Detecting Ransomware in Encrypted Web Traffic

Jaimin Modi[1], Issa Traore[1], Asem Ghaleb[1], Karim Ganame[2], Sherif Ahmed[3]

[1]University of Victoria, ECE Department, Victoria, BC Canada

jaiminmodi@uvic.ca, itraore@ece.uvic.ca, aalmekhlafy@gmail.com

[2]StreamScan, 2300 Rue Sherbrooke E, Montreal, QC, Canada

ganame@streamscan.io

[3]University of Windsor, Computer Science Department, Victoria, Canada

Sherif.SaadAhmed@uwindsor.ca

Abstract. To date, only a small amount of research has focused on detecting ransomware at the network level, and none of the published proposals have addressed the challenges raised by the fact that an increasing number of ransomware are using encrypted channels for communication with the command and control (C&C) server, mainly, over the HTTPS protocol. Despite the limited amount of ransomware-specific data available in network traffic, network-level detection represents a valuable extension of system-level detection as this would provide early indication of ransomware activities and allow disrupting such activities before serious damage can take place. To address the aforementioned gap, we propose, in the current paper, a new approach for detecting ransomware in encrypted network traffic that leverages network connections, certificate information and machine learning. We observe that network traffic characteristics can be divided into 3 categories – connection based, encryption based, and certificate based. Based on these characteristics, we explore a feature model that separates effectively ransomware traffic from normal traffic. We study three different classifiers: random forest, SVM and logistic regression. Experimental evaluation on a diversified dataset yields a detection rate of 99.9% and a false positive rate of 0% for random forest, the best performing of the three classifiers.

Keywords: ransomware detection, encrypted web traffic, machine learning, network traffic.

1 Introduction

Within the broad categories of malware, ransomware has gained a threatening popularity in the modern Internet world. Ransomware can broadly be categorized into 2 major types: locker ransomware and crypto ransomware. Locker ransomware locks the system and prevents users from using it. Crypto ransomware encrypts the files in the machine making them inaccessible to the user.

Maintaining adequate communication channel between an infected device and the command and control server (C&C) is an essential characteristic of modern malware. Increasingly, such communications are taking place using web protocols, such as HTTPS. According to a report by Cyren, a leading enterprise cybersecurity service

provider, 37% of malware now utilize HTTPS [2]. The same report also stated that every major ransomware family since 2016 has used HTTPS communication [2]. Although several approaches have been published in the literature for ransomware detection, the current approaches overwhelmingly focus on analyzing system data available on the endpoint with limited or no consideration for the C&C communications. Network level detection is made more complicated because, as aforementioned, malware communications are increasingly carried over HTTPS encrypted channels. Only a limited amount of research have been published on detecting malware in encrypted traffic [6] [7] [8] [9]. To the best of our knowledge, all the existing works have focused on general malware, while none of them has addressed the challenges pertaining to ransomware specifically. The fact that, to date, only a handful of works have been conducted on detecting ransomware at the network level could be due to the limited amount of ransomware-specific information available in network traffic data compared with system level data. On the one hand, because of the scarcity of such ransomware-specific data in network traffic, identifying and processing such information poses significant challenges. On the other hand, the development of an efficient and effective network level detector can represent a valuable extension of system level detection as this would provide early indication of ransomware activities and allow disrupting these activities before a serious damage can take place.

We propose, in the current paper, a new model for detecting ransomware from encrypted network traffic. The proposed model consists of a unified set of features that enables detecting ransomware from the network traffic data. We use an inspiration from a previous work on general malware detection from encrypted traffic by Strasak [11] to develop our initial feature model. The approach consists of generating initially log files from network captures using the BRO intrusion detection system (IDS). We construct the flows by aggregating network packets from the log files and calculate the feature values. The features are based on three dominant characteristics – connection based, encryption based, and certificate based. We utilize a wide variety of ransomware families and normal traffic data for constructing a diversified dataset to evaluate our approach. Three machine learning classifiers – random forest, SVM and logistic regression – are explored. Experimental evaluation results are presented for balanced and imbalanced datasets. The remainder of the paper is structured as follows. Section 2 summarizes and discusses related works. Section 3 presents our datasets. Section 4 describes our feature model. Section 5 presents the experimental evaluation of our proposed approach. Section 6 makes concluding remark.

2 Related Works

Ransomware detection has been studied extensively in recent years. The existing works can be divided into three major categories: static, dynamic and hybrid. While conducting a study on ransomware detection using static analysis, Kharaz observed that only 10 engines out of 60 could detect ransomware [1]. As a result, many of the existing ransomware detection approaches use dynamic or hybrid analysis. Dynamic approaches can be divided into system level and network approaches.

As mentioned above, the majority of the dynamic approaches work at the system level. Examples of such proposals include UNVEIL by Kharaz et al. [12], ShieldFS by Continella et al. [13] and ELDERAN by Sgandurra et al. [14]. A much smaller number of proposals have been published on network-based dynamic analysis, including works by Cabaj et al. [3], Omar et al. [5], Almashhadani et al. [4], Wan et al. [10] and Salehi and colleagues [15]. We revisit those related work in the following.

In 2018, Cabaj and colleagues [3] studied two crypto ransomware families – cryptowall and locky. The study explored the network communication characteristics for both ransomware families. They observed that the interactions of ransomware are based on HTTP POST messages in 3 major stages – registration with the C&C server, exchanging public keys for encryption, and acknowledging successful encryption. They proposed a Software Defined Networking (SDN)-based detection approach where a vector containing the payload values from the aforementioned 3 stages is built and the corresponding centroid is calculated. After optimizing a threshold value, new samples can be predicted based on the value of the centroid. The experimental evaluation was based on 250 samples of cryptowall and locky each, and around 13,000 normal samples. The evaluation yielded a detection rate of 97%-98% with 4%-5% of false positives. The detection system was designed such that it can measure the threshold value for the payload. Such system can be evaded by ransomware authors by dividing the ransomware payload amount such that it appears as normal payload to the detection system. Also, the experiment was based only on 2 ransomware families.

