.

   The Management module displays a panel populated with the sensor configuration information.

Figure 4.1. Sensor Panel



2. You can modify the following information for the sensor:

   - Sensor name
   - Parser type
   - Schema information
   - Threat triage information

For more information on using the Sensor panel to modify sensor information, see Using the Management Module.

3. Click **Save** to save your changes.

4.3.2. Deleting a Sensor

You can delete any sensor listed in HCP Management module.

> **Important**
>
> You must take the sensor offline before deleting it.

1. Select the check box next to the appropriate sensor in the Sensors table.

   You can delete more than one sensor at a time by clicking multiple check boxes.

2. From the Actions menu, choose **Delete**.

4.4. Monitoring Sensors

You can use the Metron Error Dashboard to monitor sensor error messages and troubleshoot them.

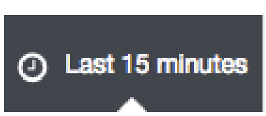
The Metron user interface provides two dashboards: the Metron Dashboard and the Metron Error Dashboard. The first dashboard, the Metron Dashboard, provides sensor-specific data that can be used to identify, investigate, and analyze cybersecurity data. This first dashboard is described extensively in the HCP User Guide. The second dashboard, the Metron Error Dashboard, receives information on all errors detected by HCP. This section describes the Error Dashboard in detail and provides instruction on how to use the dashboard to monitor sensor errors and troubleshoot problems and contains the following sections:

- Displaying the Metron Error Dashboard [71]
- Default Metron Error Dashboard [72]
- Loading Metron Templates [73]

4.4.1. Displaying the Metron Error Dashboard

The Metron Dashboard user interface defaults to displaying the Metron Dashboard. To display the Metron Error Dashboard, complete the following steps:

1. Click  (Load Saved Dashboard icon) in the upper right corner of the Metron Dashboard, then choose **Metron Error Dashboard** from the list of dashboards.

2. Click 

(timeframe tab) in the upper right corner of the Metron Error Dashboard to choose the timeframe you need the error dashboard to cover

The Metron Error dashboard receives the following information for all error messages:

- Exception
- Hostname - which machine the error occurred on
- Stack trace
- Time - When the error occurred
- Message
- Raw Message - original message
- Raw_message_bytes - The bytes of the original message
- Hash - To determine if there is a duplicate message
- Source_type - Identifies source sensor
- Error type - Parser error, etc.

4.4.2. Default Metron Error Dashboard

The following list contains a description of each of the sections that display by default in the Metron Error dashboard.

Total Error Messages	The total number of error messages received during the interim you have specified.
Unique Error Messages	The total number of unique error messages received during the interim you have specified.
Errors Over Time	A **detailed message panel** displays the raw data from your search query.
Error Source	When you submit a search query, the 500 most recent documents that match the query are listed in the **Documents** table which is displayed in the center of the **Discover** window.
Errors by Error Type	A list of all of the fields associated with a selected index pattern. This list is displayed on the left side of the **Discover** window.
Error Type Proportion	Use the **line chart** when you want to display high density time series. This chart is useful for comparing one series with another.
Errors by Type	You can use the **mark down widget panel** to provide explanations or instructions for the dashboard.
List of Errors	You can use a **metric panel** to display a single large number such as the number of hits or the average of a numeric field.

The default Error dashboard should look similar to the following:

Figure 4.2. Error Dashboard

4.4.3. Loading Metron Templates

HCP provides templates for the Metron UI dashboards. You might want to load or reload these templates if the Metron UI is not displaying the default dashboard panes, or if you would like to return to the default format.

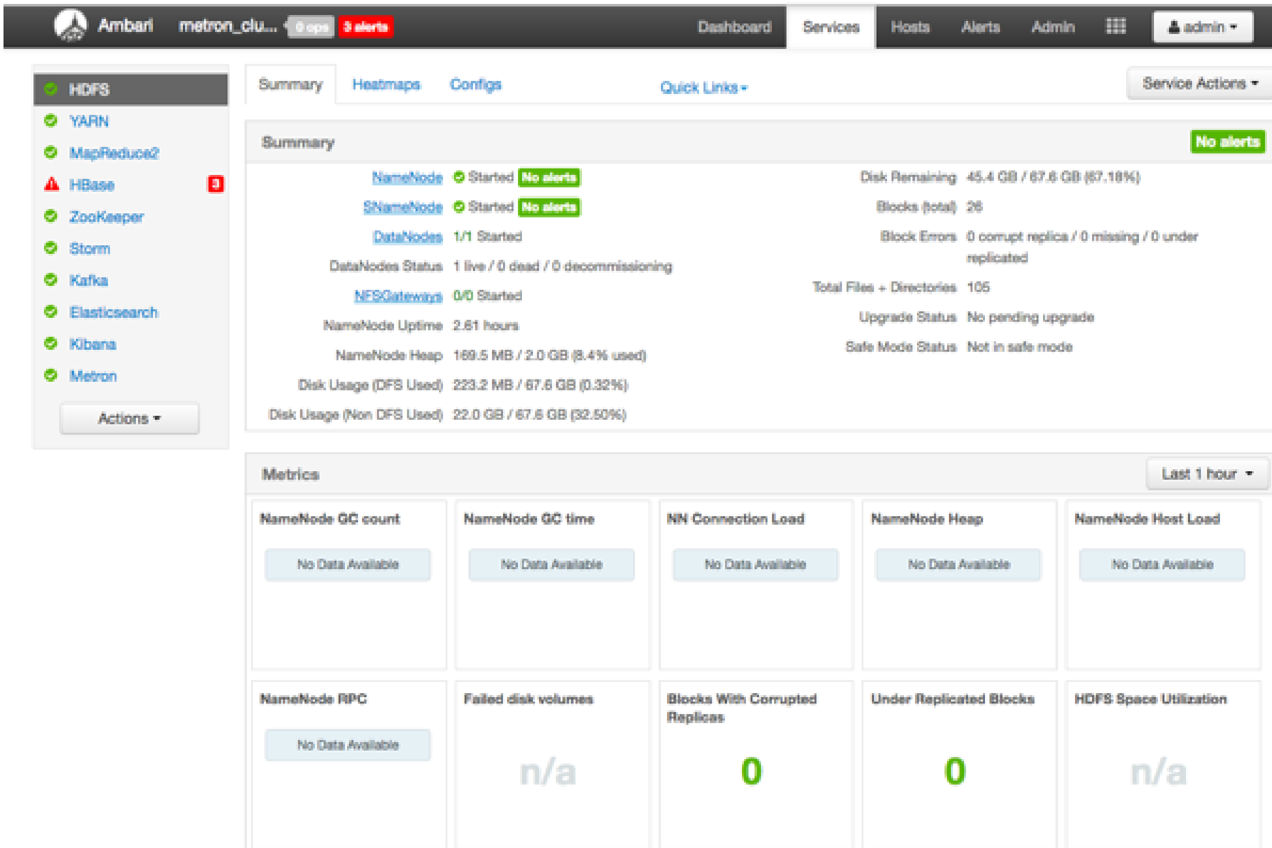
To load the Metron templates, complete the following steps:

