Detecting Android Malware Leveraging Text Semantics of Network Flows

Shanshan Wang, Qiben Yan, *Member, IEEE*, Zhenxiang Chen, Bo Yang, Chuan Zhao, and Mauro Conti, *Senior Member, IEEE*

***Abstract*— The emergence of malicious apps poses a serious threat to the Android platform. Most types of mobile malware rely on network interface to coordinate operations, steal users' private information, and launch attack activities. In this paper, we propose an effective and automatic malware detection method using the text semantics of network traffic. In particular, we consider each HTTP flow generated by mobile apps as a text document, which can be processed by natural language processing to extract text-level features. Then, we use the text semantic features of network traffic to develop an effective malware detection model. In an evaluation using 31 706 benign flows and 5258 malicious flows, our method outperforms the existing approaches, and gets an accuracy of 99.15%. We also conduct experiments to verify that the method is effective in detecting newly discovered malware, and requires only a few samples to achieve a good detection result. When the detection model is applied to the real environment to detect unknown applications in the wild, the experimental results show that our method performs significantly better than other popular anti-virus scanners with a detection rate of 54.81%. Our method also reveals certain malware types that can avoid the detection of anti-virus scanners. In addition, we design a detection system on encrypted traffic for bring-your-own-device enterprise network, home network, and 3G/4G mobile network. The detection model is integrated into the system to discover suspicious network behaviors.**

***Index Terms*— Malware detection, HTTP flow analysis, text semantics, machine learning.**

I. Introduction

THE proliferation of mobile devices has opened the mobile era and rapidly increased the popularity of mobile apps. The explosive growth of mobile communication brings substantial burden to the mobile security management. A recent report [1] shows that the number of apps in the Google Play Store has risen from 16 thousand in December 2009 to more than 2 million in February 2016. Notably, the mobile traffic amount has reached 3.7 exabytes per month [1] in 2015. However, the uprising of mobile application is highly impaired by the prevalent malware. Among different operating systems, Android becomes the most popular platform due to its open architecture [2]. Unfortunately, smartphones running on Android system have gradually become the main target of attackers and are infected by malicious apps. This circumstance reveals the urgency of enforcing mobile app security.

A recent survey [3] studies a wide variety of malware's malicious behaviors, and classifies the existing mobile malware detection methods into two categories, including static analysis and dynamic analysis methods. Numerous previous works employ static analysis to detect privacy leakage [4], malware [5], and vulnerabilities [6] in Android apps. However, static analysis is challenged by the code polymorphism and code obfuscation of malware [7], which are used to generate variants of malware to evade detections. Dynamic analysis methods modify the device OS to track and access sensitive information at runtime [8]. Dynamic analysis is promising [9] but requires a sufficiently large set of executions to cover apps' behaviors. Thus, performing dynamic analysis on resource-constrained smart devices is challenging.

To better counteract mobile malware, researchers start exploring new solutions for mobile malware detection based on network traffic. Using network traffic to discover hidden malware is promising, as malware usually launch malicious behaviors through network connections. In this study, we focus on comprehensively analyzing mobile network traffic to identity mobile malware. We examine the traffic flow header using N-gram method from the natural language processing (NLP). Then, we propose an automatic feature selection algorithm based on chi-square test to identify meaningful features. These automatically selected features are used to build an SVM classifier for malware detection. We have verified the effectiveness of this method with extensive experiments. In addition, the detection model is effectively applied in our detection system to discover malware at different types of network environments. The main contributions of this study are summarized as follows:

- We propose a novel solution to perform malware detection using NLP methods by treating mobile traffic as documents. We apply an automatic feature selection

Manuscript received May 5, 2017; revised August 10, 2017 and October 16, 2017; accepted October 16, 2017. Date of publication November 8, 2017; date of current version January 29, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61672262, Grant 61573166, Grant 61472164, Grant 61572230, Grant 61702218, and Grant 61702216, in part by the Natural Science Foundation of Shandong Province under Grant ZR2014JL042 and Grant ZR2012FM010, in part by the Shandong Provincial Key R&D Program under Grant 2016GGX101001, in part by the CERNET Next Generation Internet Technology Innovation Project under Grant NGII20160404, and in part by NSF under Grant CNS-1566388. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Vrizlynn L. L. Thing. *(Corresponding author: Zhenxiang Chen.)*

S. Wang, Z. Chen, B. Yang, and C. Zhao are with the Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250000, China (e-mail: czx@ujn.edu.cn).

Q. Yan is with the Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, NE 68588 USA.

M. Conti is with the Department of Mathematics, University of Padova, 35122 Padua, Italy.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.2771228

algorithm based on N-gram sequence to obtain meaningful features from the semantics of traffic flows.
- Extensive experimental results based on real world network flows show that the proposed model outperforms prior state of arts by achieving 99.15% malicious flow detection rate. Applying the model in the wild app detection, the result shows that our model performs better than other anti-virus scanners. It can also detect several malware types that can evade the detection of anti-virus scanners.
- We propose a detection framework that can cope with encrypted HTTPS and non-encrypted HTTP traffic in home networks, Bring-Your-Own-Device (BYOD) [10] enterprise networks, and 3G/4G mobile networks. Malware detection model is built and deployed on servers, which makes it suitable for large-scale deployment.

The rest of the paper is organized as follows. Section II highlights the related works. Section III introduces the background and motivation of the proposed solution. Section IV discusses the methodology in detail. Section V presents our evaluation. The limitation and discussion of the proposed solution are presented in Section VI. Finally, Section VII elaborates the conclusion of the study.

II. Related Work

The researchers have investigated the malware identification and private information tracking extensively. Due to the previously mentioned limitations of static and dynamic analyses, researchers begin to analyze and identify malicious apps using network traffic. We divide these related methods into three categories. They are network signature based methods, statistical feature based methods and lexical feature based methods respectively.

A. Network Signature

The signature-based detection methods evaluate the malware according to the predetermined malware signatures. Griffin *et al.* [11] extracted 48 bytes code sequence as a string signature of malware. In addition, automatic generation of network signatures has been explored in various previous works [12]–[14]. Most of these studies focused on worm fingerprinting. Perdisci *et al.* [15] focused on generating network signatures for mobile malware from their HTTP traffic. They analyzed the structural similarities among malicious HTTP traffic trace and then clustered the similar HTTP traffic. As for each HTTP traffic cluster, they automatically generated network signatures for each type of malware. In general, signature-based methods lack adaptability. It can detect known attacks, but has limited ability to handle novel attacks.

B. Packet/Flow Statistical Features

A combination of host-based statistics with SNORT rules to detect botnets was introduced in [16]. The authors showed that it is possible to detect malicious traffic using statistical features computed from network traffic data, which motivated further research in this field. Arora *et al.* [17] compared malware's traffic with benign network traffic and finally found the deviation of the malware on the network behavior. The statistical features they used contained average packet size, average duration of flow, the radio of incoming to outgoing bytes and other 13 features. In addition, a framework called AppScanner [18] was implemented that can achieve automatic fingerprinting and real-time identification of Android apps using the statistical feature of encrypted network traffic. Conti *et al.* [19] identified user actions by analyzing the statistical features of Android encrypted traffic. The statistical features-based methods only characterize traffic in a coarse manner, so they may trigger a high misjudgment rate.

C. Packet/Flow Textual Features

Some studies utilize text analysis for malware detection. Asdroid [20] detected stealthy behavior in Android app by identifying the disparity between UI textual semantics and program behaviors. However, it only used a few keywords to identify sensitive operations such as "send SMS" and "call phone". WHYPER [21] used NLP techniques to identify sentences that described the need for a given permission in the app description. Nan *et al.* [22] proposed a framework called UIPicker for identifying personal user information on a large scale, and this framework was based on a novel combination of NLP, machine learning and program analysis techniques. The N-gram model in NLP has been used in an automatic network protocol identification system designed for traffic analysis [23]. Recently, Recon *et al.* [24] is proposed to reveal and control personal identifiable information (PII) leaks in mobile network traffic, in which the key/value pairs are used for identifying PII. However, none of the works mentioned in this section aims at searching for the rich semantic features in the HTTP request header.

Our method uses lexical features of HTTP header to discover malicious behaviors. On one hand, comparing with signature-based methods, it can adapt swiftly to detect new attacks. On the other hand, with specific fields of traffic, it has a fine grained characterization on flows than statistical features, which greatly improves malware detection performance. Different from the existing approaches, the proposed one detects Android malware by combing with N-Gram method in NLP to search for the rich semantic features in the HTTP request header, when the extracted semantic features attain enhanced detection capability compared with coarse-grained features used in the existing works. In addition, we propose a detection system to cope with encrypted and non-encrypted traffic in home networks, BYOD enterprise networks, and 3G/4G mobile networks, which can be easily deployed.

