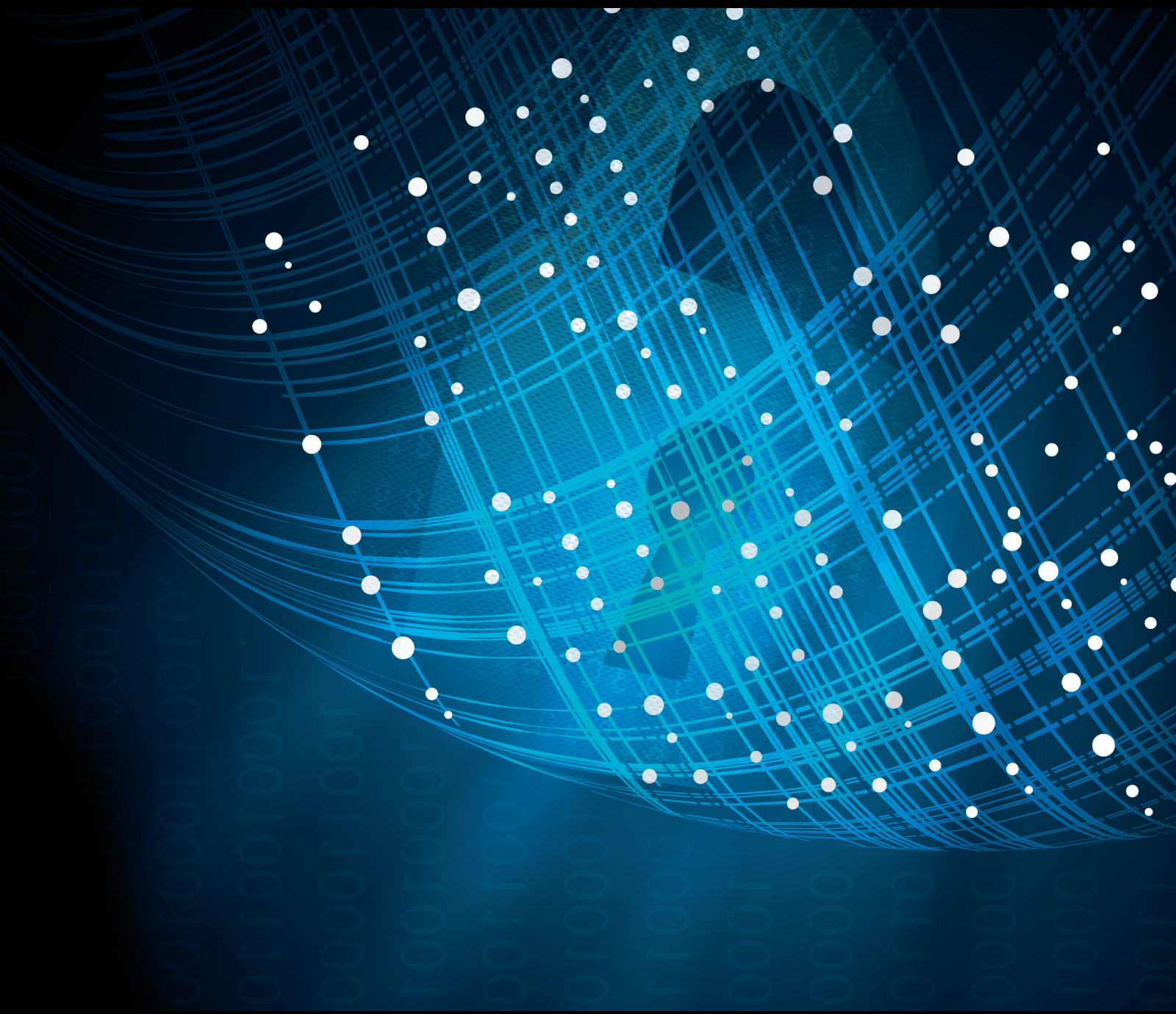
Security and Communication Networks

Malware Analysis and Vulnerability Detection Using Machine Learning

Lead Guest Editor: Farrukh A. Khan
Guest Editors: Muhammad Faisal Amjad, Yin Zhang, and Hammad Afzal



Malware Analysis and Vulnerability Detection Using Machine Learning

Security and Communication Networks

Malware Analysis and Vulnerability Detection Using Machine Learning

Lead Guest Editor: Farrukh A. Khan
Guest Editors: Muhammad Faisal Amjad, Yin Zhang, and Hammad Afzal

Contents

Hindawi
Security and Communication Networks
Volume 2020, Article ID 8841544, 20 pages
https://doi.org/10.1155/2020/8841544

Research Article

MD-MinerP: Interaction Profiling Bipartite Graph Mining for Malware-Control Domain Detection

Tzung-Han Jeng,[1,2] **Yi-Ming Chen**,[2] **Chien-Chih Chen**,[1] **and Chuan-Chiang Huang**[1]

[1]*Chunghwa Telecommunication Laboratories, Taoyuan, Taiwan*
[2]*National Central University, Taoyuan, Taiwan*

Correspondence should be addressed to Tzung-Han Jeng; tzunghan@cht.com.tw

Received 9 April 2020; Revised 10 July 2020; Accepted 16 October 2020; Published 29 October 2020

Academic Editor: Hammad Afzal

Despite the efforts of information security experts, cybercrimes are still emerging at an alarming rate. Among the tools used by cybercriminals, malicious domains are indispensable and harm from the Internet has become a global problem. Malicious domains play an important role from SPAM and Cross-Site Scripting (XSS) threats to Botnet and Advanced Persistent Threat (APT) attacks at large scales. To ensure there is not a single point of failure or to prevent their detection and blocking, malware authors have employed domain generation algorithms (DGAs) and domain-flux techniques to generate a large number of domain names for malicious servers. As a result, malicious servers are difficult to detect and remove. Furthermore, the clues of cybercrime are stored in network traffic logs, but analyzing long-term big network traffic data is a challenge. To adapt the technology of cybercrimes and automatically detect unknown malicious threats, we previously proposed a system called *MD-Miner*. To improve its efficiency and accuracy, we propose the *MD-Miner*^{P} here, which generates more features with identification capabilities in the feature extraction stage. Moreover, *MD-Miner*^{P} adapts interaction profiling bipartite graphs instead of annotated bipartite graphs. The experimental results show that *MD-Miner*^{P} has better area under curve (AUC) results and found new malicious domains that could not be recognized by other threat intelligence systems. The *MD-Miner*^{P} exhibits both scalability and applicability, which has been experimentally validated on actual enterprise network traffic.

1. Introduction

Cybercrimes are becoming increasingly serious with the proliferation of Internet devices and applications. One of the most frequently used tools for cybercrimes is malicious domains to perform phishing, XSS, and other attacks. Internet attack organizations generally use code obfuscation techniques to generate a large number of polymorphic variants with the same malware [1] before establishing more than one command and control (C&C) server. Cybercriminals and malware authors leverage not only hidden and slow APT attacks but also various techniques, such as DGAs and domain-flux, to make them successful. By adopting technologies such as DGAs, these servers change their domain names and corresponding IP addresses over time to prevent being blocked by antivirus software or intrusion prevention systems [2]. The detection of malicious domains is difficult because of the defense dilemma caused by the long-term attack and the volatility of their domain names. However, malware generally exhibit footprints that show where they have been. The clue to tracking cybercrimes is in the network traffic; the challenge is how to analyze the huge amount of network traffic. Of the applications in malicious domains, botnets are considered the most damaging by enterprises.

A set of infected and controlled entities can be viewed as a botnet [3]. The botnet structure is composed of three main components: (1) the bots, (2) the command and control servers (C&C), and (3) the threat actor, or bot herder itself; bot, which refers to a remote victim computer, usually without the victim's knowledge; and C&C server, responsible for managing the trunk host that controls the entire botnet and passes along the bot herder's instructions. Once the botnet deployment is complete and launches a cyber-

attack, the distributed denial-of-service (DDoS) shuts down the victim organization's Internet service, and the APT leads to additional damage. Compromised hosts need the Internet as a communication bridge to perform cybercrimes, such as receiving instructions or stealing sensitive data and returning it to the C&C servers [4, 5]. The impact of botnets is great enough that several studies have focused their attention on the discovery of botnets, which has continued to be a hot topic [6–10].

To defend against cyber-attacks, many organizations have established systems such as intrusion detection systems (IDS) to detect and log suspicious traffic, but these produce many false alarms that dull their vigilance [11]. Unlike advanced traffic analysis techniques that require large amounts of computational resources and time, the domain blacklist matching method can instantly detect malicious domains and further disrupt their communications. However, the methods to perform string changes to domain names are simple, cheap, and fast, indicating that using domain blacklists to prevent attacks is effective but difficult to update in real time. Therefore, automating the maintenance of the domain blacklist is indispensable to improve the information security of organizations.

As described in our previous research [12], discovering botnets is important, and detecting C&C servers is vital to analyze APT events. Malicious domain names commonly require an Internet connection to communicate with compromised hosts, but tracking or mining them from the global public Internet has been a difficult problem. Fortunately, such processes leave footprints, and most enterprises leverage proxy servers as intermediate HTTP communications between internal computers and the Internet that result in logging footprints. Thus, systems can take advantage of packet capturing systems to obtain the HTTP communication records. However, one of the bottlenecks in analyzing network traffic is a single workstation can easily have millions of packets each day, which inhibits manually analyzing such traffic without automated intelligence systems. Therefore, we proposed the *MD-Miner* (*MD* stands for malicious domain) that adapts big data analysis with a scalability framework. The process utilizes network traffic to build a *Process*-domain annotated graph that discovers who is connecting with what. The *MD-Miner* uses user-agent plus client-IP as a feature to distinguish the distinct processes and incorporates this into the annotated bipartite graph to become the *Process*-domain annotated graph. The evaluation in [12] shows that the *MD-Miner* can determine a part of unknown domains that has a high probability of being malicious and demonstrates great identifiability, but there is still room for further improvement.

Inheriting from our previous research [12], we built a new scalable network-level behavior system called *MD-Miner*^{P} (^{P} represents Plus) that is based on the Hadoop and Spark cluster architecture. The design effectively uses an incremental clustering algorithm to handle large amounts of data. The *MD-Miner*^{P} has evolved unique analytic capabilities that constantly examine the subtle clues left in proxy or network traffic logs to discriminate malicious domains.

This article demonstrates the steps to convert the *MD-Miner* to the *MD-Miner*^{P} through two key points. First, the *MD-Miner*^{P} replaces the annotated bipartite graph with an interaction profiling bipartite graph that better represents the association of Internet interactions. Second, the *MD-Miner*^{P} exploits more connection factors to construct features with classification capabilities. In addition to the user-agent plus client-IP (*Process*), the *MD-Miner*^{P} uses HTTP requests, domain IP addresses, and domain name lexical characteristics. The *MD-Miner*^{P} leverages the user-agent plus client-IP building *Process*-domain interaction profiling graph to acquaint process queries that leverage HTTP requests to build the *Trace*-domain interaction profiling graph and determine the interactions between the client-server. The system also leverages the IP address of the destination domain to build the IP-domain interaction profiling graph to identify corresponding relations of the IP used by the domain name. The lexical algorithm is also used to extract variations in the domain string. Finally, these features are aggregated to frame the malicious domain detector. Related works and observations related to improvements of the *MD-Miner*^{P} are detailed in Section 2.

The evaluation stage in Section 4 uses the CyberGraph [13] to verify new malicious domains found by the *MD-Miner*^{P} in addition to the previously used K-fold cross-validation. The CyberGraph is a novel potential malicious domain verification analysis platform that retrieves different types of observable intelligence from different sources to produce a series of observations over time. This allows users to judge threats on the Internet. The CyberGraph is committed to integrating standardized and structured information through a vast and complex network intelligence.

The remainder of this paper is organized as follows. Section 2 describes the background and the assumptions and observations of our approach. Section 3 provides implementation detail of *MD-Miner*^{P} and formulates the research contribution. Moreover, our design goals and core concepts are introduced and a simple example is used to illustrate the data flow of the framework. Section 4 shows the results from our evaluation using ISP-confirmed real-world network traffic to determine the effectiveness of the proposed system. Finally, a summary of the contributions and future research developments are presented in Section 5.

2. Background and Related Work

The principles of related techniques used by the *MD-Miner*^{P} to generate the domain features are described in this section. The *MD-Miner*^{P} has two major evolutions: improvements to the annotated bipartite graph and additional significant features. There are different annotated bipartite graphs imported for feature extraction. In [14, 15], two systems called *Segugio* and *Doctrina* are built from different annotated bipartite graphs with the DNS logs. These systems extract DNS answer-based features, time-based features, domain name-based features, and TTL value-based features of the DNS traffic to detect malicious domain activities. We used annotated bipartite graphs to develop a system, called *MD-Miner*, that monitors the network traffic to build a

Process-domain annotated graph, as shown in Figure 1, to represent who is connecting to what [12]. The *MD-Miner* has abundant DNS logs available and is a scalable architecture. As shown in Figure 1, there are only malicious, benign, and unknown labels in the annotated bipartite graph, but the content of the network traffic log is not as simple as the DNS log. Therefore, the *MD-Miner*^{P} replaces the annotated bipartite graph with an interaction profiling bipartite graph, which is detailed in Section 3 and has experimental results that show promise for its application.

2.1. User-Agent. The first factor in the network traffic log is the user-agent in the HTTP header sent along with a request for an Internet server, which is often but not always sent from a web browser. The intent is to inform the server of the capabilities of the software used by the client. The implementation of a classifier for user-agent strings with support vector machines is described in [16]. On the other hand, as mentioned in [12], the text area of a binary-analyzed result for malware suggests that when the user-agent string is hard-coded in the malware's text area, the user-agent and malicious activities have a considerable degree of correlation. Anomalous user-agent strings were considered in [17] to determine the association with malware activities. However, dedicated user-agent strings that define attackers can easily evade detection by changing their form. Therefore, the *MD-Miner* [12] proposed a *Process*-domain annotated graph that uses user-agent strings and the client-IP in the network traffic as a feature to differentiate the network activity that was emitted from the same process and stores the information about who is connecting to what. In this annotated bipartite graph, the nodes represent either the *Process* nodes (p_1~p_4) or domain nodes (d_1~d_5), and an edge connects a *Process* to a domain if the connection occurred during the considered traffic observation time window. The classification results are used here to construct more effective bipartite graphs based on its composition using the factors described below.

2.2. HTTP Request. The HTTP network traffic contains significant important information to detect malicious interactions between malware-controlled domains and malware-compromised machines. HTTP is an application layer protocol that uses headers to transfer metadata over a client-server model where the client sends a request to a server, which responds with the available appropriate resource. The HTTP requests are important in Internet interactions, making this the second factor used to extract domain features. Many works have confirmed that the vast majority of malware leverages HTTP as a communication bridge with a cybercriminal's C&C server to perpetrate malicious activities [18, 19]. Such tricks are not only used in the majority of SPAM botnets but also operated on the APT [20–25]. In addition, the malware sample network activity experiments in [26] indicate that approximately 75% of malware samples trigger network activities and generate HTTP traffic. A malware clustering system was introduced in [26] to analyze the structural similarities between malicious HTTP requests in network traffic and used the application path and query string to calculate the distance between malware to clustering malware to obtain its signature. In addition, the HTTP request contained in the headers includes the path (e.g., /path/data) and query (e.g., ?key = value&key2 = value2) as ensconced interactive information between the client and server. References [27, 28] tried to detect malicious phishing web sites using path and query keywords by comparing the relevancy of terms within their URLs. One risk level is the similarity between the path and query terms based on Google Trend and Yahoo Clue. In studies that use HTTP protocols to detect suspicious packets [29, 30], the similarity from the URL path, parameter, and value could identify the packet as malicious or benign.

The *MD-Miner*^{P} refers to the *Trace-Channel* interaction profiling graph proposed by our previous research on the *CC-Tracker* [19], which extends similar observations to [26]. The observation is that different malware samples that rely on the same web server application have similarly structured queries and related URL sequences. To reduce the complexity of the computing similarity between HTTP requests, we simplified the HTTP request as *Trace*, as shown in Figure 2. The upper part of Figure 2 shows that the *Trace* takes a raw HTTP request of "GET /web page.php? key1 = value1&key2 = value2&k3 = v3" as an example, where *m* indicates the method to query the URL and *p* denotes the queried page. The remaining terms used to query the URL are *n* and *v*, which are after the question mark and are in the form of a key = value pair, where *n* indicates the parameter name of the queried URL and *v* denotes the parameter. As the parameter values are relatively easy to change, all parameter values are replaced with the same symbol, which ignores the parameter values [19]. Therefore, the original HTTP request can be simplified to "GET_/web page.php?| key1|key2|k3|," as shown in the lower part of Figure 2.

2.3. IP Address. The Internet protocol (IP) address is a unique logical digital address assigned to each hardware-equipped network and is recognized by the other devices through the IP address. Benign and malicious domains also have their own IP address and the correspondences are recorded in the network traffic files. The IP addresses are more stable than other metrics, such as the URL and DNS. That is, the domain string can easily change while the IP address is generally fixed. Cybercrimes create a specific technique called obfuscation to change the domain name string, which has been identified and summarized as having four basic types [31]. In contrast, the IP address holds two inborn traits that make it more difficult to change: stability with time and address space skewness [32–34]. If it can be proven that the IP address used by a domain name *d* is positively related to a known malicious activity, then the domain name may be considered as malicious. Considering these two characteristics, the *Segugio* [14] and *Doctrina* [15] approaches successfully transformed the correspondence between the IP and domain names into features to mine for malicious domain names from the DNS logs. Moreover, some research used domain IP mapping as a trait to find

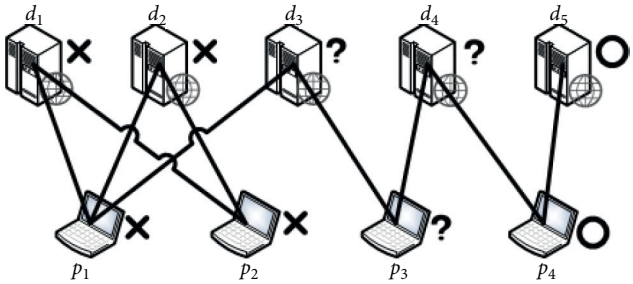


Figure 1: *Process*-domain annotated graph.

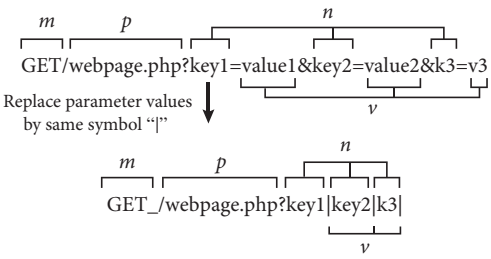


Figure 2: Common indication to simplify the HTTP request to *Trace*.

network threats [35, 36]. While these detection methods can still be improved, they prove that IP addresses could be an effective identification factor. The *MD-Miner*^{P} takes advantage of the mapping between the domain and IP address to become the third factor. This approach employs the interaction profiling bipartite graph concept to construct the IP-domain interaction profiling graph from network traffic logs to produce effective detection features.

2.4. Lexical Analysis. Manipulating the domain name is another common practice for cybercrimes. Previous research [37] has shown that nearly one-third of all websites in the world are potentially malicious. Many malicious URLs follow obfuscation methods that make the URL strings similar to benign URLs to avoid detection. However, studying various detection methods by analyzing the diversity of domain strings allows designing effective malicious URL detection solutions [38]. These develop lexical features that excavate the divergence of URLs by analyzing the statistical properties of URL strings. The adjective lexical describes the relation to a vocabulary of words and the associated lexical analysis is based on the characteristics of the URL string to determine the lexical features that represent the features of a URL name. Lexical features refer to the actual text without other external information of the URL string. The intention is to make malicious URLs "look" different to experts when compared with benign ones [27].

Most lexical features commonly used for such classifications include the statistical properties of the URL string, like the numerical information regarding the feature lengths (URL length, top-level domain length, primary domain length, etc.) and the number of special characters [39]. The extracted information is obfuscation-resistant and useful. One lexical analysis approach is called the bag-of-words (BoW), which builds a dictionary as a feature set by referring to all the different types of words in all URLs. When a URL includes a word in the dictionary, the value of the feature is 1; otherwise, it is 0. The *MD-Miner*^{P} developed a kind of BoW approach to adapt to big data and accelerate the computing, which is described in detail in Section 3.

Due to the lack of scalability of previous research [26–30], this was restricted to a small amount of material. Therefore, this paper proposes a *MD-Miner*^{P} system which is mainly used to extract hidden malicious threats from long period and large amount of network traffic logs. Our approach takes full advantage of the concept that Internet communications for a specific purpose will invoke similar interactions. The *MD-Miner*^{P} uses attributes in the network traffic log to create representative characteristics for each domain, which answers four important questions. (1) Who is connecting what (*Process*-domain interaction profiling graph)? (2) How to interact with what (*Trace*-domain interaction profiling graph)? (3) What domain name used what IP (IP-domain interaction profiling graph)? (4) What does it "look" like (lexical analysis)? An exhaustive description of how to use the unique methodologies proposed in this paper to establish effective classification features for each domain is given in the following section.

3. *MD-Miner*^{P} Implementation

The concept of the *MD-Miner*^{P} is to track known and discover unknown malicious network domains, which are designated as a channel for attackers to perform malicious acts. Looking at network communications from this perspective allows finding similar traces of connections, and the victim machine generally attempts to connect to malicious or newly created domains. Therefore, the *MD-Miner*^{P} is based on the following main intuitions:

(1) Victim clients tend to connect malicious domain families.

(2) Malware belonging to the same family tend to connect to partially overlapping malware-controlled domains.

(3) Benign applications rarely connect to domains that exist only to provide malicious functionality.

(4) Cybercriminals prepare multiple malicious domains to prevent single-point failure.

(5) Malicious domains reuse the same IP addresses.

(6) Domain names with the same purpose often "look the same."

To take advantage of these points, we proposed a new malicious domain detection system called *MD-Miner*^{P}. The first part of this section gives a detailed explanation for the capture of network domain features. The second part elaborates on the implementation details of the *MD-Miner*^{P} based on the MapReduce framework.

3.1. Domain Features. For each domain in the network traffic, the *MD-Miner*^{P} creates four feature vectors. Three

feature vectors are generated based on the interaction profiling bipartite graph, and the other feature vector is generated based on the lexical analysis. The intuitions described in Section 2 are used to generate relevant features through the interaction profiling bipartite graph, as shown in Figure 3.

In the interaction profiling bipartite graph, the domain represents the node on one side of the binary graph, and the *CF* stands for "connection factor," which is the node on the other side. The connection factors include the *Process*, *Trace*, and *Address* used by the domain. The *Process* indicates the user-agent plus client-IP, *Trace* indicates the simplified HTTP request, and *Address* indicates the domain IP address. The *MD-Miner*^{P} defines three interaction profiling bipartite graphs using these connection factors, where $G_P = (P, D, E_{PD})$ represents the interaction profiling bipartite graph for *Process*, $G_T = (T, D, E_{TD})$ is for *Trace*, and $G_A = (A, D, E_{AD})$ is for *Address*. Node set D represents the domain nodes with $d_i \in D$, node set P represents the *Process* nodes with $p_i \in P$, node set T represents the *Trace* nodes with $t_i \in T$, and node set A represents the *Address* nodes with $a_i \in A$. The edge sets are called E_{PD} in G_P, E_{TD} in G_T, and E_{AD} in G_A. The *Process* p_i connects a domain d_j with an edge $e_{ij} \in E_{PD}$, the *Trace* t_i connects a domain d_j with an edge $e_{ij} \in E_{TD}$, and the *Address* a_i connects a domain d_j with an edge $e_{ij} \in E_{AD}$. The features of different aspects of the network domain can be described from the interaction profiling bipartite graphs for different *CF*s. Communications with the same purposes interact through similar *CF*s. For example, the d_1 and d_2 conduct similar communications as shown in Figure 3. Once the interaction profiling bipartite graph is constructed, the next step is to extract the domain feature vectors from each graph, as detailed below.

Each domain name needs to go through three phases to extract the feature vector by analyzing the interaction profiling bipartite graph. The first phase is to mark the domain node, which obtains benign and malicious domain intelligence (whitelist/blacklist) from a public or private reputation database. If the domain exists in the whitelist, it is marked as *DomainWhite*; if it exists in the blacklist, it is a known malicious domain and marked as *DomainBlack*. All remaining domains are marked as *DomainUnknown*, which are the primary targets for further classification to mine malicious domains that are not recorded in the threat intelligence but are actually hidden.

The second phase is to label each *CF* node as *White*, *Black*, *Mix*, *Unknown*, or *Leaf*. The labeling method is based on the labeled domain nodes where each *CF* node is linked. Three numbers are counted for each *CF* node, namely, $White_{sum}$, $Black_{sum}$, and $Unknown_{sum}$, These are the number of edges of a *CF* node connected to different *DomainWhite*, the number of edges for different *DomainBlack*, and the number of edges for different *DomainUnknown*. Each *CF* node in the interaction profiling bipartite graph is then labeled with its own $White_{sum}$, $Black_{sum}$, and $Unknown_{sum}$. The labeling method is as follows, where the *CF* nodes in the lower part of Figure 4 illustrate the labeling method.

(1) *White*: $White_{sum} > 0$ & $Black_{sum} = 0$ (circle)

(2) *Black*: $\text{Black}_{\text{sum}} > 0$ & $\text{Black}_{\text{sum}} > \text{White}_{\text{sum}}$ (cross)

(3) *Mix*: $Blacksum > 0$ & $White_{sum} > Black_{sum}$ (circle sign combined with cross)

(4) *Unknown*: $Black_{sum} = 0$ & $White_{sum} = 0$ (question mark)

(5) *Leaf*: Connect to a single domain only (triangle)

The third phase is to compute the feature values for each domain node. Figure 4 shows the interaction profiling bipartite graph G_P, where the *Process* feature values of the domain node d_3 are calculated using the G_P as an example. Five values are counted from the attributes of the labeled *Process* nodes to which d_3 is linked: S_P, W_P, B_P, M_P, and U_P, where S_P is the total number of *Process* nodes linked to d_3; W_P is the number of *Process* nodes linked to d_3 and labeled as *White*; B_P is the number of *Process* nodes linked to d_3 and labeled as *Black*; M_P is the number of *Process* nodes linked to d_3 and labeled as *Mix*; and U_P is the number of *Process* nodes linked to d_3 and labeled as *Unknown*. The six following *Process* feature values of d_3 are calculated using the following formulas.

(1) Fraction of *White Process* nodes, $w_P = |W_P|/|S_P|$

(2) Fraction of *Black Process* nodes, $b_P = |B_P|/|S_P|$

(3) Fraction of *Mix Process* nodes, $m_P = |M_P|/|S_P|$

(4) Fraction of *Unknown Process* nodes, $u_P = |U_P|/|S_P|$

(5) Fraction of *Leaf Process* nodes, $l_P = |L_P|/|S_P|$

(6) Fraction of *total Process* nodes, $s_P = |S_P|$

The feature values for d_3 obtained from the above six formulas are 1⁄6, 2⁄6, 1⁄6, 1⁄6, 1⁄6, and 6. Following the same pattern applied to the interaction profiling bipartite graph G_T allows using d_3 to obtain w_T, b_T, m_T, u_T, l_T, and s_T. Applying this to the interaction profiling bipartite graph G_A allows using d_3 to obtain w_A, b_A, m_A, u_A, l_A, and s_A. All the domain nodes are assigned their own 18 feature values in the same way.

The lexical features are those acquired based on the properties of a domain name or string. The motivation is that the domain-based "appearance" should be able to identify the malicious nature of a domain. The *MD-Miner*^{P} directly uses the BoW model, which loses information on the order of tokens that belong to the top-level and primary domains. This is done by creating a separate dictionary for each fragment. The lexical features also include the statistical properties of the domain, such as the length of its name and the number of "." characters.

3.2. MapReduce Algorithm. The *MD-Miner*^{P} is based on two important phases to detect potentially malicious domains, as shown in Figure 5: domain feature extraction and random forest classifier. First, the *MD-Miner*^{P} constructs the domain node feature vector by taking the network traffic log and benign\malicious domain intelligence stored in the domain threat intelligence database as inputs. The domain feature extraction phase consists of four parts that extract 22 features of each domain node: (1) *Process*, (2) *Trace*, (3) *Address*, and (4) lexical feature extractions. Second, the *MD-Miner*^{P} adopts Spark parallel processing to build a random forest

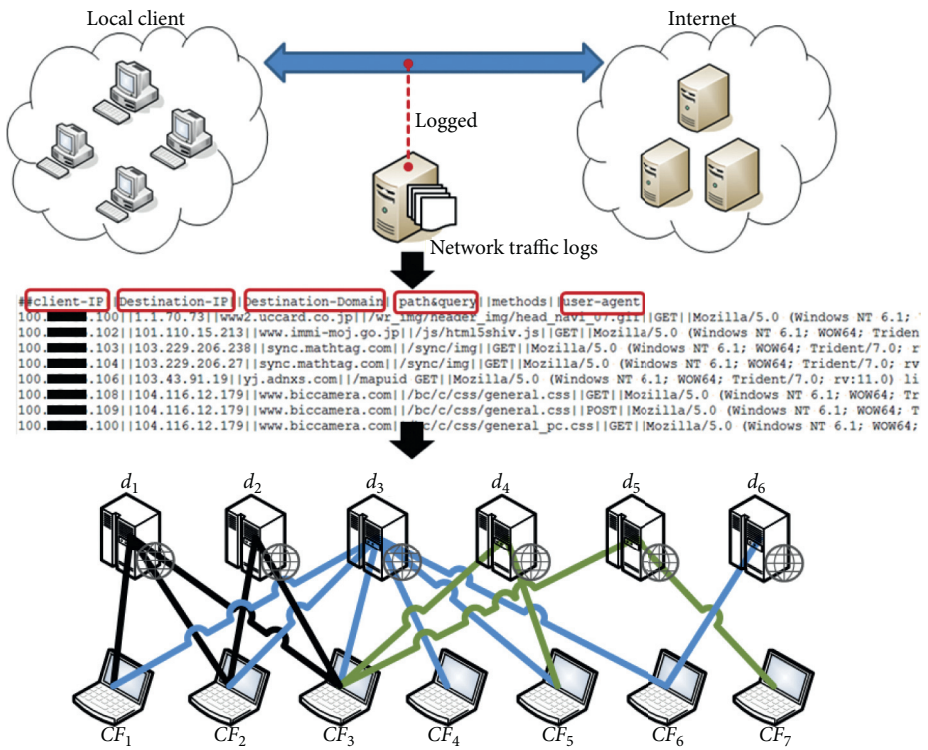


Figure 3: An illustration of the conversion of network traffic into the interaction profiling bipartite graph.

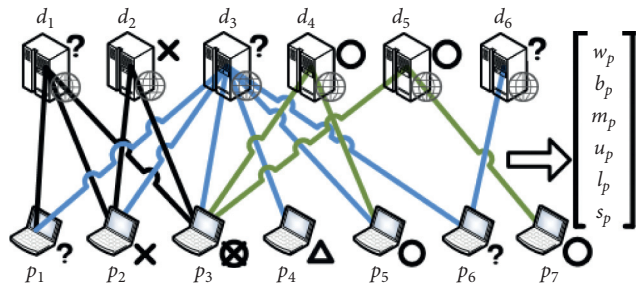


Figure 4: Illustration of the process feature vector generated from the interaction profiling bipartite graph.

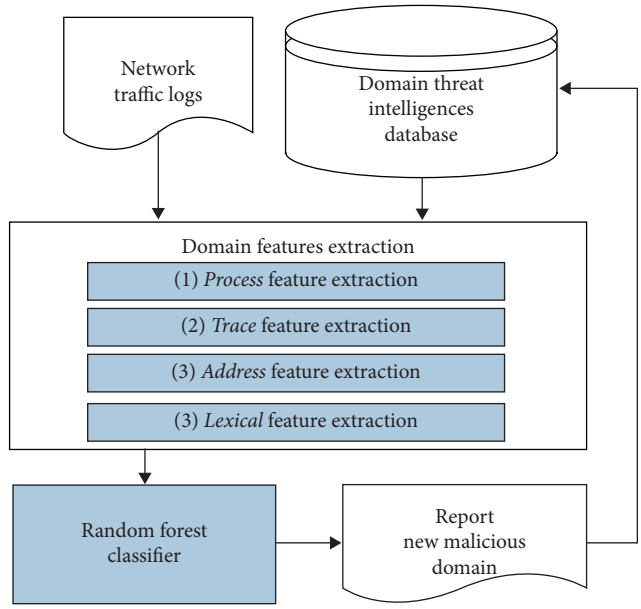


Figure 5: Flow diagram for the *MD-Miner*^{P} process.

classifier based on the decision tree model, which is employed to detect malicious domains.

Parts (1)–(3) of the domain feature extraction are based on a similar concept of using the interaction profiling bipartite graph to obtain adjacent information as features. The *MD-Miner*^{P} designs four MapReduce jobs to realize feature extraction of the interaction profiling bipartite graph: (1) domain node labeling, (2) *CF* node labeling, (3) interaction profiling bipartite graph building, and (4) behavior feature calculating. Taking part (1) as an example, the following is a detailed description of the MapReduce jobs for the *Process* feature extraction when the *Process* nodes are used as *CF* nodes.

The domain node labeling job first utilizes multiple input mechanisms of the map phases with the network traffic and whitelist/blacklist as the input and domain as the key. Parallel label domain nodes are either *DB* (*DomainBlack*), *DW* (*DomainWhite*), or *DU* (*DomainUnknown*) in the reduce phase based on shuffle and sorting mechanisms. An example of the data flow for domain node labeling is shown in Figure 6.

The next job after labeling the domain nodes is to label the *CF* nodes. As described in Section 3.1, the label of a *CF* node is determined from the connected domain nodes. The five label types are *White*, *Black*, *Mix*, *Unknown*, and *Leaf*. The input to the *CF* node labeling job is the output of the domain node label from the previous step. Therefore, the MapReduce job at this step takes the *CF* (e.g., *Process*) node as the key and the domain node as the value in the map phase. In the reduce phase, the number of occurrences for *DB*, *DW*, and *DU* for each *CF* node are counted and the corresponding labels are calculated. The *Process* nodes are taken as the *CF* nodes as an example, and Figure 7 shows the data flow of the labeled *CF* nodes.

The next job is to build the interaction profiling bipartite graph to aggregate the labeled domain nodes and labeled *CF* nodes into a dataset. In the map phase, the output of the domain and *CF* node labeling jobs are taken as the inputs to use the advantages of multiple input mechanisms with the identity of the *CF* (e.g., *Process*) node as the key. The *CF* node labels are annotated for each record to obtain the interaction profiling bipartite graph in the reduce phase. Figure 8 shows an example of the data flow to build an interaction profiling bipartite graph during this job.

The interaction profiling bipartite graph constructed in the above jobs allows calculating the behavior features for each domain node. In the map phase, the constructed interaction profiling bipartite graph output from the previous job is taken as the input, where the domain node is the key. In the reduce phase, each domain node obtains its neighbor's information (labels of *CF* nodes) through the shuffle and sorting mechanism. Therefore, the *MD-Miner*^{P} can compute the behavior features of each domain node in parallel. Figure 9 shows an example of the parallel computing behavior features in the job.

Parts (2) and (3) can be implemented as similar MapReduce jobs for G_T and G_A. The only difference is that Part (1) uses the *Process* (user-agent + client-IP) and domain nodes to construct the interaction profiling bipartite graph G_P, Part (2) uses the *Trace* nodes instead of the *Process* nodes to build the interaction profiling bipartite graph G_T, and Part (3) uses the *Address* (destination IP address) nodes to replace the *Process* nodes and construct the interaction profiling bipartite graph G_A.

The lexical feature extraction in Part (4) uses distributed caching mechanisms to store dictionaries for both the primary and top-level domains and gives each term an index number. The distributed caching mechanism allows calculating the lexical features in a single map phase, including the length of the domain, the number of "." characters, and the index numbers of the top-level and main domains.

Once each domain in the dataset has its own 22 feature values based on the above steps, the *MD-Miner*^{P} performs two steps to employ the random forest classifier based on Spark, which is a unified analytics engine for large-scale data processing. The first step constructs a classifier RF_C by taking all the *DB*, *DW*, and their feature values in the dataset as the training set and inputs them into the random forest algorithm. The second step is to use the classifier RF_C to identify all unknown domains labeled as *DU* in the dataset.

4. Evaluation

The *MD-Miner*^{P} mines stealthy malicious domains for enterprise-scale big network traffic data. Therefore, the *MD-Miner*^{P} is deployed for enterprise network environments. The deployed network environments are called ENT_{N1} and ENT_{N2}, which are both real-world companies based in Taiwan with thousands of networked clients that install and run antivirus software. The ENT_{N1} is a medium-scale company and its compliance with security management rules is relatively relaxed. The organization's network traffic was collected for 8 months (Jan 1, 2018, to Aug 31, 2018). The ENT_{N2} is a large-scale company that follows strict security and information management regulations with a collected network traffic period of 2 weeks (Aug 1, 2018, to Aug 15, 2018). Table 1 gives further details for both datasets.

The experiment presented in this paper is based on the two large network traffic datasets ENT_{N1} and ENT_{N2} and evaluates the overall performance of the *MD-Miner*^{P} from three perspectives. First, k-fold cross-validation was employed to evaluate the classification capabilities of *MD-Miner*^{P}. Second, the actual instances demonstrate the ability of *MD-Miner*^{P} to mine hidden malicious domains. Finally, the ability of *MD-Miner*^{P} to handle big data is demonstrated by adjusting the number of nodes in the parallel computing cluster and observing its operational performance.

To perform the k-fold cross-validation, we begin by marking all the known samples in the dataset as n (negative) or p (positive), where n is interpreted as benign and p is interpreted as malicious. A prediction result produced from classifying a sample with the model is divided into four types. First, true positive (*TP*) indicates the result of the classifier to predict the sample is p when it is; second, false positive (*FP*) indicates the result of the classifier predicts the sample is p when it is n; third, true negative (*TN*) indicates the result when the classifier predicts the sample is n when it is; and fourth, false negative (*FN*) indicates the result when the classifier predicts the sample is n when it is p.

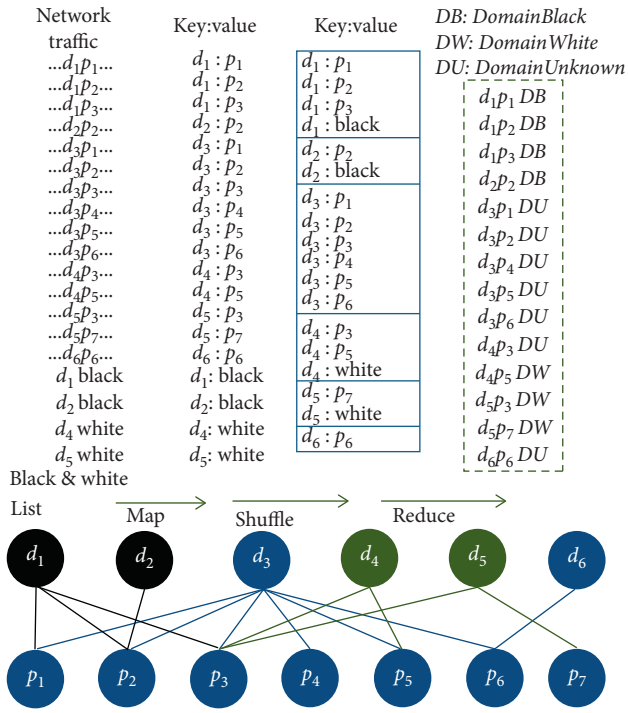


Figure 6: Illustration of domain node labeling for MapReduce jobs.

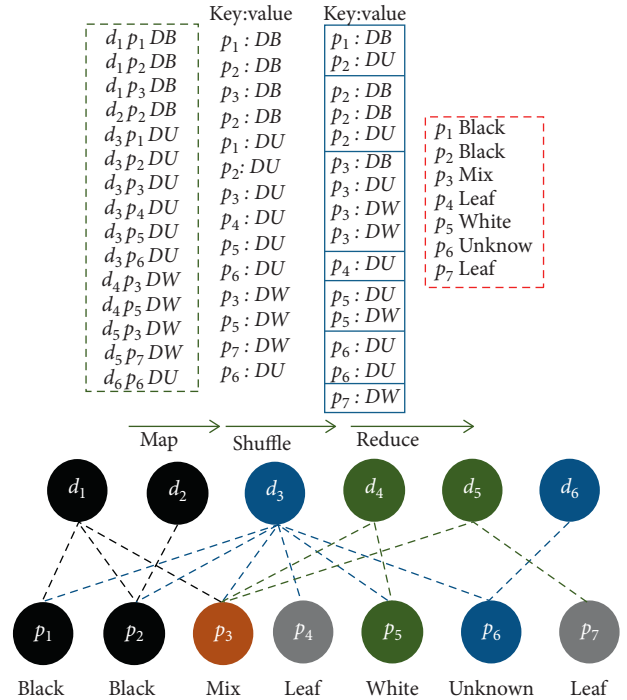


Figure 7: Illustration of the *Process* node labeling for the MapReduce job.

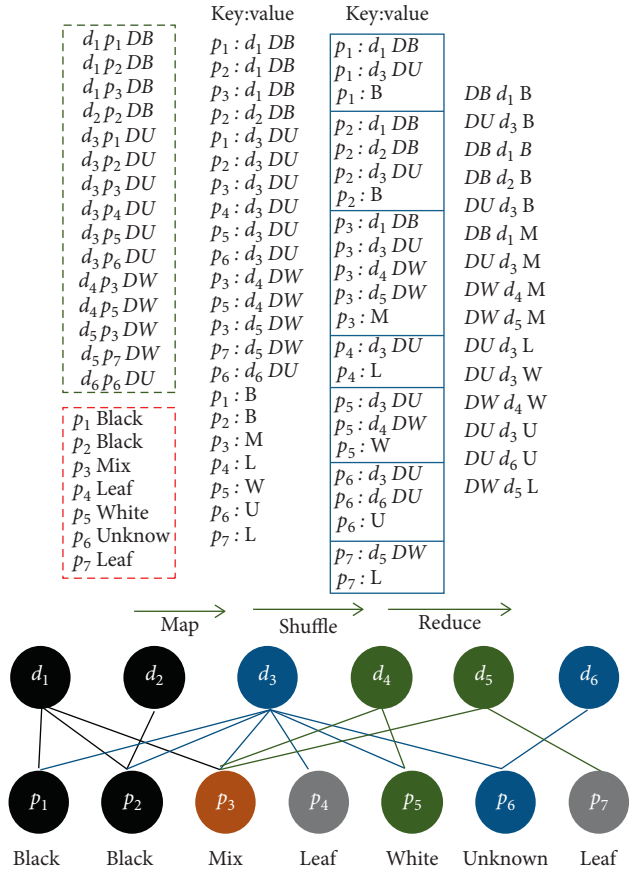


Figure 8: Illustration of interaction profiling bipartite graph building for the MapReduce job.

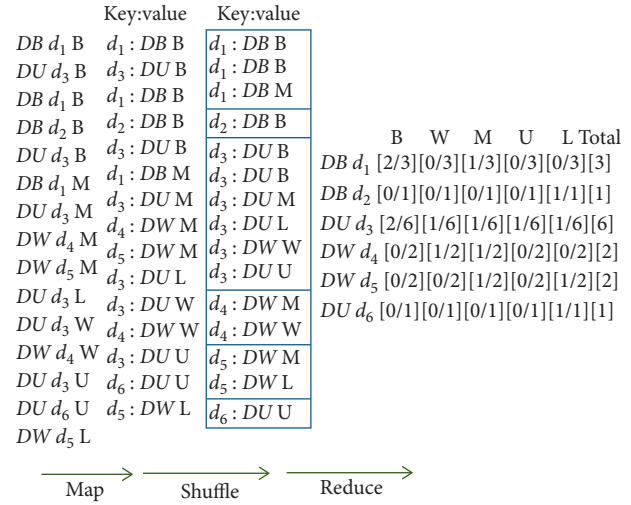


Figure 9: Illustration of the process behavior feature computation for the MapReduce job.

The above description indicates that the first step to prepare the benchmark is to label the ENT_{N1} and ENT_{N2} datasets. Labeling *DW* required employing commercial and public intelligence as whitelists, such as Bluecoat, and the collection of the top 1 million most famous domain names from alexa.com. Labeling the *DB* data required checking if the entire domain name string matches an existing domain from a commercial domain blacklist. When the tagged datasets (ENT_{N1} and ENT_{N2}) are ready, we can generate evaluation criteria for different feature vectors. To conduct a

Table 1: Attributes of the experimental datasets.

	Size (GB)	No of Client-IPs	No of destination domains	No of HTTP requests
ENT_{N1}	34.2	2,336	4,912,810	230,350,760
ENT_{N2}	172.7	13,829	11,946,898	442,036,055

comprehensive evaluation, different metrics are needed: precision, recall, F-measure, accuracy, and AUC. Furthermore, each metric is calculated through a cross-validation process.

The k-fold cross-validation is a resampling procedure used to evaluate machine learning models for limited data samples. The original samples are randomly divided into k equally sized subsamples, where a single subsample is retained as the data for the validation model, while the other k-1 subsamples are used to train the model. The cross-validation process is then repeated k times (called folds), with each k subsample being used only once as the verification data. The results of the k-folds can then be averaged to produce a single estimate. The advantage of the k-fold cross-validation process is that each data sample only needs to be tested once and used to train k-1 times [40]. This paper adopts the 10-fold cross-validation procedure.

4.1. 10-Fold Cross-Validation for MD-Miner^P. The first experiment deployed *MD-Miner*^{P} to the real-world ENT_{N1} and ENT_{N2} datasets and determined their 10-fold cross-validation results. The *MD-Miner*^{P} constructed three interaction profiling bipartite graphs (G_P, G_T, and G_A) by applying the feature extraction method described in Section 3. The three feature vectors were then generated using G_P, G_T, and G_A; each feature vector contained six feature values. Moreover, we used the proposed lexical analysis method to generate the fourth feature vector that contains four feature values. In addition, a feature vector containing 22 feature values was generated by merging the above four feature vectors. The performance of each feature vector was confirmed by observing the classification results calculated by different metrics as shown in Table 2. In addition, we used the ROC curve to show the ability of the classification model to all classification thresholds as shown in 10 and 11

The above experimental results show that when the *MD-Miner*^{P} was deployed to the ENT_{N1} dataset, the *Address* feature vector performed the best, which the AUC and F-measure were as high as 0.99. Although the AUC of the other three feature vectors is greater than 0.8, the recall metric is low, indicating that these features are only applicable to partial data. However, combining feature vectors can improve the overall ability of classification. When *MD-Miner*^{P} was deployed in ENT_{N2} dataset, the characteristic of combining features resulted in a more significant increase in overall classification capacity. Since ENT_{N2} belongs to a relatively diversified dataset, the recall value of general feature vectors is low. However, by combining the feature vectors, the recall can be significantly improved. Furthermore, in both the ENT_{N1} or ENT_{N2} datasets, the AUCs of the feature vectors that combined the other four were above 0.98, which indicates outstanding discrimination.

4.2. Interaction Profiling Bipartite Graph versus Annotated Bipartite Graph. The second experiment was to prove that the interaction profiling bipartite graph leveraged here is better than the annotated bipartite graph adopted in previous studies [12, 14, 15], which are compared in Figure 12. The annotated bipartite graph only considers the benign and malicious attributes of the connected domain when extracting features, as described in Section 3. The interaction profiling bipartite graph further considers additional aspects, such as outlier domains. Therefore, for the same *CF*, the annotated bipartite graph exports three feature values, while the interaction profiling bipartite graph brings out six feature values.

The experiment also utilized the ENT_{N1} and ENT_{N2} datasets. The annotated and interaction profiling bipartite graphs were formed using the same datasets and the same *CF* (selected from *Process*, *Trace*, and *Address*) to generate three feature vectors, which were combined into a fourth feature vector. Comparing the 10-fold cross-validation AUCs of the four feature vectors generated from the annotated bipartite graph and interaction profiling bipartite graph shows which bipartite graph had a better recognition effect. It is noted that the lexical feature vector is not included in the experiment because it only compares two bipartite graphs. The implementation of the annotated bipartite graph is based on previous studies [12, 14, 15], and the interaction profiling bipartite graph is defined in Section 3.

Figures 13–16 are the results of experiments based on the ENT_{N1} dataset. Figure 13 shows that, for the *Process CF*, the annotated bipartite graph had an AUC of 0.85 and the interaction profiling bipartite graph had an AUC of 0.85. Figure 14 shows that, for the *Trace CF*, the annotated bipartite graph had an AUC of 0.74 and the interaction profiling bipartite graph had an AUC of 0.80. Figure 15 shows that, for the *Address CF*, the annotated bipartite graph had an AUC of 0.62 and the interaction profiling bipartite graph had an AUC of 1.00. Figure 16 shows the experiment that combined the feature values generated by the *Process*, *Trace*, and *Address CF*s gave AUCs for the annotated bipartite graph and interaction profiling bipartite graph of 0.94 and 1.00, respectively.

Figures 17–20 are the results of experiments based on the ENT_{N2} dataset. Figure 17 shows that, for the *Process CF*, the annotated bipartite graph had an AUC of 0.79 and the interaction profiling bipartite graph had an AUC of 0.97. Figure 18 shows that, for the *Trace CF*, the annotated bipartite graph had an AUC of 0.54 and the interaction profiling bipartite graph had an AUC of 0.75. Figure 19 shows that, for the *Address CF*, the annotated bipartite graph had an AUC of 0.68 and the interaction profiling bipartite graph had an AUC of 0.96. Figure 20 shows the experiment that combined the feature values generated from the *Process*, *Trace*, and *Address CF*s. The annotated and interaction

Table 2: Cross-validation results of MD-Miner^P for different feature vectors of ENT_{N1} and ENT_{N2}.

