MalDetect: A Structure of Encrypted Malware Traffic Detection

Article *in* Computers, Materials and Continua · January 2019

DOI: 10.32604/cmc.2019.05610

6 authors, including:

Jiyuan Liu
National University of Defense Technology
6 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Yingzhi Zeng
National University of Defense Technology
19 PUBLICATIONS **100** CITATIONS
SEE PROFILE

Yuexiang Yang
National University of Defense Technology
59 PUBLICATIONS **317** CITATIONS
SEE PROFILE

Computers, Materials & Continua CMC, vol.60, no.2, pp.721-739, 2019

MalDetect: A Structure of Encrypted Malware Traffic Detection

Jiyuan Liu[1], Yingzhi Zeng[2], Jiangyong Shi[2], Yuexiang Yang[2,*],

Rui Wang[3] and Liangzhong He[4]

Abstract: Recently, TLS protocol has been widely used to secure the application data carried in network traffic. It becomes more difficult for attackers to decipher messages through capturing the traffic generated from communications of hosts. On the other hand, malwares adopt TLS protocol when accessing to internet, which makes most malware traffic detection methods, such as DPI (Deep Packet Inspection), ineffective. Some literatures use statistical method with extracting the observable data fields exposed in TLS connections to train machine learning classifiers so as to infer whether a traffic flow is malware or not. However, most of them adopt the features based on the complete flow, such as flow duration, but seldom consider that the detection result should be given out as soon as possible. In this paper, we propose MalDetect, a structure of encrypted malware traffic detection. MalDetect only extracts features from approximately 8 packets (the number varies in different flows) at the beginning of traffic flows, which makes it capable of detecting malware traffic before the malware behaviors take practical impacts. In addition, observing that it is inefficient and time-consuming to re-train the offline classifier when new flow samples arrive, we deploy Online Random Forest in MalDetect. This enables the classifier to update its parameters in online mode and gets rid of the re-training process. MalDetect is coded in C++ language and open in Github. Furthermore, MalDetect is thoroughly evaluated from three aspects: effectiveness, timeliness and performance.

Keywords: Network intrusion detection, encrypted traffic, online learning.

1 Introduction

Traffic encryption is a practical way to protect the security and privacy of application data, including credit card details, passwords and sensitive personal information. It is claimed that HTTPS (SSL/TLS encrypted) traffic grew over 90% year over year. NSS Labs predicted 75% of traffic was going to be encrypted by 2019 [Austin (2016)]. At the same time,

[1] Student of College of Computer, National University of Defense Technology, Hunan, China.
[2] Faculty of College of Computer, National University of Defense Technology, Hunan, China.
[3] CEO of AppBugs Inc, USA.
[4] Faculty of China Mobile (Su Zhou) Software Technology Co., Ltd.
[*] Corresponding Author: Yuexiang Yang. Email: yyx@nudt.edu.cn.

HTTPS is found to be utilized in 37% malwares [Arna (2017)]. In our dataset, 23.35% of malwares use TLS protocol to secure their communication. Security products use TLS inspection to 'look inside' the secure tunnel so as to protect their users from treats which try to sneak past defenses. However, most businesses are not doing TLS inspection and leaving the door wide open for malwares [Arna (2017)]. TLS inspection firstly decrypts TLS traffic, then checks for threats, finally re-encrypts and sends it on its way. This MITM (Man-In-The-Middle) method results in lots of drawbacks. For instance, the TLS parameters used in end hosts are required to be stored in security products, which violates the privacy of legitimate traffic. Additionally, decryption and re-encryption processes are time-consuming, causing visible delays in data transmission.

Fortunately, some observable data fields, exposed in TLS connections, can be used to build a specific malware traffic classifier with statistical methods. Anderson et al. [Anderson, Paul and McGrew (2016)] thoroughly analyzed the building process of TLS connections and compared the preferences of TLS protocol parameters of legitimate traffic and malware traffic. They proved it is an efficient and effective way to use TLS protocol parameters to predict whether a traffic flow is generated by malware or not. In order to establish a TLS connection, several formatted flow packets will be transmitted between two hosts, such as *Client Hello* packet and *Server Hello* packet. They are not encrypted and contain observable data fields mentioned above. In our implementation, Libpcap [Jacobson and McCanne (2009)] is used to capture these packets.

It is worth to mention that the captured packets are distributed in the beginning part of traffic flows. Fig. 1 shows the building process of TLS connection between client and server. We can see that the capturing time ranges from the starting time of connection to an epoch before the transmission of application data. Additionally, the captured packets contain the ones of none-TLS handshake process, including *SYN* packet, *SYN ACK* packet, *ACK* packet and a few data packets in some occasions. They are not removed, because several traffic features, such as the duration of capturing, are designed with information in them. Technically, packet capturing is done right after the arriving time of *Change Cipher Spec* packet. In this period, no TLS encrypted application data is transferred. This guarantees that all features extracted as the input of machine learning classifiers can be gathered before malicious malware data is exchanged. We assign TLS-APP time to the consuming time between the last packet captured for feature extraction and the first packet of application data transferring process. In Section 5.3, we compare TLS-APP time, T_a in Fig. 1, with the time which MalDetect needs to handle with a traffic flow. The result shows that the former time is much longer than the latter one in magnitudes. As a result, MalDetect can give out the traffic flow detection result before the flow transfers malware data. That is to say, MalDetect is able to detect the malware traffic before it carries out illegal actions. This offers the opportunity to completely prevent malicious behaviors.