Omar et al. [5] introduced NetConverse, a model to detect Windows ransomware using machine learning classification. The model aggregates packets into unique conversations based on a 5-tuple: protocol, source/destination IP address, and source/destination port values. The authors extracted 9 feature values, and used a dataset consisting of 210 ransomware samples and 264 goodware samples collected from VirusTotal. Various machine learning classifiers were studied including Bayes Network, Multilayer perceptron, J48, KNN, Random forest, and LMT. The accuracy of 97.1% was obtained using J48 classifier (their best performing classifier) with a false positive rate of 1.60%. Despite the relatively high accuracy, the feature model consists of very basic features, such as the total number of bytes, duration, and IP addresses. Modern ransomware can easily evade such feature model.

Almashhadani et al. [4] presented a comprehensive behavioural analysis by using locky ransomware for case study. The authors used a dataset containing 6 locky ransomware samples and 10 normal samples collected from various trustable sources. They extracted 18 features that were further classified into packet-level and flow-level features. Random forest, LibSVM, Bayes network and random tree machine learning classifiers were used for prediction. Random tree achieved the best accuracy at 97.92% for packet-based feature model with a false positive rate of 0.021%. For flow-based feature model, Bayes network achieved the highest accuracy at 97.08% with 0.029% false positive rate. However, the model was based only on a single ransomware family, which is not very representative of the current state of the ransomware landscape in which many other families are prevalent.

Wan et al. [10] proposed a flow-based system called Biflow by aggregating the packet-based data. The exact size of the dataset was not provided, but samples from only 2 ransomware families – locky and cerber – were used for the experiment. After selecting 36 features, the J48 decision tree machine learning classifier was trained on the data. A maximum precision of 90.19% was achieved when 19 features were selected and selecting 36 features gave 89.12% precision. The experiment, however, does not provide enough proof on the results obtained in terms of detection rate and false positive rate. Also, the experiments are based on just 2 ransomware families.

Salehi and colleagues [15] worked on detecting ransomware using Domain Generation Algorithms (DGA). They extracted 3 classes of features for the detection engine – random and gibberish characters in domain, frequency of different domains, and replication of same domains. The detection engine also includes a black/white domain list to decrease the number of false positives. The model achieved a detection rate of 100% on 20 ransomware samples. The false positive rate and false negative rate were both zero. However, the model was featured such that it detects only DGA based samples and cannot detect any non-DGA based ransomware.

3 Evaluation Datasets

To the best of our knowledge, there is no publicly available ransomware detection dataset. It has been decided at the ISOT lab to fill this gap by collecting a new dataset to be shared with the research community. The collected dataset includes both network and file activities generated from ransomware as well as benign applications.

3.1 Ransomware Dataset

All the ransomware binaries were collected from the well-known antivirus aggregator VirusTotal [16]. The binaries collected in the ISOT dataset consist of 666 different samples from 20 different ransomware families. These families are the most prevalent versions of ransomware currently available. Table 1 provides a detailed breakdown of the ransomware samples from each family.

Family	Samp	%	Family	Samp	%
TeslaCrypt	348	52.3	Mole	4	0.597
Cerber	122	18.236	Satan	2	0.299
Locky	129	19.28	CTBLocker	2	0.299
CryptoShield	4	0.597	Win32.Blocker	18	2.69
Unlock26	3	0.448	Spora	5	0.747
WannaCry	1	0.149	Jaff	3	0.448
CryptoMix	2	0.299	Zeta	2	0.149
Sage	5	0.747	Striked	1	0.149
Petya	2	0.299	GlobeImposter	4	0.598
Crysis	8	1.196	Xorist	2	0.299
Flawed	1	0.149			
Total	**666**				

Table 1 – Ransomware dataset breakdown

All binaries were executed following established and commonly agreed principles inside an open source automated malware analysis system – the cuckoo sandbox [17]. During the execution of the ransomware samples, various data items were collected, including the list of operations performed on the file system, the system calls initiated, the processes initiated, the operations on the Windows registry, the full memory dump of the infected machine, and the list of files targeted. Among the aforementioned data, our interest in the current work lie specially on the network traffic dump generated during the analysis of each sample.

3.2 Normal Dataset

The normal (i.e., benign) dataset was collected in 2 different ways - manual data collection and network files from the Stratosphere project [18].

The manual data collection was done while keeping Wireshark on capture mode and visiting Alexa top 100 websites. The duration for each capture was 30 minutes and 30 total samples were collected. Each sample was collected by manually browsing through the websites on Alexa top 100 list.

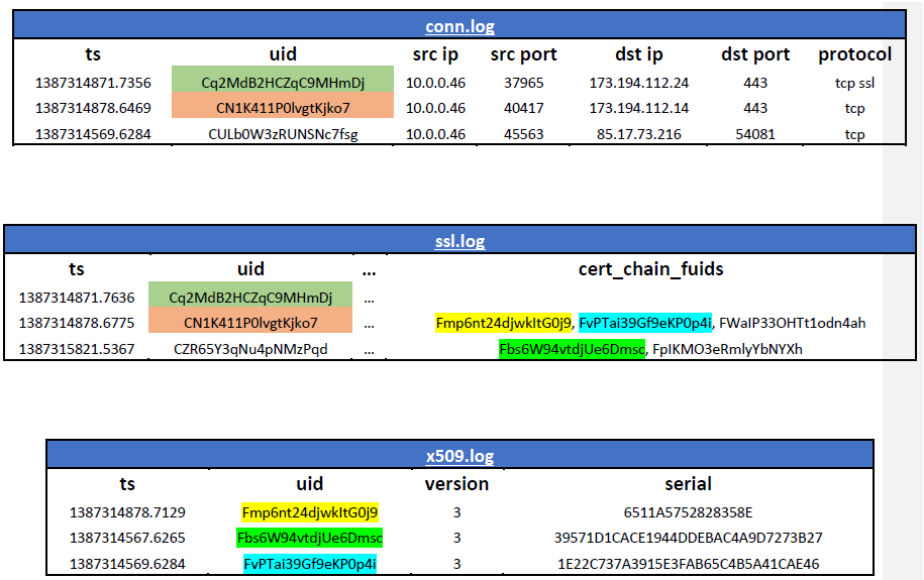
The Stratosphere Intrusion Prevention System (IPS) project has a sister project called Malware Capture Facility Project [18] that is responsible for capturing all kinds of data. The data is organized in a specific directory structure for easy access. We wrote a python script to download 10 pcap files from the Stratosphere dataset web portal [19].