Dataset	Feature vectors	Precision (%)	Recall (%)	F-measure (%)	Accuracy (%)	AUC (%)
ENT_{N1}	Process feature	97.42	28.41	43.96	95.42	83.93
	Address feature	100.00	99.72	99.86	99.98	99.97
	Trace feature	97.35	21.27	34.88	94.98	79.61
	Lexical feature	64.57	20.20	30.54	94.23	85.89
	Merged feature	100.00	99.85	99.92	99.99	99.97
ENT_{N2}	Process feature	63.19	10.85	17.74	99.82	96.88
	Address feature	100.00	48.05	63.60	99.90	95.86
	Trace feature	84.44	28.90	42.70	99.86	75.29
	Lexical feature	85.83	12.67	21.45	99.83	71.29
	Merged feature	94.04	64.38	75.28	99.93	97.34

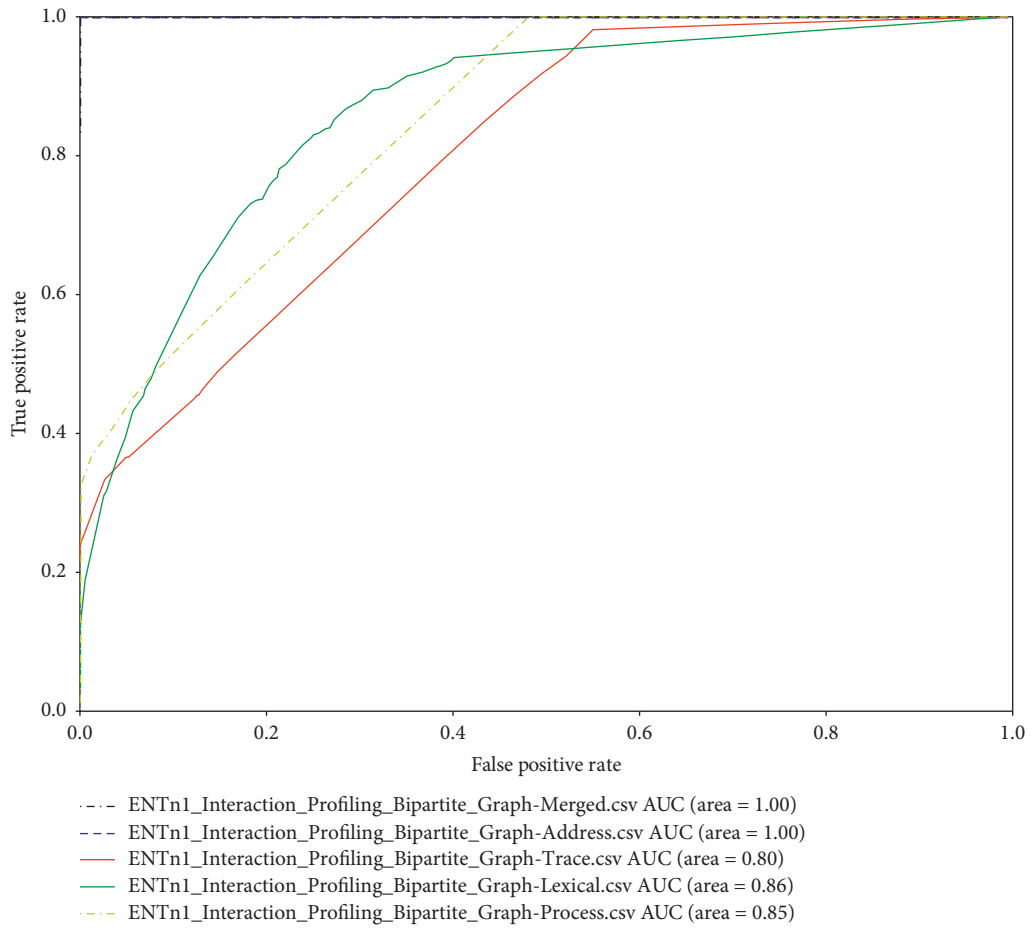


Figure 10: Illustration of the ROC curves for the five feature vectors generated from the ENT_{N1} dataset.

profiling bipartite graphs had AUCs of 0.83 and 0.95, respectively. Table 3 summarizes the AUC value of the feature vector generated by the annotated bipartite graph and interaction profiling bipartite graph in respect of classification assessment, which is used to evaluate the ability of classification.

The experiments in this section show that the data for the proposed interaction profiling bipartite graph are superior to the annotated bipartite graph in either deployed network environments of ENT_{N1} or ENT_{N2}.

4.3. Identified Malicious Domain Analysis. This section demonstrates the effectiveness of the *MD-Miner*^{P} at mining potentially malicious domains. With unknown domains in the ENT_{N2} dataset as the objects for detection, Table 4 shows the top 10 domains with the highest malicious probability as detected with the *MD-Miner*^{P}. These 10 domains were analyzed using the VirusTotal, and four were identified as malicious while the remaining six were classified as clean. Due to the limited space for digital forensics content of the domain name "folder[.]maroon91[.]com," this section

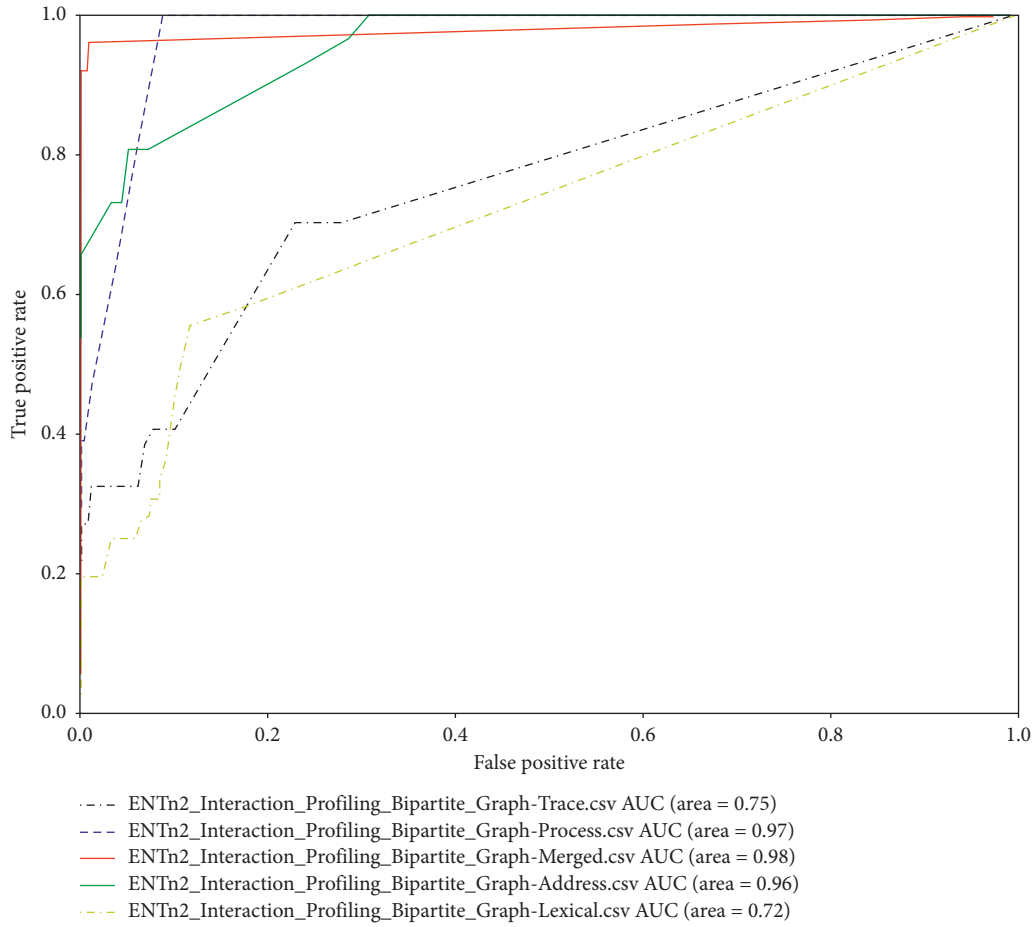


Figure 11: Illustration of the ROC curves for the five feature vectors generated from the ENT_{N2} dataset.

describes the digital forensics as an example. The evidence to classify the domain as malicious is collected from external threat intelligence, including the file communication records and passive DNS records, and the relationship graph regarding the domain based on the collected information is created.

The relationship graph of the domain "folder[.]maroon91[.]com" is constructed using CyberGraph [13], as shown in Figure 21. The figure shows the direct and indirect relationship between the domain and malicious files, malicious IP, and malicious domains. The domain "folder[.]maroon91[.]com" is not a malicious site in VirusTotal, but the malicious files are directly connected to the domain. The domain is then resolved as "221[.]228[.]214[.]69," "58[.]215[.]186[.]83," "118[.]193[.]145[.]130," and "118[.]193[.]187[.]35." The records of these IP addresses that communicate with malicious files are observed from the graph. Moreover, the domains "dsc[.]maroon91[.]com," "app[.]maroon91[.]com," "usjzx[.]maroon91[.]com," "upgrade[.]maroon91[.]com," and "folderhw[.]maroon91[.]com" were discovered to communicate with the malicious files, which have a domain sibling relationship with each other. A series of outward relationships helped identify the domain "folder[.]maroon91[.]com" as malicious. As a consequence, this section demonstrates that the *MD-Miner*^{P} can detect malicious domains that are not recognized by other reputable intelligence systems.

4.4. Performance Evaluation. From a complexity theory viewpoint, the MapReduce framework is unique in that it combines bounds on time, space, and communication. Each of these bounds would be very weak on its own: the total time available to processors is polynomial; the total space and communication are slightly less than quadratic. In particular, even though arranging the communication between processors is one of the most difficult parts of designing a MapReduce algorithm, classical results from communication complexity do not apply since the total communication available is more than linear [41]. Therefore, we use fixed dataset to measure the execution performance and scalability of MapReduce through the execution time of different cluster sizes.

The performance and scalability of the *MD-Miner*^{P} are verified by adjusting the number of nodes in the Hadoop cluster, which were two, four, and six. Each node had 24 CPUs (each is an Intel (R) Xeon (R) CPU E5-2620 2.00 GHz processor) with 32 GB of RAM. The dataset used as a benchmark to analyze the *MD-Miner*^{P} runtime is the ENT_{N2} dataset described in Table 1, which is sized at 172.7 GB.

The flow of the *MD-Miner*^{P} can be divided into three parts: data preprocessing, feature extraction, and domain classification. The feature extraction stage of the *MD-Miner*^{P} can be classified into two parts: interaction profiling bipartite

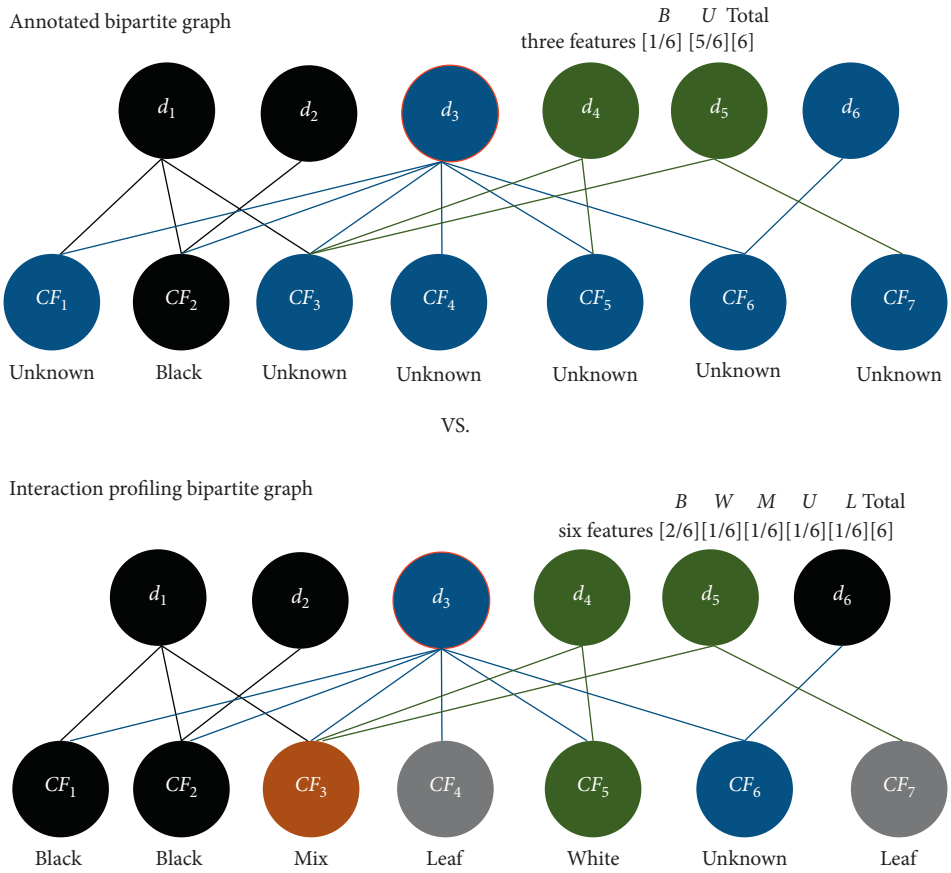


Figure 12: Difference between feature extractions for the annotated and interaction profiling bipartite graphs.

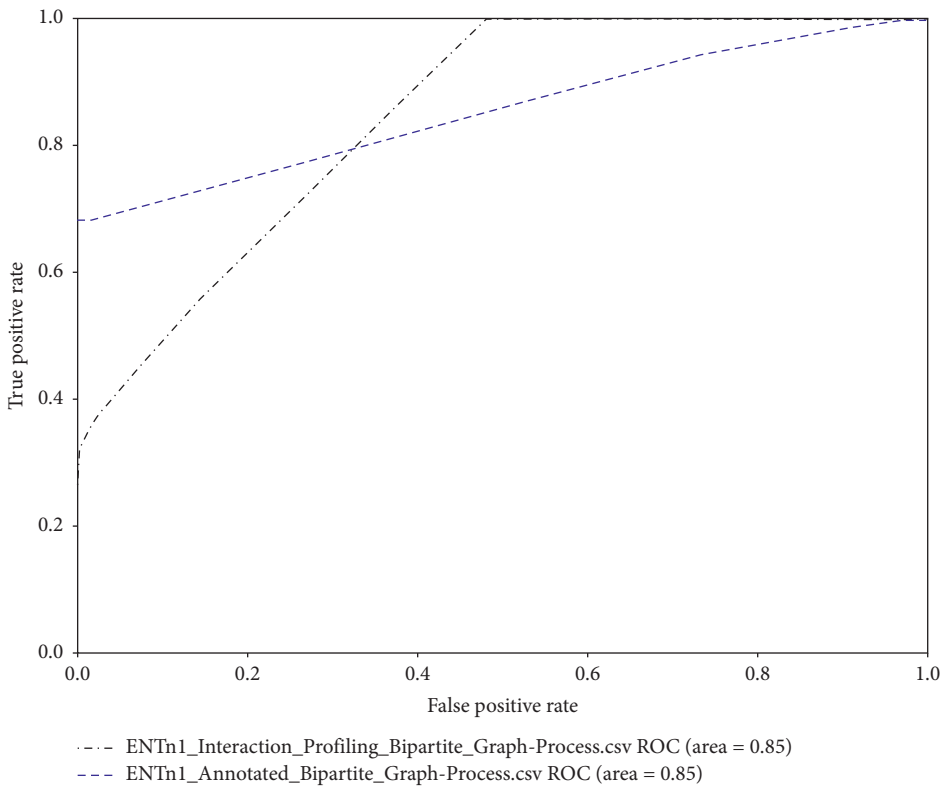


Figure 13: Illustration of the ROC curve for the two *Process* feature vectors generated from two kinds of bipartite graphs for the ENT_{N1} dataset.

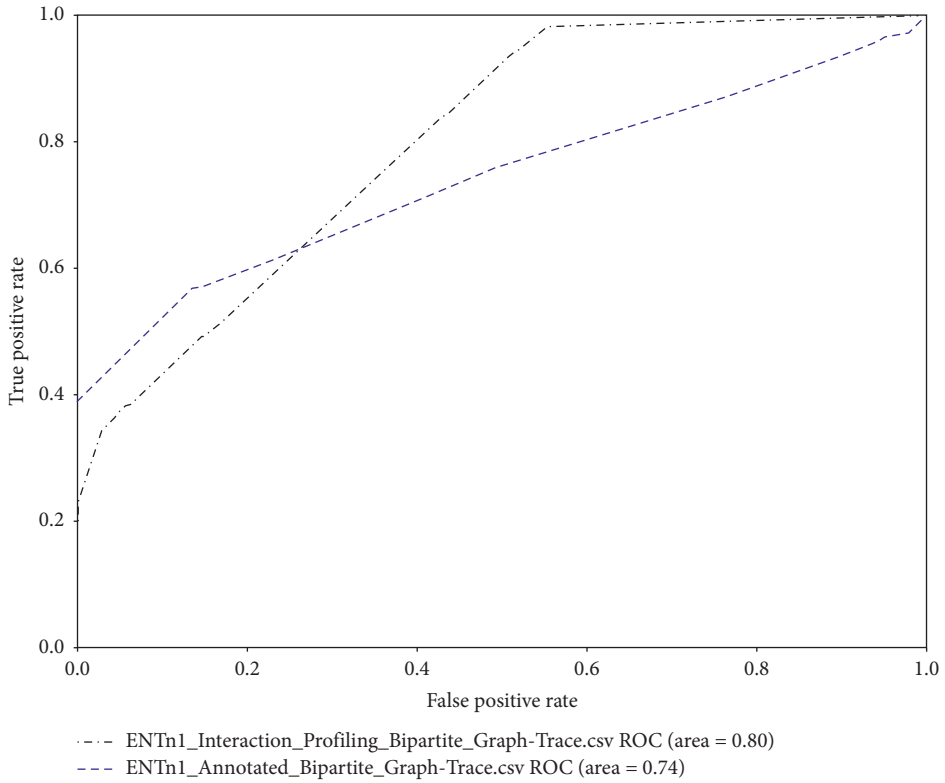


Figure 14: Illustration of the ROC curve for the two *Trace* feature vectors generated from two kinds of bipartite graphs for the ENT_{N1} dataset.

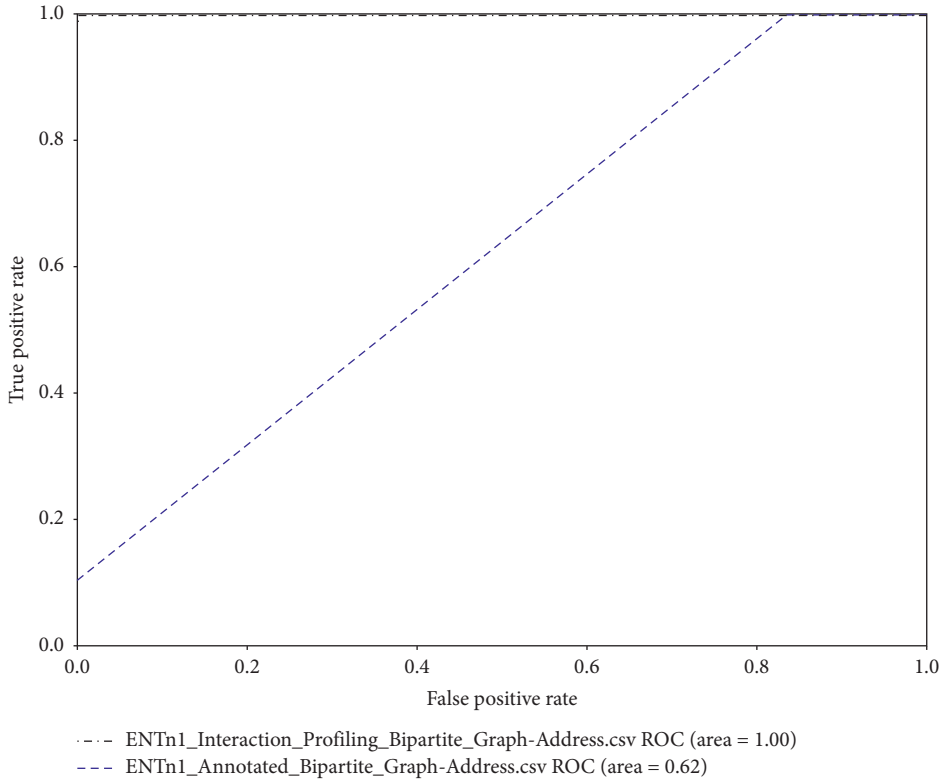


Figure 15: Illustration of the ROC curve for the two *Address* feature vectors generated from two kinds of bipartite graphs for the ENT_{N1} dataset.

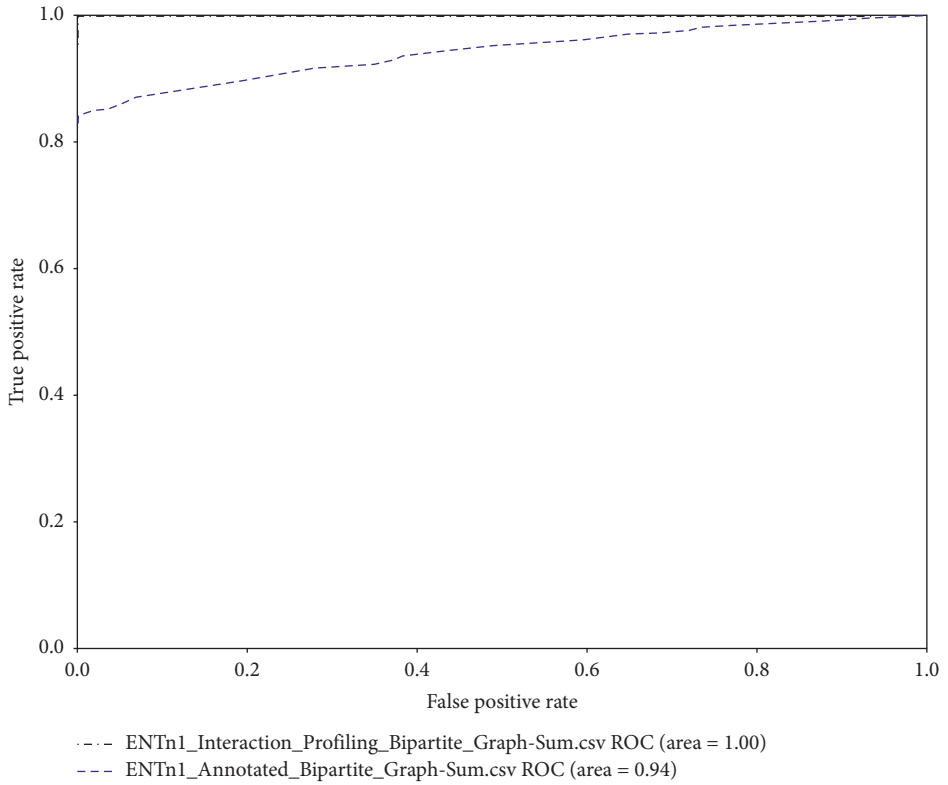


Figure 16: Illustration of the ROC curve for the combined three feature vectors generated from the two bipartite graphs for the ENT_{N1} dataset.

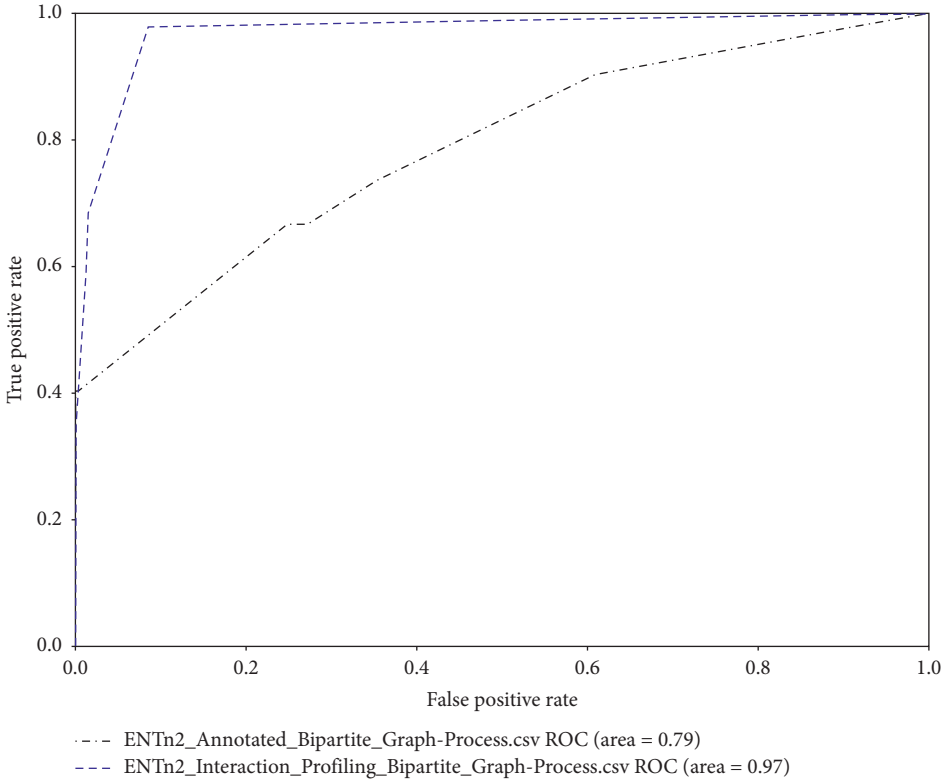


Figure 17: Illustration of the ROC curve for the two *Process* feature vectors generated from two kinds of bipartite graphs for the ENT_{N2} dataset.

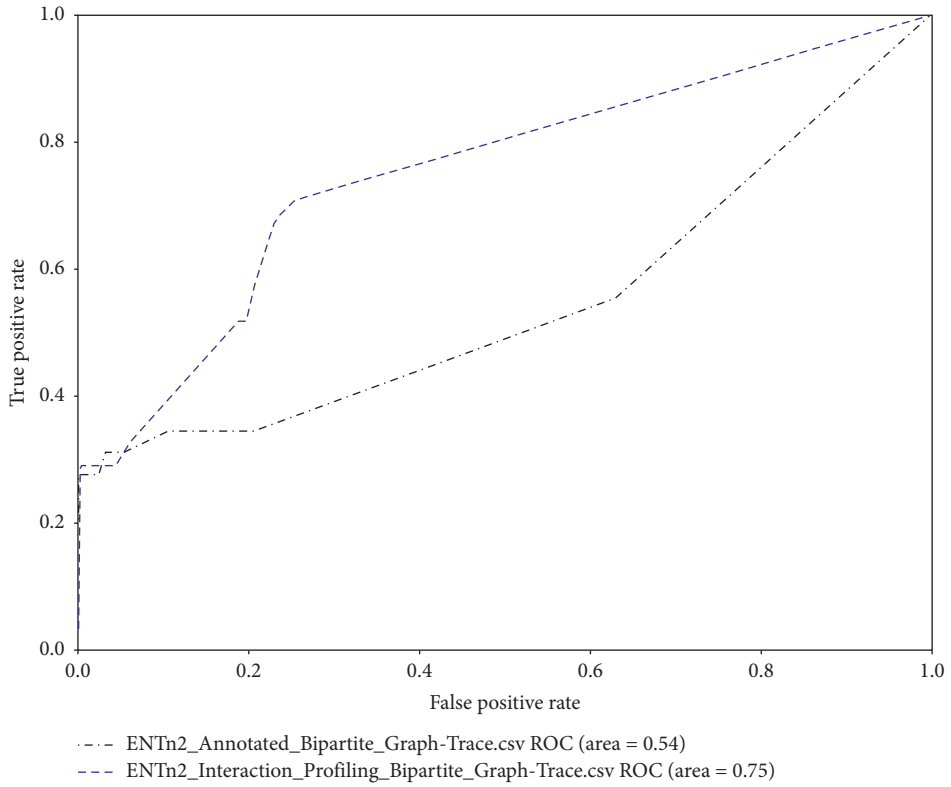


Figure 18: Illustration of the ROC curve for the two *Trace* feature vectors generated from two kinds of bipartite graphs for the ENT_{N2} dataset.

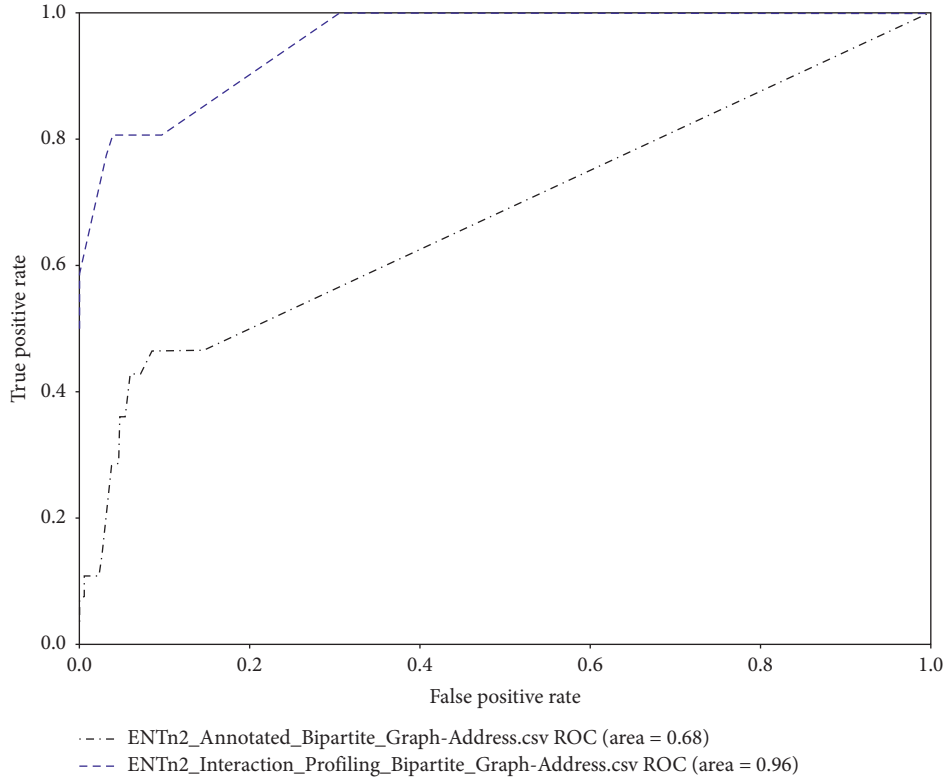


Figure 19: Illustration of the ROC curve for two *Address* feature vectors generated from the two kinds of bipartite graphs for the ENT_{N2} dataset.

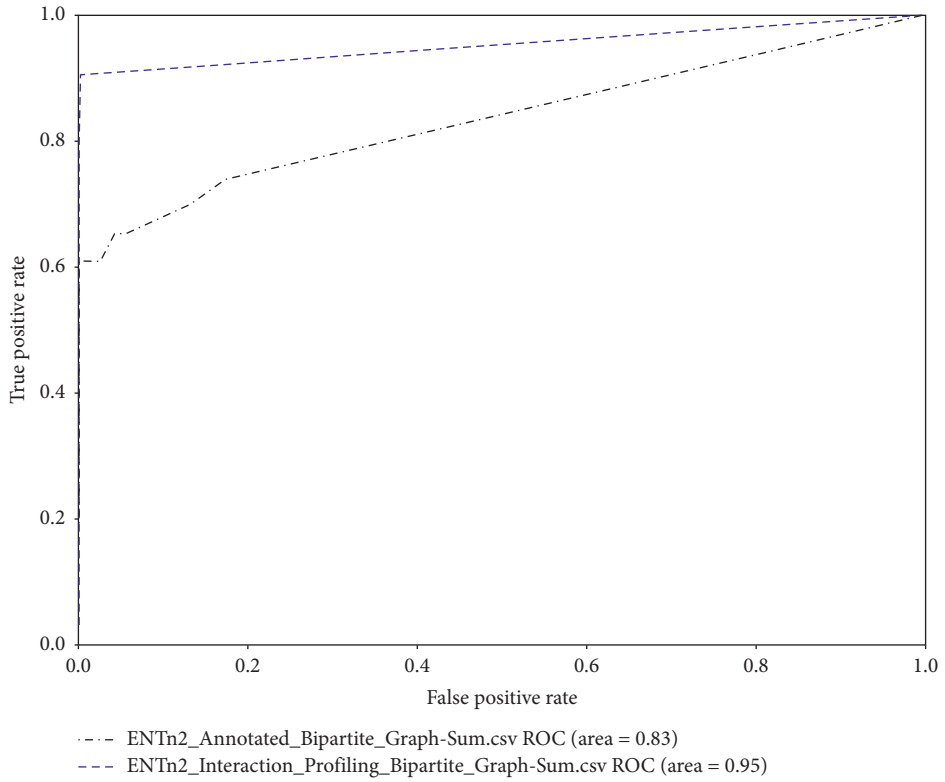


Figure 20: Illustration of the ROC curve of the combined three feature vectors generated from the two bipartite graphs for the ENT_{N2} dataset.

Table 3: Comparison of AUC values generated by annotated bipartite graph and interaction profiling bipartite graph in classification evaluation.

Dataset	Feature vectors	Interaction profiling bipartite graph (AUC) (%)	Annotated bipartite graph (AUC) (%)
ENT_{N1}	Process feature	84.93	85.17
	Address feature	99.97	62.01
	Trace feature	79.61	74.37
	Merged feature	99.97	96.31
ENT_{N2}	Process feature	96.88	79.87
	Address feature	95.86	68.84
	Trace feature	75.29	54.69
	Merged feature	95.34	83.55

Table 4: Top 10 detected malicious domains with higher detected probability.

Domain	Detection probability	Detection rate of VirusTotal
grjxr.snap-affairs.com	0.994728257	0/67
Mdaka.fbhookup.club	0.99454585	0/67
Adserver-g.juicyads.com	0.993425958	0/67
app4.getmacsoft.site	0.991589101	0/67
nofreezingmac.click	0.991547806	2/67
www.por.tw	0.985447549	1/67
extcoolff.com	0.984869145	0/67
urlspirit.spiritsoft.cn	0.981827519	1/67
bak1.spiritsoft.cn	0.981827519	1/67
folder.maroon91.com	0.981491668	0/67

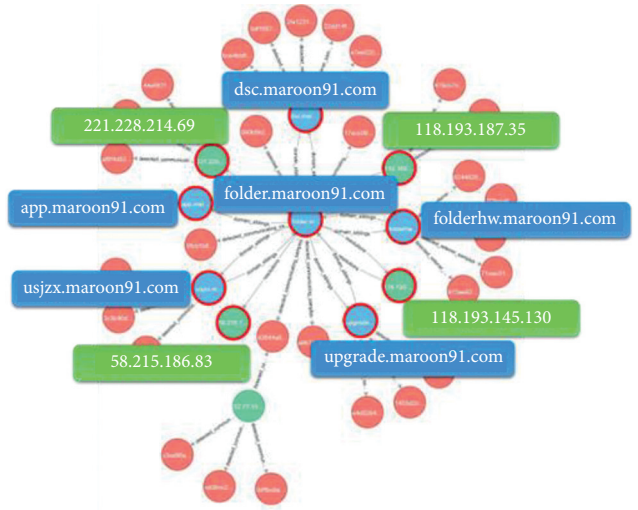


Figure 21: An example of the domain relation graph constructed from external threat intelligence.

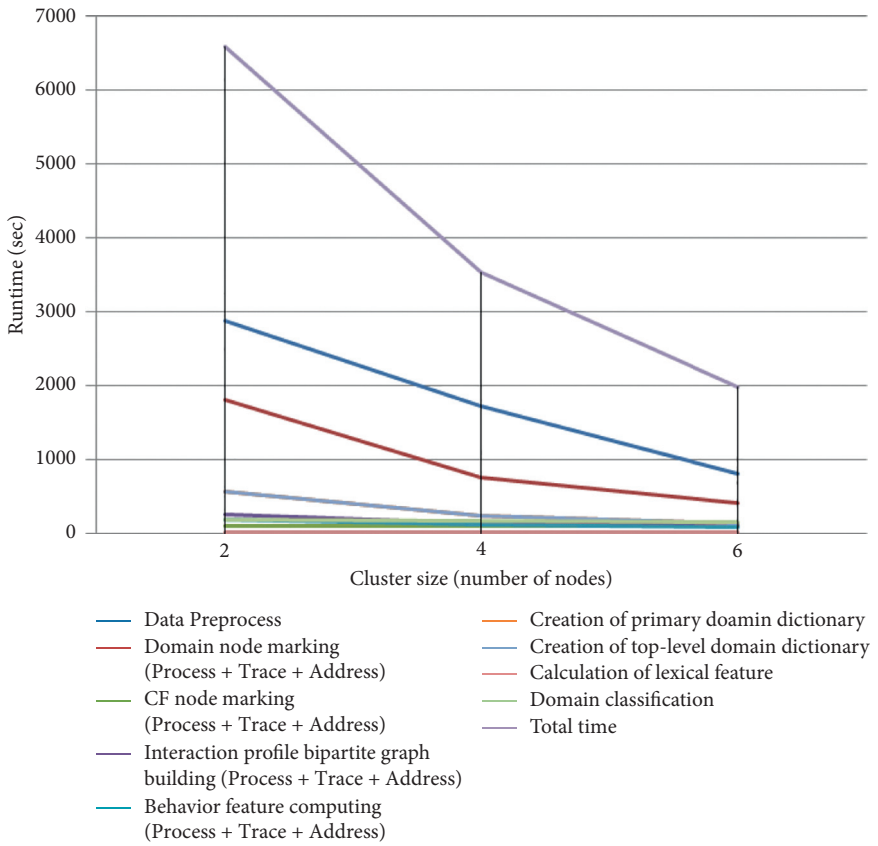


Figure 22: Runtime analysis of the *MD-Miner*^{P}.

graph mining and lexical feature extraction. The *MD-Miner*^{P} designed four MapReduce jobs to accomplish the interaction profiling bipartite graph mining: (1) domain node labeling, (2) *CF* node labeling, (3) interaction profiling bipartite graph building, and (4) behavior feature calculation. The lexical feature extraction can be divided into three MapReduce jobs: (1) creation of primary domain dictionary, (2) creation of top-level domain dictionary, and (3) calculation of lexical features. On the other hand, both the data preprocess stage and domain classification stage have only one MapReduce job. Figure 22 shows the runtime analysis of the *MD-Miner*^{P}. We observe that the data preprocess stage and the domain node labeling of the feature extraction are the primary bottleneck of the *MD-Miner*^{P} process. As the above two jobs

mainly involve I/O operations, the I/O is the primary performance bottleneck in processing the massive data. However, with an increased number of nodes, the computation time of the data preprocess stage and domain node labeling decreases substantially. The experiments show that the *MD-Miner*^{P} tends to possess a superior scalability for the MapReduce.

5. Conclusions

This paper proposes a malicious domain detection system based on a novel bipartite graph called *MD-Miner*^{P}. The interaction profiling bipartite graphs and lexical analysis adopted by the *MD-Miner*^{P} can handle big data. The mining of unknown malicious domains is accomplished by analyzing network interaction behaviors between clients and domains in big network traffic data. The *MD-Miner*^{P} is designed as a scalable system to monitor and analyze big network traffic data to find illegal network activities. Two big network traffic datasets (ENT_{N1} and ENT_{N2}), three validation aspects, and four experiments were proposed to inspect the performance of *MD-Miner*^{P}. The experiments used ROC curves and 10-fold cross-validation with known domains. The experimental results confirm that the feature extraction method proposed by *MD-Miner*^{P} as applied to ENT_{N1} obtained an AUC of 1.00 and applied to the ENT_{N2} obtained an AUC of 0.98. The experimental results of the direct comparison showed that the feature vectors extracted from the interaction profiling bipartite graph are superior to the annotated bipartite graph for both the single and merged feature vectors. In addition, verifying the unknown domain predicted as malicious by the *MD-Miner*^{P} allows the verification method to shape the relationship diagram of the domain. The relationship diagram shows that the domain is directly and indirectly associated with the IP and the domain with malicious behavior. Finally, controlling the number of nodes in the Hadoop cluster verifies that the *MD-Miner*^{P} is a system that fully satisfies the parallel computing conditions, even if the enterprise's network traffic data is large. Therefore, the *MD-Miner*^{P} is applied to conduct malicious domain data mining.

This paper has confirmed the contribution of *MD-Miner*^{P}, but it has some limitations. As described in Section 3, interaction profiling bipartite graph requires domain threat intelligence to label known domain nodes as black and white. Therefore, the quality and quantity of ground truth affect the performance of *MD-Miner*^{P}. Fortunately, collecting public and commercial domain intelligence can effectively overcome this problem. In addition, *MD-Miner*^{P} may not be suitable for DHCP network environment. This is because the proposed bipartite graph uses client-IP to locate individual hosts, and DHCP may cause different hosts to be assigned to the same IP. The solution to this challenge is to correlate DHCP logs with network traffic data to obtain the network behavior of each individual host. The final challenge is that *MD-Miner*^{P} needs to be retrained periodically to maintain detection accuracy. As cybercriminals' technology is constantly evolving, it is necessary to regularly employ *MD-Miner*^{P} and through the latest network traffic data and network threat intelligence to obtain updated domain classification model.

Future work will focus on two areas. First, for detecting malicious domain from big network traffic data, it will be considered whether this approach applies to other large log data, such as firewall and DNS logs. Furthermore, the proposed bipartite graph algorithm can be used to perform correlation analysis for multiple types of network traffic logs to optimize the detection capability. Second, the proposed algorithm is applied to the analysis of other malicious threats. For example, treat the smartphone application's dynamic analysis data (e.g., system call) and static analysis data (e.g., opcode) as *CF*, and match the threat intelligence of applications to build the interaction profiling bipartite graph of applications to mine hidden malicious applications. In addition, the *MD-Miner*^{P} mechanism can be used as the basis for a bilateral market service model [42] to collect malicious traffic. We have provided a website, https://netflowtotal.firebaseapp.com/, to prove this concept [43].

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Impeding malware analysis using conditional code obfuscation," in *Proceedings of the 15th An Network and Distributed System Security Symposium (NDSS'08)*, San Diego, CA, USA, Feburary 2008.

[2] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-version antivirus in the network cloud," in *Proceedings of the 17th An Security Symposium*, pp. 91–106, Boston, MA, USA, July 2008.

[3] A. K. Seewald and W. N. Gansterer, "On the detection and identification of botnets," *Computers & Security*, vol. 29, no. 1, pp. 45–58, 2010.

[4] G. Gu, J. Zhang, W. Lee, and BotSniffer, "Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th An Network and Distributed System Security Symposium (NDSS'08)*, San Diego, CA, USA, Feburary 2008.

[5] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th An USENIX Security Symposium (USENIX Security '08)*, pp. 139–154, Boston, MA, USA, August 2008.

[6] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kir-da, "Automatically generating models for botnet detection," in *Proceedings of the 14th An European Symposium on Research in Computer Security (ESOR-ICS'09)*, pp. 232–249, Saint-Malo, France, September 2009.

[7] H. Binsalleeh, T. Ormerod, A. Boukhtouta et al., "On the analysis of the Zeus botnet crimeware toolkit," in *Proceedings of the 2010 Eighth International Conference on Privacy, Security and Trust*, pp. 31–38, Ottawa, Canada, August 2010.

[8] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: Detecting Botnets Using MapReduce," in *Proceedings of the IEEE International Workshop on Information Forensics and Security (WIFS'11)*, pp. 1–6, Delft, The Netherlands, December 2011.

[9] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control," in *Proceedings of the 31th An. IEEE Conference on Local Computer Networks*, pp. 195–202, Florida, U.S.A, November 2006.

[10] W. Lu, M. Tavallaee, and A. A. Ghorbani, "Automatic discovery of botnet communities on large-scale communication networks," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security - ASIACCS '09*, pp. 1–10, Sydney Australia, March 2009.

[11] R. R. Panko and J. L. Panko, *Business Data Networks and Security*, pp. 145–147, Pearson, Harlow, Essex, 2015.

[12] J. H. Sun, T. H. Jeng, C. C. Chen, H. C. Huang, and K. S. Chou, "MD-Miner: behavior-based tracking of network traffic for malware-control domain detection," in *Proceedings of the 3rd An IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService'17)*, pp. 96–105, San Francisco, CA, USA, April 2017.

[13] T.-H. Jeng, W.-Y. Luo, C.-C. Huang, C.-C. Chen, K.-H. Chang, and Y.-M. Chen, "Cloud computing for malicious encrypted traffic analysis and collaboration," in *Proceedings of the TANET2018*, IJGHPC, Nanjing, China, 2019.

[14] B. Rahbarinia, R. Perdisci, and M. Antonakakis, "Segugio: efficient behavior-based tracking of malware-control domains in large ISP networks," in *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 403–414, Rio de Janeiro, Brazil, June 2015.

[15] S. Y. Chen, T. H. Jeng, C. C. Huang, C. C. Chen, and K. S. Chou, "Doctrina: annotated bipartite graph mining for malware-control domain detection," in *Proceedings of the 7th Ann. International Conference on Communication and Network Security (ICCNS'17)*, pp. 67–75, Tokyo, Japan, November 2017.

[16] R. Holley and D. Rosenfeld, "MAUL: machine agent user learning," 2010, http://cs229.stanford.edu/proj2010/HolleyRosenfeld-MAUL.pdf.

[17] N. Kheir, "Analyzing http user agent anomalies for malware detection," *Data Privacy Management and Autonomous Spontaneous Security,Lecture Notes in Computer Science*, vol. 7731, pp. 187–200, 2013.

[18] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy, "Studying spamming botnets using botlab," in *Proceedings of the 6th Ann. USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*, vol. 9, pp. 291–306, Boston, MA, USA, April 2009.

[19] T. H. Jeng, Y. M. Chen, C. C. Chen, C. C. Huang, and K. S. Chou, "Interaction profiling bipartite graph mining for malicious network activity detection," in *Proceedings of the IEEE Conference on Dependable and Secure Computing (DSC 2018)*, pp. 323–330, Kaohsiung, Taiwan, December 2018.

[20] S.-T. Liu, Y.-M. Chen, and H.-C. Hung, "N-victims: an approach to determine N-victims for APT investigations," *Information Security Applications*, vol. 7690, pp. 226–240, 2012.

[21] En.wikipedia.org. 2019. Advanced Persistent Threat. https://en.wikipedia.org/wiki/Advanced_persistent_threat.

[22] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network Security*, vol. 2011, no. 8, pp. 16–19, 2011.

[23] D. Alperovitch, "Revealed: operation shady RAT," 2011, http://www.csri.info/wp-content/uploads/2012/08/wp-operation-shady-rat1.pdf.

[24] K. F. Hong, C. C. Chen, Y. T. Chiu, and K. S. Chou, "Ctracer: uncover C&C in advanced persistent threats based on scalable framework for enterprise log data," in *Proceedings of the IEEE International Congress on Big Data*, pp. 551–558, New York, NY, USA, June 2015.

[25] K. F. Hong, C. C. Chen, T. T. Chiu, and K. S. Chou, "Scalable command and control detection in log data through UF-ICF analysis," in *Proceedings of the International Carnahan Conference On Security Technology*, pp. 293–298, Quebec, Canada, October 2015.

[26] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th An USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*, pp. 391–404, San Jose, CA, USA, April 2010.

[27] J. Ma, S. Lawrence, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious URLs," in *Proceedings of the 15th An ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*, pp. 1245–1254, Paris France, July 2009.

[28] S. Marchal, J. Francois, R. State, and T. Engel, "PhishStorm: detecting phishing with streaming analytics," *IEEE Transactions on Network and Service Management*, IEEE Transactions), vol. 11, no. 4, pp. 458–471, 2014.

[29] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of http-based malware," in *Proceedings of the 12th Ann. International Conference on Privacy, Security and Trust (PST'14)*, pp. 249–256, Toronto, Canada, July 2014.

[30] D. Chiba, T. Yagi, M. Akiyama, K. Aoki, T. Hariu, and S. Goto, *BotProfiler: Profiling Variability of Substrings in Http Requests to Detect Malware-Infected Hosts*, IEEE Trust-com/BigDataSE/ISPA, Piscataway, NJ ,USA, pp. 758–765, 2015.

[31] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the ACM Workshop on Recurring Malcode (WORM'07)*, pp. 1–8, Alexandria, VA, USA, November 2007.

[32] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *Proceedings of the Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'06)*, pp. 291–302, London, UK, September 2006.

[33] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser, "Detecting spammers with SNARE: spatio-temporal network-level automatic reputation engine," in *Proceedings of the 18th An USENIX Security Symposium (SSYM'09)*, pp. 101–118, Montreal, Canada, August 2009.

[34] M. P. Collins, T. J. Shimeall, S. Faber et al., "Using uncleanliness to predict future botnet addresses," in *Proceedings of the 7th Ann. ACM SIGCOMM Conference on Internet measurement (IMC'07)*, pp. 93–104, San Diego, CA, USA, October 2007.

[35] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *Proceedings of the 19th An. USENIX Conference on Security (USENIX Se-Curity'10)*, p. 18, Washington, DC, USA, August 2010.

[36] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: finding malicious domains using passive dns analysis," in *Proceedings of the 18th Ann. Network and Distributed System*

Security Symposium (NDSS'11), San Diego, CA, USA, Feburary. 2011.

[37] B. Liang, J. Huang, F. Liu, D. Wang, D. Dong, and Z. Liang, "Malicious web pages detection based on abnormal visibility recognition," in *Proceedings of the E-Business and Information System Security (EBISS'09)*, pp. 1–5, Wuhan, China, May 2009.

[38] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious url detection using machine learning: a survey," 2017, http://arxiv.org/abs/1701.07179.

[39] P. Kolari, T. Finin, and A. Joshi, "SVMs for the blogosphere: blog identification and splog detection," in *Proceedings of the AAAI Spring Symposium on Computational Approaches to Analyzing Weblogs*, pp. 92–99, Franco, Egypt, March 2006.

[40] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: understanding, detecting, and disrupting botnets," in *Proceedings of the First Ann. Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI'05)*, pp. 39–44, Cambridge, MA, USA, July 2005.

[41] B. Fish, "On the computational complexity of mapreduce," in *Proceedings of the International Symposium on Distributed Computing*, Delft, The Netherlands, October 2015.

[42] T.-H. Jeng, W.-M. Chan, W.-Y. Luo et al., "A cloud service integration platform for malicious traffic analysis and collaboration," in *Proceedings of the ICCBD*, Taichung, Taiwan, October 2019.

