

BIG DATA CYBERSECURITY

LAB 2 ANOMALY DETECTION

ANOMALY DETECTION IN LOGIN BEHAVIOR WITH APACHE METRON AND A SINGLE-NODE APACHE HADOOP CLUSTER

Lab Description: 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 a 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 a user logs in from the distance of 1700 miles. This activity may be flagged 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. [1]

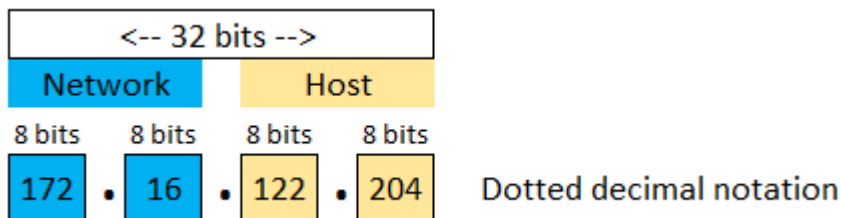
We will construct artificial data simulating two users logging into a system every second from various hosts. Each user's locations share the same first two octets of IP addresses but the third and fourth octets will be assigned randomly. To trigger a red flag, we will inject a data point showing that 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 [2]:



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 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 (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 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 [2, 3].

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 is essential to the infrastructure of the Web [2].

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 2^{128} 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 [2].

	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	2^{32}	2^{128}

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, such as AWS, Google Cloud, or Microsoft Azure.



Hadoop distributions are available from several vendors, such as Cloudera. Hadoop distributions come with a variety of applications: YARN, Ambari, Hive, Spark, Storm, Hbase, Kafka, Oozie, etc. Apache Metron is included into the Cloudera Hortonworks Cybersecurity Pack (HCP), containing Metron, Kibana 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 [4-8].

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 [5]. Its four key features are:

1. Security Data Lake / Data Vault for cost-effective storing enriched telemetry data.
2. Pluggable Framework supporting various formats, such as pcap, netflow, bro, snort, fireeye, sourcefire, and others.
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.

Public and Private Key Pair

The public and private key pair are two cryptographic keys, which are uniquely related. Commonly, each key is represented as a sequence of ASCII characters [9]. This is an example of a public key:

```
---- BEGIN SSH2 PUBLIC KEY ----
```

```
Comment: "imported-openssh-key"
```

```
AAAAB3NzaC1yc2EAAAADAQABAAQCuqBgTeBGUDKHNoxIhfZUFY5KXU  
T5jBJh8Dv4SWQQ1AAX+RqHmLU6fKwSG9V/CKBVv+pe7jMKdehDfFx2KC/d  
ylkequZ1qMd4HoONqORIDUKqFYWMaio9IE0BsCO42yXkmvN7ForqnGWLCH5  
EcLLAnefNRe3SRzzV6kxYyZ5/2EWia0GQG+ZY0BsDyJfkBxcbFCeIPTmaRC9  
DhsZT3NuaKUOSUcUOaXUQAFqO8oocBaIQWy+R6q9OJhhTOsaXvl1cVXc0Y  
blf+O8iRB2BgL8Q/wflUKa+/g2IKLMUsq/bJvCxjsaUS0OwfNTltY9HACdgcRAu  
X6lfxtQUWMcOr1ll
```



---- END SSH2 PUBLIC KEY ----

When needed, the public key is made available to everyone via a publicly accessible repository or directory. The private key is kept confidential by its owner. When information is encrypted with a public key, it can only be decrypted by the corresponding private key from the same key pair. Conversely, what was encrypted by a private key can only be decrypted by the public key belonging to the same key pair [9].

Metron 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 the Metron Profiler is available in [10].

The Metron Profiler is implemented as an Apache Storm topology using Apache Storm elements, known as bolts and spouts [11]. Metron Profiler commonly uses one or a set of the following elements:

KafkaSpout 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.

Profile Splitter Bolt is 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.

Profile Builder 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.

HBase Bolt 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.