III. Background and Motivation

The personal computer (PC) and mobile malware studies have many common characteristics. Yet, mobile devices present unique features that make them distinct from their PC counterparts [25], [26]. We elaborate the major differences as follows:

1) Limited device resource. Compared with PC, mobile devices have limited resources in terms of CPU, RAM, memory, and battery. However, most PC malware detection methods consume a wealth of time and resources. Therefore, these methods are not suitable for detecting mobile malware on mobile devices. Network traffic analysis has been proposed for mobile malware detection, which can be deployed in the gateway without requiring excessive resource usages on mobile devices.
2) Privacy concerns. Mobile device is more personal compared with PC. The attackers targeting mobile systems are generally more interested in the users' private information, such as the users' daily routines, personal contacts, and bank accounts [27]. Moreover, after the private data acquisition, the attacker can launch effective fraud attacks. Jiang and Zhou [28] have shown that more than 90% of malware-infected mobile terminals are controlled by botnets through network or SMS commands. Therefore, mining malicious behaviors through network traffic is a promising direction.
3) Communication protocols. Unlike the traditional PC malware, mobile devices utilize a minimum set of protocol types for network communications. The HTTP/HTTPS protocols are the major protocols that are used by mobile malware [15], [29]. Therefore, we need to exploit HTTP/HTTPS traffic for designing effective mobile malware detection methods.

Consequently in this paper, we focus on HTTP/HTTPS traffic based malware detection. We filtered out the SSDP, NBNS, LLMNR and DHCP traffic during the pre-processing stage, because these traffic are unrelated to malicious behaviors. As the DNS traffic does not directly relate to malicious behaviors, we also filtered out DNS traffic. In the end, HTTP and HTTPS traffic collectively constitutes the complete malicious traffic, and both HTTP and HTTPS packets' proportion in total malicious network data reaches 76.96% and 23.04%. Meanwhile, the proportion of malware samples using un-encrypted protocol HTTP for communication is 83.67%, while the rest of them are using HTTPS protocol. In particular, the focus is on the request header of HTTP flow because of its rich text semantics.

HTTP headers are typically in plaintexts. The HTTP request of a flow is comprised of three parts: request line, request header, and request body. The request line and request header are encoded by ASCII characters. An HTTP request header is an ensemble of structured fields (shown in Table I). Encoding type of the HTTP body is determined by the specific content it carries. Thus, we do not introduce the body information into each flow. In this paper, we regard the flow information as the communication language between client and server, and each flow as a text document. Word segmentation and N-gram features in NLP can be applied here for traffic processing. In fact, Yun *et al.* [23] have found great similarities between mobile traffic and natural language.

We verify whether the benign and malicious network traffic data contain useful semantic information about malware by performing a preliminary study. First, we select a few benign and malicious apps, and use our Android traffic collection platform (Fig. 2) to collect their network traffic data. For the two types of traffic data (i.e., benign and malicious), we use the word segmentation method in NLP to segment the HTTP request header data. The baseline of word segmentation includes special characters, such as ":", ";" ," " and "&". Then, we use simple filtering rules, such as filtering out the stop words (e.g., "the", "this" ,"a" and "an"), common words (e.g.,"HTTP", "html", "content-type" and "host") to remove obviously meaningless words from the word sets. Then, we calculate the weight of every word in the remaining vocabulary.

TABLE I
SEVERAL COMMON FIELDS OF HTTP REQUEST HEADER

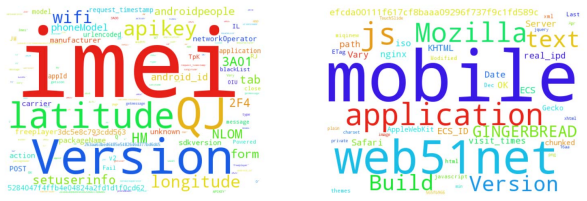
Field Name	Description
Host	This field specifies the Internet host and port number of the resource being requested, as obtained from the original URI given by the user or referring resource.
Referer	It allows the client to specify the address (URI) of the resource from which the Request-URL was obtained.
Request-URI	The URL from the request source.
Request-Method	The methods from HTTP that indicate the action to be performed on the identified resource.
User-Agent	It contains information about the user agent originating the request.

We use the term frequency to compute the word weight in the HTTP flow. The term frequency is widely used in NLP for extracting keywords. Term frequency refers to the occurrence number of a given word appearing in one document. This value is normalized in order to prevent its tendency to favor long file, as the same word's word frequency in the long file may be higher than that in the short file regardless of its importance. For a word t_i in a specific file f_j, its term frequency tf value can be expressed as follows:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}, \tag{1}$$

where $n_{i,j}$ is the number of word i appearing in the file j, and the denominator $\sum_k n_{k,j}$ is the number of all words appearing in the corpus. In our context, $n_{i,j}$ refers to the sum of occurrence number in traffic flows. Finally, the two word sets are expressed in the form of word cloud (see Fig. 1) in accordance with their respective weights.

Using this visualization process, the researchers can obtain a better understanding on the information in the flow request header. We discuss the meaning of visualizing flow request header of malware by illustrating the network traces from two exemplar apps. Fig. 1(a) shows the flow request headers from a malicious app, whereas Fig. 1(b) shows those from a benign app. We analyze the information conveyed by the two pictures. First, the most prominent word in Fig. 1(a) is the red word "imei" and the blue word "version". "Imei" is the unique identifier of the phone, and "version" can be used to identify the version of the app. Other conspicuous words, such as "longitude", "latitude", "wifi" and "apikey" are also found. These terms are closely related to device information and personal



(a) (b)

Fig. 1. Word clouds of malware and benign app's flow request header. (a) Request header of malware. (b) Request header of a benign app.

information of user. For example, actions associated with the words "longitude" and "latitude" can disclose the geographic location information of user. However, the benign network traffic only contains some common words, such as "mobile", "application" and "web51net". Thus, the maliciousness of an app can be determined by visualizing the HTTP flow request headers.

Similar to the analysis of text documents, semantic information can be extracted from traffic flows. The N-gram sequence of words in the flow is similar to that of natural language and exhibits a highly skewed frequency-rank distribution. Therefore, the N-gram sequences can be used as features to identify malicious traffic. The N-gram model is based on the assumption that the occurrence of the n-th word is only related to the preceding $n-1$ words, but is unaffected by other words. The appearance probability of a sentence is the product of the occurrence probability of each word in the sentence. In 1-gram, the appearance of each word is independent of each other. If the appearance of a word depends on only one word in front of it, we call this case as 2-gram. Similarly, different N values in N-gram can express the relationship between different words in mobile traffic.

The nature of malware recognition is a pattern classification problem. Machine learning technology is widely applied to malicious network traffic detection. When identifying Android malware using network traffic data, the network traffic data are classified as benign or malicious. After malicious network traffic is identified, the source app of the traffic can be classified as malware.

IV. Methodology

We propose a solution for detecting malicious apps using mobile traffic. The proposed method treats every HTTP flow as a document, and then uses the word segmentation based on N-gram generation to generate candidate features that can effectively characterize a specific HTTP flow. Specifically, a feature selection algorithm is developed to select meaningful features automatically from a bag-of-words for flow feature vectorization. Then, an SVM classifier is trained to automatically determine whether the unknown traffic is benign or malicious. The complete process is summarized as follows:

a) Traffic Collection: We design an Android traffic collection platform to collect the required network traffic data (Section IV-A).

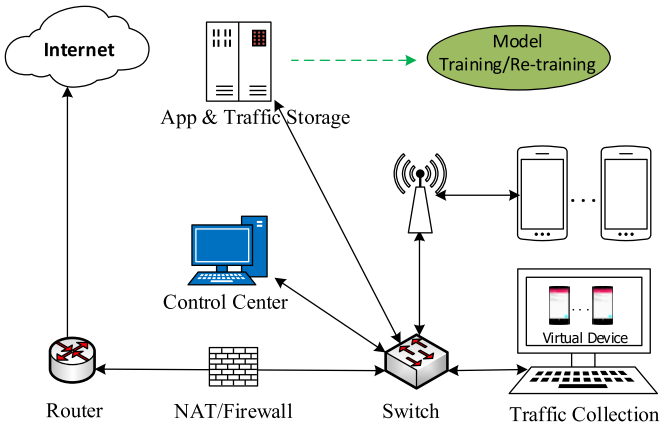


Fig. 2. The architecture of Android traffic collection platform.

b) Traffic Preprocessing: This module contains two parts: flow extraction and segmentation. The flow extraction algorithm extracts individual HTTP flow from mixed traffic and outputs the request header of every flow into one document. The content of each document is then divided into a set of words in flow segmentation (Section IV-B).

c) N-gram Generation: This component processes the word set using the N-gram method. The N-gram sequence of each word is regarded as one candidate feature to characterize HTTP flow (Section IV-C).

d) Feature Selection: The chi-square test algorithm is applied to select the best features. Accordingly, the benign and malicious mobile traffic can be distinguished (Section IV-D).

e) Model Training: The selected features from a bag-of-words are used for vectorizing every traffic flow. By instantiating these features, we derive a detection model based on a linear SVM model (Section IV-E).

