Anomaly detection in login behavior with Apache Metron and a single-node Apache Hadoop cluster

## Problem Statement

Anomalous login behavior is one of the indicators of malicious activity in a network. The same user logging into an internet-connected system from geographically remote locations within a short time period may be an indicator of malicious behavior.

We will encode potentially malicious events by distance from the geographic centroid of user’s historic logins comparing to all users. Consider analysis of geographic data of users who use an internet-connected corporate system. We can calculate a median distance from the central geographic location from where all users’ were logging in. For example, median distance is 50 miles, the standard deviation is 15 miles, and some user logged in from the distance of 1700 miles, we can flag such behavior as suspicious. On the other hand, it can be a legitimate user who connected through a VPN or a proxy server, which changed user’s location data. Thus, an anomalous geographic location will be a single red flag, which can potentially be promoted to an alert, should other evidence of suspicious behavior be revealed.

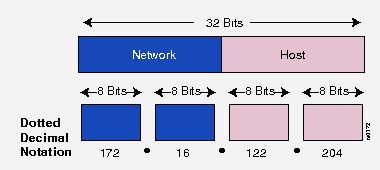
## Task

We will construct artificial data simulating two users logging into a system every second from various hosts. Each user’s locations share the same first two octets of IP addresses but the third and fourth octets will be assigned randomly. To trigger a red flag, we will inject a data point showing that a user logged from a foreign IP address. We will be using IPv4 addresses.

## IPv4 Address Format

The 32-bit IPv4 address is grouped 8 bits at a time, each group of 8 bits is an octet. Each of the four octets are separated by a dot, and represented in decimal format, this is known as dotted decimal notation. Each bit in an octet has a binary weight (128, 64, 32, 16, 8, 4, 2, 1). The minimum value for an octet is 0 (all bits set to 0), and the maximum value for an octet is 255 (all bits set to 1).

The following figure shows the basic format of a typical IPv4 address:



**IP Address Classes:**

IP addressing supports three different commercial address classes; Class A, Class B, and Class C.

In a class A address, the first octet is the network portion, so the class A address of, 10.1.25.1, has a major network address of 10. Octets 2, 3, and 4 (the next 24 bits) are for the hosts. Class A addresses are used for networks that have more than 65,536 hosts (actually, up to 16,581,375 hosts!).

In a class B address, the first two octets are the network portion, so the class B address of, 172.16.122.204, has a major network address of 172.16. Octets 3 and 4 (the next 16 bits) are for the hosts. Class B addresses are used for networks that have between 256 and 65,536 hosts.

In a class C address, the first three octets are the network portion. The class C address of, 193.18.9.45, has a major network address of 193.18.9. Octet 4 (the last 8 bits) is for hosts. Class C addresses are used for networks with less than 254 hosts.

## IP

The Internet Protocol (IP) is the method or protocol by which data is sent from one computer to another on the Internet. Each computer (known as a host) on the Internet has at least one IP address that uniquely identifies it from all other computers on the Internet.

## IPv4

IPv4 stands for Internet Protocol version 4. It is the underlying technology that makes it possible for us to connect our devices to the web. Whenever a device accesses the Internet (whether it's a PC, Mac, smartphone or other device), it is assigned a unique, numerical IP address such as 99.48.227.227. To send data from one computer to another through the web, a data packet must be transferred across the network containing the IP addresses of both devices. Without IP addresses, computers would not be able to communicate and send data to each other. It's essential to the infrastructure of the web.

