Hindawi
Security and Communication Networks
Volume 2021, Article ID 5599556, 17 pages
https://doi.org/10.1155/2021/5599556

Research Article

N-Gram, Semantic-Based Neural Network for Mobile Malware Network Traffic Detection

Huiwen Bai,[1] **Guangjie Liu**,[2] **Weiwei Liu**,[1] **Yingxue Quan**,[1] **and Shuhua Huang**[1]

[1] *School of Automation, Nanjing University of Science and Technology, Nanjing, China*
[2] *School of Electronic and Information Engineering, Nanjing University of Information Science and Technology, Nanjing, China*

Correspondence should be addressed to Guangjie Liu; gjieliu@gmail.com

Received 15 January 2021; Revised 5 April 2021; Accepted 9 April 2021; Published 23 April 2021

Academic Editor: Qi Liu

Mobile malware poses a great challenge to mobile devices and mobile communication. With the explosive growth of mobile networks, it is significant to detect mobile malware for mobile security. Since most mobile malware relies on the networks to coordinate operations, steal information, or launch attacks, evading network monitor is difficult for the mobile malware. In this paper, we present an N-gram, semantic-based neural modeling method to detect the network traffic generated by the mobile malware. In the proposed scheme, we segment the network traffic into flows and extract the application layer payload from each packet. Then, the generated flow payload data are converted into the text form as the input of the proposed model. Each flow text consists of several domains with 20 words. The proposed scheme models the domain representation using convolutional neural network with multiwidth kernels from each domain. Afterward, relationships of domains are adaptively encoded in flow representation using gated recurrent network and then the classification result is obtained from an attention layer. A series of experiments have been conducted to verify the effectiveness of our proposed scheme. In addition, to compare with the state-of-the-art methods, several comparative experiments also are conducted. The experiment results depict that our proposed scheme is better in terms of accuracy.

1. Introduction

Thanks to the advance of the mobile communication networks, recent years are able to witness the opening of the mobile era. And, the smart IoT (Internet of Things) devices and smart applications may continue increasing in the 5G\6G communication networks. However, continuous improvement in mobile device hardware and mobile communication technologies has not only led to a highly interconnected world but also a world grown highly vulnerable. The explosive growth of mobile communication brings substantial burden to the mobile security management. According to a recent report [1], the number of apps in the Google Play Store has risen from 16000 in December 2009 to more than 2 million in February 2016. And, the mobile traffic amount has reached 3.7 exabytes per month in 2015. However, the increase in 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, mobile devices running with the Android system have gradually become the main target of attackers and are infected by malicious apps. This circumstance reveals the urgency of enforcing mobile communication network security.

To address the problem of the mobile malware, many researchers have to pay attention to the detection of malware apps. Evading traffic-based methods is difficult, as malware usually launches malicious behaviors through network connections, including receiving commands from servers, transferring the stolen data, and so on. In addition, detecting malware traffic has many advantages in terms of ease of deployment. Thus, detecting malware based on network traffic has the potential to greatly reduce the threat of malicious activity in the mobile communication networks. For this reason, researchers start exploring new solutions for mobile malware detection based on network traffic.

Network-traffic-based methods can be divided into three types [3], namely, network signature-based analysis, lexical-feature-based analysis, and statistical-feature-based analysis, respectively. Signature-based detection collects domains and signatures from known malware and compares them against suspicious pieces of code in order to determine whether they are malicious or benign. However, signature-based methods lack adaptability. It can detect known attacks but has limited ability to handle novel ones. The statistic-based methods detect the malware traffic by utilizing the header information of the packets, for example, IP layer information or TCP layer information. The commonly used statistical features contain packet size, duration of flow, the radio of the packets, and so on. And, these features are combined with classifiers, e.g., Random Forest, K-Nearest Neighbor, and Decision Tree. However, the performance of these methods highly depends on the manually engineered features and some private traffic information, hence dramatically limiting their accuracy and generalizability.

The lexical-based methods detect the malware traffic according to the text analysis of the network traffic. On the 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 of flows than statistical features, which greatly improves malware detection performance. However, these methods typically consider plaintext traffic such as HTTP traffic. In the recent research [3], the researchers have decrypted the encrypted TLS traffic, before classifying the traffic. In addition, manually analyzing and extracting protocol feature domains for mobile malware detection has limitations. The extracted features fail to detect mobile malware traffic when new protocols are employed by the malware. Because of the existence of various types of the network application protocols, there are no fixed domains for network traffic flows. It is difficult to manually extract meaningful semantic features for all protocols separately. As it requires a lot of manual resources, and when new protocols appear, existing semantic features failed to cover them. To address this problem, we employ the neural network model to detect mobile network traffic in an end-to-end way.

In this paper, we propose an N-gram, semantic-based neural method to detect mobile malware network traffic. In our scheme, we transfer each five-tuple-based network flow into the text file that contains several domains. Each domain in the text consists of 20 words and each word in the text is a byte content in the network flow. To analyze the network traffic semantic, the proposed neural model processes the network traffic flow representation in two stages. It first produces continuous domain vectors from word representations with domain composition. These domain representations are treated as inputs of network flow composition to get network flow representation. Network flow representations are then used as features for flow-level malware traffic detection. And, each flow has an associated label, which is required to train the algorithm as we employ supervised method.

We employ the deep learning method to process the generated texts to find the flows generated by the mobile malware, as deep learning methods can avoid manually analyzing and selecting features, and these methods have a higher learning capability compared to traditional machine learning methods like Random Forest and Support Vector Machine [4]. One of the main drivers of this work is to assess the applicability of deep learning advances to the lexical-based mobile malware detection problem. Therefore, we have studied the adequacy of different deep learning architectures. And, for the proposed model, we have conducted several experiments to find the best performance of the architecture and parameters. In addition, the size of the text also the impacts the model performance; thus, we have conducted several experiments on the text size.

This paper's contributions can be summarized as follows:

(1) We propose a novel network traffic process method that converts network flows into texts, and the words in the text consists of a byte content of the network flow payload. Each network flow is divided into several domains and each domain consists of several character-level words.

(2) We propose an N-gram semantic neural model to detect mobile malware traffic. It first produces continuous domain vectors from word representations' domain composition using convolutional neural network (CNN) model. Afterward, several domain vectors are processed as a domain representation. Then, these domain representations are treated as inputs of network flow composition to get network flow representation using gated recurrent unit (GRU). Network flow representations are then used as features for flow-level malware traffic detection.

(3) We conducted a series experiment on real-world network traffic, which contains different types of protocols to adjust the architecture and parameters so as to obtain a better performance. Extensive experimental results show that the proposed model outperforms prior state of arts by achieving 92.6% malicious flow detection rate.

The remainder of this paper is organized as follows. In the next section, we summarize the related work on the existing mobile malware detection methods. In Section 3, the proposed N-gram, semantic-based neural method is introduced. In Section 4, we firstly introduce and describe the selected dataset in our research and then a series of experiments are conducted on the datasets. Finally, in Section 5, we conclude our research and the future work.

2. Related Work

There are many proposed security mechanisms to detect mobile malware and protect users' mobile devices from attacks. Existing typical methods can be classified into 3 categories [5], namely, static-based methods, dynamic-based methods, and hybrid-based methods. In addition to the typical methods, some researchers employ network traffic to detect mobile malware.

2.1. Typical Methods. Static-based methods detect malware by extracting statistic features from APPs, which unpack or disassemble them without running the APPs [6–10]. These statistic features can be extracted from application permission list [11, 12], sensitive Application Programming Interface (API) calling or critical code segments in the source code [13–15]. Static-based methods are widely used for vetting apps. However, there exist several key challenges that static-based methods are facing, for example, the static-based methods cannot detect certain source code tampering operations.

Dynamic-based methods examine the features of APPs, which are running, and the features of the execution behaviors. In these methods [16–20], the information, namely, memory utilization, system calls, network connections, and battery power, is employed to detect the mobile malware. However, there still remain some issues in dynamic-based methods. On the one hand, these methods cannot fully traverse the execution path of the software. And, on the other hand, these methods cannot detect certain malicious behaviors if an app is protected by runtime security mechanisms (e.g., DexGuard).

In order to avoid the limitations of both static-based and dynamic-based methods, hybrid-based methods, the combination of both the mechanisms, are employed to detect the mobile malware. These methods have a two-step process, wherein initially static analysis is performed before the dynamic one [21–23]. However, the combination of the two methods consumes more resources and results in less improvement.

2.2. Network-Traffic-Based Methods. To address the limitation of the typical methods, the researchers have investigated the malware identification and private information tracking extensively. The researchers begin to analyze and identify malicious apps using network traffic, as almost all the attackers use mobile networks to obtain sensitive information of the user or interact with its malicious APPs. Network-behavior-based method can be divided into three categories [3], namely, network signature-based methods, lexical-feature-based methods and statistical-feature-based methods, respectively.

The signature-based methods detect the malware traffic according to the predetermined malware signatures. Griffin et al. [24] 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 [25–27]. Most of these studies focused on worm fingerprinting. Perdisci et al. [28] 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. They can detect known attacks but have limited ability to handle novel ones.