A. Traffic Collection

The traffic collection platform is used for collecting two types of Android traffic data: malicious traffic generated by malicious apps, and benign traffic generated by benign apps. The traffic collection framework is deployed in the cyber security and privacy lab of University of Jinan [30]. At the gateway of the lab, a firewall and NAT server are present to ensure the safety of the traffic collection framework. As shown in Fig. 2, the platform comprises of the following three components: Control Center, App & Traffic Storage server, and Traffic Collection server. The Control Center connects to the Traffic Collection server, as well as App & traffic Storage server via a LAN switch. The Control Center is responsible for scheduling tasks and assigning Android apps from the App & traffic Storage server to a traffic collection machine in the Traffic Collection component. Meanwhile, we use real devices in the collection process to execute the malware samples who can escape the emulator. The collected traffic data are finally transferred to the App & traffic Storage server. All the collected traffic is then used to train/re-train an effective detection model.

Each traffic collection machine typically carries out two tasks: the execution of app and collection of network traffic

data generated by the app. We execute the apps on simulators, which are driven by a page walk-through algorithm integrated with an Android tool called Monkey [31]. The Monkey tool can randomly send events to the Android device when the app is running. The page walk-through algorithm explores all the components on each activity, which can trigger the malicious behaviors behind the activity pages. We execute a single app at one time on a simulator device to ensure that the network traffic data generated by each app is the ground truth. Accordingly, no app runs in the background when collecting app traffic. We then collect the network traffic packets of app during execution. These traffic data are saved as PCAP format files.

B. Traffic Preprocessing

We obtain numerous network traffic data through the traffic collection platform. In the actual malicious traffic detection, the network flow is a basic unit that the app interacts with the network. However, several HTTP flows generated by apps are mixed. Therefore, we must separate each complete flow from the mixed traffic file and further segment the flow.

1) Flow Extraction: We design an algorithm that can extract each flow and output the request header of each flow to a document. The specific process is described in Algorithm 1. The input of this algorithm is the collected traffic (i.e., initial traffic files), and the output is the documents that contain the request header of multiple HTTP flows. This algorithm is implemented using a combination of Python script and T-shark command [32], which can be used to obtain the request header of each HTTP flow.

Algorithm 1 The Algorithm of Obtaining HTTP Flow's Request Header

Data: Network traffic data (a pcap file that contains multiple HTTP flows) generated by one app.
Result: Text documents that hold HTTP flows' header.
Initialize the index id of every HTTP flow to 0;
while *true* **do**
  Use T-shark command to write the header of the HTTP flow which is pointed by the current index id to a text document;
  Get the bytes number of this text documents;
  if *the bytes number of the text document is smaller than a fixed number* **then**
    break;
  else
    Save this text document to the target folder;
    Delete the text document at the current location;
  end
  Increase the index id of HTTP flow;
end

2) Flow Segmentation: The segmentation of the flow request header is a challenging task, as no standard token (e.g., whitespace or punctuation) is available to cut flows into words. The left block in Fig. 3 displays an example of HTTP flow header. HTTP flow header contains considerable information, such as requesting method, encoding type, requested URL, and browser information which are arranged in a certain order. Many special characters can be included in the HTTP flow, such as ",",":",";", "&" and other characters in the flow. We can split such strings into separated words using these special characters. Finally, the request header is separated into a word set of "GET", "dm", "ad-maker.info", "acc", etc.

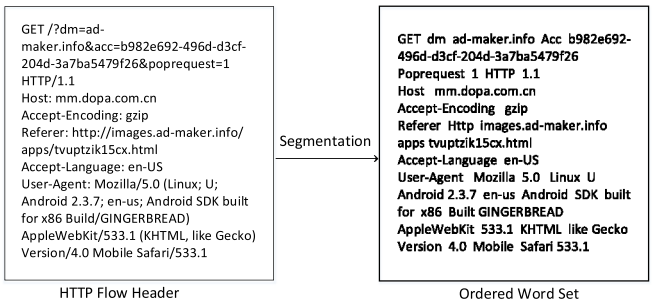


Fig. 3. Word segmentation of one HTTP flow's request header.

Given that not all divided words can be used in the detection of malware, we remove meaningless words in the traffic flow in accordance with prior knowledge. This procedure saves the operating costs of the method. We establish several filtering rules to remove meaningless words. The principles of filtering rules are described as follows:

a) *Meaningless Low-Frequency Words Removal:* The collected words contain a number of file names, such as picture names, JavaScript names and css file names. These file names have no regularity and are generally unrelated to malware's behaviors, so we remove these "low-frequency but meaningless words" based on the suffix ".jpg", ".png", ".gif", ".js", ".css".
b) *Common High-Frequency Words Removal:* Some fields in the HTTP flows, such as "content-length", "en-us" and "expires", appear in nearly every HTTP flow, which does not contribute to the traffic classification, thus are removed.
c) *Stop Words Removal:* Stop words include common words, such as "the", "is" and "were". We remove such words since they do not provide any contributions toward characterizing the HTTP flow header.

C. N-Gram Generation

N-gram is a sequence comprising several consecutive words. In the field of NLP, the N-gram is a sub-sequence comprising N elements that are included in a particular sequence of at least N elements.

The N-gram generation module is designed to provide semantic information for HTTP flow header. To achieve this goal, the N-gram generation module translates each incoming word set (derived from a flow segmentation) into an N-gram sequence. Fig. 4 presents examples of turning a word set of flows into N-gram sequences. The leftmost part is the initial word set, wherein each row represents a flow. These words form 1-gram sequence. The middle and rightmost portions are

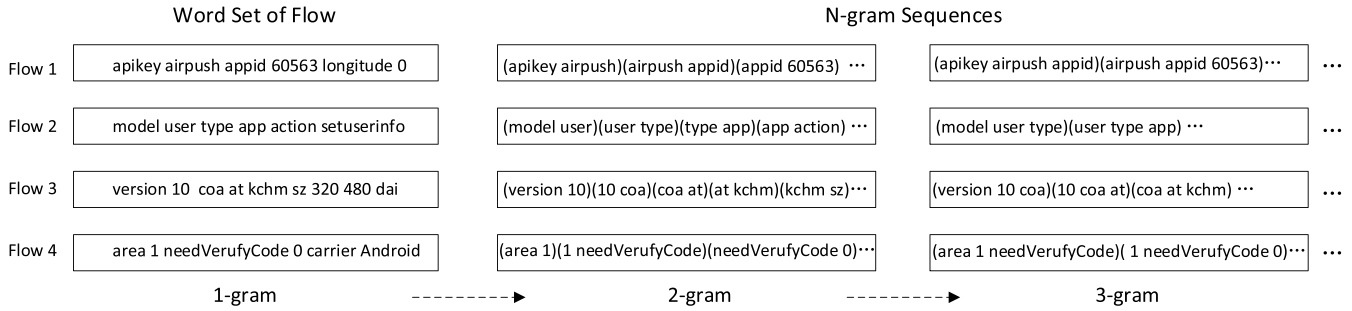


Fig. 4. Example of translating word sets of ?ows to N-gram sequences.

N-gram sequences generated by the corresponding word set in each flow. The middle part represents the 2-gram sequence when N is equal to 2, and the rightmost part represents the 3-gram sequence when N is 3. For example, the header of flow 1 yields a collection of "apikey airpush appid 60563 longitude 0", and its 1-gram sequence is identical to the initial collection, while its 2-gram sequence is "(apikey airpush)(airpush appid)(appid 60563)..." and its 3-gram sequence is "(apikey airpush appid)(airpush appid 60563)...".

The N-gram provides contextual information for extracting meaningful word sets. For example, from the 1-gram sequences "apikey" or "airpush" of flow 1, not only can we obtain the meaning of a single word as provided, but we also determine that no obvious dependency exists between the words. From the 2-gram sequences "(apikey airpush)" of flow 1, we can see that the appearance of word "airpush" is affected by the word "apikey".

The selection of N value is of utmost importance, as it reflects the rule of word appearance in HTTP flow header. Here, we perform experiments with different values of N, and the detail of which is described in Section V-B.

D. Feature Selection

The selection of features is critical, because they can directly affect the performance of the model. In the previous step, we use N-gram sequence as flow features. However, we need to reduce the number of features owing to the following reasons:

1) *Complexity reduction.* Many data mining algorithms need much more time and resources when the number of features increases. Therefore, reducing the number of features is important for saving time and resources.
2) *Noise reduction.* Additional features do not always translate to the improvement of the algorithm performance. On the contrary, they may bring serious model over-fitting issue. As a result, selecting a set of appropriate features can reduce the chance of model over-fitting.

Not all N-gram sequences in our feature set are useful to the malware detection model. These unessential sequences can negatively affect the normal operation of the algorithm. Therefore, we filter out the features using chi-square test [33], which is a univariate feature selection algorithm.