1. Display the Ambari UI:

```
https://$METRON_HOME:8080
```

2. Click the **Services** tab and select Kibana in the left pane of the window.

Figure 4.3. Ambari Services Tab

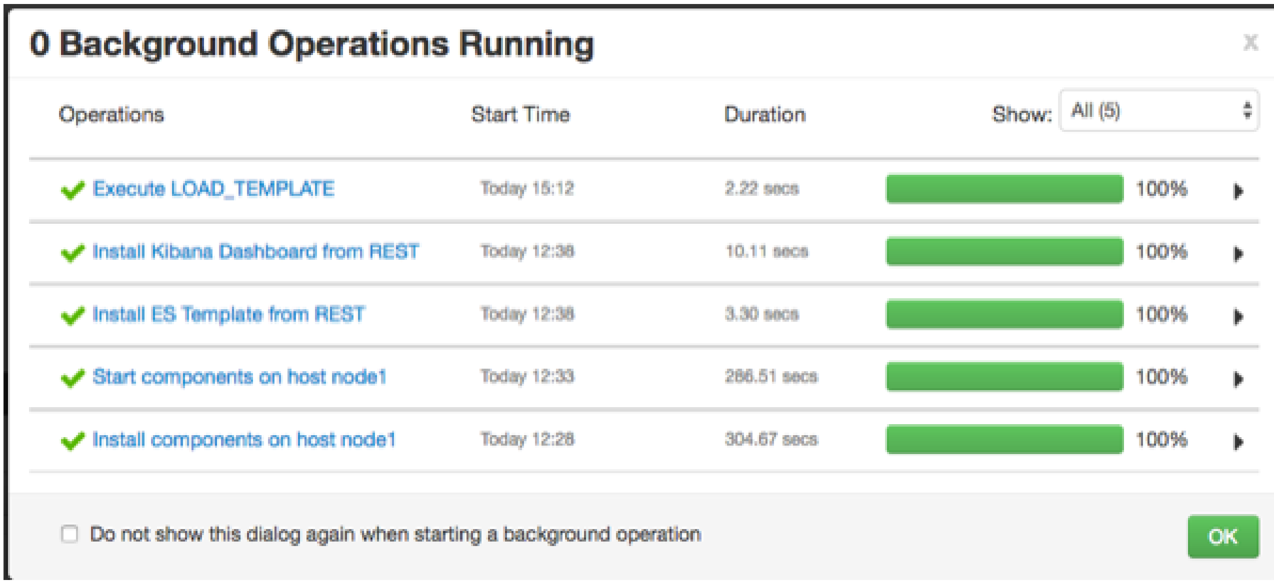


3. From the **Service Actions** pull down menu, select **Load Template**.
4. Click the **OK** button to confirm your selection.

Figure 4.4. Confirmation Dialog Box

Ambari displays a dialog box listing the background operations it is running.

Figure 4.5. Amberi Background Operations



5. Click the **OK** button to dismiss the dialog box.

   Ambari has completed loading the Metron template. You should be able to see the default formatting in the Metron dashboards.

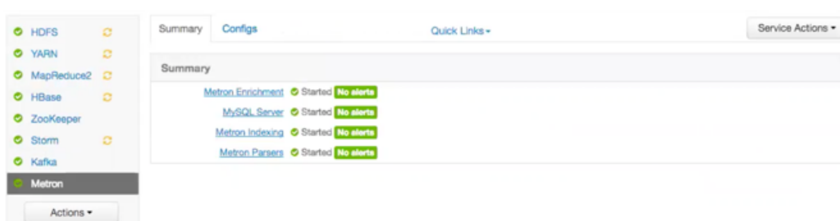
4.5. Starting and Stopping Parsers

You might want to stop or start parsers as you refine or focus your cybersecurity monitoring. You can easily stop and start parsers by using Ambari.

To start or stop a parser, complete the following steps:

1. Display the Ambari tool and navigate to **Services** > **Metron** > **Summary**.

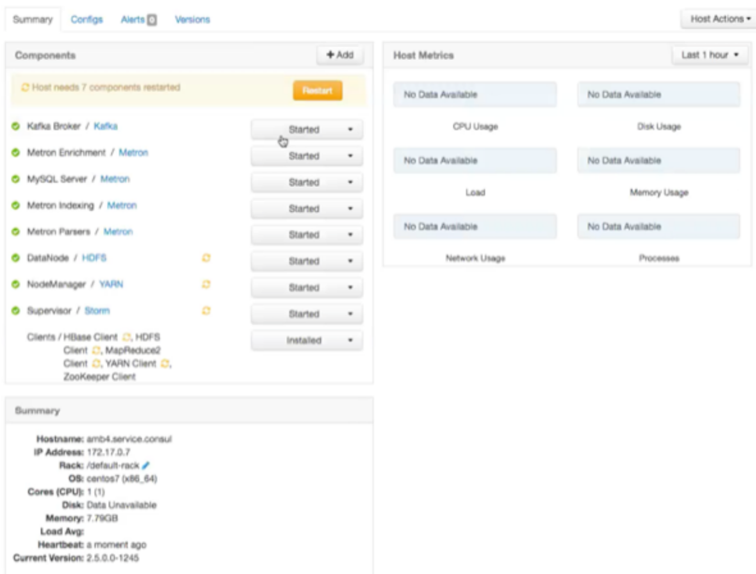
   Figure 4.6. Ambari Metron Summary Window

2. Under Summary, click on **Metron Parsers** to display the **Components** window.

   The Components window displays a list of Metron hosts and which components reside on each host.

Figure 4.7. Components Window



3. Click the **Started/Stopped** button by **Metron Parsers** to change the status of the Parsers then click the **Confirmation** button to verify that you want to start or stop the parsers.

   Ambari displays the **Background Operation Running** dialog box.

4. Click **Stop Metron Parsers**.

   Ambari displays the **Stop Metron Parsers** dialog box.

5. Click the entry for your Metron cluster, then click **Metron Parser Stop** again.

   Ambari displays a dialog box for your Metron cluster which lists the actions as is stops the parsers.

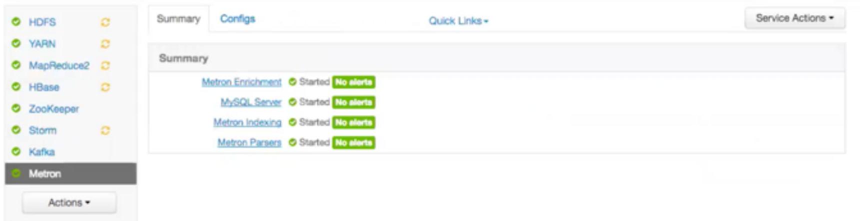
4.6. Starting and Stopping Enrichments

You might want to stop or start enrichments as you refine or focus your cybersecurity monitoring. You can easily stop and start enrichments by using Ambari.

To start or stop the enrichments, complete the following steps:

1. Display the Ambari tool and navigate to **Services** > **Metron** > **Summary**.

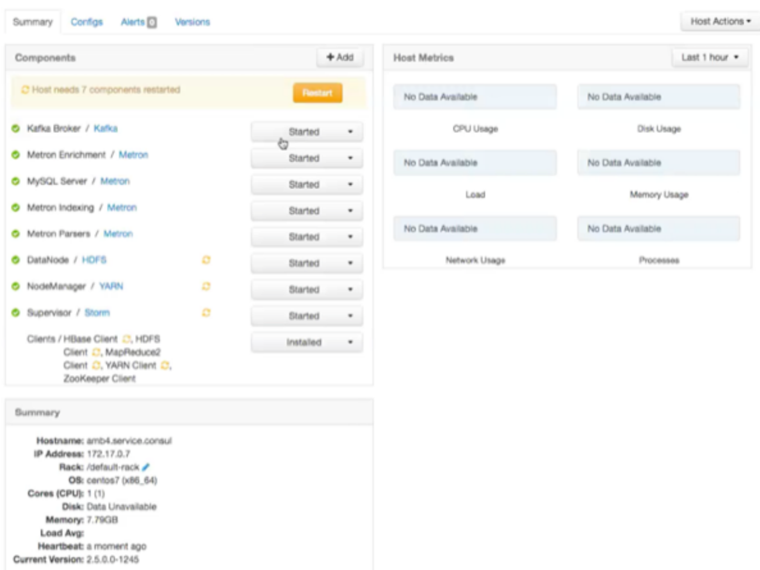
Figure 4.8. Ambari Metron Summary Window



2. Under Summary, click on **Metron Enrichments** to display the **Components** window.

   This window displays a list of HCP hosts and which components reside on each host.

Figure 4.9. Components Window



3. Click the **Started/Stopped** button by **Metron Enrichments** to change the status of the Enrichments then click the **Confirmation** button to verify that you want to start or stop the enrichments.

   Ambari displays the **Background Operation Running** dialog box.

4. Click **Stop Metron Enrichments**.

   Ambari displays the **Stop Metron Enrichments** dialog box.

5. Click the entry for your Metron cluster, then click **Metron Enrichments Stop** again.

   Ambari displays a dialog box for your Metron cluster which lists the actions as is stops the enrichments.

