Batch Analysis of Network Security Monitoring Data

Renu Tiwari, Jiang Xing Kun

January 2021

1 Introduction

This project is implementation for Batch Analysis of Network Security Monitoring Data. The goal is to load the data to a stream, transform to session data, save the session data to storage, do some lightweight analysis and visualize the results of the analysis.

1.1 Data set

The original data set shared was in network data format that is with .pcap extension. Wire-shark is used to load the data-set and then export the data to .csv file. The data set has 150k rows of data and 7 columns:

1. **No** - unique id for each row
2. **Source** - The IP address of the source.
3. **Destination** - The IP address of the destination/target.
4. **Protocol** - The type of protocol used to communicate.
5. **Length** - An integer value representing length of packets transferred.
6. **Info** - Additional messages/information shared/communicated while transferring of the data.

2 System Architecture

2.1 System Architecture Diagram

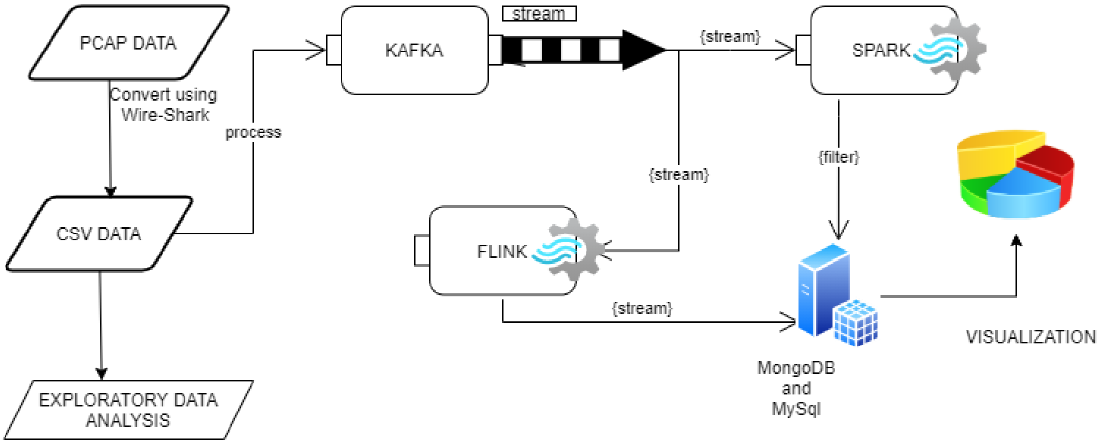


Figure 1: System architecture diagram

2.2 Data Conversion

The initial data file was in .pcap format. Wire-shark is used to load the data and export it as .csv data. Wire-shark was used as to explore the use of software into this project and moreover it was a network monitoring related data. Thus, for initial first step of analysis wire-shark seemed to be a good choice.

2.3 Pre-process Stage - Exploratory Data Analysis

The .csv data then was loaded and processed using NetworkSecurity_EDA.ipynb file. The data is loaded into a pandas data frame and then basic stastics are driven out. Few of the statistics are :

1. Count of different source IP address is 34.
2. Count of different destination IP address is 42.
3. Count of different protocols over which transmission happened is 13. The list of protocols is ['CIP' 'TCP' 'TLSv1' 'CIP CM' 'CIP I/O' 'TLSv1.2' 'SSH' 'SSDP' 'ARP' 'ICMP' 'UDP' 'ENIP' 'GVCP']
4. Top 3 protocols via which data is communicated is -
   (a) Protocol - Count