Chi-square test is a statistical test that is widely used to determine whether the expected distributions of categorical variables significantly differ from those observed. This test utilizes a measurement method to assign the feature a certain weight, called *chi-squared test value*, to characterize the correlation between categories. Chi-square test can set a fixed threshold or select top K chi-squared test values for feature selection. High test result means high importance of the feature to the characterized category.

We test whether a specific feature in the N-gram sequence set is significant or not depending on its term frequency (see Eq. 1) in two opposite data sets (i.e., N-gram sequence sets from malicious and benign flows). Specifically, we determine whether a feature t (an N-gram sequence) and a category c (malicious) are independent of each other. If yes, then we regard feature t as insignificant to classify class c. In other words, we cannot determine whether a flow belongs to class c using feature t. The formula for calculating the chi-squared test value of a feature t and class c is defined follows:

$$\chi^2(t, c) = \frac{(N_{t,c} - E_{t,c})^2}{E_{t,c}},$$

where $N_{t,c}$ refers to the term frequency of feature t in class c, and $E_{t,c}$ is the expected term frequency of feature t in class c when they are independent of each other.

E. Model Training

The model training mainly consists of two steps: one is converting text type word set into numeric vectors and the other is feeding the computed numeric vectors into SVM algorithm to train the malware detection model.

1) Word Vectorization: After the HTTP header segmentation operation, each flow is transformed into a word set. Their N-gram sequences are then generated and regarded as features. After the feature selection, we obtain a bag-of-words. Using the bag-of-words, binary values for each flow can be produced. We ignore the frequency of a given N-gram sequence appearing in one flow and focus only on whether each N-gram sequence appears in the bag-of-words. If the given N-gram sequence exists in the bag-of-words, then the corresponding value with the N-gram sequence is "1". Otherwise, the value is "0". This encoding type is also known as one-hot encoding [34]. Through this encoding method, each HTTP flow can be transformed into a numeric vector with a dimension that is equal to the length of the bag-of-words.

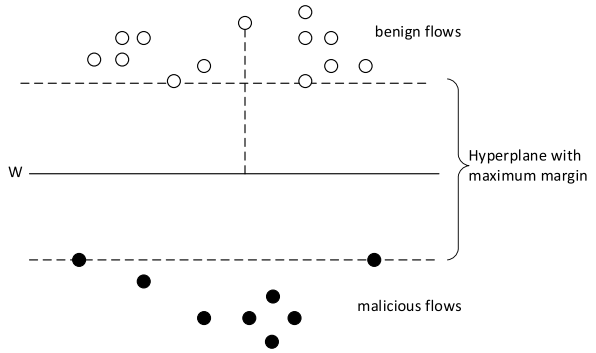


Fig. 5. The schematic diagram of SVM algorithm.

2) Model Building: We identify malicious traffic and discover malware by searching for the source app of malicious traffic. We use an SVM [35] model to learn the features of two classes of elements. SVM is a supervised algorithm that learns a hyperplane with a maximum margin for classification (i.e., classifying unknown traffic data as benign or malicious). Given two class vectors as training data, a linear SVM determines the hyperplane that separates two classes with a maximal margin. The schematic of the SVM algorithm is depicted in Figure 5. One class is associated with malicious traffic, while the other is associated with benign network traffic. SVM considers a hyperplane that satisfies the classification requirement, and ensures the points in the training set are as far as possible from the classification surface. In particular, the method searches for a classification surface to maximize the margin on both sides of the classification.

The detection model of linear SVM simply maps the feature vector V of an HTTP flow x to the direction of the hyperplane. The corresponding detection function F is given as follows:

$$F_{(x)} =< W, V >,$$

where W denotes the weight vector, which should be continuously adjusted during the training process. $F_{(x)} > 0$ (or a given value) indicates malicious activity while $F_{(x)} < 0$ corresponds to benign traffic flows. Moreover, $F_{(x)} = 0$ represents the separating hyperplane. An unknown HTTP flow x is classified by calculating its F_x value to determine which side of the hyperplane it falls onto, which in turn determines whether the HTTP flow is benign or malicious.

The trained SVM model can also be loaded and utilized. Unknown traffic files (mixed flows) can be processed using the method described in Section IV-B (i.e., flow extraction and segmentation) and Section IV-C (i.e., N-gram generation). For the features with built-in bag of words, every HTTP flow is transformed into a numerical vector after vectorization. The numeric vector is then used as input to the detection model. After that, the detection model determines whether a specific HTTP flow in the traffic file is benign or malicious, and whether the source APK of the same traffic file is a benign app or a certain type of malware. When the new malware is added to the training samples, we will re-train and update the classifier for detecting such new type of malware.

V. Evaluation

We implement the proposed method using Python with Scikit-Learn, a popular machine learning library. In this section, we first introduce all the training data sets used in the experiment. The data sets contain the app data set and a mobile traffic data set generated by the apps (Section V-A). Subsequently, we analyze the influence of parameter N and feature number K on the detection model, after which the best detection is selected based on the detection performance of different parameters (Section V-B). We also analyze the number of training samples for the detection model to achieve good detection results (Section V-C). We compared Our method with other machine learning methods, static detection methods, and traffic flow-based detection methods (Section V-D). The effectiveness of our model on newly collected malware samples is well evaluated (Section V-E). Finally, the detection model is applied to a real-world environment (Section V-F). In addition, we consider the identification of encrypted malicious traffic (Section V-G).

A. Data Sets

Our initial malicious app set is obtained from VirusShare [36], a repository of malware samples made accessible to security researchers. This malware set contains 8203 samples, which have been shared by the website from 2014 to 2016. More than 9000 apps are obtained from Baidu mobile assistant [37], a popular third party market. These apps are collected from 2014 to 2016. Not all apps from the Android market are benign. To ensure the accuracy of the training set, we upload each downloaded app to VirusTotal [38], and then identify the benign apps based on the detection results from VirusTotal. Only those apps that VirusTotal determined as benign are included to our benign app set. Ultimately, our app sets contain 8203 malicious apps and 8168 benign apps. Using our own designed traffic collection platform (Fig. 2), the malware and benign apps generate 14.2 GB and 12.9 GB of network traffic data, respectively. We extract 113,735 and 166,973 HTTP flows respectively from these traffic data.

Notably, not all network traffic data generated by malicious apps correspond to malicious traffic. Many malware take the form of repackaged benign apps; thus, malware can also contain the basic functions of a benign app. Subsequently, the network traffic they generate can be characterized by mixed benign and malicious network traffic (i.e., most of the traffic are benign traffic while only a small portion is malicious network traffic). To further ensure the label accuracy for the training set, we perform a verification on the network traffic generated by the identified malicious apps. We extract the host or target IP field of each flow and upload it to VirusTotal. If the host or target IP is malicious, then it is considered as a malicious flow. This malicious network traffic flow is added to our malicious traffic data set as the training set sample. Finally, we obtain 5,258 malicious flows labeled as malicious traffic. We also clean the benign flows by removing incomplete flows. After the cleanup, we obtain 31,706 benign flows. The final data statistics are shown in the Table II.

TABLE II
DATASET STATISTICS

	Malware	Benign apps
App Number	8203	8168
Collected Traffic Data	14.2GB	12.9GB
Total Flow Number	113,735	166,973
Flows Labeled Confidently	5258	31706
Training Flow Number	5258	31706
Training Flow Data	219MB	1.01GB

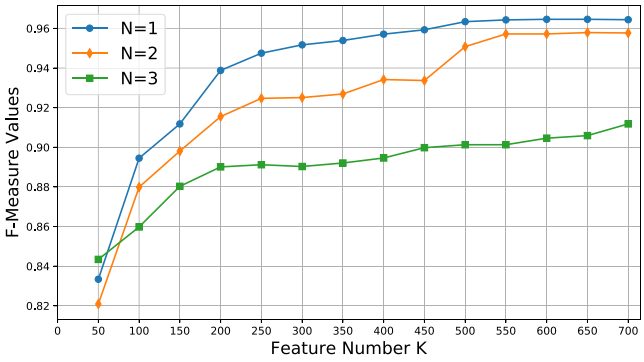


Fig. 6. Parameter setting according to F-Measure values at different N and feature number K.

B. Parameter Setting

A good detection model should identify as many malware apps as possible and minimize the False Positive Rate (FPR). Compared with several other model evaluation metrics, F-Measure value is a relatively fair metric to understand and compare the performance of malware detection [39], because that malware or malicious traffic recognition is actually an imbalanced classification problem, and our data (i.e., 5258 malicious samples and 31706 benign samples) proves this derivation. In imbalanced problems, the detection rate or FPR cannot determine whether a model is good or poor. On the contrary, F-Measure comprehensively considers the detection rate and error rate of identifying malicious samples, and thus is suitable for model evaluation. We therefore utilize the F-Measure value as the metric of evaluating parameter N and feature number K.