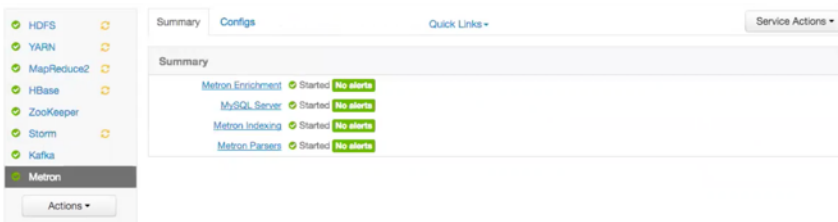
4.7. Starting and Stopping Indexing

You might want to stop or start indexing as you refine or focus your cybersecurity monitoring. You can easily stop and start indexing by using Ambari.

To start or stop indexing, complete the following steps:

1. Display the Ambari tool and navigate to **Services** > **Metron** > **Summary**.

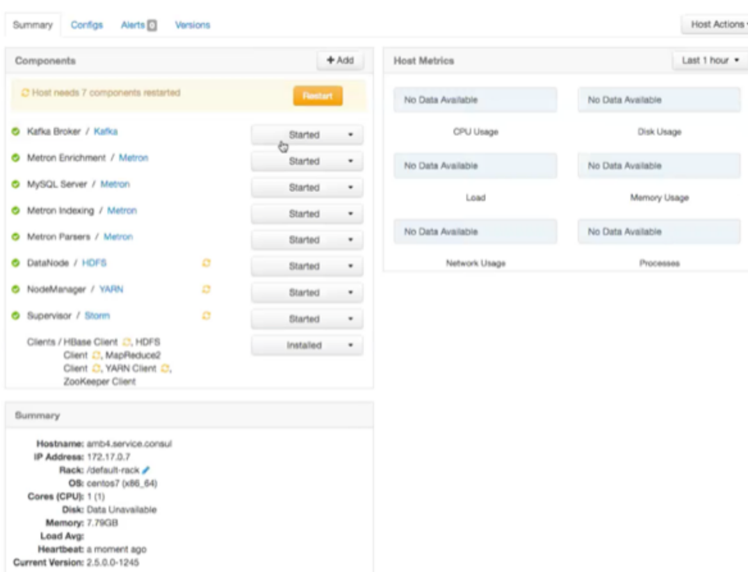
   Figure 4.10. Ambari Metron Summary Window

2. Under Summary, click on **Metron Indexing** to display the **Components** window.

   This window displays a list of HCP hosts and which components reside on each host.

   Figure 4.11. Components Window

3. Click the **Started/Stopped** button by **Metron Indexing** to change the status of the Indexing then click the **Confirmation** button to verify that you want to start or stop the indexing.

   Ambari displays the **Background Operation Running** dialog box.

4. Click **Stop Metron Indexing**.

Ambari displays the **Stop Metron Indexing** dialog box.

5. Click the entry for your Metron cluster, then click **Metron Indexing Stop** again.

   Ambari displays a dialog box for your Metron cluster which lists the actions as it stops the indexing.

4.8. Modifying the Elasticsearch Template

You can modify the Elasticsearch template to change the settings in your HCP environment. Some of these settings will help optimize your system performance.

- indexing.workers
- indexing.executors
- bolt.hdfs.batch.size

1. Display the Ambari tool and navigate to **Services** > **Metron** > **Summary** > **Advanced**.
2. Click **Advanced metron-env**.

   Ambari displays the contents of the `metron-env` file which includes the elasticsearch.properties template.

3. Modify the appropriate properties, then click **Save** at the top of the window.

5. Concepts

This chapter provides more in-depth information about the terminology used in the rest of this guide. This chapter contains detailed information on the following:

- Parsers [80]
- Telemetry Data Source Parsers Bundled with Hortonworks Cybersecurity Suite [84]
- Enrichment Framework [87]

5.1. Parsers

Parsers are pluggable components that transform raw data (textual or raw bytes) into JSON messages suitable for downstream enrichment and indexing. Data flows through the parser bolt via Kafka and into the enrichments topology in Storm. Errors are collected with the context of the error (for example stacktrace) and the original message causing the error and sent to an error queue. Invalid messages as determined by global validation functions are also treated as errors and sent to an error queue.

HCP supports two types of parsers: Java and general purpose. Each of these parsers plus the parser configuration are described in the following sections.

- Java Parsers [80]
- General Purpose Parsers [80]
- Parser Configuration [81]

5.1.1. Java Parsers

The Java parser is written in Java and conforms with the MessageParser interface. This kind of parser is optimized for speed and performance and is built for use with higher-velocity topologies. Java parsers are not easily modifiable; to make changes to them, you must recompile the entire topology.

Currently, the Java adapters included with HCP are as follows:

- org.apache.metron.parsers.ise.BasicIseParser
- org.apache.metron.parsers.bro.BasicBroParser
- org.apache.metron.parsers.sourcefire.BasicSourcefireParser
- org.apache.metron.parsers.lancope.BasicLancopeParser

5.1.2. General Purpose Parsers

The general purpose parser is primarily designed for lower-velocity topologies or for quickly setting up a temporary parser for a new telemetry. General purpose parsers are defined using a config file, and you need not recompile the topology to change them. HCP supports two general purpose parsers: Grok and CSV.

Grok parser

The Grok parser class name (parserClassName) is `org.apache.metron,parsers.GrokParser`.

Grok has the following entries and predefined patterns for `parserConfig`:

`grokPath`	The patch in HDFS (or in the Jar) to the Grok statement
`patternLabel`	The pattern label to use from the Grok statement
`timestampField`	The field to use for timestamp
`timeFields`	A list of fields to be treated as time
`dateFormat`	The date format to use to parse the time fields
`timezone`	The timezone to use. UTC is the default.

CSV Parser

The CSV parser class name (parserClassName) is org.apache.metron,parsers.csv.CSVParser

CSV has the following entries and predefined patterns for `parserConfig`:

`timestampFormat`	The date format of the timestamp to use. If unspecified, the parser assumes the timestamp is ms since UNIX epoch.
`columns`	A map of column names you wish to extract from the CSV to their offsets. For example, `{ 'name' : 1,'profession' : 3}` would be a column map for extracting the 2nd and 4th columns from a CSV.
`separator`	The column separator. The default value is ",".

5.1.3. Parser Configuration

The configuration for the various parser topologies is defined by JSON documents stored in ZooKeeper. The JSON document is structured in the following way:

parserClassName	The fully qualified class name for the parser to be used.
sensorTopic	The Kafka topic to send the parsed messages to.
parserConfig	A JSON Map representing the parser implementation specific configuration.
fieldTransformations	An array of complex objects representing the transformations to be done on the message generated from the parser before writing out to the Kafka topic. The fieldTransformations is a complex object which defines a transformation that can be done to a message. This transformation can perform the following:

- Modify existing fields to a message
- Add new fields given the values of existing fields of a message
- Remove existing fields of a message

5.1.3.1. fieldTransformation Configuration

In this example, the host name is extracted from the URL by way of the URL_TO_HOST function. Domain names are removed by using DOMAIN_REMOVE_SUBDOMAINS, thereby creating two new fields (`full_hostname` and `domain_without_subdomains`) and adding them to each message.

Figure 5.1. Configuration File with Transformation Information

The format of a fieldTransformation is as follows:

input	An array of fields or a single field representing the input. This is optional; if unspecified, then the whole message is passed as input.
output	The outputs to produce from the transformation. If unspecified, it is assumed to be the same as inputs.
transformation	The fully qualified class name of the transformation to be used. This is either a class which implements FieldTransformation or a member of the FieldTransformations enum.
config	A String to Object map of transformation specific configuration.

HCP currently implements the following fieldTransformations options:

REMOVE

This transformation removes the specified input fields. If you want a conditional removal, you can pass a Metron Query Language statement to define the conditions under which you want to remove the fields.

The following example removes `field1` unconditionally:

```
{
...
    "fieldTransformations" : [
          {
            "input" : "field1"
          , "transformation" : "REMOVE"
          }
                      ]
}
```

The following example removes field1 whenever field2 exists and has a corresponding value equal to 'foo':

```
{
...
  "fieldTransformations" : [
          {
            "input" : "field1"
          , "transformation" : "REMOVE"
          , "config" : {
              "condition" : "exists(field2) and field2 ==
 'foo'"
                       }
          }
                      ]
}
```

IP_PROTOCOL

This transformation maps IANA protocol numbers to consistent string representations.