IPv6

IPv6 is the sixth revision to the Internet Protocol and the successor to IPv4. It functions similarly to IPv4 in that it provides the unique, numerical IP addresses necessary for Internet-enabled devices to communicate. However, it does have a major difference: it utilizes 128-bit addresses. Therefore, it can support 2128 Internet addresses or 340,282,366,920,938,463,463,374,607,431,768,211,456 of them to be exact. The switch has been in progress for the last decade. However, progress has been slow — only a small fraction of the Web has switched over to the new protocol. In addition, IPv4 and IPv6 essentially run as parallel networks—exchanging data between these protocols requires special gateways.

|  |  |  |
| --- | --- | --- |
|  | **Internet Protocol version 4 (IPv4)** | **Internet Protocol version 6**  **(IPv6)** |
| Deployed | 1981 | 1999 |
| Address Size | 32-bit number | 128-bit number |
| Address Format | Dotted Decimal Notation: 132.114.251.53 | Hexadecimal Notation:  3FFE:F200:0234:AB00:0123:4567:8901:ABCD |
| Prefix Notation | 132.114.0.0/24 | 3FFE:F200:0234::/48 |
| Number of Available Addresses | 232 = 4,294,967,296 | 2128 = 340,282,366,920,938,463,463,374,607,431,768,211,456 |

## Apache Hadoop

Hadoop is an open-source big data ecosystem allowing distributed processing of large datasets, cluster scalability from one node to thousands. Hadoop supports data redundancy and can be deployed on premises, remote physical locations or virtual environments, e.g. VMware, or in a public cloud, e.g. AWS, Microsoft Azure, etc. Hadoop distributions are available from several vendors, e.g. Hortonworks, Cloudera, etc. Hadoop distributions come with a variety of applications: YARN, Ambari, Hive, Spark, Storm, Hbase, Kafka, Oozie, etc. Apache Metron is included into the Hortonworks Cybersecurity Pack (HCP), containing Metron, Kivana and Elasticsearch. HCP can be integrated as an add-on to a larger distribution of Hadoop—Hortonworks Data Platform—with the help of an Ambari management pack.

## Apache Metron

Metron is a cyber security application framework that allows to ingest, process and store diverse security data feeds at scale. Metron can detect cyber anomalies and enable a rapid response. Its four key features are:

1. Security Data Lake / Data Vault for cost-effective storing enriched telemetry data.

2. Pluggable Framework supporting various fomats, such as pcap, netflow, bro, snort, fireye, sourcefire, etc.

3. Metron is a SIEM—Security Information and Event Management system.

4. Threat Intelligence Platform containing anomaly detection and machine learning algorithms for real-time data analytics.

**Assignment**

Perform the assignment following the steps described below. Make screenshots when asked, then insert the screenshots into a new document (using Microsoft Word or similar software). Add a short description of each screenshot. After completion of the assignment, submit the document via WebCampus.

## Step 1. Obtain Putty—an SSH client for Windows.

This lab focuses on Windows users. Linux and Max users may complete the assignment as well.

**Windows users**. Download the Putty SSH client from <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html> . Download either 32bit or 64 bit version of putty.exe from the Alternative Binary Files section.

**Linux users** may use a command-line SSH implementation. A guide is available at <https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server>

**Mac users** may use Putty for Mac OS or other SSH client. Guides are available at <https://www.ssh.com/ssh/putty/mac/> and <https://support.rackspace.com/how-to/logging-in-with-an-ssh-private-key-on-linuxmac/>

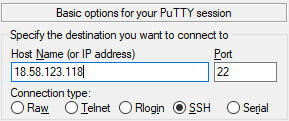
## Step 2. Obtain a private key.

Download a private RSA key from WebCampus. You can find **metron-cluster.ppk** in the Course Documents section. Save the key at a location on your computer. You can create a new folder, for example, C:\tmp\keys\.

Note: Normally, private keys are not shared. This is done to simplify the assignment and reduce your workload.

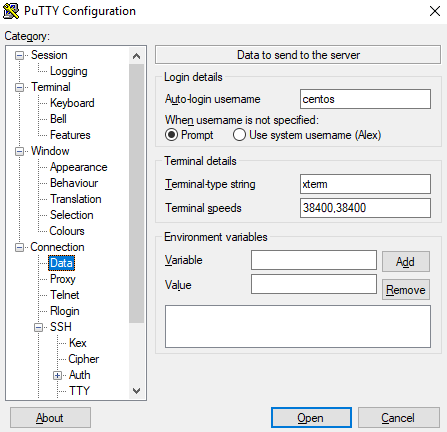
## Step 3. Create a new SSH connection in Putty.