The statistic-based methods detect the malware traffic according to the header information of the packets instead of the payload information. Aresu et al. [29] showed how it is possible to group mobile botnets families by analyzing the HTTP traffic they generate. The authors create malware clusters by looking at specific statistical information that is related to the HTTP traffic. This approach also allows to extract signatures with which it is possible to precisely detect new malware that belong to the clustered families. Lashkar et al. [30] proposed a detection and characterization system for detecting meaningful deviations in the network behavior of a smart-phone application with 9 traffic statistic-feature measurements. The authors employed five classifiers to verify the performance of these features, namely, Random Forest, K-Nearest Neighbor, Decision Tree, Random Tree, and Regression. Arora et al. [31] 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 13 other features.

In addition, there exist several researchers that employ deep-learning-based methods to detect mobile malware network traffic. Bendiab et al. [32] proposed a novel IoT malware traffic analysis approach using deep learning and visual representation for faster detection and classification of new malware (zero-day malware). The detection of malicious network traffic in the proposed approach works at the package level, reducing significantly the time of detection with promising results due to the deep learning technologies used. Feng et al. [33] proposed a two-layer method to detect malware in Android APPs. The first layer is permission, intent, and component-information-based static malware detection model. In the second layer, a new method CACNN, which cascades CNN and AutoEncoder, is used to detect malware through network traffic features of APPs.

The lexical-based methods detect the malware traffic according to the text analysis of the traffic. Android [34] 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 [35] used NLP techniques to identify sentences that described the need for a given permission in the app description. Nan et al. [36] 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 [37]. Ren et al. [38] 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. Wang et al. [3] proposed an automatic malware detection method using the text semantics of network traffic. In particular, the authors considered each HTTP flow generated by mobile apps as a text document, which can be processed by natural language processing to extract text-level features. The text semantic features of network traffic were utilized to develop a malware detection model. In addition, the authors designed 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.

Through literature research, using network traffic and lexical analysis to discover hidden malware is promising. In this research, our scheme utilizes deep learning algorithm to extract traffic lexical to discover malicious behaviors in mobile network traffic. Dee-learning-based methods are expected to learn from highly complicated domains to gain higher accuracy than machine-learning-based methods with more functionality. In addition, we process network flows into character-level text files. Each word in the text files is a byte content in the network flows. We examine the performance by a published dataset that contains the network traffic consisting of multitypes of protocols, including both encrypted and nonencrypted network traffic.

3. Proposed Approach

The following sections present the proposed mobile malware network traffic detection neural model, which computes continuous vector representations for network flows of variable types of protocols. An overview of the approach is displayed in Figure 1. The proposed approach processes the network traffic flow representation in two stages. It first produces continuous domain vectors from word representations with domain composition. These domain representations are treated as inputs of flow composition to get flow representation. The flow representations are then used as features for flow-level traffic classification.

3.1. Network Traffic Semantic Analysis.

3.1. Network Traffic Semantic Analysis. Network traffic is the data transmitted over networks, using a series of network protocols, e.g., IP, TCP, and HTTP. The network protocols are a set of rules that must be observed when exchanging information between peer entities that communicate with each other in the networks. Just as languages are employed by humans to communicate, there are also languages between the devices in networks, namely, the network protocols. The network protocol is composed of three elements: semantics, syntax, and temporal. Different network devices must use the same network protocol to communicate with each other. However, as the large number of protocols in networks, it is difficult to automatically analyze and extract the key information from the network traffic.

For a section of network traffic, it can be regarded as a domain set S, as shown in Figure 2. Decompose S into domains $S_1, S_2, S_3, \ldots, S_n$. Let the total length of S be L, for a given domain S_i, as long as the length of each segment L_i is being determined; the network traffic can be described segment by segment according to the value of L_i. The length of L_i is not always fixed. In addition, depending on the value of S_i, the optional domain S_{i-j} may appear. Therefore, the main problem of network traffic description is the representation problem of the variable length of domains and the branching problem of optional domains.

In the following sections, we introduce the proposed neural model to model a network flow. We process the network flows as character-level texts, namely, each word in the texts is a one-byte character. Each text document of flow is divided into n_d domains, and each domain consists of n_w words. According to Figure 2, the length of the fields in the protocols is not fixed. Therefore, we employ multiwidth kernel CNN to learn the domain representations, which can represent the fields of different lengths in the domains. These domain representations are then considered as input of the flow composition using GRU, and finally the classification of flows is output through an attention layer.

3.2. Domain Composition. Before presenting a CNN with multiple kernels for domain composition, each word is represented as a low dimensional, continuous, and real-valued vector, namely, word embedding. All the word vectors are stacked in a word embedding matrix $L_w \in \mathbb{R}^{d\times|V|}$, where d denotes the dimension of word vector and $|V|$ is vocabulary size. These word vectors can be randomly initialized from a uniform distribution.

We utilize CNN to compute domain representations with a semantic composition. CNN is able to learn fixed-length vectors for domains, capture the word order in a domain, and does not depend on external dependency or constituency parse results. Therefore, we divide each text into n_d domains, and each domain contains n_w words. Specifically, we use CNN with multiple convolutional kernels of different widths to produce multi N-gram domain representations to adapt to different length value of fields in protocols. Figure 3 displays the domain composition method.

The convolutional kernels of different sizes can capture different N-gram semantics. For example, a convolutional kernel with a width of 2 essentially captures the semantics of bigram fields in a domain. A domain consisting of n_d (padded where necessary) words is denoted as

$$c_{1:n_d} = c_1 \oplus c_2 \oplus \cdots \oplus c_{n_d}, \tag{1}$$

where $\oplus$ is the concatenation operator. Each word c_i is mapped to its embedding representation $e_i \in \mathbb{R}^d$. The input of a linear layer is the concatenation of word embeddings in a fixed-length window size l_c, which is denoted as

$$I_c = \left[e_i; e_{i+1}; \ldots; e_{i+l_c-1}\right] \in \mathbb{R}^{d\cdot l_c}, \tag{2}$$

where l_c be the width of a convolutional kernel. For the multiwidth kernel convolution, given a collection of convolution kernel sizes $K = \left\{k_1, k_2, \ldots, k_{n_k}\right\}$, after the convolution operation with the i^{th} kernel k_i is applied, the output of each kernel is as follows:

$$O_j^{k_i} = W_j^{k_i} \cdot I_{k_i} + b_{k_i}, \tag{3}$$

where $W_j^{k_i} \in \mathbb{R}^{l_{oc}\times d\cdot l_c}$, $b_{k_i} \in \mathbb{R}^{l_{oc}\times d\cdot l_c}$, l_{oc} is the output length of the linear layer. To capture global semantics of a domain, we feed the outputs of linear layers to an average pooling layer,

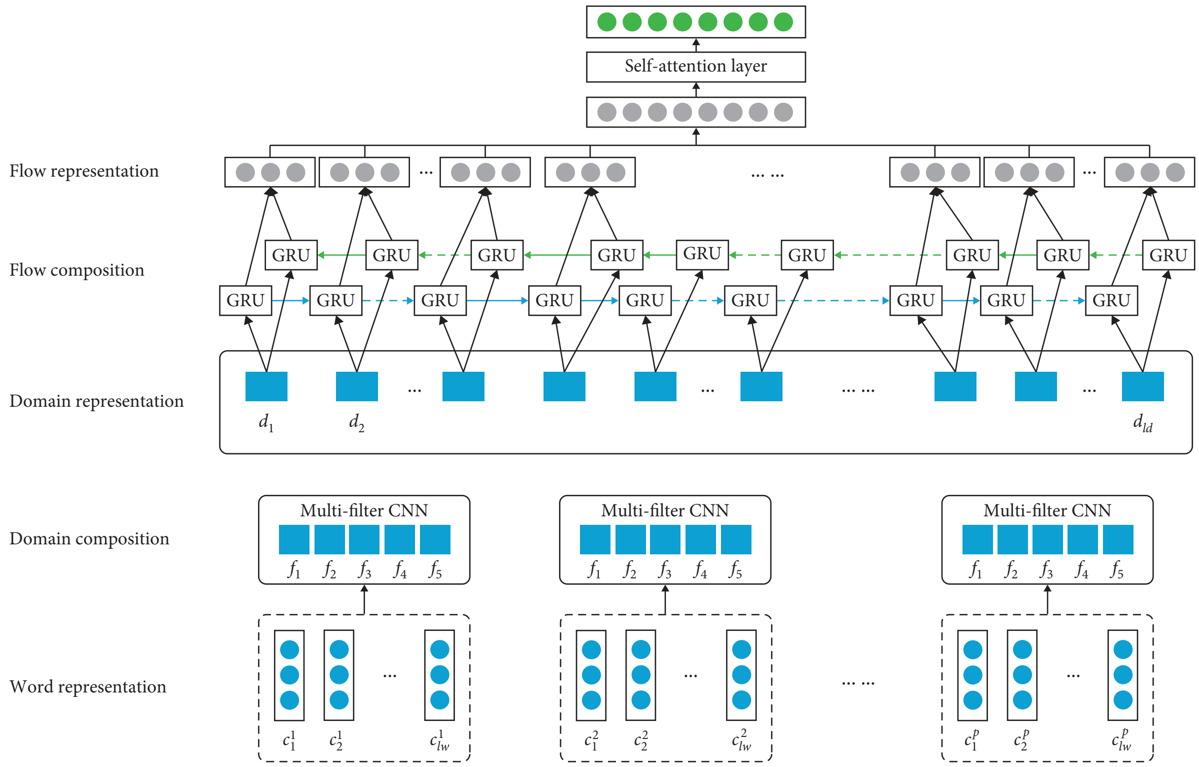
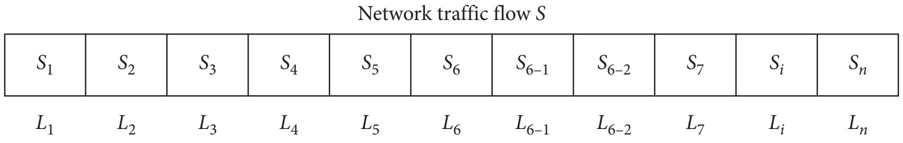


Figure 1: The description of our proposed N-gram, semantic-based neural method to detect mobile malware network traffic.