The following example maps the `protocol` field to a textual representation of the protocol:

```
{
...
    "fieldTransformations" : [
          {
            "input" : "protocol"
          , "transformation" : "IP_PROTOCOL"
          }
                      ]
}
```

STELLAR, lo

This transformation executes a set of transformations expressed as Stellar Language statements.

The following example adds three new fields to a message:

utc_timestamp	The UNIX epoch timestamp based on the timestamp field, a dc field which is the data center the message comes from and a dc2tz map mapping data centers to timezones.
url_host	The host associated with the url in the url field.
url_protocol	The protocol associated with the url in the url field.

```
{
```

```
...
    "fieldTransformations" : [
          {
           "transformation" : "STELLAR"
          ,"output" : [ "utc_timestamp", "url_host",
 "url_protocol" ]
          ,"config" : {
            "utc_timestamp" : "TO_EPOCH_TIMESTAMP(timestamp,
 'yyyy-MM-dd
HH:mm:ss', MAP_GET(dc, dc2tz, 'UTC') )"
           ,"url_host" : "URL_TO_HOST(url)"
           ,"url_protocol" : "URL_TO_PROTOCOL(url)"
                      }
          }
                      ]
   ,"parserConfig" : {
      "dc2tz" : {
                "nyc" : "EST"
               ,"la" : "PST"
               ,"london" : "UTC"
                }
    }
}
```

Note that the dc2tz map is in the parser config, so it is accessible in the functions.

5.2. Telemetry Data Source Parsers Bundled with Hortonworks Cybersecurity Suite

Telemetry data sources are sensors that provide raw events that are captured and pushed into Kafka topics to be ingested in Hortonworks Cybersecurity Package (HCP) powered by Metron. This section describes the telemetry data sources bundled with HCP 1.0:

- Snort [84]
- Bro [85]
- YAF (NetFlow) [85]
- Indexing [86]
- pcap [86]

For information about how to add telemetry data sources, see Adding a New Telemetry Data Source.

5.2.1. Snort

Snort is one of the more popular network intrusion prevention systems (NIPS). Snort monitors network traffic and produces alerts that are generated based on signatures from community rules. HCP plays the output of the packet capture probe to Snort and whenever Snort alerts are triggered. HCP uses the kafka-console-producer to send these alerts to a

Kafka topic. After the Kafka topic receives Snort alerts, they are retrieved by the parsing topology in Storm.

By default, the Snort parser is configured to use ZoneId.systemDefault() for the source `timeZone` for the incoming data and MM/dd/yy-HH:mm:ss.SSSSSS as the default `dateFormat`. Valid timezones are per Java's ZoneId.getAvailableZoneIds(). DateFormats should be valid per options listed on the following website: https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html. Below is a sample configuration with the `dateFormat` and `timeZone` explicitly set in the parser config.

```
"parserConfig": {
"dateFormat" : "MM/dd/yy-HH:mm:ss.SSSSSS",
"timeZone" : "America/New_York"
}
```

Note

When you install and configure Snort, you must configure Snort to include the year in the timestamp by modifying the `snort.conf` file as follows:

```
# Configure Snort to show year in timestamps
config show_year
```

This is important for the proper functioning of indexing and analytics.

5.2.2. Bro

The Bro ingest data source is a custom Bro plug-in that pushes DPI (deep packet inspection) metadata into HCP.

Bro is primarily used as a DPI metadata generator. HCP does not currently use the IDS alert features of Bro. HCP integrates with Bro by way of a Bro plug-in, and does not require recompiling of Bro code.

The Bro plug-in formats Bro output messages into JSON and puts them into a Kafka topic. The JSON message output by the Bro plug-in is designed to be parsed by the HCP Bro parsing topology.

DPI metadata is not a replacement for packet capture (pcap), but rather a complement. Extracting DPI metadata (API Layer 7 visibility) is expensive, and therefore is performed on only selected protocols. You should enable DPI for HTTP and DNS protocols so that, while the pcap probe records every single packets it sees on the wire, the DPI metadata is extracted only for a subset of these packets.

5.2.3. YAF (NetFlow)

The YAF (yet another flowmeter) data source ingests NetFlow data into HCP.

Not everyone wants to ingest pcap data due to space constraints and the load exerted on all infrastructure components. NetFlow, while not a substitute for pcap, is a high-level summary of network flows that would be contained in the pcap files. If you do not want to ingest pcap, then you should at least enable NetFlow. HCP uses YAF to generate IPFIX (NetFlow) data from the HCP pcap probe, so the output of the probe is IPFIX instead of raw packets. If NetFlow is generated instead of pcap, then the NetFlow data goes to the generic parsing topology instead of the pcap topology.

5.2.4. Indexing

The Indexing topology takes data ingested into Kafka from enriched topologies and sends the data to an indexing bolt configured to write to one or more of the following indices:

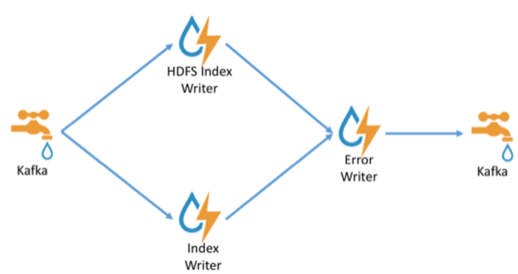
- Elasticsearch or Solr
- HDFS under `/apps/metron/enrichment/indexed`

Indices are written in batch and the batch size is specified in the Enrichment Configuration file by the batchSize parameter. This configuration is variable by sensor type.

Errors during indexing are sent to a Kafka topic named `indexing_error`.

The following figure illustrates the data flow between Kafka, the Indexing topology, and HDFS.

Figure 5.2. Indexing Architecture



5.2.5. pcap

Packet capture (pcap) is a performant C++ probe that captures network packets and streams them into Kafka. A pcap Storm topology then streams them into HCP. The purpose of including pcap source with HCP is to provide a middle tier in which to negotiate retrieving packet capture data that flows into HCP. This packet data is of a form that libpcap-based tools can read.

The network packet capture probe is designed to capture raw network packets and bulk-load them into Kafka. Kafka files are then retrieved by the pcap Storm topology and bulk-loaded into Hadoop Distributed File System (HDFS). Each file is stored in HDFS as a sequence file.

HCP provides three methods to access the pcap data:

- Rest API
- pycapa
- DPDK

There can be multiple probes into the same Kafka topic. The recommended hardware for the probe is an Intel family of network adapters that are supportable by Data Plane Development Kit (DPDK).

5.3. Enrichment Framework

Enrichments add additional context to the streaming message. The enrichment framework takes the data from the parsing topologies that have been normalized into the HCP data format (JSON files) and performs the following enhancements:

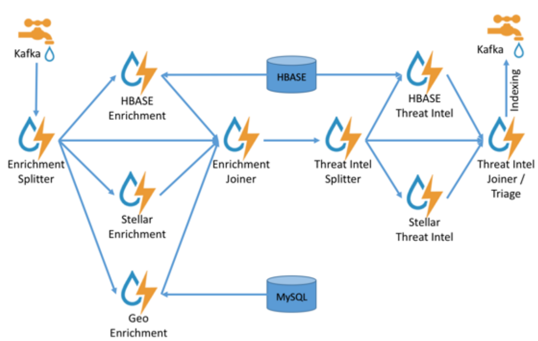
- Enriches messages with external data from data stores by adding new information based on existing fields in the messages
- Marks messages as threats based on data in external data stores
- Marks threat alerts with a numeric triage level based on a set of Stellar rules

The configuration for the enrichment topology is defined by JSON documents stored in ZooKeeper. HCP features two types of configurations:

- Sensor Enrichment Configuration [87]
- Global Configuration [92]

The following figure illustrates the enrichment flow for both individual sensor enrichment and threat intelligence enrichment.

Figure 5.3. HCP Enrichment Flow



5.3.1. Sensor Enrichment Configuration

The sensor enrichment configuration configures enrichments for a given sensor (for example, Snort). The sensor enrichment configuration configures two types of enrichments: individual sensor enrichments and threat intelligence enrichments. The configuration for both types of enrichments is a complex JSON object with the following top-level fields:

index	The name of the sensor
batchSize	The size of the batch that is written to the indices at once
enrichment	A complex JSON object representing the configuration of the enrichments

threatIntel — A complex JSON object representing the configuration of the threat intelligence enrichments

The remaining configuration differs for the two types of enrichments. See the following sections for information about both of these configuration types.