Start Putty. First, enter the IPv4 address of your single-node Hadoop cluster. This IP address was sent in a separate email. Make sure SSH is selected.

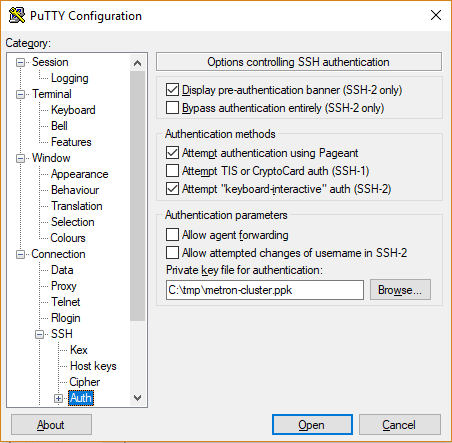


In Putty, use left-side tree-like navigation panel.

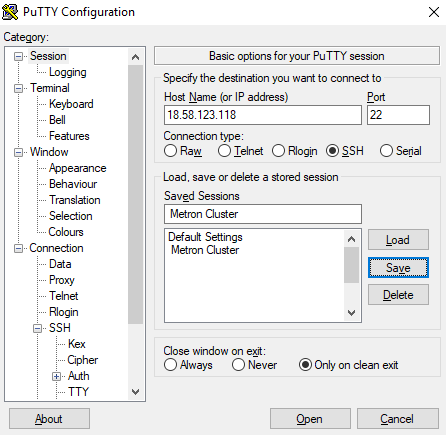
Under Connection, click Data and enter centos in the Auto-login username.



Under Connection, expand SSH and click on the Auth item. Then in the right panel, click the Browse button and locate the previously downloaded RSA key.



Do NOT click the Open button yet. Scroll the left-hand-side navigation bar to the top and click the Session item. Next, in the Session box type Metron Cluster and save the connection by clicking the Save button.

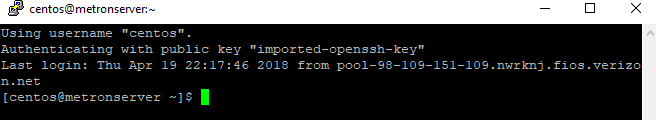


Make a screenshot

The connection should be saved successfully.

## Step 4. Establish a connection.

Start Putty. Load the previously stored Metron Cluster connection. For this, click on Metron Connection and click the Load button. Click the Open button. The connection should be established. A similar window should be displayed.



Make a screenshot

## Step 5. Environment variables.

METRON\_HOME variable should point to Metron’s home directory.

ZOOKEEPER variable contains host(s) and port(s) of cluster nodes to establish a connection with Zookeper in a comma-separated format, e.g. node1:2181,node2:2181.  Since we use a single-node cluster, there will be a single host.

BROKERLIST – a variable containing a comma separated list of nodes and ports, where Kafka brokers are running, e.g. node1:6667,node2:6667. Since we use a single-node cluster, there will be a single host.

ES\_HOST – a variable containing a host and port where Elasticsearch master is up and running, e.g. node1:9200.

We will set these variables in the environment file, which is used in CentOS 7 Linux operating system to set variable values when a computer boots. Thus, if a connection is interrupted for any reason, it will not be necessary to set these values manually anymore.

Start the vi text editor as a super user. To do so, type the following command and press Enter:

sudo vi /etc/environment

Now, you need to press the i button on your keyboard to switch the vi editor to the Insert mode.

Either type the four lines below, or copy and paste then into the vi editor. To paste the date into vi, simply right click somewhere in the vi window.

METRON\_HOME="/usr/metron/0.4.3"

ZOOKEEPER="metronserver.localdomain:2181"

ES\_HOST="metronserver.localdomain:9200"

BROKERLIST="metronserver.localdomain:6667"

First, press the Esc (Escape) button on your keyboard to exit the Insert mode. Then save the file using the following key combination

:wq

And press Enter.