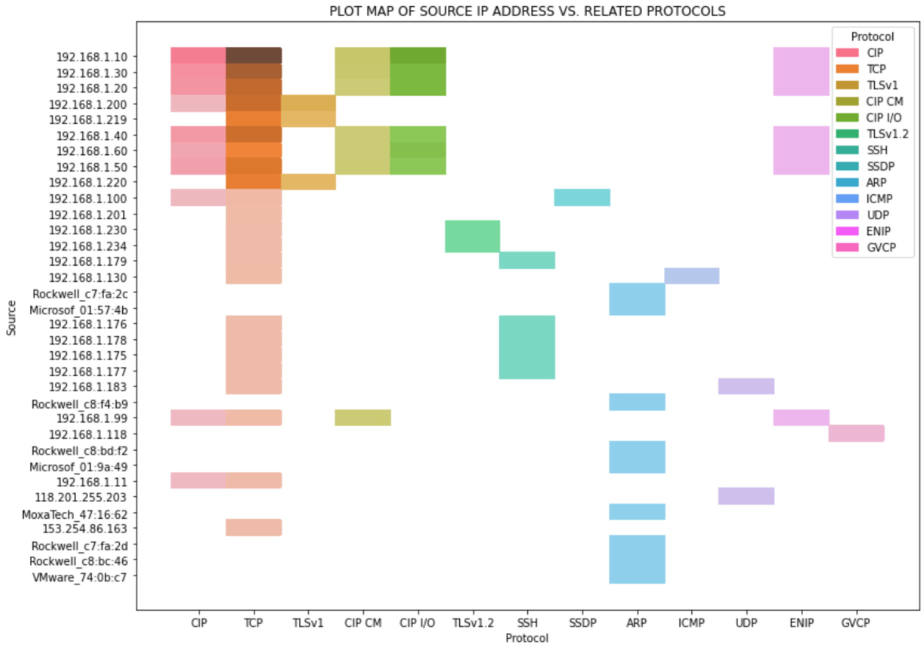


Figure 2: Source IP distribution over different protocols

(b) TCP - 95141

(c) CIP I/O - 25320

(d) CIP - 22188

5. The most number of packets transferred are of length 64 and 60.

   (a) Packet Length - Count

   (b) 64 - 25084

   (c) 60 - 25084

   (d) 86 - 9030

6. The most communicated source and destination IP address is ”192.168.1.10” with a total count of 33219.

7. For experimentation, the Info part was also explored. Apparently, there also seems to be lot of useful info which can be used. Such as, below are some of the top information message communicated. Mostly, which are related to HMI_PLANT, HMI_PLANT_AUTO or HMI_PLANT_RESET.

   (a) Info Detail - Count

   (b) ’HMI_PLANT’ - Service (0x4d) - 1184

(c) Success: 'HMI_PLANT' - Service - 1183

(d) Success: 'HMI_PLANT_AUTO' - Service (0x4d) - 1137

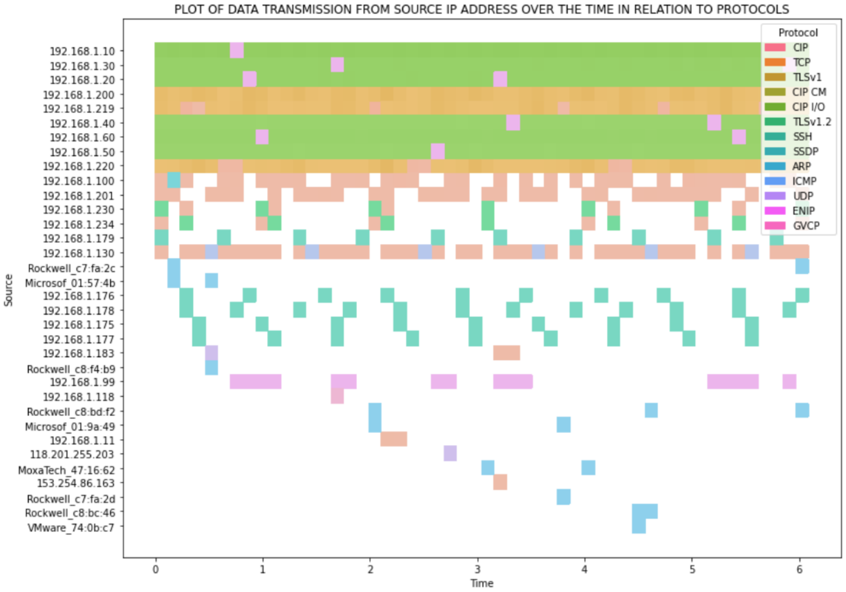


Figure 3: Source IP distribution over time

2.4 Streaming

2.4.1 Kafka Streams

As per Shree et al.(2017), it is "fast, scalable, distributed stream processing platform and fault tolerant messaging system." It is designed for low latency and high throughput.[1]. According to Bejeck (2018), "Kafka Streams is a library that allows you to perform per-event processing of records". Data is processed as it is available. Thus it is thought of a good option to use as a stream server.

The github repo contains folder named as "readData-sendKafka", reads the csv data and stream it using Kafka. The "Listener.java" file contains two topics named as "topic1" and "streaming". "KakfaPro" and "csv2KafkaStream" files uses these two listeners, Java's method BufferedReader to load csv to stream data. Once the kafka server starts, whole the dataset is visible at "localhost:8088/streamData" url.