The features extracted from above packets can be separated into three classes. The first class is derived from classical flow features, including the duration of capturing, inbound/ outbound packet number, etc. The second class mainly contains TLS related features. For example, a set of cipher suites, which are the methods of encrypting application data, are offered in *Client Hello* packet for a server to choose. The occurrences of various cipher

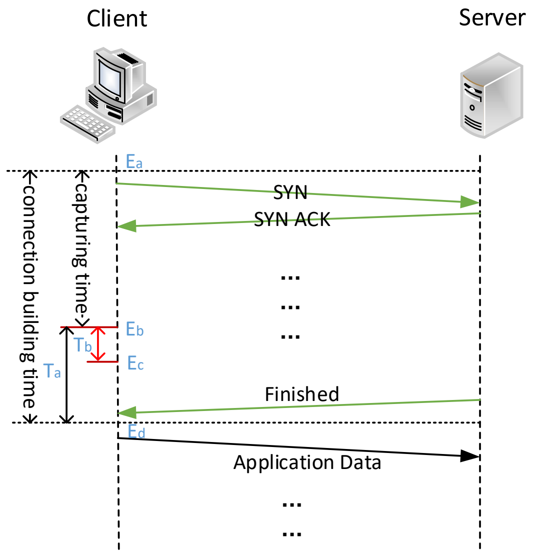

Figure 1: Time epochs and periods

suites are formatted as a feature vector. Additionally, the server chooses one cipher suite and responses the client with *Server Hello* packet. Then the cipher suite value appeared in *Server Hello* packet is considered as a feature. The field values of certificates transmitted in TLS handshake process are classified as an independent category. The technique of transforming certificate field values into features is similar to that of second class.

MalDetect adopts an Online Random Forest model to classify malware flows apart from legitimate flows. The model, proposed by Saffari et al. [Saffari, Leistner, Santner et al. (2009)], is constructed in online mode and updates its parameters when new samples arrive. This, comparing with adopting offline model, is beneficial when MalDetect handles with new threats, for there is no need to re-train and re-deploy the new model. It saves labor cost and improves quality of service.

MalDetect supports binary classification (*Legitimate* and *Malware Generated*) and multi-classification (*Legitimate* and various types of network flows generated by different malware families). We evaluated the FNR and FDR of MalDetect in the two modes using public datasets. The result shows MalDetect can effectively distinguish malware traffic flows. At the same time, the ability of learning new threats was also tested. In addition, we got the response time of MalDetect and thoroughly analyzed the benefits of a quick response. At last, we tested the throughput, for it is critical when deploying MalDetect into highspeed network devices.

The main contributions of this paper are summarized as following:

- We propose MalDetect, a structure of encrypted malware traffic detection. It is experimentally shown to have low FNR, FDR and high throughput. In addition, MalDetect can learn new malware traffic fast and effectively improve the detection rate of unknown threats if new samples are given.

- We only use approximately 8 packets at the beginning of traffic flows to extract features. This enables MalDetect to determine whether a flow is generated by malware before illegal behaviors are taken, which can better protect network users from malwares. This point is seldom considered in current intrusion detection methods. The Features are carefully selected and can be referred in the context of TLS traffic real-time classification.

- MalDetect is programed in C++ language and the source code is released in Github[3].

The rest of the paper is organized as following. Section 2 overviews several closely related works and points out MalDetect's advantages over them. Section 3 introduces the deployment scenarios in three categories. Section 2 thoroughly describes the structure of MalDetect. In Section 5, MalDetect is evaluated from three aspect: effectiveness, timeliness and performance. Section 6 states the conclusion.

2 Related work

Traditional network IDSs (Intrusion Detection System) detect if a traffic flow carries an attack with DPI method. But they are becoming useless when the traffic adopts encryption protocol. Erlacher et al. [Erlacher, Woertz and Dressler (2016)] built a TLS interception proxy with real-time Libpcap in order to provide DPI embedded IDS with decrypted application data. Sherry et al. [Sherry, Lan, Popa et al. (2015)] constructed a system called Blindbox. It can perform DPI directly on the encrypted traffic and simultaneously provide the functionality of middle-boxes. However, these packet inspection-based methods increase the probability of exposing TLS application data.

Malware traffic detection is an important field of traffic classification with restricting traffic classes into *Legitimate* and *Malware Generated*. Nguyen et al. [Nguyen and Armitage (2008)] conducted a survey of traffic classification techniques using machine learning. They thoroughly analyzed the limitations of *port-based* and packet inspection methods and highlighted the advantages of statistical methods. Statistical methods are more effective and efficient than the other two when applications adopt encryption techniques to secure their network data. Velan et al. [Velan, Čermák, Čeleda et al. (2015)] described several most widely used encryption protocols and then presented an overview of current approaches for the classification and analysis of encrypted traffic. They claimed that the initial stage of encryption protocols, including TLS, provides abundant information which can be used to classify different types of traffic.

Moore et al. [Moore, Zuev and Crogan (2013)] collected a large set of flow features which can be used in traffic classification. They provided reference for the selection of our features, although encrypted traffic was not taken into account. Kumano et al. [Kumano, Ata, Nakamura et al. (2014)] used the beginning packets of connections to calculated flow features. However, they only considered the classical flow features, such as average packet size, but ignored the details of encryption protocols. Chen et al. [Chen, Li, Tseng et al. (2017)] built a deep learning model named TSDNN to detect malware flows. However, we argue that the model building and detecting processes require a relatively long time, which

[3]https://github.com/IsaacLJY/MalDetect

makes TSDNN hard to deploy in high speed network. Cheng et al. [Cheng, Xu, Tang et al. (2018)] proposed an abnormal network flow feature sequence prediction approach, which focus on the DDoS attack detection. This deviates from our aim to build a structure for general malware traffic detection.

Anderson et al. [Anderson, Paul and McGrew (2016)] proposed a method to detect encrypted malware traffic without decryption. They reviewed the building process of SSL/TLS connections in detail and extracted flow features from data fields exposed in the packets of this process. Then the features were quantified into binary vectors which were used to train a L1 Logistic Regression model. In their further research, an open source project named Joy was released [McGrew and Anderson (2016)]. Joy adopted a large collection of flow features, including TLS related data, byte distribution, sequence of packet length and time, etc. At the same time, they noticed two problems, inaccurate ground truth and non-stationarity, and solved them in Anderson et al. [Anderson and McGrew (2017)]. Additionally, various machine learning classifiers were compared in Anderson et al.[Anderson and McGrew (2017)]. Random Forest model was considered as the most robust classifier in the domain of encrypted malware traffic detection. Compared with Anderson et al. [Anderson, Paul and McGrew (2016); McGrew and Anderson (2016); Anderson and McGrew (2017)], we redesign features extracted from flow packets. Instead of using packets of the complete flows, we only extracted features from packets before the end of TLS handshake process. This enables MalDetect to detect malware flows before illegal behaviors are taken. Additionally, MalDetect utilizes Online Random Forest [Saffari, Leistner, Santner et al. (2009)] model, which trains the classifier in online mode and avoids re-training and re-deploying when new samples arrive. Anderson et al. [Anderson and McGrew (2016)] also integrated contextual flow data, i.e. HTTP and DNS information, into their features. We give up doing so for the extraction and quantification processes are RAM-consuming and time-consuming.