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, applying a function, aggregations, joins, database communication, and more. A stream grouping defines how a stream should be partitioned among Storm bolt's tasks. More details on Apache Storm are available in [11]

Apache Kafka is a distributed streaming platform with three key capabilities [12]: (1) Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system, (2) Store streams of records in a fault-tolerant durable way, and (3) 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 in [12].

Apache HBase is a NoSQL database, which operates within the Hadoop ecosystem. HBase is open-source non-relational distributed database, which originates from Google Bigtable. 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 stored as a separate data file. More details about HBase are available in [13, 14].

Elasticsearch is an open-source distributed search and analytics engine, which uses REST architecture. Elasticsearch 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 in [6].

Lab Environment: Perform the assignment following the steps outlined below. Make screenshots when asked, then insert the screenshots into a new document (using Microsoft Word or similar software). Add a short description for each screenshot. After completion of the assignment, submit the document electronically.

Lab Files that are Needed: metron-cluster.ppk.

LAB EXERCISE/STEP 1

Windows users need to obtain Putty—an SSH client for Windows. Linux and Mac users may use the built-in terminal.

Windows users must download the Putty SSH client from [15]. Either 32bit or 64 bit version of putty.exe can be downloaded from the Alternative Binary Files section.

LAB EXERCISE/STEP 2

Obtain a private key



Note: Normally, private keys are not shared. Instead, a user would generate a key pair comprised of public and private keys, copy the public key to a server, and then use the private key to establish a secure SSH connection. This assignment is simplified to reduce student workload.

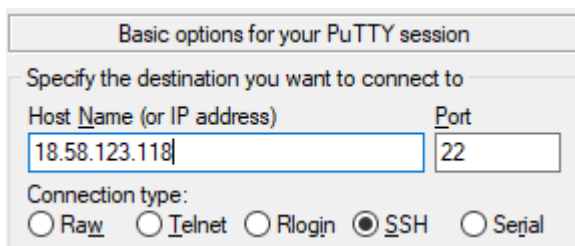
Obtain a private RSA key from the lab assistant. The key file name is metron-cluster.ppk. Save the key at a location on your computer.

LAB EXERCISE/STEP 3

Obtain an IP address of your personal single-node Hadoop cluster. Establish an SSH connection with the server using the private key.

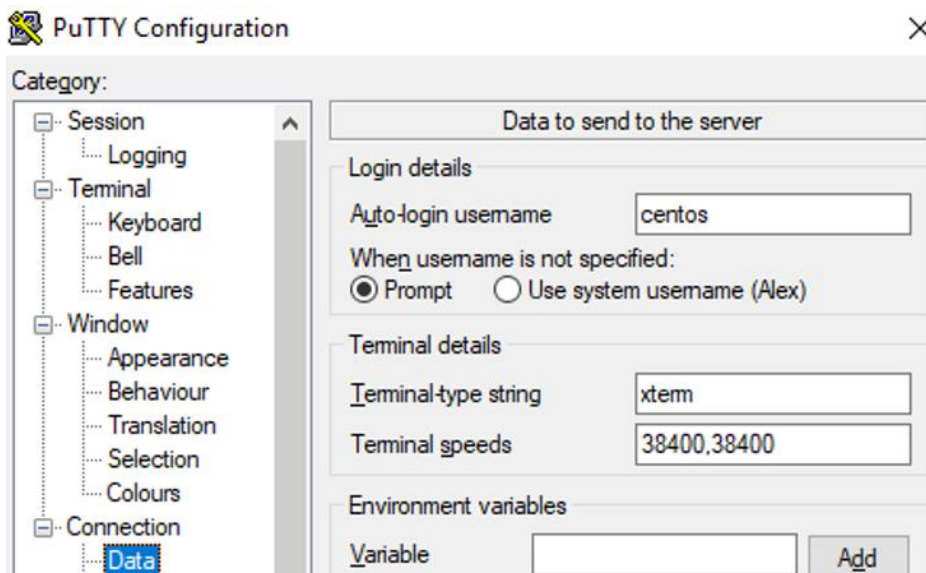
Instructions for Windows Users

Start Putty. First, enter the IPv4 address of your single-node Hadoop cluster. This IP address must be obtained from the lab assistant. Make sure SSH is selected.