5.3.1.1. Individual Sensor Enrichments

HCP includes the following individual sensor enrichments:

Geo — Provides GeoIP information, which includes coordinates, city, state, and country information, to any external IP address.

Asset — Provides the host name for an IP address. If the IP address is known, then the enrichment provides everything else that is known of the asset from the LDAP, AD, or enterprise inventory stores.

User — Provides the user that owns the session/alert associated with the IP-application pair.

The JSON documents for the individual enrichment configurations are structured as follows:

Table 5.1. Individual Enrichment Configuration Fields

Field	Description	Example
`fieldToTypeMap`	In the case of a simple HBase enrichment (in other words, a key/value lookup), the mapping between fields and the enrichment types associated with those fields must be known. This enrichment type is used as part of the HBase key.	`"fieldToTypeMap" : { "ip_src_addr" : [ "asset_enrichment" ] }`
`fieldMap`	The map of enrichment bolts names to configuration handlers which know how to split the message up. The simplest of which is just a list of fields. More complex examples would be the STELLAR enrichment which provides STELLAR statements. Each field is sent to the enrichment referenced in the key.	`"fieldMap": {"hbaseEnrichment": ["ip_src_addr","ip_dst_addr"]}`
`config`	The general configuration for the enrichment.	`"config": {"typeToColumnFamily": { "asset_enrichment" : "cf" } }`

The `config` map is intended to house enrichment-specific configurations. For example, `hbaseEnrichment` specifies the mappings between the enrichment types to the column families.

The `fieldMapcontents` contain the routing and configuration information for the enrichments. Routing defines how the messages is split up and sent to the enrichment adapter bolts. The simplest `fieldMapcontents` provides a simple list as in:

```
"fieldMap": {
     "geo": [
       "ip_src_addr",
       "ip_dst_addr"
```

```
      ],
      "host": [
        "ip_src_addr",
        "ip_dst_addr"
      ],
      "hbaseEnrichment": [
        "ip_src_addr",
        "ip_dst_addr"
      ]
      }
```

For the `geo`, `host`, and `hbaseEnrichment`, this is sufficient. However, more complex enrichments might contain their own configuration. Currently, the `stellar` enrichment requires a more complex configuration, such as:

```
"fieldMap": {
       ...
      "stellar" : {
        "config" : {
          "numeric" : {
                      "foo": "1 + 1"
                      }
          ,"ALL_CAPS" : "TO_UPPER(source.type)"
        }
      }
    }
```

Whereas the simpler enrichments just need a set of fields explicitly stated so they can be separated from the message and sent to the enrichment adapter bolt for enrichment and ultimately joined back in the join bolt, the stellar enrichment has its set of required fields implicitly stated through usage. For instance, if your Stellar statement references a field, it should be included and if not, then it should not be included. We did not want to require users to make explicit the implicit.

The other way in which the Stellar enrichment is somewhat more complex is in how the statements are executed. In the general purpose case for a list of fields, those fields are used to create a message to send to the enrichment adapter bolt and that bolt's worker will handle the fields one by one in serial for a given message. For Stellar enrichment, we wanted to have a more complex design so that users could specify the groups of stellar statements sent to the same worker in the same message (and thus executed sequentially). Consider the following configuration:

```
    "fieldMap": {
      "stellar" : {
        "config" : {
          "numeric" : {
                      "foo": "1 + 1"
                      "bar" : TO_LOWER(source.type)"
                      }
         ,"text" : {
                   "ALL_CAPS" : "TO_UPPER(source.type)"
                   }
        }
      }
    }
```

We have a group called `numeric` whose Stellar statements will be executed sequentially. In parallel to that, we have the group of Stellar statements under the group `text` executing. The intent here is to allow you to not force higher latency operations to be done sequentially.

5.3.1.2. Threat Intelligence Enrichments

HCP provides an extensible framework to plug in threat intelligence sources. Each threat intelligence source has two components: an enrichment data source and an enrichment bolt. The threat intelligence feeds are bulk loaded and streamed into a threat intelligence store similarly 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. Hortonworks recommends using a threat feed aggregator such as Soltra to dedup and normalize the feeds via STIX/TAXII. HCP provides an adapter that is able to read Soltra-produced STIX/TAXII feeds and stream them into HBase. HCP additionally provides a flat file and STIX bulk loader that can normalize, dedup, and bulk load or stream threat intelligence data into HBase even without the use of a threat feed aggregator.

The JSON documents for the threat intelligence enrichment configurations are structured in the following way:

Table 5.2. Threat Intelligence Enrichment Configuration

Field	Description	Example
`fieldToTypeMap`	In the case of a simple HBase threat intelligence enrichment (in other words, a key/value lookup), the mapping between fields and the enrichment types associated with those fields must be known. This enrichment type is used as part of the HBase key.	`"fieldToTypeMap" : { "ip_src_addr" : [ "malicious_ips" ] }`
`fieldMap`	The map of threat intelligence enrichment bolts names to fields in the JSON messages. Each field is sent to the threat intelligence enrichment bolt referenced in the key.	`"fieldMap": {"hbaseThreatIntel": ["ip_src_addr","ip_dst_addr"]}`
`triageConfig`	The configuration of the threat triage scorer. In the situation where a threat is detected, a score is assigned to the message and embedded in the indexed message.	`"riskLevelRules" : { "IN_SUBNET(ip_dst_addr, '192.168.0.0/24')" : 10 }`
`config`	The general configuration for the threat intelligence.	`"config": {"typeToColumnFamily": { "malicious_ips" : "cf" } }`

The `config` map is intended to house threat intelligence specific configurations. For instance, the `hbaseThreatIntel` threat intelligence adapter specifies the mappings between the enrichment types and the column families.

The `triageConfig` field utilizes the following fields:

Table 5.3. triageConfig Fields

Field	Description	Example
riskLevelRules	The mapping of Metron Query Language (see above) queries to a score.	"riskLevelRules" : { "IN_SUBNET(ip_dst_addr, '192.168.0.0/24')" : 10 }

Field	Description	Example
aggregator	An aggregation function that takes all non-zero scores representing the matching queries from `riskLevelRules` and aggregates them into a single score.	"MAX"

The supported aggregator functions are as follows:

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

The following is an example configuration for the YAF sensor:

```
{
  "index": "yaf",
  "batchSize": 5,
  "enrichment": {
    "fieldMap": {
      "geo": [
        "ip_src_addr",
        "ip_dst_addr"
      ],
      "host": [
        "ip_src_addr",
        "ip_dst_addr"
      ],
      "hbaseEnrichment": [
        "ip_src_addr",
        "ip_dst_addr"
      ]
    }
  ,"fieldToTypeMap": {
      "ip_src_addr": [
        "playful_classification"
      ],
      "ip_dst_addr": [
        "playful_classification"
      ]
    }
  },
  "threatIntel": {
    "fieldMap": {
      "hbaseThreatIntel": [
        "ip_src_addr",
        "ip_dst_addr"
      ]
    },
    "fieldToTypeMap": {
      "ip_src_addr": [
        "malicious_ip"
      ],
      "ip_dst_addr": [
        "malicious_ip"
      ]
```

```
    },
    "triageConfig" : {
      "riskLevelRules" : {
        "ip_src_addr == '10.0.2.3' or ip_dst_addr == '10.0.2.3'" : 10
      },
      "aggregator" : "MAX"
    }
  }
}
```

5.3.1.3. Using Stellar to Set up Threat Triage Configurations

The threat triage configuration defines conditions by associating them with scores. Because this is a per-sensor configuration, this fits nicely within the sensor enrichment configuration held in ZooKeeper. This configuration fits well within the threatIntel section of the configuration like so:

```
{
  ...
  ,"threatIntel" : {
             ...
           , "triageConfig" : {
                     "riskLevelRules" : {
                                 "condition1" : level1
                               , "condition2" : level2
                                  ...
                                        }
                     ,"aggregator" : "MAX"
                             }
                  }
}
```

riskLevelRules: Correspond to the set of condition to numeric level mappings that define the threat triage for this particular sensor.

aggregator: An aggregation function that takes all non-zero scores representing the matching queries from riskLevelRules and aggregates them into a single score.