3.3 Network Activity Data from Bro IDS Log Files

The Bro IDS is an open-source framework that, besides its intrusion detection capability, is most commonly used as network traffic analyzer [20]. Amongst its many features, the one that we utilise in the current research is the comprehensive set of log files that record network activities. We use for our feature extraction three of the Bro generated log files, which contain adequate information about the encrypted traffic. More details on the three log files are provided as follows:

1. *Conn.log – Connection record:* Each line in conn.log contains information about packets that flow between two IP addresses. It contains multiple columns such as IP address, protocols, ports, number of packets, duration, label, etc.
2. *Ssl.log – SSL record:* This log file contains all information related to the SSL/TLS handshake that helps encrypt all the data between 2 endpoints. It has also the information about which certificates were used during the encryption process. It includes important columns such as SSL/TLS version number, server name, cipher suite used and so forth.
3. *X509.log – Certificates record:* This log file contains all information related to the certificate records involved in the SSL handshake. The columns describe information such as validity time, certificate serial number, signature algorithms used and so forth.

The advantage of using the log files is that they are interconnected with each other through unique identifiers (id). This helps in extracting complete information for any given packet for further inspection.



conn.log

ts	uid	src ip	src port	dst ip	dst port	protocol
1387314871.7356	Cq2MdB2HCZqC9MHmDj	10.0.0.46	37965	173.194.112.24	443	tcp ssl
1387314878.6469	CN1K411P0lvgtKjko7	10.0.0.46	40417	173.194.112.14	443	tcp
1387314569.6284	CULb0W3zRUNSNc7fsg	10.0.0.46	45563	85.17.73.216	54081	tcp

ssl.log

ts	uid	...	cert_chain_fuids
1387314871.7636	Cq2MdB2HCZqC9MHmDj	...	
1387314878.6775	CN1K411P0lvgtKjko7	...	Fmp6nt24djwkItG0j9, FvPTai39Gf9eKP0p4i, FWaIP33OHTt1odn4ah
1387315821.5367	CZR65Y3qNu4pNMzPqd	...	Fbs6W94vtdjUe6Dmsc, FpIKMO3eRmlyYbNYXh

x509.log

ts	uid	version	serial
1387314878.7129	Fmp6nt24djwkItG0j9	3	6511A5752828358E
1387314567.6265	Fbs6W94vtdjUe6Dmsc	3	39571D1CACE1944DDEBAC4A9D7273B27
1387314569.6284	FvPTai39Gf9eKP0p4i	3	1E22C737A3915E3FAB65C4B5A41CAE46

Figure 1– Interconnection of log files

The log files are easy to load as data frame and hence the connections can easily be analysed. Figure 1 explains the interconnection of log files in detail. The 'uid' column connects the conn.log and ssl.log files. As shown in the Figure 1, uid | *Cq2MdB2HCZqC9MHmDj*| connects these 2 log files. The 'cert_chain_fuids' column in the ssl.log connects all the corresponding certificate records in the x509.log file.

Log file	Count		
	Normal	**Ransomware**	**Total**
Connection records (conn.log)	1,010,157	883,582	1,893,739
SSL records (ssl.log)	79,832	10,890	90,722
Certificate records (x509.log)	63,159	13,604	76,763
Total	1,153,148	908,076	2,061,224

Table 2 – Total number of records from log files in the dataset

The certificates are issued by certificate authorities (CA) of two types – root CA and intermediate CA. A browser considers a CA to be trusted if it is included in the browser trusted CA list. Not all intermediate CAs are present in the browser trusted list. For an

intermediate CA to be trusted, it should have been issued by another trusted intermediate CA or the root CA. When a user visits a website, the browser checks if the CA is included in the browser trusted CA list. If not, then it will check the certificate of the issuing CA. This way it continues in the certificate chain until it finds a CA that is present in the trusted list. This chain is present in the ssl.log file as the '*cert_chain_fuids*' column. Each unique value in this column has a corresponding certificate record details in the x509.log file.

We collected a total of 45 normal samples and 666 ransomware samples. These samples, when processed through the Bro IDS give out log files. Table 2 shows the final numbers of records involved in the combined dataset in terms of each log file.

4 Feature Model

In this section, we present our feature model, and explain in detail the creation of a flow dataset from the Bro log files.

4.1 Creating flows

After collecting the log files through Bro IDS, the next step is to preprocess the data and calculate the feature values. Before calculating the feature values, it is important to build the flows from the log files. A flow is defined as the set of unidirectional packets observed over some predefined time window that share some common attributes [21]. In our work, a flow is defined as a collection of packets that share the same source IP, destination IP, source port, destination port and protocol. There are multiple log files that are generated from Bro IDS depending upon the type of pcap files provided as input. The three major log files that are used for building flows in the current work are – ssl.log, conn.log and x509.log. Since the traffic is encrypted, we start by building our flow from the ssl.log file. An overview of the connection information provided by the ssl.log, conn.log and x509.log files is provided in Figure 2.

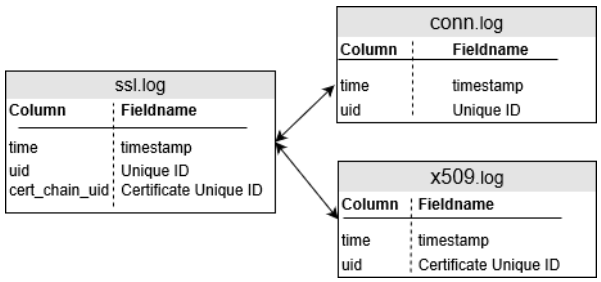


Figure 2 – Bro log files connection information

The important point while building the flows is to connect all the log files. As it can be seen from Figure 2, ssl.log and conn.log files can be connected through the Unique ID of connection (UID) column. The UID is a string that is assigned by Bro to each packet while separating it from pcap files. There are some connections that do not need to be secured and so they may be present only in the conn.log. Hence not all values of

‘uid’ column in conn.log file are present in the ssl.log file. Similarly, the ssl.log and x509.log files can be connected using the certificate identifier (cert_id) column from ssl.log file. The ‘cert_chain_uid’ column in the ssl.log file is a vector of strings such that each string value is present in the ‘id’ column of x509.log file. If the ‘cert_id’ column is empty, then the corresponding packet does not contain any certificate related information. Ultimately for a given flow, three different types of values will be added to it as the key: ssl values, connection values and certificate values. The basic algorithm for building a flow is as follows:

1. Initialise an empty dictionary named flow dictionary. Beginning from the first packet of ssl.log file, look up for the ‘uid’ value in the conn.log file. Extract the 5 attributes – source IP, destination IP, source port, destination port, and protocol – from conn.log file. These 5 attributes will act as the tuple. Check whether the tuple exists in the flow dictionary. If not, then add the tuple to the flow dictionary. The tuple acts as the key.