3 Deployment scenarios

Before describing the structure of MalDetect, we illustrate three deployment scenarios. But the use of MalDetect is not restricted to the three.

3.1 Malware detection helper

MalDetect is able to help the detection of malwares in personal computers. Most of malwares, such as Trojans, Adware, Botnets, etc. stay in contact with their master hosts or communicate with other infected machines and there is a tendency that they are more likely to adopt TLS protocol in their communications. MalDetect is capable of distinguishing whether a TLS encrypted flow is generated by these malwares or not. That is to say, MalDetect can provide the network ports which the malware is currently using. This way, the infected machine can locate the malware more easily. For example, Alice deployed MalDetect in her Windows computer and noticed the warning-the flow, [170.20.10.3, 12115, 202.108.23.152, 443, TLS], was detected as a malware flow. Then she opened the Task Manager to look up which process was using port 12115. She scanned the source executable file of this process with antivirus application and finally found the file was a malware. In a smarter way, we do not need to check the malware manually, but utilize MalDetect into antivirus applications. Almost antivirus applications, such as Avast, McAfee etc. provide the functionality of real-time malware detection. They can directly use the results of MalDetect and take the operations Alice did in the example.

3.2 Malicious domain name detection helper

MalDetect can help the detection of malicious domain name. Some malwares, especially botnets, use a large amount of domain names in their illegal behaviors. Most of current methods use DNS traffic to detect malicious domains by looking at spatial characteristics. MalDetect exploits information in malware communication traffic rather than DNS traffic. If MalDetect discovered a malware flow, we can look up the domain name corresponding to the IP addresses in the flow and record it in a list. The domain names inside are much more likely to be malicious, which can provide additional information to help current techniques improve detection rate.

3.3 Intrusion prevention in SDN (Software Defined Network)

SDN enables programmatically network configuration so as to improve network performance and monitoring [Benzekki, El Fergougui and Elbelrhiti Elalaoui (2016)]. It disassociates the data plane (forwarding network packets) and the control plane (routing process). One functionality we concern here is SDN controller of control plane can enable and terminate the communication of two hosts. When deploying in SDN environment, MalDetect is able to report its detection results to SDN controller. This way, SDN controller can cut off the malware flow, which benefits a lot especially when SDN is in a DDoS attack.

4 Structure design

The structure of MalDetect is shown in Fig. 2. MalDetect does not act as a middle box which blocks the traffic flow. Instead, it is constructed beside the traffic road and keeps an eye on the packets. When a packet arrives, it duplicates the packet content from network card for further steps. This enables MalDetect to detect malware traffic without causing extra delay in the communication between client and server. Libpcap, a C++ library, is invoked in MalDetect to capture the network traffic constantly. After capturing the network packets, MalDetect handles them through two relatively separated modules, i.e., Feature Extraction and Training & Detecting. Feature Extraction requires the captured packets from Libpcap as input. Then it selects some informative byte fields to represent the target flow. Finally, the values of these fields are quantified into numeric vectors, the output of this module. The numeric vectors are sent to next module, Training & Detecting, so as to train a machine learning classifier or be detected by the well-trained classifier. The two modules are thoroughly introduced in the following.

4.1 Feature extraction

In this module, a set of pre-defined features, i.e. the values of byte fields in packet content, are extracted and the quantification of these features are performed in order to output the

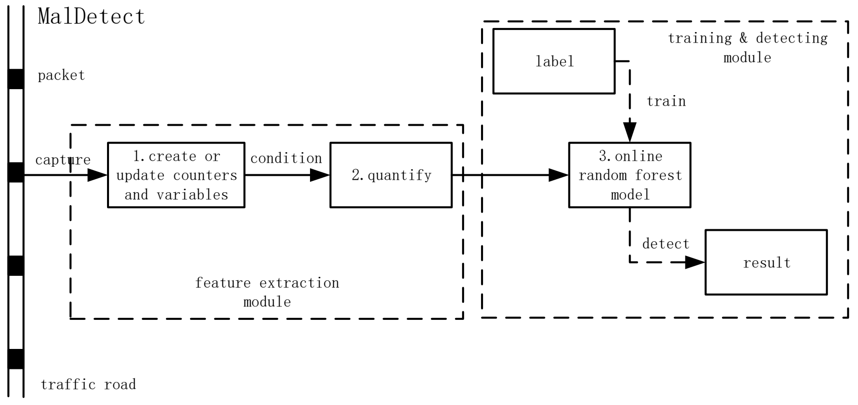
MalDetect
packet
capture
1.create or update counters and variables
condition
2.quantify
feature extraction module
label
train
training & detecting module
3.online random forest model
detect
result
traffic road

Figure 2: Structure design of MalDetect

formatted numeric vectors for next module.

4.1.1 Feature

The effectiveness and performance of MalDetect is closely related to the selection of flow features. Therefore, we analyzed the specifics of packet exchanged between two hosts in detail. Fig. 3 shows the typical building process of a TLS connection. We define client as the host who sends the first packet and the other one is server. From epoch 1 to epoch 2, client and server are trying to build a TCP connection through a three-way handshake. Next, client sends *Client Hello* packet in which there are a set of alternative field values, such as cipher suites and extensions, for server to choose. Server responses with the *Server Hello* packet to inform the chosen field values. Together with *Server Hello* packet, *Certificate* packet, which contains the server certificate assigned by authorities, and *Server Hello Done* packet, which indicates server finishes transferring it messages, are sent. Then, client tells server the key to encrypt content of further packets via *Client Key Exchange* packet. Additionally, *Change Cipher Spec* packet and *Finished* packet are exchanged between client and server. So far, the TLS connection has been built, and the two hosts begin to transfer *Application Data* packets, the encrypted data. MalDetect extracts the field values in the unencrypted packets mentioned above as features. It is worth to mention that all the packets used in MalDetect are transmitted before epoch 3 and no information which malwares want to exchange are transferred. This provides the basis of detecting malware traffic before malwares take out their illegal actions.