Network traffic flow S

S_1	S_2	S_3	S_4	S_5	S_6	S_{6-1}	S_{6-2}	S_7	S_i	S_n
L_1	L_2	L_3	L_4	L_5	L_6	L_{6-1}	L_{6-2}	L_7	L_i	L_n

Figure 2: Examples of message structure features. (1) The length S_4 is determined by S_3. (2) Depending on the value of S_6, optional field $S_{6\text{-}1}$ or $S_{6\text{-}2}$ may appear.

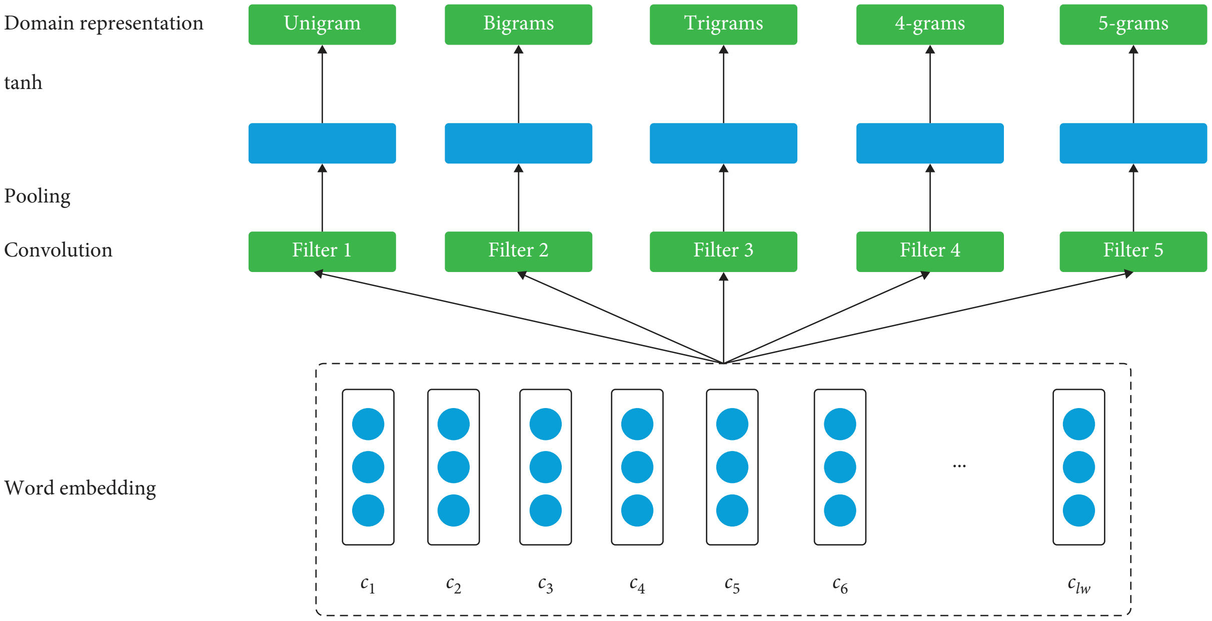


Figure 3: Patter composition with multikernel convolutional neural network. *kernel i* represents the kernel with width of *i*.

resulting in an output vector with fixed-length. We further add hyperbolic tangent (tanh) to incorporate pointwise nonlinearity. The output of each convolution operation becomes

$$O^{k_i} = \left[o_1^{k_i}, o_2^{k_i}, \ldots, o_N^{k_i}\right], \tag{4}$$

where N is the number of ci convolution kernel. The set of vector outputs by the multikernel CNN is denoted as

$$O = \left[O^{k_1}, O^{k_2}, \ldots, O^{k_{n_k}}\right], \tag{5}$$

where O is the domain representation and it is the input of the flow composition. In this work, we use five convolutional kernels whose widths are 1, 2, 3, 4, and 5 to encode the semantics of unigrams, bigrams, trigrams, 4-grams, and 5-grams in a domain. Each kernel consists of a list of linear layers with shared parameters.

3.3. Flow Composition.

3.3. Flow Composition. The obtained domain vectors are fed to a flow composition component to calculate the flow representation. We utilize a GRU approach for flow composition in this part. GRU is a variant of Recurrent Neural Network (RNN); it is simpler than Long Short-Term Memory (LSTM) RNN. Since GRU is a variant of LSTM, it can also solve the problem of long dependence in the RNN network. As displayed in Figure 4, a common GRU unit consists of an update gate (z_t) and a reset gate (r_t).

Specifically, the transition function of the GRU is calculated as follows:

$$\begin{aligned}
\widetilde{h}_j^t &= \left(1 - z_j^t\right) h_j^{t-1} + z_j^t \widetilde{h}_j^t, \\
\mathrm{net}_z^t &= x^t W^z + h^{t-1} U^z, \\
z_j^t &= \sigma\left(\mathrm{net}_z^t\right)^j, \\
\mathrm{net}_{\widetilde{h}}^t &= x^t W + \left(r^t \circ h^{t-1}\right) U, \\
\widetilde{h}_j^t &= \tanh\left(\mathrm{net}_{\widetilde{h}}^t\right)^j, \\
\mathrm{net}_r^t &= x^t W^r + h^{t-1} U^r, \\
r_j^t &= \sigma\left(\mathrm{net}_r^t\right)^j, \\
\mathrm{net}_y^t &= h^t W^y, \\
y_j^t &= \sigma\left(\mathrm{net}_y^t\right)^j,
\end{aligned} \tag{6}$$

where σ is the sigmoid function, tanh is the tangent function, x^t is one of the input vectors, and $\circ$ represents the product, namely, the product of corresponding elements. The subscript j represents the index of the node and the superscript t represents the time. $W^y \in R^{hd \times yd}$ represents the parameter matrix from the hidden layer to the output layer, hd and yd are the number of nodes in the hidden layer and output layer, respectively. $W^z \in R^{xd \times hd}$ and $U^z \in R^{hd \times hd}$ represent the connection matrix between the input and the last hidden layer to update gate (z_t), respectively, and xd represents the dimension of the input data. $W^r \in R^{xd \times hd}$ and $U^r \in R^{hd \times hd}$ represent the connection matrix between the input and the previous hidden layer to reset gate (r_t), respectively. $W \in R^{zd \times hd}$ and $U \in R^{hd \times hd}$ represent the connection matrix between the input and the hidden layer to the selected state $(\widetilde{h}_t)$ at the previous time, respectively.

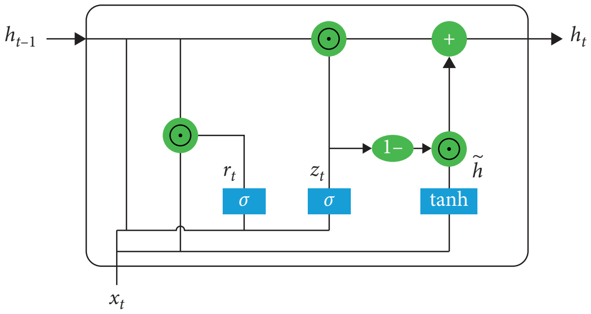


Figure 4: GRU unit architecture.

In this research, since we prefer not to discard any part of the domain semantics to get a better flow representation, the GRU units in the model are always on. Given the domain vectors as input, flow composition produces a fixed-length flow vector as output.

3.4. Traffic Classification Based on Semantic.

3.4. Traffic Classification Based on Semantic. The composed flow representations can be naturally regarded as semantic features for flow for classification without feature engineering. In addition, we add a self-attention layer for the flow representations to improve the classifying performance. The self-attention layer architecture is shown in Figure 5.

The degree of influence between the output information obtained by the GRU at each time point is the same. In order to highlight the importance of some output results to classification, the attention of weighting is employed. The attention mechanism has been widely used in various NLP tasks for the past few years. The attention mechanism is essentially a weighted sum. The set of vectors input by the GRU layer is expressed as $[h_1, h_2, \ldots, h_t]$. The attention process can be described as

$$\begin{aligned}
e_t &= \sigma\left(W_a \cdot h_t + b_a\right), \\
a_t &= \mathrm{softmax}\left(e_t\right), \\
m_t &= a_t \cdot h_t, \\
p &= \sum_{i=1}^{n} m_i,
\end{aligned} \tag{7}$$

where y_t is the output of the GRU model. And, W_a represent the weighted matrix during the training process, b_a represent the biases. The a_t means the attention matrix, and p is the output of the attention layer.