The current supported aggregation functions are:

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

5.3.2. Global Configuration

Global enrichments are applied to all data sources as opposed to other enrichments that are applied at the field level. In other words, every message from every sensor is validated against the global configuration rules. The format of the global enrichment is a JSON string-to-object map that is stored in ZooKeeper. For example:

This configuration is stored in ZooKeeper and looks something like the following:

```
{
  "es.clustername": "metron",
  "es.ip": "node1",
  "es.port": "9300",
  "es.date.format": "yyyy.MM.dd.HH",
  "fieldValidations" : [
              {
                "input" : [ "ip_src_addr", "ip_dst_addr" ],
                "validation" : "IP",
                "config" : {
                    "type" : "IPV4"
                           }
              }
                       ]
}
```

For a complete list of all of the Stellar language functions currently supported by HCP, see Appendix A.

5.3.2.1. Global Validation

Inside the global configuration is a framework that validates all messages coming from all parsers. This is performed using validation plug-ins that make assertions about fields or whole messages.

The format for this framework is a `fieldValidations` field inside the global configuration. This is associated with an array of field validation objects structured as follows:

input
: An array of input fields or a single field. If this value is omitted, then the whole messages is passed to the validator.

config
: A string-to-object map for validation configuration. This value is optional if the validation function requires no configuration.

validation
: The validation function to be used. This is one of the following:

  - STELLAR: Execute a Query Language statement. Expects the query string in the condition field of the configuration.
  - IP: Verifies that the input fields are an IP address. By default, if no configuration is set, the value is assumed to be IPV4, but you can specify the type by passing in the type with either IPV6 or IPV4.
  - DOMAIN: Validates that the fields are all domains.
  - EMAIL: Validates that the fields are all email addresses.
  - URL: Validates that the fields are all URLs.
  - DATE: Validates that the fields compose a date. Expects format in the configuration.
  - INTEGER: Validates that the fields compose an integer. String representation of an integer is allowed.

REGEX_MATCH	Validates that the fields match a regular expression. Expects pattern in the configuration.
NOT_EMPTY	Validates that the fields exist and are not empty (after trimming).

5.3.3. Using Stellar for Queries

You can use Stellar to create queries.

The Stellar query language supports the following:

- Referencing fields in the enriched JSON
- Simple boolean operations: and, not, or
- Simple arithmetic operations: *, /, +, - on real numbers or integers
- Simple comparison operations <, >, <=, >=
- if/then/else comparisons (in other words, if var1 < 10 then 'less than 10' else '10 or more')
- Determining whether a field exists (via `exists`)
- The ability to have parenthesis to make order of operations explicit
- User defined functions

The following is an example of a Stellar query:

```
IN_SUBNET( ip, '192.168.0.0/24') or ip in [ '10.0.0.1', '10.0.0.2' ] or
 exists(is_local)
```

This query evaluates to “true” precisely when one of the following is true:

- The value of the ip field is in the 192.168.0.0/24 subnet.
- The value of the ip field is 10.0.0.1 or 10.0.0.2.
- The field is_local exists.

5.3.4. Using Stellar to Transform Sensor Data Elements

You can use Stellar to customize sensor data elements to more useful information. For example, you can transform a timestamp to be specific to your timezone.

```
TO_EPOCH_TIMESTAMP(timestamp, 'yyyy-MM-dd HH:mm:ss', MAP_GET(dc, dc2tz,
 'UTC'))
```

For a message with a timestamp and dc field, we want to transform the timestamp to an epoch timestamp given a timezone that we will look up in a separate map, called dc2tz.

This converts the timestamp field to an epoch timestamp based on the following:

- Format yyyy-MM-dd HH:mm:ss

- The value in dc2tz associated with the value associated with field dc, defaulting to UTC

The following is a list of Stellar transformation functions currently supported by HCP:

TO_LOWER(string)	Transforms the first argument to a lowercase string
TO_UPPER(string)	Transforms the first argument to an uppercase string
TO_STRING(string)	Transforms the first argument to a string
TO_INTEGER(x)	Transforms the first argument to an integer
TO_DOUBLE(x)	Transforms the first argument to a double
TRIM(string)	Trims white space from both sides of a string
JOIN(list, delim)	Joins the components of the list with the specified delimiter
SPLIT(string, delim)	Splits the string by the delimiter. Returns a list.
GET_FIRST(list)	Returns the first element of the list
GET_LAST(list)	Returns the last element of the list
GET(list, i)	Returns the i'th element of the list (i is 0-based).
MAP_GET(key, map, default)	Returns the value associated with the key in the map. If the key does not exist, the default will be returned. If the default is unspecified, then null will be returned.
DOMAIN_TO_TLD(domain)	Returns the TLD of the domain
DOMAIN_REMOVE_TLD(domain)	Removes the TLD of the domain.
REMOVE_TLD(domain)	Removes the TLD from the domain
URL_TO_HOST(url)	Returns the host from a URL
URL_TO_PROTOCOL(url)	Returns the protocol from a URL
URL_TO_PORT(url)	Returns the port from a URL
URL_TO_PATH(url)	Returns the path from a URL
TO_EPOCH_TIMESTAMP(dateTime, format, timezone)	Returns the epoch timestamp of the dateTime given the format If the format does not have a timestamp and you wish to assume a given timestamp, you may specify the timezone optionally.

5.3.5. Management Utility

You should store your configurations on disk in the following structure, starting at $BASE_DIR:

- **global.json:** The global configuration
- **sensors:** The subdirectory containing sensor-enrichment configuration JSON (for example, snort.json or bro.json)

By default, this directory is deployed by the Ansible infrastructure at `$METRON_HOME/config/zookeeper`.

While the configs are stored on disk, they must be loaded into ZooKeeper to be used. You can use the `$METRON_HOME/bin/zk_load_config.sh utility` program to do this.

This has the following options:

-f,–force	Force operation
-h,–help	Generate Help screen
-i,–input_dir <DIR>	The input directory containing configuration files with names such as "$source.json"
-m,–mode <MODE>	The mode of operation: DUMP, PULL, or PUSH
-o,–output_dir (DIR)	The output directory that will store the JSON configuration from ZooKeeper
-z,–zk_quorum <host:port, [host:port]*>	ZooKeeper quorum URL (zk1:port,zk2:port,...)

Following are some usage examples:

- To dump the existing configs from ZooKeeper on the single-node vagrant machine: `$METRON_HOME/bin/zk_load_configs.sh -z node1:2181 -m DUMP`
- To push the configs into ZooKeeper on the single-node vagrant machine: `$METRON_HOME/bin/zk_load_configs.sh -z node1:2181 -m PUSH -i $METRON_HOME/config/zookeeper`
- To pull the configs from ZooKeeper to the single-node vagrant machine disk: `$METRON_HOME/bin/zk_load_configs.sh -z node1:2181 -m PULL -o $METRON_HOME/config/zookeeper -f`

Appendix A. Stellar Language Functions

This section provides Stellar language functions supported by Hortonworks Cybersecurity Package (HCP) powered by Apache Metron.

The Stellar query language supports the following:

- Referencing fields in the enriched JSON
- Simple boolean operations: `and`, `not`, `or`
- Simple arithmetic operations: `*`, `/`, `+`, `-` on real numbers or integers
- Simple comparison operations <, >, <=, >=
- if/then/else comparisons (for example, `if var1 < 10 then 'less than 10' else '10 or more'`)
- Determining whether a field exists (via `exists`)
- The ability to have parenthesis to make order of operations explicit
- User defined functions

The following keywords need to be single quote escaped in order to be used in Stellar expressions:

Table A.1. Stellar Language Keywords

not	else	exists	if	then
and	or	in	==	!=
/<=	/>	/>=	/+	/-
/<	?	/*	/	,

Stellar Language Inclusion Checks ("in" and "not in")

- "in" supports string contains. e.g., "'foo' in 'foobar' == true"
- "in" supports collection contains. e.g., "'foo' in ['foo', 'bar'] == true"
- "in" supports map key contains. e.g., "'foo' in { 'foo' : 5} == true"
- "not in" is the negation of the in expression. e.g., "'grok' not in 'foobar' == true`"

Stellar Language Comparisons (`<`, `<=`, `>`, `>=`)

- If either side of the comparison is null then return false.
- If both values being compared implement number then the following:
  - If either side is a double then get double value from both sides and compare using given operator.
  - Else if either side is a float then get float value from both sides and compare using given operator.