2.4.2 Java Spark Context Streams

The data is received using Java Spark Context library. Kafka library has createDirectStream() methods which stream data into JavaInputDStream instance.

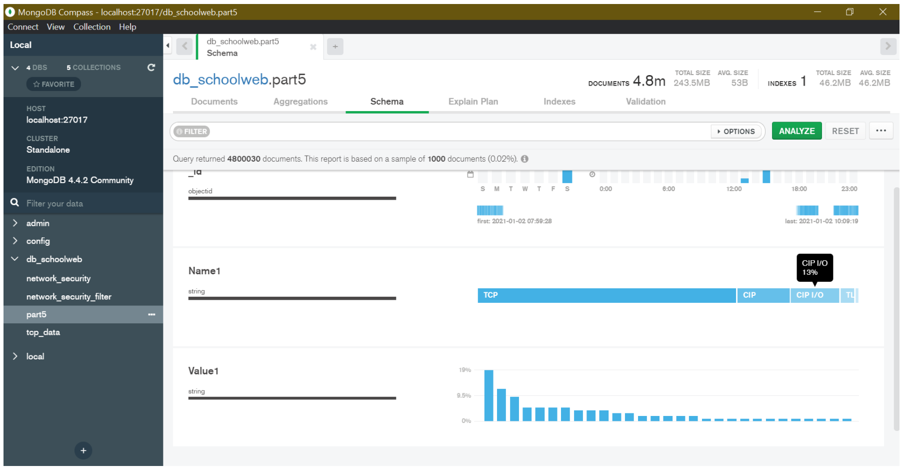
Other methods such as flatmap and reduceByKey is used on the stream data.

2.4.3 Flink

For experimenting, Flink is also used to receive the stream data from Kafka. FlinkKafkaConsumer09 class loads the topic "topic1" and "streaming" into system. The major filtration is done on Spark.

2.4.4 Filtering Data

The filtrating of streamed data is done on two times and data is stored into respective tables. The RDD is read from JavaInputStream and for each RDD element, a check is used to see if the stream is using "TCP" protocol or not. If there's "TCP" protocol used, then data is processed further and stored in "tcp_data" table. The second type of filtration is done based on the length of streamed data. If the length of row is greater than 10 and it's communicated using protocol, then it's stored into "part5" table.



MongoDB Compass - localhost:27017/db_schoolweb.part5

HOST localhost:27017
CLUSTER Standalone
EDITION MongoDB 4.4.2 Community

db_schoolweb.part5

DOCUMENTS 4.8m TOTAL SIZE 243.5MB AVG. SIZE 53B
INDEXES 1 TOTAL SIZE 46.2MB AVG. SIZE 46.2MB

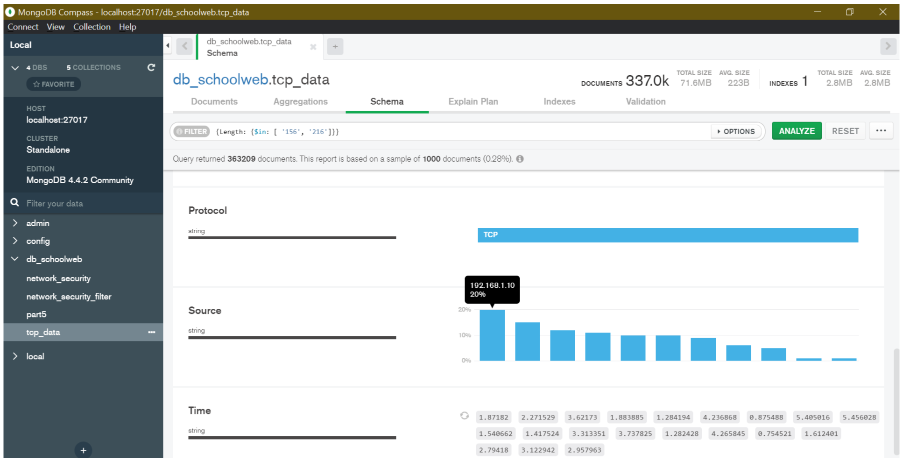
Query returned 4800030 documents. This report is based on a sample of 1000 documents (0.02%).

first: 2021-01-02 07:59:28
last: 2021-01-02 10:09:19

Figure 4: Analysis of data stored on basis of length

2.5 MongoDB and MySQL

For storage of data primarily, MongoDb is used. It is easily configurable and fast also. The spark module is written to stream the filtered data from above step to database. Initially, the data is stored into mysql database. Later and for final use, MongoDB is used. MongoDb Compass is used for hosting database. MongoClient is used with Java to load the collections and store the data into "tcp_data". MySQL is used in storage of "part5" table.