2. Each log file consists of column values that are based on their corresponding properties. Add the corresponding column values from conn.log and ssl.log file. Add the associated label from the conn.log file in the end. Note that the tuple acts as the common key for fetching values in both log files.

3. Using the ‘cert_id’ column value/s from ssl.log file, extract all column values from x509.log file. Add the values to the certificate dictionary with id acting as the primary key. If the column value is empty, move to the next step.

4. Follow steps 1-3 until all the packets are scanned in ssl.log file.

5. For the values of ‘id’ column in conn.log file that are not present in the ssl.log file, search if the tuple exists as key value in the flow dictionary. If it exists, then add the data as the connection values. If the tuple does not exist, create a new tuple and add it to the flow dictionary. These tuples will only contain column values from conn.log file. Since no record of this tuple exists in ssl.log and x509.log files, -1 value is inserted for such cases.

Figure 3 depicts a snapshot of the three log files from the Stratosphere (normal) dataset. The arrows show the links between the three log files. In the end, the packets that share the 5 common attributes – source & destination IP, source & destination port, protocol – are combined to form a flow. The corresponding values are used to calculate the features. To illustrate flow building, let’s use the sample data shown in Figure 3. To build a flow from the given data, we start with the data from ssl.log file and execute the following steps:

1. Since the flow dictionary is empty, we start by scanning the packets from the first row of the ssl.log file. The ‘uid’ column value *CcXBkT1wBYk452YOMb* is looked up in the conn.log table.

2. The flow tuple (10.0.0.46, 173.194.112.14, 40417, 443, tcp) is formed from the conn.log file for the corresponding ‘uid’ look up. This tuple is then added

to the flow dictionary. Note that the flow tuple acts as the common key for fetching data in both log files.

3. The current ssl packet has three values associated in the 'cert_id' column. These values are matched with the 'id' column in the x509.log table. These values are then added to the certificate dictionary with id acting as the key in the dictionary.

4. Repeat steps 1-3 until all packets in ssl.log are scanned. Add the corresponding ssl and connection values in the respective dictionaries.

5. In the end, notice the highlighted row in the conn.log file. The 'uid' value *CBZhyK2S1nJDzisEja* is not present in the ssl.log file. Here we check if the related tuple (10.0.0.46, 88.171.246.178, 59795, 80, tcp) is present in the flow dictionary. If yes, then the connection column values will be added to the dictionary; otherwise a new tuple key will be created.

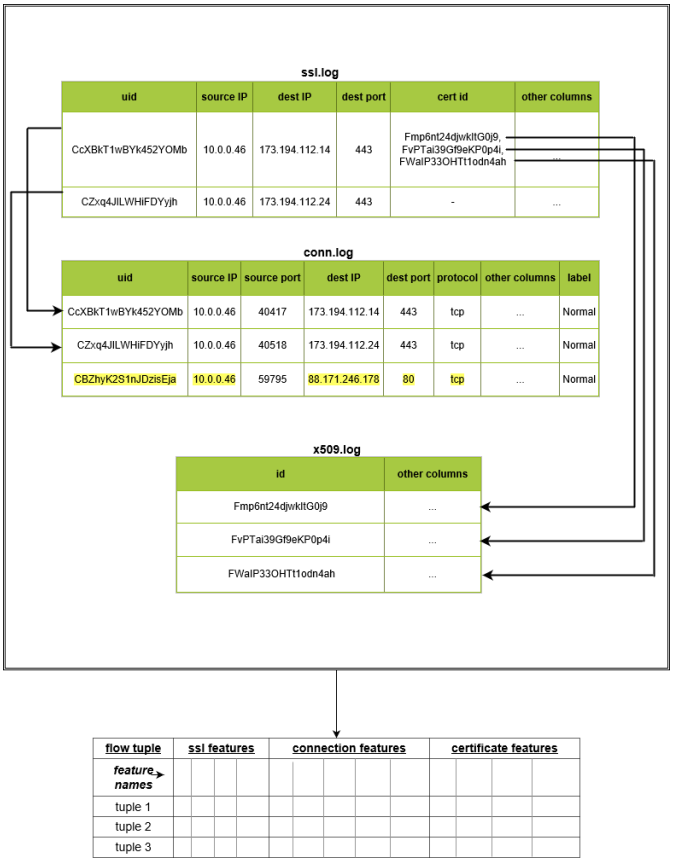


Figure 3 – Connecting log files to create a flow dataset

The final dataset is prepared based on the flow mechanism explained above.

4.2 Feature Definitions

Our feature model consists of 3 groups of features carved out from each of the log files. In total, the model consists of 28 features, which are calculated from the flows. We describe in detail each of the features in the following.

4.2.1 Conn.log features

Number of flows – Each flow is an aggregation of ssl and non-ssl (connection) packets. This feature represents the total number of packets in each flow.

Average duration of flows – Each packet in a flow has a value that is the duration it took in seconds. The duration value for each packet of a given flow is stored in a list. In the end, the mean is calculated for the list. This mean value is represented by this feature.

$$Average\ duration = \frac{\sum(duration\ of\ each\ packet)}{total\ number\ of\ packets}$$

Standard deviation of flows duration – Using the same list as above, this feature calculates the standard deviation for each flow.

$$Duration\ standard\ deviation = \sqrt{\frac{\sum |duration\ of\ packet_i - average\ duration|^2}{total\ number\ of\ packets}}$$

Percent of standard deviation of flows – For each flow, we now have a standard deviation and average value of the duration. Using the standard deviation and average duration values, the upper limit and lower limit are calculated. The lower limit is the *(average – standard deviation)* and the upper limit is the *(average + standard deviation)*. For a given flow, the values of packets which are out of this defined limit are counted. The average is computed by taking the ratio of the count to the total number of packets in a flow. For example, if there are 5 packets in a flow and there are 2 packets which are outside of the defined limits, the value of this feature for that flow is 2/5 = 0.4.

Originator payload size – The size of the payload sent by the originator for each packet in a given flow is extracted from the conn.log file. This feature adds up all those values for all the connection records in a given flow.