To confirm that variables’ values have been set successfully, close the Putty window, then start Putty again and connect to your single-node cluster. Type the following commands (and hit Enter after each line) to see the values stored in these environment variables.

echo $METRON\_HOME

echo $ZOOKEEPER

echo $ES\_HOST

echo $BROKERLIST

Make a screenshot

## Step 6. Pull Zookeper configuration.

Zookeper is a coordinator within an Apache Hadoop cluster. ZooKeeper is a centralized service for maintaining configuration, naming, providing distributed synchronization, and providing group services in distributed applications. Zookeper service itself is distributed and highly reliable.

Pull the Zookeeper configuration information into a local folder using the following command:

sudo $METRON\_HOME/bin/zk\_load\_configs.sh --mode PULL -z $ZOOKEEPER -o $METRON\_HOME/config/zookeeper/ -f

Make a screenshot

## Step 7. Configure Metron Profiler

This step involves several big data applications from Hadoop Ecosystem: Metron, Strom, Kafka and Elasticsearch.

## 7.1. Metron Profiler.

A set of slides have been presented on Apache Metron earlier. This step uses the Profiler module of Metron application. Profiler is a feature extraction mechanism that can generate a profile describing the behavior of an entity. An entity might be a server, user, subnet or application. Once a profile has been generated defining what normal behavior looks-like, models can be built that identify anomalous behavior.

Any field contained within a message can be used to generate a profile. A profile can even be produced by combining fields that originate in different data sources. A user has considerable power to transform the data used in a profile by leveraging the Stellar language. A user only needs to configure the desired profiles and ensure that the Profiler topology is running.

Detailed description of Metron Profiler is available at <https://metron.apache.org/current-book/metron-analytics/metron-profiler/index.html>

The Metron Profiler is implemented as an Apache Storm topology using Apache Storm elements, known as bolts and spouts.

Metron Profiler commonly uses one or a set of the following elements:

**KafkaSpout** - A spout that consumes messages from a single Kafka topic. In most cases, the Profiler topology will consume messages from the indexing topic. This topic contains fully enriched messages that are ready to be indexed. This ensures that profiles can take advantage of all the available data elements.

**ProfileSplitterBolt** - The bolt responsible for filtering incoming messages and directing each to the one or more downstream bolts that are responsible for building a profile. Each message may be needed by 0, 1 or even many profiles. Each emitted tuple contains the ‘resolved’ entity name, the profile definition, and the input message.

**ProfileBuilderBolt** - This bolt maintains all of the state required to build a profile. When the window period expires, the data is summarized as a ProfileMeasurement, all state is flushed, and the ProfileMeasurement is emitted. Each instance of this bolt is responsible for maintaining the state for a single Profile-Entity pair.

**HBaseBolt** - A bolt that is responsible for writing to HBase. Most profiles will be flushed every 15 minutes or so. If each ProfileBuilderBolt were responsible for writing to HBase itself, there would be little to no opportunity to optimize these writes. By aggregating the writes from multiple Profile-Entity pairs these writes can be batched, for example.

## 7.2. Apache Storm.

Apache Storm is a free and open source distributed real-time computation system for stream processing. Storm serves for real-time distributed processing of data streams. Storm may be applied to real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, etc. Storm process data at high rates, a benchmark demonstrated more than a million tuples processed by Storm per second per node.

Storm real-time processing is packaged into **topologies**, which run persistently. A **topology** is a graph (network) of spouts and bolts that are connected with stream groupings. The **stream** is the core abstraction in Storm. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed mode. A **spout** is a source of streams in a Storm **topology**. Generally, spouts will read tuples from an external source and emit them into the topology. All processing in topologies is done in **bolts**. **Bolts** can do anything from filtering, functions, aggregations, joins, talking to databases, and more. A **stream grouping** defines how a stream should be partitioned among Storm bolt's tasks.