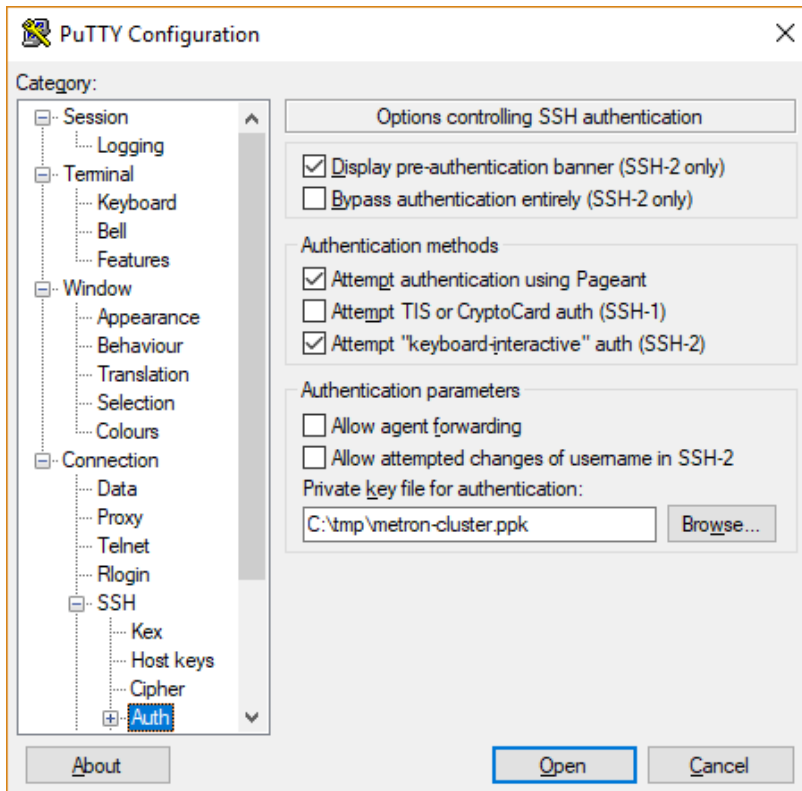


In Putty, use left-side tree-like navigation panel.

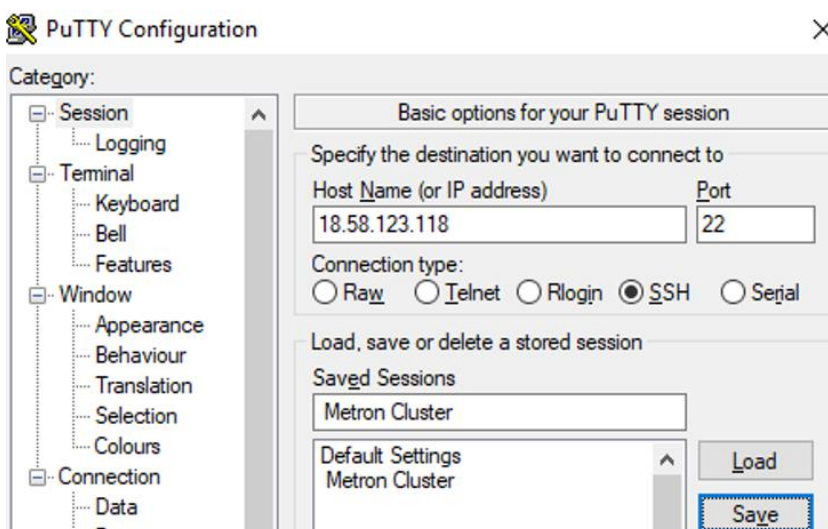
Under Connection, click Data and enter *centos* in the Auto-login username.