[43] T. Hastie, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, pp. 316-317, Springer, New York, NY, USA, 2009.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 8810708, 12 pages
https://doi.org/10.1155/2020/8810708

Research Article

Automatic Analysis Architecture of IoT Malware Samples

Javier Carrillo-Mondejar, Juan Manuel Castelo Gomez, Carlos Núñez-Gómez, Jose Roldán Gómez, and José Luis Martínez

Research Institute of Informatics (I3A), Universidad de Castilla-La Mancha, Albacete 02071, Spain

Correspondence should be addressed to Javier Carrillo-Mondejar; javier.carrillo@uclm.es

Received 30 March 2020; Revised 16 July 2020; Accepted 7 October 2020; Published 26 October 2020

Academic Editor: Yin Zhang

The weakness of the security measures implemented on IoT devices, added to the sensitivity of the data that they handle, has created an attractive environment for cybercriminals to carry out attacks. To do so, they develop malware to compromise devices and control them. The study of malware samples is a crucial task in order to gain information on how to protect these devices, but it is impossible to manually do this due to the immense number of existing samples. Moreover, in the IoT, coexist multiple hardware architectures, such as ARM, PowerPC, MIPS, Intel 8086, or x64-86, which enlarges even more the quantity of malicious software. In this article, a modular solution to automatically analyze IoT malware samples from these architectures is proposed. In addition, the proposal is subjected to evaluation, analyzing a testbed of 1500 malware samples, proving that it is an effective approach to rapidly examining malicious software compiled for any architecture.

1. Introduction

The appearance of the Internet of Things (IoT) has greatly improved the application of technology in the everyday lives of people. Years ago, digital interaction between an individual and technology was in general only through a computer. With the development of smartphones, that communication became a more mobile, personal, and continuous task. And then, the IoT appeared to change all the previous concepts and insert technology into almost every imaginable object. Smart houses, eHealth, or smart cities are just a few examples of contexts that have their origin in the application of the IoT. Thus, not only has it helped to complement existing scenarios but it has also given rise to the ones in which technology is applied.

As a consequence, the volume of data that is now digitally handled has vastly increased as well. However, although the emergence of the IoT has clearly benefited people, the same positive verdict cannot be passed when speaking of the security measures implemented on the devices. Unfortunately, developers opted to prioritize usability over security, especially during the IoT's conception, when the thought of someone compromising an entire network by simply attacking a switch was unthinkable.

Therefore, there was a huge underestimation of the requirements that these devices and the information that they handle demand. Nowadays, this issue is being acknowledged, and companies are working on improving the protection, but they are still quite vulnerable, added to the fact that a great number of old devices is still being used. This makes the IoT the perfect environment for cybercriminals to operate in. They can gain access to very sensitive and valuable information with little effort. Recent studies [1] show the magnitude of the problem. Only in the first quarter of 2019, a hundred million attacks were detected on smart devices, a figure seven times greater than the number found in 2018. Unsurprisingly, the *Mirai* malware family was behind 39% of them, taking advantage of old devices with unpatched vulnerabilities. Another sample which exploits a trivial attack, namely, the brute-force, *Nyadrop*, closely followed *Mirai* and reached a percentage of 38.57%.

These attacks were the result of poorly designed security measures on the devices and could have been easily mitigated by just changing the default user and password of the device for a more secure one. Instead, they ended up affecting companies such as Twitter, Amazon, Spotify, and Netflix, costing them millions of dollars and affecting their customer's trust [2].

As mentioned above, most IoT attacks do not have their origin in new malware samples, but are based on previous ones that were successful. New versions of old attacks appear every day with minor modifications, but the way they work remains almost identical. Having information about how a sample interacts with the compromised device, and what actions it carries out, allows investigators to protect the device or, at least, limit its expansion over the network. For this reason, the ability to identify which malware samples are alike, that is, those that belong to the same family, can have a huge impact when determining what actions to be taken in order to reduce the impact of a cyberincident.

In addition, besides the existence of multiple operating systems, there are also several architectures used by IoT devices, such as ARM, PowerPC, MIPS, and x86. With the aim of expanding the range over which cybercriminals can carry out their attacks, they develop samples for more than one. This means that numerous pieces of malware have their origin in a sample, and then it is adapted to work on other architectures. Consequently, its behaviour remains similar, with only its structure varying in order to be compatible with them. This allows the malware analyst to analyze malware families independently of the architecture for which the sample was designed.

This analysis is neither a trivial task nor a speedy one. The number of existing samples, added to the appearance of new ones almost every minute, makes it impossible for an investigator to study all of them. Therefore, it is necessary to develop automatic solutions, such as architectures or frameworks, which can speed up the process and be able to examine multiple samples at once. In order to achieve that, a change of approach is needed: instead of focusing on the features that differentiate a sample, now it is mandatory to determine which characteristics allow a piece of malware to be grouped with another, as well as selecting the ones that can be collected and interpreted automatically.

Therefore, the contributions of this study are as follows:

- We study the current state of malware analysis, focusing on the development of automatic solutions to perform examinations
- We present a series of static and dynamic characteristics that are useful to automatically categorize malware samples
- We propose a modular framework for the automatic analysis and clustering of malware samples from the most widely used architectures, based on the evaluation of their static and dynamic features
- We evaluate the proposal with a testbed of nearly 1,500 pieces of malware, confirming its usefulness when analyzing and clustering samples from different IoT architectures

The rest of the paper is organized as follows. Section 2 describes the IoT's architecture, its malware threats, and how to obtain useful characteristics from them. An architecture to automatically cluster malware samples from different IoT architectures is presented in Section 3. An evaluation of the proposal through the analysis of 1500 malware samples is carried out in Section 4. Finally, our conclusions are presented in Section 5.

2. Background

As discussed in the previous section, the IoT environment is the perfect target for cybercriminals to attack. This section presents the problem related to the large number of devices with different architectures connected to the Internet, lists the reasons for the rise of IoT security threats, and defines the concepts of malware analysis and characterization. Then, the Service-Oriented Architecture (SOA) software paradigm used in the design of the framework is introduced. In addition, we present a review of the proposals from the research community in regard to this paper.

2.1. The IoT Environment. The IoT allows developers to model use cases that in the past were not feasible due to the specific limitations of traditional client-server architectures: resource centralization, expensive devices, and high latencies, among others. The IoT environment creates room for new contexts such as Industry 4.0 [3] and smart homes [4]. Its structure can be divided into three fundamental building blocks: the Cloud Layer, the Network Layer, and the Devices Layer. Figure 1 shows the hierarchy formed by these layers. Frequently, end devices interact with other IoT devices as well as with large data centers in the cloud layer to carry out the tasks (sometimes computationally intensive ones) assigned to these end devices. Accordingly, more and more end devices are exposed to the Internet every day, so it is important to adopt appropriate security measures if we do not want to expose our end devices to external attackers.

Another main problem of the IoT environment is the considerable heterogeneity of the devices that comprise it. Although it is important to define security, analysis, and clustering mechanisms against malware layer by layer, our work focuses on the constrained-resource devices of the device layer. These devices are built with different hardware specifications and run different operating systems. One of the most significant specifications is the processor architecture used by such devices. Each processor and its instruction set are designed in a specific way. For example, ARM is a more energy-usage-concerned architecture than x86-64. In our case, the proposed framework focuses specifically on modelling Intel 80386, x86-64, MIPS, ARM, and PowerPC architectures.

2.2. Threats. By scrutinizing the aforementioned recent studies focused on evaluating new trends in IoT malware, a drop in the number of attacks via Telnet can be observed for the second quarter of 2019. Now, the value almost reaches 60%, 20% less than in the previous one. This

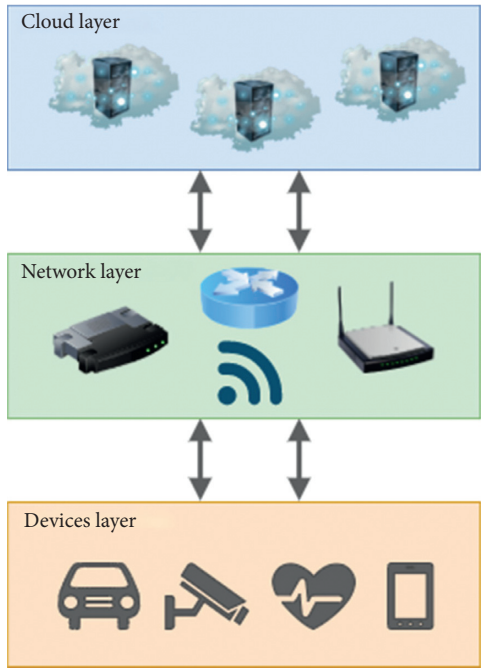


Figure 1: IoT environment architecture.

statistic can be seen as an encouraging one if we deduce that the decrease was due to developers no longer using that service, which is well-known to be deprecated and unsafe. The most worrisome data are that there are few changes in the most common malware families with respect to previous years, meaning that old attacks are still being successful. In addition, the number of malware samples is still growing and expanding into more areas [1]. Some of the main causes of the rapid growth in cybercrime in the IoT are the following:

Number of connected devices: during the year 2020, this figure is forecasted to reach 20.4 billion [5], with 5.8 billion of them being used in the enterprise and automotive market [6]. This means that there are more IoT devices than conventional ones, e.g., smartphones or computers. Therefore, it is preferable for cybercriminals to perform large-scale attacks in this environment rather than in the traditional one, as they can target more victims.

Implemented security measures: as briefly mentioned above, IoT devices can be easily compromised by carrying out simple brute-force or dictionary attacks. This is mainly due to the usage of weak default login credentials. Although it may seem ludicrous, the combination of user and password such as "admin-admin" or "admin-1234" is not that uncommon.

Data handled: the application of the IoT has led to the generation of data that previously did not exist or only did so in a smaller quantity. eHealth is a good example of this circumstance: metrics such as heart rate, blood pressure, or oxygen levels were only stored in special facilities such as hospitals or medical centers and were only available to restricted personnel. Nowadays, these data are also measured and stored by smart watches or smart bracelets that are connected to the cloud and create personal profiles for each user.

Limited computational capacity of the devices: this makes them easy to crash, which is quite convenient when a cybercriminal wants to perform a DoS (Denial of Service) attack. The number of petitions that can be handled by these devices is far more limited than in conventional ones. In addition, it hinders the task of using antiviruses or cryptography algorithms, since the current versions are only supported by more powerful devices.

2.3. Malware Characterization. Characterization can be explained as a process in which a set of features are extracted from someone or something. This makes it possible to describe each item in an unambiguous way. Thus, malware characterization is the process of identifying and extracting these features from each malicious sample. In this field, the characteristics are divided into the following categories:

Static features: here, the focus is on the analysis of the intrinsic characteristics of a binary file without executing its code in the system. Information such as the strings that appear in it, its sections, architecture, opcodes, cyclomatic complexity, or entropy belongs to this category. The main advantage is that static characteristics are quick to extract automatically. On the other hand, the usefulness of the features may be affected if the sample is packed or obfuscated (i.e., disassembly code and strings).

Dynamic features: here, the target is the analysis of the behavior of the sample at runtime by monitoring the different actions that it carries out in the system. The data are extracted from the communication that the malware performs through the network and its interaction with the system, such as system calls or open files, among others. One of its disadvantages is that only characteristics of the executed portions of code are captured, so the criminals include monitoring detection techniques that prevent the sample from executing entirely. In addition, the extraction of dynamic features is more time consuming than the retrieval of static features due to the fact that the sample must be executed for a short period of time.

2.4. SOA. SOA is a software design paradigm in which modules work as independent services providing a specific interface to be called upon. They communicate through an Enterprise Service Bus (ESB) which is formed of one or several protocols, allowing the addition of services with little effort. In order to call each service when it is needed, an orchestration process is used [7]. Under this scheme, it is possible to add new components or new protocols. In addition, this architecture allows the easy integration of multiple SOA-based applications.

2.5. Related Work. As far as the authors are aware, there are no approaches available in the literature that jointly tackle the task of analyzing large numbers of malware samples specifically designed for the IoT and that of classifying or clustering them. On the contrary, most of the approaches try to describe specific malware samples or families, as mentioned in Section 2.5.1. In terms of automatically analyzing a great number of malware samples, there are some articles, but they focus only on Linux-based operating systems for x86 architectures, as is shown in Section 2.5.2. Finally, Section 2.5.3 covers approaches focused on classifying IoT malware, but these do not take into account all IoT architectures or families and neither do they study both static and dynamic features.

2.5.1. Malware Survey. Pa et al [8] presented a Telnet honeypot for different IoT architectures. They conducted a study of the malware that was aimed at this service, showing the problem that it suffers from when it is accessible from the Internet. The authors also presented the first sandbox that supported different architectures and executed the binaries and commands received through their honeypot.

Cozzi et al. [9] presented a complete malware study aimed at Linux-based operating systems. They statically and dynamically analyzed more than 10,000 samples distributed among the main architectures, namely, ARM, PowerPC, and MIPS, among others. They presented the main techniques used by malware and numerically expressed their use in the samples that made up their dataset. To carry out their analysis, they introduced the first malware analysis framework aimed at analyzing Linux-based malware.

Costin et al. [10] introduced a study of 60 families of IoT malware. The authors studied the timeline of events related to each family as well as the most relevant vulnerabilities used by them. For the dynamic analysis, the authors presented a sandbox compatible with the main IoT architectures based on the open source project Cuckoo Box [11].

2.5.2. Linux-Based Sandbox. Limon [12] is a sandbox for analyzing Linux-based malware. It collects calls to the operating system as well as capturing network traffic. Its main problem is that it only supports binary analysis in x86 architectures, and the operating system used to perform dynamic analysis is based on Ubuntu, which is not a very common operating system in the IoT. Similar problems are present in Detux [13], which, although it supports five architectures, is based on the Debian operating system. Detux only performs basic static analysis and network analysis, ignoring malware behavior within the operating system.

Chang et al. [14] proposed a sandbox for analyzing malware samples in the IoT. It is able to collect network packages and malware behavior in the system. To test the functionality of their sandbox, they experimented with the Zollard botnet.

2.5.3. Classification. Nghi Phu et al. [15] presented a framework for analyzing and classifying malware in the IoT. Their framework supports the MIPS architecture and extracts features related to malware interaction with the system in order to train a machine learning model.

Alhanahnah et al. [16] suggested a new approach to classifying IoT malware compiled for different architectures. Its method is based on generating signatures at a high level since these are more robust and vary less between architectures.

Su et al. [17] introduced a method for malware classification in IoT environments. It is based on converting malware into an image and a convolutional neural network for classification. It is able to classify a sample into malware or goodware and recognizes two malware families: Mirai and Gafgyt.

Kumar et al. [18] proposed a new approach to differentiate between malicious and benign applications based on a ranking of permissions used in Android IoT devices. Their methodology included an improvement on the random forest algorithm, achieving an increase in the accuracy of malware detection.

Lei et al. [19] presented a system for malware detection on Android-based IoT devices. They proposed the use of event groups instead of API calls to capture malware behaviour at a higher level than in API level. They trained and evaluated their system with a dataset of around 15,000 and 29,000 benign and malicious Android apps, respectively.

3. Proposed Architecture

This section describes the proposed SOA-based modular framework for analyzing and classifying malware samples from different IoT architectures. It consists of six modules which are invoked as services by the orchestrator of the system, which is responsible for using each module and processing the information extracted in each of the stages. Due to its modular structure, each of the modules that make up the system can be used independently (i.e., deploying a virtual machine to execute commands from a honeypot or even for adding new components). These services use our Enterprise Service Bus (ESB), which allows us to integrate any new component easily. Figure 2 shows a global view of our architecture.

3.1. System Overview. The system uses an executable file from any of the architectures supported as input, analyzes it, and produces a cluster based on the similarity that it has with other previously examined files as output. Although the proposal is designed for malware analysis purposes, it is valid for clustering other types of executables. The following sections describe in detail the modules of which our system is composed.

3.2. The Orchestrator. This is the main module of the system and the one in charge of making the pipeline that interconnects the rest of the modules. Once it obtains a sample, it uses the static analysis module to obtain the information necessary to continue with the next phase. Then, it uses the deployment module to check whether the architecture of the analyzed file is supported, that is, whether there is a virtual

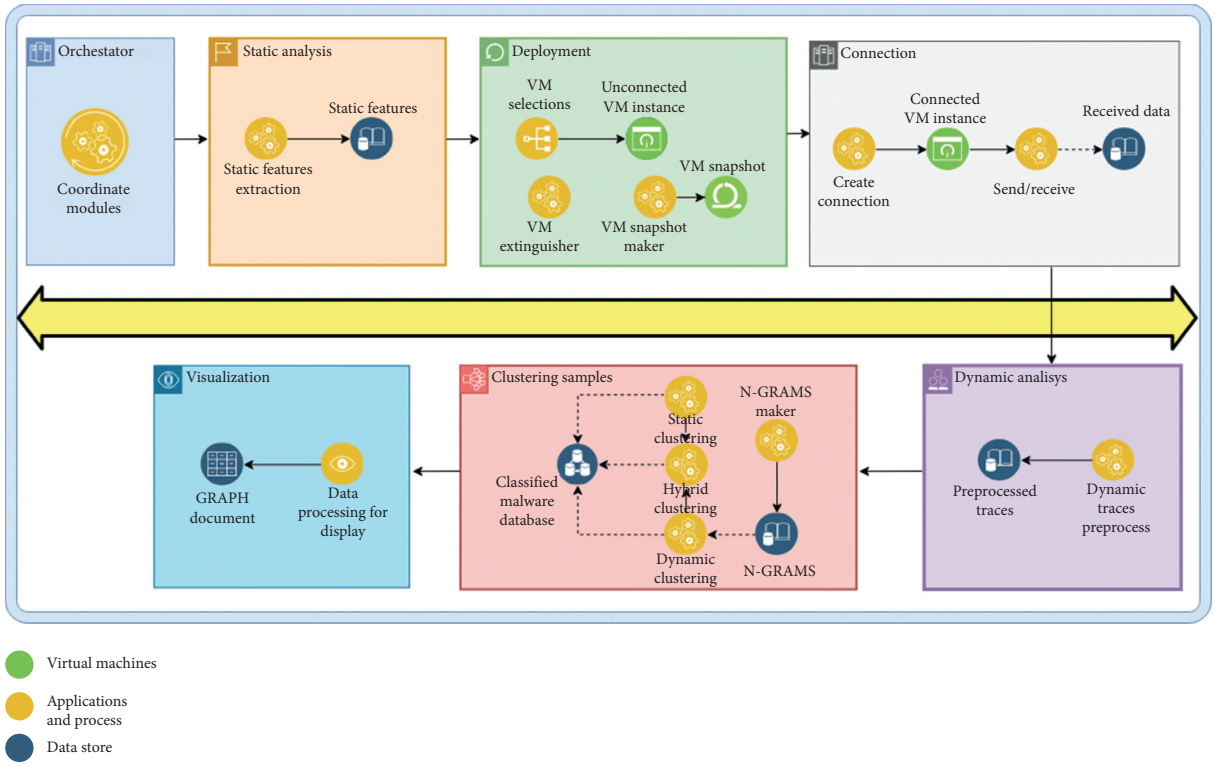


Figure 2: The proposed architecture for the analysis and clustering of IoT malware.

machine that supports that architecture, and if it is, it starts the virtual machine instance.

Once the virtual machine is on, it connects to it through the connectivity module and then proceeds with dynamic analysis, executing the file with the monitoring tool indicated in the configuration files. Then, the file is executed for a certain time which is indicated through the configuration commands of the framework. Once that timeout has elapsed, it obtains the result in the form of execution traces, destroys the virtual machine, and recovers the previous snapshot of the machine.

Finally, it calculates the similarity with other analyzed samples and adds it to the corresponding cluster if the similarity index is greater than the established threshold. The sample will be added to the cluster in which the most similar sample is located. If the threshold is not reached, a new cluster will be created to include the analyzed file.

Additionally, if the display parameter is active, it will calculate the similarity between all the samples and generate a graph connecting all of them.

3.3. Static Analysis. This module is responsible for obtaining and parsing the Executable Linkable Format (ELF) files. It is built upon *radare2* [20], a reverse engineering suite, and automates the process of obtaining information contained in the headers of the ELF files, as well as data regarding their sections. The static analysis module collects the following information.

Information file: characteristics of the headers of the executable file, such as architecture, whether the binary has been stripped of the symbols or not, and whether it was compiled with static or dynamic libraries.

Entropy: this measures the lack of predictability of a data set. In binary analysis, a high entropy value indicates that the sample is obfuscated or packed.

Cyclomatic complexity: this is a metric used in software engineering to calculate, in a quantitative way, the complexity at a logical level of a program or function [21]. Cyclomatic complexity is calculated for each of the functions found in the disassembled code.

Opcodes: the sequence of operation codes (opcodes) of all the functions present in the disassembly of the program are extracted and stored.

Libraries: the name of the shared libraries used by the program.

Sections: the sections into which the executable is divided are extracted, also determining their permissions and entropy.

Functions: the name of the functions imported from the libraries and used by the program.

Strings: all text strings present in the sample.

Hash: the hash to uniquely identify the executable.

3.4. Deployment Module. This module is responsible for starting the virtual machine, shutting it down, or restarting it. Its input is the architecture for which the malware was developed, which is searched for in the library in order to determine whether it can be emulated or not. It uses libvirt

[22] to manage the virtualization platforms and the QEMU [23] emulator as hardware virtualizer. To emulate an architecture, it has to be supported by QEMU, and a guest domain in an eXtensible Markup Language (XML) must be defined. This file contains the configuration of the machine in libvirt, that is, its storage, CPU architecture, kernel image, and network properties. Once the machine has been started, the module returns a handler, which allows you to shut down or restart the machine as well as to see which machines are currently active. Finally, when a machine is stopped, a previous snapshot of the machine is recovered in order to have a malware-free image for the next analysis. In this way, this module provides the flexibility to add user-defined virtual machines and uses them in our framework.

3.5. Connectivity and Dynamic Analysis. This is the module responsible for establishing connection with the virtual machine. It allows the upload and download of files through the Secure Copy Protocol (SCP) and the execution of commands through the Secure SHell (SSH). It provides the flexibility to upload any file type and execute commands in the virtual machine. For example, it can upload an executable file or script and use any type of monitoring tool available in the virtual machine for extracting information about its behavior, such as *strace* [24] or *systemtap* [25]. Finally, download the monitored traces and parses the collected data. The parsing function is responsible for extracting the executed syscalls from the execution traces as well as their parameters and results. Table 1 shows an example of a run sequence and the syscall data.

3.6. Clustering of Samples. This is in charge of clustering the binary files based on some of the previously extracted features. Given two executable files, it calculates the index of similarity between them and, if this is greater than a set threshold (set through the configuration parameters), these samples are considered to be related and, therefore, will be part of the same cluster. To calculate the similarity, the module uses the following approaches:

Dynamic approach. We use the execution traces obtained in the dynamic analysis to generate sequences of *syscall* names of size n (set through the configuration parameters), which are known as n-grams. An example for a sequence of size $n = 4$ is shown in Table 1, resulting in the following set of n-grams: (*brk, socket, fcntl64*, and *fcntl64*), (*socket, fcntl64, fcntl64,* and *setsockopt*), and (*fcntl64, fcntl64, setsockopt,* and *brk*). In order to determine the similarity, we use the Jaccard index [26] as a metric, which, for two sets of n-grams, is calculated as

$$\text{jaccard}\left(s_1, s_2\right) = \frac{\left|s_1\right| \cap \left|s_2\right|}{\left|s_1\right| \cup \left|s_2\right|}, \tag{1}$$

Table 1: Format execution trace.

Syscalls	Parameters	Results
Brk	0x32000	0x32000
Socket	AF_INET, SOCK_RAW, IPPROTO_TCP	0
fcntl64	0, F_GETFL	0x2
fcntl64	0, F_SETFL, O_RDWR|O_NONBLOCK	0
Setsockopt	0, SOL_IP, IP_HDRINCL, [1], 4	0
Brk	0x33000	0x33000

where the numerator indicates the number of unique subsets that are present in both sets, and the denominator indicates the total number of unique subsets between s_1 and s_2. The result is a value between 0 and 1 which indicates the degree of similarity between two sets of n-grams.

Static approach. We use two metrics to measure the similarity between two executable files. The first is based on sequences of opcodes of size n extracted from the disassembled code. This is calculated in the same way as in the dynamic approach but using opcodes instead of *syscalls*. The second is based on the cyclomatic complexity of each of the functions present in the disassembled binary. A distance function is used for the calculation of the similarity between two executable files. This function is formalized as follows:

$$\text{distance}\left(s_1, s_2\right) = \sum_{i=0}^{|F|} \frac{\min\left(f_i^{s_1}, f_i^{s_2}\right)}{\max\left(f_i^{s_1}, f_i^{s_2}\right)} x \frac{1}{F}. \tag{2}$$

For example, let us consider two executables with five and seven functions, the first with cyclomatic complexities 3, 5, 3, 7, and 4 and the second with complexities, 3, 3, 6, 6, 4, 5, and 2. The first sample has two functions with cyclomatic complexity 3, one with 5, one with 7, and another with 4. In the second sample, we have two functions with cyclomatic complexity 3, two with 6, one with 4, one with 5, and another with 2. We normalize the vectors so that they have the same number of elements, and the vectors (0, 2, 1, 1, 0, 1) and (1, 2, 1, 1, 2, 0) are obtained. Therefore, the similarity index between the two vectors is 0.5 and is calculated as follows: ((0/1 + 2/2 + 1/1 + 1/1 + 0/2 + 0/1)/6).

Hybrid approach. The hybrid approach allows clustering using the indexes described above. To do this, it assigns a weight to each of the indexes to calculate the final similarity index. The weight of each index can be configured in the framework configuration files.

3.7. Visualization. Its function is to visually represent the groupings generated based on the approaches described above. We denote f as a function that defines whether two malware samples are similar or not using the following expression:

$$f\left(x = \text{metric}\left(s_1, s_2\right)s_1, s_2\right) = \begin{cases} 1, & x \geq z \\ 0, & x < z \end{cases}; z \quad x \in [0,1]; s_1, s_2 \in D, \tag{3}$$

where z being the selected threshold for determining the similarity between two samples, namely, s_1 and s_2, both belonging to the dataset of samples, which is defined as D. It generates a graph file in *dot* format [27] in which the nodes represent the executable files, and an edge between two nodes represents the fact that between them there is a similarity greater than the established threshold. The generation of the graphs is computationally expensive since it calculates the similarity for each different pair of samples.

4. Experiments and Results

In this section, the experiments and results obtained using our malware analysis and clustering framework are presented.

4.1. Overview. In order to test the platform described in Section 3, we built different custom virtual machines using *buildroot* [28], which automates the process of building an embedded Linux system. In total, we built machines for the five most widely used architectures in the current IoT market, namely, Intel 80386, x86-64, MIPS, ARM, and PowerPC, generating a file system and a compilation of a kernel image for each one. We used *strace* as a monitoring tool to obtain the execution traces.

To perform the analysis, we used different samples of Linux-based malware which targets IoT devices. The samples are distributed among the five architectures mentioned. The malware samples are labeled using AVClass [29], which categorizes them using a ranking of the labels provided by different antivirus engines. Table 2 summarizes the number of pieces of malware used for each architecture and how many of them are packed and labeled.

Finally, we used our framework to analyze all the samples and visualize the relationships between them according to the metrics described in Section 3.4. The following sections show the results obtained after analyzing the entire set of samples described above in terms of static and dynamic points of view.

4.2. Static. In this section we present the results of the analysis and clustering processes using the static features described in Section 3. We use a threshold, which can be adjusted by the user, of 0.8 to determine whether two samples are related for both metrics. This value selection is based on an empirical study which is out of the scope of this paper.

4.2.1. n-grams. We use the n-grams of the operation codes extracted in the static analysis process. The size was empirically determined to be four by using cross validation. Since the operation codes are architecture dependent, we generated clusters for each of the architectures independently. Figure 3 shows the graphic for all architectures in the study, namely, MIPS, PowerPC, x64, x86, and ARM. The nodes represent malware samples and the edges indicate whether there is a similarity greater than 0.8 at the n-gram level. Gray is used to represent malware samples that do not have a label and the rest of the colours represent each of the families that have been labeled (*AVClass*) in the dataset. As can be seen, there are different clusters formed mainly of samples from the same family. In some cases, there are related samples from several families. This may be because some of the samples are packed and, if they use the same packer, they may share the same code routines to unpack the executable at run time. One of the disadvantages of using static features is that they can be affected by code obfuscation. This metric can also be affected depending on whether the executable is compiled with static linking or with dynamic linking, since those binaries compiled with static linking could have more unique n-gram sequences because the functions imported from the libraries are included in the binary itself. In general terms, the proposed architecture detects well the families of malware samples for all the architectures.

4.2.2. Cyclomatic Complexity. We use cyclomatic complexity to cluster the samples. Since the metric is extracted from disassembled programs and depends on the assumptions of the compiler and the assembly code that it generates, we cluster the samples for each of the architectures independently. This is because, after looking at several executable files available for different architectures (e.g., *busybox*), we observe that the cyclomatic complexity for the same functions varies according to the architecture. Although it is not very different between one and the other, it does change even if they have been compiled with the same compilation options. Figure 4 shows the graph for all the architectures used in this paper. As we can see, the clusters generated belong to the same family, and there are several small clusters for the same family, such as *Gafgyt, Tsunami*, *or Mirai* for the ARM architecture. This is due to the fact that this metric measure similarity at a structural level between two samples. Therefore, it can also be affected by obfuscated code. In addition, if a sample is compiled in a static way and another in a dynamic way, there will not be a structural similarity between them (those compiled with static linking have imported library functions within the executable instead of being resolved at runtime as in binaries compiled with dynamic linking).

Observing the graphs generated for both metrics (Figures 3 and 4), it can be seen that, in general, the clusters created using n-grams are made up of more samples than those produced using cyclomatic complexity. In either case, most of the connected samples are related to others from their own family without producing many false positives.

4.2.3. Dynamic. In this section, we present the results obtained in the clustering process using the dynamic characteristics extracted in Section 3.5 and the metric described in the same section. As was done in Section 4.2.2, we use a threshold of 0.8 to match two malware samples. We use sequences of n-grams of size four for the *syscalls* executed for

Table 2: The number of malware samples distributed for each of the architecture.

Arch	Samples	Packed	Labeled
Intel 80386	279	58	211
X86-64	344	168	134
MIPS	318	63	288
ARM	246	24	200
PowerPC	275	12	258
	1462	325	1091

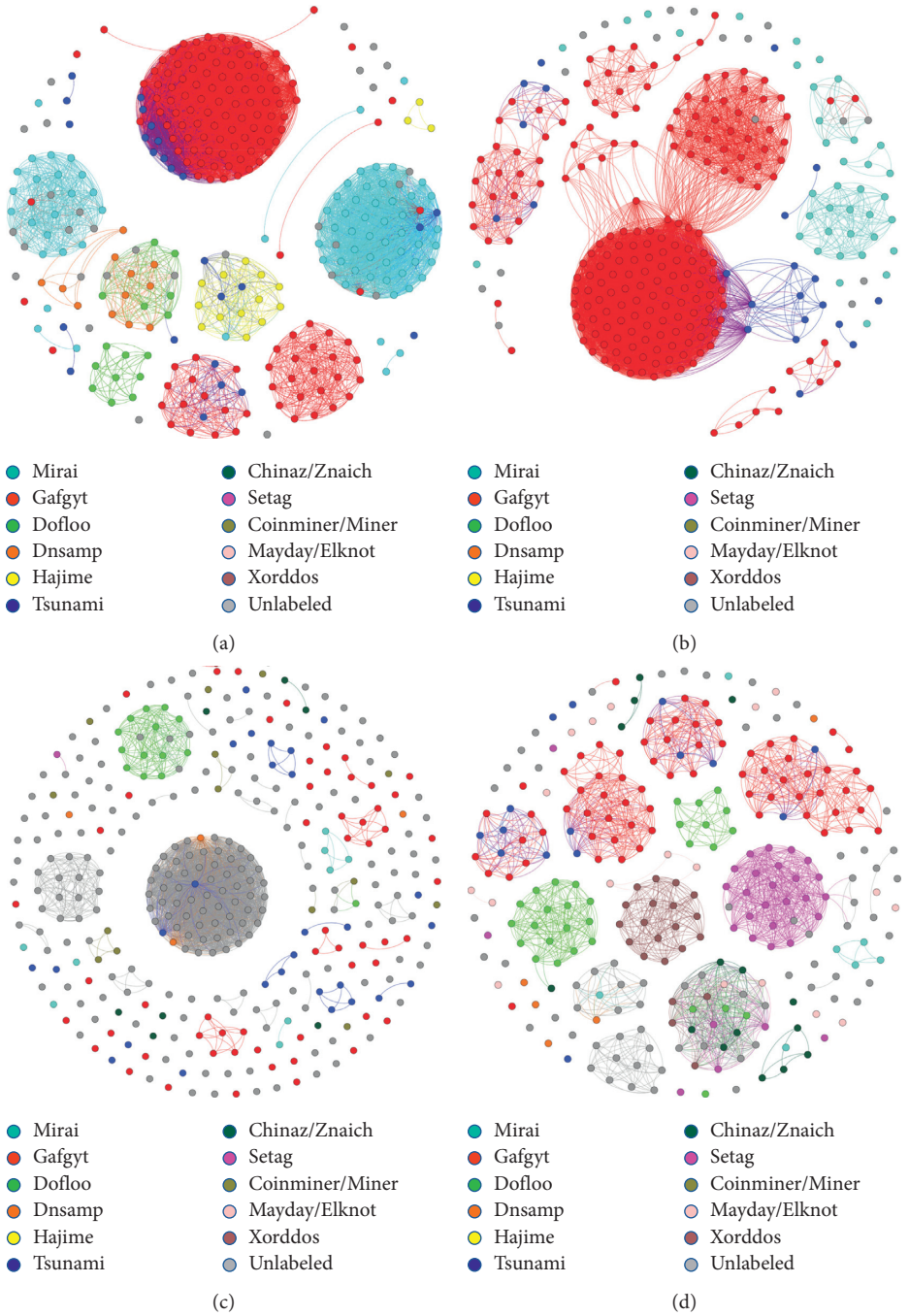
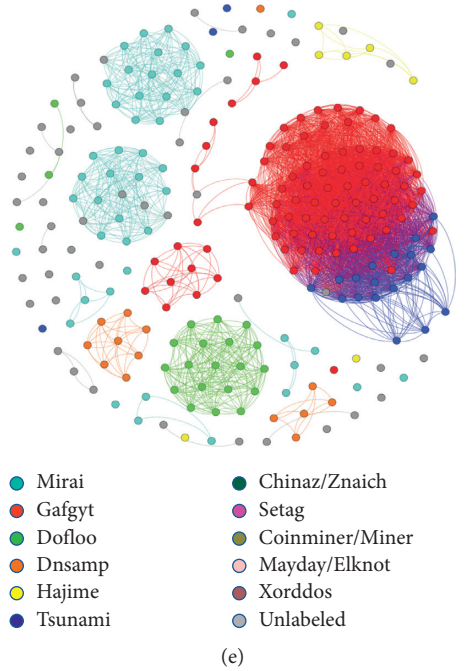
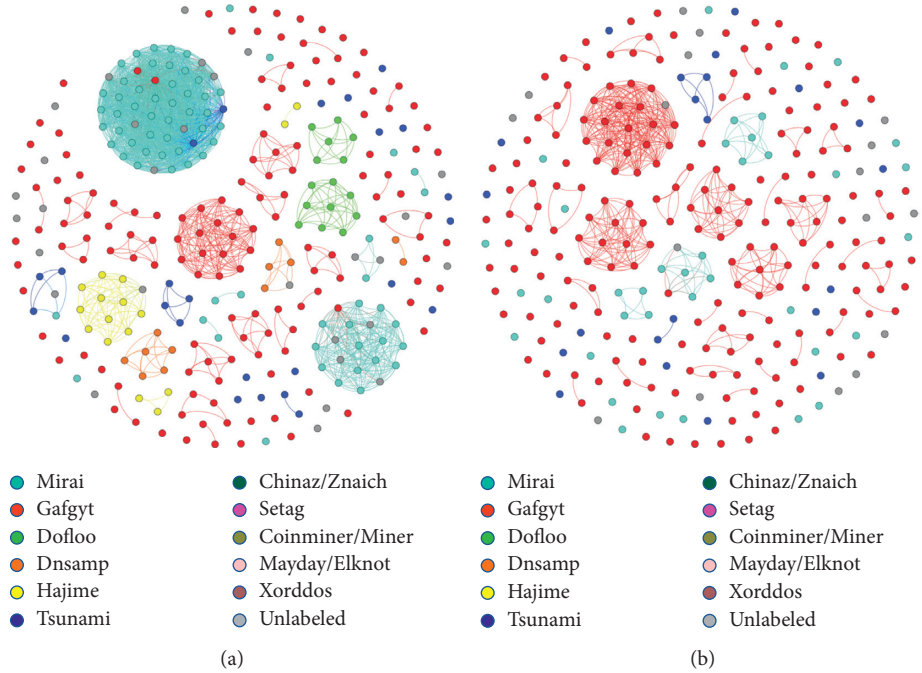


Figure 3: Continued.



(e)

Figure 3: Clusters generated for the MIPS (a), PowerPC (b), x64 (c), x86 (d), and ARM (e) architectures using n-grams and the Jaccard index to calculate the similarity.



(a) (b)

Figure 4: Continued.

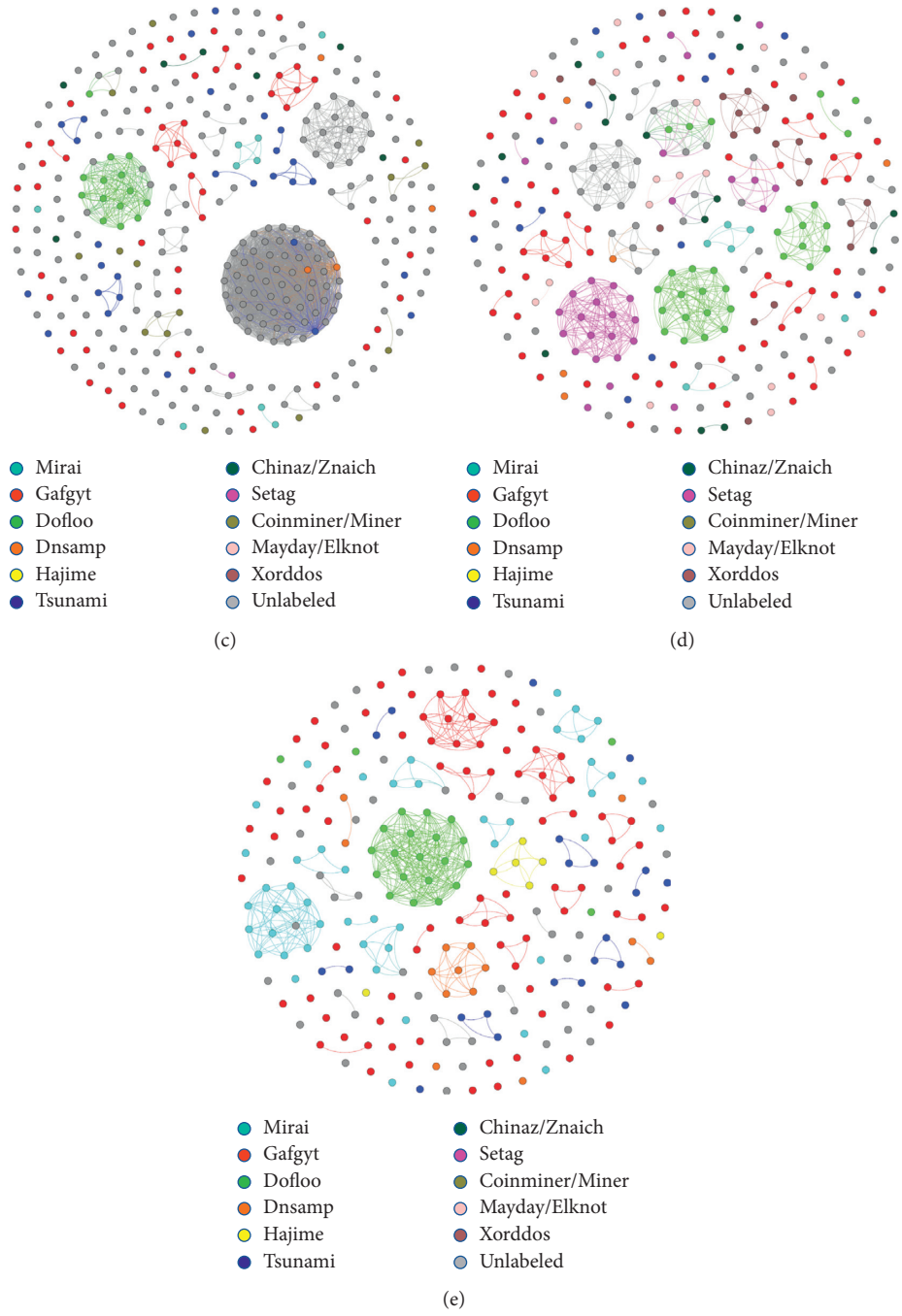


Figure 4: Clusters generated for the MIPS (a), PowerPC (b), x64 (c), x86 (d), and ARM (e) architectures using cyclomatic complexity and the custom function described in Section 3.

each of the samples. Since the *syscalls* are petitions to the operating system to request a service (e.g., create a socket and kill a process), and these have the same name in any Linux-based operating system, using them for clustering allows us to find similarities between the execution traces of samples from different architectures.

Figure 5 shows the clusters generated using the *syscalls* traces as features. On the left, each sample is colored depending on the architecture to which it belongs. On the right, each sample is colored depending on the family to which they belong, with gray indicating the unlabelled ones. It can be observed that there are clusters that are formed of samples from different architectures, such as MIPS, PowerPC, and Intel 80386. If we observe these same clusters in the family-categorized image, it can be seen that the samples belong to a particular malware family. In addition, it can be noticed that the clusters are made up of samples from the same family, and that, based on their behavior, pieces of malware from different architectures have been categorized into the same cluster.

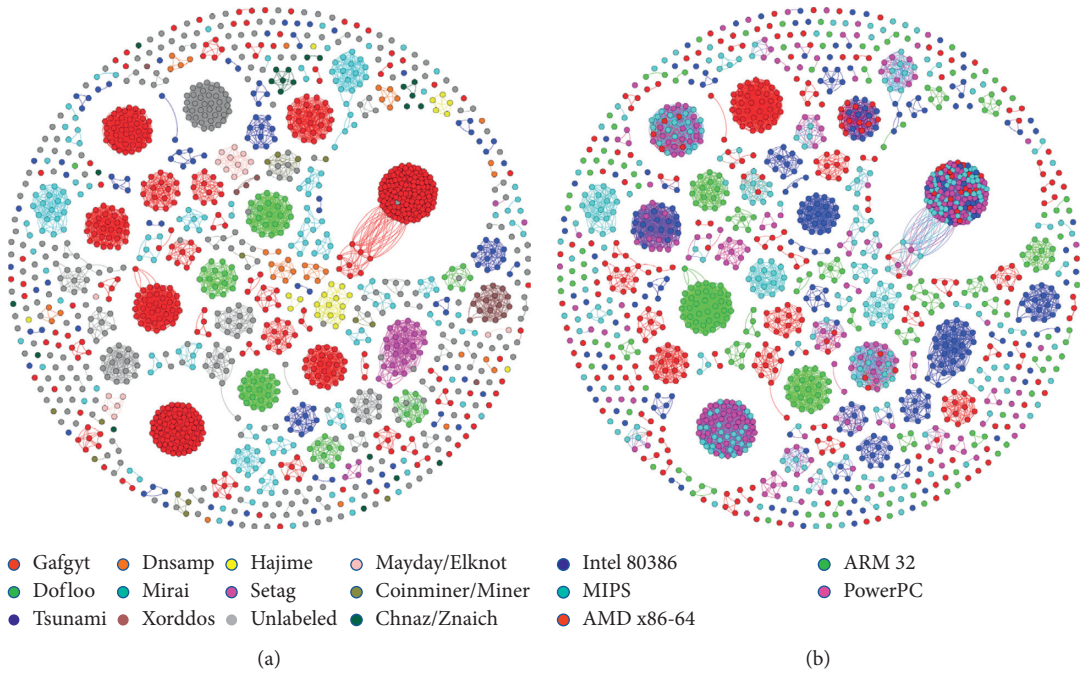


FIGURE 5: Clusters generated for all architectures using the execution traces obtained in the dynamic analysis. The *n*-gram size used for the *syscalls* sequence is four. The edges connect those samples with a similarity index greater than 0.8.

Finally, we observe that there are different clusters for the same family. Unlike the previous case, in which the samples may appear different depending on the architecture for which they were compiled or the different compilation options, now it may indicate that they belong to different campaigns of the same family. Malware is constantly evolving, and its creators add new functionalities or use existing ones from other pieces of malware that have proven effective and beneficial. Also, it should be noted that the original source code of some of the most widely used malware families is available on the Internet, such as *Gafgyt* or *Mirai* [18], and there may be variants created by different authors.

5. Conclusions

In this proposal, we have addressed IoT malware analysis, focusing on the automatization of the examining process. Our motivation for this is the huge increase in cyberattacks that have been carried out in this environment over recent years, which has led to the impossibility of manually studying the samples as the number is too immense. After evaluating the proposals from the community, it has been observed that there were none that focused on both analyzing (statically and dynamically) a large number of IoT malware samples at once and providing compatibility with several architectures.

Consequently, a multiarchitecture framework for automatic malware analysis and clustering has been presented. The proposal, which is based on a modular approach and supports samples from five different IoT architectures, namely, ARM, PowerPC, MIPS, Intel 8086, and x64-86, is able to extract static and dynamic features from a sample and compare it with previous analyzed ones, categorizing it into families depending on the similarity. In addition, besides saving a considerable amount of time when examining pieces of malware, it offers flexibility to the user, allowing them to define their own emulated architectures and to adapt the threshold used to determine whether a sample is categorized into a family or not.

The proposal has been evaluated through the examination of nearly 1,500 malware samples from the five architectures that are supported by the framework, offering promising results and proving its effectiveness when clustering malware samples. Especially relevant is the outcome of the dynamic analysis, in which the proposal has been able to cluster samples from multiple malware campaigns, even if they were designed for different architectures. In addition, it has been detected that, when clustering using the static features, samples may appear different depending on the architecture for which they were compiled or the different compilation options. Other factors, such as code obfuscation, also hinder the task, although the results generated by the static analysis are also satisfactory.

Given the good results offered by the framework when tested and knowing the importance of improving the analysis of malware samples, there are several lines of research that could be followed to complement this proposal. Some such projects could be to

Study the network communications made by the malware samples when they are executed and use them as a feature to cluster them

Expand the visualization features, offering the user an interactive representation of the results, allowing them to directly browse through the different samples or filter them by selecting certain characteristics.

Add other IoT architectures so that samples designed for them could also be examined.

Employ other metrics to determine sample similarity, and even to use advanced machine learning techniques to add a layer of intelligence to the framework.

Data Availability

The sample data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, the JCCM under the project SB-PLY/17/-180501/-00035, the Spanish Education, Culture and Sports Ministry under grants FPU 17/03105 and FPU 17/02007, the University of Castilla-La Mancha under the contract 2018-PREDUCLM-7476 and the project 2020-GRIN-28846, and the Spanish State Research Agency under the project PEJ2018-003001-A.

References

[1] D. Demeter, M. Preuss, and Y. Shmelev, "IoT: a malware story-securelist," 2019.

[2] The Council of Economic Advisers - United States of America and CEA Report, *The Cost of Malicious Cyber Activity to the U.S. Economy*, The Council of Economic Advisers, Washington, DC, USA, 2018.

[3] E. L Xua and L. Ling, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, pp. 2941–2962, 3 2018.

[4] P. P. Gaikwad, J. P. Gabhane, and S. S. Golait, "A survey based on smart homes system using internet-of-things," in *proceedings of the 2015 International Conference on Computation of Power Energy, Information and Communication (ICCPEIC)*, Piscataway, NJ, USA, 2015.

[5] Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017 https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016.

[6] Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020 https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io.

[7] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *The Organization for the Advancement of Structured Information Standards*, vol. 12, 2006.

[8] Y. M. P. Pa, S. Suzuki, K. Yoshioka et al., "IoTPOT: a novel honeypot for revealing current IoT threats," *Journal of Information Processing*, vol. 24, no. 3, pp. 522–533, 2016.

[9] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *in proceedings of the 2018 IEEE Symposium on Security and Privacy*Security and Privacy, Francisco, CA, USA, July 2018.

[10] A. Costin and J. Zaddach, "IoT malware: comprehensive survey," *Analysis Framework and Case Studies*, BlackHat, Las Vegas, NV, USA, 2018.

[11] C. Guarnieri, "Cuckoo sandbox-automated malware analysis," 2016, https://cuckoosandbox.org/.

[12] https://github.com/monnappa22/Limon.

[13] detuxsandbox/detuxhttps://github.com/detuxsandbox/detux.

[14] K.-C. Chang, R. Tso, and M.-C. Tsai, "IoT sandbox: to analysis IoT malware zollard," in *Proceedings of the Second International Conference on Internet of Things*, Data and Cloud Computing, New York, NY, USA, September 2017.

[15] T. N. Phu, K. H. Dang, D. N. Quoc, N. T. Dai, and N. N. Binh, "A novel framework to classify malware in mips architecture-based IoT devices," *Security and Communication Networks*, vol. 2019, Article ID 4073940, 13 pages, 2019.

[16] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, "Efficient signature generation for classifying cross-architecture IoT malware," in *Proceedings of the 2018 IEEE Conference on Communications and Network Security (CNS)*, Beijing, China, June 2018.

[17] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, July 2018.

[18] R. Kumar, X. Zhang, R. U. Khan, and A. Sharif, "Research on data mining of permission-induced risk for android IoT devices," *Applied Sciences*, vol. 9, 2019.

[19] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: event-aware android malware detection against model degrading for IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668–6680, 2019.

[20] R. Team, *Radare2 Book*, 2017.

[21] A. H. Watson, D. R. Wallace, and T. J. McCabe, "Structured testing: a testing methodology using the cyclomatic complexity metric," *US Department of Commerce, Technology Administration, National Institute of Standards and Technology*, vol. 500, 1996.

[22] libvirt The virtualization API https://libvirt.org/.

[23] Q. E. M. U. 2020, https://www.qemu.org/.

[24] strace trace system calls/signals, https://linux.die.net/man/1/strace.

[25] SystemTap, https://sourceware.org/systemtap/.

[26] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, Cambridge university press, Cambridge, UK, 2014.

[27] Graphviz-Graph Visualization Software, http://www.graphviz.org/documentation/.

[28] Buildroot-making embedded Linux easy, https://buildroot.org/.

