Detection of malicious Encrypted Web Traffic Using Machine Learning

by

Jay Shah

B.Eng. Gujarat Technological University, 2014

A Project Report Submitted in Partial Fulfillment of the Requirements for

the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering at University of Victoria, Victoria, British Columbia, Canada

Supervisory Committee

Detection of malicious Encrypted Web Traffic Using Machine Learning

by

Jay Shah

B.Eng., Gujarat Technological University, 2014

Supervisory Committee

Dr. Amirali Baniasadi, (Department of Electrical and Computer Engineering)

Supervisor

Dr. Issa Traore, (Department of Electrical and Computer Engineering)

Co-Supervisor

Table of Contents

List of Figures

List of Tables

Abstract

An increasing amount of web traffic is currently encrypted using HTTPS. While most of the HTTPS traffic is legitimate, a growing slice is generated by malware. The use of the HTTPS protocol by malware makes its detection more challenging. The current approach is to detect HTTPS malware traffic by using HTTPS interceptor proxies. This method requires decrypting the traffic on the fly, which poses some threat to the data and communication security and privacy. The goal of this project is to detect HTTPS malicious traffic without decryption. We propose a new detection model that leverages the underlying HTTPS certificate characteristics and connection data that are fed to a machine learning classifier. Our model consists of a set of features extracted from log files generated from the Bro Intrusion Detection System (IDS), which are classified using the XGBoost algorithm. Experimental evaluation is conducted using a public dataset, yielding encouraging results.

Chapter 1 Introduction

1.1 Context

Basic web communication through the Hypertext Transfer Protocol (HTTP) can be read by anyone who manages to see the packets between clients and servers. Encryption is necessary to protect the privacy of end users, and the confidentiality and integrity of communications. HTTPS is standard encryption of HTTP with either Transport Layer Security (TLS) or Secure Socket Layer (SSL).

According to the Google report from 2017, the use of HTTPS has been growing [1]. The report shows that the desktop users load more than half of the web pages over the HTTPS. Based on the research, Windows users load 60% of visited websites over the HTTPS by using the Chrome browser, and Mac users' number is 72% for the same purpose. All over the Internet, the use of encrypted traffic has been growing fast. Hence, malware has also started using HTTPS to secure their own communications and evade detection. As per the Cisco report from 2016 [2], although the majority of the malware traffic is still using unencrypted HTTP traffic, there was a steady 10-12% of malicious communications using HTTPS.

TLS is a cryptographic protocol, which provides privacy for applications. TLS provides a complex set of recognizable parameters which allow many inferences to be made about both the client and server involved in communications. It is generally implemented on top of common application protocols, such as HTTP for web or SMTP for email. There are two phases in TLS: Handshake and Data transfer. In the handshake phase, different algorithms and parameters are required for secure data transfer. Symmetric keys are agreed upon once the handshake phase is completed. After the handshake phase, the application data is transferred between client and server in the form of encrypted records.

Some studies have shown that 60% of network traffic use TLS [3]. TLS uses X.509 certificates for server authentication. X.509 is a complex document: it may generate different kinds of errors while using it. Some of the available certificates belong to malicious websites and should not be trusted or used by clients. These malicious web servers should not be visited.

The detection of HTTPS malware traffic is difficult and more complicated because of the interference of encryption with the ability of traditional detection techniques. The most common solution to dealing with HTTPS traffic in organizations is to install HTTPS interceptor proxies.

This involves installing special certificates in the organization computers to open and inspect HTTPS traffic of the employees. The HTTPS interceptor is placed between the client and the server. The mechanism of the HTTPS interceptor is to decrypt the incoming traffic, scan for malicious software, encrypt the traffic again, and then send it to the destination if deemed trustable. The disadvantage of using HTTPS interceptor is that it is expensive and computationally demanding. Because it requires decrypting the traffic, the HTTPS interceptor does not provide secure and private communication, which should be the main function of the HTTPS.

1.2 Project Objectives and Approach

In theory, the detection of malware HTTPS traffic is possible with good accuracy without decrypting the traffic [4]. This kind of solution is very appealing, because it allows avoiding using HTTPS interceptors. The privacy and security of communication would then be maintained. Such procedure is faster for malware detection.

The objective of this MENG project is to explore the feasibility of an approach to detecting malware without decrypting HTTPS traffic using machine learning. The proposed approach consists by analyzing HTTPS traffic, of identifying and extracting some characteristics or features that could be used to detect malware without decrypting HTTPS traffic.

Specifically, we extract a set of features from the data logs generated by the Bro Intrusion Detection System (IDS) [6]. Bro provides a passive, open source network traffic analyzer. It provides a security monitor that inspects all traffic on a link in depth for signs of suspicious activity. Bro IDS is able to process network traffic captured in pcap file format and provides all the information about connections, SSL handshakes, and X.509 certificates. It can generate log files by processing pcap files. Bro logs provide detailed information about HTTPS traffic.

Our approach to detect encrypted malware consists of using a subset of Bro log files to extract a set of features that are classified using the XGBoost machine learning algorithm. Experimental evaluation of the approach is conducted using a public dataset of HTTPS network traffic capture.

1.3 Related Work

The detection of HTTPS malware traffic is difficult without decryption. There are some research papers which use decryption to detect HTTPS malware traffic. Their features are mostly based on unencrypted TLS/SSL handshake messages and the certificates. Deep Packet Inspection (DPI) and digital signatures are traditional approaches to detect malware, but they are not applicable on encrypted traffic. The decrypted network traffic does not provide privacy to the users.

The research paper "k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach" to detect malware in HTTPS traffic using k-NN classification. The information of HTTPS connections is very limited and depends on the number of uploaded bytes and downloaded bytes, and also the duration of connections. The main aim of this research work is to detect the secure HTTPS connection related to malware families. The testing is done using ECM linear classifier. The metric indexing k-NN classification approach provides efficiency to detect malware in HTTPS traffic over dataset of few high-dimensional network traffic descriptors, which reduced false positive rate [8].

The research paper "Deciphering Malware's use of TLS (without Decryption)" to detect malware HTTPS traffic without decryption. The main aim of this work is to study 18 different malware families, which contain thousands of unique malware samples and ten-of-thousands of malicious TLS flows. Malware is more difficult to classify because their use of TLS. This study provides more high-level information about malware's use of TLS. According to the study, malware targets to the weak cipher suites. The data features are collected using unencrypted TLS handshake messages. These features are used to classify malware and perform family attribution by using rules or classifier [4].

Our approach to detect malicious HTTPS traffic without decryption is unique from other work. To detect malicious HTTPS traffic, we analyzed the different log files generated by Bro IDS. Bro logs provide comprehensive information about HTTPS traffic. We used XGBoost machine learning algorithm to classify the traffic. Our features are based on three different log files: - conn.log, ssl.log, and x509.log.

1.4 Objectives

The structure of the remaining chapters of the report is as follows:

- Chapter 2 provides information about Bro log files and the interconnection between log files as well. Also, it describes the SSL connector and how the connection-4 tuples are created.
- Chapter 3 presents the proposed feature model. It describes the different features based on connection, ssl and certificates.
- Chapter 4 summarizes the system architecture and system implementation. It also describes the evaluation process and presents the detection performance results.
- Chapter 5 concludes the report and summarizes possible future work.