More details on Apache Storm are available at <http://storm.apache.org/releases/1.0.6/Concepts.html>

## 7.3. Apache Kafka

Apache Kafka is a distributed streaming platform with three key capabilities:

* Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
* Store streams of records in a fault-tolerant durable way.
* Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

* Building real-time streaming data pipelines that reliably get data between systems or applications
* Building real-time streaming applications that transform or react to the streams of data

The Kafka cluster stores streams of records in categories called topics. Each record consists of a key, a value, and a timestamp. Kafka has four core modules:

* The Producer API allows an application to publish a stream of records to one or more Kafka topics.
* The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
* The Streams API allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
* The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

More details on Kafka are available at <https://kafka.apache.org/intro>

## 7.4. Apache HBase.

Apache HBase is a NoSQL database, which operates within the Hadoop ecosystem. HBase is open-source non-relational distributed database, which originates from the Google Bigtable process. One of the main features of HBase is that it works in column-oriented fashion, grouping table columns into column families. Each column family is stores as a separate data file. More details about HBase are available at <https://mapr.com/blog/in-depth-look-hbase-architecture/>

## 7.5. Elasticsearch

Elasticsearch is an open-source distributed search and analytics engine, which uses REST architecture. Elasticsearch is commonly used for is commonly used for log analytics, full-text search, and operational intelligence use cases. Elasticsearch can be connected with Kibana, a visualization tool, to provide near-real time analytics using large volumes of data. More details about Elasticsearch are available at <https://aws.amazon.com/elasticsearch-service/what-is-elasticsearch/>

## 7.6. Configuration

First, we’ll configure the Metron Profiler to emit a tuple every 1 minute.

Login into your Apache Hadoop cluster using the Ambari web interface. To do so, navigate to the following address in your browser. Substitute ipaddress with the IPv4 address you received in a separate email for this assignment:

<http://ipaddress:8080>

You will be prompted for login and password. Use the following credentials:

User name: admin

Password: admin

In Ambari, use the left-hand-side navigation panel. Scroll down and click the Metron item.



In the top section of the screen, locate and click the Configs tab.



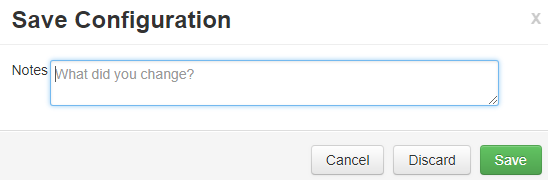
Locate and click the Profiler tab.



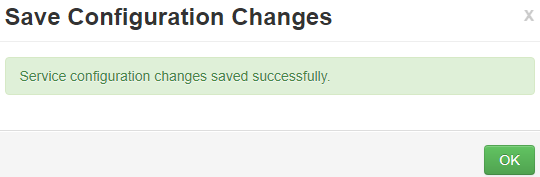
In the Profiler settings, locate the Period duration box. Change the value to 1. Then click the green Save button to store the change.



Ambari will display a Save Configuration window. It is a good practice to make a comment on changes. You can enter “Set Metron Profiler period duration = 1” or a similar message and click the Save button:



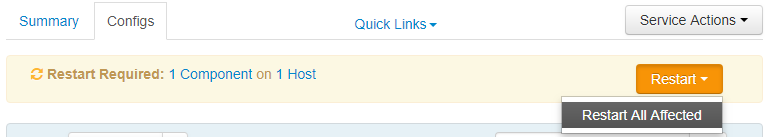
A window confirming the change has been saved will pop up. Click OK.



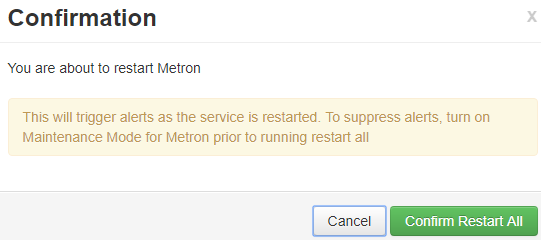
Switch to the Putty window, and execute the following command needed for a successful restart (type it and press Enter):

sudo storm kill profiler

In a few moments, an orange Restart button will appear on the right. Click it, and then click Restart All Affected.