- Else if either side is a long then get long value from both sides and compare using given operator.
- Otherwise get the int value from both sides and compare using given operator.

- If both sides are of the same type and are comparable then use the compareTo method to compare values.
- If none of the above are met then an exception is thrown.

Stellar Language Equality Check (`==`, `!=`)

Below is how the `==` operator is expected to work:

- 1. If either side of the expression is null then check equality using Java's `==` expression.
- Else if both sides of the expression are of Java's type Number then:
  - If either side of the expression is a double then use the double value of both sides to test equality.
  - Else if either side of the expression is a float then use the float value of both sides to test equality.
  - Else if either side of the expression is a long then use long value of both sides to test equality.
  - Otherwise use int value of both sides to test equality
- Otherwise use equals method compare the left side with the right side.

The `!=` operator is the negation of the above.

Table A.2. Stellar Language Functions

Function	Description	Input	Returns
ABS	Returns the absolute value of a number	number - The number to take the absolute value of	The absolute value of the number passed in
BIN	Computes the bin that the value is in given a set of bounds	• value - the value to bin • bounds -A list of value bounds (excluding min and max) in sorted order	Which bin N the value falls in such that bound(N-1) <value <= bound(N). No min and max bounds are provided, so values small than the 0'th bound go in the 0'th bin, and values great than the last bound go in the M'th bin.
BLOOM_ADD	Adds an element to the bloom filter passed in	• bloom - The bloom filter • value* - The values to add	Bloom Filter
BLOOM_EXISTS	If the bloom filter contains the value	• bloom - The bloom filter • value - The value to check	True if the filter might contain the value and false otherwise
BLOOM_INIT	Returns an empty bloom filter	• expectedInsertions - The expected insertions	Bloom Filter

Function	Description	Input	Returns
		• falsePositiveRate - The false positive rate you are willing to tolerate	
BLOOM_MERGE	Returns a merged bloom filter	• bloomfilters - A list of bloom filters to merge	Bloom Filter or null if the list is empty
DAY_OF_MONTH	The numbered day within the month. The first day within the month has a value if 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The numbered day within the month
DAY_OF_WEEK	The numbered day within the week. The first day of the week, Sunday, has a value of 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The numbered day within the week.
DAY_OF_THE_YEAR	The day number within the year. The first day of the year has value of 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The day number within the year
DOMAIN_REMOVE_SUBDOMAINS	Remove subdomains from a domain	• domain - Fully qualified domain name	The domain without the subdomains. (For example, DOMAIN_REMOVE_SUBDOMAINS ('mail.yahoo.com') yields 'yahoo.com')
DOMAIN_REMOVE_TLD	Removes the top level domain (TLD) suffix from a domain	• domain - Fully qualified domain name	The domain without the TLD. (For example, DOMAIN_REMOVE_TLD('mail.yahoo.co.uk') yields 'mail.yahoo')
DOMAIN_TO_TLD	Extracts the top level domain from a domain	• domain - Fully qualified domain name	The domain of the TLD. (For example, DOMAIN_TO_TLD('mail.yahoo.com.uk') 'yields 'co.uk')
ENDS_WITH	Determines whether a string ends with a suffix	• string - The string to test • suffix - The proposed suffix	True if the string ends with the specified suffix and false if otherwise
ENRICHMENT_EXISTS	Interrogates the HBase table holding the simple HBase enrichment data and returns whether the enrichment type and indicator are in the table	• enrichment_type - The enrichment type • indicator - The string indicator to look up • nosql_table - The NoSQL table to use • column_family - The column family to use	True if the enrichment indicator exists and false otherwise
ENRICHMENT_GET	Interrogates the HBase table holding the simple HBase enrichment data and retrieves the tabular value associated with the enrichment type and indicator	• enrichment_type - The enrichment type • indicator - The string indicator to look up • nosql_table - The NoSQL table to use • column_family - The column family to use	A map associated with the indicator and enrichment type. Empty otherwise.
FILL_LEFT	Fills or pads a given string with a given character, to a given length on the left	• input - string • fill - the fill character • len - the required length	The filled string

Function	Description	Input	Returns
FILL_RIGHT	Fills or pads a given string with a given character, to a given length on the right	• input - string • fill - the fill character • len - the required length	Last element of the list
FORMAT	Returns a formatted string using the specified format string and arguments. Uses Java's string formatting conventions	• format - string • arguments - object(s)	A formatted string
GEO_GET	Look up an IPV4 address and returns geographic information about it.	• ip - The IPV4 address to look up • fields – Optional list of GeoIP fields to grab. Options are locID, country, city postalCode, dmaCode, latitude, longitude, location_point • len - the required length	If a Single field is requested, a string of the field. If multiple fields are requested, a map of string of fields. Otherwise null.
GET	Returns the i'th element of the list	• input - List • i - The index (0-based)	First element of the list
GET_FIRST	Returns the first element of the list	• input - List	First element of this list
GET_LAST	Returns the last element of the list	• input - List	Last element of the list
HLLP_CARDINALITY	Returns HyperLogLogPlus-estimated cardinality for this set.	• input - hyperLogLogPlus - the hllp set	Long value representing the cardinality for this set
HLLP_INIT	Initializes the set	• p (required) - The precision value for the sparse set. • sp - The precision value for the sparse set. If sp Is not specified the sparse set will be disabled.	A new HyperLogLogPlus set
HLLP_MERGE	Merge hllp sets together	• hllp1 - First hllp set • hllp2 - Second hllp set • hllpn - Additional sets to merge	A new merged HyperLogLogPlus estimator set
HLLP_OFFER	Add value to set	• hyperLogLogPlus - The hllp set • o - Object to add to the set	The HyperLogLogPlus set with a new object added
IN_SUBNET	Returns true if an IP is within a subnet range	• ip - The IP address in string form • cidr+ - One or more IP ranges specified in CIDR notation (for example, 192.168.0.0/24)	True if the IP address is within at least one of the network ranges and false if otherwise
IS_DATE	Determines if the date contained in the string conforms to the specified format	• date - The date in string form • format - The format of the date	True if the date is in the specified format and false if otherwise

Function	Description	Input	Returns
IS_DOMAIN	Tests if a string is a valid domain. Domain names are evaluated according to the standards RFC1034 Section 3, and RFC1123 section 2.1.	• address - The string to test	True if the string is a valid domain and false if otherwise
IS_EMAIL	Tests if a string is a valid email address	• address -The string to test	True if the string is a valid email address and false if otherwise
IS_EMPTY	Returns true if string or collection is empty or null and false if otherwise	• input - Object of string or collection type (for example, list)	True if the string or collection is empty or null and false if otherwise
IS_INTEGER	Determines whether or not an object is an integer	• x - The object to test	True if the object can be converted to an integer and false if otherwise
IS_IP	Determine if a string is an IP or not	• ip - An object which we wish to test is an IP • type (optional) - Object of string or collection type (for example, list) one of IPv4 or IPv6. The default is IPv4.	True if the string is an IP and false if otherwise
IS_URL	Tests if a string is a valid URL	• url - The string to test	True if the string is a valid URL and false otherwise
JOIN	Joins the components in the list of strings with the specified delimiter	• list - List of strings • delim - String delimiter	String
LENGTH	Returns the length of a string or size of a collection. Returns 0 for empty or null strings.	• input - Object of string or collection type (for example, list)	Integer
MAAS_GET_ENDPOINT	Inspects ZooKeeper and returns a map containing the name, version, and url for the model referred to by the input parameters	• model_name - The name of the model • model_version - The optional version of the model. If the model version is not specified, the most current version is used.	A map containing the name, version, url for the REST endpoint (fields named name, version, and url). Note that the output of this function is suitable for input into the first argument of MAAS_MODEL_APPLY.
MAAS_MODEL_APPLY	Returns the output of a model deployed via Model as a Service. Note: Results are cached locally 10 minutes.	• endpoint - A map containing name, version, and url for the REST endpoint • function - The optional endpoint path; default is 'apply' • model_args - A dictionary of arguments for the model (these become request params)	The output of the model deployed as a REST endpoint in map form. Assumes REST endpoint returns a JSON map.
MAP_EXISTS	Checks for existence of a key in a map	• key - The key to check for existence • map - The may to check for existence of the key	True if the key is found in the map and false if otherwise
MAP_GET	Gets the value associated with a key from a map	• key - The key • map - The map	The object associated with the key in the map. If no value is associated with the