Finally, we add a linear layer to transform flow vector to real-valued vector whose length is class number C. Afterward, we add a softmax layer to convert real values to conditional probabilities, which is calculated as follows:

$$P_i = \frac{\exp\left(y_i\right)}{\sum_{i'=1}^{C} \exp\left(y_{i'}\right)}. \tag{8}$$

We conduct experiments in a supervised learning method, where each network flow in the training data is accompanied with its label.

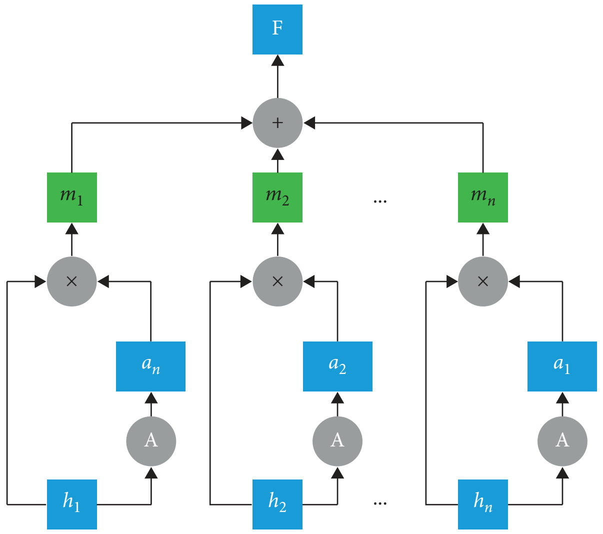


Figure 5: Self-attention layer architecture.

4. Experimental Evaluation

In this section, a series of experiments are conducted and the corresponding results are analyzed in detail. In the beginning, we introduce the experiment environment, including the selected dataset, data processing, and the basic evaluating metrics scheme. Then, we compare the model classification performance using different components in the proposed semantic neural model. In addition, a comparative experiment to verify malware detection performance is conducted between the proposed scheme and several influential algorithms proposed in recent years.

4.1. Selected Dataset and Data Processing

4.1.1. Selected Dataset. For this work, we have made use of real data from CICAndMal2017 [15] datasets, which collect 4,354 malware and 6,500 benign apps from VirusTotal [39], Contagio security blog [40], and previous researchers [15, 41, 42]. The collected network traffic data are described in Table 1. Benign apps were collected from Google play market published in 2015, 2016, and 2017. These APPs were collected based on their popularity and identified based on the detection results from VirusTotal [39]. Only those APPs that VirusTotal determined as benign are included to the benign APP set. The malware network traffic consists of 4 categories, namely, adware, ransomware, scareware, and SMS malware. Each category has different malware family. Adware has Dowgin, Ewind, Feiwo, Gooligan, Kemoge, koodous, Mobidash, Selfmite, Shuanet, and Youmi family. Ransomware has Charger, Jisut, Koler, LockerPin, Simplocker, Pletor, PornDroid, RansomBO, Svpeng, and WannaLocker family. Scareware has AndroidDefender, AndroidSpy.277, AV for Android, AVpass, FakeApp, FakeApp.AL, FakeAV, FakeJobOffer, FakeTaoBao, Penetho, and VirusShield family. SMSmalware has BeanBot, Biige, FakeInst, FakeMart, FakeNotify, Jifake, Mazarbot, Nandrobox, Plankton, SMSsniffer, and Zsone family.

Table 1: Description of CICAndMal2017 dataset.

Items	APP numbers	Composition	Collected traffic data	
Benign	5065	2015	19.0G	6.0G
		2016		6.8G
		2017		5.9G
Malware	429	Adware	19.2G	8.4G
		Ransomware		3.4G
		Scareware		5.1G
		SMS malware		2.3G

Because of the sample errors and the problem of inconsistent labelling in different datasets, CICAndMal2017 retains 5000 (malware 429 and benign 5,065). In addition, as most of the advanced malware employs the evasion or transformation technique to evade detection (code permutation, register renaming, idle activation) [21], the malware behaviors are triggered only after connecting network update, or over time after the restart process. In order to trigger the malware network behaviors, the network traffic data capture occurs in 3 stages, namely, 3 minutes after the app is installed, 15 minutes before, and after restarting the smartphone.

4.1.2. Data Processing. As mobile software commonly employs the TCP and UDP protocol to execute the network activity, we handle TCP and UDP connection as the main interaction granularity between the apps and the networks. In this research, we have designed a tool with the use of C++ programme to process the network traffic, called PKTPT (packet processing tool), and the PKTPT architecture is shown in Figure 6. The network traffic data processing contains 3 stages, namely, traffic segmenting, data generation (including data extracting, trimming, and padding), and data transformation.

Traffic Segmenting. Traffic segmenting stage processes the network traffic into flows based on the 5-tuple (protocol, source IP, source port, destination IP, destination port). In this research, we use bidirectional flows, which have the same IP address and port. Further details of traffic segmenting are explained in Algorithm 1. PKTPT first separates packets in the specific packet queue according to source address, destination address, source port, destination port, and transport layer protocol. Then, each packet is taken out in order, and according to the 5-tuple, the same flow packets are processed as the same flow node in the flow table, which is implemented using hash_map (a data struct in the C++ program).

Data Generation. In the previous stage, the network traffic is processed into TCP and UDP flows. In this stage, the packet in the flow is processed. Each flow consists of several frames of packets, and each frame is generally encapsulated in a fixed structure: Ethernet II header, Internet Protocol header, Transport header, and application payload. In our framework, we only use the application payload of the packets, as

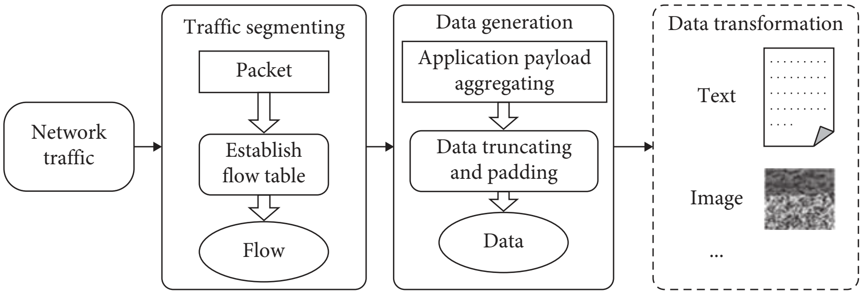


Figure 6: Description of data processing.

```
WHILE (packet reading a packet)
  IF (hash_search(packet 5-tuple information m_fe) = = FALSE)
    FUNC ( ) {
      creat flow: Flow New
      establish the hash mapping of flow: HM(m_fe)
      put the flow into hash_table
    }
  ELSE
    The packet belongs to Flow A
    IF (the packet arrival time is out of the set timeout value)
      Delete Flow A
      FUNC( )
    END IF
  END IF
  flow ← packet
END WHILE
```

Algorithm 1: Traffic segmenting.

the application payload is the data that interact with each other on both sides of the communication, and the header data of each layer is used to establish communication channels, such as host IP addresses and ports. Truncation and zero-padding are necessary as different flows have different sizes of application payload. In this research, we extract the first *n* byte data in each flow and save them into a csv file, using spaces to separate byte data. We also take the flow data size as a parameter of the impactor of performance for deep learning model.

Data Transformation. In this research, we employ text-processing method to process the network traffic. Indeed, packet contents with different protocols can be treated as texts following different grammars. Due to different protocol structures, there are different domains in packets. We employ character-level word as vocabulary in text processing and establish models to learn the semantic relationships between them and protocol structures of a specific protocol. Then, the established models extract domain and flow feature according to the semantic to classify the malware and benign traffic.

Therefore, we transform the flow data obtained in the previous stage into texts. Each flow data is processed as a text file consisting of several words. We split the words using space, namely, the words in the text files is the character level. Then, the texts are fed into a deep learning model to be classified. In addition, some existing work converts traffic to images. In order to compare the effectiveness of the two processing methods, we have also developed the function to convert traffic to images in the stage of data transformation.

4.2. Experiment Environment and Evaluation Metrics. Our experiments are executed on Ubuntu 16.04 LTS with 64 GB of RAM and one GPU card (NVIDIA GTX 1080Ti 11 GB). For the experimental implementation, we used Tensorflow-gpu 1.12.0 and Keras 2.2.4 operated with Python 3.6.10. The deep learning models are constructed, trained, and tested by Keras using the Tensorflow-gpu backend.

We separate the flow-based dataset into learning and test data. Next, the learning data are separated into training and validation data. And, the ratio of training, validation, and test data is 8 : 1 : 1. Then, the model is trained using the training and validation data, and the model performance is measured using the test data. The description of dataset is shown in Table 2.

In our cases, we employ the mini-batch gradient decent method in the training process. In this method, the data are divided into several batches and the parameters are updated according to the batches. In this process, there are two parameters, namely, batch size and iterations, which represent the number of samples for each training and the number of iterations for completing a total sample training,

Table 2: Description of dataset for binary-classification experiment.