Responder payload size – This feature is like the above feature. Except here we add the payload size of each packet sent back by the responder.

Ratio of responder to originator payload size – The final values for the above two features are used here. This feature calculates the ratio of the total bytes sent by the responder to the total sent by the originator for a given flow.

Percentage of established connections – The conn.log file provides information about the state of connection for each packet. There are 13 possible states of connection. Detailed information about what each state represents can be found out in Bro official documentation [20]. Based on that information and the states of the packets, they are classified as either established or non-established. Connection states [S0, S1, SF, REJ, S2, S3, RSTO, RSTR] are labeled as established connections and [RSTOS0, RSTRH, SH, SHR, OTH] are labeled as non-established connections. The ratio is calculated as:

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

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

Inbound packets – This feature corresponds to the total number of packets sent by the responder. After establishing the flow, this feature is the total number of packets in that flow.

Outbound packets – This corresponds to the total number of packets sent by the host. After establishing the flow, this feature is the total number of packets in that flow.

Periodicity average – Every packet comes with a timestamp. The timestamp for each packet is unique and it can be used to determine the periodicity in a given flow. For a given flow, the time difference between all the packets is calculated in the ascending order of their times. The time difference is again calculated and stored in a list. The mean is calculated in the end. Figure 4 explains the calculation in detail.

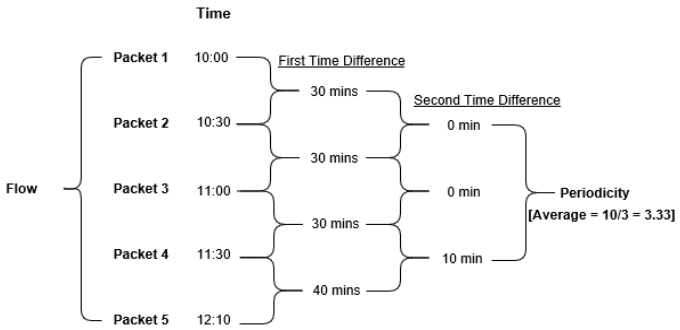


Figure 4 – Computing periodicity in a flow

Periodicity standard deviation – Using the same list as above, this feature calculates the standard deviation for the periodicity of the packets in a flow.

4.2.2 Ssl.log features

SSL flow ratio – Some packets in a flow are ssl packets while others are connection (non-ssl) packets. This feature is the ratio of ssl packets count to the non-ssl packets count.

SSL-TLS ratio – Secure Socket Layer (SSL) is the protocol used for encrypting the web traffic between the server and browser. Transport Layer Security (TLS) is an upgraded protocol on top of SSL. Since SSL protocol is less secure, ransomware authors try to use it for communication with the host. This feature is the ratio of the number of TLS packets to the total number of SSL and TLS packets in a flow.

$$R = \frac{TLS}{TLS + SSL}$$

SNI-SSL ratio – Server Name Indication (SNI) allows the server to host certificates for multiple websites under the same IP address. When a browser connects to a website whose IP address holds multiple websites, SNI helps in connecting the browser to that specific site. This way, the browser receives the certificate for that website only. However, it has been observed that the value of this feature is '-' for ransomware packets rather than the name of the host in case of normal packets. A counter is incremented if the value of SNI is '-'. This feature calculates the ratio of the number of SNI values to the total number of SSL packets in the flow.

SNI as IP – Sometimes, the SNI is not able to resolve the hostname and it just stores the IP address as its value. For a given flow, this feature has three possible values:

- -1: If the value of SNI for any packet in a flow is equal to the IP address but not equal to the destination IP address.
- 0: If the value of SNI for any packet in a flow is equal to the IP address and also equal to the destination IP address.
- 1: If the value of SNI for any packet in a flow does not contain any kind of IP address.

Certificate path average – The certificate path contains multiple values based on the fact that a browser traces untill it finds a trusted authority. The total number of certificates that the browser traced is present in the ssl.log file. This feature computes the average of the total number of certificates present in the certificate path field.

Self-signed certificate ratio – Many organizations use self-signed certificates for SSL encryption instead of using the CA authorities to avoid spending on related costs. Ransomware authors may take advantage of this fact and issue self-signed certificates. Bro is able to determine if any communication has a sel- signed certificate. It will create a column 'self_signed_certificate' and increment the count. This feature calculates the ratio of this count to the total number of certificates in a flow.

4.2.3 X509.log features

Public key average – The certificate log file has a feature that describes the length of the pubic key in terms of bits. This feature counts the length of the key for each packet and then calculates the average for the flow.

Average certificate validity – All certificates have a validity date. This is mentioned in the certificate log file as the valid before and after dates. Using these two features, the validity is calculated for each packet in terms of days and stored in a list for each incoming log. Finally, the mean is calculated for the given flow.

Standard deviation of certificate validity – This feature uses the same list as above for calculating the validity. However, this feature computes the standard deviation instead of the mean.

Certificate validity during capture – This feature uses the timestamp value of the certificate log file. The timestamp defines the time during which the corresponding packets were captured. This feature is a binary value as to whether the certificate was valid at the time of capture or not. Figure 5 explains the concept in detail.

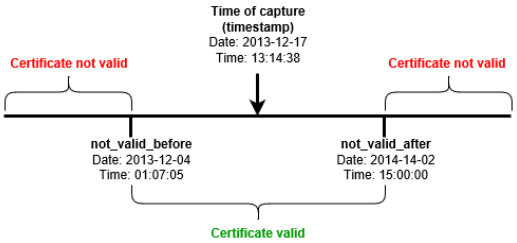


Figure 5 – Determining the validity of a certificate from the time of capture

Average certificate age – This feature calculates the ratio of two certificate time periods. Figure 6 explains the two time periods. The first time period is the difference between the time of capture and the issue date of the certificate. The second time period is the duration of the certificate validity period. This ratio is called certificate age. In the end, the average is computed for all certificate age values in a flow.

$$Certificate\ age = \frac{first\ time\ period}{second\ time\ period}$$

$$Average\ Certificate\ age = \frac{\sum Certificate\ age}{Total\ packets\ in\ a\ flow}$$

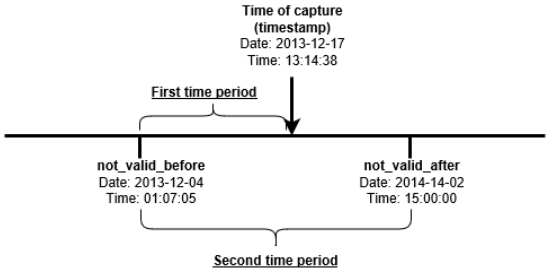


Figure 6 – Determining certificate age for a packet

Number of certificates – The certificate serial number is the unique number through which any certificate can be identified. This value is used to count the number of certificates for each flow in the dataset.