We firstly extracted a large range of features from network flows and tested them in our experiment setting. At last, 23 most robust features are selected and used in MalDetect. The features are collected from each flow independently and can be separated into three classes as following.

Packet feature
This class of features are not related with TLS protocol. They macroscopically describe flows from three dimensions: size, number and time. In this way, 7 features are listed.

Inbound Bytes: It refers to sum of the sizes of transmitted packets from server to client in capturing time.

Outbound Bytes: It is similar to Inbound Bytes but the considered packet direction is from client to server.

Inbound Packets: It counts the number of transmitted packets from server to client in capturing time.

Outbound Packets: It is similar to Inbound Packets but the considered packet direction is from client to server.

Duration: It describes the length of capturing time.

SPL: It refers to the sequence of packet length. For a specific flow packet, we discretized its length into 11 equally sized bins. The packet whose length is in range [0, 150) goes into the first bin, the packet whose length is in range [150, 300) goes into the second bin, and so on. The packet goes into the last bin if its length is greater than 1500 bytes. For instance, the vector of bins is [0,0,0,0,0,0,0,0,0,0,0] at the initial stage. MalDetect captures a packet and its length is 200, then the vector of bins is updated into [0,1,0,0,0,0,0,0,0,0,0].

SPT: It refers to the sequence of packet time. When capturing a flow packet, MalDetect calculates the time interval since the last flow packet. Then the time interval is discretized into 11 equally sized bins, which is similar to the calculation of SPL. The size of the bins is set to 50 ms.

TLS protocol feature

The TLS protocol details, such as TLS version and cipher suite, are discussed between client and server in the initial stage of TLS connections. They are transferred in several formatted flow packets without encryption. We mainly analyzed the byte fields of *Client Hello* packet and *Server Hello* packet. At last, 8 features are selected and listed below.

TLS Version: Client designates the TLS protocol version used in further packet exchange. In MalDetect, four most widely used TLS versions, i.e. SSL 3.0, TLS 1.0, TLS 1.1 and TLS 1.2, are concerned.

Offered Cipher Suites: Cipher suite is a set of algorithms, including a key exchange algorithm, a bulk encryption algorithm and a message authentication code algorithm, and helps secure a network connection. At the beginning of discussion of TLS protocol details, client sends *Client Hello* packet. This packet contains a set of cipher suites, which indicates client supports these cipher suites in further information exchange. Therefore, the byte values of these cipher suites are extracted as a feature.

Selected Cipher Suite: When receiving the *Client Hello* packet, server always responses with *Server Hello* packet in which a cipher suite is specified. This cipher suite should be one of Offered Cipher Suites. Server and client will use it in further packet transmission.
Offered Compression Methods: *Client Hello* packet provides a list of lossless compression methods to compress the application data. The field values are extracted as a feature.

Selected Compression Method: In *Server Hello* packet, a compression method in Offered Compression Methods is selected for further packet transmission.

Offered Extensions: There are various TLS extensions which provides additional information about the TLS connection. Extension type is always at the beginning of extension bytes, while extension details are closely followed. For example, *Server Name*, one of the most common extension, specifies the name of server which client is attempting to connect. The first two bytes is 0x0000, indicating the following bytes are *Server Name* fields. Additionally, the bytes are well formatted and can be easily extracted. In MalDetect, we only consider the appearance of extension types.

Selected Extensions: Server responses to several extensions mentioned in *Client Hello* packet in some occasions. MalDetect also analyzes the corresponding fields in *Server Hello* packet and records the appearance of extension types.

TLS Packet Ratio: There are some packets carrying no TLS protocol information in capturing time, such as the packets for TCP connection building and ACK packets. The other packets, such as *Client Hello* packet, are closely related to TLS protocol. MalDetect calculates the ratio of the later packets.

Certificate feature
Typically, certificates are transferred in the building process of TLS connection. They are formatted in X.509 and consist of abundant information which is able to help client identify the server. 8 features are extracted in *Certificate* packet.

Certificate Number: Usually, multiple certificates are contained in *Certificate* packet.

Bad Certificate Number: Certificate is assigned by trusted authorities and critical to identify the server. The badly formatted certificate is probably self-signed. So, the bad certificate number is extracted as a feature in MalDetect.

Certificate Version Ratio: In a signed certificate, the version is firstly declared. Because most of flows have more than one certificates, the ratio of certificate version is calculated.

Certificate Extension Ratio: There are a list of extensions in certificate. Each extension has its own ID, expressed as Object Identification, together with either a critical and non-critical indication. MalDetect extracts the byte values of extension IDs as features.

Certificate Validity Mean: In a signed certificate, 13 bytes indicate a UTC time before which the certificate is not valid. We call it *notBefore* time. Following *notBefore* time, there is a *notAfter* time, meaning the certificate is not valid after the time epoch. Therefore, MalDetect calculates the difference of *notBefore* time and *notAfter* time as certificate validity. The mean of validities of certificates are computed as a feature.

Certificate Public Key Length Mean: A public key is used together with the matching private key to prove the identity of the peer (authentication). The mean of public key lengths is calculated.

Certificate Public Key Algorithm Ratio: It refers to algorithm to generate the public key inside the certificate. The bytes value is extracted.

Certificate Signature Algorithm Ratio: It refers to the signature of the certificate created by the issuer. This signature proves that the claimed issuer of the certificate is the real issuer since the signature can be verified by using the public key from the issuer's certificate.