		Training	Validation	Test	Total
Number of samples	Benign	165306	20663	20663	206632
	Malware	144088	18011	18011	180110
	Total	309394	38674	38674	386742

respectively. A single training iteration for all batches propagated forward and backward can be denoted as an epoch. The number of learning epochs is set to 200, and the learning rate of optimizers (i.e., rmsprop and Adam) is set to 0.001.

To evaluate the detection effectiveness of the proposed scheme, the following terms are used for determining the quality of the classification models:

Accuracy: it estimates the ratio of the correctly recognized network traffic flows to the entire test dataset.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{9}$$

where TP represents True Positive, TN represents True Negative, FP represents False Positive, and FN represents False Negative.

Precision: it estimates the ratio of the correctly identified malware traffic flows to the total number of samples classified to malware class. It is denoted as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{10}$$

Recall: it estimates the ratio of the correctly identified malware traffic flows to the number of all malware traffic flows. It is denoted as

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{11}$$

F1-score: it is the harmonic mean of Precision and Recall. It is denoted as

$$F1 - \text{score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \tag{12}$$

4.3. Comparative Experiments of Different Model Architectures. As the proposed model is an N-gram, semantic-based neural model, we conduct several experiments on the main components, namely, domain composition and flow composition. In addition, to avoid the impact of the flow payload length, we select 6 types of lengths of flow payload for each experiment.

4.3.1. Domain Composition. We first analyze the performance of the mobile malware detection using different domain composition architectures of the proposed model. For domain composition, we conduct an experiment to compare the impact of the single width kernels and the multiwidth kernels, and for the single width kernels, we select 5 types of widths, namely, 1, 2, 3, 4, and 5. For the flow composition, we employ the same architecture, namely, GRU to learn the flow representation and employ an attention layer to classify the flows at last.

The detailed description of the different domain composition architectures is given in Table 3.

In Figure 7, we provide the mobile malware detecting accuracy with different domain composition architectures in the case of different text sizes. And, the proposed multiwidth kernel gives the best results for accuracy for the mobile malware traffic detection. The accuracy of the proposed model for malware traffic detection can reach 0.924 at a text length of 1000, which is better than the single-width kernels.

In addition, we provide a detailed description of classification performance metrics of different domain composition architectures in Table 4.

In the network flows with various types of protocols, the domain consists of several characters, which has an indeterminate amount. Even in a single-protocol flow, different domains can have different amounts of characters. Therefore, the single-type CNN kernels fail to represent all of these cases. The character-level domain can obtain an accuracy over 0.90, as all the domains can be split into the set of characters. However, character-level domains reduce the relationship between the multicharacter domains. Multitype kernel CNN is able to represent the domains better. With multiwidth kernel CNN, we can obtain various field representations carried by different N-gram representations, which can avoid the deficiency of single-width CNN. Type 6 and Type 7 have an accuracy of 0.906 and 0.901, respectively. However, the accuracy of these two multikernel CNNs is still lower than that of the Type 8 and Type 9, as the latter two types represent domains more comprehensively. The accuracy of the two types of CNN is basically the same, namely, 0.926. A kernel of width 6 in Type 9 is already represented by the previous five types of kernels and does not affect the causal relationship between domains. To obtain a better classification performance, we add weights for each multikernel representation. And, the parameters are updated by feedback, and the model is able to obtain a better domain expression by adjusting the weight.

4.3.2. Flow Composition. In this section, we analyze the performance of the mobile malware detection using different flow composition architectures of the proposed model. For flow composition, we conduct an experiment to compare the impact of different deep learning architectures, namely, one-layer GRU, tow-layer GRU, one-layer LSTM, and two-layer LSTM. For domain representation, we selected the multikernel CNN with 5 different kernels according to Section 4.3.1.

The detailed description of the different flow composition architectures is provided in Table 5.

In Figure 8, we provide the mobile malware detecting accuracy with different flow composition architectures in case of different text sizes. The accuracies of the four models for malware traffic detection are similar, which are all more than 0.92. Both LSTM and GRU architectures are able to

Table 3: Description of the different domain composition architectures.

Domain composition number	Domain representation architecture description in detail
1	Unigram CNN kernel with width of 1
2	Bigram CNN kernel with width of 2
3	Trigram CNN kernel with width of 3
4	4-gram CNN kernel with width of 4
5	5-gram CNN kernel with width of 5
6	Multi N-gram CNN kernels with widths of 1, 3, and 5
7	Multi N-gram CNN kernels with widths of 2 and 4
8	Multi N-gram CNN kernels with widths of 1, 2, 3, 4, and 5
9	Multi N-gram CNN kernels with widths of 1, 2, 3, 4, 5, and 6

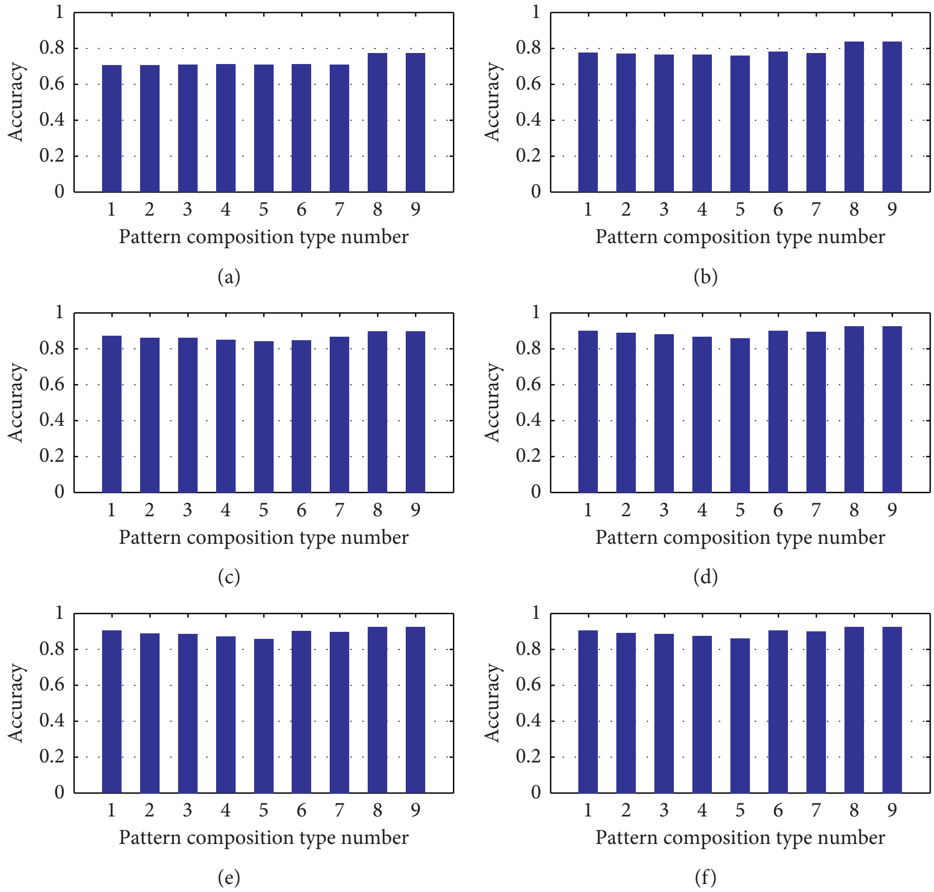


Figure 7: Accuracy of different deep learning architectures in case of different text sizes. (a) Text size of 200. (b) Text size of 500. (c) Text size of 700. (d) Text size of 1000. (e) Text size of 1200. (f) Text size of 1500.

Table 4: Description of detecting performance metrics of different domain composition architectures with different text sizes.

Domain composition type number	Evaluation metrics	Text size (byte) of the transformed data from network flow					
		200	500	700	1000	1200	1500
1	Precision	0.775	0.828	0.935	0.942	0.945	0.950
	Recall	0.657	0.724	0.820	0.863	0.863	0.864
	F1-score	0.711	0.773	0.873	0.900	0.902	0.905
	Accuracy	0.706	0.773	0.874	0.903	0.905	0.907
2	Precision	0.773	0.821	0.924	0.929	0.930	0.933
	Recall	0.657	0.721	0.810	0.850	0.849	0.854
	F1-score	0.710	0.768	0.863	0.888	0.888	0.891
	Accuracy	0.707	0.769	0.863	0.890	0.891	0.894

Table 4: Continued.