Chapter 2 Data Source

2.1 Bro Logs

We define our feature model by analyzing the logs generated by Bro; these logs provide detailed information about HTTPS traffic. Bro generates different kinds of log files as follows:

- conn.log: TCP/UDP/ICMP connections
- ssl.log: A record of SSL sessions, including certificates being used.
- x509.log: X.509 certificate information.
- http.log: All HTTP requests with their replies.
- Smtp.log: a description of SMTP activity.
- ftp.log: A log of FTP session activity.
- dns.log: DNS queries with their responses.
- dpd.log: A summary of protocols used on non-standard ports.
- files.log: Description of files transferred over the network and all the information collected from different protocols.

The files in Bro logs are interconnected to each other through unique keys. In each log file, every line includes a unique key that links it to lines in other log files.

conn.log

```
#fields ts  uid id.orig_h   id.orig_p   id.resp_h   id.resp_p   proto   service duration
#types  time    string  addr    port    addr    port    enum    string  interval    coun
1313419436.849921   CwanTMNRUVvOo0qoc   24.224.184.216  5284    147.32.84.229   13363
1313419491.716585   CXnT6TlMWQ3aZblpVi  147.32.84.59    39050   74.125.232.214  443 tcp
1313419491.933269   C7YJ15lCItEN93UQf   147.32.84.46    7242    203.106.50.35   80  tcp
1313419489.666680   CL9AFkluvxnjK12R4j  66.249.66.119   44738   147.32.84.188   80  tcp
```

ssl.log

orig_h id.orig_p	id.resp_h id.resp_p	version	cipher	curve	server_name	resumed	last_alert	next_protocol	established	cert_chain_fui
ing addr port	addr port	string	string	string	bool string	string	bool	vector[string]	vector[string]	string st
CSeVKJD1sZK5LFv14	147.32.84.59 38661	217.77.163.138	443	SSLv3	TLS_RSA_WITH_RC4_128_SHA	-	-	T	-	- T	- - - - -
CXnT6T1MWQ3aZb1pVi	147.32.84.59 39050	74.125.232.214	443	TLSv10	TLS_RSA_WITH_RC4_128_SHA	-	-	F	-	- T	F3LzkQ3xQH0N6Ks64k
CaHmID3kkWGfjTuY34	147.32.84.59 39051	74.125.232.214	443	TLSv10	TLS_RSA_WITH_RC4_128_SHA	-	-	F	-	- T	FzAtC63h6JccbC0o11
CGnw7e39Qzggf2f7Qb	147.32.86.126 60835	89.185.253.133	443	-	- - - F	-	- F	-	- - - - -	- - - - -

x509.log

1313419485.169830	FYoeA52dl71e7BnED9	3	4AF56A7BE6A0C025AB8C60B5AB40D73C	CN=*.dropbox.com,OU=IT,O=Dropbox\\, Inc.,L=San
1313419491.756879	F3LzkQ3xQH0N6Ks64k	3	1F19F6DE35DD63A142918AD52CC0AB12	CN=mail.google.com,O=Google Inc,L=Mountain View,
1313419491.881272	FzAtC63h6JccbC0o11	3	1F19F6DE35DD63A142918AD52CC0AB12	CN=mail.google.com,O=Google Inc,L=Mountain View,
1313419501.886237	FoUnnI2nNksJXq9KKg	3	4AF56A7BE6A0C025AB8C60B5AB40D73C	CN=*.dropbox.com,OU=IT,O=Dropbox\\, Inc.,L=San

Figure 1: Interconnection of log files

Figure 1 shows an example of how log files are connected to each other by sharing the same unique key. For instance, the second connection listed in conn.log (CXnT6T1MWQ3aZb1pVi) is linked to a ssl record in ssl.log by using the same unique key, which in its turn has a certificate key in x509.log, wherein the certificate is described. As shown in the figure, the first and last records of ssl.log file do not have certificate path (see last column), which indicates that there is no certificate for them.

Although multiple log files are generated using Bro IDS, our proposed features are extracted using only three log files as follows:

1. Connection record: The conn.log file contains information about IP addresses, ports, protocols, states of connection, number of packets etc. Each line corresponds to a group of packets and describes the connection between two endpoints.
2. SSL record: The ssl.log file provides information about SSL/TLS handshakes and encryption establishment process. It contains versions of SSL/TLS, server names, ciphers used, certificate path, subjects and issuer.
3. Certificate record: The x509.log file contains the certificate record describing the information about certificate, such as certificate serial number, common names, time validities, subjects, certificate key length, etc.

The certificate record from x509.log file represents a certificate in Bro IDS. The main aim of the certificate in HTTPS is to check the credibility of the web server with certificate authorities (CA). There are two types of CA authorities: a root CA and an intermediate CA. A certificate can be trusted only if it is issued by a CA, which is included in the device as trusted. If the certificate is not issued by a trusted CA, the web browser checks the certificate issuer until a trusted CA is found. All certificates from the root certificate to end-user certificate are called Certificate Path. The Certificate Path is stored in SSL record in ssl.log. SSL records contain a list of unique keys aiming towards a x509.log file, where all the certificates are described.

Figure 2: Certificate and Certificate Path

Figure 2 illustrates the log entries related to certificate and certificate path. In the figure, the highlighted segments correspond to certificate records. Examples of certificate path are Google Internet Authority, *.google.com, geo trust.com as shown in the figure.

2.2 SSL connector and Connection 4-tuples

A SSL connector is a data structure consisting of a set of connection records that are extracted from a combination of three records: a connection record, an ssl record, and a certificate record. The connection record and the ssl record have the same unique key, and this unique key has a certificate path in the ssl record. By using the certificate path, we can locate corresponding certificate in the x509 log file. As mentioned above, some ssl records have no certificate path; so

no certificate is available for them. An SSL record having a certificate path depends on many factors, such as whether the ssl handshake was successful or not, and so on.

A connection 4-tuple is the combination of ssl connectors that share the same 4-tuple of source IP, destination IP, Port, and protocol.

Connection 4-tuples can be used to describe malicious traffic, such as, the behaviour of malware connecting to a command and control (C&C) server. They can also be used to capture and express benign traffic patterns, such as how authorized users are connecting to legitimate websites.