[29] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "A tool for massive malware labeling," in *Proceedings of the International Symposium on Research in Attacks Intrusions, and Defenses*, Paris, France, September 2016.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 6724513, 11 pages
https://doi.org/10.1155/2020/6724513

Research Article

SLAM: A Malware Detection Method Based on Sliding Local Attention Mechanism

Jun Chen,[1] **Shize Guo**,[1] **Xin Ma**,[1] **Haiying Li**,[2] **Jinhong Guo**,[2] **Ming Chen**,[2] **and Zhisong Pan**[1]

[1]*Command and Control Engineering College, Army Engineering University of PLA, Nanjing 210007, China*
[2]*University of Electronic Science and Technology of China, Chengdu 611731, China*

Correspondence should be addressed to Zhisong Pan; hotpzs@hotmail.com

Received 31 December 2019; Revised 14 June 2020; Accepted 29 August 2020; Published 25 September 2020

Academic Editor: Yin Zhang

Since the number of malware is increasing rapidly, it continuously poses a risk to the field of network security. Attention mechanism has made great progress in the field of natural language processing. At the same time, there are many research studies based on malicious code API, which is also like semantic information. It is a worthy study to apply attention mechanism to API semantics. In this paper, we firstly study the characters of the API execution sequence and classify them into 17 categories. Secondly, we propose a novel feature extraction method based on API execution sequence according to its semantics and structure information. Thirdly, based on the API data characteristics and attention mechanism features, we construct a detection framework SLAM based on local attention mechanism and sliding window method. Experiments show that our model achieves a better performance, which is a higher accuracy of 0.9723.

1. Introduction

The rapid development in computers and Internet technology is also coupled with rapid growth in malicious software (malware). Malware such as viruses, Trojans, and worms also changed expeditiously and became the most severe threat to the cyberspace. Malware is usually installed and operated on a user's computer or other terminal without user's permission, which infringes on the legitimate rights and interests of users. It gains control over computer systems through changing or malfunctioning normal process execution flow.

According to the latest China Internet Security Report 2018 (Personal Security Chapter) released by 360 Security in April 2019, 360 Internet Security Center intercepted 270 million new malicious program samples on PC in 2018, with an average of 752,000 new malicious program samples on PC everyday [1]. In addition, malware often uses confusion, encryption, deformation, and other technologies to disguise itself in order to avoid being detected by antivirus software. Such a large number of malware and complex countermeasure technologies have brought serious challenges to network security.

To face these challenges, researchers conduct a series of studies. They use static analysis, dynamic analysis, and hybrid analysis for executable files, and extract a series of features, which includes Opcodes, API calls, and binaries. After that, they take machine learning to construct the detection model and achieve good results. However, in reality, deep learning in machine learning area is especially worth focusing on, due to its powerful expression ability.

Up to now, malware detection methods based on deep learning mainly focus on image [2], signal [3], and Application Programming Interface (API) sequence [4]. It may not stimulate the potential ability of deep learning model if we just simply transform malware into an input vector. How to effectively use expert knowledge to process data, transform it into the input needed by deep learning model, and design a specific deep learning model are the key to improve the effectiveness of deep learning model in detecting malware area.

We have noticed that, in the field of machine learning, the attention mechanism has been used very successfully, especially in the fields of Natural Language Processing (NLP), image, and machine Q and A. Attention mechanism has significantly improved performance, which demonstrates the powerful ability of deep learning in solving practical problems. For example, the latest XLNet model [5] builds a content-based and context-based attention mechanism by using a two-stream attention mechanism. However, the seq2seq problem and the malware classification are still different. How to effectively transfer the attention mechanism originated from translation problems to the field of malware classification according to practical problems is a subject worth exploring.

In this paper, we firstly analyze the attributes of the APIs and further divide them into 17 categories. Based on the category, we construct semantic and structure-based feature sequences for API execution sequences. Then, according to this feature sequence, we design a sliding local attention mechanism model SLAM for detecting malware. The experimental results show that our feature extraction method is very effective. Our contributions are as follows:

(1) Analyze the characters of the API execution sequence and classify the APIs into 17 categories, which provides a fine-grained standard to identify API types

(2) Implement a 2-dimensional extraction method based on both API semantics and structural information, which enhances a strong correlation of the input vector

(3) Propose a detection framework based on sliding local attention mechanism, which achieves a better performance in malware detection

The remaining of the paper is organized as follows. Section 2 is a brief background on malware classification. The detailed API execution sequence portrait is explained in Section 3.1. Also, the detailed attention mechanism is explained in Section 3.2. Data source and experimental results are discussed in Section 4. Section 5 summarizes the paper and outlines future work.

2. Related Work

The field of malicious code classification and detection is currently divided into traditional methods and machine learning methods. The traditional methods rely on a large amount of expert knowledge to extract the malicious features by reverse analyzing the binary code to achieve the purpose of classification and detection [6, 7]. Features extracted by manual analysis are highly accurate. However, this requires a considerable amount of manpower [8, 9].

As the malicious virus grows exponentially, the way of extracting features by manual analysis is becoming more and more expensive for this situation. Machine learning methods are highly generalized and do not require much manual work. Machine learning, because of its powerful learning ability, can learn some feature information that cannot be extracted manually. However, these methods based on machine learning are very susceptible to interference. Some existing methods, such as converting malicious code into pictures and signal frequency [2, 3], which ignore the original semantics of the code, are easily interfered. As long as the malicious code author adds some byte information or modifies the distribution of the file, the classifier can be confused. Venkatraman and Alazab [10] use the visualization of the similarity matrix to classify and detect zero-day malware. Visualization technology helps people to better understand the characteristics of malicious code, but they have not explored the application of deep learning.

In the work of [6, 11, 12], they use the ASM file generated by disassembly to convert the assembly bytecode into pixel features and then use CNN to learn. Although this method takes advantage of some program information, malware authors can still make confusion by inserting external assembly instructions. Zhang et al. [13] use SVM to build a malicious code detection framework based on semi-supervised learning, which effectively solves the problem that malicious code is difficult to be marked on a large scale and has achieved good results. There are also some methods that are based on API calls in [14]. They treat the file as a list containing only 0 or 1, with 0 and 1 representing whether or not the associated API appears. Their experiments show that the Random Forest classifier achieves the best result. This method mainly relies on the malicious API which could be emerged on a series of call sequence, and only the exact execution sequence can make damage on the computer system.

In the work of [15], they construct behavior graphs to provide efficient information of malware behaviors using extracted API calls. The high-level features of the behavior graphs are then extracted using neural network-stacked autoencoders. On the one hand, their method of extracting behavioral graphs is very precise and helps to express the true meaning of the program fragments. On the other hand, their input vectors are constructed based on the whole sample, and the output of the model is the classification result of the whole sample. In fact, malicious fragments are only partial, which makes the malicious behavior graph easy to be overwhelmed.

Liu et al. [4] use image texture, opcode features, and API features to describe the sample files. By using the Shared Nearest Neighbor (SNN) clustering algorithm, they obtain a good result in their dataset. Qian and Tang [16] analyze the API attributes and divide them into 16 categories. They propose a map color method based on categories and occurrence times for a unit time the API executed according to its categories. Then, they use the CNN model to build a classifier. Xiaofeng et al. [17] propose a new method based on information gain and removal of redundant API

fragments, which effectively reduce the length of the API call sequence. The handled API call sequence is then entered into the LSTM model for training. Uppal et al. [18] use call grams and odds ratio to select the top-ranked feature segments, which are used to form feature vectors and are used to train the SVM model.

On the one hand, the above methods based on the API execution sequence are accurate, which reflect the dynamic execution information of the program. However, on the other hand, due to program execution control, in a long execution sequence, the actual malicious execution code is very small or overwhelmed by a large amount of normal execution code. If the model does not learn the key malicious information, it will easily be bypassed by malicious code specifically disguised. There are also other machine learning methods to learn the features. Ma et al. [19] analyze the local maliciousness about malware and implements an anti-interference detection framework based on API fragments, which can effectively detect malware. Anderson and Roth [20] offer a public labeled benchmark dataset for training machine learning models to statically detect malicious PE files. While they complete baseline models based on gradient boosted decision tree model without any hyperparameter optimization, it will still help researchers study further in this field.

In the work of [21], they extract features based on the frequency of the API and compare neural networks with other traditional machine learning methods. In the work of [22], the implemented Markov chain-based detector is compared with the sequence alignment algorithm, which outperforms detector based on sequence alignment. In the work of [23], they represent the sequences of API calls invoked by Android apps during their execution as sparse matrices and use autoencoders to autonomously extract the most representative and discriminating features from these matrices, which outperform more complex and sophisticated machine learning approaches in malware classification.

These methods expand the space for extracting malicious features and improve the applicable scale of the machine learning method, which achieve good results. However, they also have some limitations, mainly reflecting in the following aspects. Firstly, manual methods have high accuracy but require a lot of manpower, which make them unsuitable for analyzing a large amount of malicious code. Secondly, machine learning is greatly influenced by the training set and its practicality is weak. For example, we have performed an experiment, in which an image-based malware classifier can achieve 0.99 accuracy rate. However, after changing dataset, its performance drops sharply to about 0.73. Thirdly, when the sample is confused, the training model is difficult to achieve good results.

In fact, no matter if it is converted to images [24], signals, frequency, and other characteristics, it cannot truly express malicious code. The method of extracting more efficient sequences by n-gram slicing [25, 26] only retains the sequential features of malicious code execution. The models trained with the features extracted by the common methods will have a poor effect.

Therefore, it is worth in-depth and long-term research to explore how to design a detection framework with the help of prior knowledge of malware so that we can apply deep learning to malware detection better. Recently, the XLNet model [5], which employs attention mechanisms, has achieved remarkable success in NLP, translation problems, and machine question and answer. It indicates that there is a new stride on deep learning. In response to this situation, for exploring, we further study how to apply attention mechanism in the field of malware classification.

3. Our Method

We first analyze the attributes of the API and divide APIs into 17 categories based on its functionality and official definition. After that, we analyze the characteristics of the attention mechanism and construct a sliding local attention mechanism model (SLAM) according to our data characteristics.

3.1. API Analysis. The API we studied here mainly refers to the system call function under Windows system. According to the Windows official document, the total number of Windows API is more than 10,000, but most API functions are not frequently used.

We firstly extract the 310 most commonly used API from our dataset and then classify them according to their functional characteristics and their harm to the system, which is different from the work of [16]. By studying its harm to the system, we could be better at representing the structural information for the API execution sequence. Finally, we divide these API into 17 categories and colored them, as shown in Table 1, which make the structural information more intuitive.

Based on our classification of API categories, we can represent an API execution sequence as an API category call sequence, which helps us to look at the API execution sequence from a higher API category perspective. Thus, it can be used to represent the structure information of the API execution sequence, which will help us get the information of the API execution sequence from a higher perspective. Here, we can think that it has obtained structural information for the API call sequence.

3.2. Attention Mechanism Analysis. The attention mechanism is a deep learning model which is mainly used in computer vision and NLP. Especially, employed into the complex NLP field, such as machine translation, reading comprehension, machine dialogue, and other tasks, the attention mechanism model can fully demonstrate its learning ability. In essence, the attention mechanism imitates on the processing of the human brain, that is, mainly

Table 1: API category classification description.

Category	Index	Colored	Description
Undefine	0		Undefined API in category dictionary
Net	1		API related to network operations includes socket, wsa, etc.
File	2		API related to file operations includes read, write, copy, etc.
Process	3		API related to process operations includes thread, process, etc.
Reg	4		API related to registry operations.
Device	5		API related to device operations includes mouse, keystone, etc.
Cert	6		API related to cert operations includes encrypt, decrypt, etc.
System	7		API related to system operations includes dll, error, etc.
Service	8		API related to services operations
Window	9		API related to window operations includes findwindow, drawwindow, etc.
Memory	10		API related to memory operations includes readmemory, writememory, etc.
Privilege	11		API related to privilege operations
Com	12		API related to com operations includes createinstance, etc.
Message	13		API related to message operations includes sendmessage, recieve, etc.
Debug	14		API related to debugger operations
Shell	15		API related to shell operations
Data	16		API related to data operations includes buffer, etc.
Session	17		API related to session operations includes encrypt, decrypt, etc.

focuses on some key part from the massive input information. The attention mechanism can be described by the following formula:

$$\text{Attention}(Q, K, V) = F(Q, K)V. \tag{1}$$

In this formula, Q represents a query vector and K and V represent a set of key-value pairs. Through this formula, we can query the weight value of Q in the global context. Since different q value correspond to different weight values, it achieves the purpose of paying attention to the key parts. Recent popular deep learning models, such as BERT [27] and XLNet [5], are based on attention mechanisms and are successfully applied on the NLP field, which demonstrate their powerful machine learning capabilities. Because of the existence of context in NLP and the problem of out-of-order in sentence, it will greatly restrict the effectiveness of some deep learning model. In response to this problem, XLNet uses a two-stream attention mechanism to extract key values from both a content and context perspective, thereby it significantly improves performance. This is instructive for us to apply attention mechanism on the field of malware classification. We will explore the application of attention mechanisms according to the characteristics of malware.

3.3. Detection Framework. Based on both the API and attention mechanism analysis in the previous section, we will build our own feature extraction methods and build targeted detection framework. The whole process is divided into 4 parts: data processing, feature extraction, model construction, and result output.

3.3.1. Data Processing. We use the Cuckoo software [28] to build a virtual sandbox that captures the sequence of API calls for executable programs. We then collect all the APIs that appeared in the sample and build an API dictionary to map the API to a unique number by using word2vec [29]. Through this conversion, an API call sequence can be converted into a number sequence. The process can be defined as follows. Define transferAPI function, which can be used to obtain the API's number according to the API dictionary:

$$\text{API\_num} = \text{transfer API}(\text{API}). \tag{2}$$

3.3.2. Feature Extracting. We select 310 API which are frequently used by the samples and divide them into 17 categories. Then, based on the frequency of the category

Table 2: API execution sequence transfer description.

	7	GetSystemTimeAsFileTime	1
	10	NtAllocateVirtualMemory	11
	10	NtFreeVirtualMemory	8
	10	NtAllocateVirtualMemory	11
	10	NtAllocateVirtualMemory	11
	10	NtAllocateVirtualMemory	11
	10	NtAllocateVirtualMemory	11
	7	SetUnhandledExceptionFilter	13
	7	LdrLoadDll	4
	7	LdrLoadDll	4
	7	LdrGetProcedureAddress	23
	7	LdrUnloadDll	5
	13	NtCreateMutant	25
	10	NtCreateSection	41
	10	NtMapViewOfSection	78
	7	LdrLoadDll	4
	7	LdrGetProcedureAddress	29
	7	LdrUnloadDll	5
	13	NtCreateMutant	6
	10	NtCreateSection	9
	10	NtMapViewOfSection	24

tags appeared, a category dictionary is built so that the category can be uniquely represented as a number. Through the category dictionary, we can convert an API execution sequence into a number sequence. Because this number sequence contains the category information of the API execution sequence, it can be used to represent the structural information of the API execution sequence. For example, we obtain an API execution sequence by Cuckoo sandbox (Virus Share 0a83777e95be86c5701aa-ba0d9531015 from virus share website [30]). Then, through the category mapping, we can get its category call sequences, as shown in Table 2. The process can be described as follows.

Firstly, we define transferToAPICategory function, which can be used to obtain the API's category by category dictionary.

Secondly, we define indexAPICategory function, which can be used to obtain the index of the API category.

Then, we can get that

$$\begin{aligned} c_i &= \text{transfer To API Category (API)}, \quad \text{where API} \in \text{API\_Set}, \\ i &= \text{index API Category}, \quad \text{where } (c_i) \in \text{API Category}. \end{aligned} \tag{3}$$

Furthermore, we can construct a two-dimensional input vector as shown below. Define findIndex function, which is used to obtain the index of the API according to the category dictionary. Then, we can get that

$$\begin{aligned} \text{Input\_Vector} &= \langle \text{API-Sequence}, \text{Category-Sequence} \rangle, \\ \text{Category\_num} &= \text{find Index (API)} = \text{index API Category}(c_i), \\ \text{API\_Sequence} &= \langle \text{transfer API}(\text{API}_1), \ldots, \text{transfer API}(\text{API}_{2000}) \rangle, \\ \text{Category\_Sequence} &= \langle \text{find Index}(\text{API}_1), \ldots, \text{find Index}(\text{API}_{2000}) \rangle. \end{aligned} \tag{4}$$

If the length of the sequence is not enough 2000, then 0 is added; otherwise, it is truncated. Through these operations, we can extract two-dimensional input vectors.

3.3.3. Model Construction. According to the characteristics of the API execution sequence with length of 2000, several adjacent API calls actually have practical meaning, that is, the entire API execution sequence has certain local significance. Therefore, we design a local attention mechanism to acquire the features of these adjacent APIs with local significance. Furthermore, drawing on the idea of CNN, a sliding window method in a certain step size is used to scan the entire API execution sequence. After that, we use CNN to gain the weight value of sliding local attention. The Softmax function is finally used to output result. The entire structure is shown below in Figure 1 and the entire process can be described by the following Algorithms 1–3.

In Algorithm 1, we define a function SPLIT_TENSOR, which is used to handle tensor for the Local Attention Structure. In Algorithm 2, we define a function LOCAT_ATTENTION, which is used to output local tensor. In Algorithm 3, we construct the SLAM Framework by the function MAKE_SLAM.

3.3.4. Result Output. The whole process is divided into the training phase and detecting phase. The training phase is mainly used to train the model. In the detection phase, samples are entered into the trained model to produce an output.

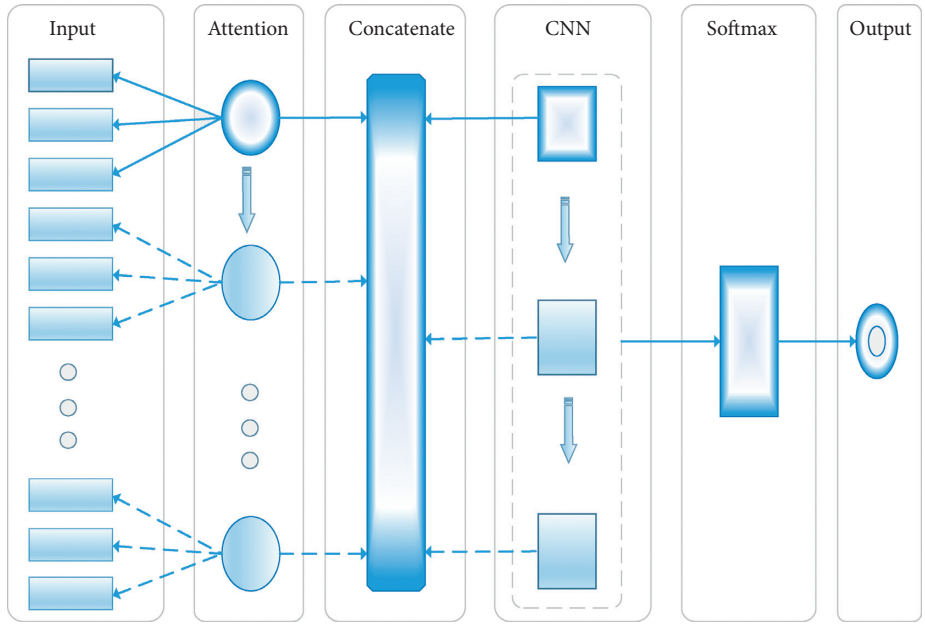


Figure 1: Sliding local attention model.

4. Experiment Result and Evaluation

4.1. Dataset and Environment. In order to make our model more convincing, here we use the public dataset (the data set of Alibaba 3rd Security Algorithm Challenge [31]). The dataset consists of API call sequences which are generated by the windows executable program in the sandbox simulation. It is mainly composed of normal program, infected virus, Trojan Horse program, mining program, DDoS Trojan, and extortion virus. We treat these five malicious types as a same malicious type. Then, the dataset is classified into two categories, that is, normal samples and malicious samples. The sample size of the dataset is shown in Table 3.

As shown in Table 3, the normal data is 110000 and the malware data is 27770. Since the number of normal samples and the number of malicious samples are very different, we adopt a random sampling method to construct the dataset, and a total of 9192 samples are selected.

The runtime environment of the experiment includes Ubuntu 14.06 (64 bit), 16 GB memory, 10G Titank GPU.

4.2. Experiment Result and Analysis. In order to evaluate our model, we choose Accuracy, Precision, Recall, and $F1$-Score as evaluation criteria. Define TP for True Positive, which is the number of samples classified as normal category correctly. Define FN for False Negative, which is the number of samples classified as malicious category correctly. Define TN for True Negative, which is the number of samples classified as malicious category wrongly. Define FP for False Positive, which is the number of samples classified as normal category wrongly. Then, these evaluation criteria could be defined as follows:

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}},\\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}},\\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}},\\
F1 - \text{Score} &= \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}.
\end{aligned} \tag{5}$$

We adopt the 10-fold crossvalidation method to validate our model SLAM and obtain their average value for evaluation.

The confusion matrix for our model SLAM is as shown in Table 4. The accuracy of our model SLAM is shown in Figure 2. As can be seen from Figure 2, the accuracy of our model SLAM is at least 0.9586, the highest is 0.9869, and the average is 0.9723, which achieves a good classification effect.

We select the best 7-fold model to further evaluate its other indicators. The ROC curve for our model SLAM is shown in Figure 3. From Figure 3, we can see that the ROC curve area is about 0.9870.

The classify report for our model SLAM is as shown in Table 5. From Table 5, we can see that the Precision, Recall, and $F1$-score indication are about 0.9869. From the results of these experiments, we can see that our model SLAM achieves a good classification result.

By further analyzing our model, we can know that it can obtain the local information contained in the API execution sequence through the local attention mechanism, which will be beneficial to the classifier. Also, it successfully scans the entire API execution sequence by sliding the window, which obtains a broad view. Meanwhile, the two-dimensional input vector we construct contains both API semantics and structure information, which will greatly enhance the relevance of the input information. In general, effectively extracting data

```
Input: tensor, index, step_size
Output: temp_tensor
  function SPLIT_TENSOR (tensor, index, step_size)
    construct a Lambda expression according to keras
    Initialize temp_tensor
    temp_tensor = cut tensor according to its index from index to index + step_size
    return temp_tensor
end function
```

Algorithm 1: Split tensor vector.

```
Input: query, key, value
Output: local_tensor
    function LOCAL_ATTENTION (query, key, value)
      Initialize local_tensor
      Initialize F function (from Attention mechanism)
      local_tensor = F (q, k) v
      return local_tensor
    end function
```

Algorithm 2: Construct local attention structure.

```
Input: input_vector, step_size
Output: output
    function MAKE_SLAM (input_vector)
      Initialize length
      Initialize CNN function from CNN layer
      Initialize Softmax function from Dense layer
      Initialize concatenate as a middle layer
      length = get the length of input_vector
      for (index = 0, index < length, index += step_size) do
        temp_tensor = SPLIT_TENSOR (input_vector, index, step_size)
        local_tensor = LOCAL_ATTENTION (temp_tensor, temp_tensor, temp_tensor)
        append local_tensor into concatenate
        end for
        tensor = CNN (concatenate)
        output = Softmax (tensor)
        return output
    end function
```

Algorithm 3: Construct SLAM framework.

features and designing a targeted model framework based on data characteristics is the reason why our model SLAM achieves good results.

4.3. Comparison with Other Input. To verify the validity of our 2-dimensional feature extraction method, we compare them with different feature extraction method by our model SLAM. We use the 1-dimensional API index sequence with no structural information as a comparison and use the accuracy rate as an indicator. We still use 10-fold cross-validation and the results are shown in Figure 4.

The comparison results of the average accuracy are shown in Table 6. From Table 6, we can see that the 1-d input accuracy is 0.9484 and the 2-d input accuracy is 0.9723. It can be seen that the 2-dimensional feature extraction method is higher than the 1-dimensional feature extraction

Table 3: The sample size of the dataset.

Normal	Malware
110000	27770

Table 4: The confusion matrix of our model.

	Normal	Malware
Normal	475	6
Malware	6	432

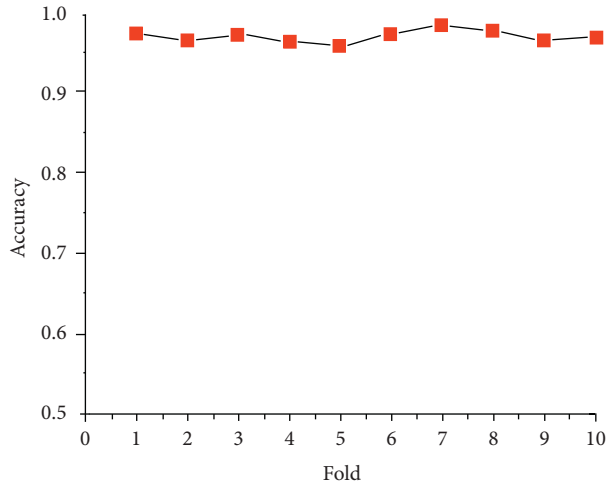


Figure 2: The accuracy of SLAM for 10-fold crossvalidation.

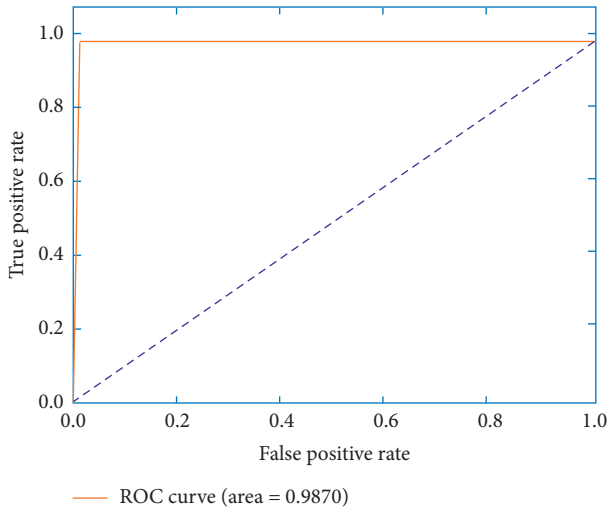


Figure 3: The ROC curve for SLAM.

Table 5: The classify report for our model.

	Precision	Recall	$F1$-score	Support
Normal	0.9875	0.9875	0.9875	481
Malware	0.9863	0.9863	0.9863	438
Macro avg	0.9869	0.9869	0.9869	919

method by an average of nearly 3 percentage points. This proves the effectiveness of our 2-dimensional feature extraction method based on semantics and structure.

4.4. Comparison with Other Models. For comparison, we choose three baseline models.

4.4.1. Baseline Model 1. Random Forest is an emerging, highly flexible machine learning algorithm with broad application prospects, which is often used in many competitions. In the work of [32], they also use random forest as one of models, and the result of the random forest model were the best. Therefore, we choose random forest as our baseline model, and its parameters are set as follows: $n\_estimators = 500$ and $n\_jobs = -1$.

4.4.2. Baseline Model 2. In the work of [33], they use an Attention_CNN_LSTM model to detect malware, which we call it ACLM and treat it as our baseline model.

4.4.3. Baseline Model 3. In the work of [5], they use a two-stream attention mechanism model based on content and

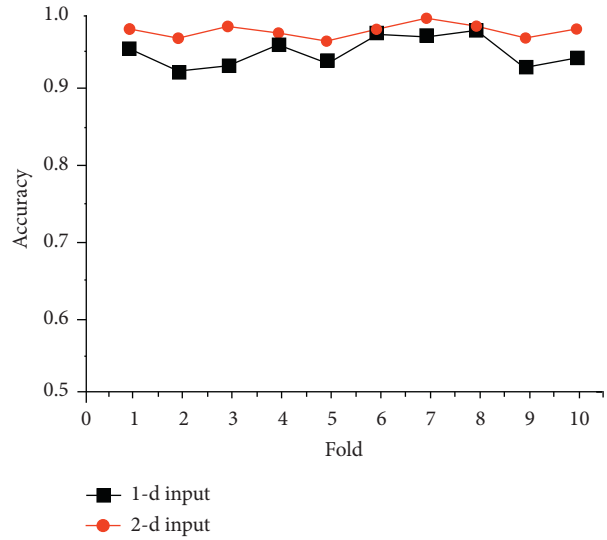


Figure 4: Comparison with different dimensions.

Table 6: Comparison with different inputs.

Input	Accuracy
1-dimensional API	0.9484
2-dimensional API	0.9723

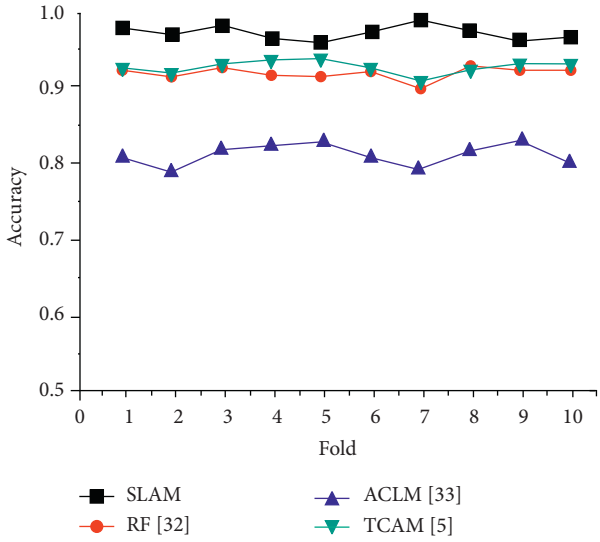


Figure 5: Comparison accuracy with 10-fold crossvalidation.

context information to resolve the NLP problem. By drawing on their ideas, we construct a two-stream CNN-Attention model as a baseline model called TCAM.

4.4.4. Comparison Result. We use 10-fold crossvalidation to verify these models. The results of the comparison are shownin Figure 5.

We count the average accuracy of these models based on 10-fold crossvalidation. The comparison results are shown

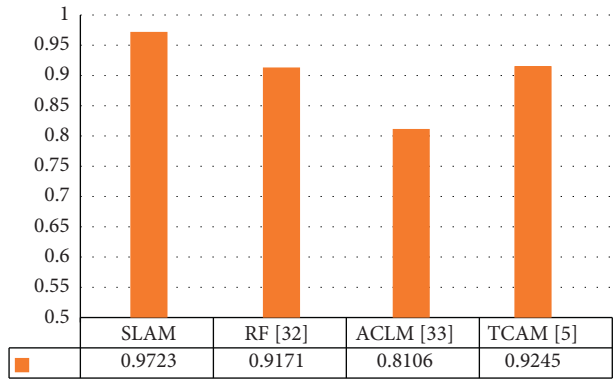


Figure 6: Comparison average accuracy for models.

Table 7: The classify report for our model.

	Accuracy	Precision	Recall	*F*1-score	Support
SLAM	0.99	0.99	0.99	0.99	919
RF [32]	0.9270	0.93	0.93	0.93	919
ACLM [33]	0.81	0.82	0.82	0.82	919
TCAM [5]	0.9325	0.93	0.93	0.93	919

below in Figure 6. In Figure 6, our SLAM model accuracy is 0.9723, the RF model accuracy is 0.9171, the ACLM model accuracy is 0.8106, and the TCAM model accuracy is 0.9245.

Also, we select the best ones from the 10-fold cross-validation of these models and compare them by Accuracy, Precision, Recall, and *F*1-score. The results of the comparison are shown in Table 7.

From these comparison results in Figures 5 and 6 and Table 7, we can see that our model has a better classification effect. The Accuracy, Precision, Recall, and *F*1-score for our model SLAM are all about 0.99. According to these results, we conduct an in-depth analysis.

For the RF [32] model, it is a classic traditional machine learning method, which basically represents the limit of the

traditional machine learning method, but it is difficult to go beyond deep learning.

For the ACLM [33] model, due to the API execution sequence of up to 2000, the extraction based on the attention mechanism will be diluted. Because it is in such a long sequence, it will be difficult to really notice the key parts.

For the two-stream TCAM [5] model migrated according to the content and context idea, some of its ideas are worth learning, but the malware is different from the NLP. Thus, it still needs to be improved according to the target.

Our model SLAM is based on the sliding local attention mechanism, which can well match the data characteristics of the API execution sequence, so that it achieves the best classification effect.

5. Conclusion

We analyze the characteristics of the API execution sequence and present a 2-dimensional extraction method based on semantics and structure information. Furthermore, according to the API data characteristics and attention mechanism, we design and implement a sliding local attention detection framework. The experimental results show that our feature extraction method and detection framework have good classification results and high accuracy. In the future work, we will further explore the application of attention mechanisms in the malware detection area.

Data Availability

The dataset of Alibaba 3rd Security Algorithm Challenge can be obtained from https://tianchi.aliyun.com/competition/entrance/231668/information.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the grants from the National Key Research and Development Program of China (Project no. 2018YFB0805000).

References

[1] 360 Security Report, 2019, http://zt.360.cn/1101061855.php?dtid=1101062370did=610142397.

[2] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security—VizSec'11*, Pittsburgh, PA, USA, July 2011.

[3] L. Nataraj, "A signal processing approach to malware analysis," Dissertations & theses—gradworks, University of California, Santa Barbara, CA, USA, 2015.

[4] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 9, pp. 1336–1347, 2017.

[5] Z. Yang, Z. Dai, Y. Yang et al., "XLNet: generalized autoregressive pretraining for language understanding," 2019, https://arxiv.org/abs/1906.08237.

[6] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Proceedings of the 2005 in 2005 IEEE Symposium on Security and Privacy (S&P'05)*, IEEE, Oakland, CA, USA, May 2005.

[7] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *Proceedings of the USENIX Security Symposium*, vol. 13, San Diego, CA, USA, August 2004.

[8] F. Cohen, "Computer viruses," *Computers & Security*, vol. 6, no. 1, pp. 22–35, 1987.

[9] D. M. Chess and S. R. White, "An undetectable computer virus," in *Proceedings of the Virus Bulletin Conference*, vol. 5, Orlando, FL, USA, September 2000.

[10] S. Venkatraman and M. Alazab, "Use of data visualisation for zero-day malware detection," *Security and Communication Networks*, vol. 2018, Article ID 1728303, 13 pages, 2018.

[11] D. G. Llaurad, *Convolutional Neural Networks for Malware Classification*, Rovira i Virgili University, Tarragona, Spain, 2016.

[12] L. D. Vu Duc, "Deepmal: deep convolutional and recurrent neural networks for malware classification," 2018, https://arxiv.org/pdf/2003.04079.

[13] K. Zhang, C. Li, Y. Wang, X. Zhu, and H. Wang, "Collaborative support vector machine for malware detection," *Procedia Computer Science*, vol. 108, pp. 1682–1691, 2017.

[14] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures,"vol. 121, pp. 171–182, in *Proceedings of the Ninth Australasian Data Mining Conference*, vol. 121, Australian Computer Society, Inc., Ballarat, Australia, January 2011.

[15] F. Xiao, Z. Lin, Yi Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Mathematical Problems in Engineering*, vol. 2019, Article ID 8195395, 10 pages, 2019.

[16] Q. Qian and M. Tang, "Dynamic API Call sequence visualization for malware classification," *IET Information Security*, vol. 13, no. 4, pp. 367–377, 2018.

[17] L. Xiaofeng, Z. Xiao, J. Fangshuo, Y. Shengwei, and S. Jing, "ASSCA: API based sequence and statistics features combined malware detection architecture," *Procedia Computer Science*, vol. 129, pp. 248–256, 2018.

[18] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware detection and classification based on extraction of API sequences," in *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2337–2342, IEEE, New Delhi, India, September 2014.

[19] X. Ma, S. Guo, W. Bai, J. Chen, S. Xia, and Z. Pan, "An API semantics-aware malware detection method based on deep learning," *Security and Communication Networks*, vol. 2019, Article ID 1315047, 9 pages, 2019.

[20] H. S. Anderson and P. Roth, "Ember: an open dataset for training static PE malware machine learning models," 2018, https://arxiv.org/pdf/1804.04637.

[21] M. Alazab and S. Venkatraman, "Detecting malicious behaviour using supervised learning algorithms of the function calls," *International Journal of Electronic Security and Digital Forensics*, vol. 5, no. 2, pp. 90–109, 2013.

[22] M. Ficco, "Comparing API call sequence algorithms for malware detection," in *Advances in Intelligent Systems and Computing*, Springer, Berlin, Germany, 2020.

[23] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on autoencoders and API-images," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 26–33, 2020.

[24] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, ACM, Chicago, IL, USA, pp. 21–30, October 2011.

[25] C. Liangboonprakong and O. Sornil, "Classification of malware families based on *n*-grams sequential pattern features," in *Proceedings of the 2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 777–782, IEEE, Melbourne, Australia, June 2013.

[26] G. Bala Krishna, V. Radha, and K. V.G. Rao, "ELC-PPW: ensemble learning and classification (LC) by positional patterns weights (PPW) of API calls as dynamic *n*-grams for malware perception," *International Journal of Simulation–Systems, Science & Technology*, vol. 18, no. 1, 2017.

[27] J. Devlin, M. W. Chang, K. Lee et al., "Bert: pre-training of deep bidirectional transformers for language understanding," 2018, https://arxiv.org/abs/1810.04805.

[28] Cuckoo Sandbox, 2019, https://cuckoosandbox.org/.

[29] Word2vec, 2019, https://code.google.com/p/word2vec/.

[30] Virusshare Website, 2019, https://virusshare.com/.

[31] Alitianchicontest, https://tianchi.aliyun.com/competition/introduction.htm?spm=5176.11409106.5678.1.4354684cI0fYC1?raceId=231668s.

[32] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "MalInsight: a systematic profiling based malware detection framework," *Journal of Network and Computer Applications*, vol. 125, pp. 236–250, 2019.

[33] S. Luo, Z. Liu, B. Ni et al., "Android malware analysis and detection based on attention-CNN-LSTM," *Journal of Computers*, vol. 14, no. 1, pp. 31–44, 2019.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 8810817, 12 pages
https://doi.org/10.1155/2020/8810817

Research Article

GFD: A Weighted Heterogeneous Graph Embedding Based Approach for Fraud Detection in Mobile Advertising

Jinlong Hu,[1,2] **Tenghui Li**,[1,2] **Yi Zhuang**,[1,2] **Song Huang**,[1,2] **and Shoubin Dong**[1,2]

[1]*School of Computer Science and Engineering, South China University of Technology, Guangzhou, China*
[2]*Communication and Computer Network Laboratory of Guangdong, South China University of Technology, Guangzhou, China*

Correspondence should be addressed to Jinlong Hu; jlhu@scut.edu.cn

Received 9 April 2020; Revised 28 July 2020; Accepted 25 August 2020; Published 4 September 2020

Academic Editor: Hammad Afzal

Online mobile advertising plays a vital role in the mobile app ecosystem. The mobile advertising frauds caused by fraudulent clicks or other actions on advertisements are considered one of the most critical issues in mobile advertising systems. To combat the evolving mobile advertising frauds, machine learning methods have been successfully applied to identify advertising frauds in tabular data, distinguishing suspicious advertising fraud operation from normal one. However, such approaches may suffer from labor-intensive feature engineering and robustness of the detection algorithms, since the online advertising big data and complex fraudulent advertising actions generated by malicious codes, botnets, and click-firms are constantly changing. In this paper, we propose a novel weighted heterogeneous graph embedding and deep learning-based fraud detection approach, namely, GFD, to identify fraudulent apps for mobile advertising. In the proposed GFD approach, (i) we construct a weighted heterogeneous graph to represent behavior patterns between users, mobile apps, and mobile ads and design a weighted metapath to vector algorithm to learn node representations (graph-based features) from the graph; (ii) we use a time window based statistical analysis method to extract intrinsic features (attribute-based features) from the tabular sample data; (iii) we propose a hybrid neural network to fuse graph-based features and attribute-based features for classifying the fraudulent apps from normal apps. The GFD approach was applied on a large real-world mobile advertising dataset, and experiment results demonstrate that the approach significantly outperforms well-known learning methods.

1. Introduction

Online mobile advertising plays a vital role in the mobile app ecosystem. One of the popular models in mobile app advertising is known as cost per action (CAP), where payment is based on user action, such as downloading and installing an app on the user's mobile device. This CAP model may incentivize malicious mobile content publishers (typically app owners) to generate fraudulent actions on advertisements to get more financial returns [1–3]. Some traditional methods and techniques have been used for detecting and stopping click fraud, such as threshold-based method [4], CAPTCHA [5], splay tree [6], TrustZone [7], power spectral density analysis [8], and social network analysis [9].

To automatically detect mobile advertising fraud behaviors, machine learning methods have been successfully applied to find fraud patterns in data, distinguishing suspicious advertising fraud operation from normal one [10–14]. As for learning model with attribute features, researchers usually use several attributes from each sample to train a learning model to identify the fraud behaviors. Unfortunately, such approaches may suffer from labor-intensive feature engineering and robustness of the detection algorithms, since the online advertising big data and complex fraudulent advertising actions generated by malicious codes, botnets, and click-firms are constantly changing. What is more, fraudsters could easily adjust their fraud patterns based on existing fraud detection attributes and rules to avoid being detected. Recently, some researchers try to use the relationship between information entities to construct a graph model and then use the graph mining or learning methods to identify the changing fraud behaviors

[15–17]. All these methods obtain useful insights into the learning mechanism to classify fraud behaviors from normal activities. Intuitively, if we could combine the complementary information from attributes of sample data and relationship between entities (e.g., users, apps, and ads), we will be able to improve the accuracy and robustness of fraud detection.

However, to unleash the power of attribute-based information and graph-based information, we have to address a series of challenges. First, to take advantage of the characteristic of graph, we should construct a suitable graph, which could potentially represent the interaction behaviors between information entities such as users, apps, and ads. Second, an efficient graph learning method should be developed to learn the useful structural and semantic representation information from constructed graph [18, 19], particularly learning from heterogeneous graph [20]. Third, fusing different kinds of information from sample attributes and node representation is difficult for their inherent heterogeneity and high-order characteristics.

To address the above challenges, in this paper, we propose a weighted heterogeneous graph embedding and deep learning-based fraud detection approach, namely, GFD, to identify fraudulent apps for mobile advertising. In the proposed GFD approach, (i) considering behavior patterns between users, mobile apps, and mobile ads, we construct a weighted heterogeneous graph to represent mobile app advertising behavior and propose a new weighted metapath to vector algorithm, namely, WMP2vec, to learn low-dimensional latent representation (graph-based features) for apps' nodes in the weighted heterogeneous graph; (ii) we use a time window based statistical analysis method to extract intrinsic features (attribute-based features) from the tabular sample data; (iii) we present a hybrid convolutional neural network model to fuse graph-based features and attribute-based features for classifying the fraudulent apps from normal apps.

We evaluate GFD approach and WMP2vec algorithm on a real-world dataset from one of the mobile advertising platforms in China. Results show that WMP2vec reaches higher performance than three well-known graph embedding algorithms in the constructed weighted heterogeneous graph, and GFD approach achieves highest classification performance compared with Support Vector Machine (SVM), Random Forest (RF), and Fully Connected Neural Networks (FCNN).

The rest of the paper is organized as follows. We introduce GFD approach to detect fraudulent apps with deep neural networks and heterogeneous graph embedding algorithm WMP2vec in Section 2. We present the experimental results and discussion in Section 3. In Section 4, we introduce the related work. We conclude this paper in Section 5.

2. Proposed Approach

The flow chart of the proposed GFD approach is shown in Figure 1. First, we propose a weighted heterogeneous graph embedding method to learn the node representation, including constructing the weighted heterogeneous graph and the WMP2vec algorithm. Second, we use statistical analysis method to extract attribute-based features from the tabular sample data. Third, we introduce the deep neural networks to fuse the attribute-based features and graph-based features for identifying fraudulent apps from normal ones.

2.1. Data Description. We collect advertising log data of mobile apps from a mobile advertising platform. Our mobile advertisement dataset contains the following attributes: user ID, a code to identify a unique mobile user; app ID, a code to identify a unique mobile app; ad ID, a code to identify a unique mobile advertisement; geographical attributes, a series of user geographical attributes used to detect anomalies, including encrypted IP and city; action type, user behavior related to the ads, such as viewing, clicking, app downloading start, app downloading completion, and app installation completion; action time, the time-stamp when the action happened; and device attribute, user device related attributes, such as device ID, device system models, and screen size.

A seven-day mobile advertising log dataset in June 2015 was studied in this paper, and some examples of our raw data are shown in Table 1.

2.2. Weighted Heterogeneous Graph Embedding. In this section, we firstly propose the problem definition and construct the weighted heterogeneous graph, and then we present WMP2vec algorithm to learn latent representation of nodes in weighted heterogeneous graph.

2.2.1. Problem Definition

(1) Given. An undirected weight heterogeneous graph $G = \langle V, E, W \rangle$ is given, where V is a set of app nodes, ad nodes, and user nodes; E is a set of undirected weight edges between any two types of nodes: app nodes and user nodes, user nodes and ad nodes, and ad nodes and app nodes; W is the set of weight of edges.

(2) Task. The task is to learn the d-dimensional latent representations $X_e \in \Re^{|V| \times d}$ (where $d \ll |V|$) for nodes, which could capture the structural and semantic relations among nodes in the graph G, and the representations could be used for classifying fraudulent apps.

2.2.2. Weighted Heterogeneous Graph Construction. Let U be the set of user nodes, let A be the set of app nodes, and let P be the set of advertisement nodes. If there exists an action from user $u \in U$ to advertisement $a \in A$ through app $p \in P$, we form edges from u to p, from u to a, and from p to a, respectively, such that $E_1 = U \times P$, $E_2 = U \times A$, and $E_3 = P \times A$ are the edges set of heterogeneous graph G. The set of weight is $W = \{w_{up}, w_{ua}, w_{pa}\}$, where the weights w_{up}, w_{ua}, and w_{pa} are defined proportional to the behavioral centrality of u to p, u to a, and p to a, respectively. The calculation

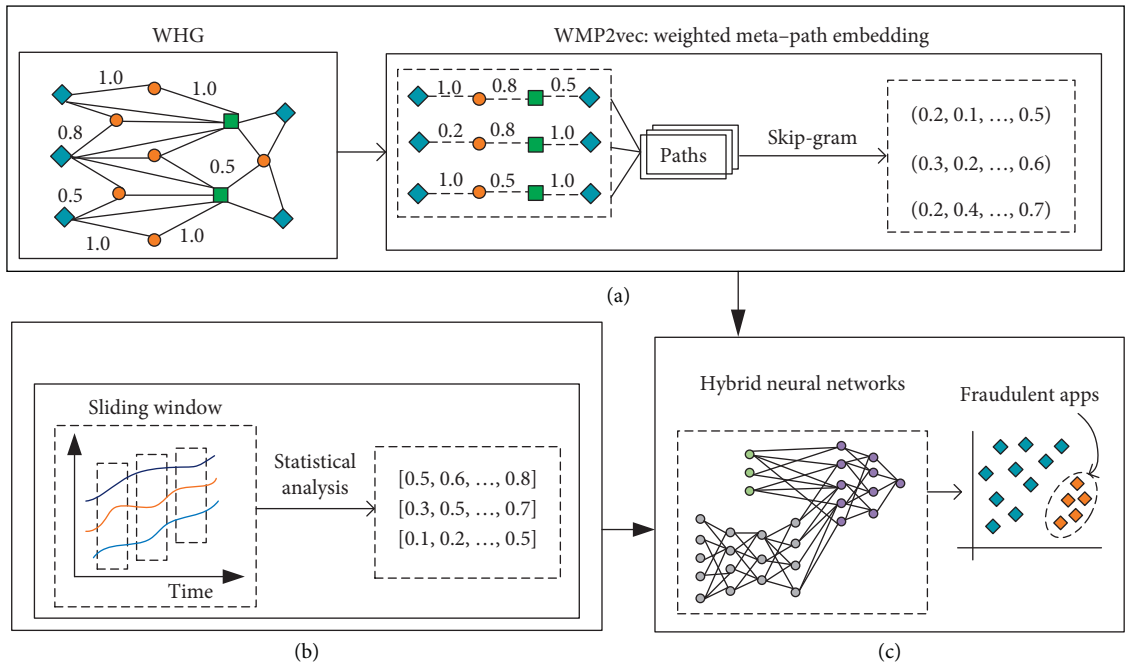


Figure 1: The flow chart of GFD approach. (a) Graph-based feature extraction. (b) Attribute-based feature extraction. (c) Deep Fusion.

Table 1: Example of the mobile advertising log data.

User ID	App ID	Ad ID	Action
B360**	*369	*103	Viewing
B360**	*369	*103	Clicking
Xjnh**	*370	*125	Viewing
Xjnh**	*370	*125	Downloading
Lmsv**	*412	*130	Downloaded
Lmsv**	*412	*130	Installing
Lmsv**	*412	*133	Installed

formula of w_{up} is shown in equation (1) and so on for w_{ua} and w_{pa}.

$$w_{\text{up}} = \frac{C_{\text{up}}}{\sum_{j \in \Omega(u)} C_{uj}}, \tag{1}$$

where C_{up} is the times of user u operating on advertisement p and $\Omega(u)$ is the set of operations of user u on all the advertisements.

2.2.3. Graph Embedding Algorithm. In this section, based on the sequence generation method from metapath based random walk in heterogeneous graph [20], we propose WMP2vec algorithm to generate random walk sequence in weighted heterogeneous graph and embed sequence to representation vector with Skip-Gram [21] for nodes.

(1) Weighted Metapath Based Random Walk. We predefined number of walks per node n, the number of walk sequences l, and a metapath M. The metapath is defined as a path in the heterogeneous graph G with its metatemplate $T_G = (Z, R)$, where $Z = \{Z_u, Z_p, Z_a\}$ and $R = \{R_u, R_p, R_a\}$. Each node v and each edge e are associated with mapping functions $\varphi(v)$: $V \longrightarrow Z$ and $\phi(e)$: $E \longrightarrow R$, respectively.

Supposing that current node is v^i, the relationship between v^i and next node v^{i+1} is R_i; that is, $\phi(v^i, v^{i+1}) = R_i$.

For walk sequences generation, we go through the metapath scheme l times, and each time generates one corresponding walk sequence. In the first time, we use two different selecting methods (first phase and second phase), because there are no limits to edge weight in the beginning. After first time, we use the method in the second phase to select next node.