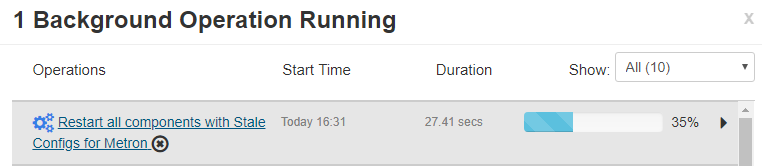


Confirm your action by clicking the Confirm Restart All button in a pop up window.

  
Notice, that a blue bubble showing that one operation is pending appeared in the top left section of the interface. Click it.



A new window, showing the operation progress will be displayed. Wait until it completes.



Make a screenshot

We need to make one more configuration adjustment. Switch to the Putty window and execute the following two commands. It is possible to copy this text and then right click in the Putty window to paste it.

sudo sed -i 's/master: false/master: true/g' /etc/elasticsearch/elasticsearch.yml

sudo service elasticsearch restart

## Step 8. Create Data Generator

We will create a new sensor for artificial data and name it **auth**. A synthetic data generator is needed to feed it. A process is needed to generate authentication events per second for a set of users where IPv4 addresses are randomly chosen while first and second octets of these IP addresses match. The Python code below will do this.

Edit the ~/gen\_data.py  file and paste the following into it (Note: ~ in Linux denotes a user’s home directory). To do so, execute the following command in Putty:

sudo vi ~/gen\_data.py

Press i button on the keyboard in order to switch to the Insert mode in the vi text editor.

Paste the following Python code into the file:

#!/usr/bin/python

import random

import sys

import time

domains = { 'user1' : '173.90', 'user2' : '156.33' }

def get\_ip(base):

return base + '.' + str(random.randint(1,255)) + '.' + str(random.randint(1, 255))

def main():

freq\_s = 1

while True:

user='user' + str(random.randint(1,len(domains)))

epoch\_time = int(time.time())

ip=get\_ip(domains[user])

print user + ',' + ip + ',' + str(epoch\_time)

sys.stdout.flush()

time.sleep(freq\_s)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Make a screenshot

Press the Esc (Escape) button on your keyboard to exit the Insert mode. Then save the file using the following key combination

:wq

And press Enter.

## Step 9. Create the auth Parser

The code below in JSON format will be used to create the **auth** parser. It is designed to parse three attributes: user name, IPv4 address of a device used to login into the system and a timestamp of the event. The parser will also add geo hash for each IP address. The geo hash is added as an identifier of a geographic location. This is done using Stellar – a programming language built into Metron.

To create this parser, create a new file using the vi editor:

sudo vi $METRON\_HOME/config/zookeeper/parsers/auth.json

Press i on the keyboard to switch to the Insert mode and paste the following code in JSON format:

{

"parserClassName" : "org.apache.metron.parsers.csv.CSVParser"

,"sensorTopic" : "auth"

,"parserConfig" : {

"columns" : {

"user" : 0,

"ip" : 1,

"timestamp" : 2

}

}

,"fieldTransformations" : [

{

"transformation" : "STELLAR"

,"output" : [ "hash" ]

,"config" : {

"hash" : "GEOHASH\_FROM\_LOC(GEO\_GET(ip))"

}

}

]

}

Make a screenshot

Press the Esc (Escape) button on your keyboard to exit the Insert mode. Then save the file using the following key combination

:wq

And press Enter.

Create a new Kafka topic by executing the following command. This should be entered in a single line then press enter:

sudo /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper $ZOOKEEPER --create --topic auth --partitions 1 --replication-factor 1

Make a screenshot

## Step 10. Create Profiles for Enrichment

Two profiles are needed to accomplish the assignment.

First, **locations\_by\_user** is a collection of geo hashes of the locations where a user logged in from.