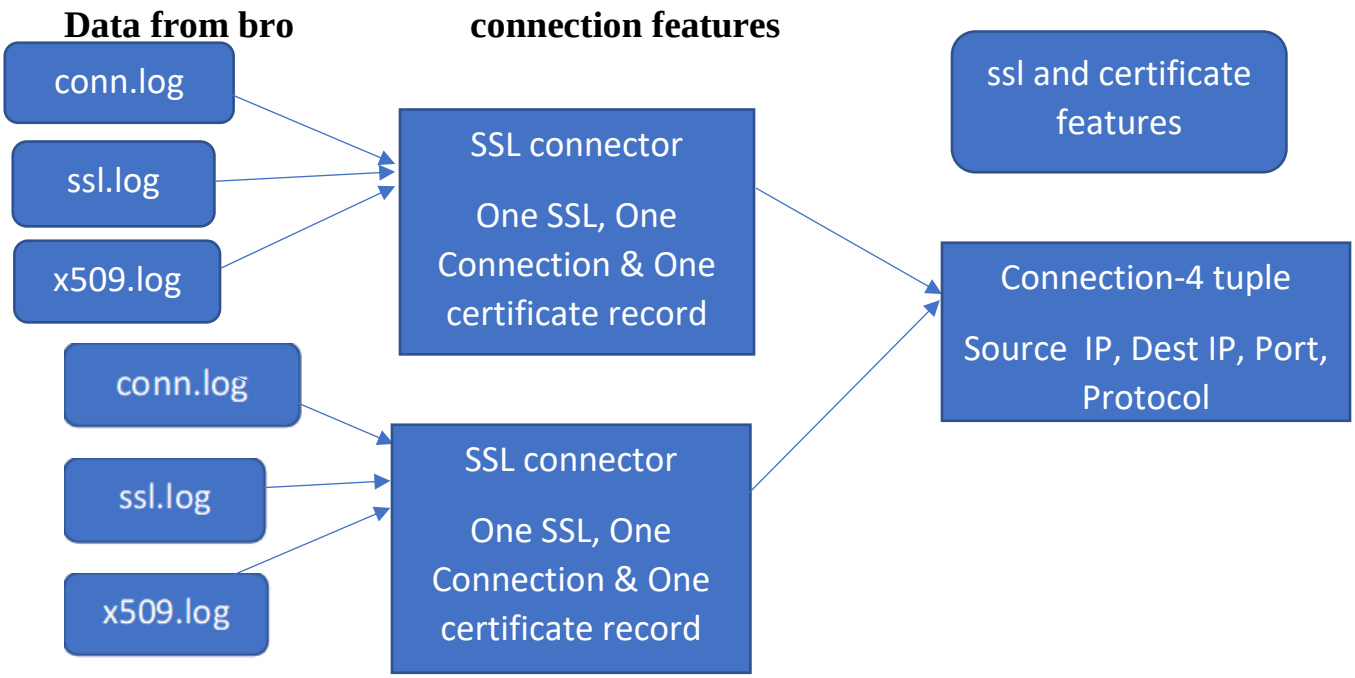


Figure 3: Generation of ssl connectors and connection 4-tuples

As depicted by Figure 3, the connection 4-tuples are derived from three log files – conn.log, ssl.log, and x509.log files. The algorithm for generating connection 4-tuple goes through each record of the three mentioned log files. The following algorithm is used to process each record of the log files and extract connection 4-tuples:

- For each SSL record in ssl.log file:

- Using the unique key from the SSL record (ssl.log), find the connection records from conn.log with the same key.
- Check for the certificate path in the SSL record. If it is not empty, then use that path as the unique key to search for the certificate record in the x509.log.
- By combining the above records, SSL connectors are obtained for each unique key. Each SSL connector also has a 4-tuple (SrcIP, DstIP, Port, and Protocol) associated with it. Each 4-tuple derived from the SSL connector is compared against the set of connection 4-tuples. If a match is found, then the SSL connector data is added to that. In case a match is not found, a new connection 4-tuple is created and then the SSL connector data is added alongside.

Chapter 3 Features Model

We derive our features from the connection 4-tuple. A separate feature vector is generated for each connection 4-tuple.

Our proposed feature model consists of three categories of features: connection features, SSL features, and certificate features. The connection features capture traffic behavior characteristics outside (i.e. unrelated to) encryption. These include information about IP addresses, ports, protocols, state of connection, number of packets, and so on. The SSL features capture the information about the SSL/TLS handshake and establishment phases in the encryption process. These include the SSL/TLS version, cipher used, server name, certificate path, subjects, and so on. The certificate features capture the information about certificate serial numbers, common names, validation periods of certificates, digital signature algorithms, etc.

3.1 Connection features

These features are derived from connection records from conn.log, and consist of the following:

1. *Number of SSL connectors and connection records*: SSL connector contains the information of three log files: ssl.log, conn.log, and x509.log. This feature is applied to each SSL connector and connection record, and defined as follow:

   $$R_1 = S_l + C_r$$

   Where S_l is the number of SSL connectors and C_r is the number Connection records.

2. *Payload bytes from originator and responder*: The number of payload bytes the originator and responder sent for all connection records from the conn.log. This feature is defined as follows:

   $$R_2 = \frac{r}{r + o}$$

   Where *r* is the number of bytes from responder and *o* is the number of bytes from originator.

3. *Ratio of established states of connection*: There is a state of connection for each connection record. There are 13 different types of states, divided into two groups: established connections and non-established connection states. The states, which have successful TCP handshake or just attempt to handshake are in established states (SF, S1, S2, S3, RSTO,

RSTR). The states, which have unsuccessful handshake, are in non-established states (OTH, SO, REJ, SH, SHR, RSTOS0, RSTRH). The ratio is defined as:

$$R_3 = \frac{e}{e+n}$$

Where *e* is the number of established states and *n* is the number of non-established states.

3.2 SSL features

These features are derived from SSL records from ssl.log, and described as follows:

1. *SNI (Server Name Indication) as IP*: there are some SSL records, which have SNI as IP address. So, they have SNI IP as destination IP address. The feature (R_4)value is 0 if any SSL record has SNI as IP and the feature value is 1 if there is no SNI as IP in SSL record.
2. *Ratio of TLS records*: All the records have the version of TLS or SSL protocols, which are used for encryption. These include TLS1.0, TLS 1.1, TLS 1.2, TLS 1.3, SSL 1.0, SSL 2.0 and SSL 3.0. SSL is older than TLS, and as such it is rarely used; mostly all the normal traffic use TLS. The following ratio is defined for this feature:

$$R_5 = \frac{TLS}{TLS + SSL},$$

Where TLS is the number of SSL records that have TLS protocol, and SSL is the number of SSL records that have SSL protocol.

3. *Status of the certificate paths*: Some of the SSL records have certificate path, while others have not. There are two types of certificate authorities (CA): a root CA and an intermediate CA. If the certificate was not issued by a trusted CA, then the web browser would check recursively whether the certificate of the issuing CA was issued by a trusted CA until the trusted CA is found. The certificate path is from the root certificate to the end-user certificate. The certificate path is stored in SSL record in Bro in ssl.log and used to define feature R_6. The datatype for this feature is binary. If the certificate path is available in SSL record, then the feature value is 0 and if the certificate path is not available then the feature value is 1.

4. *Ratio of self-signed certificate*: By using Bro, we can check whether the end-user certificate is self-signed or not. This information is in SSL records. The feature is defined as follows:

$$R_7 = \frac{s}{c}$$

Where *s* is the number of self-signed certificates and *c* is the total number of all certificates.

3.3 Certificate features

These features are based on certificate records in x509.log and some of them are connected to ssl.log as well. The following features are defined under this category:

1. *Validation period of the certificates*: The validity of a certificate can be checked by using the capture time and the validity period of the certificate. If the capture time is within the certificate validity period, then it is valid. The datatype of this feature (R_8) is Boolean. If the certificate is valid, then the feature value is true, and the feature value is false if the certificate is not valid.
2. *Number of domains in certificate SAN DNS*: The SAN (Subject Alternative Names) describes which domains belong to this certificate. For example, a Google certificate SAN DNS is {*.google.com, *. google.ca, *. google.co.in, *. google.cl, *. google.co.uk, *. google.de}.

   For each new incoming certificate, the number of DNS in SAN is stored in a list. The average is calculated from the list.

   Assuming that a connection tuple includes n SSL connectors, this feature value is defined as:

$$R_9 = \frac{\sum_{i=1}^{n} n_i}{n}$$

   Where n_i is number of domains in the i^{th} SSL connector.
3. *Ratio of SSL records with certificate path*: Check how many SSL records have the certificate path. The feature is defined as follows:

$$R_{10} = \frac{C_r}{S_r}$$

   Where C_r is the number of certificate records and S_r is number of SSL records for one connection 4- tuple.