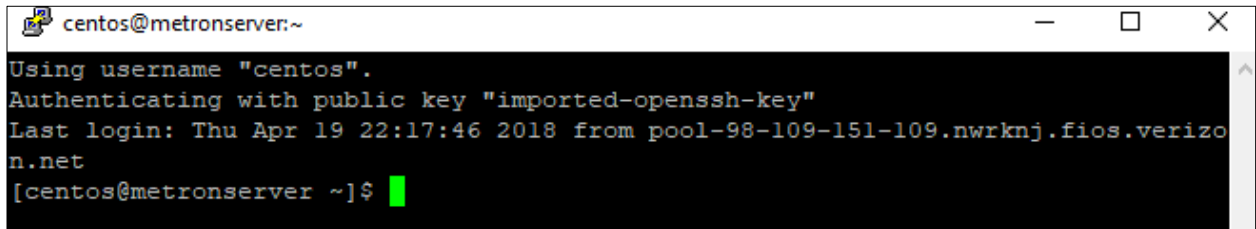
Under Connection, expand SSH and click on the Auth item. Then in the right panel, click the Browse button and locate the previously downloaded RSA key.



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. You can click the Open button to establish an SSH connection with Putty.



When starting Putty next time, the previously stored Metron Cluster connection may be loaded. 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:

A screenshot of a terminal window titled 'centos@metronserver:~'. The terminal shows the following text: 'Using username "centos".', 'Authenticating with public key "imported-openssh-key"', 'Last login: Thu Apr 19 22:17:46 2018 from pool-98-109-151-109.nwrknj.fios.verizon.net', and '[centos@metronserver ~]\$' with a green cursor.

Make a screenshot and store it in the submission document.

Instructions for Mac and Linux Users

To establish a connection with the remote Metron single-node cluster, the obtained private key must be specified. Start a terminal and type the following command:

```
ssh -i /path/to/metron-cluster.ppk centos@ipaddress
```

where /path/to/metron-cluster.ppk must be substituted with the actual path to the key. The *ipaddress* in the command above must be substituted with the actual IPv4 address of the Metron server, which must be obtained from the lab assistant [16].

After establishing the connection, continue to Step 4.

Make a screenshot and store it in the submission document.

LAB EXERCISE/STEP 4

Setup Bash shell environment variables. You will need to execute commands in a terminal connected to the Metron server.

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 Zookeeper in a comma-separated format, e.g. zoonode1:2181,zoonode2:2181. Since we use a single-node cluster, there will be a single host.



BROKERLIST variable stores a comma separated list of nodes and ports, where Kafka brokers are running, e.g. kafkanode1:6667, kafkanode2:6667. 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. elasticnode1:9200.

We will use the environment file, which is used in CentOS 7 Linux operating system to set variable during logon. 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
```

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 them into the vi editor. To paste the data 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"
HDP_HOME="/usr/hdp/current"
METRON_VERSION="0.4.3"
```

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
echo $HDP_HOME
echo $METRON_VERSION
```

Make a screenshot and store it in the submission document.

LAB EXERCISE/STEP 5

Pull Zookeeper Configuration

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

Pull the Zookeeper configuration information into a local folder using the following command (this must be typed as a single line):