Average number of domains in SAN – Subject Alternative Name (SAN) helps in combining multiple host names into one common certificate. This is helpful as organizations do not have to issue new certificate for every new host name. This feature stores the count of different host names for every incoming packet and calculates the average in the end.

$$Average\ domains\ in\ SAN = \frac{\sum(different\ host\ names\ in\ all\ packets)}{Total\ packets\ in\ a\ flow}$$

Certificate-ssl logs ratio – All records related to a certificate can be traced through its unique id in ssl log data. However, not all ssl logs have certificate related data. A flow contains packets that may have logs related to just ssl or ssl and certificate combined. This feature is the ratio of the total number of certificate logs to its corresponding ssl logs for a flow.

SNI in SAN DNS – Server Name Indication (SNI) contains the host name. Subject Alternative Name (SAN) contains the name of multiple host names within the same certificate. Generally, the value of SNI should be present in the SAN DNS column of certificate log data. The value of this feature is 0 if not present and 1 if present.

CN in SAN DNS – The Common Name (CN) value in the certificate log file contains the information about the host name and details of the certificate for the packet. Generally, this host name value should be present in the SAN DNS column of certificate log data. The value of this feature is 0 is not present and 1 if present.

5 Classification and Evaluation Experiments

In this section, we conduct several experiments to assess the effectiveness of our proposed approach. We explain in detail the data pre-processing steps, and then calculate and discuss the obtained performance results.

5.1 Data Preparation/Preprocessing

Splitting the data: The data can be divided using two methods as follows:

- Train-test split: consists of randomly splitting the data over certain ratio.
- K-fold cross validation: the data is divided into K equal parts. (K-1) samples are used for training and K samples are used for testing.

We explored both splitting options in our work. In the first case, the dataset was divided into an 80:20 ratio, i.e., 80% data was the training data and 20% data was set aside for testing, while in the second case, we used the K value as 10.

Data Transformation: The values for most of the features in the dataset have varying ranges. The purpose of data transformation is to bring all the features into a similar range while not distorting the information conveyed within the data. This is also known as feature scaling. Since our dataset contains negative values that have significant importance, we utilize the min-max scaling technique. The feature values are transformed into range of [-1, 1] after data transformation.

Data Imbalance: For our dataset, the ratio of ransomware to normal is approximately 1:6. Hence, we analyze our classifiers from two perspectives – current dataset as it is and balanced dataset analysis using the Synthetic Minority Oversampling (SMOTE) oversampling technique.

Hyper-parameter Tuning: All machine learning algorithms have parameters that can be tuned depending on the problem, and tuning these parameters for a classifier is called hyper-parameter tuning. Setting the right values for hyper-parameters can give the best output from the model. We utilize the grid searching technique to tune the hyper-parameters. In this technique, as the name suggest, the algorithm loops through the grid for all possible combinations of model parameters. In the end, it gives out the combination that gives the best output.

5.2 Experiment Results

In the current paper, we use and compare three different classification algorithms, including logistic regression (LR), random forest (RF) and support vector machine (SVM). After extracting the features from the pcap files, we prepare the dataset for training the machine learning classifiers. Each row in the data is a flow represented by the tuple followed by corresponding feature values. The dataset has 15,524 rows of data and 30 columns, out of which 13,058 samples are normal and 2,466 samples are ransomware. This means the percentages of normal and ransomware data are approximately 85% and 15%, respectively.

Classifier	**Accuracy (%)**	
	Without SMOTE	**With SMOTE**
Logistic Regression	94	94
Random Forest	100	100
Support Vector Machine	95	94

Table 3 – Accuracy score for the classifiers for 'train_test_split' set

Classifier	**(Average) Accuracy (%)**	
	Without SMOTE	**With SMOTE**
Logistic Regression	91	96
Random Forest	98	100
Support Vector Machine	91	96

Table 4 – Accuracy score for the classifiers for 10-fold cross validation set

As discussed in the previous section, our current feature model has 28 features from three different log files. An extra column is included in the dataset which mentions the family of the ransomware for all the samples. Table 3 shows the accuracy results for all classifiers with and without smote for the '*train_test_split*' set. Similarly, Table 4 shows the accuracy for the 10-fold cross validation. We present and discuss the performance results for each of the separate classifiers in the following.

Logistics Regression: Table 3 shows that smote has no effect on accuracy for '*train_test_split'* method. However, by looking at the confusion matrix shown in Figure 7, it can be observed that smote reduces the number of false negatives by a considerable amount. It also increases the number of false positives. False negatives are highly dangerous for any system. On the other hand, false positives can be cumbersome and costly in handling them. In general, a trade-off must be made between these two metrics. Table 6 shows the classification report for the logistic regression classifier.

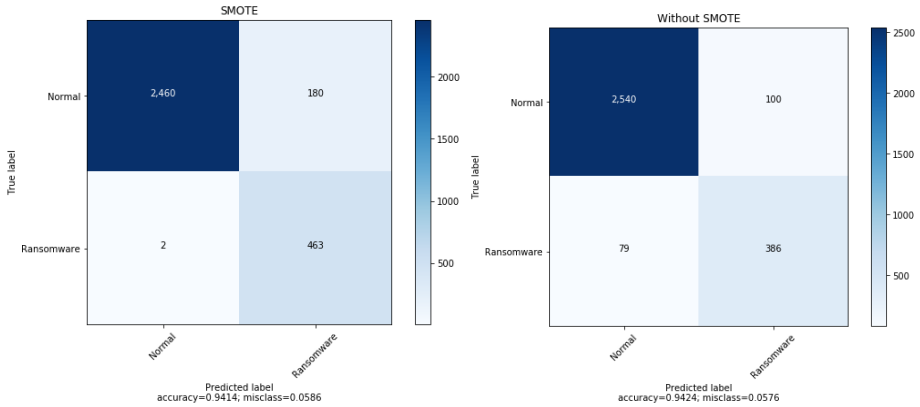


Figure 7– Comparing the confusion matrices for the logistic regression classifier (with and without SMOTE)