4.1.2 Technique

Network flows are processed separately in MalDetect. A flow is uniquely identified by five elements: source IP, source port, destination IP, destination port and protocol. A filter of TLS network traffic is applied when MalDetect captures packets via Libpcap. Therefore, the vector, [source IP, source port, destination IP, destination port], is used to represent a flow. MalDetect records this vector in an array and allocates independent RAM space, which consists of multiple counters and variables, corresponding to each flow. The moment MalDetect captures the first packet of a new flow, counters and variables are initialized. When new packets arrive, it updates them according to the extracted byte field values. The end time of a specific flow capturing should be carefully selected, for there are two common TLS connection building processes actually.

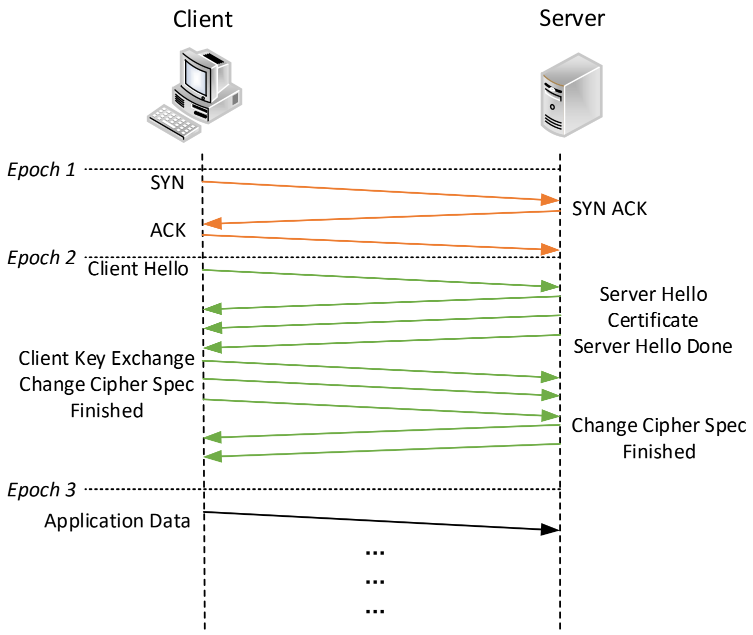

Figure 3: Typical building process of TLS connection

Fig. 3 shows a typical flow packet exchange process of TLS connection. Before epoch 3, *Client Hello* packet, *Server Hello* packet and *Certificate* packet, together with TCP handshake packets, are used to update the counters and variables kept in RAM space. MalDetect does not finish capturing packet when *Certificate* packet arrives, because of the existence of the other type of TLS connection building process which requires no *Certificate* packet. Once client builds a TLS connection with server successfully, they store the symmetric key, which is used to encrypt application data, in their caches. Client and server perform session resumption without transferring certificates, when they want to exchange data again. Therefore, MalDetect finishes capturing after receiving *Change Cipher Spec* packet which shows up in two types of TLS connection.

There are some differences between the updating of counters and variables of three parts of features introduced in Section 4.1.1. For Flow features, all captured packets are helpful if the flow is currently recorded in MalDetect. However, the information of TLS protocol features is only included in *Client Hello* packet and *Server Hello* packet. At the same time, all *Certificate Features* can be inferred from *Certificate* packet. Therefore, the packet type is distinguished and non-related ones, such as *Server Hello Done* packet, are ignored in the extraction of later two parts of features so as to speed up MalDetect.

Most of features collected in the beginning are field values, and they are not able to be inputted into Online Random Forest model. Therefore, quantification is performed the moment MalDetect finishes capturing required packets of a flow. For counters, we use their direct values. For example, Inbound Bytes in Packet features counts sizes of captured packets. Its value is numeric and directly used. For variables of a flow, we use the appearance of field values. For instance, four TLS versions, TLS 1.0, TLS 1.1, TLS 1.2 and TLS 1.3, are concerned in MalDetect, so a vector with four 0 or 1 are used to represents the appearance of TLS versions. If MalDetect observes a flow adopts TLS 1.2, it quantifies TLS Version feature into [0, 0, 1, 0]. Because a flow may contain multiple certificate, the numeric vectors of Certificate features are the average values of vectors generated from single certificate. If a flow contains tree certificates, A, B and C. The Certificate Version vectors of them are [0, 1, 0], [1, 0, 0] and [0, 1, 0] respectively. Then the final vector for Certificate Version is [0.33333, 0.66666, 0] ([(0+1+0)/3, (1+0+1)/3, (0+0+0)/3]).

The implementation of feature extraction module is presented in Algorithm 1.

4.2 Training and detecting

The input of Training & Detecting module is the numeric vector generated in Feature Extraction module. This module consists of two modes - training and detecting. In training mode, labels are required. How to label the flows captured from network card is critical. Since the training dataset is known in advance, a special mark can be made in the flows to distinguish the flow types before training. In *Client Hello* packet, there is a field called *Random Bytes* generated by client randomly. We use the last byte of this field to indicate the type of a flow and set the rest bytes to 0. *Random Bytes* occupies 28 bytes, which means it is in a separately low probability, that the assigned value conflicts with previous value.

The value of *Random Bytes* is set according to the types of flow in dataset pre-processing. We also developed a tool named PcapEditor to help label the network flows. The required input of this tool is malware traffic flows (currently support *.pcap* format) and corresponding label. The number of labels is not restricted into two types (*Legitimate* and *Malware Generated*). PcapEditor maintains a label list which is pre-defined in configuration file. The configuration file is editable, and users can write their own labels in it. But it should be noticed that sequence and number of these labels must be same as the ones in MalDetect, because MalDetect uses the label list to distinguish flow types. The source code of PcapEditor is also open in Github[4].