We use the 10-fold cross validation method to verify the performance of the detection model. The training set is split into 10 subsets and the model is trained using 9 subsets as training data, while the generated model is validated on the remaining data (i.e., the remaining subset is used as a test set to compute the performance measure). The complete training and testing process is repeated for 10 times, and we use the average metric value as the performance indicator. Fig. 6 shows the F-Measure values for different N values and feature number K. In the figure, the horizontal axis represents the number of features ranging from 50 to 700, while the interval is set to 50. The vertical axis represents the values of F-Measure. Lines with different colors and markings represent the different N values, which vary from 1 to 3. The blue line represents F-Measure values when N equals 1 and feature number K ranges from 50 to 700.

TABLE III
DIFFERENT METRICS OF THE DETECTION MODEL AT $N = 1$ AND $K = 600$

Metric	Accuracy	FPR	Recall	Precision	F-Measure	AUC
Value(%)	95.31	0.45	99.15	95.53	97.30	97.26

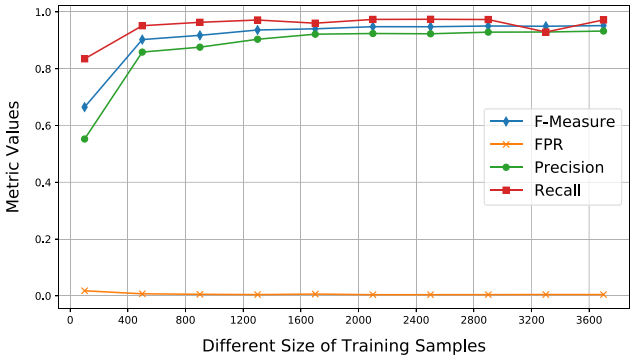


Fig. 7. The F-Measure, FPR, precision and recall values using different sizes of training set.

As shown in Fig. 6, $N = 1$ achieves better F-measure performance than that of $N = 2$ and $N = 3$. Consequently, we set the value of N as 1. This result also suggests that the frequency of each word in HTTP flows is nearly unaffected by previous words (i.e., different words are independent from one another). In addition, as the feature number K increases, F-Measure value also increases. However, the increasing trend gradually slows down. When the number of features reaches 600, the value of F-Measure keeps stable. Thus, the feature number should be set as 600. Based on the experimental results, the optimal values of the detection model are $N = 1$ and the feature number $K = 600$ for our experiment. At this point, the identification rate for traffic flows reaches 99.15% whereas the misjudgment rate for benign traffic is only 0.45%.

We also evaluate the time efficiency of the solution in relation to traffic flow. The experiments are conducted using Windows 7 with Intel (R) Core(TM) i5-4460 with 3.20 GHZ CPU and 16.0 GB memory. It takes around 5 hours and 42 minutes to train and validate the detection model using 31,706 benign and 5,258 malicious flows. Table III shows the common evaluation metrics using the best model (i.e., $N = 1$ and $K = 600$).

C. Evaluation of Sample Size Requirements

We design an experiment to answer the question: "how many training samples are needed for the detection model to reliably detect malware?" The number of labeled instances is changed from 100 to 3700 in measuring the effect of sample size on the final performance of our method. Fig. 7 illustrates the final result.

The horizontal axis in Fig. 7 refers to the different sizes of the training samples which range from 100 to 3700. The vertical axis corresponds to the indicators of F-Measure, FPR, precision and recall for different training set sizes. Three of the metrics present an increasing trend as the size of the training

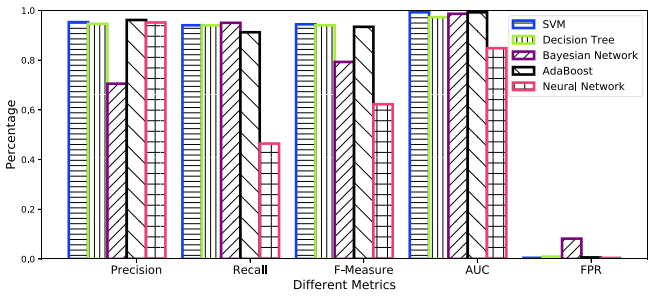


Fig. 8. The comparison of SVM and other machine learning algorithms.

set increases, however, the increasing trend plateaus eventually. Specifically, when the training set contains only 100 samples, the detection rate of the model on malicious flow is 55.24%. When the size of the training set increases to 500, the detection effect of the model is greatly improved, and the recognition rate has reached 85.86% on malicious network traffic.

In addition, the FPR is very low, and is even lower as the size of training set increases. When the training set contains only 100 samples, the misjudgment rate on benign malware samples is extremely low at 1.78%. When the training set contains 3700 samples, the recognition rate on malicious samples reaches 93.27%. When all training samples (31706 benign samples and 5258 malicious samples) are used, this model can get a detection rate of 99.15% on malicious flows and obtains an FPR of only 0.45%. The experimental results indicate that only a small amount of data samples are needed to train an accurate and reliable detection model.

D. Comparison With Existing Approaches

The detection model performs well on known data sets (Section V-A). However, the proposed solution should be compared with other methods for verification. For this purpose, we select other machine learning algorithms, static detection method and mobile traffic-based detection methods for comparison.

1) Comparison With Other Machine Learning Algorithms: Here, we compare the classification performance of SVM with other popular machine learning algorithms. We have selected several popular classification algorithms, such as Decision Tree, Bayesian Network, AdaBoost and Neural Network. For all algorithms, we attempt to use multiple sets of parameters to maximize the performance of each algorithm. As seen from Fig. 8, the trained models using different algorithms have different performance metrics. Specifically, the FPR rate of SVM is lower than decision tree, Bayesian Network, and Adaboost. Although the FPR value of Neural Network is lower than SVM, SVM algorithm has a higher recall, precision, F-Measure, and Area Under a Curve (AUC) performance. Overall, the experimental results show that SVM outperforms other methods.

2) Comparison With the Static Analysis Method: We first compare the performance of our method with a well-known static detection method called Drebin [40]. The Drebin method is selected because most obtained malicious apps in this study are from the Drebin project. We use our traffic collection platform to collect the network traffic generated by these malware samples. Then, we use the method described in Section IV to process traffic information, and ultimately create a malicious network traffic detection model. An app may produce considerable network traffic. To ensure a reasonable comparison with the Drebin method, we determine the source app of each network traffic. In particular, once an app generates malicious network traffic, the app is predicted as a certain type of malware. Finally, our detection model obtains a detection rate of 92.22% on malicious apps and a misjudgment rate of only 1.33% on regarding benign apps as malware ones. Drebin can detect approximately 94% of malware samples at an FPR of nearly 1%. Therefore, our approach is slightly inferior to the Drebin method. The result may be caused by our lack of malware samples. We are able to collect the network traffic from a vast majority of malicious apps but fail to do so on a portion of them.

In addition, the Drebin project presents a list of undetected malware. We extract 85 malware samples from the list, indicating that these 85 malware samples are not correctly classified by the method of Drebin. We attempt to use our trained model to classify the traffic data generated by these 85 malware samples. Finally, we correctly identify 12 malware samples from the network traffic of the malware samples above. We further analyze these 12 detected malware and find that they come from different malware families, namely, Gappusin, SpyPhone, Glodream, Plankton, DroidSheep and Sdisp. The malware samples from Gappusin family account for 6 of the 12 samples. We further analyze the result for other possible causes. Gappusin is a popular malware family and its main functionality is to induce malicious charges [29]. Gappusin malware samples can interact with networks several times, which makes our method suitable for identifying them. Meanwhile, we select two popular anti-virus scanners to detect the malware types–AegisLab (foreign produced) and 360 (domestic produced). The detection results of AegisLab and 360 are shown in Table IV, which also prove that our method can be used as a supplement to boost the performance of the static analysis methods.

3) Comparison With the Network-Level Statistical Method: We previously mentioned several related works on malware detection based statistical features of network traffic. Most of the malware detection works are based on the statistical characteristics of network traffic. The study in [17] observes and analyzes multiple network traffic data, and summarizes 16 statistical features that can effectively distinguish between benign and malicious network traffic data. In addition, they identify seven particularly important features (Table V) from 16 statistical features. Their experimental results show that the seven features are sufficient to effectively distinguish between benign and malicious flows.

We use our data set (described in the section V-A) to extract the aforementioned seven statistical features of 31,706 benign traffic flows and 5,258 malicious traffic flows. We still use the linear SVM algorithm without changing any of its related parameters. The final training model attains a detection rate of 86.35% on malicious traffic, and an FPR of 3.65%.