For the first phase, when the length of walk sequence is less than 2, the next node in the sequence is randomly selected from the neighbors set $N_{t+1}(v^i)$ of current nodes, which meet the requirements of metapath M [20]. The transition probability from v^i to v^{i+1} is defined as follows:

$$p_1\left(v^{i+1} \mid v^i, M\right) = \begin{cases} 0, & \phi\left(v^i, v^{i+1}\right) \neq R_i, \\ \dfrac{1}{\left|N_{t+1}\left(v^i\right)\right|}, & \phi\left(v^i, v^{i+1}\right) = R_i. \end{cases} \tag{2}$$

For the second phase, when the length of walk sequence is between 2 and $l^*|R|$, the transition probability is restricted by a weight bias β. Supposing that the latest weight of edge of relationship R_i is w_i, the weight should be in the range of $[w_i - \beta, w_i + \beta]$. The transition probability from v^i to v^{i+1} is defined as follows:

$$p_2\left(v^{i+1} \mid v^i, M_w\right) = \begin{cases} 0, & \phi\left(v^i, v^{i+1}\right) \neq R_i, \\ 0, & \phi\left(v^i, v^{i+1}\right) = R_i; w_{v^{i+1},v^i} \notin [w_i - \beta, w_i + \beta], \\ \dfrac{1}{\left|C\left(v^i\right)\right|}, & \phi\left(v^i, v^{i+1}\right) = R_i; w_{v^{i+1},v^i} \in [w_i - \beta, w_i + \beta], \end{cases} \tag{3}$$

where $C(v^i)$ is the set of neighbors meeting the requirement.

(2) Embedding Sequence to Vector with Skip-Gram. Based on the weighted metapath random walk sequences, we use Skip-Gram model [21] and negative sampling [22] to learn low-dimensional representation of nodes.

A description of our proposed WMP2vec algorithm method is shown in Algorithm 1.

2.3. Attribute-Based Feature Extracting. From the raw log data (tabular data) of mobile advertising, we defined a time window (t hours) and divide original data into $24/t$ data block for one day (24 hours). Then, a plain statistical analysis is performed on each field in each data block. The ratio of the unique value of the field to the total number of records in the specified time window is computed. The attribute-based feature corresponding to one mobile app could be represented as a feature matrix with $24/t$ rows.

2.4. Hybrid Neural Network for Classification. To take advantage of the graph-based features and attribute-based features, we propose a hybrid convolutional neural networks (HNN) model to fuse and learn both information in GFD approach. The overview of the hybrid neural networks is shown in Figure 2.

In HNN model, the first layer (input layer) contains attribute-based feature matrix $X_s \in \mathfrak{R}^{N\times t\times m}$ and graph-based feature$X_e \in \mathbb{R}^{N\times d}$, where N is the number of samples, t is the number of time windows by one day (24 hours), m is the dimension of attribute-based feature in a time window, and d is the dimension of node embedding.

A convolutional part includes two convolutional layers, and the output of the first convolutional layer is

$$C_1 = \text{active}(z) = \text{active}\left(X_s{}^* W_{C_1} + b_{c_1}\right), \tag{4}$$

where $W_{C_1} \in \mathbb{R}^{w_1\times w_1}$ and $b_{c_1} \in \mathbb{R}$ are the convolution kernel and bias, respectively, w_1 is the size of the kernel, * indicates the convolution operation, and the active function is $\text{relu}(x) = \max(0, x)$.

The second convolutional layer is constructed as follows:

$$C_2 = \text{acitve}\left(C_1 * W_{C_2} + b_{c_2}\right), \tag{5}$$

where $W_{C_2} \in \mathbb{R}^{w_2\times w_2}$ and $b_{c_2} \in \mathbb{R}$ are the convolution kernel and bias, respectively. w_2 is the size of the kernel.

C_2 is flattened to $X_c \in \mathbb{R}^{N\times d_0}$, where d_0 is the number of elements in C_2.

We concatenate X_c and X_r into a single metric $X \in \mathbb{R}^{N\times(d_0+d)}$ to be the input of the first fully connected layer l_1. l_1 is constructed as follows:

$$l_1 = \text{active}\left(XW_1 + b_1\right), \tag{6}$$

where $W_1 \in \mathbb{R}^{(d_0+d)\times d_1}$ and $b_1 \in \mathbb{R}^{d_1}$ are weight and bias, respectively, and d_1 is the number of neurons in the first fully connected layer.

The second fully connected layer l_2 is constructed as follows:

$$l_2 = \text{active}\left(l_1W_2 + b_2\right), \tag{7}$$

where $W_2 \in \mathbb{R}^{d_1\times d_2}$ and $b_2 \in \mathbb{R}^{d_2}$ are weight and bias, respectively, and d_2 is the number of neurons in the second fully connected layer.

In the output of HNN, $\widehat{y} \in (0, 1)$ is the probability of an application to be a fraudulent application.

$$\widehat{y} = \sigma\left(l_2W_o + b_o\right), \tag{8}$$

where $W_o \in \mathbb{R}^{d_2\times 1}$ and $b_o \in \mathbb{R}^{d_2}$ are weight and bias, respectively, and $\sigma(\cdot) = \text{sigmoid}(\cdot)$ is the sigmoid function.

The cross-entropy function with l2-regularization is used to calculate the loss of the hybrid convolutional neural network model.

$$L(y, \widehat{y}) = -y\log(\widehat{y}) - (1 - y)\log(1 - \widehat{y}) + \frac{\lambda}{2}||\theta||^2. \tag{9}$$

3. Experiments

3.1. Data Description and Preprocessing. A real-world dataset was collected from a mobile advertising platform in China. The dataset consists of seven days with around 2 M users, 3.5 K apps, and 1 K advertisements per day. We partition our log data into seven subsets with one-day period and conduct experiments on each subset to evaluate our model. The proportion of fraudulent apps is about 2–4 percent in the total 3,500 apps each day. More details of the dataset are described in Section 2.1.

3.2. Evaluation Metric. In this paper, we define the fraudulent apps by positive samples and the other apps by negative samples. The Average Precision (AP) and the Area Under ROC Curve (AUC) are used to evaluate proposed algorithm and approach.

The AP criterion summarizes the Precision-Recall performances at different threshold levels and corresponds to area under the Precision-Recall curve. The ROC curve is

Input: The weighted heterogeneous information graph $G = <V, E, W>$, a meta-path scheme M, walks per node n, longest walk length per walk l, embedding dimension d, neighborhood size k
Output: The latent node embedding $X \in R^{|V| \times d}$
Initialize X, random walk sequence $S = \varnothing$
for $v \in V$ **do**
for $i = 1 \longrightarrow n$ **do**
S_t = WeightedMetaPathRandomWalk (G, M, v, l)
$S = S + S_t$
end
end
X = HeterogeneousSkipGram (X, k, S)
return X
WeightedMetaPathRandomWalk (G, M, v, l)
initialize random walk array $S = [v]$, weight array $w_i = [1.0]$, $i = 1 \ldots |M|/2$
relationship array $R = [R_1, \ldots, R_{|M|/2}, R_{|M|/2}, \ldots, R_1]$
for $j = 1 \longrightarrow l$ **do**
for $k = 1 \longrightarrow |M|/2$ **do**
if $j = 1$ **then**
    draw u and w according to equation (2) with relationship $R[k-1]$
    $S[j+1] = u, W_k[j+1] = w$
else
    draw u and w according to equation (3) with relationship $R[k-1]$
    if u does not exist **then** return S
else $S[j+1] = u, W_k[j+1] = w$
end
    for $k = |M|/2 + 1 \longrightarrow |M| - 1$**do**
        draw u and w according to equation (3) with relationship $R[k-1]$
    if u does not exist **then** return S
else $S[j+1] = u$, $W_{|M|-k}[j+1] = w$
end
end
return S

Algorithm 1: The WMP2vec algorithm.

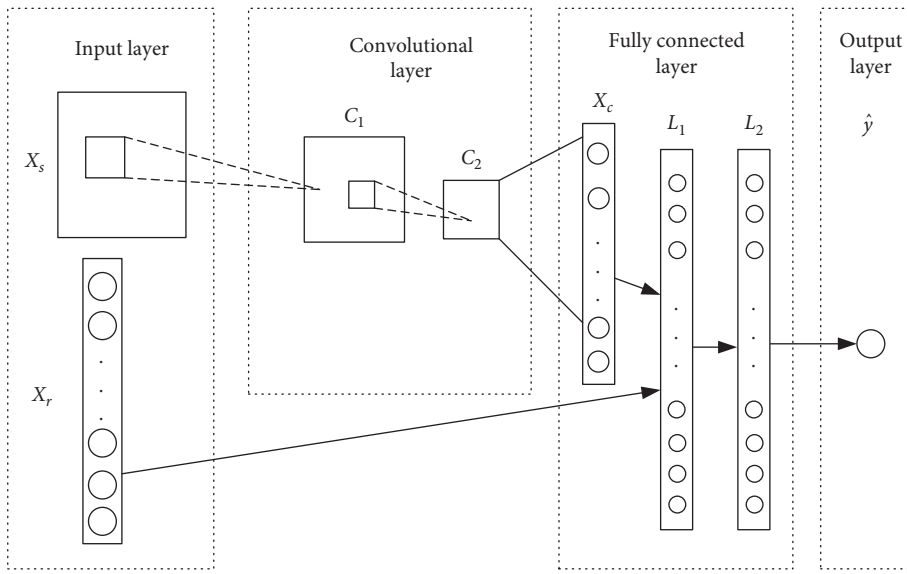


Figure 2: Hybrid convolutional neural networks for fraud detection.

created by plotting the true positive rate against the false positive rate at various threshold settings. The AUC is the total area under the ROC curve.

3.3. Evaluation of WMP2vec Algorithm. In this section, we use WMP2vec algorithm to learn the embedding vector of the nodes (apps) from the constructed weighted heterogeneous graph and then take their embedding vectors as the input of Random Forest (RF) model to classify fraudulent apps.

Based on Section 2.2.2, we construct a weighted heterogeneous graph and define a metapath: app-user-ad-user-app (PUAUP); that is, $M = Z_2 \longrightarrow^{R_1} Z_1 \longrightarrow^{R_3} Z_3 \longrightarrow^{R_3} Z_1 \longrightarrow^{R_1} Z_2$, which represents the heterogeneous semantic of fraud publishers (apps) that mimic legitimate users to act on the ads from the apps.

3.3.1. Comparison Models and Parameters. We compare the AP and AUC of the WMP2vec model with three well-known graph embedding models: DeepWalk [23], Node2vec [24], and Metapath2vec [20]. The compared algorithms and their parameters are as follows:

(1) DeepWalk: DeepWalk [23] is the first graph embedding model based on Word2vec. We use Skip-Gram model [21] and hierarchical softmax [25] with gradient descent to learn the node representation. Negative sampling technique [22] is used to accelerate the Skip-Gram model. The count of random walk is 30, and the walk length is 40.

(2) Node2vec: Node2vec [24] extends DeepWalk algorithm through introducing backward probability p and forward probability q. The same random walk parameters (count = 30 and length = 40) are used with DeepWalk, and the negative sampling technique is also used. In addition, we use $p = 0.5$ and $q = 0.2$ for backward probability and forward probability, respectively.

(3) Metapath2vec: Metapath2vec [20] uses the metapath based random walk to construct node sequences and then leverages Skip-Gram to perform node embedding. The metapath in this study is PUAUP. The count of random walk is 30, and the walk length is 10.

(4) WMP2vec: We use the same parameters (count = 30, length = 10, and metapath = PUAUP) with Metapath2vec, and the weighted bias β is 0.1 additionally.

In all the compared models, we train Skip-Gram model with window size of 5, and the negative samples is 5 in negative-sampling. The graph-based feature of each node is a 32-dimensional vector. The parameters of the RF model are as follows: the number of weak learners is 150, max. deep is 5, and min. sample leaf is 5.

3.3.2. Experimental Results. Tables 2 and 3 show the experimental results by comparing the AP and AUC over 10-fold cross-validation for seven days. The WMP2vec model reached highest AP value in six days and highest AUC value in three days over all seven days. The Metapath2vec model reached highest AP value in one day and highest AUC value in two days over seven days. Thus, WMP2vec outperforms all other models, such that WMP2vec > Metapath 2vec > Node2vec > DeepWalk.

Table 2: Classification results for all embedding models in AP (mean ± std).

Date	DeepWalk	Node2vec	Metapath2vec	WMP2vec
1st June	0.168 ± 0.082	0.343 ± 0.057	0.344 ± 0.065	0.384 ± 0.055
2nd June	0.318 ± 0.137	0.384 ± 0.079	0.342 ± 0.067	0.421 ± 0.084
3rd June	0.281 ± 0.075	0.439 ± 0.122	0.401 ± 0.110	0.459 ± 0.114
4th June	0.313 ± 0.073	0.335 ± 0.085	0.360 ± 0.096	0.409 ± 0.099
5th June	0.308 ± 0.121	0.379 ± 0.102	0.387 ± 0.074	0.411 ± 0.091
6th June	0.199 ± 0.113	0.230 ± 0.089	0.369 ± 0.097	0.404 ± 0.102
7th June	0.244 ± 0.054	0.297 ± 0.075	0.371 ± 0.094	0.353 ± 0.075

Table 3: Classification results for all embedding models in AUC (mean ± std).

Date	DeepWalk	Node2vec	Metapath2vec	WMP2vec
1st June	0.788 ± 0.027	0.801 ± 0.038	0.837 ± 0.038	0.877 ± 0.030
2nd June	0.828 ± 0.032	0.848 ± 0.025	0.864 ± 0.028	0.893 ± 0.027
3rd June	0.935 ± 0.023	0.950 ± 0.024	0.912 ± 0.028	0.921 ± 0.024
4th June	0.824 ± 0.032	0.824 ± 0.040	0.849 ± 0.045	0.879 ± 0.045
5th June	0.799 ± 0.060	0.835 ± 0.055	0.853 ± 0.052	0.849 ± 0.038
6th June	0.757 ± 0.076	0.891 ± 0.042	0.844 ± 0.046	0.856 ± 0.041
7th June	0.804 ± 0.031	0.820 ± 0.027	0.844 ± 0.038	0.825 ± 0.049

3.3.3. Impacts of Parameters. In this subsection, we evaluate the impacts of parameters over the classification task: (i) count of random walk, walk length, and window size of Skip-Gram in WMP2vec and Metapath2vec model; (ii) weighted bias β of WMP2vec. We compare the AP and AUC values in the dataset from one day.

(1) Count of Random Walk. Figure 3 shows the experimental results by comparing the AP and AUC with different count of random walk, with fixed walk length of 5. When the count of random walk is larger than 30, WMP2vec and Metapath2vec models have better performance than count = 10, respectively. In addition, the values of AP and AUC have

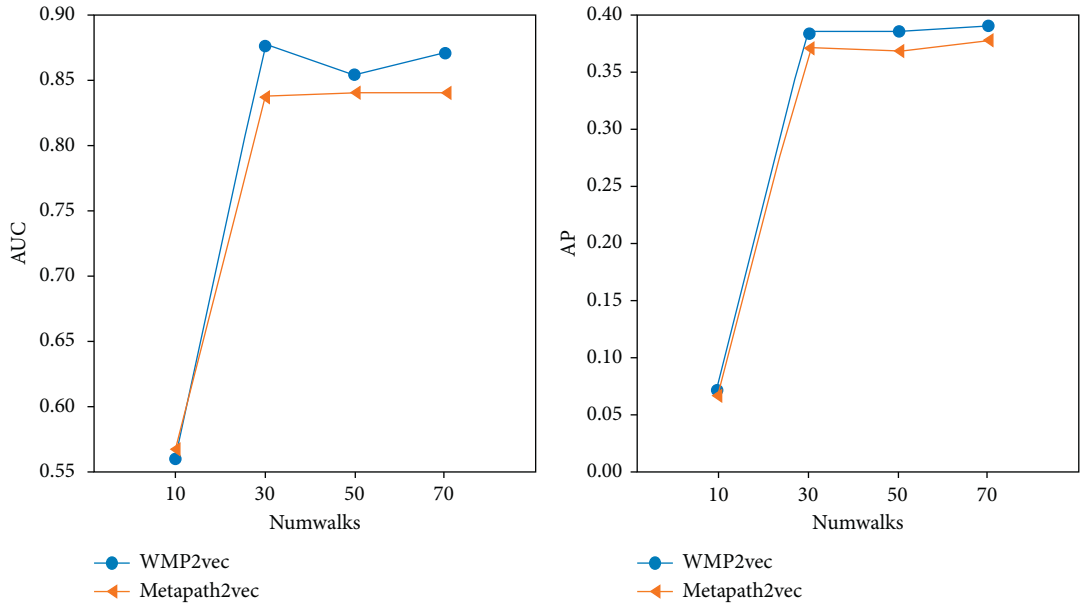


Figure 3: Comparing impacts of the different count of random walk with AUC and AP.

slight changes when the count of random walk is 30, 50, or 70.

(2) Walk Length. Figure 4 shows the experimental results by comparing the AP and AUC with different walk length (length = 5, 10, 20, 50, and 80), with fixed count of random walk of 10. WMP2vec and Metapath2vec models reach better performance when the walk length ≥10. In addition, when the length changes from 10, 20, and 50 to 80, the AP values change very little and the AUC values have some fluctuations.

(3) Window Size of Skip-Gram. Figure 5 shows the experimental results by comparing the AP and AUC with different window size (size = 3, 4, 5, 6, and 7) over the classification task. The best performance of models is reached when the window size is 5.

(4) Weighted Bias of WMP2vec. Figure 6 shows the experimental results by comparing the AP and AUC with different weighted bias β of WMP2vec ($\beta = 0.1$, 0.3, 0.5, 0.7, and 1.0) over the classification task. As the weighted bias β increases, the performance of WMP2vec gets closer to the performance when $\beta = 1.0$. The values of AP and AUC change very little when $\beta \geq 0.5$.

3.4. Evaluation of Hybrid Neural Network. In this section, we evaluate the classification performance of HNN model for fusing graph-based features and attribute-based features in GFD approach. As the flow of GFD approach in Figure 1, we extract the attribute-based features and the graph-based features and then use HNN model to fuse two kinds of features to identify fraudulent apps.

3.4.1. Features Extraction. Based on Section 2.3, we divide the log data for each app into 24 parts per day; that is, the time window is one hour. We calculate the ratio of records whose attributes take a certain value to all records in each time window, and we calculate them for each of 22 attributes in total, such as anonymized user id, advertisement id, country id, and device operating system. In addition, we calculate the ratio for browsing behavior and other actions on ads of users, respectively. Finally, we get 24 features for a time window (one hour), and the dimension of attribute-based features of each app is 24×24 for one day.

Based on Section 2.2.2 and Section 3.3, for the graph-based feature extraction, we construct the weighted heterogeneous graph of user-app-ad and then extract the graph-based feature through training by using WMP2vec. The dimension of graph-based features for each app is 32.

3.4.2. Comparison Models and Experiment Setup. We compare the proposed HNN with Support Vector Machine (SVM), Random Forests (RF), and Fully Connected Neural Networks (FCNN).

(i) SVM : SVM is an effective widely used two-class classification model. The RBF kernel is used and penalty parameter C is 0.9.

RF : RF is a well-known ensemble learning method that operates by constructing a multitude of decision trees at training time. The number of decision trees is 200 with depth of 5. Minimum samples split and minimum samples leaf are set to 5, respectively.

FCNN : FCNN is a fully connected neural network. The number of hidden layers is 4, with 100 neurons in each layer. The learning rate is 0.001 and the keep probability of dropout is 0.9.

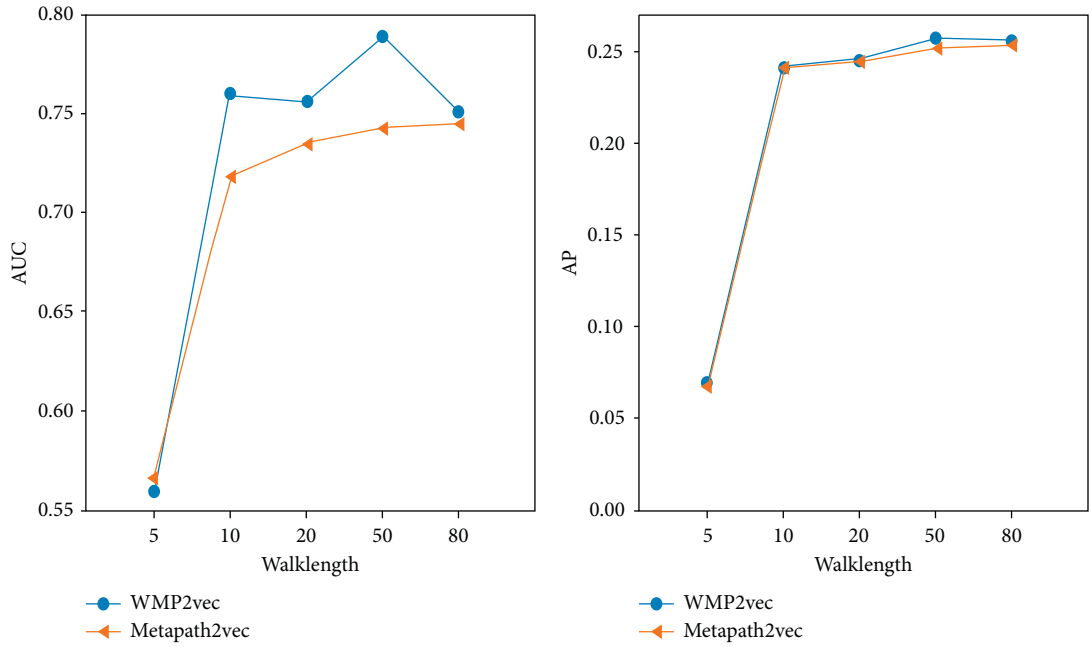


Figure 4: Comparing impacts of the different walk length with AUC and AP.

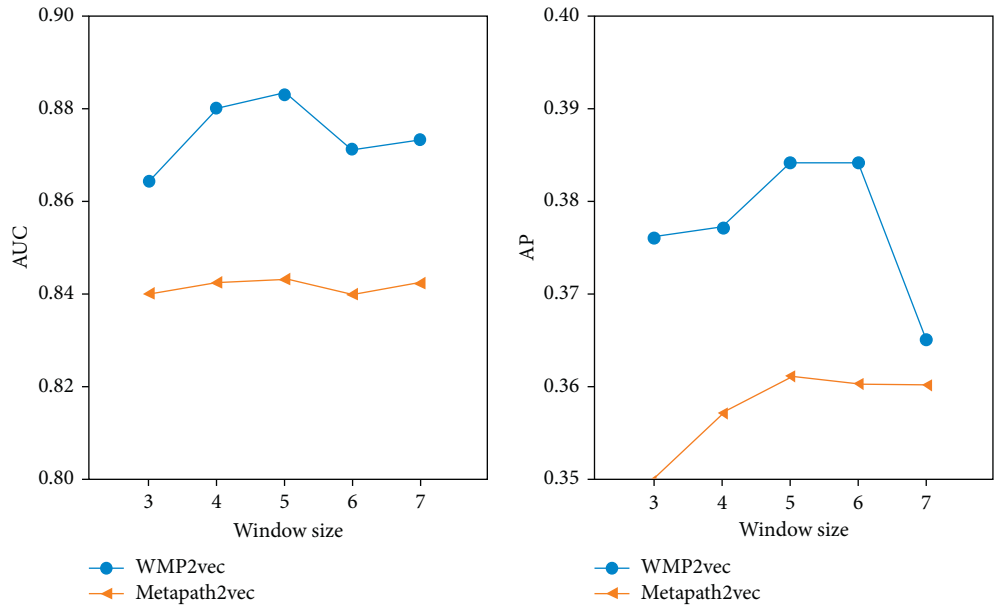


Figure 5: Comparing impacts of the different window size with AUC and AP.

HNN : HNN is the fusing model proposed in this study. The number of convolutional layers is 2, and the kernel size is 3×3. The number of fully connected layers is 2 with 100 neurons, using activation function "ReLU," and the keep probability of dropout is 0.9. The learning rate is 0.0001, the weight decay factor of learning rate is 0.98, and the batch size is 100.

In order to make sure that all models could learn the same knowledge from the dataset, when training the comparison models, we flatten the attribute-based features into a 576-dimensional vector. Furthermore, the vector is concatenated with graph-based features, and the dimension of total input vector is $576 + 32 = 608$.

We randomly divide the negative samples and positive samples of the dataset into three subsets 8 : 1 : 1, respectively, and combine the corresponding positive and negative example subsets into training (80%), validation (10%), and test (10%) sets. In order to handle the imbalanced category problem between fraudulent and nonfraudulent apps, we adopt upsampling technique during training.

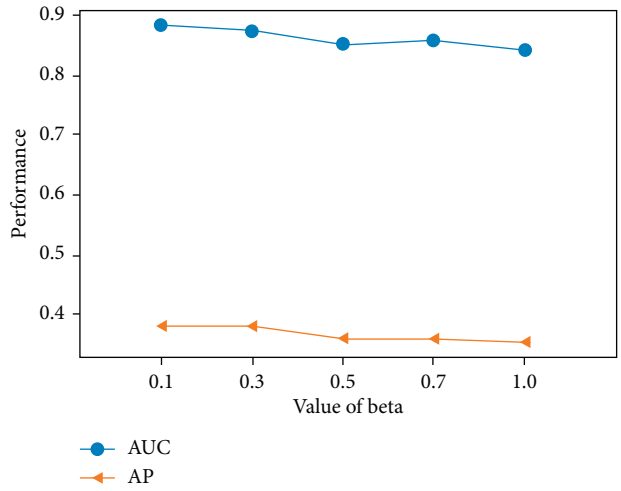


Figure 6: Comparing impacts of the weighted bias β with AUC and AP.

3.4.3. Experimental Results. The experimental results are shown in Tables 4 and 5. The HNN model proposed in this study reaches the highest AP value in six days and the highest AUC value in four days over all seven days. The FCNN, RF, and SVM models have similar performance to AUC measure, and Table 4 shows that HNN > FCNN > RF > SVM with AP measure. Thus, HNN outperforms all other models in terms of AP and AUC measures.

3.4.4. Comparative Experiments without Graph-Based Features. To show the contribution of graph-based feature extraction in proposed GFD approach, we remove the graph-based features in our dataset. When the proposed HNN model has only attribute-based features as input and no graph-based features as input, the HNN model leaves only the fully connected part to work, since the convolution part of HNN model has no input. This also means that the working HNN model would change to a fully connected neural network, that is, FCNN model, in this setting. So we use the SVM, RF, and FCNN models in this comparative experiment. The results are shown in Tables 6 and 7. Comparing the performances of models with/without graph-based features in Tables 4 and 5 and Tables 6 and 7, we could find that the FCNN model with graph-based features reaches better performance than the model without the graph-based features in both AP and AUC measures, while the performance improvement of SVM and RF models is not obvious with graph-based features.

3.4.5. Impacts of Parameters. (1). Time Windows t. Time window in attribute-based feature extraction of GFD approach decides the dimension of attribute-based features. We designed experiments to show the impact of time window, and the result is shown in Table 8. The size of time window is set to be 1, 3, and 6 hours. The continuous increase in size of time window makes HNN perform worse AP values. The other models seem to be not sensitive to the size of time window.

(2). Number of Convolutional Layers in HNN Model. We compare the effect of the number of convolutional layers of 1, 2, and 3 in HNN model and show the results in Table 9. The AUC and AP values achieve a high level when the number of convolutional layers is 2.

(3). Number of Fully Connected Layers in HNN Model. We set the number of fully connected layers to be from 1 to 4, and the experiment result is shown in Table 10. When the number of fully connected layers is 2, the HNN model reaches the highest performance.

(4). Activation Functions in HNN Model. We compare three well-known activation functions, ReLU, tanh, and Sigmoid, in HNN model, and the experiment results are shown in Table 11. The AUC values of the models with different activation functions are similar, and ReLU is slightly better than others. In terms of AP, ReLU is obviously better than the other two activation functions.

4. Related Work

Our work is related to existing studies on attribute-based fraud detection and graph-based fraud detection with machine learning. The challenges of fraud detection problem in mobile advertising system are summarized as accuracy requirement, throughput requirement, and the ability to combat the latest fraud methods [1].

Attribute-based fraud detection approaches have been used in fraud detection domain. Crussell et al. [26] built decision trees based on the features extracted from their dataset for classification. Liu et al. [27] proposed a binary SVM classifier to determine whether two UIs are likely to lead to equivalent states. This classification is used to simulate user interaction in the context of ad clicking. In order to classify malicious publishers, Mouawi et al. [11] evaluated KNN, SVM, and ANN based on features extracted from dataset, and the experimental results show that all three classifiers give very promising result. Haider et al. [2] proposed an ensemble-based method to classify each

Table 4: The comparison of models with AP.

Model	1st June	2nd June	3rd June	4th June	5th June	6th June	7th June
SVM	0.443	0.552	0.566	0.321	0.394	0.261	0.353
RF	0.547	0.329	0.645	0.505	0.428	0.636	0.489
FCNN	0.584	0.678	0.638	0.541	0.574	0.538	0.541
HNN	0.632	0.689	0.752	0.567	0.586	0.592	0.592

Table 5: The comparison of models with AUC.

Model	1st June	2nd June	3rd June	4th June	5th June	6th June	7th June
SVM	0.941	0.960	0.970	0.942	0.961	0.916	0.937
RF	0.919	0.942	0.949	0.937	0.944	0.938	0.945
FCNN	0.876	0.956	0.936	0.965	0.957	0.940	0.958
HNN	0.919	0.952	0.981	0.962	0.965	0.954	0.963

Table 6: AP of SVM, RF, and FCNN without graph-based features.

Model	1st June	2nd June	3rd June	4th June	5th June	6th June	7th June
SVM	0.389	0.547	0.449	0.393	0.304	0.357	0.366
RF	0.526	0.490	0.639	0.512	0.507	0.628	0.536
FCNN	0.433	0.605	0.636	0.528	0.547	0.526	0.517

Table 7: AUC of SVM, RF, and FCNN without graph-based features.

Model	1st June	2nd June	3rd June	4th June	5th June	6th June	7th June
SVM	0.889	0.944	0.947	0.950	0.951	0.930	0.938
RF	0.908	0.940	0.945	0.932	0.950	0.932	0.948
FCNN	0.907	0.950	0.931	0.940	0.933	0.913	0.951

Table 8: AUC and AP of HNN, FCNN, RF, and SVM with different time widows.

Model	AUC			AP		
	1 hour	3 hours	6 hours	1 hour	3 hours	6 hours
HNN	0.95	0.91	0.91	0.63	0.56	0.44
FCNN	0.88	0.89	0.87	0.58	0.56	0.54
RF	0.92	0.92	0.92	0.55	0.48	0.55
SVM	0.94	0.95	0.95	0.44	0.45	0.45

Table 9: AUC and AP of HNN with different number of convolution layers.

Model	AUC			AP		
	1	2	3	1	2	T3
HNN	0.922	0.942	0.950	0.605	0.630	0.405

Table 10: AUC and AP of HNN with different number of fully connected layers.

Model	AUC				AP			
	1	2	3	4	1	2	3	4
HNN	0.926	0.935	0.903	0.863	0.575	0.660	0.649	0.632

individual ad display as fraudulent or nonfraudulent. Gabriel et al. [28] evaluated the performance of logistic regression, gradient trees, and deep learning method in credit card fraud detection and proved that deep learning method outperforms the other compared methods.

Graph-based fraud detection approaches have been studied recently. Hu et al. [15] proposed a weighted graph propagation algorithm to identify the fraudulent apps in the user-app bipartite graphs. Vasumati et al. [29] applied decision trees to classify spam publishers based on constructed

Table 11: AUC and AP of HNN with different activation functions.

Model	AUC			AP		
	ReLU	tanh	Sigmoid	ReLU	tanh	Sigmoid
HNN	0.926	0.923	0.921	0.634	0.625	0.623

feature vector and computed spam score for each of the spam publishers by constructing a bipartite graph between users and publishers to find fraud publishers. What is more, the natural language processing (NLP) models known as Word2vec [23] have been applied to graph embedding, such as DeepWalk [10], Node2vec [21], and Metapath2vec [22]. Zheng et al. [30] proposed an unsupervised method to detect abnormal users and items through deep joint network embedding. Yu et al. [16] proposed a deep embedding approach for anomaly detection in dynamic networks by learning network representations which can be updated dynamically as the network evolves.

Mobile advertising fraud detection is still challenging; however, ensemble learning methods were usually the winner algorithms in fraud detection competition [10], and deep learning and graph learning are recently the most promising methods in this area.

There are two key differences between our proposed approach and existing works. First, we used app id, ad id, and user id from the real-world dataset to construct a weighted heterogeneous graph with these three types of nodes and proposed the graph embedding algorithm for mobile advertising fraud detection. The popular existing datasets, such as TalkingData dataset [31], usually have one or two types of entities (e.g., app id), so there are not enough entities to construct a heterogeneous graph as we did in this paper. Second, we proposed a fusing model to combine attribute-based and graph-based information for mobile advertising fraud detection by graph embedding and deep learning methods.

5. Conclusion

In this paper, we focus on the fraud detection problem in mobile advertising to detect fraudulent publishers. We propose a novel weighted heterogeneous graph and deep learning-based fraud detection approach, namely, GFD, to identify fraudulent apps for mobile advertising. Based on the relationship of users, publishers, and advertisement in mobile ad system, we construct a weighted heterogeneous graph and proposed a weighted metapath based graph embedding approach, named WMP2vec, to learn structural features of publishers in the graph. Furthermore, we construct a hybrid convolutional neural network to learn high-order features from attribute-based features and graph-based features. The experimental results in a real-world dataset show that our method is effective in classifying fraudulent apps for mobile advertising system.

There are two limitations in the work presented here. First, the dataset is limited to one mobile advertising dataset. In order to be more generalizable, it would be important to see whether the proposed GFD approach excels in more fraud detection datasets. Second, the dataset is limited to seven days. In the complex and dynamic online advertising environment, more time is still needed to evaluate the proposed approach.

Despite being focused on mobile advertising fraud detection in this presentation, the proposed GFD approach could be generalized to benefit many other online applications (e.g., e-commerce) that involve relationship between several types of entities. Future work should focus on the robustness and accuracy of our proposed model for other large-scale online datasets.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This work was supported in part by the Natural Science Foundation of Guangdong Province of China (Grant no. 2018A030313309), the Innovation Fund of Introduced High-End Scientific Research Institutions of Zhongshan (Grant no. 2019AG031), and the Fundamental Research Funds for the Central Universities, SCUT (Grant no. 2019KZ20).

References

[1] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna, "The dark alleys of madison avenue: understanding malicious advertisements," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 373–380, Vancouver, BC, Canada, November 2014.

[2] C. M. R. Haider, A. Iqbal, A. H. Rahman, and M. S. Rahman, "An ensemble learning based approach for impression fraud detection in mobile advertising," *Journal of Network and Computer Applications*, vol. 112, pp. 126–141, 2018.

[3] V. Dave, S. Guha, and Y. Zhang, "Measuring and fingerprinting click-spam in ad networks," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 175–186, ACM, New York, NY, USA, August 2012.

[4] H. Haddadi, "Fighting online click-fraud using bluff ads," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 21–25, 2010.

[5] A. Rodrigo, R. JGB de Queiroz and E. R. Cavalcanti, A proposal to prevent click-fraud using clickable captchas," in *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability Companion*, pp. 62–67, IEEE, Gaithersburg, MD, USA, June 2012.

[6] D. Antoniou, M. Paschou, E. Sakkopoulos et al., "Exposing click-fraud using a burst detection algorithm," in *Proceedings of the 2011 IEEE Symposium on Computers and*

Communications (ISCC), pp. 1111–1116, IEEE, Kerkyra, Greece, June 2011.

[7] W. Li, H. Li, H. Chen, and Y. Xia, "Adattester: secure online mobile advertisement attestation using trustzone," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 75–88, Florence Italy, May 2015.

[8] J. Kwon, J. Kim, J. Lee, H. Lee, and P. Adrian, "PsyBoG: power spectral density analysis for detecting botnet groups," in *Proceedings of the 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*, pp. 85–92, IEEE, Fajardo, PR, USA, October 2014.

[9] M. Faou, L. Antoine, D. Décary-Hétu et al., "Follow the traffic: stopping click fraud by disrupting the value chain," in *Proceedings of 2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pp. 464–476, IEEE, Auckland, New Zealand, December 2016.

[10] R. Oentaryo, Ee-P. Lim, M. Finegold et al., "Detecting click fraud in online advertising: a data mining approach," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 99–140, 2014.

[11] R. Mouawi, M. Awad, C. Ali, H. Imad, H. El, and A. Kayssi, "Towards a machine learning approach for detecting click fraud in mobile advertizing," in *Proceedings of 2018 International Conference on Innovations in Information Technology (IIT)*, pp. 88–92, IEEE, Al Ain, United Arab Emirates, November 2018.

[12] G. S. Thejas, K. G. Boroojeni, K. Chandna, I. Bhatia, S. S. Iyengar, and N. R. Sunitha, "Deep learning-based model to fight against ad click fraud," in *Proceedings of the 2019 ACM Southeast Conference*, pp. 176–181, ACM, Kennesaw, GA, USA, April 2019.

[13] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17*, ACM, Halifax, NS, Canada, pp. 1–7, August 2017.

[14] G. S. Thejas, J. Soni, K. G. Boroojeni et al., "A multi-time-scale time series analysis for click fraud forecasting using binary labeled imbalanced dataset,"Proceedings of 2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), vol. 4, pp. 1–8, IEEE, Bengaluru, India, December 2019.

[15] J. Hu, J. Liang, and S. Dong, "iBGP: a bipartite graph propagation approach for mobile advertising fraud detection," *Mobile Information Systems*, vol. 2017, Article ID 6412521, 2017.

[16] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: a flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2672–2681, London, UK, July 2018.

[17] L. Bertrand, F. Braun, C. Olivier, and M. Saerens, *A Graph-based, Semi-supervised, Credit Card Fraud Detection System: International Workshop on Complex Networks and Their Applications*, Springer, Cham, Switzerland, 2016.

[18] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: a survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[19] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: a survey," *IEEE transactions on Big Data*, vol. 6, no. 1, pp. 3–8, 2018.

[20] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 135–144, ACM, Halifax, NS, Canada, August 2017.

[21] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, https://arxiv.org/abs/1301.3781.

[22] A. Mnih and K. Kavukcuoglu, "Learning word embeddings efficiently with noise-contrastive estimation," in *Proceedings of Advances in Neural Information Processing Systems*, NIPS, Lake Tahoe, Nevada, pp. 2265–2273, January 2013.

[23] P. Bryan, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, New York, NY, USA, pp. 701–710, August 2014.

[24] A. Grover and J. Leskovec, "node2vec: scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, New York, NY, USA, pp. 855–864, August 2016.

[25] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," *Aistats*, vol. 5, pp. 246–252, 2005.

[26] J. Crussell, R. Stevens, and H. Chen, "Madfraud: investigating ad fraud in android applications," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 123–134, Bretton Woods, NH, USA, June 2014.

[27] B. Liu, S. Nath, R. Govindan, and J. Liu, "DECAF: detecting and characterizing ad fraud in mobile apps," in *Proceedings of 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp. 57–70, Renton, WA, USA, April 2014.

[28] R. Gabriel, C. Stancil, M. Sun, S. Adams, and B. Peter, "Horse race analysis in credit card fraud—deep learning, logistic regression, and gradient boosted tree," in *Proceedings of 2017 Systems and Information Engineering Design Symposium (SIEDS)*, pp. 117–121, IEEE, Charlottesville, VA, USA, April 2017.

[29] D. Vasumati, M. Sree Vani, R. Bhramaramba, and O. Yaswanth Babu, "Data mining approach to filter click-spam in mobile ad networks," in *Proceedings of Int'l Conference on Computer Science, Data Mining & Mechanical Engg.(ICCDMME'2015)*, Bangkok, Thailand, April 2015.

[30] M. Zheng, C. Zhou, J. Wu, S. Pan, J. Shi, and Li Guo, "Fraudne: a joint embedding approach for fraud detection," in *Proceedings of 2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, Rio de Janeiro, Brazil, July 2018.

[31] Kaggle Inc, "Talking data adtracking fraud detection challenge can you detect fraudulent click traffic for mobile app ads?," 2018, https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 8840058, 11 pages
https://doi.org/10.1155/2020/8840058

Research Article

An Efficient and Effective Approach for Flooding Attack Detection in Optical Burst Switching Networks

Bandar Almaslukh

Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia

Correspondence should be addressed to Bandar Almaslukh; b.almaslukh@psau.edu.sa

Received 27 March 2020; Revised 8 July 2020; Accepted 11 July 2020; Published 5 August 2020

Academic Editor: Muhammad Faisal Amjad

Optical burst switching (OBS) networks are frequently compromised by attackers who can flood the networks with burst header packets (BHPs), causing a denial of service (DoS) attack, also known as a BHP flooding attack. Nowadays, a set of machine learning (ML) methods have been embedded into OBS core switches to detect these BHP flooding attacks. However, due to the redundant features of BHP data and the limited capability of OBS core switches, the existing technology still requires major improvements to work effectively and efficiently. In this paper, an efficient and effective ML-based security approach is proposed for detecting BHP flooding attacks. The proposed approach consists of a feature selection phase and a classification phase. The feature selection phase uses the information gain (IG) method to select the most important features, enhancing the efficiency of detection. For the classification phase, a decision tree (DT) classifier is used to build the model based on the selected features of BHPs, reducing the overfitting problem and improving the accuracy of detection. A set of experiments are conducted on a public dataset of OBS networks using 10-fold cross-validation and holdout techniques. Experimental results show that the proposed approach achieved the highest possible classification accuracy of 100% by using only three features.

1. Introduction

Optical burst switching (OBS) in networks has become an important dynamic sub-wavelength switching technique and a solution for developing the new type of Internet backbone infrastructure [1]. The OBS network mainly consists of three types of nodes, namely, core nodes, ingress, and egress. The core nodes represent the intermediate nodes, which are designed to reduce the processing and buffering of the optical data burst using a control data packet with specific information, namely, burst header packets (BHPs) [2].

In a network with burst traffic, OBS plays an essential role for packet switching with a higher level of necessary details than other existing networks' switching techniques. However, this type of switching is still suffering from several challenges such as security and quality of service (QoS) due to BHP flooding attacks. The function of BHP in OBS is to reserve the unused channel for the arrival of a data burst (DB). This function can be exploited by attackers to send fake BHPs without DB acknowledgment. Such fake BHPs can affect the network and reduce its performance through decreasing bandwidth utilization and increasing data loss, leading to a denial of service (DoS) attack [3], which is one of the most crucial security threats to networks.

Several methods have been proposed to tackle DoS and BHP flooding attacks on OBS networks in the literature and have achieved satisfactory results [4–6]. However, due to the limited capability of OBS core switches, developing a lightweight method that can attain high accuracy with a small number of features is still a challenging issue for developers and researchers.

In this research, an effective and efficient approach is proposed for securing the OBS networks. Thus, the main objective of the work is to develop a lightweight ML model for detecting BHP flooding attacks based on the information gain (IG) feature selection method and a decision tree (DT) classifier. To achieve this objective, two key research

questions are formulated to answer throughout this study. The first research question is does the feature selection method improve the effectiveness of the DT model to detect the BHP flooding attacks. The second research question is does the feature selection method improve the efficiency of the DT model for detecting the BHP flooding attacks. Actually, the lightweight property of the model comes from the fact that only a small number of features are used to build the classifier. The model will be evaluated using a public OBS dataset based on a set of performance metrics such as accuracy, precision, recall, and *F*-measure.

The remainder of the research is organized as follows:

(i) In Section 2, related works are introduced to give details about the proposed approaches and methods of DoS attack on different networks.

(ii) Section 3 presents the proposed approach architecture for detecting the BHP flooding attacks on OBS networks.

(iii) Section 4 explains the experimental setup and results in more detail.

(iv) Section 5 presents the conclusion of the study.

2. Related Works

Nowadays, machine learning (ML) methods have been used in many intrusion detection systems (IDSs) to detect several types of network attacks. However, feature selection methods are also used to select the significant features of network traffic without reducing the performance of the IDSs [7]. Feature selection is the process of selecting the best set of features that can be most effective for classification tasks [8, 9]. The high number of features may decrease the performance and accuracy of many classification problems [10, 11].

In the field of optimization, feature selection methods are classified in three main approaches: embedded, wrapper, and filter methods [12]. For the filter methods, there are two major types of evaluation: subset feature evaluation and groups of individual feature evaluation. In the groups of individual feature evaluation, heuristic or metaheuristic filter methods or even the hybrid of them is utilized for ranking the features and then the best of them is selected based on some thresholds [11, 13]. In contrast, the subset feature evaluation methods find the subset of candidate features using a certain measure or a certain strategy. They compare the previous best subset with the current subset for finding the candidate subset of features. In the groups of individual feature evaluation methods, the redundant features are kept in the final subset of selected features according to their relevance but the group of subset feature evaluation methods removes the features with similar ranks. In general, the filter methods are considered as classifier-independent approaches [13]. The wrapper methods are classifier-dependent approaches that take each time a subset of features from the total features and calculate the accuracy of classifiers to find the best subset. Therefore, they are time consuming compared with filter methods [14]. The embedded methods combine wrapper and filter methods [15]. In this study, a filter-based method is used for feature selection.

In the literature review of intrusion detection, a set of ML and deep learning (DL) methods have been widely used to detect different types of attacks in several works [16–20]. Meanwhile, a set of related works have also been proposed for detecting BHP flooding attacks using different ML methods like the decision tree (DT) method in [21]. This work evaluated the performance of the adopted method using different metrics and reported a 93% accuracy rate in classifying the classes of BHP flooding attack. Liao et al. [22] introduced a classification approach to classify the access patterns of various users using sparse vector decomposition (SVD) and rhythm matching methods. This study demonstrates that the approach is able to distinguish between the intruders and the legal users in the application layer.

Xiao et al. [23] offered an effective scheme for detecting a distributed DoS attack (DDoS) using the correlation of the information generated by the data center and the k-nearest neighbors (KNNs) method. They analyzed the flows of data traffic at the center to identify normal and abnormal flows. In [24], the authors proposed an approach for detecting DDoS attacks based on seven features and using an artificial neural network (ANN) method with a radial basis function (RBF). This NN-RBF approach can classify the data traffic into attack or normal classes by sending the IP address of the incoming packets from the source nodes to be filtered in the alarm modules which then decide if these data packets can be sent to the destination nodes.

The authors in [25] applied a data mining method for detecting a DDoS attack using the fuzzy clustering method (FCM) and a priori association algorithm to categorize the data traffic patterns and the status of the network. Another ML approach in [26] used a DT method with a grey relational analysis for detecting DDoS attacks. They also applied the pattern matching technique to the data flows for tracing back the estimated location of the attackers.

Alshboul [27] investigated the use of rule induction nodes for BHP classification in OBS networks. The author applied a set of data mining methods to the public OBS network dataset. He reported that the repeated incremental pruning to produce error reduction (RIPPER) rule induction algorithm, Naïve Bayes (NB), and Bayes Net were able to achieve a predictive accuracy of 98%, 69%, and 85%, respectively.

Chen et al. [28] developed a detection method to identify a DDoS attack using ANN. A set of different simulated DoS attacks were used for training the ANN model to recognize abnormal behaviors. Li et al. [29] offered different types of ANN models, including learning vector quantization (LVQ) models, to differentiate traffic associated with DDoS attacks from normal traffic. The authors converted the values of the dataset features into a numerical format before feeding them into the ANN model.

In [30], the authors presented a probabilistic ANN approach for classifying the different types of DDoS attacks. They categorized the DDoS attacks and normal traffic by applying radial basis function neural network (RBF-NN)

coupled with a Bayes decision rule. Nevertheless, the approach concentrated on the events of unscrambling flash crowds generated by DoS attacks.

Li and Liu [31] proposed a technique that integrates the network intrusion prevention system with SVM to improve the accuracy of detection and reduce the incidents of false alarms. In [32], Ibrahim offers a dynamic approach based on distributed time-delay ANN with soft computing methods. This approach achieved a fast conversion rate, high speed, and a high rate of anomaly detection for network intrusions.

Gao et al. [33] introduced a data mining method for analyzing the piggybacked packets of the network protocol to detect DDoS attacks. The advantage of this method is to retain a high rate of detection without manual data construction. Hasan et al. [34] proposed a deep convolutional neural network (DCNN) model to detect BHP flooding attacks on OBS networks. They reported that the DCNN model works better than any other traditional machine learning models (e.g., SVM, Naïve Bayes, and KNN). However, due to the small number of samples in the dataset and the limited resource constraints of OBS switches, such deep learning models are not effective tools to detect BHP flooding attacks and they are not computationally efficient to run in such network.

3. Proposed Approach

The proposed approach in this paper consists of two main phases: feature selection and classification. The input of the approach is a set of OBS dataset features collected from network traffic. The output of the approach is a class label of the BHP flooding attacks. The flowchart of the proposed approach is illustrated in Figure 1.

In the feature selection phase of the approach, the input features of OBS network traffic are prepared for processing by using the information gain (IG) feature selection method. The purpose of IG is to rank the features and discover the merit of each of them according to the information gain evaluation of the entropy function. The output of the feature selection phase is a scored rank of features in decreasing order according to their merit, whereby adding any feature decreases the features merit.

This is then followed by the classification phase, in which the dataset with selected features will be used to train and test the DT classifier to detect attacks on OBS networks. The output of the classification phase is a DT trained model that is able to classify the BHP flooding attacks and return the class label of that attack. The following sections explain the methods used in the two phases of the proposed approach.

3.1. Information Gain (IG) Feature Selection Method. Information gain (IG) is a statistical method used to measure the essential information for a class label of an instance based on the absence or presence of the feature in that instance. IG computes the amount of uncertainty that can be reduced by including the features. The uncertainty is usually calculated by using Shannon's entropy (E) [35] as

$$E(D) = \sum_{i=1}^{n} P_i \log_2 (P_i), \tag{1}$$

where n represents the number of class labels and P_i is the probability that an instance i in a dataset D can be labeled as a class label c by computing the proportion of instances that belong to that class label for the instance i as follows:

$$\frac{|D_{i \in C}|}{D}. \tag{2}$$

A selected feature f divides the training set into subsets $D_1, D_2, \ldots, D_v$ according to the values of f, where f has v distinct values. The information required to get the exact classification is measured by

$$\text{Reminder}(f) = \sum_{j=1}^{v} \frac{|D_j|}{D} \times E(DJ), \tag{3}$$

where $|D_j|/D$ represents the weight of j^{th} subset, $|D|$ is the number of instances in the dataset D, $|D_j|$ is the number of instances in the subset D_j, and $E(D_j)$ is the entropy of the subset D_j. Therefore, the IG of every feature is calculated as

$$\text{IG}(f) = E(D) - \text{Reminder}(f). \tag{4}$$

After calculating the IG for each feature, the top k features with the highest IG will be selected as a feature set because it reduces the information required to classify the flooding attack.