Domain composition type number	Evaluation metrics	Text size (byte) of the transformed data from network flow					
		200	500	700	1000	1200	1500
3	Precision	0.778	0.818	0.918	0.920	0.924	0.926
	Recall	0.659	0.720	0.805	0.843	0.844	0.847
	F1-score	0.714	0.765	0.858	0.880	0.883	0.885
	Accuracy	0.709	0.766	0.858	0.883	0.886	0.887
4	Precision	0.777	0.822	0.912	0.906	0.912	0.912
	Recall	0.661	0.720	0.798	0.831	0.832	0.836
	F1-score	0.714	0.714	0.851	0.867	0.870	0.872
	Accuracy	0.712	0.762	0.851	0.870	0.873	0.875
5	Precision	0.770	0.809	0.902	0.894	0.859	0.896
	Recall	0.660	0.712	0.791	0.820	0.821	0.823
	F1-score	0.711	0.757	0.843	0.855	0.856	0.858
	Accuracy	0.708	0.758	0.843	0.859	0.860	0.862
6	Precision	0.784	0.836	0.900	0.946	0.944	0.949
	Recall	0.662	0.732	0.799	0.862	0.864	0.863
	F1-score	0.718	0.781	0.847	0.902	0.902	0.904
	Accuracy	0.713	0.781	0.848	0.904	0.905	0.906
7	Precision	0.775	0.822	0.926	0.931	0.934	0.936
	Recall	0.659	0.720	0.815	0.861	0.862	0.863
	F1-score	0.712	0.768	0.867	0.895	0.896	0.898
	Accuracy	0.709	0.774	0.868	0.898	0.900	0.901
8	Precision	0.847	0.894	0.954	0.969	0.971	0.971
	Recall	0.715	0.784	0.848	0.880	0.881	0.881
	F1-score	0.775	0.836	0.898	0.922	0.924	0.924
	Accuracy	0.772	0.836	0.899	0.924	0.925	0.926
9	Precision	0.848	0.896	0.954	0.969	0.971	0.971
	Recall	0.716	0.784	0.849	0.880	0.881	0.882
	F1-score	0.776	0.836	0.899	0.923	0.924	0.924
	Accuracy	0.772	0.837	0.899	0.924	0.925	0.926

process sequential information, i.e. the previous input is related to the subsequent input. And, they can solve the problem of the disappearance of the governor in the long sequence of recursive neural network. The detailed description of classification performance metrics is depicted in Table 6, and it can be seen that increasing the LSTM layers or GRU layers does not improve detection performance. However, increasing LSTM layers or GRU layers can increase the model complexity and the time consumed of the training process. And, since the detection effect of LSTM and GRU is similar, while the training time of GRU is shorter than that of LSTM, we opted to use one-layer GRU as the flow composition in our model.

4.4. Comparative Experiments

4.4.1. Different Models. In this section, we analyze the performance of the mobile malware detection in different cases, namely, the deep learning architectures and different text sizes. To verify the effectiveness of the proposed scheme, we have implemented different deep learning architecture models to detect malware traffic. In order to find the most suitable architecture model, a variety of deep learning architectures have been built for comparison purposes, namely, MLP, TextCNN, Bi-GRU, Bi-LSTM, and the proposed model. The detailed description of the different architecture models is provided in Table 7.

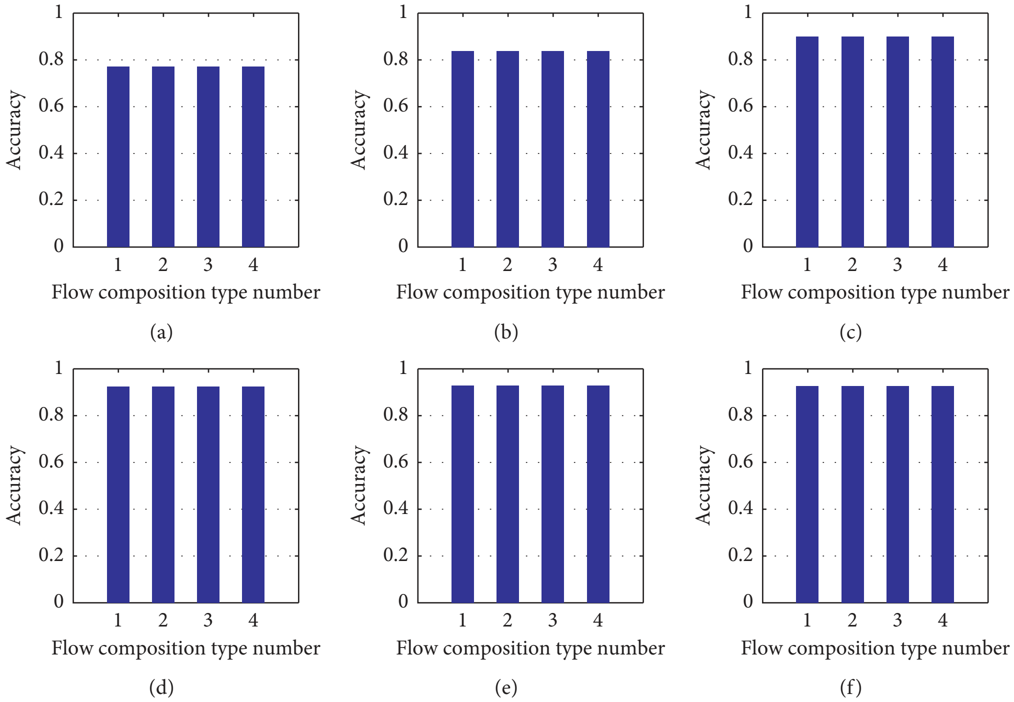
For Table 2, there are several supplemental instructions as follows: (1) Embedding (input_dimension, output_dimension) converts a positive integer (index value) to a dense vector of fixed size. (2) FC(n) denotes the fully connected layer with n output dimensions. (3) Conv1D(kernels, kernel_size) denotes the convolutional layer, where the first parameter denotes the number of the kernels and the second parameter represents the kernel size. In TextCNN architecture, the second layer contains 4 kinds of convolution kernels with different kernel sizes. (4) LSTM(n) denotes the LSTM layer and n denotes the output dimensions. Bi-LSTM stands for the Bidirectional LSTM, and its parameters are the same with LSTM. GRU has the same usage with LSTM.

We have provided the detailed description of classification performance metrics of different deep learning architectures with different text sizes in Table 8.

The MLP architecture consists of 3 fully connected layers that have the lowest classification performance in all the text sizes. The second type of deep learning architecture, TextCNN, has a convolutional layer that consists of 4 types of kernels with different kernel sizes. It has an improvement to MLP in detecting the malware traffic. However, compared to the other three types of deep learning architectures, the classification performance of TextCNN is slightly weaker.

Table 5: Description of the different flow composition architectures.

Flow composition type number	Flow representation architecture description in detail
1	One-layer GRU
2	Two-layer GRU
3	One-layer LSTM
4	Two-layer LSTM.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 8: Accuracy of different flow composition architectures in case of different text sizes. (a) Text size of 200. (b) Text size of 500. (c) Text size of 700. (d) Text size of 1000. (e) Text size of 1200. (f) Text size of 1500.

Table 6: Description of detecting performance metrics of different flow composition architectures with different text sizes.

Flow composition type number	Evaluation metrics	Text size (byte) of the transformed data from network flow					
		200	500	700	1000	1200	1500
1	Precision	0.847	0.894	0.954	0.969	0.971	0.971
	Recall	0.715	0.784	0.848	0.880	0.881	0.881
	F1-score	0.775	0.836	0.898	0.922	0.924	0.924
	Accuracy	0.772	0.836	0.899	0.924	0.925	0.926
2	Precision	0.846	0.894	0.955	0.969	0.971	0.971
	Recall	0.716	0.785	0.848	0.880	0.881	0.881
	F1-score	0.775	0.836	0.898	0.922	0.924	0.924
	Accuracy	0.772	0.836	0.899	0.924	0.925	0.926
3	Precision	0.848	0.895	0.954	0.970	0.971	0.971
	Recall	0.715	0.784	0.848	0.879	0.881	0.880
	F1-score	0.776	0.836	0.898	0.922	0.924	0.923
	Accuracy	0.772	0.836	0.899	0.924	0.925	0.925
4	Precision	0.846	0.894	0.956	0.970	0.971	0.971
	Recall	0.716	0.785	0.849	0.880	0.881	0.882
	F1-score	0.776	0.836	0.899	0.923	0.924	0.925
	Accuracy	0.772	0.837	0.899	0.925	0.926	0.926

Table 7: Description of deep learning architecture models to detect mobile malware traffic.

Deep learning type number	Deep learning architecture	Architecture detail description
1	MLP	Embedding(10000,64)-FC (1024)-FC(512)-FC(64)-FC(1)
2	TextCNN	Embedding(10000,64)-{Conv1D(100, kernel_size)-Maxpooling()-Flatten()}(kernel_size = 3, 4, 5)-Dropout(0.5)-FC(32)-FC(1)
3	Bi-LSTM	Embedding(10000,64)-LSTM(128)-LSTM128)-FC(1)
4	Bi-GRU	Embedding(10000,64)-GRU(128)-GRU(128)-FC(1)
5	Proposed model	Embedding(10000,64)- {Conv1D(100, kernel_size)-Maxpooling()}(kernel_size = 1, 2, 3, 4, 5)-GRU(100)-Attention()-FC(1)

Table 8: Description of detecting performance metrics of different deep learning architectures with different text sizes.