TABLE IV

COMPARISON BETWEEN OUR METHOD AND OTHER TOOLS ON PARTIAL UNDETECTED MALWARE WITH DREBIN ($\times = undetected$, $\checkmark = detected$)

Apk Name	Family	Drebin	Our method	AegisLab	360
0a4d010be1742edb7cd76ffef94e7a1d4cc8c1def1ac37930403ff9f5ccc4841	SpyPhone	×	✓	×	×
2bff76df09ffa038c77ad44ecced13f1125c258b2fa60fd229002f7f1dafda15	Gappusin	×	✓	✓	✓
2da5ac324ed6e0f53c61a97f4bf9f850ed27276daa1681d0df0a91465d456cc4	Glodream	×	✓	✓	✓
4fca2064417bf4c9e96131aaa59021a56f6d01c18d74d085cf01cb9e9c4e6e25	Gappusin	×	✓	✓	✓
00fb209d9f9f1b68b4e13076b555ca1a6da4482f5d1c9af189704e5b5195a09e	Plankton	×	✓	×	×
0a073153ba87aebad2f47bc8a0eb3a40f4f24599c7f3926502c070133a09c1c0	Gappusin	×	✓	✓	×
1eac7abecf8d332eb46c304ed539ce25fc1e7327ad3917986d6beac2774c5110	Gappusin	×	✓	✓	✓
6cefc340b511e4d69c9850786cdd80d6ec7c6cf694800dae8afa3d9482353621	Plankton	×	✓	✓	✓
4d873af4820deacba019f177b37f13495910568bf3547b8acdc10c06daf094b5	Sdisp	×	✓	✓	×
4f6bf46228a819f7a28b1304cf20bf2bc661ce3a23a7d759fa19eb5c2cc29ae4	DroidSheep	×	✓	✓	✓
5ee2beebcf60dbcab19c8908c9cddfcae34934eeaeb54b4b95a01fc823aada67	Gappusin	×	✓	✓	✓
6affb9db0486bcde3222ec2512c706554694097b46b247b0e3075e0a22ed5867	Gappusin	×	✓	✓	✓

TABLE V

NETWORK-LEVEL STATISTICAL FEATURES THAT CAN DISTINGUISH BETWEEN BENIGN AND MALICIOUS TRAFFIC

ID	Traffic Feature
1	Average Packet Size (in bytes)
2	Ratio of Incoming to Outgoing Bytes
3	Average Number of Bytes Received per Second
4	Average Number of Packets Sent per Flow
5	Average Number of Packets Received per Flow
6	Average Number of Bytes Sent per Flow
7	Average Number of Bytes Received per Flow

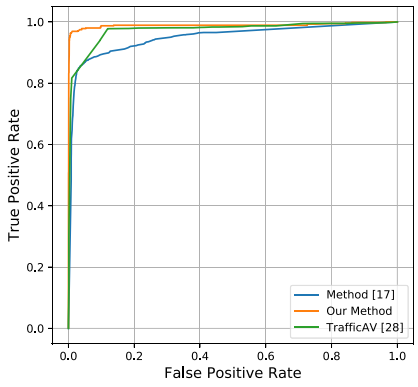


Fig. 9. The ROC curve comparison of our method, network-level statistical method and TrafficAV.

The experiment result is shown in Fig. 9 as a ROC curve, while the detection rate (TPR) is plotted against the FPR for the different thresholds of the detection methods. As shown by the ROC curve, our method represented by green line outperforms the network-level statistical method represented by blue line.

We further analyze the advantages of our approach by comparing it with the network-level statistical method. In terms of detection rate, the present model is significantly better than the statistical feature-based method regardless of the Accuracy or FPR. One reason can be that the statistical feature selection is very difficult, as it requires researchers to not only observe and analyze multiple network traffic data, but also to obtain a certain degree of understanding on the network behavior of malicious apps. For example, some malicious applications can continuously interact with the remote server, and constantly receive instructions to download malicious codes; hence, the downloaded traffic is far greater than the uploaded traffic.

To identify the distinctive features of benign and malicious traffic, researchers must understand the malware's interactive modes to a certain extent. However, as anti-detection capabilities of malware continue to increase, malware is increasingly difficult to detect. Consequently, benign and malicious apps have become more difficult to distinguish. Thus, simply using statistical feature to distinguish between malicious behavior and benign behavior is increasingly challenging.

4) Comparison With Other HTTP Header Analysis Methods: Given that our method focuses on analyzing HTTP flows, we select two methods corresponding to specific HTTP fields for malware detection. TrafficAV [29] shows some useful features obtained from HTTP requests that can be used to detect malware, and utilizes several specific fields in HTTP request headers. Accordingly, the features of HTTP request headers are identified based on the findings from TrafficAV, after which the features are trained for the detection model using SVM algorithm. The identified features include "host", "request-method", "request-url" and "user-agent" and their descriptions are listed in Table I. Our original app data and traffic data are also used to extract the above four features, after which 24,098 malicious HTTP requests and 94,898 benign HTTP requests are obtained. The detection model is tested using the linear SVM algorithm and the 10-fold cross validation. The model attains a detection rate of 91.01% on malicious traffic. The ROC curve comparison of TrafficAV and our method is shown in Fig. 9. The findings suggest that TrafficAV performs better compared to the statistical feature method; however, our method is even better than TrafficAV.

We also compare our method with another malware detection method based on HTTP specific field called DroidClassifier [41]. DroidClassifier focuses on HTTP header fields and extracts five features from HTTP request: "host", "referrer", "request-URL", "user-agent", and "content-type". This method is a score-based classification method according to these HTTP header fields. The final detection rate of DroidClassifier is 94.33% which is lower than that of our method.

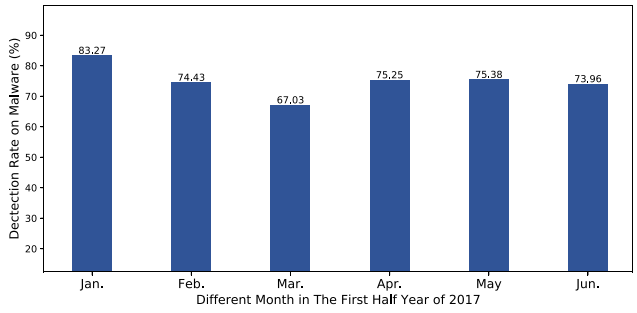


Fig. 10. The detection rate of our method in detecting new malware discovered in 2017.

E. Evaluation Using New Malware Samples

To bypass the anti-virus scanners, the attackers may attempt to generate new malware variants to poison and cheat the detection mode. In order to evaluate the effectiveness of our model using new malware samples, we use our collected data set (dated from 2014-2016) to train a detection model and test on new malware samples collected from VirusShare in 2017.

We downloaded 2812 newly discovered malware in the first half year of 2017 from VirusShare website and divided the test samples by months. Malware numbers in each month are 275, 485, 285, 602, 731 and 434 respectively. In total, 6.22GB network traffic from malware samples are collected. We extract 3532, 13787,13527, 16280, 25750 and 10861 HTTP flows from the traffic data. The trained model is applied to detect these HTTP flows and the detection rates on malware in different months are shown in Fig. 10. Our model achieves the best detection performance (83.27%) on malware of Jan. 2017, whereas it discovers 67.03% malware samples of Mar. 2017. The result shows that the freshness of malware indeed affects the model's detection performance, but our model can effectively detect most of new malware. The experimental results prove that the model trained by the malware samples from 2014-2016 can be effective in detecting new malware discovered in 2017.

F. Evaluation Using Apps in the Wild

We verify the accuracy of our detection model using apps downloaded from Android application markets. These apps differ from those apps from which the traffic data are obtained in the training process. The traffic data generated by these wild apps are obtained from the traffic generation and collection platform. These traffic data are processed (i.e., flow extraction, flow segmentation, and N-gram generation), and transformed to word vectors. Using the trained detection model, each flow's label (i.e., malicious or benign) is determined. The app is marked as malware, when it contains malicious flows.

In the wild app set of 1407 apps, 655 malicious apps are confirmed by the detection report of VirusTotal. The 655 malicious apps are filtered by 56 anti-virus scanners in VirusTotal; however, each scanner in VirusTotal can only detect part of these malware samples. By contrast, our detection model can identify 359 out of 655 apps, thereby verifying the capability of our model to scan wild apps.

We compare the performance of our model against nine selected anti-virus scanners which are ANG, AegisLab, Cyren, NANO-Antivirus, Sophos, Antiy-AVL, McAfee, F-Secure and BitDefender respectively. The detection rate of each scanner is sourced from the VirusTotal service. The detection rates of common anti-virus scanners vary considerably. The best scanner is AegisLab which can detect 46.39% of malware, whereas the scanner BitDefender only discovers 13.64% of the malware in the wild app set. Fig. 12 shows the detailed statistics. For this wild app set, our method provides the best performance with a detection rate of 54.51%, which outperforms nine other anti-virus scanners.

Fig. 13 shows the Venn diagram of the detection effort from our method and 56 other anti-virus scanners listed by VirusTotal. In the figure, the A and B union represents 726 apps that our method labels as malware. The B and C union represents 655 apps that anti-virus scanners determine as malware. The shared B area represents 359 apps that both our method and other anti-virus scanners in VirusTotal determine as malware. The C area represents 296 apps that our model determines as benign, but the other anti-virus scanners in VirusTotal identify as malware.