3.2. Decision Tree Method. Decision tree (DT) is a tree-like model of decisions with possible consequences that is commonly used in the fields of data mining, statistics, and machine learning [36]. In machine learning, the goal of DT is to build a model that predicts or classifies the value of a target class based on a learning process from several input features. The tree model that has a target class label with discrete values is called a classification tree model. In this model, the tree leaves constitute the values of the class label and the tree branches constitute aggregations of features that produce this class label.

DT learning is a simple process to represent the features for predicting or classifying instances. DT models are created by splitting the input feature set into subsets that establish the successor nodes of the children, thereby establishing the tree root node. Based on a set of splitting rules on the values of the features, the splitting process for each derived subset is repeated in a recursive manner [36]. This recursive manner is stopped when the splitting process no longer adds values to the predictions or when the subset of nodes have all the same values of the target class label.

The DT can be described also as a mathematical model to support the categorization, description, and generalization of a given dataset.

Assume the dataset comes in the form of records as follows:

$$(x, y) = (x_1, x_2, x_3, \ldots, x_k, y), \tag{5}$$

where the variable y is a dependent target variable that we need to generalize or classify. The vector x consists of the features $x_1, x_2, x_3, \ldots, x_k$, which are led to the variable y.

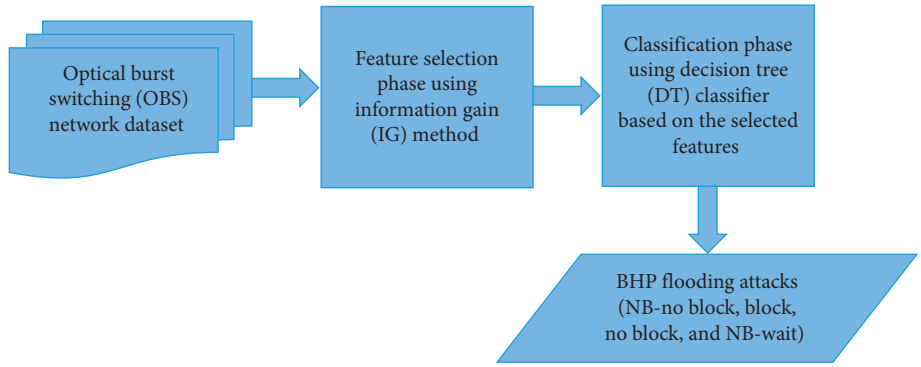


Figure 1: Flowchart of the proposed approach.

In principle, the DT is based on the C4.5 algorithm [37], which is an updated version of the ID3 algorithm [38]. C4.5 can avoid the overfitting problem of ID3 by using the rule-post pruning technique to convert the building tree into a set of rules.

DT is used in the proposed approach because it is simple, very intuitive, and easy to implement. Furthermore, it deals with missing values, requires less effort in terms of data preprocessing, and does not need to scale or normalize the data [36].

4. Experiments and Discussion

The experiments of this research are implemented using a popular open source tool called the Waikato Environment for Knowledge Analysis (Weka) software [39], which offers a rich toolbox of machine learning and data mining methods for preprocessing, analyzing, clustering, and classification. It offers Java-based graphical user interfaces (GUIs). The implementation was performed on a laptop with an Intel Core i7 CPU processor, 2.0 GHz, 8 GB RAM, and a Windows 10 64 bit operating system. Due to the scarcity of OBS historical data, the experiments were conducted on a public optical burst switching (OBS) network dataset [1].

4.1. OBS Network Dataset Description. The OBS network dataset is a public dataset, available from the UCI Machine Learning Repository [1]. It contains a number of BHP flooding attacks on OBS networks. There are 1,075 instances with 21 attributes as well as the target class label. This target label has four types of classes, which are NB-no block (not behaving-no block), block, no block, and NB-wait (not behaving-wait). All dataset features have numeric values except for the node status feature that takes a categorical value out of three values: B (behaving), NB (not behaving), and potentially not behaving (PNB). The description of the dataset features is given in Table 1.

Table 2 shows the number of instances for each class in the dataset, while Figure 2 shows the distribution of instances over different types of BHP flooding attacks. This figure is deduced from the dataset.

4.2. Evaluation Measures. The experimental results will be evaluated using four evaluation measures. These measures are precision, recall, F-measure, and accuracy. The following equations show how these evaluation measures are computed:

$$\begin{aligned} \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{recall}\,(\text{sensitivity}) &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ F - \text{measure} &= 2 * \frac{(\text{precision} * \text{recall})}{(\text{precision} + \text{recall})}, \\ \text{accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \end{aligned} \tag{6}$$

where FP is the number of false positives, FN is the number of false negatives, TP is the number of true positives, and TN is the number of true negatives.

4.3. Results and Comparisons. In this section, the experimental results for both the feature selection and classification phases of the proposed approach are given in detail. The average rank score and average merit of features from the IG feature selection method are shown in Table 3 and are based on a 10-fold cross-validation with stratified sampling in order to guarantee that both training and testing sets have the same ratio of classes.

In Table 3, the dataset features are ranked in decreasing order according to their significance to target classes. The reason behind this variation in the feature significance is that the target class has four categorical labels, and for each label, different values for each feature are assigned. Therefore, the rank score from the IG method determines how much each feature contributes to the target class label.

The rank scores in Table 3 show that the "packet received," "10-run-AVG-drop-rate," and "flood status" features have higher scores than all the other features. Thus, the hypothesis that those first three features (packet received, 10-run-AVG drop-rate, and flood status) are more influential and more correlated to the labels of target class will be checked experimentally in the following paragraphs.

Table 1: The description of the dataset features.

No.	Feature name	Feature description
1	Node	It is a numeric feature representing the number of node that sends the data traffic.
2	Utilized bandwidth rate	It is a numeric feature representing the rate of bandwidth used.
3	Packet drop rate	It is a numeric feature representing the rate of packet drop.
4	Reserved bandwidth	It is a numeric feature denoting the initial reserved bandwidth assigned to a given node.
5	Average delay time per sec	It is a numeric feature denoting the average delay time per second for each node. It is also called end-to end delay feature.
6	Percentage of lost packet rate	It is a numeric feature representing the percentage rate of lost packets for each node.
7	Percentage of lost byte rate	It is a numeric feature representing the percentage rate of lost bytes for each node.
8	Packet received rate	It is a numeric feature representing the packet received rate per second for each node based on the reserved bandwidth.
9	Used bandwidth	It is a numeric feature represents the bandwidth used or what each could reserve from the reserved bandwidth.
10	Lost bandwidth	It is a numeric feature denoting the lost amount of bandwidth by each node from the reserved bandwidth.
11	Packet size byte	It is a numeric feature denoting the packet size in bytes allocated explicitly for each node to transmit. For instance, if the data size is 1440 bytes and there are 60 bytes for (IP header 40 bytes) + (UDP header 20 bytes), then all headers will be added to the data size to get 1500 byte as follows: packet size = ((data size 1440 bytes) + (IP header 40 bytes) + (UDP header 20 bytes)) = 1500 bytes.
12	Packet transmitted	This is a numeric feature representing the total packets transmitted per second for each node based on the reserved bandwidth.
13	Packet received	This is a numeric feature representing the total packets received per second for each node based on the reserved bandwidth.
14	Packet lost	This is a numeric feature representing the total packets lost per second for each node based on the lost bandwidth.
15	Transmitted byte	This is a numeric feature representing the total bytes transmitted per second for each node.
16	Received byte	It is a numeric feature denoting the total bytes received per second for each node based on the reserved bandwidth.
17	10-run-AVG-drop-rate	This is a numeric feature representing the rate of average packets that drop for 10 consecutive iterations and runs.
18	10-run-AVG-bandwidth-use	It is a numeric feature representing the average bandwidth that is utilized for 10 consecutive iterations and runs.
19	10-run-delay	This is a numeric feature representing the time of average delay for 10 consecutive (run) iterations.
20	Node status	This is a categorical feature. It is an initial classification of nodes based on the rate of packet drop, used bandwidth, and average delay time per second. The categorical values are B for behaving, NB for not behaving, and PNB for potentially not behaving.
21	Flood status	This is a numeric feature that represents the percentage of flood per node. It is based on the packet drop rate, medium, and high level of BHP flood attack in case behaving (B).
22	Class label	This feature is a categorical feature that represents the final classification of nodes based on the packet drop rate, reserved bandwidth, number of iterations, used bandwidth, and packet drop rate. The categorical values of the class label are NB-no block, block, no block, and NB-wait

Table 2: The number of instances for each class in the OBS network dataset.

Class label	NB-no block	Block	No block	NB-wait	Total
No. of instances	500	120	155	300	1075

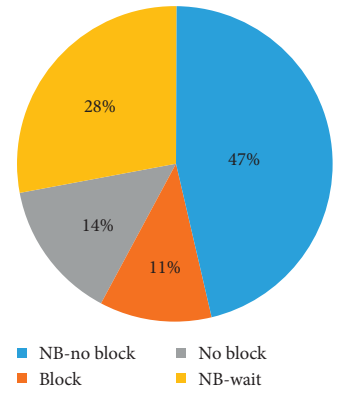


Figure 2: The distribution of instances for each class in the OBS network dataset.

To accept or reject this hypothesis, the evaluation results of the DT method are presented using all features and the combinations of the three selected features. These evaluation results are reported based on the holdout and 10-fold cross-validation techniques. For the holdout technique, the dataset is divided into 75% for training and 25% for testing. Before applying the DT method for classifying the types of BHP flooding attacks and getting the results, an analysis of the DT

Table 3: Rank score of IG feature selection method for all features in the dataset.

Feature name	Feature no.	Average rank	Average merit
Packet received	13	1.8 ± 1.17	1.402 ± 0.079
10-run-AVG-drop-rate	17	4.1 ± 1.64	1.306 ± 0.06
Flood status	21	4.2 ± 1.78	1.309 ± 0.052
Used bandwidth	9	4.3 ± 3.66	1.285 ± 0.191
10-run-AVG-bandwidth-use	18	4.3 ± 1.62	1.282 ± 0.122
Received byte	16	5.3 ± 3.35	1.241 ± 0.163
Packet lost	14	6.7 ± 3.32	1.175 ± 0.15
Packet drop rate	3	9.3 ± 2	1.053 ± 0.083
Packet received rate	8	9.8 ± 1.6	1.018 ± 0.043
Percentage of lost byte rate	7	9.8 ± 1.08	1.018 ± 0.044
Utilized bandwidth rate	2	10 ± 2.32	1.05 ± 0.074
Percentage of lost packet rate	6	10.8 ± 1.08	1.009 ± 0.017
Average delay time	5	12 ± 2.79	0.9 ± 0.19
Reserved bandwidth	10	12.8 ± 1.66	0.899 ± 0.1
Node status	20	14.9 ± 0.3	0.488 ± 0.004
10-run-delay	19	16.2 ± 0.98	0.36 ± 0.104
Full bandwidth	4	17.2 ± 0.6	0.146 ± 0.007
Transmitted byte	15	17.7 ± 0.46	0.146 ± 0.007
Packet transmitted	12	18.8 ± 0.6	0.146 ± 0.007
Node	1	20.1 ± 0.3	0.017 ± 0.006
Packet size byte	11	20.9 ± 0.3	0 ± 0

parameters is investigated to tune and select the best values of these parameters.

Practically, the DT classifier (J48) in Weka performs the pruning process based on a set of parameters, which are the subtree raising, the confidence factor, and the minimal number of objects. The default values of these parameters are true, 0.25, and 2, respectively. The subtree raising is the parameter that can be used to move the node of the tree upwards towards the root that can replace other nodes during the pruning process. Confidence factor is a threshold of acceptable error in data through pruning the DT and this value should be smaller. However, in the proposed approach, the values of subtree raising and confidence factor parameters are set to have the default values. The minimal number of objects is very important parameter to represent the minimal number of nodes in a single leaf. It is used to obtain smaller and simpler decision trees based on the nature of the problem. For tuning the minimal number of objects parameter, we try a set of different values for selecting the best value of this parameter. Figure 3 shows the accuracies of proposed approach at different values of minimal number of objects in the range from 2 to 5. These accuracies are obtained using the holdout technique with 75% training and 25% testing.

As shown in Figure 3, it is clear that the best values of minimal number of objects in a single leaf are 1 and 2 that generate a simple and accurate DT model. The value of this parameter is set to be 2 to make the DT model moderately simple.

Once the values of DT parameters are selected, the evaluation results of the proposed approach are reported in

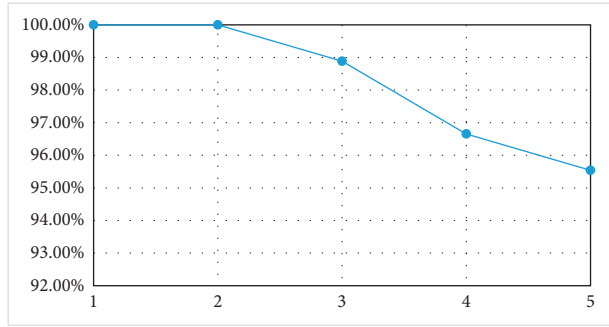


Figure 3: The accuracies of proposed approach at different values of minimal number of objects.

next tables and figures. Table 4 presents the evaluation results of the holdout technique for classifying BHP flooding attacks using all features in the dataset. Figure 4 shows the confusion matrix of classification for the 25% testing set.

Table 5 illustrates the evaluation results of the holdout technique for classifying the BHP flooding attacks using the first three selected features (packet received, 10-run-AVG-drop-rate, and flood status) of the dataset, and Figure 5 shows the confusion matrix of this evaluation result.

From Tables 4 and 5, as well as from Figures 4 and 5, it is clear that the selected features improved the values of evaluation measures for the DT method to classify the BHP flooding attacks. Moreover, for efficiency, detecting attacks using only three features is more efficient for the OBS core switches, which have limited resources.

To validate the evaluation results, other experiments for the DT classification method based on the 10-fold cross-validation technique were conducted using all features and using the first three selected features from the IG feature selection method. Table 6 shows the evaluation results, and Figure 6 shows the confusion matrix for classifying the BHP flooding attacks using all features based on the 10-fold cross-validation technique.

Similarly, Table 7 and Figure 7 present the evaluation results and the confusion matrix, respectively, for classifying the BHP flooding attacks using the first three selected features based on the 10-fold cross-validation technique.

The evaluation results in Tables 6 and 7 and Figures 6 and 7 validate the evaluation results of the 10-fold cross-validation technique that confirm the remarkable performance of the proposed approach. After further investigation, the evaluation results of the DT classification methods using one and two features from the first three selected features are compared with the previous results of the holdout and the 10-fold cross-validation techniques and are shown in Figure 8.

Table 8 shows and summarizes a comparison between the proposed approach and the recent related works on the OBS network dataset. In this comparison, we can see that the proposed work achieves the highest accuracy result with a small number of features compared to all these recent works.

The results presented in Figure 8 and Table 8 prove the hypothesis of the proposed approach that says that the first three selected features using the IG method are more

Table 4: Evaluation results of holdout technique using all features of the dataset.

	Evaluation measure				
Class label	FP rate	Precision	Recall	*F*-measure	Accuracy
NB-no block	0.039	0.950	1.000	0.975	
Block	0.000	1.000	1.000	1.000	
No block	0.000	1.000	1.000	1.000	97.7695
NB-wait	0.000	1.000	0.925	0.961	
Weighted avg.	0.017	0.979	0.978	0.978	

	NB-no block	Block	No block	NB-wait
NB-no block	115	0	0	0
Block	0	31	0	0
No block	0	0	43	0
NB-wait	6	0	0	74

Figure 4: The confusion matrix of classification for the 25% testing set using all features of the dataset.

Table 5: Evaluation results of holdout technique using the first three selected features of the dataset.

	Evaluation measure				
Class label	FP rate	Precision	Recall	*F*-measure	Accuracy
NB-no block	0.000	1.000	1.000	1.000	
Block	0.000	1.000	1.000	1.000	
No block	0.000	1.000	1.000	1.000	100
NB-wait	0.000	1.000	1.000	1.000	
Weighted avg.	0.000	1.000	1.000	1.000	

	NB-no block	Block	No block	NB-wait
NB-no block	115	0	0	0
Block	0	31	0	0
No block	0	0	43	0
NB-wait	0	0	0	80

Figure 5: The confusion matrix of classification for the 25% testing set using the first three selected features of the dataset.

Table 6: Evaluation results of 10-fold cross-validation technique using all features of the dataset.

	Evaluation measure				
Class label	FP rate	Precision	Recall	*F*-measure	Accuracy
NB-no block	0.007	0.992	1.000	0.996	
Block	0.000	1.000	1.000	1.000	
No block	0.000	1.000	1.000	1.000	99.6279
NB-wait	0.000	1.000	0.987	0.993	
Weighted avg.	0.003	0.996	0.996	0.996	

influential and more correlated to the labels of BHP flooding attacks than any of the other features.

4.4. Result Analysis. For analyzing the results and linking the results with conclusion, we show how the proposed feature selection method can improve the model from three different angles: reducing overfitting, improving accuracy, and reducing training and testing (prediction) time.

From the definition of the overfitting problem, it occurs when the training errors are low or very low and the validation errors are high or very high. Therefore, reducing the

	NB-no block	Block	No block	NB-wait
NB-no block	500	0	0	0
Block	0	120	0	0
No block	0	0	155	0
NB-wait	4	0	0	296

Figure 6: The confusion matrix of classification for the 10-fold cross-validation testing sets using all features in the dataset.

Table 7: Evaluation results of 10-fold cross-validation technique using the first three selected features of the dataset.

	Evaluation measure				
Class label	FP rate	Precision	Recall	*F*-measure	Accuracy
NB-no block	0.000	1.000	1.000	1.000	
Block	0.000	1.000	1.000	1.000	
No block	0.000	1.000	1.000	1.000	100
NB-wait	0.000	1.000	1.000	1.000	
Weighted avg.	0.000	1.000	1.000	1.000	

	NB-no block	Block	No block	NB-wait
NB-no block	500	0	0	0
Block	0	120	0	0
No block	0	0	155	0
NB-wait	0	0	0	300

Figure 7: The confusion matrix of classification for the 10-fold cross-validation testing sets using the first three selected features of the dataset.

overfitting problem requires to reduce the gap between the training and validation error. To show how the proposed method can reduce the overfitting problem, we depict the training error against the validation error in Figure 9 with different sets of features, which are ordered according to rank score given in Table 3. The training percentage is set to 75%, and the validation percentage is 25%. We notice that the gap between the training and validation error is decreased as the number of features is decreased until the gap reaches zero approximately when using the three selected features of the proposed method. We also notice that the overfitting problem is eliminated with 14 and 7 features. In our opinion, the overfitting problem is eliminated with 14 and 7 features because of an implicit pruning functionality implemented by the used decision tree algorithm (J48). In addition, it is clear that the accuracy is improved by the three selected features.

To evaluate the efficiency of the proposed feature selection approach, the average time of building and testing the DT model is computed. The DT model is trained on 75% of the dataset which consists of 806 instances and tested on 25% of the dataset which consists of 269 instances. Table 9 shows the computed average time of training and testing the DT model using all features and using our three selected features.

As shown in Table 9, we can see that the DT model has a lower average time for training and testing using our three selected features than using all features. In terms of time complexity, represented by O notation, the overall average time of the DT method is $O(m \times n)$, where m is the number of features and n is the number of instances [40]. Because the number of features in classification problems is limited, the running time will be $O(C \times n)$, where C is a constant time. Therefore, the time complexity of the DT method is $O(n)$ for classification problems. The advantage of the proposed approach is that it reduces the number of features to three features (reducing C), which leads to faster running time compared with using all features. This confirms that the approach is able to detect the attacks more efficiently, especially in congested network with limited computing resources.

We can conclude that reducing the features to three and using the pruning process of the DT classifier helped the proposed approach to reduce the overfitting problem and classify the OBS flooding attacks. Consequently, all performance results clarified the effectiveness and efficiency of the DT model based on selected features to classify BHP flooding attacks. This reveals that the proposed approach is more accurate and suitable for real-time detection in the limited computing capability of OBS core switches.

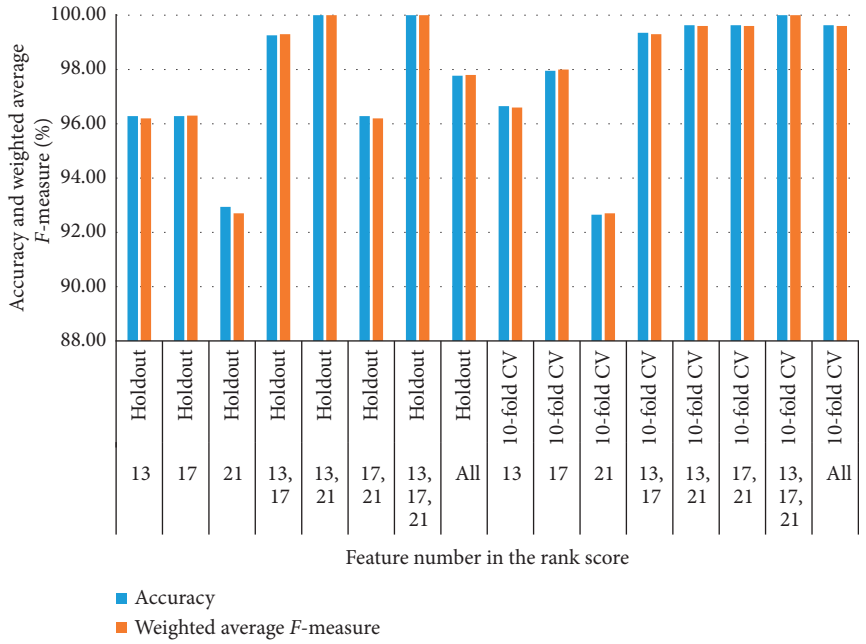


Figure 8: Evaluation results of accuracy and *F*-measure for the DT classification method using a combination of the three selected features compared to all features.

Table 8: Accuracy and number of features of the proposed BHP flooding attack detection approaches using OBS network dataset.

Ref.	Year	Approach	# Of features	Accuracy (%)
[34]	2018	Naïve Bayes	21	79
[21]	2018	Features selection using chi-square testing (CHI) + decision tree for classification	7	87
[34]	2018	Support vector machine	21	88
[34]	2018	K-nearest neighbor	21	93
[27]	2018	Repeated incremental pruning to produce error reduction (RIPPER) rule induction algorithm	21	98
[4]	2019	Features selection using Pearson correlation coefficient (PCC) + semisupervised machine learning with k-mean	8	95.6
[34]	2018	Deep convolutional neural network (DCNN)	21	99
This work	2020	Features selection using information gain (IG) + decision tree for classification	**3**	**100**

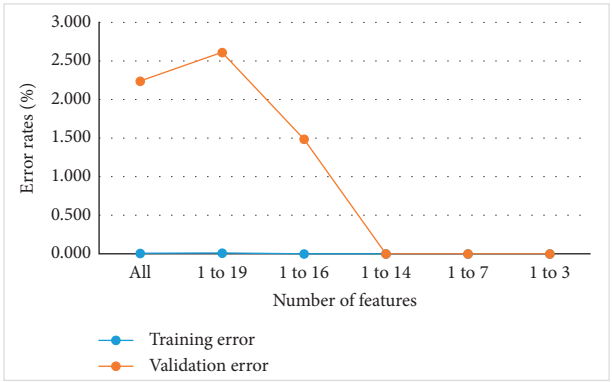


Figure 9: Training and validation error rates with different sets of features.

Table 9: Average time of training and testing the DT model using all features and using our three selected features.

Dataset features	Average time taken to build the model (seconds)	Average time taken to test the model (seconds)
Using all features	0.22	0.03
Using our three selected features	0.01	0.001

5. Conclusion and Future Work

In this paper, an effective and efficient approach using the information gain (IG) feature selection method and the decision tree (DT) classifier is proposed to detect BHP flooding attacks on OBS networks. The approach starts with selecting the most important features of OBS network traffic to improve the accuracy and efficiency of attack detection in OBS switches that have limited resources. A set of experiments is conducted on an OBS network dataset using 10-fold cross-validation and holdout techniques to evaluate and validate the approach. The experimental results demonstrate that the proposed approach can classify the class labels of OBS nodes with 100% accuracy by using only three features. The comparison with recent related works reveals that the proposed approach is suitable for OBS network security in terms of effectiveness and efficiency.

One of the limitations of the proposed approach is the lack of evaluation on more OBS datasets that can be varied in size and types of attacks due to unavailable OBS datasets other than the dataset used in the experiments of this study. Moreover, because the proposed approach is based on the decision tree method for classification, the training time is relatively expensive in case of large training datasets. However, by reducing the number of features of the proposed approach and the emergence of high-speed processors, this limitation is no longer a major problem. In future work, a large set of OBS network data will be collected for further evaluation of the proposed approach and will be made available for researchers in the field. This is due to lack of public OBS network datasets other than the dataset used in this research work.

Data Availability

The OBS-network dataset used in this study is publicly available at the UCI Machine Learning Repository [1].

Conflicts of Interest

The author declares that there are no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Acknowledgments

This study was supported by the Deanship of Scientific Research at Prince Sattam Bin Abdulaziz University.

References

[1] A. Rajab, C.-T. Huang, M. Al-Shargabi, and J. Cobb, "Countering burst header packet flooding attack in optical burst switching network," *Information Security Practice and Experience*, Springer, in *Proceedings of the International Conference on Information Security Practice and Experience*, pp. 315–329, Springer, Melbourne, Australia, December 2016.

[2] T. Venkatesh and C. Murthy, *An Analytical Approach to Optical Burst Switched Networks*, Springer, Boston, MA, 2010.

[3] N. Sreenath, K. Muthuraj, and G. V. Kuzhandaivelu, "Threats and vulnerabilities on TCP/OBS networks," in *Proceedings of the 2012 International Conference on Computer Communication and Informatics*, pp. 1–5, IEEE, Coimbatore, India, January 2012.

[4] M. K. H. Patwary and M. Haque, "A semi-supervised machine learning approach using K-means algorithm to prevent burst header packet flooding attack in optical burst switching network," *Baghdad Science Journal*, vol. 16, no. 3 Supplement, pp. 804–815, 2019.

[5] A. D. A. Rajab, "A machine learning approach for enhancing security and quality of service of optical burst switching networks," University of South Carolina, Columbia, South Carolina, Dissertation, 2017.

[6] S. S. Chawathe, "Analysis of burst header packets in optical burst switching networks," in *Proceedings of the 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pp. 1–5, IEEE, Cambridge, MA, USA, November 2018.

[7] A. Alazab, M. Hobbs, J. Abawajy, and M. Alazab, "Using feature selection for intrusion detection system," in *Proceedings of the 2012 International Symposium On Communications and Information Technologies (ISCIT)*, pp. 296–301, IEEE, Gold Coast, QLD, Australia, October 2012.

[8] J. Li, K. Cheng, S. Wang et al., "Feature selection: a data perspective," *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–45, 2017, https://arxiv.org/abs/1601.07996.

[9] N. Dessì and B. Pes, "Similarity of feature selection methods: an empirical study across data intensive classification tasks," *Expert Systems with Applications*, vol. 42, no. 10, pp. 4632–4642, 2015.

[10] E. J. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of Machine Learning and Data Mining*, Springer, Boston, MA, USA, 2017.

[11] S. S. Kannan and N. Ramaraj, "A novel hybrid feature selection via Symmetrical Uncertainty ranking based local memetic search algorithm," *Knowledge-Based Systems*, vol. 23, no. 6, pp. 580–585, 2010.

[12] G. Chandrashekar, F. Sahin, and E. Engineering, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[13] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," *Knowledge and Information Systems*, vol. 34, no. 3, pp. 483–519, 2013.

[14] M. M. Kabir, M. M. Islam, and K. Murase, "A new wrapper feature selection approach using neural network," *Neurocomputing*, vol. 73, no. 16–18, pp. 3273–3283, 2010.

[15] A. Mirzaei, Y. Mohsenzadeh, and H. Sheikhzadeh, "Variational relevant sample-feature machine: a fully Bayesian approach for embedded feature selection," *Neurocomputing*, vol. 241, pp. 181–190, 2017.

[16] M. M. Hassan, A. Gumaei, A. Alsanad, M. Alrubaian, and G. Fortino, "A hybrid deep learning model for efficient intrusion detection in big data environment," *Information Sciences*, vol. 513, pp. 386–396, 2020.

[17] F. A. Khan and A. Gumaei, "A comparative study of machine learning classifiers for network intrusion detection," in *Proceedings of the International Conference on Artificial Intelligence and Security*, pp. 75–86, Springer, New York, NY, USA, July 2019.

[18] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "TSDL: a two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30373–30385, 2019.

[19] M. Alqahtani, A. Gumaei, H. Mathkour, and M. B. Ismail, "A genetic-based extreme gradient boosting model for detecting intrusions in wireless sensor networks," *Sensors*, vol. 19, no. 20, p. 4383, 2019.

[20] A. Derhab, M. Guerroumi, A. Gumaei et al., "Blockchain and random subspace learning-based IDS for SDN-enabled industrial IoT security," *Sensors*, vol. 19, no. 14, p. 3119, 2019.

[21] A. Rajab, C.-T. Huang, M. Al-Shargabi, and Networking, "Decision tree rule learning approach to counter burst header packet flooding attack in optical burst switching network," *Optical Switching and Networking*, vol. 29, pp. 15–26, 2018.

[22] Q. Liao, H. Li, S. Kang, C. Liu, and C. Networks, "Application layer DDoS attack detection using cluster with label based on sparse vector decomposition and rhythm matching," *Security and Communication Networks*, vol. 8, no. 17, pp. 3111–3120, 2015.

[23] P. Xiao, W. Qu, H. Qi, and Z. Li, "Detecting DDoS attacks against data center with correlation analysis," *Computer Communications*, vol. 67, pp. 66–74, 2015.

[24] R. Karimazad and A. Faraahi, "An anomaly-based method for DDoS attacks detection using RBF neural networks," in *Proceedings of the International Conference on Network and Electronics Engineering*, vol. 11, pp. 44–48, Noida, India, December 2011.

[25] R. Zhong and G. Yue, "DDoS detection system based on data mining," in *Proceedings of the 2nd International Symposium on Networking and Network Security*, pp. 2–4, Jinggangshan, China, April 2010.

[26] Y.-C. Wu, H.-R. Tseng, W. Yang, and R.-H. Jan, "DDoS detection and traceback with decision tree and grey relational analysis," in *Proceedings of the 2009 Third International Conference on Multimedia and Ubiquitous Engineering*, pp. 306–314, IEEE, Qingdao, China, June 2009.

[27] R. Alshboul, "Flood attacks control in optical burst networks by inducing rules using data mining," *IJCSNS International Journal of Computer Science and Network Security*, vol. 18, no. 2, pp. 160–167, 2018.

[28] J.-H. Chen, M. Zhong, F.-J. Chen, and A.-D. Zhang, "DDoS defense system with turing test and neural network," in *Proceedings of the 2012 IEEE International Conference on Granular Computing*, pp. 38–43, IEEE, Hangzhou, China, August 2012.

[29] J. Li, Y. Liu, and L. Gu, "DDoS attack detection based on neural network," in *Proceedings of the 2010 2nd International Symposium on Aware Computing*, pp. 196–199, IEEE, Tainan, China, November 2010.

[30] V. Akilandeswari and S. M. Shalinie, "Probabilistic neural network based attack traffic classification," in *Proceedings of the 2012 Fourth International Conference on Advanced Computing (ICoAC)*, pp. 1–8, IEEE, Chennai, India, December 2012.

[31] H. Li and D. Liu, "Research on intelligent intrusion prevention system based on snort," in *Proceedings of the 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, vol. 1, pp. 251–253, IEEE, Changchun, China, August 2010.

[32] L. M. Ibrahim, "Anomaly network intrusion detection system based on distributed time-delay neural network (DTDNN)," *Journal of Engineering Science and Technology*, vol. 5, no. 4, pp. 457–471, 2010.

[33] N. Gao, D.-G. Feng, and J. Xiang, "A data-mining based DoS detection technique," *Chinese Journal of Computers*, vol. 29, no. 6, pp. 944–951, 2006.

[34] M. Z. Hasan, K. M. Z. Hasan, and A. Sattar, "Burst header packet flood detection in optical burst switching network using deep learning model," *Procedia Computer Science*, vol. 143, pp. 970–977, 2018.

[35] C. Largeron, C. Moulin, and M. Géry, "Entropy based feature selection for text categorization," in *Proceedings of the 2011 ACM Symposium On Applied Computing*, pp. 924–928, ACM, Taichung, Taiwan, March 2011.

[36] J. Friedman, "A recursive partitioning decision rule for nonparametric classification," *IEEE Transactions on Computers*, vol. C-26, no. 4, pp. 404–408, 1977.

[37] J. R. Quinlan, *C4. 5: Programming For Machine Learning*, Morgan Kaufmann, vol. 38, p. 48, San Mateo, CA, USA, 1993.

[38] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[39] A. Marzano, D. Alexander, O. Fonseca et al., "The evolution of bashlite and mirai IoT botnets," in *Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00813–00818, IEEE, Natal, Brazil, June 2018.

[40] J. Bekker and J. Davis, "Estimating the class prior in positive and unlabeled data through decision tree induction," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, Febraury 2018.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 8826038, 15 pages
https://doi.org/10.1155/2020/8826038

Research Article

BLATTA: Early Exploit Detection on Network Traffic with Recurrent Neural Networks

Baskoro A. Pratomo,[1,2] **Pete Burnap**,[1] **and George Theodorakopoulos**[1]

[1]*School of Computer Science and Informatics, Cardiff University, Cardiff, UK*
[2]*Informatics Department, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia*

Correspondence should be addressed to Baskoro A. Pratomo; me@baskoroadi.web.id

Received 8 April 2020; Revised 25 June 2020; Accepted 9 July 2020; Published 4 August 2020

Academic Editor: Muhammad Faisal Amjad

Detecting exploits is crucial since the effect of undetected ones can be devastating. Identifying their presence on the network allows us to respond and block their malicious payload before they cause damage to the system. Inspecting the payload of network traffic may offer better performance in detecting exploits as they tend to hide their presence and behave similarly to legitimate traffic. Previous works on deep packet inspection for detecting malicious traffic regularly read the full length of application layer messages. As the length varies, longer messages will take more time to analyse, during which time the attack creates a disruptive impact on the system. Hence, we propose a novel early exploit detection mechanism that scans network traffic, reading only 35.21% of application layer messages to predict malicious traffic while retaining a 97.57% detection rate and a 1.93% false positive rate. Our recurrent neural network- (RNN-) based model is the first work to our knowledge that provides early prediction of malicious application layer messages, thus detecting a potential attack earlier than other state-of-the-art approaches and enabling a form of early warning system.

1. Introduction

Exploits are attacks on systems that take advantage of the existence of bugs and vulnerabilities. They infiltrate the system by giving the system an input which triggers malicious behaviour. As time passes, the number of bugs and vulnerabilities increases, along with the number of exploits. In the first quarter of 2019, there were 400,000 new exploits [1], while more than 16 million exploits have been released in total. Exploits exist in most operating systems (OSs); hence, detecting exploits early is crucial to minimise potential damage.

By exploiting a vulnerability, attackers can, for example, gain access to remote systems, send a remote exploit, or escalate their privilege on a system. Exploits-DB [2] is a website that archives exploits, both remote and local ones. The number of existing remote exploits on the website is almost double that of the local ones, which suggests remote exploits are more prevalent. No physical access to the system is required to execute a remote exploit; thus, the attack can be launched from anywhere in the world.

Remote exploits normally carry a piece of code as a payload, which will be executed once a vulnerability has been successfully exploited. An exploit is analogous to a tool for breaking into a house, and its payload is something the burglar would do once they are inside the house. Without this payload, exploits would be merely a tool to demonstrate that an application is vulnerable. Exploit payloads may take many forms; they could be written in machine code, a server-side scripting language (e.g., PHP, Python, and Ruby), or OS specific commands (e.g., Bash).

One way to detect exploits is to scan network traffic for their presence. In doing this, the exploit can be detected before it arrives at the vulnerable system. If this is achieved, earlier action can be taken to minimise or nullify the damage. There is also no need to run the exploit in a clone server or virtual machine (VM) to be analysed—as it is usually the case in host-based detection approaches, making this approach more time efficient to block and provide rapid response to attacks. Therefore, detecting exploits in network traffic is a promising way to prevent remote exploits from infecting protected systems.

Detecting exploits on the wire has challenges: firstly, processing the vast amount of data without decreasing network throughput below acceptable levels; quality of service is still a priority. Secondly, there are various ways to encode exploit payloads [3], by modifying the exploit payload to make it appear different, yet still achieve the same goal. This technique makes it easy to evade any rule-based detection. Lastly, encrypted traffic is also a challenge; attackers may transmit the exploit with an encrypted protocol, e.g., HTTPS.

There are many ways to detect exploits in network traffic. Rule-based detection systems work by matching signatures of known attacks to the network traffic. Anything that matches the rule is deemed malicious. The most prevalent open-source intrusion detection system, Snort [4], has a rule that marks any traffic which contains byte 0×90 as shellcode-related traffic. This rule is based on the knowledge that most x86-based shellcodes are preceded by a sequence of *no operation* (NOP) instructions in which the bytes normally contain this value. However, this rule can easily be evaded by employing other NOP instructions, such as the "0×41 0×49" sequence. Apart from that, rule-based detection systems are susceptible to zero-day attacks for which no detection rule exists. Such systems are unable to detect these attacks until the rule database is updated with the new attack signature.

Machine learning (ML) algorithms are capable of classifying objects and artefacts based on features exhibited in data and handle various modalities of input. ML has been successfully applied in many domains with a high success rate, such as image classification, natural language processing, speech recognition, and even intrusion detection system. There has been much research on implementing machine learning to address network intrusion detection [5]. Researchers typically provide training examples of malicious and legitimate traffic to the ML algorithm that can then be used to determine whether new (unseen) traffic is malicious. However, there are three key limitations with existing research: firstly, most of the research uses old datasets, e.g., either KDD99 or DARPA99 [6]. The traffic in those datasets may not represent recent network traces since network protocols have evolved during these years, as have the attacks. Secondly, many of the previous works focus solely on the header information of network packets and process packets individually [6]. Yet, it is known that exploits may exhibit similar statistical attributes to legitimate traffic at a header-level and use evasion techniques such as packet fragmentation to hide their existence [3]. Therefore, we argue that network payload features may capture exploits better, and this area is still actively expanding as shown by the number of research mentioned in Table 1. This argument brings us to the third limitation: existing methods that use payload features, i.e., byte frequencies or n-grams, usually involve reading the payload of whole application layer messages (see Section 2). The issue is that these messages can be lengthy and spread over multiple network packets. Reading the whole messages before making a decision may lead to a delay in detecting the attack and gives the exploit time to execute before an alert is raised.

We, therefore, propose Blatta, an early exploit detection system which reads application layer messages and predicts whether these messages are likely to be malicious by reading only the first few bytes. This is the first work to our knowledge that provides early prediction of malicious application layer messages, thus detecting a potential attack earlier than other state-of-the-art approaches and enabling a form of early warning system. Blatta utilises a recurrent neural network- (RNN-) based model and n-grams to make the prediction. An RNN is a type of artificial neural networks that takes a sequence-based vector as input—in this case, a sequence of n-grams from an application layer message—and considers the temporal behaviour of the sequence so that they are not treated independently. There has been a limited amount of research on payload-based intrusion detection which used an RNN or its further development (i.e., long short-term memory [24, 25] and gated recurrent unit [20]), but earlier research did not make predictive decisions early as they only used 1-grams of full-length application layer messages as features, which lacks contextual information—a key benefit of higher-order n-grams. Other work that does consider higher-order sequences of n-grams (e.g. [12, 29]) is also yet to develop methods that provide early-stage prediction, preferring methods that require the full payload as input to a classifier.

To evaluate the proposed system, we generated an exploit traffic dataset by running exploits in Metasploit [30] with various exploit payloads and encoders. An exploit payload is a piece of code that will be executed once the exploit has successfully infiltrated the system. An encoder is used to change how the exploit payload appears while keeping its functionality. Our dataset contains traffic from 5,687 working exploits. Apart from that, we also used a more recent intrusion detection dataset, UNSW-NB15 [31], thus enabling our method to be compared with previous works.

To summarise, the key contributions of this paper are as follows:

(i) Proposed an early prediction of exploit traffic, Blatta, using a novel approach of using an RNN and high-order n-grams to make predictive decisions while still considering temporal behaviour of a sequence of n-grams. Blatta is the first, to the best of our knowledge, who introduces early exploit detection. Blatta detects exploit payloads as they enter the protected network and is able to predict the exploit by reading 35.21% of application layer messages on average. Blatta thus enables actions to be taken to block the attack earlier and therefore reduce the harm to the system.

(ii) Generated a dataset of exploit traffic with various exploit payloads and encoders, resulting in 5,687 unique connections. The exploit traffic in the dataset was ensured to contain actual exploit code, making the dataset closer to reality.

The rest of this paper is structured as follows: Section 2 gives a summary of previous works in this area. Section 3

Table 1: Related works in exploit detection. Unlike previous works, Blatta does not have to read until the end of application layer messages to detect exploit traffic.

Paper	Features	Detection method	Dataset(s)	Learning type	Protocol(s)	Early prediction
PAYL [7]	Relative frequency count of each 1-gram	Based on statistical model and Mahalanobis distance	D, SG	U	HTTP, SMTP, SSH	No
RePIDS [8]	Mahalanobis distance map which is originated from relative frequency count of each 1-gram, filtered by PCA.	Based on statistical model and Mahalanobis distance	D, M	U	HTTP	No
McPAD [9]	2v-grams	Multi one-class SVM classifier	D, M	U	HTTP	No
HMMPayl [10]	Byte sequences of the L7 payload.	Ensemble of HMMs	D, M, DI	U	HTTP	No
Oza et al. [11]	Relative frequency count of each 1-gram.	Based on statistical model	D, M, SG	U	HTTP	No
OCPAD [12]	High-order n-grams ($n > 1$).	Based on the occurrence probability of an n-grams in a packet	M, SG	U	HTTP	No
Bartos et al. [13]	Information from HTTP request headers and the lengths	SVM	SG	S	HTTP	No
Zhang et al. [14]	Packet header information and HTTP and DNS messages	Naïve Bayes, Bayesian network, SVM	D, SG	S	DNS, HTTP	No
Decanter [15]	HTTP messages	Clustering	SG	U	HTTP	No
Golait and Hubbali [16]	Byte sequence of the L7 payload	Probabilistic counting deterministic timed automata	SG	U	SIP	No
Duessel et al. [17]	Contextual n-grams of the L7 payload	One-class SVM	SG	U	HTTP, RPC	No
Min et al. [18]	Words of the L7 payload	CNN and random forest	I	S	HTTP	No
Jin et al. [19]	2v-grams	Multi one-class SVM classifier	M	U	HTTP	No
Hao et al. [20]	Byte sequence of the L7 payload	Variant gated recurrent unit	I	S	HTTP	No
Schneider and Bottinger [21]	Byte sequence of the L7 payload	Stacked autoencoder	O	U	Modbus	No
Hamed et al. [22]	n-grams of base64-encoded payload	SVM	I	S	All protocols in the datasets	No
Pratomo et al. [23]	Byte frequency of application layer messages	Outlier detection with deep autoencoder	SW	U	HTTP, SMTP	No
Qin et al. [24]	Byte sequence of the L7 payload	Using a recurrent neural network	O	S	HTTP	No
Liu et al. [25]	Byte sequence of the L7 payload	Using a recurrent neural network with embedded vectors	D, O	S	HTTP	No
Zhao and Ahn [26]	Disassembled instructions of bytes in network traffic	Employing Markov chain-based model and SVM	SG	S	Not mentioned	No
Shabtai et al. [27]	n-grams of a file and n-grams of opcodes in a file, then calculated TF/IDF of those n-grams	Various ML algorithm, e.g., random forest, decision tree, Naïve Bayes, and few others	SG	S	File classification	No
SigFree [28]	Disassembled instructions of bytes in application layer payload	Analyses of instruction sequences to determine if they are code	SG	Non-ML	HTTP	No
Proposed approach	High-order n-grams of application layer messages	Uses of recurrent neural network to early predict exploit traffic	SW, SG	S	HTTP, FTP	Yes

D = DARPA99; M = McPAD attacks dataset [9]; I = ISCX 2012; SG = self-generated; DI = DIEE; SW = UNSW-NB15; O = others; U = unsupervised; S = supervised; non-ML = non-machine learning approach.

explains the datasets we used to test our proposed method. How Blatta works is explained in Section 4. Then, Section 5 explains our extensive experimentation with Blatta. We also discuss possible evasion techniques to our approach in Section 6. Finally, the paper concludes in Section 7.

2. Related Works

The earliest solution to detecting exploit activities used pattern matching and regular expression [4]. Rules are defined by system administrators and are then applied to the

network traffic to identify a match. When a matching pattern is found, the system raises an alert and possibly shows which part of the traffic matches the rule. The disadvantage of this approach is that the rules must be kept up to date. Any variation to the exploit payload which is done by using encoders may defeat such detection.

ML approaches are capable of recognising previously seen instances of objects, and such methods aim to generalise to unseen objects. ML is able to learn patterns or behaviour from features present in training data and classify which class an unseen object belongs to. However, to work, suitable hand-crafted features must be engineered for the ML algorithm to make an accurate prediction. Some previous works defined their feature set by extracting information from application layer message. In [14] and [15], the authors generated their features from HTTP request URI and HTTP headers, i.e., host, constant header fields, size, user-agent, and language. The authors then clustered the legitimate traffic based on those features. Golait and Hubballi [16] tracked DNS and HTTP traffic and calculated pairwise features of two events to see their similarity. These features were obtained from the transport and application layer. One of their features is the semantical similarity between two HTTP requests.

Features derived from a specific protocol may capture specific behaviour of legitimate traffic. However, this feature extraction method has a drawback. A different set of features is needed for every application layer protocol that might be used in the network. Some research borrows the feature extraction method from natural language processing problems to have protocol-agnostic features, using n-grams.

One of the first leading research in payload analysis is PAYL [7]. It extracts 1-grams from all bytes of the payload as a representation of the network traffic and is trained over a set of those 1 grams. PAYL [7] measures the distance between the new incoming traffic with the model with a simplified Mahalanobis distance. Similar to PAYL, Oza et al. [11] extracts n-grams of HTTP traffic with various n values. They compared three different statistical models to detect anomalies/attacks. HMMPayl [10] is another work based on PAYL which uses Hidden Markov Models to detect anomalies. OCPAD [12] stores the n-grams of bytes in a probability tree and uses one-class Naïve Bayes classifier to detect malicious traffic. Hao et al. [20] proposed an autoencoder model to detect anomalous low-rate attacks on the network, and Schneider and Bottinger [21] used stacked autoencoders to detect attacks on industrial control systems. Hamed et al. [22] developed probabilistic counting deterministic timed automata which inspects byte values of application layer messages to identify attacks on VOIP applications. Pratomo et al. [23] extract words from the application layer message and detect web-based attacks with a combination of convolutional neural network (CNN) and random forest. A common approach that is found on all of the aforementioned works is they read all bytes in each packet or application layer message and do not decide until all bytes have been read, at which point it is possible the exploit has already infected the system and targeted the vulnerability.

Other research studies argue that exploit traffic is most likely to contain shellcode, a short sequence of machine code. Therefore, to get a better representation of exploit traffic, they performed static analysis on the network traffic. Qin et al. [24] disassemble byte sequences to opcode sequences and calculate probabilities of opcode transition to detect shellcode presence. Liu et al. [25] utilise n-grams that comprise machine instructions instead of bytes. SigFree [28] detects buffer overflow attempts on Internet services by constructing an instruction flow graph for each request and analysing the graph to determine if the request contains machine instructions. Any network traffic that contains valid machine instructions is deemed malicious. However, none of these consider that an exploit may also contain server-side scripting language or OS commands instead of shellcode.

In this paper, we proposed Blatta, an exploit attack detection system that (i) provides early detection of exploits in network traffic rather than waiting until the whole payload has been delivered, enabling proactive blocking of attacks, and (ii) is capable of detecting payloads that include malicious server-side scripting languages, machine instructions, and OS commands, enhancing the previous state-of-the-art approach that only focuses on shellcode. The summary of the proposed method and previous works is also shown in Table 1.

Blatta utilises a recurrent neural network- (RNN-) based model which takes a sequence of n-grams from network payload as the input. There has been limited research on developing RNNs to analyse payload data [20, 24, 25]. All of these works feed individual bytes (1 grams) from the payload as the features to their RNN model. However, 1 grams do not carry information about context of the payload string as a sequence of activities. Therefore, we model the payload as high-order n-grams where $n > 1$ as to capture more contextual information about the network payload. We directly compare 1-grams to higher-order n-grams in our experiments. Moreover, while related works such as [19], [17], and OCPAD [12] previously used high-order n-grams, they did not utilise a model capable of learning sequences of activities and thus not capable of making early-stage predictions within a sequence. Our novel RNN-based model will consider the long-term dependency of the sequence of n-grams.

An RNN-based model normally takes a sequence as input, processes each item sequentially, and outputs the decision after it has finished processing the last item of the sequence. The earlier works which used the RNN has this behaviour [24, 25]. While for Blatta to be able to early predict the exploit traffic, it takes the intermediate output of the RNN-based model, not waiting for full-length message to be processed. Our experiments show that this approach has little effect to accuracy and enables us to make earlier network attack predictions while retaining high accuracy and a low false positive rate.

3. Datasets and Threat Model

Several datasets have been used to evaluate network-based intrusion detection systems. DARPA released the KDD99 Cup, IDSEVAL 98, and IDSEVAL 99 datasets [32]. They

have been widely used over time despite some criticism that it does not model the attacks correctly [33]. The Lawrence Berkeley National Laboratory released their anonymised captured traffic [34] (LBNL05). More recently, Shiravi et al. released the ISCX12 dataset [35] in which they collected seven-day worth of traffic and provided the label for each connection in XML format. Moustafa and Slay published the UNSW-NB15 dataset [31] which had been generated by using the IXIA PerfectStorm tool for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours. This dataset provides PCAP files along with the preprocessed traffic obtained by Bro-IDS [36] and Argus [37].