Deep learning type number	Evaluation metrics	Text size (byte) of the transformed data from network flow					
		200	500	700	1000	1200	1500
1	Precision	0.671	0.672	0.682	0.713	0.718	0.722
	Recall	0.584	0.594	0.599	0.614	0.617	0.621
	F1-score	0.625	0.631	0.638	0.660	0.664	0.667
	Accuracy	0.625	0.633	0.640	0.658	0.661	0.665
2	Precision	0.711	0.761	0.827	0.843	0.869	0.878
	Recall	0.624	0.669	0.694	0.727	0.749	0.757
	F1-score	0.665	0.712	0.755	0.781	0.802	0.813
	Accuracy	0.666	0.713	0.749	0.779	0.800	0.812
3	Precision	0.768	0.817	0.883	0.916	0.914	0.936
	Recall	0.651	0.716	0.779	0.837	0.848	0.859
	F1-score	0.705	0.763	0.828	0.874	0.879	0.896
	Accuracy	0.701	0.764	0.829	0.878	0.883	0.899
4	Precision	0.767	0.816	0.883	0.915	0.908	0.934
	Recall	0.651	0.718	0.777	0.833	0.847	0.853
	F1-score	0.704	0.762	0.826	0.872	0.876	0.892
	Accuracy	0.700	0.765	0.827	0.875	0.881	0.895
5	Precision	0.847	0.894	0.954	0.969	0.971	0.971
	Recall	0.715	0.784	0.848	0.880	0.881	0.881
	F1-score	0.775	0.836	0.898	0.922	0.924	0.924
	Accuracy	0.772	0.836	0.899	0.924	0.925	0.926

The reason may be that these kinds of architecture are different in the way of learning features. Both can learn the characteristics of malware and benign software. The 1st and 2nd types use CNN structure to build the model, while the 3rd and 4th types use RNN structure to build the model. Since RNNs are more advantageous in time series, Types 3 and 4 outperform Types 1 and 2. For the proposed N-gram semantic neural model, the performance is improved compared to Types 3 and 4, as we combine CNN and RNN structures to further improve the effectiveness. Thus, the proposed model that employs an N-gram semantic neural model to solve the mobile malware traffic detection problem in an end-to-end way is recommended.

Specially, in [33], a two-layer deep learning method is proposed to detect android malware. The first layer employs the static features about permission, intent, and component information. And, the second layer employs the network traffic features to receive the classification results from the first layer. In our research, we reimplement the network traffic detecting method and conduct a comparative experiment.

The implemented deep learning model parameters are shown in Table 9. In the research [33], each network flow was converted to a 24*24 image, namely, 784 bytes of network flow data were utilized. In the comparison experiment, we used the first 700 bytes of data from the network flow in our proposed scheme.

The binary classification performance metrics is shown in Table 10.

According to the comparative experiments, the proposed scheme performs better than the scheme in [33]. The average accuracy of the [33] model for detecting the malware from benign traffic is 0.768, while the accuracy of our proposed model can reach 0.899. The difference in detection accuracy is mainly due to the different deep learning models. Scheme [33] uses CNN architecture while our scheme employs CNN combined GRU architecture. These two kinds of architectures are different in the way of learning features. Both can learn the characteristics of malware and benign software. However, the proposed model can acquire the features of the time dimension while CNN can only learn the spatial features.

4.4.2. Different Text Size. The text size is another important impactor for the classification results, and we use the text

Table 9: Description of the reimplemented parameters of ref [33].

Layer	Operation	Input	Kernel	Output
1	Convolution	28*28*1	5*5*32	28*28*32
2	Max pool	28*28*32	2*2**32	14*14*32
3	Convolution	14*14*32	5*5*64	14*14*64
4	Max pool	14*14*64	2*2*64	7*7*64
5	Fully connected	7*7*64	—	1024
6	Fully connected	1024	—	2/4

Table 10: Comparative experiment result of binary classification performance between our scheme and ref [33].

Evaluation metrics	Our scheme	Ref [33]
Precision	0.954	0.770
Recall	0.848	0.765
F1-score	0.898	0.766
Accuracy	0.899	0.768

size as the parameter to develop several experiments. We extract and splice the application layer payload of the successive packets in each flow as flow payload data. And, the text size is the length of the flow payload data. Flow payload with different sizes may contain different amounts of information, which can affect the detection of malware. For testing this parameter, we selected 6 sizes of text, namely, 200 bytes, 500 bytes, 700 bytes, 1000 bytes, 1200 bytes, and 1500 bytes. In the case of flows more than the size of the texts, we have disregarded the following payload, and in the case of flows with less than the size of the texts, we have padded them with zeros. Figure 9 depicts the mobile malware traffic detection accuracy with different text sizes using different deep learning architectures.

Figure 9 depicts that the performance of the model increases with the size of the input text. When the input text size is below 1000 bytes, the performance of the model is poor, including precision, recall, and accuracy. When the input text size is 1000 bytes, the model performance is relatively high, with an accuracy of about 0.91 using the proposed model. When the input text size is 1200 bytes, the model performance curve starts to flatten out and the performance indexes no longer increase significantly. Even when the input text size is 1500 bytes, the performance of the model tends to decline. The larger the size of the text used, the more features can be captured, and the more accurate the detection of mobile malware traffic. However, when a certain length is reached, the content of traffic payload cannot continue to provide effective features to help improve the detection accuracy.

4.4.3. Binary-Classification in HTTP Scenario. In [3], each HTTP flow generated by mobile apps is converted into a text document, which can be processed by natural language processing to extract text-level features. They examine the traffic flow header using the N-gram method from the natural language processing (NLP). Then, an automatic feature selection algorithm based on chi-square test is utilized to identify meaningful features. These automatically selected features are used to build an SVM classifier for malware detection. In our research, we reimplement the network traffic detecting method and conduct a comparative experiment. In this process, we divided the HTTP dataset into training set and test set according to the ratio of 8 : 2. The HTTP dataset is described in Table 11.

The binary classification performance metrics is shown in Table 12. As HTTP protocol has more obvious features in semantic, both methods have high accuracy in detecting malicious traffic from HTTP protocol traffic. Both approaches make full use of the semantic information of HTTP traffic. The difference is that the approach in Ref [3] analyzes and extracts specific field contents as features manually, while our approach learns valid information automatically by deep learning models.

4.5. Discussion. With the proposed N-gram semantic based neural model, the essential purpose of this paper has been achieved, namely, detecting the different protocol types of mobile malware traffic in an end-to-end way by transferring the traffic flow into character-level text.

The existing detecting method needs manual analysis of traffic semantic information generated by the specific protocol, which failed to detect the mobile malware traffic hiding in other protocols. To address this problem, we proposed to employ deep learning methods to learn the text converted from network flows. We segment the network traffic into flows according to Algorithm 1. Then, the application payload of the packets in each flow is extracted and is waiting to be transferred into other forms that can be fed into deep learning algorithms. In our research, the flow payload data are transferred into the character-level text form. Application layer payload consists of the user behavior data expressed by different types of application protocols, i.e., HTTP and DNS. We utilize a multiwidth kernel CNN as the domain composition to learn the text semantic. And, different width kernels have different weights updated in the training progress. And, we proved the advantage of multikernel CNN through experiments. Afterward, we select the GRU architecture as the flow composition in the model by conducting comparative experiments on 4 types of deep learning architectures, namely, 1-layer LSTM, 1-layer GRU, 2-layer LSTM, and 2-layer GRU.

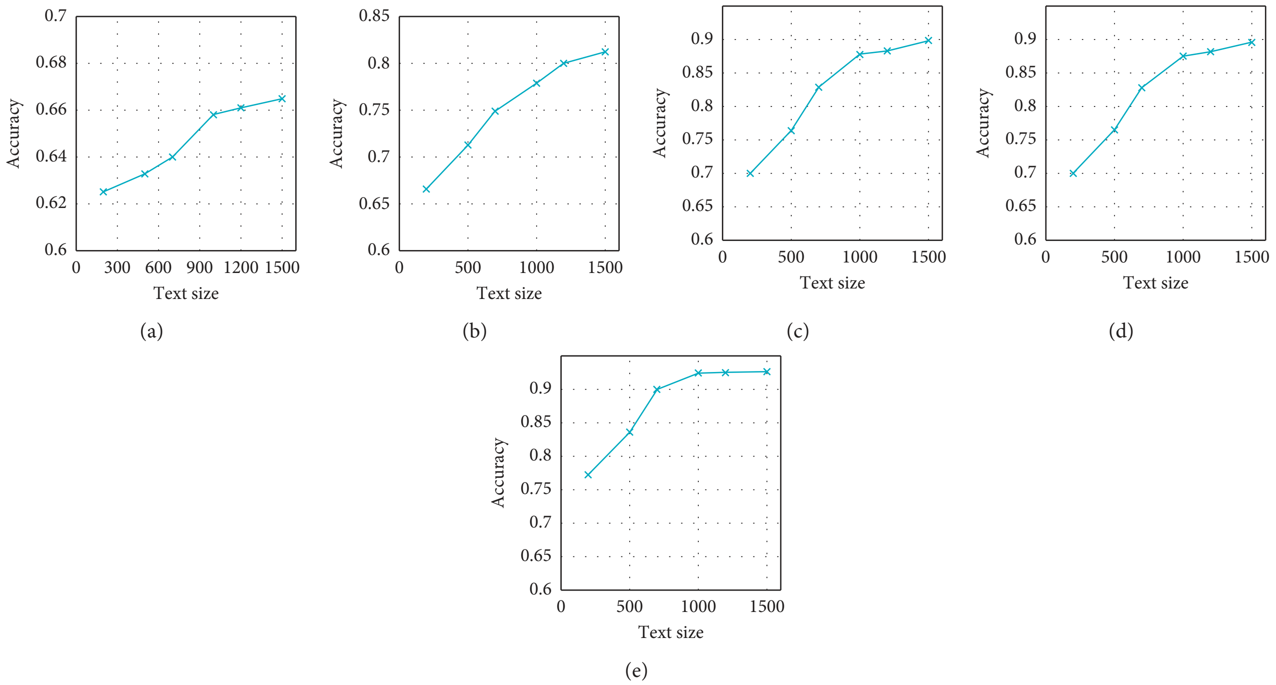


Figure 9: Accuracy of different text sizes in case of different deep learning architectures. (a) MLP architecture. (b) TextCNN architecture. (c) LSTM architecture. (d) GRU architecture. (e) Proposed architecture.