```
sudo $METRON_HOME/bin/zk_load_configs.sh --mode PULL -z
$ZOOKEEPER -o $METRON_HOME/config/zookeeper/ -f
```

Make a screenshot and store it in the submission document.

LAB EXERCISE/STEP 6

Configure Metron Profiler

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

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

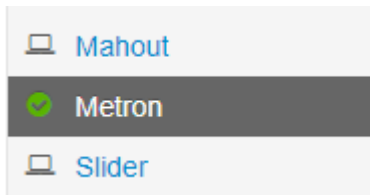
You will be prompted for login and password. Use these credentials:

User name: admin

Password: admin



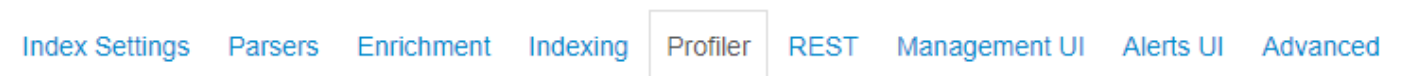
In Ambari, use the left-hand-side navigation panel. Scroll down and click the Metron item.



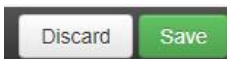
In the top section of the screen, locate and click the Configs tab.



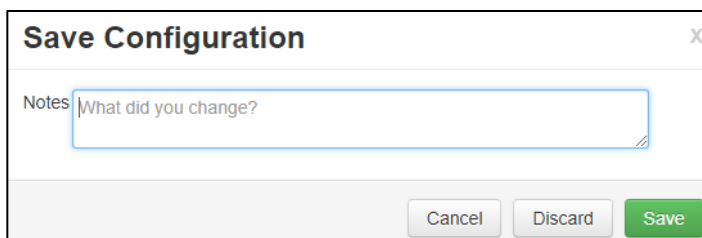
Locate and click the Profiler tab.



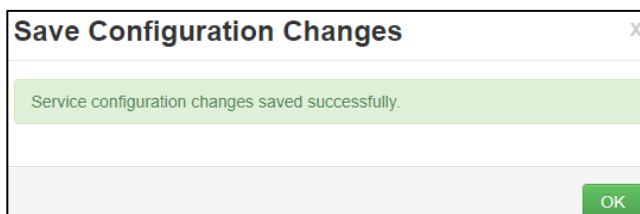
In the Profiler settings, locate the Period duration box. Change the value to 1. Then click the green Save button to store the change.



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.

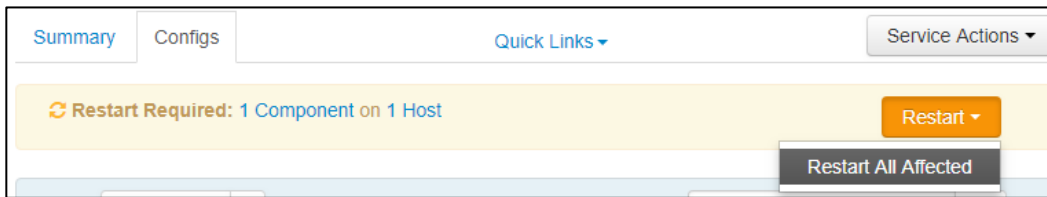


Switch to the Putty window, and execute the following command needed for a successful restart (type it and press Enter):

```
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.



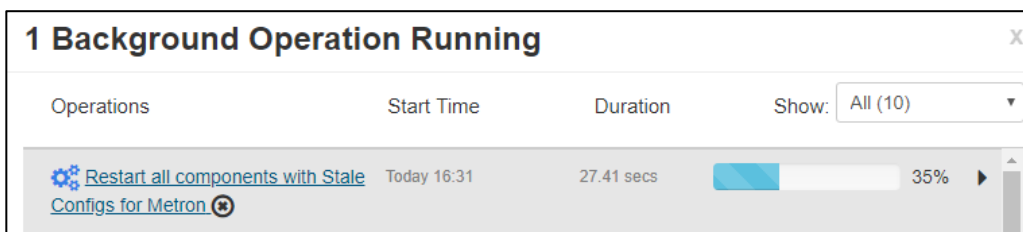
Confirm your action by clicking the Confirm Restart All button in a pop up window.



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 and store it in the submission document.