Second, **geo\_distribution\_from\_centroid** - the statistical distribution of the distance between a login location and the geographic centroid of the user’s previous logins from the past two minutes. In a real-world application, this time span would be longer.

We will store bothe locations\_by\_user and geo\_distribution\_from\_centroid in the same file, profiler.json. Using the vi editor, create a new file profiler.json in the $METRON\_HOME/config/zookeeper/ directory.

Execute the command:

sudo vi $METRON\_HOME/config/zookeeper/profiler.json

Press i on the keyboard to switch to the Insert mode. Then paste the following (paste is done with a right click of a mouse):

{

"profiles": [

{

"profile": "geo\_distribution\_from\_centroid",

"foreach": "'global'",

"onlyif": "exists(geo\_distance) && geo\_distance != null",

"init" : {

"s": "STATS\_INIT()"

},

"update": {

"s": "STATS\_ADD(s, geo\_distance)"

},

"result": "s"

},

{

"profile": "locations\_by\_user",

"foreach": "user",

"onlyif": "exists(hash) && hash != null && LENGTH(hash) > 0",

"init" : {

"s": "MULTISET\_INIT()"

},

"update": {

"s": "MULTISET\_ADD(s, hash)"

},

"result": "s"

}

]

}

Make a screenshot

Press the Esc (Escape) button on your keyboard to exit the Insert mode. Then save the file using the following key combination

:wq

And press Enter.

## Step 11. Enrich authentication Events

We will need to enrich the authentication records for use in the threat triage. The following attributes should be added:

**geo\_distance**: the distance between the current geohash and the geographic centroid

**geo\_centroid**: the geographic centroid

**dist\_median**: the median distance between a user’s login location and the geographic centroid

**dist\_sd**: the standard deviation of the distance between a user’s login location and the geographic centroid

**geo\_outlier**: whether geo\_distance is more than 5 standard deviations from the median of all users

We need to set up a triage rule associating a score and setting an alert if geo\_outlier is true. In an actual setting, this rule should be more complex than in this assignment.

Edit $METRON\_HOME/config/zookeeper/enrichments/auth.json:

sudo vi $METRON\_HOME/config/zookeeper/enrichments/auth.json

Then press i to switch to the Insert mode and paste the following code in JSON format with Stellar transformations:

{

"enrichment": {

"fieldMap": {

"stellar" : {

"config" : [

"geo\_locations := MULTISET\_MERGE( PROFILE\_GET( 'locations\_by\_user', user, PROFILE\_FIXED( 2, 'MINUTES')))",

"geo\_centroid := GEOHASH\_CENTROID(geo\_locations)",

"geo\_distance := TO\_INTEGER(GEOHASH\_DIST(geo\_centroid, hash))",

"geo\_locations := null"

]

}

}

,"fieldToTypeMap": { }

},

"threatIntel": {

"fieldMap": {

"stellar" : {

"config" : [

"geo\_distance\_distr:= STATS\_MERGE( PROFILE\_GET( 'geo\_distribution\_from\_centroid', 'global', PROFILE\_FIXED( 2, 'MINUTES')))",

"dist\_median := STATS\_PERCENTILE(geo\_distance\_distr, 50.0)",

"dist\_sd := STATS\_SD(geo\_distance\_distr)",

"geo\_outlier := ABS(dist\_median - geo\_distance) >= 5\*dist\_sd",

"is\_alert := exists(is\_alert) && is\_alert",

"is\_alert := is\_alert || (geo\_outlier != null && geo\_outlier == true)",

"geo\_distance\_distr := null"

]

}

},

"fieldToTypeMap": { },

"triageConfig" : {

"riskLevelRules" : [

{

"name" : "Geographic Outlier",

"comment" : "Determine if the user's geographic distance from the centroid of the historic logins is an outlier as compared to all users.",

"rule" : "geo\_outlier != null && geo\_outlier",

"score" : 10,

"reason" : "FORMAT('user %s has a distance (%d) from the centroid of their last login is 5 std deviations (%f) from the median (%f)', user, geo\_distance, dist\_sd, dist\_median)"

}

],

"aggregator" : "MAX"

}

}

}

Make a screenshot

Press the Esc (Escape) button on your keyboard to exit the Insert mode. Then save the file using the following key combination

:wq

And press Enter.