In detecting mode, MalDetect captures the traffic from assigned network card. When extracting the value of *Random Bytes*, it is going to find out the value is randomly generated

[4]The link of the source code is https://github.com/IsaacLJY/PcapEditor

Algorithm 1 Feature extraction module

```
Input: Packet bytes
Output: Quantified vector
 1: filter out non-TLS traffic
 2: get flow ID (src.ip, src.port, dest.ip, dest.port)
 3: if flow ID not in list then
 4:     if the packet is SYN then
 5:         create counters and variables of this flow
 6:     end if
 7: else
 8:     find the counters and variables of this flow
 9: end if
10: update packet features
11: if the packet contains Client Hello or Server Hello content then
12:     update TLS protocol features
13: else if the packet contains Certificate content then
14:     update certificate features
15: else if the packet contains Change Cipher Spec content then
16:     quantify features of this flow
17:     remove the counters and variables of this flow
18:     return quantified vector
19: else if the packet contains Application Data content then
20:     remove counters and variables of this flow
21: end if
```

by client and performs detecting the types of flow, rather than training the classifier. In order to show detecting results, MalDetect prints the flow ID, [source IP, source port, destination IP, destination port, protocol], and the flow type prediction. Actually, MalDetect can be integrated into threats warning system or threats handling system, such as IDS, instead of operating as an independent tool, which is going to be a part of our future work.

In this module, MalDetect adopts Online Random Forest model as the classifier. Usually, random forest is trained in offline mode, which requires the entire training data in advance. However, training data is generated continuously in practice. Especially for malware traffic detection, new types of network traffic are emerging continuously. Online Random Forest keeps the previous knowledge about threats but also learns new threats incrementally. Besides, this model combines online bagging and extremely randomized forests, and has comparable performance with common Random Forest model, claimed by Saffari et al. [Saffari, Leistner, Santner et al. (2009)]. Meanwhile, it builds and tests each tree independently, hence the training and testing is able to be performed in parallel. This accelerates MalDetect's training procedure and shortens the response time of detecting. Additionally, MalDetect is designed to support multiple flow labels. For common binary classification, binary decomposition, e.g., one-vs-all, is deployed in order to solve multi-class classification. But such decomposition leads to higher computational burden for the building of several binary classifiers, and very unbalanced data distribution where the majority of the samples are from negative class. The model is naturally built in multi-class mode without the decomposition mentioned above and satisfies the need of our problem maximum.

5 Evaluation

When assessing MalDetect, we aim to answer three questions. First, can MalDetect detect malware traffic flows precisely? Second, is MalDetect able to detect malware traffic before illegal actions are taken? Third, what are the performance overheads of MalDetect? Therefore, we measured MalDetect in three aspects: effectiveness, timeliness and performance. Additionally, the dataset used in evaluation is introduced at the beginning.

5.1 Dataset

Most researchers use their own private traffic dataset in evaluation. However, the credibility of results is closely related to the dataset. So, we use two public datasets in our research, i.e., CTU-13 dataset and MCFP dataset.

- CTU-13 dataset [Garcia, Grill, Stiborek et al. (2014)]-This dataset contains thirteen scenarios and was captured in the CTU University, Czech Republic, in 2011. On each scenario, a specific malware which used several protocols and performed different actions, was executed. It consists of three types of traffic i.e., malware traffic, legitimate traffic and background traffic. They are clearly labeled at the level of flow. In our research, background traffic was ignored.

- MCFP dataset [Erquiaga, García and García Garino (2017)]-This dataset has been collected by a group of researchers in CTU University since January, 2017. The malware traffic is generated by executing the malware for long terms, up to three weeks or even months, and it keeps a variety of captures from different types of malwares, such as Trojans, Adware, Botnets, etc. Currently, over two hundred of scenarios are captured and open to the public.

In our datasets, 76.65% malwares do not adopt TLS protocol. Therefore, we only use traffic of the rest 23.35% malwares. Then we submit the malwares into VirusTotal [Total (2012)] to get the detection results of various security vendors. The results of Avast company are used as malware traffic labels in our evaluation. In addition, we search *Random Bytes* fields of flows for MalDetect training mode and set their values according to the types of malwares. Finally, we get 319152 malware flows and 63998 legitimate flows. Tab. 1 shows the traffic flow information of different types.

5.2 Effectiveness

Six types of network traffic, as shown in Tab. 1, are concerned in our test. At the beginning, we ignored the different types of malware flows and treated all of them as one type. The

Table 1: Malware flow information

Type	Label value	Label indicator	Flow number
Adw	0	$0x\{00\}^{27}00$	63998
Drp	1	$0x\{00\}^{27}01$	29757
Rtk	2	$0x\{00\}^{27}02$	112815
Susp	3	$0x\{00\}^{27}03$	30592
Trj	4	$0x\{00\}^{27}04$	35118
Legitimate	5	$0x\{00\}^{27}05$	110870

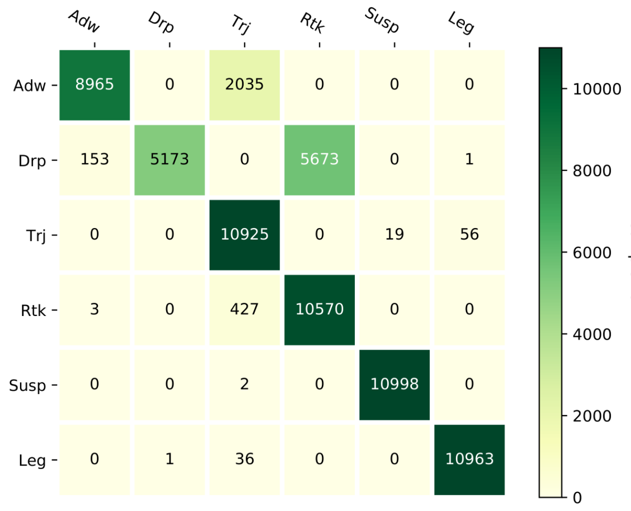

Figure 4: Confusion matrix

binary classification was performed by setting the labels to *Legitimate* and *Malware Generated*. We selected 10^5 flows, 10^4 for five malware types each and 5×10^4 for *Legitimate*, as training set, while 2×10^4 flows were randomly selected as test set with the same proportion. Then we used Tcpreplay [Turner (2011)] to replay the traffic flows and deployed MalDetect in the assigned network card to capture the flow packets and start to train and detect. The detecting result shows the FNR was 0.8%, which means 8 out of 1000 flows generated were considered as legitimate flows. At the same time, the FDR was 0.09%, meaning only 9 out of 10000 flows, which MalDetect reported as *Malware Generated*, were legitimate.