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
```

LAB EXERCISE/STEP 7

Create a 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 and store it in the submission document.

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.

LAB EXERCISE/STEP 8

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, Metron programming language.

To create the parser, make 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 and store it in the submission document.

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 and store it in the submission document.

LAB EXERCISE/STEP 9

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 both `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 and store it in the submission document.

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.

LAB EXERCISE/STEP 10

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_distribut
ion_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 c
entroid 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 and store it in the submission document.

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.

LAB EXERCISE/STEP 11

Execute Demonstration

At this point, all the necessary configurations are complete. We need to push the configuration data to Zookeeper, which serves as a coordinator in this setup. Execute the following command in the 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 and store it in the submission document.

Start the parser using the following command:

```
sudo $METRON_HOME/bin/start_parser_topology.sh -k $BROKERLIST -z $ZOOKEEPER -s auth
```



Make a screenshot and store it in the submission document.

Push synthetic data generated by the Python program we set up into the auth topic with the following command (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 and store it in the submission document.

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 the 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 and store it in the submission document.

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"
  }
}
```

LAB EXERCISE/STEP 11

Be sure to terminate your AWS Metron VM to avoid budget depletion. Resources created in a cloud environment under your account have associated costs. AWS offers more than 60 products at a free tier with associated free tier usage limits [17].

What to submit

Submit a Word (or other text editor) document with embedded screenshots made as requested in the assignment and a brief description for each screenshot.

References

- [1] GitHub. "Metron. Geographic Login Outliers," [Online]. Available: https://github.com/apache/metron/tree/master/use-cases/geographic_login_outliers. [Accessed Aug 8, 2019].
- [2] IPv4 Address Components. [Online]. Available: http://penta2.ufrgs.br/trouble/ts_ip.htm. [Accessed Aug 8, 2019].
- [3] IPv6 vs. IPv4. [Online]. Available: <https://phoenixts.com/blog/ipv6-vs-ipv4/>. [Accessed Aug 8, 2019].



- [4] The Apache Software Foundation, "Apache Hadoop," [Online]. Available: <https://hadoop.apache.org/>. [Accessed Aug 8, 2019].
- [5] The Apache Software Foundation, "Apache Metron," [Online]. Available: <https://metron.apache.org/current-book/>. [Accessed Aug 8, 2019].
- [6] Elasticsearch. <https://aws.amazon.com/elasticsearch-service/what-is-elasticsearch/>. [Accessed Aug 8, 2019].
- [7] <https://www.elastic.co/products/kibana>. [Accessed Aug 8, 2019].
- [8] Hortonworks, "Hadoop Tutorial – Getting Started with HDP," [Online]. Available: <https://hortonworks.com/tutorial/hadoop-tutorial-getting-started-with-hdp/section/1/>. [Accessed Aug 8, 2019].
- [9] J. Ellingwood, "Configuring SSH key authentication in Linux," *digitalocean.com*, [Online]. Available: <https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server>. [Accessed Aug 8, 2019].
- [10] The Apache Software Foundation, "Metron Profiler," [Online]. Available: <https://metron.apache.org/current-book/metron-analytics/metron-profiler/index.html>. [Accessed Aug 8, 2019].
- [11] The Apache Software Foundation, "Apache Storm," [Online]. Available: <https://storm.apache.org/releases/current/Concepts.html>. [Accessed Aug 8, 2019].
- [12] The Apache Software Foundation, "Apache Kafka. A distributed streaming platform," [Online]. Available: <https://kafka.apache.org/intro>. [Accessed Aug 8, 2019].
- [13] The Apache Software Foundation, "Apache HBase Reference Guide," [Online]. Available: <https://hbase.apache.org/book.html>. [Accessed Aug 8, 2019].
- [14] MapR, "An In-Depth Look at the HBase Architecture," [Online]. Available: <https://mapr.com/blog/in-depth-look-hbase-architecture/>. [Accessed Aug 8, 2019].
- [15] S. Tatham, "Download PuTTY: latest release," April, 2017. [Online]. Available: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>. [Accessed May 15, 2017].
- [16] B. Ohearn, "Rackspace Support Network. Log in with an SSH private key on Linux and Mac," *support.rackspace.com*. [Online]. Available: <https://support.rackspace.com/how-to/logging-in-with-an-ssh-private-key-on-linuxmac/>. [Accessed Aug 8, 2019].



[17] AWS, "AWS Free Tier," [Online]. Available:
<https://aws.amazon.com/free/>. [Accessed Aug 8, 2019].