Moustafa and Slay [31] captured the network traffic in two days, on 22 January and 17 February 2015. For brevity, we refer to those two parts of UNSW-NB15 as UNSW-JAN and UNSW-FEB, respectively. Both days contain malicious and benign traffic. Therefore, there has to be an effort to separate them if we would like to use the raw information from the PCAP files, not the preprocessed information written in the CSV files. The advantage of this dataset over ISCX12 is that UNSW-NB15 contains information on the type of attacks and thus we are able to select which kind of attacks are needed, i.e., exploits and worms. However, after analysing this dataset in depth, we observed that the exploit traffic in the dataset is often barely distinguishable from the normal traffic as some of them do not contain any exploit payload. Our explanation for this is that exploit attempts may not have been successful, thus they did not get to the point where the actual exploit payload was transmitted and recorded in PCAP files. Therefore, we opted to generate our exploit traffic dataset, the BlattaSploit dataset.

3.1. BlattaSploit Dataset. To develop an exploit traffic dataset, we set up two virtual machines that acted as vulnerable servers to be attacked. The first server was installed with Metasploitable 2 [38], a vulnerable OS designed to be infiltrated by Metasploit [30]. The second one was installed with Debian 5, vulnerable services, and WordPress 4.1.18. Both servers were set up in the victim subnet while the attacker machine was placed in a different subnet. These subnets were connected with a router. The router was used to capture the traffic as depicted in Figure 1. Although the network topology is less complex than what we would have in the real-world, this setup still generates representative data as the payloads are intact regardless of the number of hops the packets go through.

To launch exploits on the vulnerable servers, we utilised Metasploit, an exploitation tool which is normally used for penetration testing [30]. Metasploit has a collection of working exploits and is updated regularly with new exploits. There are around 9,000 exploits for Linux- and Unix-based applications, and to make the dataset more diverse, we also employed different exploit payloads and encoders. An exploit payload is a piece of code which is intended to be executed on the remote machine, and an encoder is a technique to modify the appearance of particular exploit code to avoid signature-based detection. Each exploit in Metasploit has its own set of compatible exploit payloads and encoders. In this paper, we used all possible combinations of exploit payloads and encoders.

We then ran the exploits against the vulnerable servers and captured the traffic using tcpdump. Traffic generated by each exploit was stored in an individual PCAP file. By doing so, we know which specific type of exploit the packets belonged to. Timestamps are normally used to mark which packets belong to a class, but this information would not be reliable since packets may not come in order, and if there is more than one source sending traffic at a time, their packet may get mixed up with other sources.

When analysing the PCAP files, we found out that not all exploits had run successfully. Some of them failed to send anything to the targeted servers, and some others did not send any malicious traffic, e.g., sending login request only and sending requests for nonexistent files. This supported our earlier thoughts on why the UNSW-NB15 dataset was lacking exploit payload traffic. Therefore, we removed capture files that had no traffic or too little information to be distinguished from normal traffic. In the end, we produced 5,687 PCAP files which also represent the number of distinct sets of exploit, exploit payloads, and encoders. Since we are interested in application layer messages, all PCAP files were preprocessed with tcpflow [39] to obtain the application layer message for each TCP connection. The summary of this dataset is shown in Table 2.

The next step for this exploit traffic dataset was to annotate the traffic. All samples can be considered malicious; however, we decided to make the dataset more detailed by adding the type of exploit payload contained inside the traffic and the location of the exploit payload. There are eight types of exploit payload in this dataset. They are written in JavaScript, PHP, Perl, Python, Ruby, Shell script (e.g., Bash and ZSH), SQL, and byte code/opcode for shellcode-based exploits.

There are some cases where an exploit contains an exploit payload "wrapped" in another scripting language. For example, a Python script to do reverse shell connection which uses the Bash *echo* command at the beginning. For these cases, the type of the exploit payload is the one with the longest byte sequence. In this case, the type of the particular connection is Python.

It is also important to note that whilst the vulnerable servers in our setup use a 7-year-old operating system, the payload carried by the exploit was the identical payload to a more recent exploit would have used. For example, both CVE-2019-9670 (disclosed in 2019) and CVE-2012-1495 (disclosed in 2012) can use generic/shell_bind_tcp as a payload. The traffic generated by both exploits will still be similar. Therefore, we argue that our dataset still represent the current attacks. Moreover, only successful exploit attacks are kept in the dataset, making the recorded traffic more realistic.

3.2. Threat Model. A threat model is a list of potential things that may affect protected systems. Having one means we can identify which part is the focus of our proposed approach;

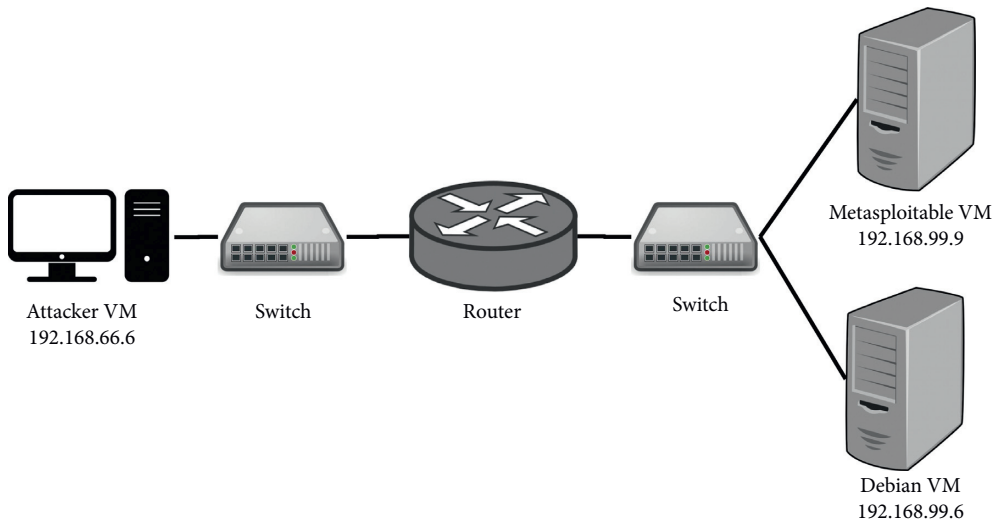


Figure 1: Network topology for generating exploit traffic. Attacker VM running Metasploit and target VMs are placed in different network connected by a router. This router is used to capture all traffic from these virtual machines.

Table 2: A summary of exploits captured in the BlattaSploit dataset.

Number of TCP connections	5687
Protocols	HTTP (3857), FTP (6), SMTP (74), POP3 (93)
Payload types	JavaScript, shellcode, Perl, PHP, Python, Ruby, Bash, SQL

Note. The numbers next to the protocols are the number of connections in the application layer protocols.

thus, in this case, we can potentially understand better what to look for in order to detect the malicious traffic and what the limitations are.

The proposed method focuses on detecting remote exploits by reading application layer messages from the unencrypted network traffic, although the detection method of Blatta can be incorporated with application layer firewalls, i.e., web application firewalls. Therefore, we can still detect the exploit attempt before it reaches the protected application. In general, the type of attacks we consider here are as follows:

(1) Remote exploits to servers that send malicious scripting languages (e.g., PHP, Ruby, Python, or SQL), shellcode, or Bash scripts to maintain control to the server or gained access to it remotely. For example, the apache_continuum_cmd_exec exploit with reverse shell payload will force the targeted server to open a connection to the attacking computer and provide shell access to the attacker. By focusing on the connections directed to servers, we can safely assume JavaScript code in the application layer message could also be malicious since normally JavaScript code is sent from the server to the client, not the other way around.

(2) Exploit attacks that utilise one of the text-based protocols over TCP, i.e., HTTP and FTP. Text-based protocols tend to be more well-structured; therefore, we can apply natural language processing based approach. The case would be similar to document classification.

(3) Other attacks that may utilise remote exploits are also considered, i.e., worms. Worms in the UNSW-NB15 dataset contain exploit code used to propagate themselves.

4. Methodology

Extracting features from application layer messages is the first step toward an early prediction method. We could use variable values in the message (e.g., HTTP header values, FTP commands, and SMTP header values), but it would require us to extract a different set of features from each application layer protocol. It is preferable to have a generic feature set which applies to various application layer protocols. Therefore, we proposed a method by using n-grams to model the application layer message as they carry more information than a sequence of bytes. For instance, it would be difficult for a model to determine what a sequence of bytes "G," "E," "T," space, "/," and so on means as those individual bytes may appear anywhere in a different order. However, if the model has 3-grams of the sequence (i.e., "GET," "ET," "T," and so on), the model would learn something more meaningful such as that the string could be an HTTP message. The advantage of using high-order n-grams where $n > 1$ is also shown by Anagram [29], method in [40], and OCPAD [12]. However, these works did not consider the temporal behaviour of a sequence of n-grams as their model was not capable of doing that. Therefore, Blatta utilised a recurrent neural network (RNN) to analyse the sequence of n-grams which were obtained from an application layer message.

An RNN takes a sequence of inputs and processes them sequentially in several time steps, enabling the model to learn the temporal behaviour of those inputs. In this case, the input to each time step is an n-gram, unlike earlier works which also utilised an RNN model but took a byte value as the input to each RNN time step [24, 25]. Moreover, these works took the output from the last time step to make decision, while our novel approach produces classification outputs at intermediate intervals as the RNN model is already confident about the decision. We argue that this approach will enable the proposed system to predict whether a connection is malicious without reading the full length of application layer messages, therefore providing an early warning method.

In general, as shown in Figure 2, the training and detection process of Blatta are as follows.

4.1. Training Stage. n-grams are extracted from application layer messages. l most common n-grams are stored in a dictionary. This dictionary is used to encode an n-gram to an integer. The encoded n-grams are then fed into an RNN model, training the model to classify whether the traffic is legitimate or malicious.

4.2. Detection Stage. For each new incoming TCP connection directed to the server, we reconstruct the application layer message, obtain a full-length or partial bytes of them, and determine if the sequence belongs to malicious traffic.

4.3. Data Preprocessing. In well-structured documents such as text-based application layer protocols, byte sequences can be a distinguishing feature that makes each message in their respective class differ from each other. Blatta takes the byte sequence of application layer messages from network traffic. The application layer messages need to be reconstructed as the message may be split into multiple TCP segments and transmitted in an arbitrary order. We utilise tcpflow [39] to read PCAP files, reconstruct TCP segments, and obtain the application layer messages.

We then represent the byte sequence as a collection of n-grams taken with various stride values. An n-gram is a consecutive sequence of n items from a given sample; in this case, an n-gram is a consecutive series of bytes obtained from the application layer message. Stride is how many steps the sliding window takes when collecting n-grams. Figure 3 shows examples of various n-grams obtained with a different value of n and stride.

We define the input to the classifier to be a set of integer encoded n-grams. Let $X = \{x_1, x_2, x_3, \ldots, x_k\}$ be the integer encoded n-grams collected from an application layer message as the input to the RNN model. We denote k as the number of n-grams taken from each application layer message. Each n-gram is categorical data. It means a value of 1 is not necessarily smaller than a value of 50. They are simply different. Encoding n-grams with one-hot encoding is not a viable solution as the resulting vector would be sparse and hard to model. Therefore, Blatta transforms the sequence of n-grams with embedding technique. Embedding is essentially a lookup table that maps an item to a dense vector with a fixed size *embedded*_dim. Using pretrained embedding vectors, e.g., GloVe [41], is common in natural language processing problems, but these pretrained embedding vectors were generated from a corpus of words. While our approach works with byte-level n-grams. Therefore, it is not possible to use the pretrained embedding vectors. Instead, we initialise the embedding vectors with random values which will be updated by backpropagation during the training so that n-grams which usually appear together will have vectors that are close to each other.

It is worth noting that the number of n-grams collected raises exponentially as the n increases. If we considered all possible n-gram values, the model would overfit. Therefore, we limit the number of embedding vectors by building a dictionary of most common n-grams in the training set. We define the dictionary size as l in which it contains l unique n-grams and a placeholder for other n-grams that do not exist in the dictionary. Thus, the embedding vectors have $l + 1$ entries. However, we would like the size of each embedded vector to be less than $l + 1$. Let ε be the size of an embedded vector (*embedded*_dim). If x_t represents an n-gram, the embedding layer transforms X to $\widehat{X}$. We denote $\widehat{X} = \{\widehat{x}_1, \ldots, \widehat{x}_k\}$ where each $\widehat{x}$ is a vector with the size of ε. The embedded vectors $\widehat{X}$ are then passed to the recurrent layer.

4.4. Training RNN-Based Classifier. Since the input to the classifier is sequential data, we opted to use a method that takes into account the sequential relationship of elements in the input vectors. Such methods capture the behaviour of a sequence better than processing those elements individually [42]. A recurrent neural network is an architecture of neural networks in which each layer takes time series data, processes them in several time steps, and utilises the output of the previous time step in the current step calculation. We refer to these layers as recurrent layers. Each recurrent layer consists of recurrent units.

The vanilla RNN has a vanishing gradient problem, a situation where the recurrent model cannot be further trained because the value to update the weight is too small; thus, there would be no point of training the model further. Therefore, long short-term memory (LSTM) [43] and gated recurrent unit (GRU) [44] are employed to avoid this situation. Both LSTM and GRU have cells/units that are connected recurrently to each other, replacing the usual recurrent units which existed in earlier RNNs. What makes their cells different is the existence of gates. These gates decide whether to pass certain information coming from the previous cell (i.e., input gate and forget gate in LSTM unit and update gate in GRU) or going to the next unit (i.e., output gate in LSTM). Since LSTM has more gates than GRU, it requires more calculations, thus computationally more expensive. Yet, it is not conclusive whether one is better than the other [44]; thus, we use both types and compare the results. For brevity, we will refer to both types as recurrent layers and their cells as recurrent units.

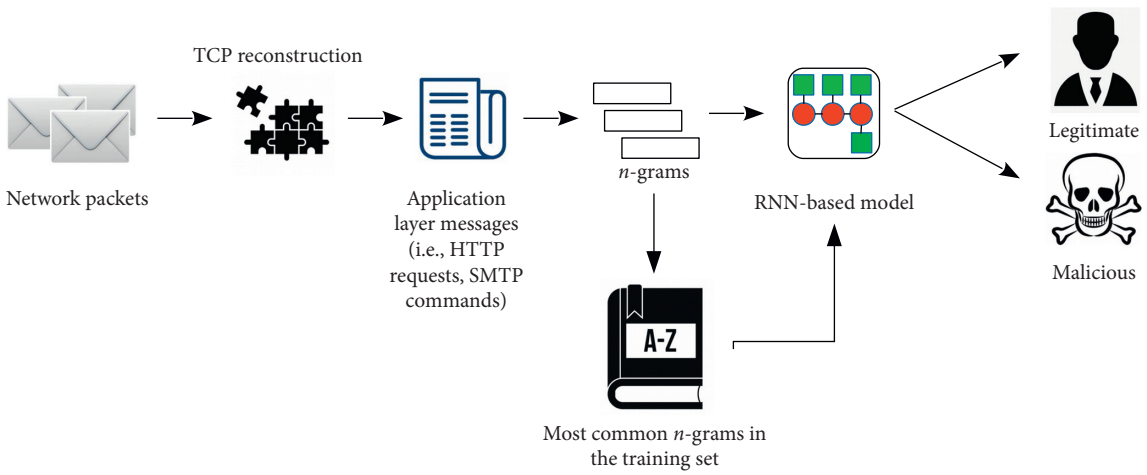


Figure 2: Architecture overview of the proposed method. Application layer messages are extracted from captured traffic using tcpflow [39]. n-grams are obtained from those messages. They will then be used to build a dictionary of most common n-grams and train the RNN-based model (i.e., LSTM and GRU). The trained model outputs a prediction whether the traffic is malicious or benign.

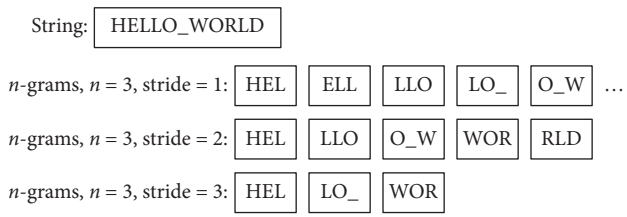


Figure 3: An example of n-grams of bytes taken with various stride values.

The recurrent layer takes a vector $\hat{x}_t$ for each time step t. In each time step, the recurrent unit outputs hidden state h_t with a dimension of $|h_t|$. The hidden state is then passed to the next recurrent unit. The last hidden state h_k becomes the output of the recurrent layer which will be passed onto the next layer.

Once we obtain h_k, the output of the recurrent layer, the next step is to map the vector to benign or malicious class. Mapping h_k to those classes requires us to use linear transformation and softmax activation unit.

A linear transformation transforms h_k into a new vector L using (1), where W is the trained weight and b is the trained bias. After that, we transform L to obtain $Y = \{y_i \mid 0 \le i < 2\}$, the log probabilities of the input file belonging to the classes with LogSoftmax, as described in (2). The output of LogSoftmax is the index of an element that has the largest probability in Y. All these forward steps are depicted in Figure 4:

$$L = W * h_k + b,$$
$$L = \{l_i \mid 0 \le i < 2\}, \tag{1}$$

$$Y = \arg\max_{0 \le i < 2} \left(\log \frac{\exp(l_i)}{\sum_{i=0}^{2} \exp(l_i)} \right). \tag{2}$$

In the training stage, after feeding a batch of training data to the model and obtaining the output, the next step is to evaluate the accuracy of our model. To measure our model's performance during the training stage, we need to calculate a loss value which represents how far our model's output is from the ground truth. Since this approach is a binary classification problem, we use *negative log likelihood* [45] as the loss function. Then, the losses are backpropagated to update weights, biases, and the embedding vectors.

4.5. Detecting Exploits. The process of detecting exploits is essentially similar to the training process. Application layer messages are extracted. n-grams are acquired from these messages and encoded using the dictionary that was built during the training process. The encoded n-grams are then fed into the RNN model that will output probabilities of these messages being malicious. When the probability of an application layer message being malicious is higher than 0.5, the message is deemed malicious and an alert is raised.

The main difference in this process to the training stage is the time when Blatta stops processing inputs and makes the decision. Blatta takes the intermediate output of the RNN model, hence requiring fewer inputs and disregarding the needs to wait for the full-length message to process. We will show in our experiment that the decision taken by using intermediate output and reading fewer bytes is close to reading the full-length message, giving the proposed approach an ability of early prediction.

5. Experiments and Results

In this section, we evaluate Blatta and present evidence of its effectiveness in predicting exploit in network traffic. Blatta is implemented with Python 3.5.2 and PyTorch 0.2.0 library. All experiments were run on a PC with Core i7 @ 3.40 GHz, 16 GB of RAM, NVIDIA GeForce GT 730, NVIDIA CUDA 9.0, and CUDNN 7.

The best practice for evaluating a machine learning approach is to have separate training and testing set. As the name implies, training set is used to train the model and the

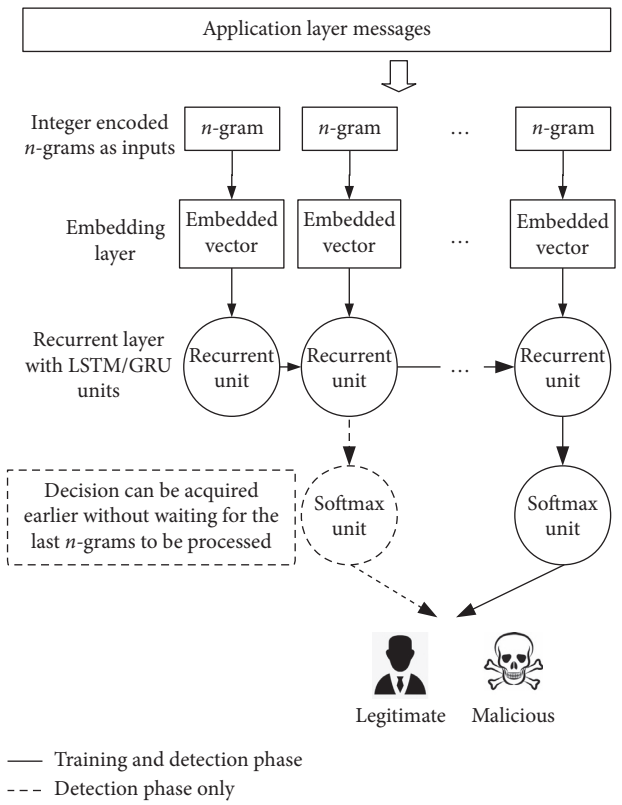


Figure 4: A detailed view of the classifier. *n*-grams are extracted from the input application layer message, which are then used to train an RNN model to classify whether the connection is malicious or benign.

testing set is for evaluating the model's performance. We split BlattaSploit dataset in a 60 : 40 ratio for training and testing set as malicious samples. The division was carefully taken to include diverse type of exploit payloads. As samples of benign traffic to train the model, we obtained the same number of HTTP and FTP connections as the malicious samples from UNSW-JAN. Having a balanced set of both classes is important in a supervised learning.

We measure our model's performance by using samples of malicious and benign traffic. Malicious samples are obtained from 40% of BlattaSploit dataset, exploit, and worm samples in UNSW-FEB set (10,855 samples). As for the benign samples, we took the same number (10,855) of benign HTTP and FTP connections in UNSW-FEB. We used the UNSW dataset to compare our proposed approach performance with previous works.

In summary, the details of training and testing sets used in the experiments are shown in Table 3.

We evaluated the classifier model by counting the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN). They are then used to calculate detection rate (DR) and false positive rate (FPR) which are metrics to measure our proposed system's performance.

DR measures the ability of the proposed system to correctly detect malicious traffic. The value of DR should be as close as possible to 100%. It would show how well our system is able to detect exploit traffic. The formula to calculate DR is shown in (3). We denote TP as the number of detected malicious connections and FN as the number of undetected malicious connections in the testing set:

$$\mathrm{DR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{3}$$

FPR is the percentage of benign samples that are classified as malicious. We would like to have this metric to be as low as possible. High FPR means many false alarms are raised, rendering the system to be less trustworthy and useless. We calculate this metric using (4). We denote FP as the number of false positives detected and N as the number of benign samples:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{N}. \tag{4}$$

5.1. Data Analysis. Before discussing about the results, it is preferable to analyse the data first to make sure that the results are valid and the conclusion taken is on point. Blatta aims to detect exploit traffic by reading the first few bytes of the application layer message. Therefore, it is important to know how many bytes are there in the application layer messages in our dataset. Hence, we can be sure that Blatta reads a number of bytes fewer than the full length of the application layer message.

Table 4 shows the average length of application layer messages in our testing set. The benign samples have an average message length of 593.25, lower than any other sets. Therefore, deciding after reading fewer bytes than at least that number implies our proposed method can predict malicious traffic earlier, thus providing improvement over previous works.

5.2. Exploring Parameters. Blatta includes parameters that must be selected in advance. Intuitively, these parameters affect the model performance, so we analysed the effect on model's performance and selected the best model to be compared later to previous works. The parameters we analysed are recurrent layer types, *n*, stride, dictionary size, the embedding vector dimension, and the recurrent layer type. When analysing each of these parameters, we use different values for the parameter and set the others to their default values (i.e., $n = 5$, stride = 1, dictionary size = 2000, *embe dd ing*_dim = 32, recurrent layer = LSTM, and number of hidden layers = 1). These default values were selected based on the preliminary experiment, which had given the best result. Apart from the modifiable parameters, we set the optimiser to stochastic gradient descent with learning rate 0.1 as using other optimisers did not necessarily increase or decrease the performance in our preliminary experiment. The number of epochs is fixed to five as the loss value did not decrease further, adding more training iteration did not give significant improvement.

As can be seen in Table 5, we experimented with variable lengths of input to see how much the difference when the number of bytes read is reduced. It is noted that some

Table 3: Numbers of benign and malicious samples used in the experiments.

Set	Obtained from	Num. of samples	Class
Training set	BlattaSploit	3406	Malicious
	UNSW-JAN	3406	Benign
Testing set	BlattaSploit	2276	Malicious
	UNSW-FEB	10855	Benign
	UNSW-FEB	10855	Malicious

Table 4: Average message length of application layer messages in the testing set.

Set	Average message length
BlattaSploit	2318.93
UNSW benign samples	593.25
UNSW exploit samples	1437.36
UNSW worm samples	848

messages are shorter than the limit. In this case, all bytes of the messages are indeed taken.

In general, reading the full length of application layer messages mostly gives more than 99% detection rate with 2.51% false positive rate. This performance stays still with a minor variation when the length of input messages is reduced down to 500 bytes. When the length is limited to 400, the false positive rate spikes up for some configurations. Our hypothesis is that this is due to benign samples have relatively short length. Therefore, we will pay more attention to the results of reading 500 bytes or fewer and analyse each parameter individually.

n is the number of bytes (n-grams) taken in each time step. As shown in Table 5, we experimented with 1, 3, 5, 7, and 9-gram. For brevity, we omitted $n = 2, 4, 6, 8$ because the result difference is not significant. As predicted earlier, 1-gram was least effective, and the detection rates were around 50%. As for the high-order n-grams, the detection rates are not much different but the false positive rates are. 5-gram and 7-gram provide better false positive rates (2.51%) even when Blatta reads the first 400 bytes. 7-gram gives lower false positive rate (8.92%) when reading first 300 bytes yet it is still too high for real-life situation. Having a higher n means more information is considered in a time step, this may lead to not only a better performance but also overfitting.

As the default value of n is five, we experimented with stride of one to five. Thus, it can be observed how the model would react depending on how much the n-grams overlapped. It is apparent that nonoverlapping n-grams provide lower false positives with around 1–3% decrease in detection rates. A value of two and three for the stride performs the worst, and they missed quite a few malicious traffic.

The dictionary size plays an important role in this experiment. Having too many n-grams leads to overfitting as the model would have to memorise too many of them that may barely occur. We found that a dictionary size of 2000 has the highest detection rate and lowest false positive rate. Reducing the size to 1000 has made the detection rate to drop for about 50%, even when the model read the full-length messages.

Without embedding layer, Blatta would have used a one-hot vector as big as the dictionary size for the input in a time step. Therefore, the embedding dimension has the same effect as dictionary size. Having it too big leads to overfitting and too little could mean too much information is lost due to the compression. Our experiments with a dimension of 16, 32, or 64 give similar detection rates and differ less than 2%. An embedding dimension of 64 can have the least false positive when reading 300 bytes.

Changing recurrent layer does not seem to have much difference. LSTM has a minor improvement over GRU. We argue that preferring one after the other would not make a big improvement other than training speed. Adding more hidden layers does not improve the performance. On the other hand, it has a negative impact on the detection speed, as shown in Table 6.

After analysing this set of experiments, we ran another experiment with a configuration based on the best performing parameters previously explained. The configuration is $n = 5$, stride = 5, dictionary size = 2000, embedding dimension = 64, and a LSTM layer. The model then has a detection rate of 97.57% with 1.93% false positives by reading only the first 400 bytes. This result shows that Blatta maintains high accuracy while only reading 35.21% the length of application layer messages in the dataset. This optimal set of parameters is then used in further analysis.

5.3. Detection Speed. In the IDS area, detection speed is another metric worth looked into, apart from accuracy-related metrics. Time is of the essence in detection, and the earlier we detect malicious traffic, the faster we could react. However, detection speed is affected by many factors, such as the hardware or other applications/services running at the same time as the experiment. Therefore, in this section, we analyse the difference of execution time between reading the full and partial payload.

We first calculated the execution time of each record in the testing set, then divided the number of bytes processed by the execution time to obtain the detection speed in kilobytes/seconds (KBps). Eventually, the detection speed of all records was averaged. The result is shown in Table 6.

As shown in Table 6, reducing the processed bytes to 700, about half the size of an IP packet, increased the detection speed by approximately two times (from an average of 8.366 kbps to 16.486 kbps). Evidently, the trend keeps rising as the number of bytes reduced. If we take the number of bytes limit from the previous section, which is 400 bytes, Blatta can provide about three times increment in detection speed or 22.17 kbps on average. We are aware that this number seems small compared to the transmission speed of a link in the network which can reach 1 Gbps/128 MBps. However, we argued that there are other factors which limit our proposed method from performing faster, such as the hardware used in the experiment and the programming language used to implement the approach. Given the approach runs in a better environment, the detection speed will increase as well.

Table 5: Experiment results of using various parameters combination and various lengths of input to the model

Parameter		No. of bytes							No. of bytes						
		All	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
n	**1**	47.22	48.69	49.86	50.65	51.77	54.99	65.32	1.18	1.19	1.21	1.21	71.31	78.7	89.43
	3	99.87	99.51	99.77	99.1	99.59	98.93	91.07	2.51	2.51	2.51	2.51	72.61	10.29	20.51
	5	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	7	99.86	99.47	99.59	99.37	99.19	98.53	97.08	2.51	2.51	2.51	2.51	2.51	8.92	80.92
	9	99.81	99.59	99.62	99.57	99.23	98.16	88.93	2.51	2.51	2.51	2.51	72.6	74.16	90.6
Stride	**1**	99.87	99.55	99.78	99.57	99.29	98.91	88.75	2.51	2.51	2.51	2.51	2.51	72.6	11.08
	2	73.39	74.11	74.01	74.45	74.69	74.62	77.82	1.81	1.81	1.81	1.81	71.92	72.46	19.86
	3	82.51	82.54	83.07	83.12	83.25	83.5	85.75	1.5	1.49	1.5	1.51	71.62	75.47	89.63
	4	99.6	99.19	99.26	99.28	98.61	98.55	98.37	1.93	1.93	1.93	1.93	1.93	74.09	10.5
	5	**99.73**	**98.95**	**98.88**	**98.65**	**98**	**95.77**	**88.29**	**1.93**	**1.92**	**1.93**	**1.93**	**1.93**	**54.16**	**90.02**
Dictionary size	**1000**	47.78	49.5	50.36	50.79	51.8	54.83	54.68	1.21	1.21	1.22	1.22	71.33	79.47	89.42
	2000	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	5000	99.87	99.37	99.75	99.79	99.62	99.69	99.66	2.51	2.51	2.51	2.51	72.61	10.03	90.61
	10000	99.86	99.44	99.74	99.55	99.44	98.55	98.33	2.51	2.51	2.51	2.51	72.61	79.06	90.15
	20000	99.84	99.81	99.69	99.24	99.21	99.43	98.91	2.51	2.51	2.51	2.51	72.61	80.46	89.64
Embedding dimension	**16**	99.89	99.65	99.7	99.67	99.22	99.09	98.81	2.51	2.51	2.51	2.51	2.51	76.77	80.94
	32	99.87	99.55	99.78	99.57	99.29	98.91	88.75	2.51	2.51	2.51	2.51	2.51	72.6	11.08
	64	**99.87**	**99.2**	**99.41**	**99.09**	**98.61**	**96.76**	**85.52**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**4.51**	**89.85**
	128	99.84	99.33	99.6	99.35	98.99	97.69	86.78	2.51	2.51	2.51	2.51	72.6	4.27	10.88
	256	99.88	99.76	99.8	99.22	99.38	98.64	90.34	2.51	2.51	2.51	2.51	72.6	80.79	90.6
Recurrent layer	**LSTM**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	GRU	99.88	99.35	99.48	99.35	99.06	97.94	86.22	2.51	2.51	2.51	2.51	2.51	78.95	8.48
No. of layers	**1**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	2	99.86	99.46	99.46	99.38	99.2	99.72	88.65	2.51	2.51	2.51	2.51	72.59	78.78	20.29
	3	99.84	99.38	99.68	99.1	99.18	98.16	87.35	2.51	2.51	2.51	2.51	2.51	74.94	10.83
		Detection rate							False positive rate						

Bold values show the parameter value for each set of experiment which gives the highest detection rate and lowest false positive rate.

Table 6: The effect of reducing the number of bytes to the detection speed.

No. of bytes	No. of LSTM layers		
	1	2	3
All	8.366 ± 0.238327	5.514 ± 0.004801	3.698 ± 0.011428
700	16.486 ± 0.022857	10.704 ± 0.022001	7.35 ± 0.044694
600	18.16 ± 0.020556	11.97 ± 0.024792	8.21 ± 0.049584
500	20.432 ± 0.02352	13.65 ± 0.036668	9.376 ± 0.061855
400	22.17 ± 0.032205	14.94 ± 0.037701	10.302 ± 0.065417
300	24.076 ± 0.022857	16.368 ± 0.036352	11.318 ± 0.083477
200	26.272 ± 0.030616	18.138 ± 0.020927	12.688 ± 0.063024

The values are average (mean) detection speed in kbps with 95% confidence interval, calculated from multiple experiments. The detection speed increased significantly (about three times faster than reading the whole message), allowing early prediction of malicious traffic.

5.4. Comparison with Previous Works. We compare Blatta results with other related previous works. PAYL [7], OCPAD [12], Decanter [15], and the autoencoder model [23] were chosen due to their code availability, and both can be tested against the UNSW-NB15 dataset. PAYL and OCPAD read an IP packet at a time, while Decanter and [23] reconstruct TCP segments and process the whole application layer message. None of them provides early detection, but to show that Blatta also offers improvements in detecting malicious traffic, we compare the detection and false positive rates of those works with Blatta.

We evaluated all methods with exploit and worm data in UNWS-NB15 as those match our threat model and the dataset is already publicly available. Thus, the result would be comparable. The results are shown in Table 7.

In general, Blatta has the highest detection rate—albeit it also comes with the cost of increasing false positives. Although the false positives might make this approach unacceptable in real life, Blatta is still a significant step towards a direction of early prediction, a problem that has not been explored by similar previous works. This early prediction approach enables system administrators to react faster, thus reducing the harm to protected systems.

In the previous experiments, as shown in Section 5.2, Blatta needs to read 400 bytes (on average 35.21% of application layer message size) to achieve 97.57% detection

Table 7: Comparison to previous works using the UNSW-NB15 dataset as the testing set.

Method	Detection rate (%)		FPR (%)
	Exploits in UNSW-NB15	Worms in UNSW-NB15	
Blatta	99.04	100	1.93
PAYL	87.12	26.49	0.05
OCPAD	10.53	4.11	0
Decanter	67.93	90.14	0.03
Autoencoder	47.51	81.12	0.99

rate. It reads fewer bytes than PAYL, OCPAD, Decanter, and [23] while keeping the accuracy high.

5.5. Visualisation. To investigate how Blatta has performed the detection, we took samples of both benign and malicious traffic and observed the input and output. We were particularly interested in the relation of n-grams that are not stored in the dictionary to the decision (unknown n-grams). Those n-grams either did not exist in the training set or were not common enough to be included in the dictionary.

On Figure 5, we visualise three samples of traffic taken from different sets, BlattaSploit and UNSW-NB15 datasets. The first part of each sample shows n-grams that did not exist in the dictionary. The yellow highlighted parts show those n-grams. The second part shows the output of the recurrent layer for each time step. The darker the red highlight, the closer the probability of the traffic being malicious to one in that time step.

As shown in Figure 5 (detection samples), malicious samples tend to have more unknown n-grams. It is evident that the existence of these unknown n-grams increases the probability of the traffic being malicious. As an example, the first five bytes of the five samples have around 0.5 probability of being malicious. And then the probability went up closer to one when an unknown n-gram is detected.

Similar behaviour also exists in the benign sample. The probability is quite low because there are many known n-grams. Despite the existence of unknown n-grams in the benign sample, the end result shows that the traffic is benign. Furthermore, most of the time, the probability of the traffic being malicious is also below 0.5.

6. Evasion Techniques and Adversarial Attacks

Our proposed approach is not a silver bullet to tackle exploit attacks. There are evasion techniques which could be employed by adversaries to evade the detection. These techniques open possibilities for future work. Therefore, this section talks about such evasion techniques and discuss why they have not been covered by our current method.

Since our proposed method works by analysing application layer messages, it is safe to disregard evasion techniques on transport or network layer level, e.g., IP fragmentation, TCP delayed sending, and TCP segment fragmentation. They should be handled by the underlying tool that reconstructs TCP session. Furthermore, those evasion techniques can also be avoided by Snort preprocessor.

Two possible evasion techniques are compression and/or encryption. Both compression and encryption change the bytes' value from the original and make the malicious code harder to detect by Blatta or any previous work [7, 12, 23] on payload-based detection. Metasploit has a collection of evasion techniques which include compression. The compression evasion technique only works on HTTP and utilises gzip. This technique only compresses HTTP responses, not HTTP requests. While all HTTP-based exploits in UNSW-NB15 and BlattaSploit have their exploit code in the request, thus no data are available to analyse the performance if the adversary uses compression. However, gzip compressed data could still be detected because they always start with the magic number 1f 8b and the decompression can be done in a streaming manner in which Blatta can do so. There is also no need to decompress the whole data since Blatta works well with partial input.

Encryption is possibly the biggest obstacle in payload-based NIDS: none of the previous works in our literature (see Table 1) have addressed this challenge. There are other studies which deal with payload analysis in encrypted traffic [46, 47]. However, these studies focus on classifying which application generates the network traffic instead of detecting exploit attacks; thus, they are not directly relevant to our research.

On its own, Blatta is not able to detect exploits hiding in encrypted traffic. However, Blatta's model can be exported and incorporated with application layer firewalls such as ShadowDaemon [48]. ShadowDaemon is commonly installed on a web server and intercepts HTTP requests before being processed by a web server software. It detects attacks based on its signature database. Since it is extensible and reads the same data as Blatta (i.e., application layer messages), it is possible to use Blatta's RNN-based model to extend the capability of ShadowDaemon beyond rule-based detection. More importantly, this approach would enable Blatta to deal with encrypted traffic, making it applicable in real-life situations.

Attackers could place the exploit code closer to the end of the application layer message. Hoping that in doing so, the attack would not be detected as Blatta reads merely the first few bytes. However, exploits with this kind of evasion technique would still be detected since this evasion technique needs a padding to place the exploit code at the end of the message. The padding itself will be detected as a sign of malicious attempts as it is most likely to be a byte sequence which rarely exist in the benign traffic.

Figure 5: Visualisation of unknown n-grams in the application layer messages and outputs of the recurrent layer for each time step. It shows how the proposed system observes and analyses the traffic. Yellow blocks show unknown n-grams. Red blocks show the probability of the traffic being malicious when reading an n-gram at that point.

7. Conclusion and Future Work

This paper presents Blatta, an early prediction system for exploit attacks which can detect exploit traffic by reading only 400 bytes from the application layer message. First, Blatta builds a dictionary containing most common n-grams in the training set. Then, it is trained over benign and malicious sequences of n-grams to classify exploit traffic. Lastly, it continuously reads a small chunk of an application layer message and predicts whether the message will be a malicious one.

Decreasing the number of bytes taken from application layer messages only has a minor effect on Blatta's detection rate. Therefore, it does not need to read the whole application layer message like previously related works to detect exploit traffic, creating a steep change in the ability of system administrators to detect attacks early and to block them before the exploit damages the system. Extensive evaluation of the new exploit traffic dataset has clearly demonstrated the effectiveness of Blatta.

For future study, we would like to train a model that can recognise malicious behaviour based on messages exchanged between clients and a server since in this paper we only consider messages from clients to a server, but not the other way around. Detecting attacks on encrypted traffic while retaining the early prediction capability could be a future research direction. It also remains a question whether the approach in mobile traffic classification [46, 47] would be applicable to the field of exploit detection.

Data Availability

The compressed file containing the dataset is available on https://bit.ly/blattasploit.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Indonesia Endowment Fund for Education (Lembaga Pengelola Dana Pendidikan/LPDP) under the grant number PRJ-968/LPDP.3/2016 of the LPDP.

References

[1] McAfee, "McAfee labs threat report—august 2019," Report, McAfee, Santa Clara, CA, USA, 2019.

[2] E. DB, "Offensive security's exploit database archive," EDB, Bedford, MA, USA, 2010.

[3] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: sneaking through your intrusion detection/prevention systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1011–1020, 2012.

[4] M. Roesch, "Snort: lightweight intrusion detection for networks," in *Proceedings of LISA'99: 13th Systems Administration Conference*, vol. 99, pp. 229–238, Seattle, Washington, USA, November 1999.

[5] R. Sommer and V. Paxson, "Outside the closed world: on using machine learning for network intrusion detection," in *Proceedings of the 2010 IEEE Symposium on Security And Privacy*, pp. 305–316, Berkeley/Oakland, CA, USA, May 2010.

[6] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: a review," *Computers & Security*, vol. 30, no. 6-7, pp. 353–375, 2011.

[7] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," *Lecture Notes in Computer Science, Recent Advances in Intrusion Detection*, in *Proceedings of the 7th International Symposium, Raid 2004*, pp. 203–222, Gothenburg, Sweden, September 2004.

[8] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu, "RePIDS: a multi tier real-time payload-based intrusion detection system," *Computer Networks*, vol. 57, no. 3, pp. 811–824, 2013.

[9] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "McPAD: a multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.

[10] D. Ariu, R. Tronci, and G. Giacinto, "HMMPayl: an intrusion detection system based on hidden markov models," *Computers & Security*, vol. 30, no. 4, pp. 221–241, 2011.

[11] A. Oza, K. Ross, R. M. Low, and M. Stamp, "HTTP attack detection using *n*-gram analysis," *Computers & Security*, vol. 45, pp. 242–254, 2014.

[12] M. Swarnkar and N. Hubballi, "OCPAD: one class naive bayes classifier for payload based anomaly detection," *Expert Systems with Applications*, vol. 64, pp. 330–339, 2016.

[13] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proceedings of the 25th {USENIX} Security Symposium*, pp. 807–822, Austin, TX, USA, August 2016.

[14] H. Zhang, D. Yao, N. Ramakrishnan, and Z. Zhang, "Causality reasoning about network events for detecting stealthy malware activities," *Computers & Security*, vol. 58, pp. 180–198, 2016.

[15] R. Bortolameotti, T. van Ede, M. Caselli et al., "DECANTeR: DEteCtion of anomalous outbound HTTP traffic by passive application fingerprinting," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 373–386, Orlando, FL, USA, December 2017.

[16] D. Golait and N. Hubballi, "Detecting anomalous behavior in voip systems: a discrete event system modeling," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 730–745, 2016.

[17] P. Duessel, C. Gehl, U. Flegel, S. Dietrich, and M. Meier, "Detecting zero-day attacks using context-aware anomaly detection at the application-layer," *International Journal of Information Security*, vol. 16, no. 5, pp. 475–490, 2017.

[18] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, "TR-IDS: anomaly-based intrusion detection through text-convolutional neural network and random forest," *Security and Communication Networks*, vol. 2018, Article ID 4943509, 9 pages, 2018.

[19] X. Jin, B. Cui, D. Li, Z. Cheng, and C. Yin, "An improved payload-based anomaly detector for web applications," *Journal of Network and Computer Applications*, vol. 106, pp. 111–116, 2018.

[20] Y. Hao, Y. Sheng, and J. Wang, "Variant gated recurrent units with encoders to preprocess packets for payload-aware intrusion detection," *IEEE Access*, vol. 7, pp. 49985–49998, 2019.

[21] P. Schneider and K. Bottinger, "High-performance unsupervised anomaly detection for cyber-physical system networks," in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy—CPS-SPC'18*, pp. 1–12, Barcelona, Spain, January 2018.

[22] T. Hamed, R. Dara, and S. C. Kremer, "Network intrusion detection system based on recursive feature addition and bigram technique," *Computers & Security*, vol. 73, pp. 137–155, 2018.

[23] B. A. Pratomo, P. Burnap, and G. Theodorakopoulos, "Unsupervised approach for detecting low rate attacks on network traffic with autoencoder," in *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–8, Glasgow, UK, June 2018.

[24] Z.-Q. Qin, X.-K. Ma, and Y.-J. Wang, "Attentional payload anomaly detector for web applications," in *Proceedings of the International Conference on Neural Information Processing*, pp. 588–599, Siem Reap, Cambodia, December 2018.

[25] H. Liu, B. Lang, M. Liu, and H. Yan, "CNN and RNN based payload classification methods for attack detection," *Knowledge-Based Systems*, vol. 163, pp. 332–341, 2019.

[26] Z. Zhao and G.-J. Ahn, "Using instruction sequence abstraction for shellcode detection and attribution," in *Proceedings of the Conference on Communications and Network Security (CNS)*, pp. 323–331, National Harbor, MD, USA, October 2013.

[27] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, p. 1, 2012.

[28] X. Wang, C.-C. Pan, P. Liu, and S. Zhu, "SigFree: a signature-free buffer overflow attack blocker," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 65–79, 2010.

[29] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: a content anomaly detector resistant to mimicry attack," *Lecture Notes in Computer Science*, in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pp. 226–248, Hamburg, Germany, September 2006.

[30] R. 7, "Metasploit penetration testing framework," 2003, https://www.metasploit.com/.

[31] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proceedings of the Military Communications and Information Systems Conference (MiLCIS)*, pp. 1–6, Canberra, ACT, Australia, November 2015.

[32] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.
[33] S. T. Brugger and J. Chow, "An assessment of the DARPA IDS evaluation dataset using snort," vol. 1, no. 2007, Tech. Report,. CSE-2007-1, p. 22, UCDAVIS Department of Computer Science, Davis, CA, USA, 2007.
[34] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A first look at modern enterprise traffic," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, p. 2, Berkeley, CA, USA, October2005.
[35] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.
[36] I Bro, 2008, http://www.bro-ids.org.
[37] Argus, 2008, https://qosient.com/argus/.
[38] R. 7, "Metasploitable," 2012, https://information.rapid7.com/download-metasploitable-2017.html.
[39] S. L. Garfinkel and M. Shick, "Passive TCP reconstruction and forensic analysis with tcpflow," Technical Reports, Naval Postgraduate School, Monterey, CA, USA, 2013.
[40] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, 2007.
[41] J. Pennington, R. Socher, and C. D. Manning, "GloVe: global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014.
[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT press, Cambridge, MA, USA, 2016.
[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, https://arxiv.org/abs/1412.3555.
[45] PyTorch, "Negative log likelihood," 2016.
[46] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.
[47] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: mobile encrypted traffic classification using multimodal deep learning," *Computer Networks*, vol. 165, Article ID 106944, 2019.
[48] H. Buchwald, "Shadow daemon: a collection of tools to detect, record, and block attacks on web applications," Shadow Daemon Introduction. 2015, https://shadowd.zecure.org/overview/introduction/.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 8850550, 20 pages
https://doi.org/10.1155/2020/8850550

WILEY | Hindawi

Research Article

HYBRID-CNN: An Efficient Scheme for Abnormal Flow Detection in the SDN-Based Smart Grid

Pengpeng Ding, Jinguo Li, Liangliang Wang, Mi Wen, and Yuyao Guan

College of Computer Technology and Science, Shanghai University of Electric Power, Shanghai 200090, China

Correspondence should be addressed to Jinguo Li; lijg@shiep.edu.cn

Received 9 April 2020; Revised 5 July 2020; Accepted 21 July 2020; Published 3 August 2020

Academic Editor: Yin Zhang

Software-Defined Network (SDN) can improve the performance of the power communication network and better meet the control demand of the Smart Grid for its centralized management. Unfortunately, the SDN controller is vulnerable to many potential network attacks. The accurate detection of abnormal flow is especially important for the security and reliability of the Smart Grid. Prior works were designed based on traditional machine learning methods, such as Support Vector Machine and Naive Bayes. They are simple and shallow feature learning, with low accuracy for large and high-dimensional network flow. Recently, there have been several related works designed based on Long Short-Term Memory (LSTM), and they show excellent ability on network flow analysis. However, these methods cannot get the deep features from network flow, resulting in low accuracy. To address the above problems, we propose a Hybrid Convolutional Neural Network (HYBRID-CNN) method. Specifically, the HYBRID-CNN utilizes a Deep Neural Network (DNN) to effectively memorize global features by one-dimensional (1D) data and utilizes a CNN to generalize local features by two-dimensional (2D) data. Finally, the proposed method is evaluated by experiments on the datasets of UNSW_NB15 and KDDCup 99. The experimental results show that the HYBRID-CNN significantly outperforms existing methods in terms of accuracy and False Positive Rate (FPR), which successfully demonstrates that it can effectively detect abnormal flow in the SDN-based Smart Grid.

1. Introduction

The Smart Grid is a grid system with automatic control and self-protection adjustment capabilities [1]. It is supported by information and communication technology to achieve reliability, security, and real-time requirements [2, 3]. The emerging network architecture Software-Defined Network (SDN) ignores the coaxial hardware structure of the network which separates the control plane and the data plane, and directly implements the virtualized configuration of the switch. It is especially suitable for mobile communication network, wired interconnection network, and sensor network in the Smart Grid [4]. The SDN improves the data transmission capability and network compatibility of the Smart Grid, but it also brings new security issues. The highly centralized network control capability and the damage caused by network abnormal flow intrusion have increased significantly [5]. As the control center of the whole network, the SDN itself may be the target of various attacks, such as DDoS, fake flow, breakthroughs in switches, and attacks on the control layer. The destruction of the SDN will cause all switches under its control to be paralyzed or disorders can have devastated effects on the entire network [6]. In the SDN, collaborative abnormal flow detection across multiple domains requires detailed flow data for each relevant domain, such as the contents of a flow table in the last few seconds. Network abnormal flow has the characteristics of potential and unforeseen attacks. Therefore, the detection technology of network abnormal flow is challenged by the demand for larger-scale and higher-dimensional flow data [7].

Recently, most of these studies are based on state transition [8] and artificial intelligence methods [9]. The method based on state transition requires manual calculation and has low recognition accuracy. The method based on artificial intelligence has more advantages in this respect

because of network big data. However, most of the researches have not carried out in-depth feature learning of network flow. For large-scale network abnormal flow detection, there are mainly two types of methods. The first type of method relies on sampling data, it uses network flow data to establish a library of attack intrusion behavior patterns, and the collected data including the host's system logs or collected from the network nodes matches the established pattern library. If the match is successful, it is proved to be an intrusion; otherwise, it is a normal behavior [10]. This method can effectively identify existing attacks and maintain them effectively and improve network security at the time. However, with the development of computers and the Internet, more and more new types of attacks appear in the field of vision. The detection accuracy of expert systems has fallen sharply. It has been unable to meet the requirements, and the sampling data itself is not accurate, which may cause the loss of useful information.