4. *SNI in SAN DNS*: The SNI is Server Name Indication that is included in the SSL record. Generally, SNI is part of SAN DNS. SAN DNS are domains in certificate record that belong to the certificate. The data type of this feature (R_{11}) is binary. If the none of the certificate records contain SNI then the feature value is 0. Else if any of the certificate records contains SNI in SAN DNS, then the value of the feature is 1.
5. *CN in SAN DNS*: The CN is Common Name which is part of the certificate record. It should be part of SAN DNS as well. The data type of this feature is binary. The value of the feature (R_{12}) is 1 if none of the certificates contain the CN in SAN DNS. Otherwise, the value of the feature is 0 if at least one of the certificates contains the CN in SAN DNS.

Chapter 4 Model Implementation and Evaluation

4.1 Dataset

An important part of the project was to choose adequate dataset for the experimental evaluation of the proposed approach. We used an existing public dataset that is available online and was collected in real network environment [5]. The dataset contains malware and normal traffic. The malware was collected from infected hosts, and the normal traffic was collected from trusted hosts. Table 3.1 shows the dataset breakdown. Our entire dataset contains 30 pcap files- 24 files containing normal samples and 6 files containing malware samples. Table 1 shows a breakdown of the data in different categories.

Type of data	**Number of samples**
Normal connection records	271,906
Malware connection records	21,521,315
Normal SSL records	31,739
Malware SSL records	116,814
Normal certificate records	16,415
Malware certificate records	59,601
Normal certificates	1145
Malware certificates	583

Table 1: Dataset Breakdown

4.2 Log Data

The data consist of network traffic packets stored in pcap files. The log files are generated from the pcap files by using Bro IDS as depicted in Figure 4.

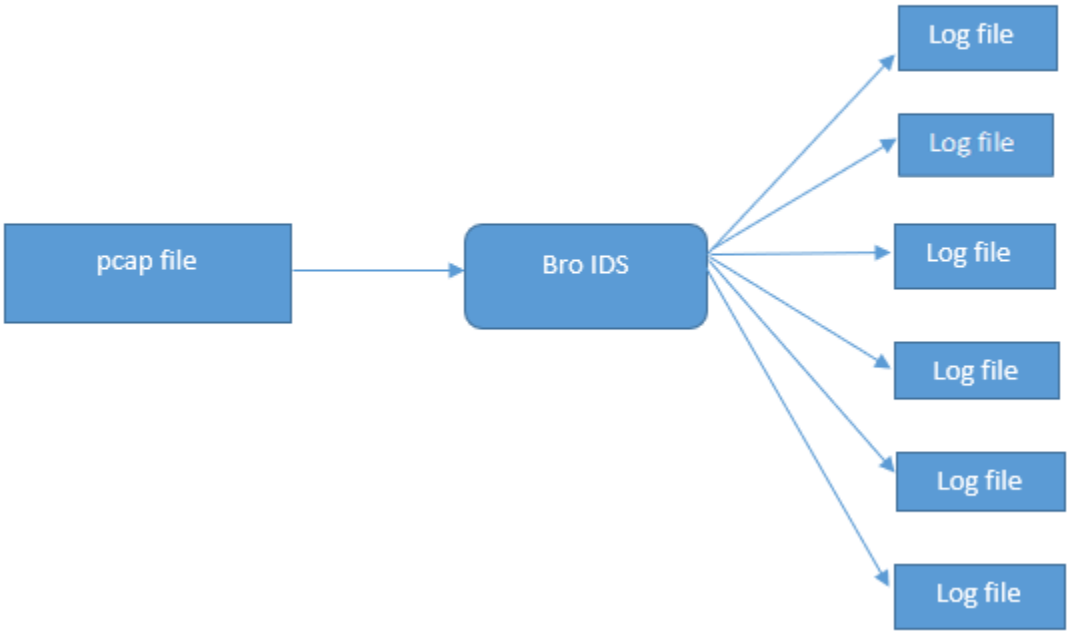


Figure 4: generating log files from pcap files

Bro works best on Unix-based system and does not require custom hardware. BroControl is an interactive shell for easily operating Bro installation on a single system or multiple systems in a traffic controlling cluster.

The command for generating log files from pcap file is shown in figure 5, and consists of the following:

Command: ***bro -r (pcap file name)***

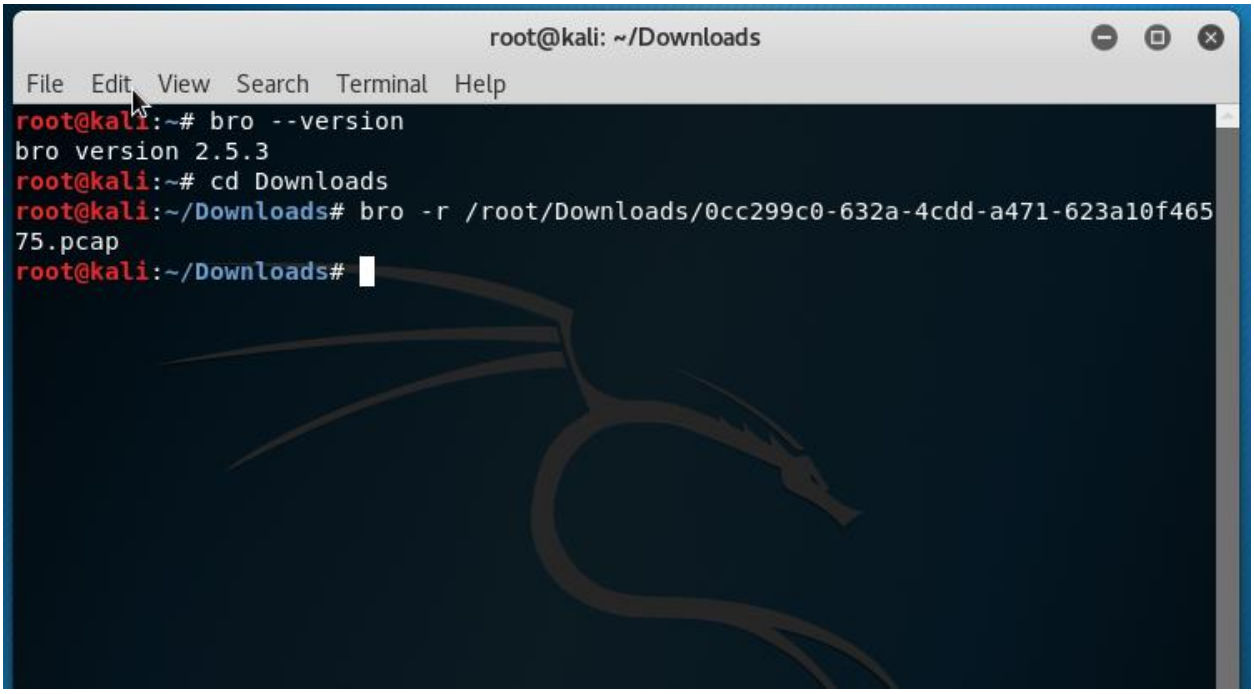


Figure 5: Bro command for creating log files

All log files are written out in a human readable format (ASCII) and organize the data into columns. Log entries are generated from the analysis of the network protocols. Figure 6 depicts the structure of a typical log entry, which consists of the following values: a timestamp, a unique connection identifier (UID), originator host/port, responder host/port (connection 4-tuple).