## Step 12. Execute Demonstration

At this point, all the necessary configuration is complete. We need to push the configuration data to Zookeeper, which serves as a coordinator in this setup. Execute the following command in Putty terminal.It should be entered in one line:

sudo $METRON\_HOME/bin/zk\_load\_configs.sh --mode PUSH -z $ZOOKEEPER -i $METRON\_HOME/config/zookeeper/

Make a screenshot

Start the parser using the following command:

sudo $METRON\_HOME/bin/start\_parser\_topology.sh -k $BROKERLIST -z $ZOOKEEPER -s auth

Make a screenshot

Push synthetic data generated by the Python program we set up into the **auth** topic with the following command, which must be entered in a single line:

sudo python ~/gen\_data.py |

/usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --broker-list $BROKERLIST --topic auth

Make a screenshot

Wait for five minutes and interrupt the Kafka producer we started in the line above by pressing Ctrl C combination on keyboard (press and hold the Ctrl button and press C button)

Push another record indicating that user1 logged from a foreign IP address (109.252.227.173) into the topology. This command must be entered in a single line in Putty terminal:

sudo echo -e "import time\nprint 'user1,109.252.227.173,'+str(int(time.time()))" | python | /usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --broker-list $BROKERLIST --topic auth

Execute the following to apply Elasticsearch query and look for geographic login outliers in the data we fed in earlier. To do this, copy and paste this code into the command line of the Putty terminal:

sudo curl -XPOST "http://$ES\_HOST/auth\*/\_search?pretty" -d '

{

"\_source" : [ "is\_alert", "threat:triage:rules:0:reason", "user", "ip", "geo\_distance" ],

"query": { "exists" : { "field" : "threat:triage:rules:0:reason" } }

}

'

Make a screenshot

Among other rows, which are false positive results (marked as alerts but in fact being regular records) the correct alert should appear:

{

"\_index" : "auth\_index\_2017.09.07.20",

"\_type" : "auth\_doc",

"\_id" : "f5bdbf76-9d78-48cc-b21d-bc434c96e62e",

"\_score" : 1.0,

"\_source" : {

"geo\_distance" : 7879,

"threat:triage:rules:0:reason" : "user user1 has a distance (7879) from the centroid of their last login is 5 std deviations (334.814719) from the median (128.000000)",

"ip" : "109.252.227.173",

"is\_alert" : "true",

"user" : "user1"

}

}

## References

Metron. Geographic Login Outliers. <https://github.com/apache/metron/tree/master/use-cases/geographic_login_outliers>

IPv6 vs. IPv4. <https://phoenixts.com/blog/ipv6-vs-ipv4/>

IPv4 Address Components. <http://penta2.ufrgs.br/trouble/ts_ip.htm>

Zookeper. <https://wiki.apache.org/hadoop/ZooKeeper>

Putty download <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

Configuring SSH key authentication in Linux. <https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server>

Putty for Mac. <https://www.ssh.com/ssh/putty/mac/>

SSH key authentication for Mac and Linux. <https://support.rackspace.com/how-to/logging-in-with-an-ssh-private-key-on-linuxmac/>

Metron Profiler. <https://metron.apache.org/current-book/metron-analytics/metron-profiler/index.html>

Apache Metron. <https://metron.apache.org/current-book/>

Apache Storm. <http://storm.apache.org/releases/1.0.6/Concepts.html>

Apache Kafka. <https://kafka.apache.org/intro>

Apache HBase. <https://mapr.com/blog/in-depth-look-hbase-architecture/>

Elasticsearch. <https://aws.amazon.com/elasticsearch-service/what-is-elasticsearch/>