Furthermore, the A area in Fig. 13 represents 367 apps that our method determines as malware, but the other 56 anti-virus scanners in VirusTotal list as benign. We then obtain the URLs from the traffic data generated by 367 apps, and then upload them to VirusTotal (i.e., the website also provides the detection service of URL). From these URLs we detect numerous malicious URLs, and ultimately based on these URLs, we can determine 167 true malware, as shown by the area filled by slash in area A. Therefore, our method can effectively identify malware samples in the wild and can identify the malware samples that are missed by other anti-virus scanners.

G. Encrypted Traffic Analysis

With the increased awareness toward the security of personal information, more and more network traffic data is encrypted by HTTPS protocol. Nowadays, more and more families of malware prefer to interact with external networks using encrypted traffic, for instance, in our collected traffic generated by malware, about 23% of the packets are HTTPS packets. Encrypted traffic is much harder to analyze, and the proposed method cannot deal with encrypted traffic directly. However, we can decrypt the encrypted traffic before it was sent to the detection server, and then apply our malware detection model.

We propose a detection system for encrypted traffic following the design of Haystack system [42]. In particular, the system leverages the VPN API on mobile devices to provide full access to the network traffic of the device. User needs to install a lightweight app on his device, which is used for authorization and to receive messages from the detection server. A forwarder in gateway performs two major functions: one is to transparently bridge packets on the VPN interface and payload data on the regular socket interface, and the other is to forward traffic to the detection server for analysis. The traffic

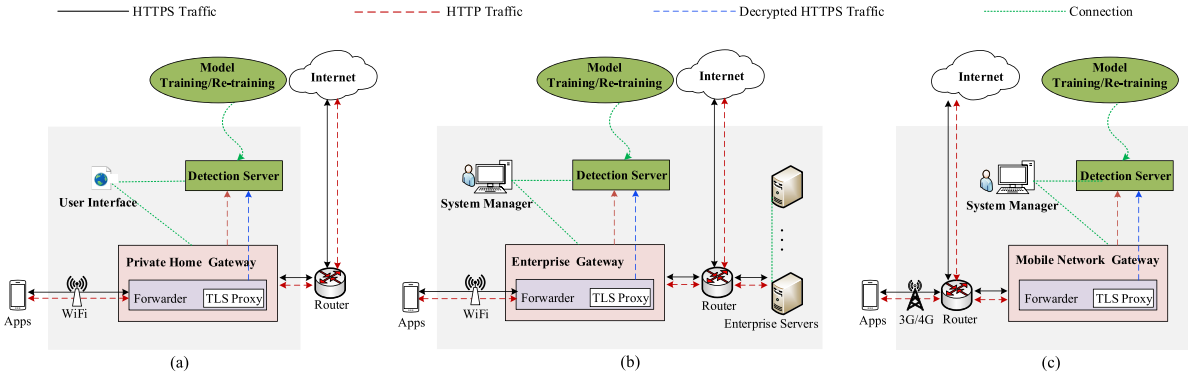


Fig. 11. Prototype system for deployment in different network scenarios with encrypted traffic analysis capability. (a) Home Network. (b) Enterprise Network (BYOD). (c) Mobile Network (3G/4G).

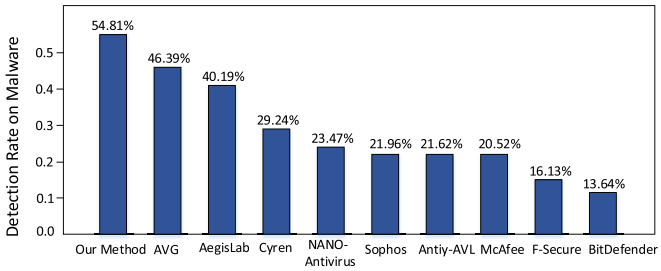


Fig. 12. Detection rate comparison on malware in the wild of our method and other anti-virus scanners.

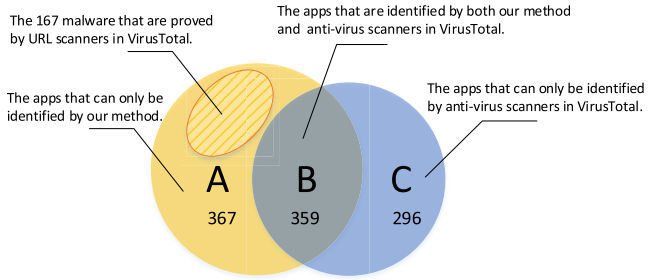


Fig. 13. The detection results on malware in the wild of our detection model and the anti-virus scanners.

analysis method is integrated into the detection server. For the decrypted network traffic, the detection model is trained to find suspicious network behaviors. Moreover, the detection model on the detection server will be regularly updated to improve the detection capability on new malware samples. The system explores the VPN interface to gain visibility into the underlying HTTPS traffic by forwarding the original packets before they are encrypted. Fig. 14 shows a flow request header of an HTTPS session in "Mi Store" app installed on an Android device. The lexical structure of this header has no difference from those of HTTP packet (i.e., Fig. 14). Therefore, the proposed method can be extended seamlessly to the HTTPS traffic generated by mobile apps.

Fig. 11 shows the architecture of our detection system for home networks, BYOD enterprise networks, and 3G/4G mobile networks. In home networks, once the detection system has been authorized by the user, encrypted HTTPS or non-encrypted HTTP traffic will be forwarded from the private home gateway to the detection server. The detection server

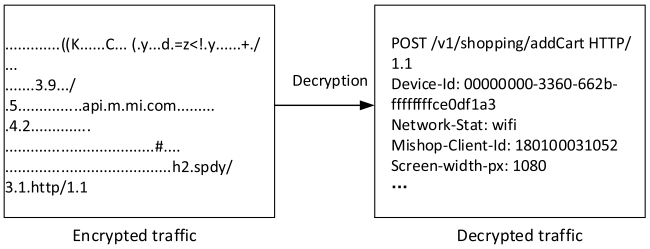


Fig. 14. The comparison of encrypted traffic and corresponding decrypted traffic.

uses our trained model to detect malicious behaviors from traffic data and push the notifications to user once a malicious action is identified. The malware detection model is built and deployed on servers, which makes it suitable for large-scale deployment. Simultaneously, it supports the malware detection for multiple devices, and is agnostic to operating system types. In order to attribute the detected malicious network flows to a user and an app, we extract the process ID (PID) of the process that generates the flow from the system's proc directory, and then map the PID to an app name using Android's Package Manager API [24], [42]. Mobile user can control to open or shut off the traffic detection service through user interface.

Fig. 11(b) shows the application of detection system for BYOD [10] enterprise network. A key challenge in BYOD enterprise network administration is the maintenance of a holistic view of devices and their application behaviors on the network. Nonetheless, observing basic device connection activities using traditional network monitoring tools is possible. But in-depth information, such as device context and detailed application connection information, is invisible to the traditional tools. These factors bring additional security risks to the BYOD enterprise networks, while our detection system can provide more visibility into these networks. The only difference of our detection system between BYOD enterprise networks and home networks is that in the enterprise networks, turning on or off detection service is controlled by the system manager. Similarly, Fig. 11(c) shows the detection system in 3G/4G mobile networks. In 3G/4G mobile network, the traffic detection service is customized for user by mobile service provider. When the network traffic passes through the

provider's router, only traffic data whose producer (user) has customized the service will be sent through our detection system.

VI. Limitations and Discussions

The previous evaluation demonstrates the efficacy of our method in detecting recent malware using their network traffic. In this section, we discuss the limitations of the proposed method.

1) Not All Malicious Behaviors Can be Triggered: During the traffic collection, we restart every app because the statistical result [28] shows that 83.3% of malware can be activated by restarting. We use the Monkey tool to randomly send events for traffic collection. Moreover, we have redesigned a page walk-through algorithm integrated with Monkey to explore all the components on each activity, which can help trigger the malicious behaviors as much as possible. Meanwhile, we add real devices in the collection components to execute some malware samples who can escape the emulator. However, inevitably, some of the malicious activities may not be fully triggered without effective user inputs. We plan to design an input generator to better emulate the user input behavior in future, so that the collected traffic will resemble real world network traffic.

2) The Dependence on Selected Features: Our method can automatically select N-gram features to establish a malware detection model. We use machine learning techniques to generate detection model. While learning techniques provide a powerful tool for determining unknown data category, they require a representative benchmark data for training. That is, the quality of the detection model is critically determined by the quality and quantity of malicious and benign network traffic. As mentioned in Section V-B, the size of selected features and the coverage of the features can significantly affect the detection performance. The proposed method can only detect unknown samples that present some common characteristics (i.e., N-gram features) with the malware samples in the training data set. If the mobile traffic of a new type of malware does not contain any features in the selected word, then the method have difficulty in identifying this type of malware. Thus, multiple training data, especially malicious traffic data are needed to improve our model detection performance in the wild. To address this issue, we plan to collect additional malware samples from various channels to further expand the coverage of selected features in the future. At present, our malware samples are mainly collected from VirusShare [36]. We will collect more latest malware samples over the time. Obtaining new malicious samples based on the detection reports of VirusTotal is also feasible. Furthermore, once the new malware is added to the training samples, we will re-train and update the classifier to detect new malware.