The UID can be used to identify all logged activities related to a given connection 4-tuple over its lifetime. It can be also used possibly across multiple log files as well. The remaining columns are protocol-specific and provide the information based on protocol-dependent activity [6]. For example, the http.log provides analysis results about Bro HTTP protocol analysis.

```
#fields ts  uid id.orig_h   id.orig_p   id.resp_h   id.resp_p
#types  time    string  addr    port    addr    port    count
1312962413.592298   CUnJJE4jOAtkXb0nzf  66.249.66.241   44895
```

Figure 6: first few fields of http.log

4.3 System architecture

The focus of the project is to check the HTTPS traffic by extracting the different features and determining whether it is malicious or not. The features are extracted from connection records, ssl records, and certificate records. We use an API that takes the log files as input, and then creates connection 4-tuples based on source IP, destination IP, destination ports and protocol. The output is classified as suspicious or normal.

The main steps involved in the detection system are outlined in figure 7 and described as follows:

- The first step is to provide the path for log files as input. Log files are generated from pcap files using Bro
- The Second step is to create connection 4-tuples (each consisting of Source IP, Destination IP, Port, Protocol) using generated log files. From generated log files we only use conn.log, ssl.log and x509.log to create connection 4-tuples.
- The Third step is to extract all the features using log files and connection 4-tuples. The features are divided into three different groups: connection-based, ssl-based and certificate-based features. A total of 12 different features are used to detect whether the traffic is malicious or not.
- The final step is to classify the network data using the XGBoost algorithm as normal or suspicious. The XGBoost algorithm is implemented using scikit learn library in Python.

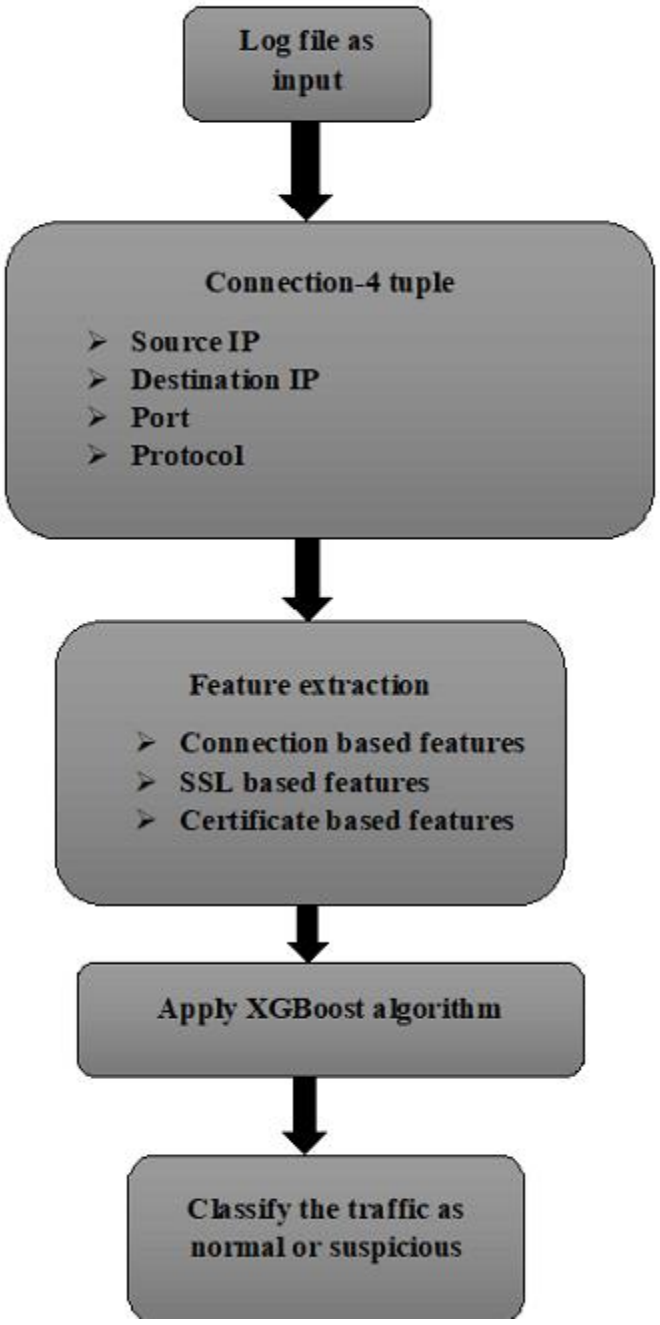


Figure 7: Detection system architecture

4.4 Machine Learning

What is Machine Learning?

Machine learning is a branch of Artificial Intelligence (AI). The aim of machine learning is to give machines access to data and let them learn for themselves. Machine learning provides the information about how computers can learn. It gives the ability to computer systems to learn from data by using statistical techniques. For this, it does not need any hardcore programming. Machine learning is used in various fields like data security, financial trading, healthcare, fraud detection, smart cars, etc [9]. Machine learning is a fast-growing development for health care industry with wearable devices and sensors which can use the data to assess a patient's health in real time.

Machine learning is defined as follow:

"Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions."

According to Standford University "Machine Learning is science of getting computers to act without being explicitly programmed."

With the use of machine learning, computer system is automatic improvise with the experience and from examples. It also provides information to computer system with different tasks, data, observations, and interaction with real world. The main aim of machine learning is to enable computers to learn their own. Machine learning algorithms provide teaching set of data, to match the patterns in observed data, build the model, and then make the prediction without explicitly pre-programming and models. Machine learning comes into the use because it can produce models rapidly that can analyze bigger and more complex data. It also provides the fast and more accurate results even on very large scale.

There are many kinds of machine learning algorithms, but all the algorithms contain following function:

- Representation: It provides classifiers that computer systems can understand easily. For example, decision trees, logistic regression, neural networks, deep learning.
- Evaluation: It evaluates the system with error rate, accuracy, score, etc.

- Optimization: Different kind of optimization can be used. For example, linear programming, quadratic programming, greedy search [10].

As shown in Figure 8, machine learning tasks are divided into supervised learning, unsupervised learning and reinforcement learning. For the good machine learning system, it requires data gathering capabilities, basic and advanced algorithms, automation process and scalability [9].

- Supervised learning algorithms: It is trained using labeled examples, such as an input, where the desired output is known.
- Unsupervised learning algorithms: The labels are not provided to learning algorithm. The algorithm explores the data and find structure within data.
- Reinforcement learning algorithms: It is used for robotics and gaming. It works on trial and error approach to determine which action provides the best result [9].