Table 11: Description of HTTP traffic dataset for 2-classification experiment.

Malware type	Benign	Malware
Train	41637	35145
Test	10409	8786
Total	52046	43931

Table 12: Comparative experiment results of binary-classification performance for detecting HTTP mobile malware traffic between our scheme and ref [3].

Evaluation metrics	Our scheme	Ref [3]
Precision	0.969	0.955
Recall	0.943	0.962
F1-score	0.956	0.941
Accuracy	0.959	0.955

To verify the effectiveness of the proposed scheme, a series of experiments are conducted. In these experiments, we compare the performance of different deep learning architectures and a recommended architecture has been the proposed N-gram, semantic-based neural model according to the experiments. In addition, we also consider the impact of the size of the text, namely, the length of the used flow payload. Experiment results show that as the text size increases, the detection effect gets better. However, when the size exceeds 1000 bytes, the improvement of detection effect becomes less.

5. Conclusion

In this paper, we present a scheme to detect mobile malware traffic by extracting flow payload and converting it into character-level text. And we employ multi-filter CNN and GRU to model the generated text into domain representations and flow representations, respectively. As the text is character-level based, no manual analysis about specific protocol semantic information is needed. To accomplish the above work, we also designed a tool, called PKTPT, to process the network traffic into flows and extract the application layer payload from each packet. From the several experiments, our proposed scheme is proved to be effective. Compared with the state-of-the-art methods, several comparative experiments are also conducted. And, the experiment results depict that our proposed scheme is better in terms of accuracy and is suitable for mobile malware detection that is likely to use various protocols. However, as more mobile software uses encryption protocols, semantic-based malware traffic detection becomes less reliable. We will study methods for detecting mobile malware traffic in the scenario of encrypted traffic in the future.

Data Availability

The data are available from http://205.174.165.80/CICDataset/CICMalAnal2017/Dataset/.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant Nos. U1836104 and 61702235 and in part by the Fundamental Research

Funds for the Central Universities under Grant No. 30918012204.

References

[1] G. Play, "Number of apps 2009–2016," 2017, http://www.statista.com/statistics/266210/.

[2] Security threat report 2014—sophos, 2017, http://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecurity-threat-report-2014.pdf.

[3] S. Wang, Q. Yan, and Z. Chen, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1096–1109, 2017.

[4] S.-H. Yoon, J.-W. Park, J.-S. Park, Y.-S. Oh, and M.-S. Kim, "Internet application traffic classification using fixed IP-port," in *Proceedings of the Asia-Pacific Network Operations and Management Symposium*, pp. 21–30, Springer, Jeju, Republic of Korea, 2009.

[5] W. Wang, M. Zhao, Z. Gao, G. Xian, Y. Li, and X. Zhang, "Constructing features for detecting android malicious applications: issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[6] C. Urcuqui-López and A. Navarro Cadavid, "Framework for malware analysis in Android," *Sistemas y Telemática*, vol. 14, no. 37, pp. 45–56, 2016.

[7] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in *Proceedings of the IEEE 25th International Conference Tools Artificial Intelligence*, pp. 300–305, Herndon, VA, USA, November 2013.

[8] L. Onwuzurike, E. Mariconti, and P. Andriotis, "MaMaDroid: detecting Android malware by building Markov chains of behavioral models (extended version)," *ACM Transactions on Information and System Security*, vol. 2, no. 2, 2019.

[9] S. Hou, Y. Ye, and Y. Song, "HinDroid: an intelligent Android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD International Conference Knowledge Discovery Data Mining*, pp. 1507–1515, ACM, Halifax, Canada, August 2017.

[10] H. Kang, J.-W. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 479174, 2015.

[11] Y. Zhang, M. Yang, and B. Xu, "Vetting undesirable behaviors in Android apps with permission use analysis," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 611–622, ACM, Berlin, Germany, November 2013.

[12] L. Sun, Z. Li, and Q. Yan, "SigPID: significant permission identification for Android malware detection," in *Proceedings of the 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 1–8, Fajardo, PR, USA, October 2016.

[13] X. Wang, K. Sun, and Y. Wang, "DeepDroid: dynamically enforcing enterprise policy on Android devices," in *Proceedings of the Network and Distributed System Security Symposium*, pp. 1–15, San Diego, CA, USA, February 2015.

[14] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[15] A. H. Lashkari, A. F. A. Kadir, and L. Taheri, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *Proceedings of the International Carnahan Conference on Security Technology (ICCST)*, pp. 1–7, Montreal, Canada, October 2018.

[16] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic Android malware detection at scale," in *Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 1666–1671, Sardinia, Italy, July 2013.

[17] M. Grace, Y. Zhou, and Q. Zhang, "RiskRanker: scalable and accurate zero-day Android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 281–294, London, UK, 2012.

[18] V. Rastogi, Y. Chen, and W. Enck, "Apps play ground: automatic security analysis of smartphone applications," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*, pp. 209–220, San Antonio, TX, USA, 2013.

[19] J. W. Jang, J. Yun, and J. Woo, "Andro-profiler: anti-malware system based on behavior profiling of mobile malware," in *Proceedings of the 23rd International World Wide Web Conference*, pp. 737-738, Seoul, Republic of Korea, 2014.

[20] A. Shabtai, U. Kanonov, and Y. Elovici, "Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method," *Journal of Systems and Software*, vol. 83, no. 8, pp. 1524–1537, 2010.

[21] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible Android malware detection and family classification using network-flows and API- calls," in *Proceedings of the International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8, Chennai, India, October 2019.

[22] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Detecting Android malware using long short-term memory (LSTM)," *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1277–1288, 2018.

[23] Malware Detection Methods, http://www.avg.com/us-en/avg-software-technology.

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

[25] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically gener- ating signatures for polymorphic worms," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 226–241, Oakland, CA, USA, May 2005.

[26] S. Singh, C. Estan, and G. Varghese, "Automated worm fingerprinting," in *Proceedings of the OSDI*, vol. 4, no. 4, San Francisco, CA, USA, 2004.

[27] V. Yegneswaran, J. T. Giffin, and P. Barford, "An architecture for generating semantics-aware signatures," in *Proceedings of the USENIX Conference on Security Symposium*, vol. 7, San Diego, CA, USA, 2005.

[28] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, p. 26, San Jose, CA, USA, 2010.

[29] M. Aresu, D. Ariu, and M. Ahmadi, "Clustering android malware families by traffic," in *Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 128–135, Fajardo, PR, USA, October 2015.

[30] A. Lashkar, A. Kadir, and H. Gonzalez, "Towards a network-based framework for android malware detection and characterization," in *Proceedings of the 2017 15th Annual*

Conference on Privacy, Security and Trust (PST), p. 233, Calgary, Canada, 2017.

[31] A. Arora, S. Garg, and S. K. Peddoju, “Malware detection using network traffic analysis in Android based mobile devices,” in *Proceedings of the 8th International Conference on Next Generation Mobile Applications, Services and Technologies*, pp. 66–71, Oxford, UK, September 2014.

[32] G. Bendiab, S. Shiaeles, and A. Alruban, “IoT malware network traffic classification using visual representation and deep learning,” in *Proceedings of the 2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pp. 444–449, Tokyo, Japan, July 2020.

[33] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, “A two-layer deep learning method for android malware detection using network traffic,” *IEEE Access*, vol. 8, no. 8, pp. 125786–125796, 2020.

[34] J. Huang, X. Zhang, and L. Tan, “AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction,” in *Proceedings of the 34th International Conference on Software Engineering*, pp. 1036–1046, Zurich, Switzerland, June 2014.

[35] R. Pandita, X. Xiao, and W. Yang, “WHYPER: towards automating risk assessment of mobile applications,” in *Proceedings of the USENIX Security Symposium*, pp. 527–542, Anaheim, CA, USA, 2013.

[36] Y. Nan, M. Yang, and Z. Yang, “UIPicker: user-input privacy identification in mobile applications,” in *Proceedings of the USENIX Security Symposium*, pp. 993–1008, Washington, DC, USA, August 2015.

[37] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, “A semantics-aware approach to the automated network protocol identification,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 583–595, 2016.

[38] J. Ren, A. Rao, and M. Lindorfer, “ReCon: revealing and controlling PII leaks in mobile network traffic,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, Singapore, June 2015.

[39] Virustotal, https://www.virustotal.com/.

[40] V. Total, “Contagio mobile malware mini dump,” 2016, http://contagiominidump.blogspot.ca/.

[41] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, “Android botnets: what urls are telling us,” in *Proceedings of the International Conference Network and System Security*, pp. 78–91, Springer, Madrid, Spain, 2015.

[42] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, “Droidkin: lightweight detection of android apps similarity,” in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 436–453, Springer, Beijing, China, September 2014.