Function	Description	Input	Returns
		• default - Optionally the default value to return if the key is not in the map	key and default is specified, then default is returned. If no value is associated with the key or default, then null is returned.
MONTH	The number representing the month. The first month, January, has a value of 0.	• dataTime - The datetime as a long representing the milliseconds since UNIX epoch	The current month (0-based)
PROFILE_FIXED	The profile periods associated with a fixed lookback starting from now	• durationAgo - How long ago should values be retrieved from? • units - The units of 'durationAgo' • config_overrides - Optional - Map (in curly braces) of name:value pairs, each overriding the global config parameter of the same name. Default is the empty Map, meaning no overrides.	The selected profile measurement timestamps. These are ProfilePeriod objects.
PROFILE_GET	Retrieves a series of values from a stored profile	• profile - The name of the profile • entity - The name of the entity • periods - The list of profile periods to grab. These are ProfilePeriod objects. • groups_list -Optional - Must correspond to the 'groupBy' list used in profile creation - List (in square brackets) of groupBy values used to filter the profile. Default is the empty list, meaning groupBy was not used when creating the profile. • config_overrides - Optional - Map (in curly braces) of name:value pairs, each overriding the global config parameter of the same name. Default is the empty Map, meaning no overrides.	The profile measurements
PROFILE_WINDOW	The profiler periods associated with a window selector statement from an optional reference timestamp.	• WindowSelector - The statement specifying the window to select. • now - Optional - The timestamp to use for now. • config_overrides - Optional - Map (in curly braces) of name:value pairs, each overriding the	Returns: The selected profile measurement periods. These are ProfilePeriod objects.

Function	Description	Input	Returns
		global config parameter of the same name. Default is the empty Map, meaning no overrides.	
PROTOCOL_TO_NAME	Converts the IANA protocol number to the protocol name	• IANA number	The protocol name associated with the IANA number
REGEXP_MATCH	Determines whether a regex matches a string	• input -String to split • delim - String delimiter	List of strings
SPLIT	Splits the string by the delimiter	• inputs - String to split • delim - String delimiter	List of strings
STARTS_WITH	Determines whether a string starts with a prefix	• string -the string to test • prefix - The proposed prefix	True if the string starts with the specified prefix and false if otherwise
STATS_ADD	Add one or more input values to those that are used to calculate the summary statistics	• stats - The Stellar statistics object. If null, then a new one is initialized • value+ - One or more numbers to add	A Stellar statistics object
STATS_BIN	Computes the bin that the value is in based on the statistical distribution.	• stats - The Stellar statistics object • value - The value to bin • bound? - A list of percentile bin bounds (excluding min and max) or a string representing a known and common set of bins. For convenience, we have provided QUARTILE, QUINTILE, and DECILE which you can pass in as a string arg. If this argument is omitted, then we assume a Quartile bin split.	Which bin N the value falls in such that bound(N-1) < value <= bound(N). No min and max bounds are provided, so values smaller than the 0'th bound go in the 0'th bin, and values greater than the last bound go in the M'th bin.
STATS_COUNT	Calculates the count of the values accumulated (or in the window if a window is used)	• stats - The Stellar statistics object	The count of the values in the window or NaN if the statistics object is null
STATS_GEOMETRIC_MEAN	Calculates the geometric mean of the accumulated values (or in the window if a window is used). See http://commons.apache.org.proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The geometric mean of the values in the window or NaN if the statistics object is null
STATS_INIT	Initializes a statistics object	• window_size - The number of input data values to maintain in a rolling window in memory. If window_size is equal to 0, then no rolling window is maintained. Using no rolling window is less memory intensive, but cannot calculate certain	A Stellar statistics object

Function	Description	Input	Returns
		statistics like percentiles and kurtosis.	
STATS_KURTOSIS	Calculates the kurtosis of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The kurtosis of the values in the window or NaN if the statistics object is null
STATS_MAX	Calculates the maximum of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The maximum of the accumulated values in the window or NaN if the statistics object is null
STATS_MEAN	Calculates the mean of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The mean of the values in the window or NaN if the statistics objects is null
STATS_MERGE	Merges statistics objects	• statistics - A list of statistics providers	A Stellar statistics object
STATS_MIN	Calculates the minimum of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The minimum of the accumulated values in the window of NaN if the statistics object is null
STATS_PERCENTILE	Computes the p'th percentile of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object • p - A double where 0<=1 representing the percentile	The p'th percentile of the data or NaN if the statistics object is null
STATS_POPULATION_VARIANCE	Calculates the population variance of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The population variance of the values in the window of NaN if the statistics object is null
STATS_QUADATIC_MEAN	Calculates the quadratic mean of the accumulated values (or in the window if the window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The quadratic mean of the values in the window or NaN if the statistics object is null
STATS_SD	Calculates the standard deviation of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The standard deviation of the values in the window or NaN if the statistics object is null
STATS_SKEWNESS	Calculates the skewness of the accumulated values (or in the window if a window is used). See http://commons.apache.org/	• stats - The Stellar statistics object	The skewness of the values in the window of NaN if the statistics object is null

Function	Description	Input	Returns
	proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics		
STATS_SUM	Calculates the sum of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The sum of the values in the window or NaN if the statistics object is null
STATS_SUM_LOGS	Calculates the sum of the (natural) log of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The sum of the (natural) log of the values in the in window or NaN if the statistics object is null
STATS_SUM_SQUARES	Calculates the sum of the squares of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The sum of the squares of the values in the window or NaN if the statistics object is null
STATS_VARIANCE	Calculates the variance of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The variance of the values in the window or NaN if the statistics object is null
STRING_ENTROPY	Computes the base-2 shannon entropy of a string.	input - string	The base-2 shannon entropy of the string (https://en.wikipedia.org/wiki/Entropy_(information_theory)#Definition). The unit of this is bits.
SYSTEM_ENV_GET	Returns the value associated with an environment variable	• env_var -Environment variable name to get the value for	String
SYSTEM_PROPERTY_GET	Returns the value associated with a Java system property	• key - Property to get the value for	String
TO_DOUBLE	Transforms the first argument to a double precision number	• Input - Object of string or numeric type	Double version of the first argument
TO_EPOCH_TIMESTAMP	Returns the epoch timestamp of the dataTime in the specified format. If the format does not have a timestamp and you wish to assume a given timestamp, you may specify the timezone optionally.	• dateTime -DateTime in string format • format - DataTime format as string • timezone - Optional timezone in a string format	Epoch timestamp
TO_INTEGER	Transforms the first argument to an integer	• Input - Object of string or numeric type	Integer version of the first argument
TO_LOWER	Transforms the first argument to a lowercase string	• Input -String	String
TO_STRING	Transforms the first argument to a string	• Input - Object	String
TO_UPPER	Transforms the first argument to an uppercase string	• Input -String	Uppercase string

Function	Description	Input	Returns
TRIM	Trims white space from both sides of a string	• Input -String	String
URL_TO_HOST	Extract the hostname from a URL	• url - URL in string form	The hostname from the URL as a string (for example URL_TO_HOST('http://www.yahoo.com/foo') would yield 'www.yahoo.com'
URL_TO_PATH	Extract the path from a URL	• url - URL in string form	The path from the URL as a string (for example URL_TO_PATH('http://www.yahoo.com/foo') would yield 'foo'
URL_TO_PORT	Extract the port from a URL. If the port is not explicitly stated in the URL, then an implicit port is inferred based on the protocol.	• url - URL in string form	The port used in the URL as an integer (for example URL_TO_PORT('http://www.yahoo.com/foo') would yield 80)
URL_TO_PROTOCOL	Extract the protocol from a URL	• url - URL in string form	The protocol from the URL as a string (for example URL_TO_PROTOCOL('http://www.yahoo.com/foo') would yield 'http'
WEEK_OF_MONTH	The numbered week within the month. The first week within the month has a value of 1.	• dataTime -The datetime as a long representing the milliseconds since UNIX epoch	The numbered week within the month
WEEK_OF_YEAR	The numbered week within the year. The first week in the year has a value of 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The numbered week within the year
YEAR	The number representing the year	• dateTime -The datetime as a long representing the milliseconds since UNIX epoch	The current year