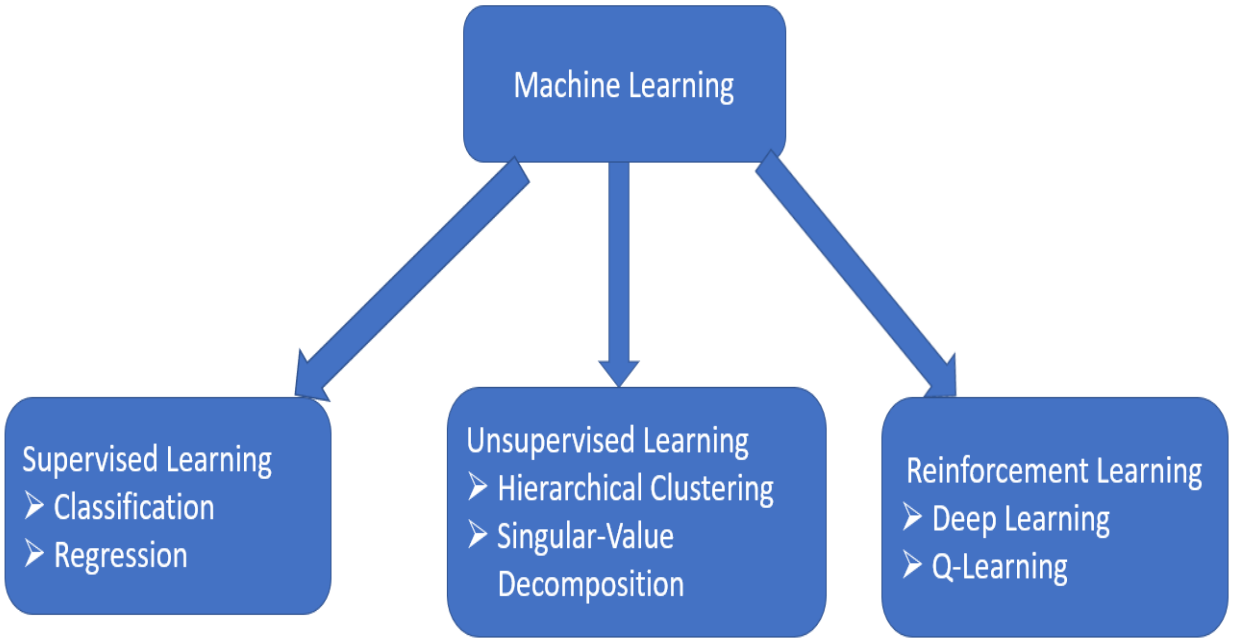


Figure 8: Machine learning tasks

4.5 System implementation

The software implementation was based on Python 3. PyCharm IDE is used to setup the Python project. We added the libraries and then created a new project using PyCharm. The steps involved in setting up the implementation for the Python project are illustrated in Figures 8 to 11 and outlined as follows:

Step 1: Create a Python project

The structure of the generated project is shown in figure 8:

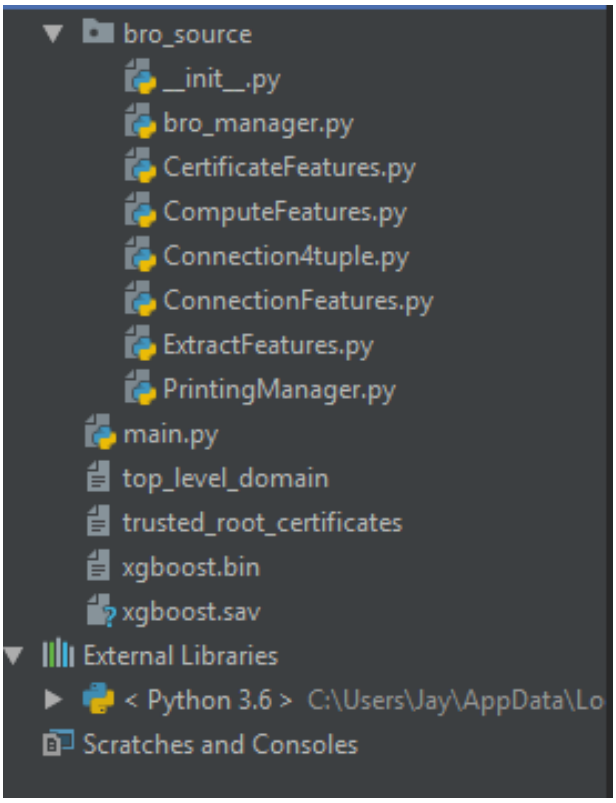


Figure 9: Python Project Structure

Step 2: Add libraries

After creating the project, add libraries as shown in figure 9:

Package	Version
beautifulsoup4	4.6.0
bs4	0.0.1
certifi	2018.4.16
chardet	3.0.4
idna	2.7
lxml	4.2.1
numpy	1.13.3
pip	18.0
pytz	2018.4
requests	2.19.1
scikit-learn	0.19.1
scipy	1.0.1
setuptools	39.0.1
termcolor	1.1.0
urllib3	1.23
xgboost	0.72

Figure 10: installing the libraries

Step 3: Implement REST resources

```
conn_split = conn_log.split('    ')
# 2-srcIpAddress, 4-dstIpAddress, 5-dstPort, 6-Protocol
connection_index = conn_split[2], conn_split[4], conn_split[5], conn_split[6]

label = "NONE"

try:
    self.tuple_connection[connection_index].add_ssl_flow(conn_log, label)
except:
    self.tuple_connection[connection_index] = ConnectionFeatures(connection_index)
    self.tuple_connection[connection_index].add_ssl_flow(conn_log, label)

self.lines_for_ssl += 1
# x509 and ssl
valid_x509_list = self.split_ssl(ssl_line, connection_index, label)

self.tuple_connection[connection_index].add_ssl_log(ssl_line, valid_x509_list,
                                                    os.path.basename(path_to_ssl_log))
```

Figure 11: Feature extraction using conn.log, ssl.log, x509.log

XGBoost Algorithm

XGBoost stands for eXtreme Gradient Boosting algorithm. It is an open source software library that provides a gradient boosting frame work for Java, C++, Python and different programming languages. It is an efficient and scalable implementation gradient boosting framework. It works on Windows, MacOS, and Linux. XGBoost supports various objective functions with regression, classification and ranking [11].

There are different types of parameters for XGBoost:

- General Parameters: Depends on the boosting model - tree model, linear model
- Booster Parameters: Reply on the types of booster
- Learning Task parameters: Linear regression, Logistic regression
- Command Line Parameters: Number of rounds for boosting, path of training data

There are different features for XGBoost:

- Speed: It can do automatically parallel computation on windows and Linux, with openmp. It is faster than Gradient Boosting Machine (GBM).
- Input type: Different types of input data:
    - Dense matrix
    - Sparse matrix
    - Data files
- Customization: It supports customize evaluation and objective functions.
- Performance: Provides better performance on different kind of datasets
- Sparsity: It accepts sparse input for both linear and tree booster.

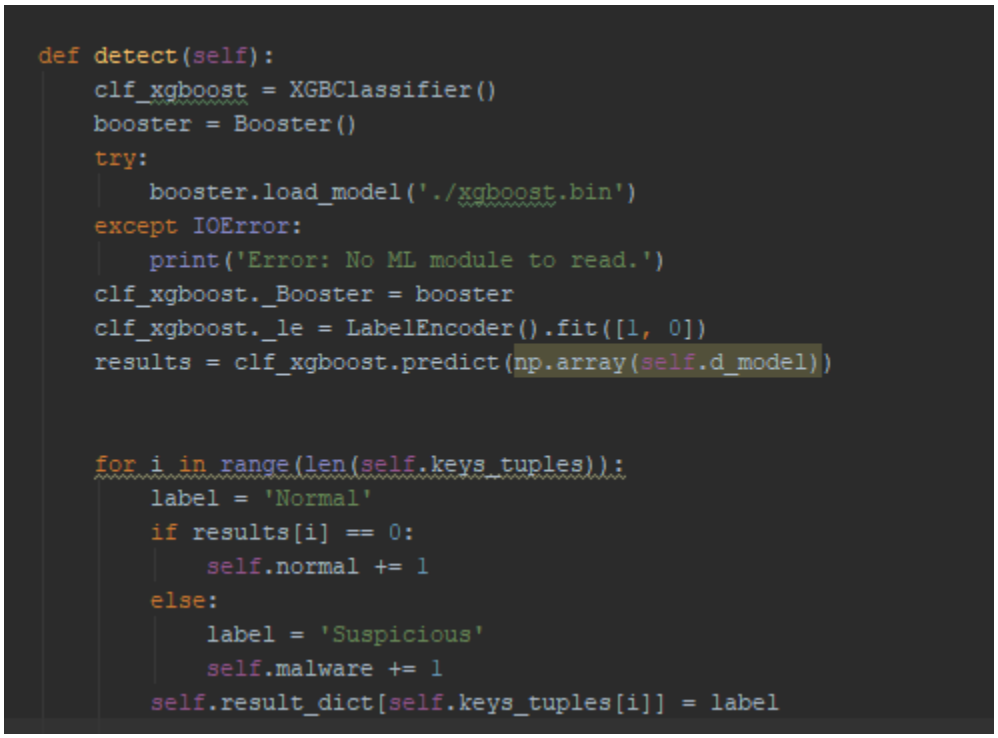
We use Scikit learn Python library. It is an open source machine learning library for Python. It provides the functionality for classification, regression and clustering algorithms. It supports different algorithms like: - support vector machines, gradient boosting, random forests, k-means [12].