Classification Report				
Metrics	**SMOTE**		**Without SMOTE**	
	Normal	Ransomware	Normal	Ransomware
Precision (%)	100	72	97	79
Recall (%)	93	100	96	83
f1 - score (%)	96	84	97	81
Support	2640	465	2640	465

Table 6 – Classification report for the logistic regression classifier

Since applying smote on the dataset gives better output, we perform hyper-parameter tuning only for that dataset. The following model parameters were utilized for tuning:

a. *C* (Inverse of regularization strength) – Often when the dataset is small and the number of parameters is high, the model tends to overfit the data. This increases the prediction error on the test data. Regularization alleviates overfitting by decreasing the complexity of the model. We vary the value of C from 0.0001 to 1000.

b. *Penalty* – The regularization can add a penalty to the parameter magnitude in multiple ways. We explore two possibilities in our work called L1 and L2 regularizations as follows:

- L1 regularization – This is also known as Lasso regression. This regularization adds absolute value of magnitude of coefficient as penalty term to the loss function.

- L2 regularization – This is also known as Ridge regression. This is like Lasso regression except it adds squared magnitude of coefficient as penalty term to the loss function.

After doing a grid search through the parameters, the logistic regression gives best accuracy at 96% for the following parameter values: *C = 1* and *Penalty = L1*. This means that accuracy does not change, however, by looking at the confusion matrix shown in Figure 8 it can be observed that there is a decrease in false positive as well as false negative rates.

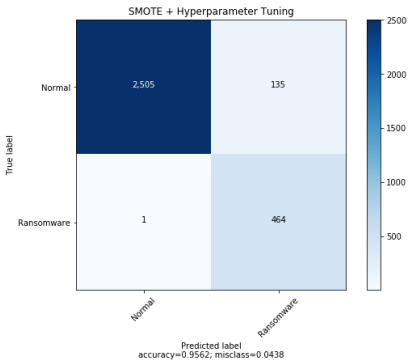


Figure 8– Confusion matrix for the logistic regression classifier after applying hyper-parameter tuning

Random Forest: Random forest gives the best accuracy amongst all the three classifiers.

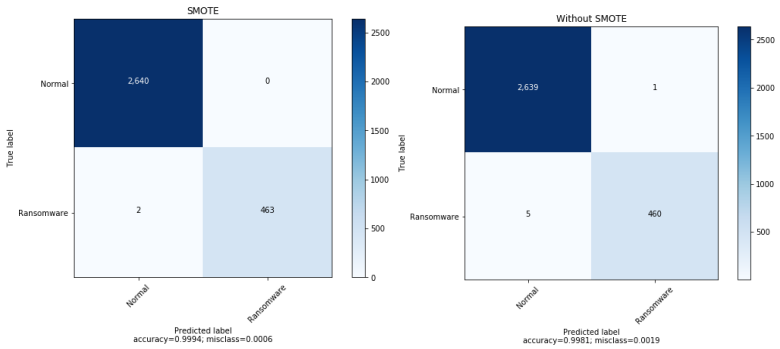


Figure 9– Comparing the confusion matrices for the random forest classifier (with and without SMOTE)

Classification Report				
Metrics	**SMOTE**		**Without SMOTE**	
	Normal	Ransomware	Normal	Ransomware
Precision (%)	100	100	100	100
Recall (%)	1.00	1.00	1.00	99
f1 - score (%)	100	100	100	100
Support	2640	465	2640	465

Table 7 – Classification report for the random forest classifier

As discussed above, false negatives should be reduced and smote specifically helps in doing that. Although the number of false negatives is not very high, it is important to reduce them to the lowest possible. Figure 9 and Table 7 show the classification results for the random forest classifier. The random forest classifier achieves an accuracy of 100% and a detection rate of almost 100% with an FPR of 0%. We obtain a FPR of 0% since we have no false positives. The only reason for applying hyper-parameter tuning is to check if this can reduce the false negatives to 0. That way we can achieve perfect detection rate. The parameters considered for tuning were as follows:

a. *n_estimators* – Random forest comprises of creating multiple decision trees. This parameter controls the number of trees to be used in the classifier. We vary the value of *n_estimator* from 1 to 200.

b. *max_features* – This parameter controls the number of features to be considered while splitting the tree in random forest. We vary the value of *max_features* from 1 to 28.

c. *max_depth* – This indicates the depth of each tree in the classifier. Higher number means it has greater number of splits, and it can capture more information from the data. We vary the value of *max_depth* from 1 to 32.

After conducting a grid search, the random forest classifier gives an accuracy of 100%. The number of false positives and false negatives also remain the same. Hence, there is no change in detection rate.

SVM

Table 4 shows that SVM does not perform best in terms of cross validation accuracy. Again, smote does well in reducing the number of false negatives. Table 8 and Figure 9 show the classification results for the SVM classifier.

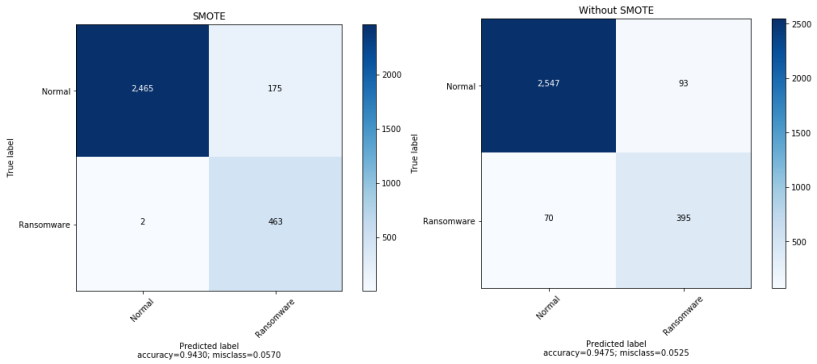


Figure 9– Comparing confusion matrix for the SVM classifier (with and without SMOTE)

Classification Report				
Metrics	**SMOTE**		**Without SMOTE**	
	Normal	Ransomware	Normal	Ransomware
Precision (%)	97	81	1.00	73
Recall (%)	96	85	93	1.00
f1 - score (%)	97	83	97	84
Support	2640	465	2640	465

Table 8 – Classification report for the SVM classifier

The parameters considered for hyper-parameter tuning are:

a. *C* – Often when the machine learning model is trained, it misclassifies the target variable. This parameter determines the area of hyperplane created by

misclassified training samples around the hyperplane. Large values of C means selecting a smaller-margin hyperplane and smaller values of C means selecting larger-margin hyperplane. Large values of C help classify the training samples correctly whereas smaller values of C create more chances of misclassification as the margin of hyperplane is high. We vary the value of C from 1 to 100.