Question: *Do the FNR and FDR mentioned above satisfy the requirement of malware traffic detection?*

Some viewers may argue that it is not good enough misclassifying 8 out of 1000 malware traffic flows. But it is fairly important to know that FNR presented is achieved in balanced samples with about 50% malware flows. In real network environment, the ratio of malware traffic is completely low. Additionally, almost all infected machines have more than one flow, which makes detecting infected machine easier.

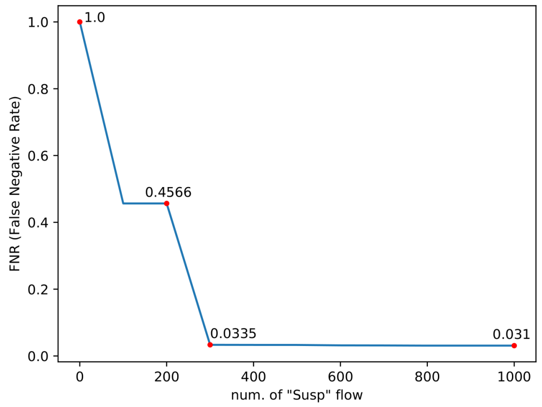

Figure 5: New threat training and detecting

MalDetect is able to give out the malware types. This requires MalDetect to be trained by flows which are clearly labeled with malware types. Tab. 1 presents the information of flows used in our test. But the number of each type of traffic flows are unbalanced. Therefore, we duplicated the less and made the numbers of all types almost equal. Then we randomly selected 133567 flows as training set, while 66000 flows were chosen as test set. Fig. 4 shows the confusion matrix. FNRs of *Susp*, *Leg* (Legitimate), *Trj* and *Rtk* are relatively low, accounting for 0.02%, 0.34%, 0.68% and 3.90% respectively, while *Adw* and *Drp* have high FNRs, 18.5% and 52.97%. This results from that a large number of flow samples generated by *Adw* and *Drp* are misclassified into *Trj* and *Rtk*. This does not mean MalDetect cannot detect the two types of malware traffic. We should notice the labels of network flows come from Avast company and may be inaccurate. This problem is called inaccurate ground truth, which is usually met in supervised machine learning task, and we will discuss it in further work. Additionally, there is no need to get the accurate malware type, for the type only provides a vague guide for further threat handling. After all, we know there is a malware traffic and can trace it to find the malware. In most cases, the malware type does not take a critical position.

Another important aspect of MalDetect is the ability to learn new treats. In this test case, we left out the traffic flows generated by *Susp* malware and used the rest to train MalDetect. Then MalDetect was trained incrementally with *Susp* malware traffic flows. we calculated the FDR when every 100 flows were input. Fig. 5 shows the FNR change of MalDetect. It can be seen that the FNR drop from 100% to 3.35% dramatically with the number of *Susp* flows ranging from 0 to 300. When 1000 *Susp* flows were input, the FNR slightly decreased to 3.1%. This illustrates that MalDetect can effectively detect new threats when training with new types of malware flows in online mode.

5.3 Timeliness

Fig. 1 shows several critical time epochs and periods of a flow in the process of detecting. E_a is the epoch when detecting starts. In the following is a period named capturing time in which MalDetect collects required information of this flow. The moment, E_b, information

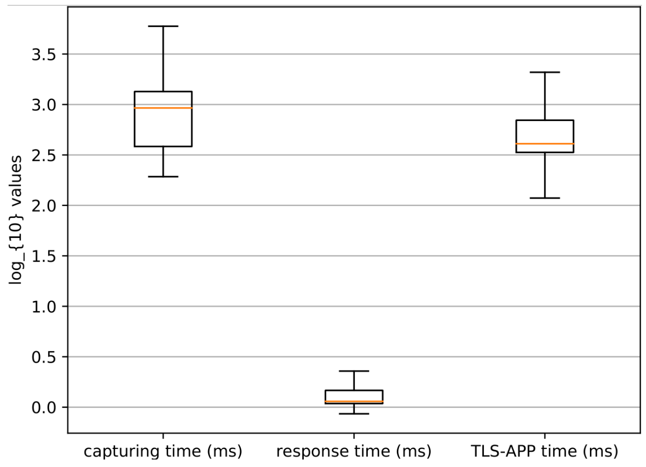

Figure 6: Time lengths comparison

collecting finishes, MalDetect performs classification with well-trained model. It is going to consume time T_b, also called MalDetect response time. Client and host start to exchange data only after TLS connection is built. Connection building time in the figure, the period from E_a to E_d, represents the time length MalDetect takes to build this flow. We assign TLS-APP time to the period from E_b to E_d.

In this test case, we aim to compare Ec with Ed. If $E_c < E_d$, MalDetect gives out detecting result before the flow transfers data packets. This helps a lot when the network flow is generated by malwares, for we can cut off the exchange of illegal information totally. This problem can be transformed into the length comparison of TLS-APP time and response time. We used 133567 flows mentioned in Subsection 5.1 to train MalDetect. Then we performed detecting with 105 flows and calculated the capturing time, response time and TLS-APP time, shown in Fig. 6. It can be seen that TLS-APP time ranges from 120 ms to 2500 ms, with most values distributed around 400 ms. At the same time, response time ranges from 0.7 ms to 2.5 ms and MalDetect requires about 1ms to classify for the majority of network flows. Therefore, there is 117.5 ms (120 ms-2.5 ms), left at least to take operations when a flow is distinguished as *Malware Generated*. Additionally, connection building time for most flows, the sum of capturing time median and TLS-APP time median, is approximately 1400 ms. Therefore, about 28.5% ((400 ms-1 ms)/1400 ms) of TLS connection building time can be used for further operations, such as sending warnings to hosts, terminating the traffic flow if MalDetect is deployed in SDN, etc.

5.4 Performance

It is not enough for MalDetect to detect malware traffic with low FNR and response quickly. MalDetect should be capable of handling with a large amount of flows at a time, for it may be deployed in highspeed network. In this case, we deployed MalDetect in a laptop operating Ubuntu 16.04.3. The laptop is embedded with Intel Core i7-6700HQ @ 2.56 GHz.