XGBoost Regressor is used for continuous target variables. It is called regression problems.

XGBoost classifier is used for categorial target variables and is called as classification problems.

Step 4: Implement XGBoost algorithm

We used the XGBoost algorithm for predicting malware traffic from the given dataset. XGBoost algorithm is used primarily because of its execution speed and model performance. Boosting is a technique that uses new model to correct the existing models until no new correction can be made. In this MENG project, we have implemented the XGBoost algorithm from the scikit learn library. We used tree booster with logistic regression which is for binary classification. As shown in Figure 12, the XGBoost algorithm is implemented using the try-catch block and the data is classified as normal or suspicious.



```
def detect(self):
    clf_xgboost = XGBClassifier()
    booster = Booster()
    try:
        booster.load_model('./xgboost.bin')
    except IOError:
        print('Error: No ML module to read.')
    clf_xgboost._Booster = booster
    clf_xgboost._le = LabelEncoder().fit([1, 0])
    results = clf_xgboost.predict(np.array(self.d_model))


    for i in range(len(self.keys_tuples)):
        label = 'Normal'
        if results[i] == 0:
            self.normal += 1
        else:
            label = 'Suspicious'
            self.malware += 1
        self.result_dict[self.keys_tuples[i]] = label
```

Figure 12: XGBoost implementation

By providing any log files to this system, we can get the result as shown in below figure. Figure 12 depicts the classification results for the dataset. The figure contains 6 columns: the first column is providing the connection number, the subsequent 4 columns are connection 4-tuples, and the last column indicates the classification result: whether the connection is normal or suspicious.

#	SrcIP	DstIP	DstPort	Protocol	Label
1	1.114.49.118	147.32.84.229	443	tcp	Normal
2	1.152.2.27	147.32.84.229	443	tcp	Normal
3	1.152.84.147	147.32.84.229	443	tcp	Normal
4	1.153.149.215	147.32.84.229	443	tcp	Normal
5	1.153.155.184	147.32.84.229	443	tcp	Normal
6	1.153.40.130	147.32.84.229	443	tcp	Normal
7	1.39.35.202	147.32.84.215	443	tcp	Suspicious
8	1.39.95.68	147.32.84.229	443	tcp	Normal
9	1.41.151.63	147.32.84.229	443	tcp	Normal
10	1.41.205.94	147.32.84.229	443	tcp	Suspicious
11	1.42.168.134	147.32.84.229	443	tcp	Normal
12	1.43.137.224	147.32.84.229	443	tcp	Normal
13	1.43.51.164	147.32.84.229	443	tcp	Normal
14	1.44.164.101	147.32.84.229	443	tcp	Normal
15	1.44.180.190	147.32.84.229	443	tcp	Normal
16	1.66.100.3	147.32.84.229	443	tcp	Normal
17	1.72.2.205	147.32.84.229	443	tcp	Normal
18	1.72.4.23	147.32.84.229	443	tcp	Normal
19	101.108.8.144	147.32.87.33	443	tcp	Normal
20	101.109.145.26	147.32.84.229	443	tcp	Suspicious
21	101.11.47.190	147.32.84.229	443	tcp	Normal

Figure 13: Classification of the traffic as normal or suspicious

4.6 Evaluation Procedure and Results

Using the existing dataset, we extract the log files from the pcap files using Bro, and then extract and classify the features using XGBoost. An intermediate data model is constructed that consists of a matrix of values where each row corresponds to a connection 4-tuple ID and the columns are the corresponding feature values. The matrix consists of 12 columns and 6,135 rows. The evaluation is conducted through 5-fold cross validation, which means in each of the 5 rounds, 20% of the data is taken as a testing data and 80% as the training data. Table 2 shows the distribution of labels in the data model.

Connection-4 tuple	Training data	Testing data	Total data
Normal	1080	270	1350

Malware	3828	957	4785
Normal + Malware	4908	1027	6135

Table 2: Normal & Malware connection-4 tuples split into Training & Testing data

Through the evaluation experiment, we measure the performance of our detector using the following metrics: accuracy (ACC), detection rate (DR), and false positive rate (FPR). The definitions of the abovementioned metrics are given as follows:

$$ACC = \frac{\mathbf{TP + TN}}{\mathbf{TP + TN + FP + FN}} \times \mathbf{100}$$

$$FPR = \frac{\mathbf{FP}}{\mathbf{FP + TN}} \times \mathbf{100}$$

$$DR = \frac{\mathbf{TP}}{\mathbf{FN + TP}} \times \mathbf{100}$$

In the above equations, TP is True Positive, FP is False Positive, TN is True Negative, and FN is False Negative. The TP is when a record is detected as malware and it is malware; the FP is detected as malware, but it is normal; TN is detected as normal and it is normal; and FN is detected as normal, but it is malware. We used 5-fold cross-validation which splits the training data into 5 subsets on random basis and does the training and testing iteratively. In each iteration, the model is trained with 4 training subsets and tested to the remaining one subset. At the end of the rounds, the overall results are obtained by averaging the results from the rounds. Figure 13 describes the process of 5-fold cross-validation [7].

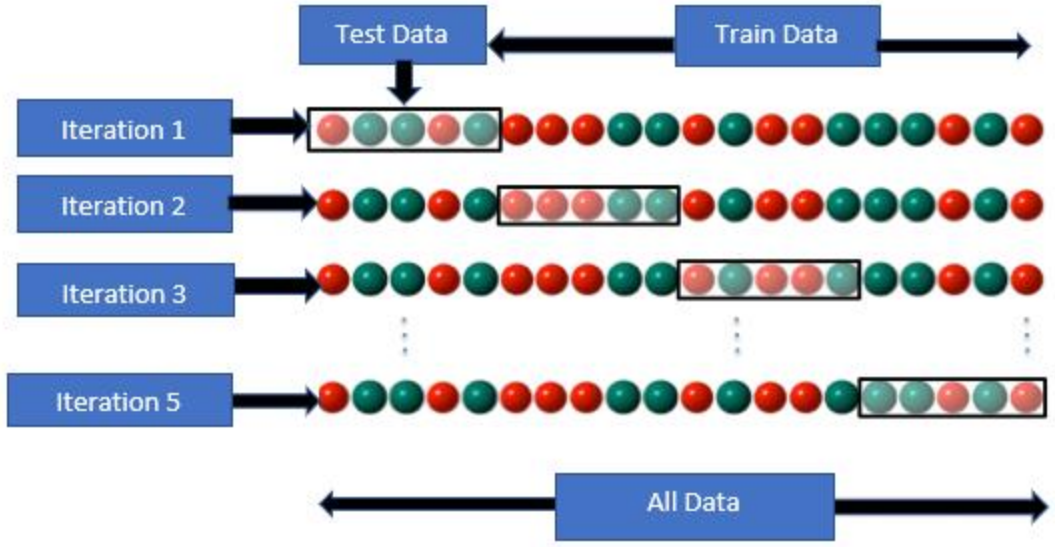


Figure 14: 5-fold cross-validation

Table 3 shows the performance results obtained for the different validation rounds, and their averages.