VII. Conclusion

Android malware is a new yet fast growing threat. At present, many research methods and anti-virus scanners fail to adapt to the growing amount and diversity of mobile malware. As a remedy, we introduce a solution for mobile malware detection using network traffic flows, which treats every HTTP flow as a document and analyzes the HTTP flow requests using NLP string analysis. The N-gram sequence generation, feature selection algorithm, and SVM algorithm are used to build an effective malware detection model. Our evaluation demonstrates the potential of this solution, and our trained model outperforms the existing approaches and identifies malicious flows with few false alarms. The detection rate for malicious flows reaches 99.15% whereas the misjudgment rate for benign traffic is only 0.45%. The effectiveness of the proposed method is further validated using newly discovered malware. When applied in a real environment, the model can detect 54.81% of the malicious apps, which is better than other popular anti-virus scanners. The experimental results also demonstrate that our model can detect malware samples that evade the detection of other anti-virus scanners. Moreover, we propose a detection system to deal with encrypted traffic in BYOD enterprise networks, home network, and 3G/4G mobile network.

References

[1] *Google Play: Number of Apps 2009–2016*. Accessed: May 2017. [Online]. Available: http://www.statista.com/statistics/266210/

[2] *Security Threat Report 2014—Sophos*. Accessed: May 2017. [Online]. Available: http://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecurity-threat-report-2014.pdf

[3] P. Faruki *et al.*, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. Tuts.*, vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015.

[4] S. Arzt *et al.*, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[5] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware through static analysis," in *Proc. ACM SIGSOFT Int. Symp.*, 2014, pp. 576–587.

[6] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "CHEX: Statically vetting Android apps for component hijacking vulnerabilities," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 229–240.

[7] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. Comput. Secur. Appl. Conf.*, Dec. 2007, pp. 421–430.

[8] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, p. 5, Jun. 2014.

[9] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 1–42, 2012.

[10] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards SDN-defined programmable BYOD (bring your own device) security," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2016, pp. 1–15.

[11] K. Griffin, S. Schneider, X. Hu, and T. Chiueh, "Automatic generation of string signatures for malware detection," *Signal Process.*, vol. 87, no. 12, pp. 2882–2895, 2009.

[12] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. IEEE Symp. Secur. Privacy*, May 2005, pp. 226–241.

[13] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proc. OSDI*, vol. 4. 2004, p. 4.

[14] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha, "An architecture for generating semantics-aware signatures," in *Proc. Conf. Usenix Security Symp.*, 2005, p. 7.

[15] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2010, p. 26

[16] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. USENIX Secur. Symp.*, 2008, pp. 139–154.

[17] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in Android based mobile devices," in *Proc. 8th Int. Conf. Next Generat. Mobile Apps, Services Technol.*, Sep. 2014, pp. 66–71.

[18] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. IEEE Eur. Symp. Secur. Secur. Privacy*, Mar. 2016, pp. 439–454.

[19] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing Android encrypted network traffic to identify user actions," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 1, pp. 114–125, Jan. 2016.

[20] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "AsDroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction," in *Proc. Int. Conf. Softw. Eng.*, 2014, pp. 1036–1046.

[21] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in *Proc. USENIX Conf. Secur.*, 2013, pp. 527–542.

[22] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. F. Wang, "UIPicker: User-input privacy identification in mobile applications," in *Proc. USENIX Conf. Secur. Symp.*, 2015, pp. 993–1008.

[23] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 583–595, Feb. 2016.

[24] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "ReCon: Revealing and controlling PII leaks in mobile network traffic," *CoRR*, vol. abs/1507.00255, Jul. 2015.

[25] S. Ramu, "Mobile malware evolution, detection and defense," EECE 571B, term survey paper, 2012.

[26] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf, "Mobile security catching up? Revealing the nuts and bolts of the security of mobile devices," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2011, pp. 96–111.

[27] Z. Chen, B. Yu, Y. Zhang, J. Zhang, and J. Xu, "Automatic mobile application traffic identification by convolutional neural networks," in *Proc. 15th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Aug. 2016, pp. 301–307.

[28] X. Jiang and Y. Zhou, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.

[29] S. Wang *et al.*, "TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic," in *Proc. IEEE/ACM Int. Symp. Quality Service (IWQOS)*, Jun. 2016, pp. 1–6.

[30] D. Cao *et al.*, "DroidCollector: A high performance framework for high quality Android traffic collection," in *Proc. 15th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Aug. 2016, pp. 1753–1758.

[31] *Android Monkey Tool*. [Online]. Available: http://developer.android.com/

[32] *Tshark—The Wireshark Network Analyzer 2.4.2*. Accessed: May 2017. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[33] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Proc. 14th Int. Conf. Mach. Learn.*, 1998, pp. 412–420.

[34] D. Mahato, S. Das, and D. P. Dash, "A novel architecture of I^2C slave using one-hot encoding technique," in *Proc. Int. Conf. Recent Trends Inf. Technol. Comput. Sci.*, 2011, pp. 1–5.

[35] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. China Machine Press, 2005, pp. 658–664.

[36] *VirusShare.Com*. Accessed: May 2017. [Online]. Available: https://virusshare.com/

[37] *Baidu Mobile Assistant*. Accessed: May 2017. [Online]. Available: http://shouji.baidu.com/

[38] *VirusTotal*. Accessed: May 2017. [Online]. Available: https://www.virustotal.com/

[39] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for Android malware detection with decompiled source code," *IEEE Trans. Depend. Sec. Comput.*, vol. 12, no. 4, pp. 400–412, Jul. 2015.

[40] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2014, pp. 1–15.

[41] Z. Li, L. Sun, Q. Yan, W. Srisa-An, and Z. Chen, "DroidClassifier: Efficient adaptive mining of application-layer header for classifying Android malware," in *Proc. 12th EAI Int. Conf. Secur. Commun.*, 2016, pp. 597–616.

[42] A. Razaghpanah *et al.* (2015). "Haystack: A multi-purpose mobile vantage point in user space." [Online]. Available: https://arxiv.org/abs/1510.01419

Shanshan Wang was born in China, in 1988. She received the B.S. degree from the University of Jinan, Jinan, China, in 2015, where she is currently pursuing the master's degree. Her research interests include data mining, network traffic analysis, and machine learning.

Qiben Yan (S'11–M'15) received the B.S. and M.S. degrees in electronic engineering from Fudan University, Shanghai, China, and the Ph.D. degree from the Computer Science Department, Virginia Tech. He is currently an Assistant Professor with the Department of Computer Science and Engineering, University of Nebraska-Lincoln. His current research interests include wireless communication, wireless network security and privacy, and mobile and IoT security. He is always interested in problems with potential real-world impact, and tries to bridge the gap between theory and practice.

Zhenxiang Chen received the B.S and M.S. degrees from the University of Jinan, Jinan, China, in 2001 and 2004, respectively, the Ph.D. degree from the School of Computer Science and Technology, Shandong University, Jinan, in 2008. He is currently a Professor with the School of Information Science and Engineering, University of Jinan. His research interests include network behavior analysis, mobile security and privacy, and hybrid computational intelligence. He has authored numerous papers in these areas.

Bo Yang is currently a Professor and the President of Linyi University, Linyi, China. He is also the Director of the Provincial Key Laboratory for Network-Based Intelligent Computing, and a Member of the Technical Committee of Intelligent Control of the Chinese Association of Automation. His main research interests include computer networks, artificial intelligence, machine learning, knowledge discovery, and data mining. He has authored numerous papers and received some important scientific awards in these areas.

Chuan Zhao received the Ph.D. degree in computer science and technology from Shandong University, China, in 2016. He is currently a Lecturer with the School of Information Science and Engineering, University of Jinan, China. His research interests include information security, cryptography, and privacy-preserving techniques. He is currently focusing on secure computation, privacy-preserving genomic data processing, privacy-preserving machine learning, and encrypted network traffic analysis.

Mauro Conti (SM'14) received the Ph.D. degree from the Sapienza University of Rome, Italy, in 2009. He was a Post-Doctoral Researcher with Vrije Universiteit Amsterdam, The Netherlands. He is currently an Associate Professor with the University of Padua, Italy. His main research interest is in the area of security and privacy. In this area, he has authored over 100 papers in topmost international peer-reviewed journals and conference. He is an Associate Editor for several journals, including the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He was the Program Chair of TRUST 2015 and the General Chair of SecureComm 2012 and the ACM SACMAT 2013.