HOST localhost:27017

CLUSTER Standalone

EDITION MongoDB 4.4.2 Community

DOCUMENTS 337.0k TOTAL SIZE 71.6MB AVG. SIZE 223B INDEXES 1 TOTAL SIZE 2.8MB AVG. SIZE 2.8MB

FILTER `{Length: {$in: [ '156', '216']}}`

Query returned **363209** documents. This report is based on a sample of **1000** documents (0.28%).

```
1.87182 2.271529 3.62173 1.883885 1.284194 4.236868 0.875488 5.405016 5.456028
1.540662 1.417524 3.313351 3.737825 1.282428 4.265845 0.754521 1.612401
2.79418 3.122942 2.957963
```

Figure 5: Analysis of TCP data on MongoDB

Kibana console was used to create, monitor and delete ElasticSearch indices, explore data loaded to ElasticSearch, and create dashboard to visualize the anomaly detection results.

3 Execution Steps

The overall pipeline of the project can be summarized in following steps:

- Start the Zookeeper server (usually in port 2181)
- Start the Kafka server (usually in port 9092)
- Created two kafka topics named as ”topic1” and ”streaming” to load the csv data.
- Use csv2KafkaStream to load csv file to the kafka topic created earlier and see it on localhost:8088/streamData.
- Execute Spark-Kafka consumer connection application.
- Start MonDb server (usually on localhost: 27017)
- Create index using Kibana console to load the data to be visualized
- Open Dashboard.html to see the visualization.

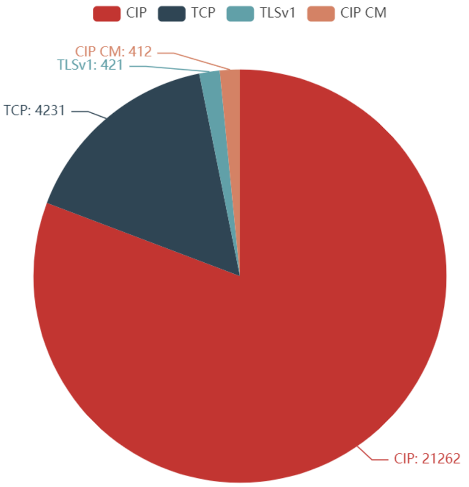


Figure 6: Pie chart visualization on basis of protocols

4 Experiments

4.1 Converting JavaInputDStream to JavaRDD

It was tried to use Spark inbuilt Filter methods on the streamed Kafka Data. As filter methods uses RDD to work on, it was tried to convert the stream to RDD. However, during the implementation of this idea, several problems were faced in typecasting and conversion of JavaInputDStream to JavaRDD as both takes different input. Also, Kafka streams data to Spark using directStream method. For using RDD, it would be beneficial if used Spark Streaming on both sides. However, RDDs are still executed from the stream data of input stream. It has method foreach(RDD), which was used to do user defined logical filtration.

4.2 Analysis on Info Data

The last part of the data contains the Info coloumn which contains various status codes and messages. As seen in EDA part, that there's high number of Info for HMI related codes.

4.3 Considering Kafka as source and Spark as consumer

A client-server architecture is tried to followed here. The Kafka side acts as server side to load the data into streams and then spark act as client to receive that stream and save it to database further. Also, this was done to experiment with more technologies and how their communication works.

4.4 Flink

As a standalone task, a small experiment on using Flink was also done. The server side Kafka remains the same. But on consumer side, Flink is used to read the stream. Further, there's filteration and storage to Database is still in progress.

4.5 Migrating from MySQL to MongoDB

In previous section, it was shared about how collections and databases are used. Initially, MySQL was used to store the filer data. But then later, it was migrated to MongoDB as using MySQL was not in the scope of the project guidelines. Moreover installation, configuration and creating connection is relatively fast in MongoDB. Both these servers were hosted locally on personal computers.

References

[1] Shree, R., Choudhury, T., Gupta, S. and Kumar, P., (2017). KAFKA: The modern platform for data management and analysis in big data domain. 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), 1. https://ieeexplore.ieee.org/abstract/document/8343593/

[2] Bejeck, B. (2018). Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API (1st ed.). Manning Publications.

[3] Rahasak, . (2020). Kafka and Zookeeper with Docker. https://medium.com/rahasak/kafka-and-zookeeper-with-docker-65cff2c2c34f