	TP	**TN**	**FP**	**FN**
Round 1	856	243	27	101
Round 2	859	231	39	98
Round 3	843	252	18	114
Round 4	871	249	21	86
Round 5	881	205	65	76
Average	862	236	34	95

Table 3: Test Results – Number of records detected under each category for each of the validation rounds

Figure 14 summarizes the evaluation results using a confusion matrix. The overall accuracy of the system is 89.48%, the FPR is 12.59%, and the DR is 90.07%.

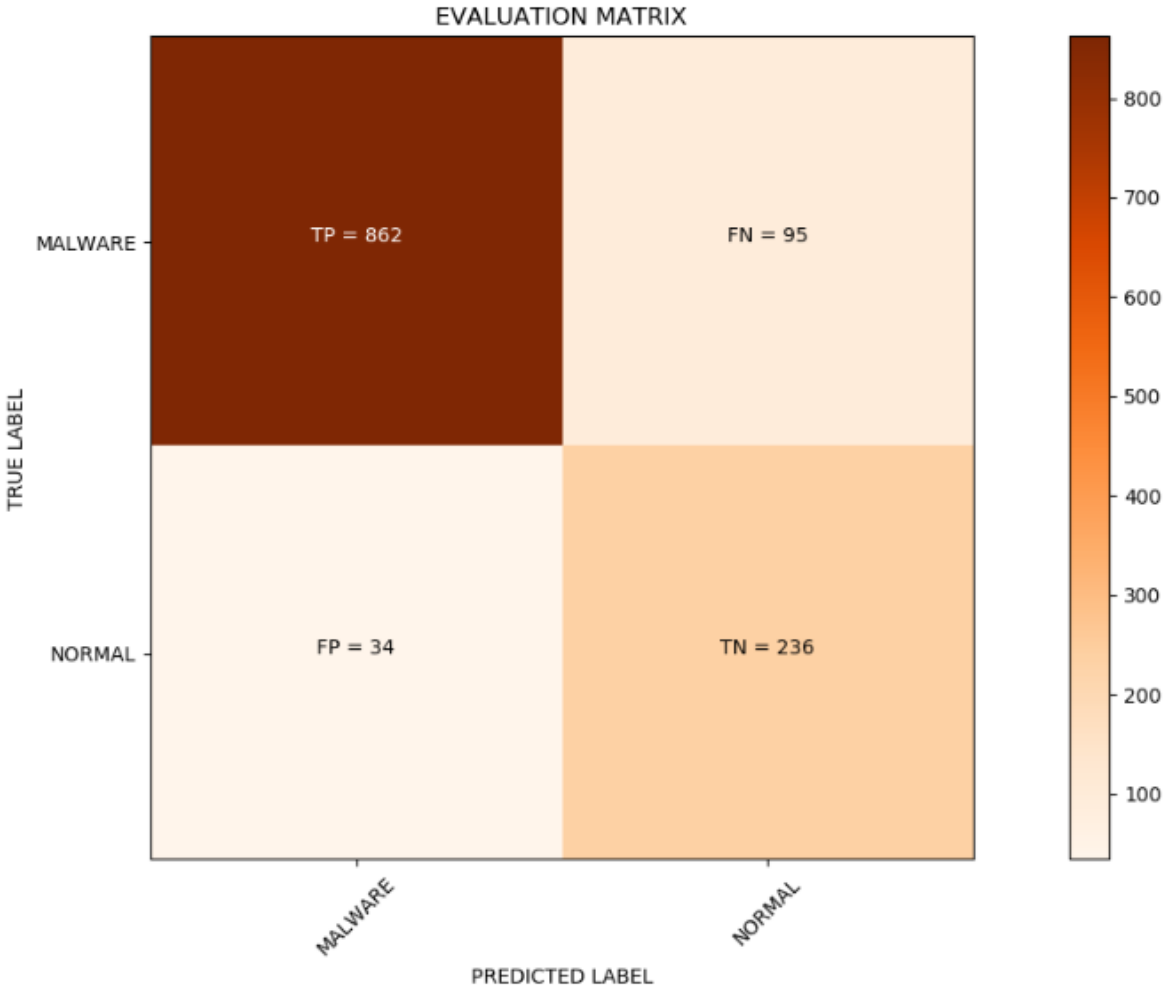


Figure 15: Evaluation results using confusion matrix

The obtained results are encouraging and indicate that we can differentiate normal and malware HTTPS usage based on the proposed feature model.

4.7 Advantages and disadvantages of the system

Advantages:

- The detection of malicious encrypted web traffic, without decrypting the traffic, and create features to distinguish normal and malware traffic are the main advantages of this system. The system is mainly use of three log files: - ssl, x509, and conn, generated by Bro IDS.
- The detection rate of the system is 90%, which is encouraging and fairly on higher side.
- The data model is robust, improvise with time and incoming traffic.
- The system is overcome with classical detection technique and provides the security and privacy to end users.

Disadvantages:

- The system is not tested under real time traffic.
- The false positive rate is 12%, which is little bit higher. But it can be reduced by discovering new features and analyze the remaining of the log files.

Chapter 5 Conclusion and Future Enhancements

The main aim of MENG project was to detect the malicious encrypted web traffic, without having to decrypt the traffic on the fly. Our proposed approach consists of extracting a set of features from Bro IDS logs to classify the normal and malware HTTPS traffic using machine learning.

We used XGBoost machine learning algorithm for classification. We evaluated our detection model using a public dataset containing a mixture of malware and benign network traffic and achieved an accuracy of 89.48%. This is encouraging and supports to some extent the claim that encrypted web traffic could be detected without decrypting the traffic on the fly.

Future Enhancements

Our goal in future work is to improve the accuracy of the algorithm. To help achieve such goal, we will extend the feature space by exploring new features from the remaining log files generated by Bro. We will also study other machine learning techniques such as random forest, support vector machine, and so on.

Furthermore, we will collect greater amount of data in order to confirm our current perform results.

References:

[1] https://transparencyreport.google.com/https/overview

[2] https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption

[3] https://link.springer.com/article/10.1007/s11416-017-0306-6

[4] David McGrew Blake Anderson, Subharthi Paul. Deciphering Malware's use of TLS (without Decryption). 6 Jul 2016. https://arxiv.org/pdf/1607.01639.pdf

[5] https://www.netresec.com/?page=PcapFiles malware PCAP files created by @moyix

[6] Bro documentation and installation guide, available at https://www.bro.org/.

[7] https://en.wikipedia.org/wiki/Cross-validation_(statistics)

[8] k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach

[9] https://en.wikipedia.org/wiki/Machine_learning

[10] https://www.techemergence.com/what-is-machine-learning/

[11] https://en.wikipedia.org/wiki/Xgboost

[12] https://xgboost.readthedocs.io/en/latest/parameter.html