The SVM classifier achieves an accuracy of 96% with value of '*C=15*'. Also, it can be seen from the confusion matrix in Figure 10 that with hyper-parameter tuning SVM improves the detection rate slightly.

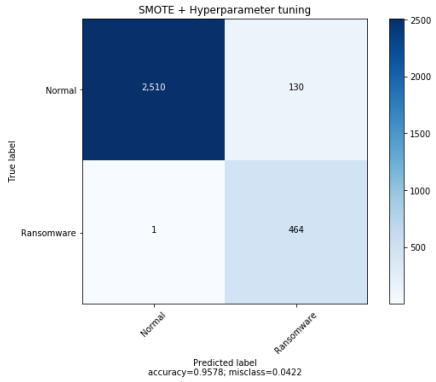


Figure 10 – Confusion matrix for the SVM classifier after applying hyper-parameter tuning

6 Conclusion

In the current paper, we proposed a new system for detecting ransomware in encrypted web traffic. Our approach consists of extracting meaningful information from network connections and certificates, and utilizing machine learning for detecting ransomware packets in network data. We explored a feature space in three broad domains of network characteristics: connection based, encryption based, and certificate based. We implemented and studied 3 different classifiers for our model, namely, logistic regression, SVM and random forest. Our experimental evaluation yields for random forest, the best performing classifier, a detection rate of 99.9% and a false positive rate of 0%. The obtained results are very encouraging and an indicator of the feasibility of effective ransomware detection from encrypted network traffic.

Although detecting ransomware in network traffic data is our main focus, identifying the family to which the ransomware belong would be beneficial. Our future work will explore how to extend our model to detect specific ransomware families, in addition to detecting individual ransomware samples. Our future work will also involve combining network-based features with the system-based features. This hybrid approach can form a very strong detection model for any future ransomware variants.

References

1. A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, “Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks,” in *Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148*, Springer-Verlag, 2015, pp. 3–24.
2. Cyren -“SSL Traffic Growth - Malware is Moving Heavily to HTTPS ".” [Online]. Available: https://www.cyren.com/blog/articles/over-one-third-of-malware-uses-https. [Accessed: 16-Jul-2019].
3. K. Cabaj, M. Gregorczyk, and W. Mazurczyk, “Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics,” *Comput. Electr. Eng.*, vol. 66, pp. 353–368, Feb. 2018.
4. A. O. Almashhadani, M. Kaiiali, S. Sezer, and P. O’Kane, “A Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware,” *IEEE Access*, vol. 7, pp. 47053–47067, 2019.
5. O. M. K. Alhawi, J. Baldwin, and A. Dehghantanha, “Leveraging Machine Learning Techniques for Windows Ransomware Network Traffic Detection,” Springer, Cham, 2018, pp. 93–106.
6. P. Prasse, L. Machlica, T. Pevn´ypevn´y, J. Havelka, and T. Scheffer, “Malware Detection by Analysing Encrypted Network Traffic with Neural Networks.” In: Ceci M., Hollmén J., Todorovski L., Vens C., Džeroski S. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2017. Lecture Notes in Computer Science, vol 10535. Springer, Cham
7. W. Niu, Z. Zhuo, X. Zhang, X. Du, G. Yang, and M. Guizani, “A Heuristic Statistical Testing Based Approach for Encrypted Network Traffic Identification,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3843–3853, Apr. 2019.
8. J. Zhang, Y. Xiang, W. Zhou, and Y. Wang, “Unsupervised traffic classification using flow statistical properties and IP packet payload,” *J. Comput. Syst. Sci.*, vol. 79, no. 5, pp. 573–585, Aug. 2013.
9. G. Stergiopoulos, A. Talavari, E. Bitsikas, and D. Gritzalis, “Automatic Detection of Various Malicious Traffic Using Side Channel Features on TCP Packets,” 2018, pp. 346–362.
10. Y.-L. Wan, J.-C. Chang, R.-J. Chen, and S.-J. Wang, “Feature-Selection-Based Ransomware Detection with Machine Learning of Data Analysis,” in *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*, 2018, pp. 85–88.
11. F. Strasák, “Detection of HTTPS Malware Traffic" Bachelor's Thesis, Czech Technical University in Prague, May 2017.
12. A. Kharraz, S. Arshad, C. Mulliner,W.K. Robertson and E. Kirda, Unveil: A large-scale, automated approach to detectingransomware, in: USENIX Security Symposium, 2016, pp. 757–772.
13. A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero and F. Maggi, ShieldFS: a self-healing, ransomware-aware filesystem, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, ACM, 2016, pp. 336–347.

14. D. Sgandurra, L. Muñoz-González, R. Mohsen and E.C. Lupu, Automated dynamic analysis of ransomware: Benefits, limitations and use for detection, arXiv preprint arXiv:1609.03020 (2016).
15. S. Salehi, H. Shahriari, M. M. Ahmadian, and L. Tazik, "A Novel Approach for Detecting DGA-based Ransomwares," in *2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology, ISCISC 2018*.
16. VirusTotal.: https://www.virustotal.com/gui/home/upload.
17. "Cuckoo Sandbox - Automated Malware Analysis." [Online]. Available: https://cuckoosandbox.org/. [Accessed: 19-Jul-2019].
18. "Normal Captures — Stratosphere IPS." [Online]. Available: https://www.stratosphereips.org/datasets-normal. [Accessed: 19-Jul-2019].
19. "Stratosphere IPS." [Online]. Available: https://www.stratosphereips.org/. [Accessed: 27-Jun-2019].
20. "Zeek User Manual v2.6.2." [Online]. Available: https://docs.zeek.org/en/stable/intro/index.html. [Accessed: 19-Jul-2019].
21. "Introduction to Cisco IOS NetFlow - A Technical Overview - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html. [Accessed: 18-Jul-2019].