Table 2: MalDetect throughput

TLS traffic Ratio	Size throughput (Mbps)	Packet throughput (10^4 packet/s)	Flow throughput (10^3 flow/s)
0.0	18984.47	403.58	101.68
0.1	2736.09	57.82	7.75
0.2	1500.23	31.52	4.31
0.3	988.04	20.64	2.89
0.4	769.34	15.97	2.23
0.5	616.94	12.74	1.75
0.6	525.7	10.79	1.46
0.7	432.85	8.83	1.26
0.8	401.55	8.15	1.1
0.9	349.36	7.05	0.98
1.0	319.67	6.41	0.87

First, 133567 flows mentioned in Subsection 5.2 were used to train MalDetect. There are not only TLS encrypted flows in real network environment, so we tested the throughput of MalDetect using the traffic with various ratio of TLS flows. The throughput of MalDetect broke the limitation of our network card, so we temporarily revised the flow capturing method to using Libpcap offline mode in order to measure the boundary of MalDetect. This way, flows are inputted from previous stored PCAP files. As a consequence, we got the throughput in Tab. 2. It can be seen that the throughput of MalDetect varies a lot with different ratios of TLS traffic. According to a white paper of NSS Lab, 40%-50% of enterprise traffic is encrypted [Lab (2018)]. In this setting, MalDetect are expected to have a throughput of about 700 Mbps. However, the through is highly affected by machine hardware. Therefore, we suggest to evaluate MalDetect's throughput before deploying it into network.

6 Conclusion

With the widely use of TLS protocol, an increasing number of malwares adopts TLS encryption method to secure their network traffic. This makes most DPI techniques ineffective. In order to tackle this problem, we promote MalDetect, a structure of encrypted malware traffic detection. It improves the current statistical methods in three aspects. First, 23 robust and easily extracted features are selected. Second, MalDetect is capable of distinguish a malware flow before malwares begin exchanging information. This helps a lot, for we can cut off the communications between malwares totally, which is advanced than the other techniques. Third, MalDetect adopts Online Random Forest as its classifier. This gets rid of the re-training and re-deploying procedures. As a result, it improves the quality of service and saves the labor cost. In our experiments, MalDetect was shown effective, timely and of high throughput to distinguish malware flows with low FNR. We opened the source code in Github. Finally, we suggest that using MalDetect independently cannot make the most of it. It is better to utilize it into threat handling system, such as IDS.

Acknowledgement: We would like to thank the developers of VirusTotal, Tcpreplay and Libpcap Library. Without them, we cannot accomplish our work. This research is supported by Education Ministry-China Mobile Research Funding under Grant No. MCM20170404.

References

Anderson, B.; McGrew, D. (2016): Identifying encrypted malware traffic with contextual flow data. *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pp. 35-46.

Anderson, B.; McGrew, D. (2017): Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1723-1732.

Anderson, B.; Paul, S.; McGrew, D. (2016): Deciphering malware's use of tls (without decryption). *Journal of Computer Virology and Hacking Techniques*, pp. 1-17.

Arna, M. (2017): Malware is moving heavily to https. https://www.cyren.com/blog/articles/over-one-third-of-malware-uses-https.

Austin, T. (2016): Nss labs predicts 75% of web traffic will be encrypted by 2019. https://www.nsslabs.com/company/news/press-releases/nss-labs-predicts-75-of-web-traffic-will-be-encrypted-by-2019/.

Benzekki, K.; El Fergougui, A.; Elbelrhiti Elalaoui, A. (2016): Software-defined networking (sdn): a survey. *Security and Communication Networks*, vol. 9, no. 18, pp. 5803-5833.

Chen, Y. C.; Li, Y. J.; Tseng, A.; Lin, T. (2017): Deep learning for malicious flow detection. *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications*, pp. 1-7.

Cheng, R.; Xu, R.; Tang, X.; Sheng, V. S.; Cai, C. (2018): An abnormal network flow feature sequence prediction approach for ddos attacks detection in big data environment. *Computers, Materials & Continua*, vol. 55, no. 1, pp. 95.

Erlacher, F.; Woertz, S.; Dressler, F. (2016): A tls interception proxy with real-time libpcap export. *41st IEEE Conference on Local Computer Networks, Demo Session*.

Erquiaga, M. J.; García, S.; García Garino, C. (2017): Observer effect: How intercepting https traffic forces malware to change their behavior. *XXIII Congreso Argentino de Ciencias de la Computación*.

Garcia, S.; Grill, M.; Stiborek, J.; Zunino, A. (2014): An empirical comparison of botnet detection methods. *Computers & Security*, vol. 45, pp. 100-123.

Jacobson, V.; McCanne, S. (2009): libpcap: packet capture library. *Lawrence Berkeley Laboratory, Berkeley, CA*.

Kumano, Y.; Ata, S.; Nakamura, N.; Nakahira, Y.; Oka, I. (2014): Towards real-time processing for application identification o f e ncrypted t raffic. *In ternational Co nference on Computing, Networking and Communications*, pp. 136-140.

Lab, N. (2018): Ssl: Enterprise's new attack frontier. https://www.nsslabs.com/resources/white-papers/ssl-enterprise-s-new-attack-frontier/.

McGrew, D.; Anderson, B. (2016): Enhanced telemetry for encrypted threat analytics. *IEEE 24th International Conference on Network Protocols*, pp. 1-6.

Moore, A.; Zuev, D.; Crogan, M. (2013): Discriminators for use in flow-based classification. *Technical Report*.

Nguyen, T. T.; Armitage, G. (2008): A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56-76.

Saffari, A.; Leistner, C.; Santner, J.; Godec, M.; Bischof, H. (2009): On-line random forests. *IEEE 12th International Conference on Computer Vision Workshops*, pp. 1393-1400.

Sherry, J.; Lan, C.; Popa, R. A.; Ratnasamy, S. (2015): Blindbox: Deep packet inspection over encrypted traffic. *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 213-226.

Total, V. (2012): Virustotal-free online virus, malware and url scanner. https://www. virustotal. com/en.

Velan, P.; Čermák, M.; Čeleda, P.; Drašar, M. (2015): A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, vol. 25, no. 5, pp. 355-374.