Another type of method is to utilize machine learning methods to perform feature extraction and detection classification after constructing features. The massive amount of network data makes machine learning methods more effective than judgment methods based on expert systems [11]. The traditional machine learning methods are just a shallow feature learning classifier. They have certain limitations when processing complex data. The feature processing that traditional machine learning must do is time consuming and requires specialized knowledge. The performance of most machine learning algorithms depends on the accuracy of the extracted features. Deep learning reduces the manual design effort of feature extractors for each problem by automatically retrieving advanced features directly from raw data [12]. Previous studies have used deep learning to classify mobile encrypted traffic and achieved excellent results [13, 14]. In [15], the authors investigated several deep learning architectures, including 1D CNN, 2D CNN, LSTM, Stacked Autoencoder (SAE), and Multilayer Perceptron (MLP) for mobile encrypted traffic classification. Based on this, this paper aims to apply the excellent feature learning capabilities of deep learning to the SDN-based Smart Grids to achieve highly accurate network abnormal flow detection.

To meet the above problems and challenges, we hope to apply the excellent feature learning capabilities of deep learning to the SDN-based Smart Grid to achieve highly accurate network abnormal flow detection. The main contributions of this article can be summarized as follows:

(i) First, we design a framework for improving the security of the Smart Grid by applying an abnormal flow detection algorithm in the SDN-based Smart Grid communication network; it can identify abnormal flow and detect the type of attack.

(ii) Second, we propose a deep learning algorithm of Hybrid Convolutional Neural Networks (HYBRID-CNN) to detect abnormal flow in the SDN-based Smart Grid communication network. The HYBRID-CNN adopts dual-channel data input, which can extract effective features from 1D and 2D flow data, use the self-attention mechanism to fuse key features, and finally use the fully connected neural network for detection.

(iii) Third, we compare the proposed method with the single model and verify the performance improvement of the hybrid model. In addition, we discuss a parameter study to optimize the HYBRID-CNN model.

(iv) Fourth, we perform a lot of experimental comparisons on the UNSW_NB15 and KDDCup 99 benchmark dataset. Experimental results show that the HYBRID-CNN significantly outperforms existing approaches in terms of accuracy and False Positive Rate (FPR).

The rest of this article is organized as follows: we discuss related work in Section 2 and introduce the system model and security requirements in Section 3. We then introduce some preliminary knowledge in Section 4. In Section 5, we introduce our proposed algorithm, and then in Section 6 we introduce experimental comparative analysis. Finally, we discuss and conclude in Sections 7 and 8.

2. Related Work

This section discusses two related types of work, namely, traditional machine learning and deep learning. In the SDN-based network controllers, using traditional machine learning and deep learning to develop flexible and efficient abnormal flow detection schemes presents some challenges. One of the main challenges is how to choose an appropriate feature selection method and another challenge is to accurately grasp the correlation between the selected feature and the abnormal flow detection task and the redundancy between these features [16].

2.1. Traditional Machine Learning. Most of the previous studies were based on traditional machine learning methods, such as Support Vector Machine (SVM), Decision Tree, and Naive Bayes. Naive Bayes algorithm is an important algorithm in the field of machine learning and data mining. It is widely used in the field of machine learning classification, such as text classification and medical diagnosis. Ashraf et al. [17] applied Naive Bayes for network intrusion detection; their basic idea is to select the most likely category based on the Bayesian algorithm under the assumption that the classification is based on feature independence. But this method is only simple shallow feature learning, and it has poor performance for large-scale network flow data. Rai et al. [18] used decision tree C4.5 to perform intrusion detection experiments on the NSL-KDD dataset. In this work, 16 attributes were selected as detection features on the dataset. The proposed algorithm can be used for feature-based intrusion detection, but its accuracy is too low, only 79.52%. Reddy et al. [19] proposed a filtering algorithm based on the SVM classifier to perform the classification task on the KDDCup 99 dataset. This method performed well on the training field but performed poorly in the test dataset and

could not effectively detect unknowns' network abnormal flow.

2.2. Deep Learning. In recent years, as a branch of machine learning, deep learning is becoming more and more popular. It is applied to intrusion detection and research shows that deep learning has completely surpassed traditional methods in performance [20]. Kwon et al. [15] utilized Deep Neural Network-based deep learning methods for flow-based anomaly detection. Experimental results evidence that deep learning can be applied to abnormal flow detection in the SDN. Long Short-Term Memory (LSTM) is a special deep learning model of Recurrent Neural Network. It can remember the input and predicted output of any period and solves the problem of gradient vanish and explosion in the Recurrent Neural Network (RNN). LSTM is widely used in the field of Natural Language Processing [21]. Existing researches have been done on abnormal flow detection based on LSTM [22], and they found that the algorithms have a significant performance improvement for sequence learning compared with traditional machine learning methods, but there is still room for improvement in detection rate and accuracy. CNN is a multi-layer network structure learning algorithm. It can learn hierarchical features from a large amount of data and has broad application prospects in the field of abnormal flow detection. Wang et al. [23] proposed an end-to-end classification method for one-dimensional Convolutional Neural Networks. This method integrates feature extraction, feature selection, and classifiers into a unified end-to-end framework and automatically learns original inputs and expectations. The nonlinear relationship between the outputs has obtained good experimental results. However, the one-dimensional data used in this method is not suitable for local feature extraction, resulting in the detection rate less than the ideal one. In [24], the authors present a new technique for network traffic classification based on a combination of RNN and CNN models that can be used for Internet of Things (IoT) traffic, which provides the best detection results. Wang et al. [25] proposed using CNN combined with LSTM to analyze and detect network flow. It utilizes CNN to learn low-level spatial features of network flow for the first time and then uses LSTM to learn high-level temporal features. The Deep Neural Network completes it automatically, and this method has achieved good results in terms of accuracy and detection rate.

Based on the above works, traditional machine learning methods that are typically used in abnormal flow detection often fail and cannot detect many known and new security threats, largely because those approaches provide less focus on accurate feature selection and classification. It is often inefficient for large-scale network flow. For the current deep learning methods like LSTM and CNN, they often pay more attention to the improvement of the model and ignore the original flow structure features. To address the above problems, we propose a HYBRID-CNN deep learning method for more accurate feature learning. The method utilizes two-channel input structure of 1D data and 2D data: using a CNN to extract local features and using a DNN to extract the global features. Specifically, a self-attention mechanism is added to select the most important features.

3. System Model View

In this section, we formalize the system model and system security requirements.

3.1. System Model. The Smart Grid uses two-way communication technology to connect many power components to ensure mutual communication between the components. Implementing the SDN on Smart Grid technology separates network control from data forwarding equipment that includes network infrastructure, thereby enabling logically centralized control and enabling the network to be programmed by a central software unit. The control layer, as the brain of the network, carries the controller software. The software-defined routing rules determine where to route flow. There are programmable network devices in the data plane to route flow according to the rules defined by the controller. The top of the module implements the function of the abnormal flow detection module. As shown in Figure 1, the SDN-based Smart Grid mainly includes the following parts [26].

3.1.1. Physical Plane. This layer is responsible for packet switching and routing. It includes the basic components of network communication in Smart Grid, such as smart meter, Power Management Unit (PMU), various sensors, and various communication equipment. Different from the traditional network, these basic components cannot make decisions independently because of no control unit. They are only responsible for collecting the generated key data and forwarding the collected data to the control layer through the programmable SDN switch infrastructure while complying with the rules defined by the controller.

3.1.2. Southbound Interface. The definition of south interface provides the communication protocol between the physical layer and the control layer. OpenFlow protocol developed by Stanford is currently the most common and standard protocol in south interface [27]. It can realize secure communication in the SDN by determining the message format from a programmable switch to controller.

3.1.3. Control Plane. As the central brain, the control layer has a SDN controller or more whose task is to manage the forwarding behavior of data flow by determining forwarding rules, which need to be written into the flow table of the programmable switch in the physical layer through the south interface.

3.1.4. Northbound Interface. The north interface definition provides an interface for communication between the control layer and the application layer and enables

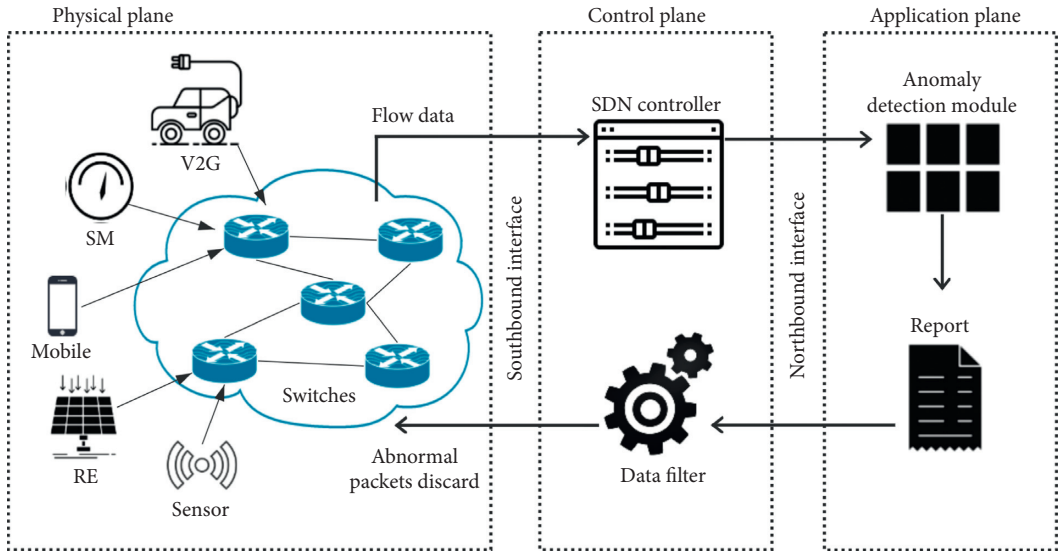


Figure 1: The system model of the SDN-based Smart Grid; it mainly includes physical plane, southbound interface, control plane, northbound interface, and application plane. Devices in the physical layer initiate access request through the Internet, and the flow collection module of the SDN controller captures all request flow statistics table information to extract flow features. HYBRID-CNN is used to detect abnormal traffic and generate abnormal reports. Then, the generated anomaly report is sent to the SDN controller through the security channel. Finally, the SDN controller discards attack packets and updates the flow table according to the received report.

application programs to program the network. It abstracts the details of data in the physical layer and allows network administrators, service providers, and researchers to customize the control rules and behaviors of their networks.

3.1.5. Application Plane. The application layer comprises many Smart Grid applications, including network security function programs such as abnormal flow detection module and flow data filtering module. All these application-defined policies need to be translated into OpenFlow rules that are transferred to the physical layer programmable switch and then transferred from the north interface to the control layer.

3.2. System Security Requirements

3.2.1. The Immovability and Concentricity of Network Architecture. The function of the Smart Grid communication network is generated with the design phase, and it is almost impossible to reconfigure the network based on the real-time needs of the network. In terms of performance and resilience, the bottlenecks will be caused by this nondynamic structure of today's Smart Grid. At the same time, the network will be vulnerable to multiple types of attacks. On the other hand, the highly centralized network control capability increases the damage caused by network abnormal flow intrusion considerably [28]. The SDN is the control center of the entire network. It may itself be the target of various attacks and these attacks will damage the SDN resulting in all its control paralysis or misbehavior of a switch can have a devastating effect on the entire network. Therefore, it is necessary to design an effective abnormal flow detection algorithm in the SDN controller.

3.2.2. The Hierarchy of Network Flow. Network flow has a distinct hierarchy, as shown in Figure 2, where the bottom row shows a sequence of flow bytes. According to a specific network protocol format, multiple flow bytes are combined into a network packet, and then multiple network packets are combined into a network flow. A network flow is divided into normal or malicious tasks, and a deep learning algorithm is used to learn hierarchical features, which has achieved good results. These studies urge us to use deep learning to learn the hierarchical features of network flow to complete the task of intrusion anomaly detection.

3.3. Working Methodology. Devices in the physical layer initiate access request through the Internet, and the flow collection module of the SDN controller captures all request flow statistics table information to extract flow features. The abnormal flow detection module includes three stages: data preprocessing, model training, and model validation, as shown in Figure 3. First, the collected flowmeter data are preprocessed, including data encoding, data normalization, data reshaping, and data split. After data preprocessing, the flow data vectors will be feature-extracted, feature-fused, and anomaly-detection-classified by the HYBRID-CNN algorithm.

In addition to the powerful anomaly flow detection above, the proposed solution performs end-to-end delivery of detection reports through the SDN as shown in Figure 1. This is achieved by incorporating the anomaly flow detection model into the core of the SDN control plane. The execution process works in the following order: (i) detection stage, (ii) reporting phase, and (iii) update phase. In the first stage, the control plane encapsulated with the anomaly flow detection model classifies the incoming flow as abnormal and normal. Then in the second stage, the report is communicated to the

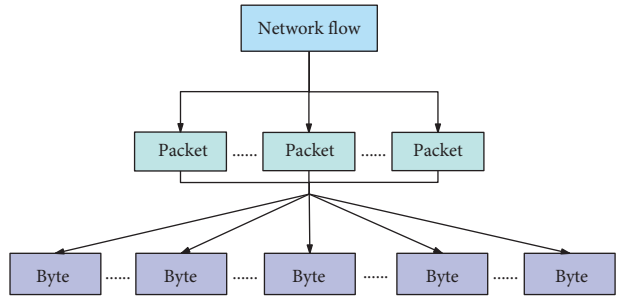


Figure 2: The structure of a network flow. Multiple bytes are combined into a packet, and then multiple packets are combined into a network flow.

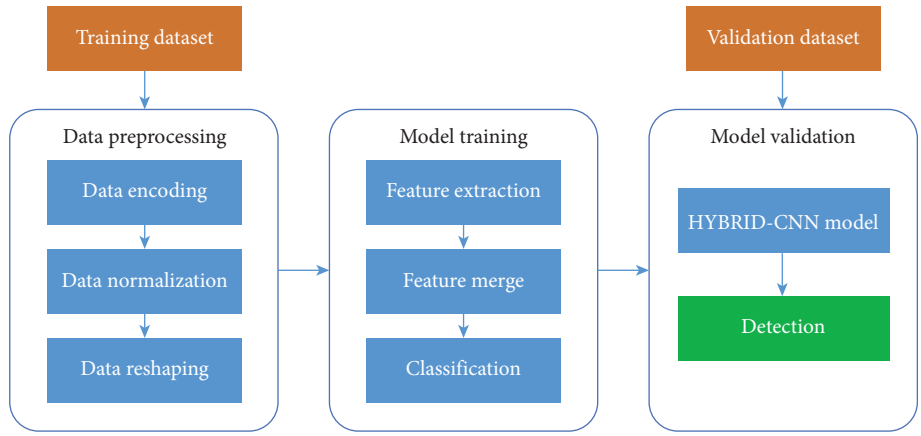


Figure 3: Working methodology of the proposed anomaly detection algorithm; it includes data preprocessing, model training, and model validation.

control plane. If the incoming flow is abnormal, the control plane discards the packet and immediately gives up communication with the requesting host. This helps protect the underlying network with malicious content and prevents it from spreading further on the network. During the update stage, the control plane updates the flow table entry of the forwarding device.

4. Preliminaries

In this section, we briefly describe the general notion used in our proposed algorithm.

4.1. Activation Function. The activation function provides the nonlinear modeling capability of the network. Rectified Linear Unit (ReLU) is the most widely used function [29]; it can keep the gradient from attenuating, thus effectively alleviating the problem of gradient disappearance; the function expression is as follows: the ReLU activation function produces 0 as an output when $x < 0$ and produces a linear with slope of 1 when $x > 0$:

$$\hat{y}' = \max(0, x). \tag{1}$$

4.2. Cross-Entropy Loss. Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. It increases as the predicted probability diverges from the actual label. In binary classification, where the number of classes **M** equals 2, the cross-entropy loss can be calculated as

$$\text{loss} = -(y\log(p)) - (1 - y)\log(1 - p). \tag{2}$$

If $\mathbf{M} > 2$ (i.e., multiclass classification), we calculate a separate loss for each class label per observation and sum the results:

$$\text{loss}' = -\sum_{c=1}^{M} y_{o,c}\log\left(p_{o,c,}\right), \tag{3}$$

where y is binary indicator (0 or 1) if class label c is the correct classification for observation o and p is predicted probability that observation o is of class c.

4.3. Optimizer. We use Adam optimizer to learn the network weight parameters. And independent adaptive learning rates are designed for different parameters with calculating the first-order moment estimation and the second-order moment estimation of the gradient. Empirical results prove that Adam has greater advantages over other optimizers in practice [30]. Moving averages of gradient $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$ and squared gradient $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$, bias corrected estimators for the

first moments $\hat{m}_t$ and second moments $\hat{v}_t = v_t/(1-\beta_2^t)$, the update rules for Adam are as follows:

$$\omega_t = \omega_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \tag{4}$$

where ω is model weights, η is the step size, and β, ε are hyperparameters.

5. Proposed HYBRID-CNN Algorithm

In this part, we first introduce the data preprocessing operation. Then, we describe the structure of HYBRID-CNN algorithm and how to detect abnormal flow.

5.1. Data Preprocessing

5.1.1. Data Encoding. The input flow data contains a variety of features; some of them are no-numeric types, so they need to be encoded as numeric types to be used as input to the neural network. Here, we use Label encoder encoding to convert discrete features to continuous features [31], such as [protocol: TCP, service: HTTP, state: FIN, ...] ⟶ [protocol: 4, service: 2, state: 2, ...].

5.1.2. Data Normalization. Data normalization can speed up the solution, improve the accuracy of the model, and prevent a feature with a particularly large value range from affecting the distance calculation. For the features that there is a very large scope in the difference between the minimum and maximum values, such as "dur," "sbytes," and "dbytes," we apply the logarithmic scaling method for scaling to obtain the features which are mapped to a range. We choose the MIN-MAX scaling method [31] and normalize the data according to the following equation:

$$X_i = \frac{X_i - X_{\min}}{X_{\max} - X_{\min}}, \tag{5}$$

where X_i denotes each data point, $X_{\min}$ denotes the minimum value from all data points, and $X_{\max}$ denotes the maximum value from all data points for each feature.

5.1.3. Data Reshaping. For CNN input, its format should be three-dimensional data (height, width, channel), and as a single sample, the channel should be 1, so that we can reshape a single flow sample with a length of $s = h^* w + 1$ to obtain a data structure similar to an image and construct a matrix M of $h^* w$, namely,

$$M' = \begin{pmatrix} M_{11} & \cdots & M_{1w} \\ \vdots & \ddots & \vdots \\ M_{h1} & \cdots & M_{hw} \end{pmatrix}. \tag{6}$$

5.1.4. Data Split. For every model we want to train, each model has two datasets: one is the training dataset and the other is the validation dataset. As shown in Figure 4, in order to separate them, we first apply the shuffle method on the dataset to generate random data and then slice the entire dataset to obtain a training dataset and a validation dataset.

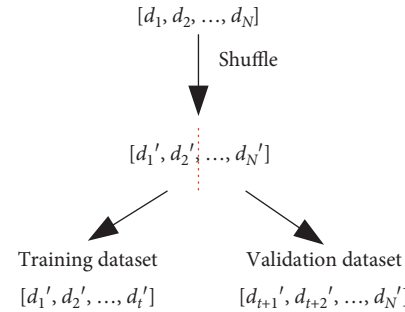


Figure 4: Data split. Using the shuffle method on the network flow dataset to generate random data, and then splitting the random data into a training dataset and a validation dataset.

5.2. HYBRID-CNN.

The structure of CNN is shown in Figure 5. It is an end-to-end deep learning model with powerful feature learning and classification capabilities. It is widely used in image classification, speech recognition, computer vision, and other fields [32].

The network flow contains both abnormal and normal flow, and HYBRID-CNN training is performed at this stage to detect misused attacks, which aims to further categorize the malicious data from stages into corresponding classification strategies, i.e., Scan, R2L, DoS, and Probe. The structure of our proposed HYBRID-CNN algorithm is shown in Figure 6. We divide it into three parts. The first part is feature extraction, the second part is feature fusion, and the third part is the detection classification.

5.2.1. Feature Extraction. In the feature extraction phase, we use the form of dual input of flow data, which aims to extract the features of flow more comprehensively. The role of the input layer is to receive input data, and the size of the input layer is consistent with the size of the input data, such as a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, or a matrix $\mathbf{M}$.

For the first input (the upper part of the blue box), every user's access flow essentially is 1D data. We utilize two layers of DNN to extract the global features of the flow. Our motivation is to learn the frequent co-occurrence of features pass by memorizing one-dimensional data. The calculation method of each neuron in the fully connected layer is

$$x_i = f\left(\sum_{i=1}^{n} \omega_{i,j} x_i + b_1\right). \tag{7}$$

After the data preprocessing, its input shape is $(h^* w, 1)$. In layer 1, we set a neuron, and the shape of the output data is $(h^* w, a)$. In the fully connected layer 2, we set b neurons, and the shape of the output data is $(h^* w, b)$. The two-dimensional data is straightened to obtain a one-dimensional feature vector of $h^* w^* b, 1$. In this process, the activation function used is ReLU to obtain the output feature O_{wide}.

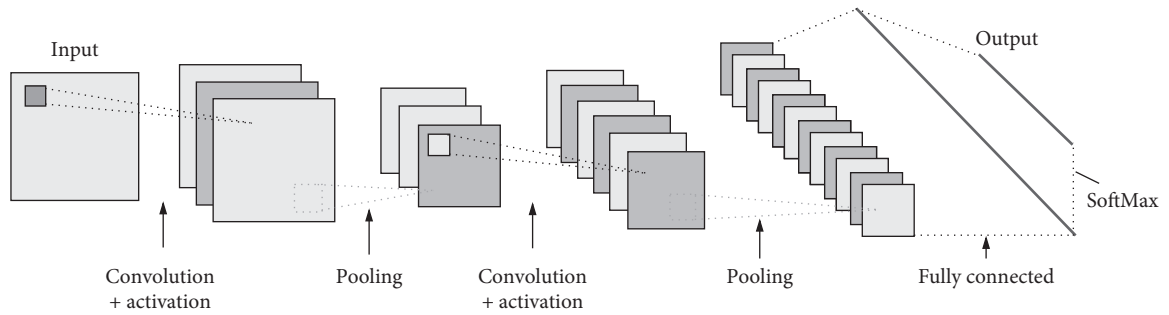


Figure 5: The structure of a CNN. It includes input layer, convolution layer, activation function, pooling layer, fully connected layer, and output layer.

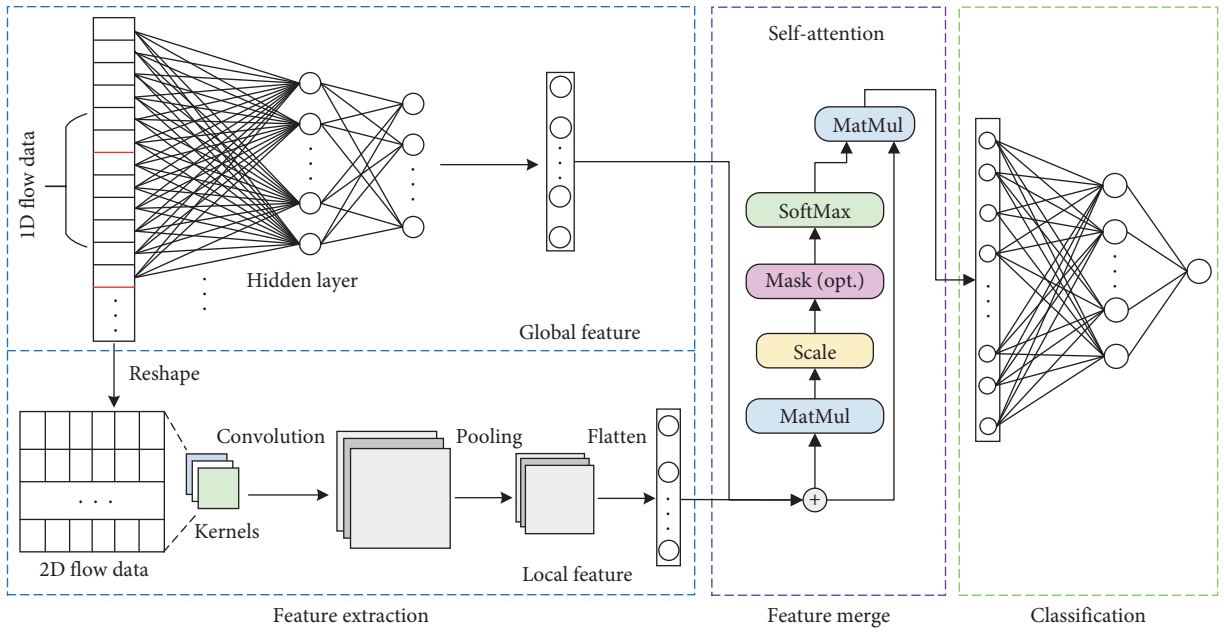


Figure 6: The structure of the proposed HYBRID-CNN algorithm; it includes feature extraction, feature merge, and classification. The feature extraction aims to extract the feature of flow more comprehensively, the self-attention mechanism aims to fuse key feature, and the classification aims to classify accurately.

For the second input (the lower part of the blue box), we reshape the one-dimensional data of the first input into a two-dimensional matrix. We believe that the deeper features can be better learned in the form of two-dimensional matrix input. The CNN uses a sliding convolution kernel to extract local features of flow data. In this part of the network, a convolution layer, a pooling layer, and a flatten layer are included.

One of the limitations of conventional neural networks is poor scalability due to the full connection of neurons; CNN overcomes this shortcoming by convolving each neuron to its neighbors instead of all neurons [33]. Set the input of the i-th layer to x^{l+1}, the output to x^l, and the convolution kernel to k. The convolution operation is performed by the following equation:

$$x^l = f\left(\sum x_i^{l-1} \otimes k_i^l + b^l\right), \tag{8}$$

where $f(\cdot)$ is a nonlinear activation function, $\otimes$ is a convolution sign, and b^l is a bias term. The pooling layer is usually placed after the convolutional layer. By performing a merge operation on a local area of the feature map, the feature has a certain spatial invariance. The merge operation reduces feature size and prevents overfitting. x^{l+1} is obtained by the following pooling:

$$x^{l+1} = \beta\ \mathrm{down}\left(x^l\right) + b, \tag{9}$$

where $\mathrm{down}(\cdot)$ represents the pooling function, β is a multiplicative bias, and b is additive bias. The reshaped shape of the input data is (h, w). We use k convolution kernels with the same shape to extract the convolution features. At first, the data shape is $(h - k + 1, k)$; after pooling, the shape of the data is $((h - k + 1)/2, k)$. Then, through the flatten layer, the data shape is $((h - k + 1)/2^*k, 1)$, and the output feature O_{CNN} is obtained.

For the two extracted features, perform feature fusion to obtain the feature $O_i(k)$:

$$O_i(k) = O_{\mathrm{wide}} + O_{\mathrm{CNN}}. \tag{10}$$

5.2.2. Feature Merge. In the feature fusion part, we use a self-attention mechanism to fuse key features. The essence of the self-attention mechanism is to observe a specific part according to the observation of the need [34].

For self-attention, we get three matrices Q (Query), K (Key), and V (Value) from the input $O_i(k)$. The self-attention mechanism obtains different representations, calculates scaled dot-product attention of each representation, and finally concatenates the results. Specifically, the current representations input into the self-attention layer, and the new representation is calculated. First, we have to calculate the point product between Q and K, and then in order to prevent the result from being too large, it will be divided by a scale $\sqrt{d_k}$, where d_k is the dimension of a query and key vector, and then the results are normalized to a probability distribution using a SoftMax operation and then multiplied by the matrix V to obtain a weighted summation representation. This operation can be expressed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{11}$$

5.2.3. Classification. After feature fusion, we use a fully connected layer for detection and classification; all neurons in the previous layer are connected to each neuron in the current layer. The fully connected layer is located before the output layer. After the extracted features are converted into a one-dimensional feature vector, they are connected to each neuron in the current layer to map the high-level features in a targeted manner:

$$x_i' = f\left(\sum_{i=1}^{n} \omega_{i,j} x_i + b_1\right). \tag{12}$$

The fully connected layer will target high-level features according to the specific tasks of the output layer perform mapping and use the SoftMax and Sigmoid activation function after mapping to get the final classification detection result (normal, abnormal, or attack types).

The output layer is a SoftMax function [35]; it normalizes K real numbers into a K probabilities distribution, after applying SoftMax, each component will be in the interval $(0, 1)$, and the components will add up to 1, which can be interpreted to map the nonnormalized output of a network to a probability distribution over predicted output classes. Set $z = (z_1, \ldots, z_K) \in \mathbb{R}^K$; the standard SoftMax function σ: $\mathbb{R}^K \longrightarrow \mathbb{R}^K$ is defined by the formula:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{for } j = 1, \ldots, K. \tag{13}$$

Hence, the predicted class would be $\hat{y}$:

$$\hat{y} = \arg\max\left[\sigma(z)_j\right]. \tag{14}$$

6. Experimental Evaluation

To evaluate the proposed abnormal flow detection scheme, we conduct the simulation on a 64-bit computer with Intel (R) i7-9750 Hz 2.60 GHz CPU, 8 GB RAM, NVIDIA GeForce RTX 2060 6G GDDR6 GPU, and 10.2 CUDA, using Python, Scikit-learn, NumPy, Pandas, TensorFlow, and Keras. The data we use comes from an online public dataset. We carried out model comparison experiments to verify that the mixed model has higher accuracy than the single model. Compared with traditional machine learning methods and deep learning methods, the experimental results show that our method is superior to these methods.

6.1. Experimental Setup

6.1.1. Experimental Data. The dataset we are using is UNSW_NB15 on network intrusion detection [36], which is a mixture of real normal activity flow and attack flow created by the Australian Network Security Center in the network laboratory using IXIA Perfect Storm tool. Table 1 is the list of features and categories.

These features are categorized into five groups:

(i) Basic features: they involve the attributes that represent protocols connections

(ii) Flow features: they include the identifier attributes between hosts (e.g., server-to-client or client-to-serve)

(iii) Content features: they encapsulate the attributes of TCP/IP; also, they contain some attributes of http services

(iv) Time features: they contain the attributes time, for example, arrival time between packets, start/end packet time, and round-trip time of TCP protocol

(v) Additional generated features: this category can be further divided into two groups: general-purpose features, whereby each of them has its own purpose, to protect the service of protocols, and connection features that are built from the flow of 100 record connections based on the sequential order of the last time feature

To label this dataset, two attributes were provided: attack_cat represents the nine categories of the attack and the normal, and label is 0 for normal and otherwise is 1.

6.1.2. Performance Metrics. The performance metrics for abnormal flow detection depend on the confusion matrix constructed for any proven classification problem [37]. Its size depends on the number of classes contained in the dataset. Its main purpose is to compare the actual tags with the predicted tags. The intrusion detection problem can be defined by a 2×2 confusion matrix, which includes normal and attack categories for evaluation. The detailed description of the confusion matrix is shown in Table 2.

TP and TN denote the conditions for correct classification, while FP and FN denote the conditions for the mistaken classification. TP and TN refer to correctly classified attack flow and normal flow, respectively, while

Table 1: Features of the UNSW_NB15 dataset.

No.	Feature name	Category
(1)	dur	Numeric
(2)	Proto	Nonnumeric
(3)	service	Nonnumeric
(4)	state	No-numeric
(5)	spkts	Numeric
(6)	dpkts	Numeric
(7)	sbytes	Numeric
(8)	dbytes	Numeric
(9)	rate	Numeric
(10)	sttl	Numeric
(11)	dttl	Numeric
(12)	sload	Numeric
(13)	dload	Numeric
(14)	sloss	Numeric
(15)	dloss	Numeric
(16)	sinpakt	Numeric
(17)	dinpakt	Numeric
(18)	sjit	Numeric
(19)	djit	Numeric
(20)	swin	Numeric
(21)	dwin	Numeric
(22)	stcpb	Numeric
(23)	dtcpb	Numeric
(24)	tcprtt	Numeric
(25)	synack	Numeric
(26)	ackdat	Numeric
(27)	smean	Numeric
(28)	dmean	Numeric
(29)	trans_depth	Numeric
(30)	response_body_len	Numeric
(31)	ct_srv_src	Numeric
(32)	ct_state_ttl	Numeric
(33)	ct_dst_ltm	Numeric
(34)	ct_src_dport_ltm	Numeric
(35)	ct_dst_sport_ltm	Numeric
(36)	ct_dst_src_ltm	Numeric
(37)	is_ftp_login	Numeric
(38)	ct_ftp_cmd	Numeric
(39)	ct_flw_http_mthd	Numeric
(40)	ct_src_ltm	Numeric
(41)	ct_srv_dst	Numeric
(42)	is_sm_ips_ports	Numeric

Table 2: Confusion matrix for binary classification problem.

Predicted	Actual	
	Negative	Positive
Negative	TN (true negative)	FP (false positive)
Positive	FN (false negative)	TP (true positive)

FP and FN refer to misclassified normal and attack records, respectively. These four items are used to generate the following performance evaluation metrics.

The Accuracy (Acc) is a measure used to evaluate the overall success rate of the model in detecting normal records and abnormal flow and is calculated as

$$\text{Acc} = \frac{\text{TN} + \text{TP}}{\text{TP} + \text{FP} + \text{TN} + \text{TP}}. \quad (15)$$

The Detection Rate (DR), also known as the True Positive Rate (TPR), is the ratio of correctly classified malicious flow instances to the total number of malicious flow instances. The calculation formula is

$$\text{DR} = \frac{\text{TP}}{\text{FN} + \text{TP}}. \quad (16)$$

The False Positive Rate (FPR) is the proportion of normal instances that are misclassified as attack flow in the total number of normal instances. The formula is

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \quad (17)$$

The Precision (Pre) represents the proportion of the actual normal samples to the samples divided into normal; the formula is

$$\text{Pre} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \quad (18)$$

The $F1$ score is used to synthesize precision and recall as an evaluation index. The formula is

$$F1\text{score} = \frac{2^{*}\text{Pre}^{*}\text{DR}}{\text{Pre} + \text{DR}}. \quad (19)$$

6.2. Performance Comparison

6.2.1. Model Comparison. For comparison, we used a single CNN model and a simple DNN model. Our proposed hybrid CNN model includes 2 input layers, 1 convolutional layer, 1 pooling layer, and 4 fully connected layers. A single CNN model includes a convolutional layer, a pooling layer, and a fully connected layer. The simple DNN model contains only 3 fully connected layers.

The configuration of the model structure parameters in this paper is shown in Figure 7. Each column is a model. The input data shape of the DNN part of our proposed hybrid CNN model is (42,1), the data shape through Dense1 is (42,128), the data shape through Dense_2 is (42,64), and then the data shape through Flatten_1 is (2688), the shape of the input data of the CNN is (6,7) through the Conv1D_1 layer, the shape of the data becomes (4,32), followed by Pooling_1, and the shape of the data becomes (2,32). In the Merge layer, the two-channel data are merged into one. After this layer, the shape of the data becomes (2752) and then passes through the Dense_3 layer. As a result, the same shape is formed in each model by these layers in turn.

As shown in Table 3, we set the initial weight parameters to random values, set the batch size to 512, and use our Adam optimizer and binary_cross-entropy loss function to compile the model. To evaluate the performance of the model, we use accuracy as a metric function during training verification.

After the model is compiled, we use the input data to perform model training in batch mode and evaluate the performance index values at the end of each epoch. One epoch means that all training datasets have undergone a complete training iteration. The training results are shown in Figure 8, where the horizontal axis represents the number of

<table>
<tr><td colspan="5">Hybrid CNN model</td><td colspan="2">Single DNN model</td><td colspan="2">Single CNN model</td></tr>
<tr><td colspan="2">Input1: (42, 1)</td><td colspan="2">Input2: (6, 7)</td><td></td><td colspan="2">Input4: (42, 1)</td><td colspan="2">Input3: (6, 7)</td></tr>
<tr><td rowspan="2">Dense_1</td><td>(42, 1)</td><td rowspan="2">Conv1D</td><td>(6, 7)</td><td></td><td rowspan="2">Dense_1</td><td>(42, 1)</td><td rowspan="2">Conv1D</td><td>(6, 7)</td></tr>
<tr><td>(42, 128)</td><td>(4, 32)</td><td></td><td>(42, 128)</td><td>(4, 32)</td></tr>
<tr><td rowspan="2">Dense_2</td><td>(42, 128)</td><td rowspan="2">Pooling</td><td>(4, 32)</td><td></td><td rowspan="2">Dense_2</td><td>(42, 128)</td><td rowspan="2">Pooling</td><td>(4, 32)</td></tr>
<tr><td>(42, 128)</td><td>(2, 32)</td><td></td><td>(42, 128)</td><td>(2, 32)</td></tr>
<tr><td rowspan="2">Flatten_1</td><td>(42, 64)</td><td rowspan="2">Flatten_2</td><td>(2, 32)</td><td></td><td rowspan="2">Flatten_1</td><td>(42, 64)</td><td rowspan="2">Flatten_2</td><td>(2, 32)</td></tr>
<tr><td>2688</td><td>64</td><td></td><td>2688</td><td>64</td></tr>
<tr><td colspan="5">Attention merge layer: (2752)</td><td colspan="2">–</td><td colspan="2">–</td></tr>
<tr><td colspan="9">Dense_3: (32)</td></tr>
<tr><td colspan="9">Dense_4: (1)</td></tr>
</table>

Figure 7: Model configuration parameters.

Table 3: Configuration parameters for different models.

Methods	Origin weights	Batch_size	Activation
Single DNN model	Random	512	ReLU
Single CNN model	Random	512	ReLU
Our proposed model	Random	512	ReLU

epochs trained, and the vertical axis represents the loss and accuracy score values. We observe that the loss of our proposed hybrid CNN model becomes smaller and smaller as the training progresses, and after 100 epochs of training, it obtains higher accuracy scores than the single CNN model and DNN model.

6.2.2. Method Comparison. To evaluate the performance of our proposed hybrid CNN model, we performed experiments on UNSW_NB15 dataset. The comparison methods selected are as follows:

(i) Naive Bayes [17]: Naive Bayes is a supervised learning classifier based on Bayes theorem. It classifies the problem by combining previous calculated likelihood and probabilities to make the next probability using Bayes rule.

(ii) SVM [19]: an SVM is a discriminative classifier formally defined by separating hyperplanes. SVM-based kernels classify the data which effectively works for most of the datasets. Discriminant function: "Linear SVM."

(iii) LSTM [22]: the improved model based on RNN for intrusion detection, using ReLU activation function, Adam optimizer, 100 epoch, and two-layer LSTM {128, 64}.

(iv) CNN-LSTM [25]: a CNN combined with LSTM to analyze and detect network flow. It utilizes CNN to learn low-level spatial features of network flow for the first time and then uses LSTM to learn high-level temporal features, using ReLU activation function, Adam optimizer, 100 epoch, and two-layer LSTM {128, 64}; two-layer CNN includes pooling layer.

Table 4 lists the performance comparison between our proposed HYBRID-CNN and some other existing methods. It is worth noting that we select a subset for experiments based on a certain training dataset ratio. The training dataset ratio is defined as the proportion of training samples. The proportion of the dataset is 60%, 70%, and 80%. In each dataset of experiments, we evaluated five methods including our proposed method and evaluated three performance metrics (Acc, DR, FPR). The experimental results in Table 4 show that our proposed HYBRID-CNN compared with other traditional machine learning methods and deep learning methods. Compared with other methods, our proposed HYBRID-CNN can reach Accuracy of 0.9564, DR of 0.9856, and FPR of 0.0442, which means that our proposed method has higher accuracy in detecting abnormal flow than other traditional methods. It is because the combination input using a DNN and CNN has better feature learning capabilities.

Figure 9 is a comparison of the training and validation accuracy and loss between our proposed HYBRID-CNN method and the other two methods. All models have been trained for 100 epochs, and performance indicators have been evaluated after each epoch. By comparison, we can find HYBRID-CNN in the training and validation process of the method; the loss convergence speed is much faster. And the best results can be achieved faster for the accuracy improvement, which is obviously better than other methods.

6.2.3. ROC Curves Comparison. We further plot the Receiver Operating Characteristic (ROC) curves of our proposed HYBRID-CNN and state-of-the-art methods on UNSW_NB15, as shown in Figure 10. The ROC curve of HYBRID-CNN is the closest one to the upper left corner,

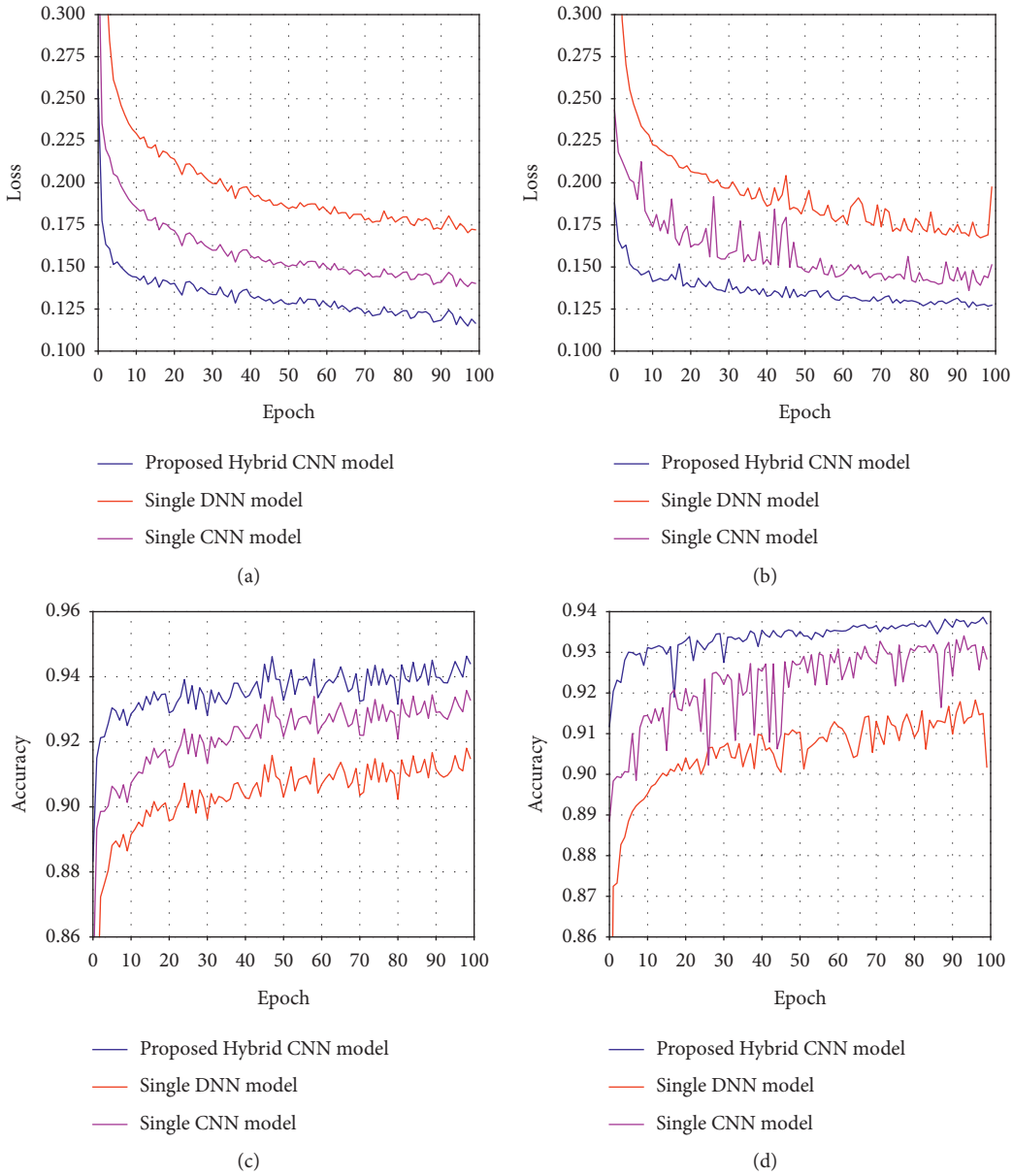


Figure 8: Comparison of different models. (a) Training loss. (b) Validation loss. (c) Training accuracy. (d) Validation accuracy.

Table 4: Performance comparison of the proposed and state-of-the-art methods.

Reference	Method	Proportion = 80%			Proportion = 70%			Proportion = 60%		
		Acc	DR	FPR	Acc	DR	FPR	Acc	DR	FPR
Ashraf et al. [17]	Naive Bayes	0.7663	0.8514	0.3841	0.7669	0.8611	0.3999	0.7655	0.8512	0.3883
Reddy et al. [19]	SVM	0.7594	0.6895	0.1170	0.7257	0.7806	0.3714	0.7346	0.7874	0.3591
Xin et al. [22]	LSTM	0.8916	0.9843	0.2724	0.8897	0.9840	0.2775	0.8894	0.9835	0.2778
Wang et al. [25]	CNN-LSTM	0.8995	0.9612	0.2095	0.8965	0.9460	0.1910	0.8955	0.9571	0.2138
Proposed method	**HYBRID-CNN**	**0.9564**	**0.9856**	**0.0442**	**0.9408**	**0.9382**	**0.0544**	**0.9386**	**0.9493**	**0.0803**

indicating better generalization ability against the other methods. All the results reported above demonstrate that HYBRID-CNN outperforms its competitors. We can conclude that HYBRID-CNN effectively handles the abnormal flow detection problem by the ability to compress the original data to more discriminative abstract features, and HYBRID-CNN is capable of efficient abnormal flow detection.

6.2.4. Computation Comparison. To deepen this investigation, Table 5 reports the number of training parameters

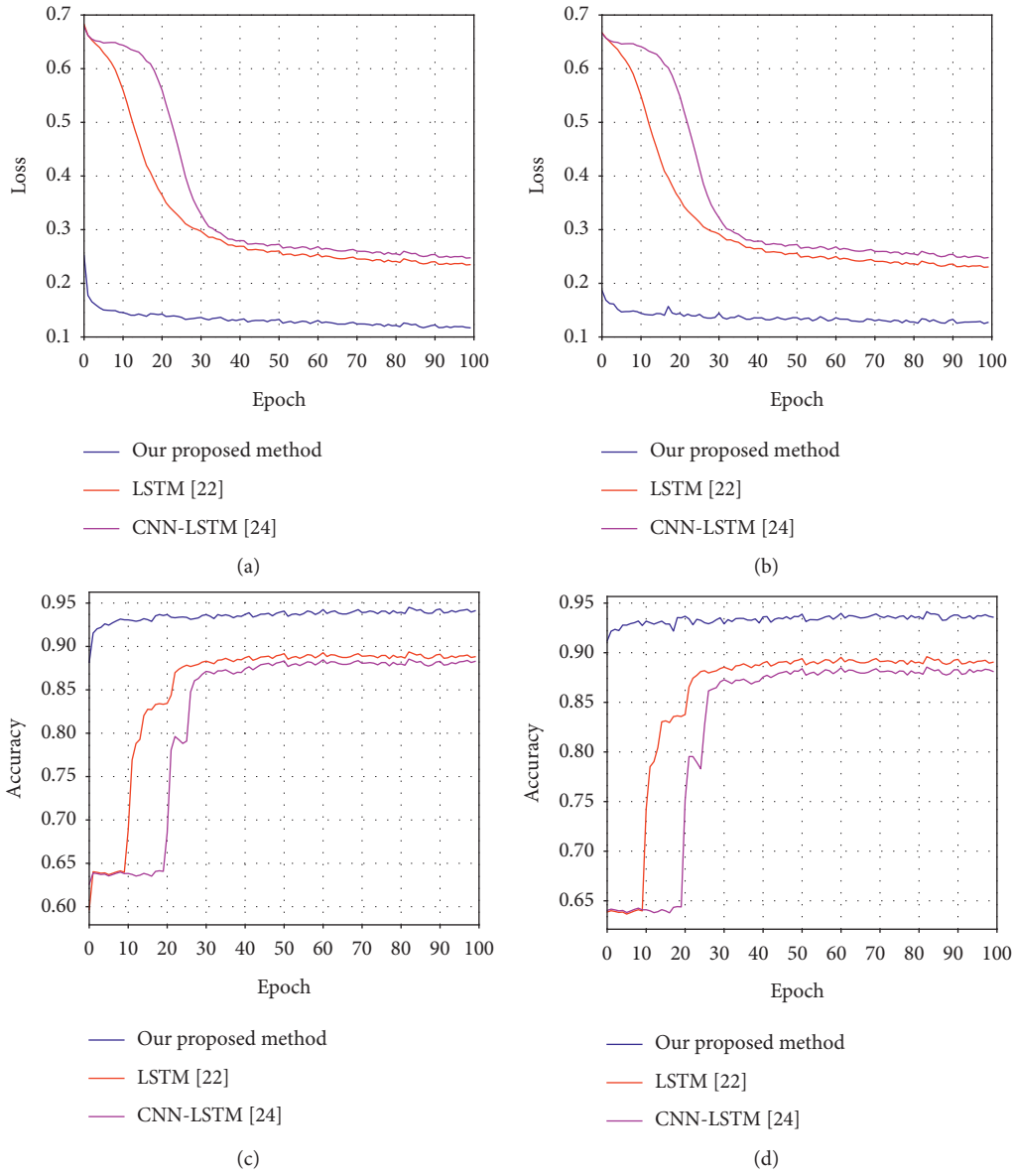


Figure 9: Comparison of different methods. (a) Training loss. (b) Validation loss. (c) Training accuracy. (d) Validation accuracy.

(in millions) and running time required for both the proposed HYBRID-CNN and state-of-the-art methods. We use GPU to accelerate the training speed of all models. It can be noticed that, when training on the UNSW_NB15 dataset, the proposed HYBRID-CNN has fewer trainable parameters and lower training time and testing time. This outcome results from the use of CNN in the proposed method, which can realize efficient parallel computation, and we use as small number of parameters as possible in the structure.

6.3. Parameter Study. There are various configurable hyperparameters in the model, such as Batch_size α, number of convolution kernels β, convolution kernel size γ, and optimizer ε. These hyperparameters can only be configured manually but cannot be optimized automatically through the training process, which will greatly affect the performance of the model. Batch_size α is the number of training samples of the neural network after one forward-propagation and back-propagation operation, which means how many samples will be used to evaluate the loss in each optimization process; β is the number of different convolution kernels used in convolution operation, how many convolution kernels there are, and how many feature maps will be generated after convolution; γ is the size of convolution kernels. Each convolution kernel has three dimensions of length, width, and depth. In a convolution layer of CNN, the length and width of convolution kernels need to be manually configured. Optimizer ϵ is the type of optimizer used to optimize loss and then update weight parameters. Therefore, we deeply analyzed the influence of these super parameters on the performance of our proposed hybrid

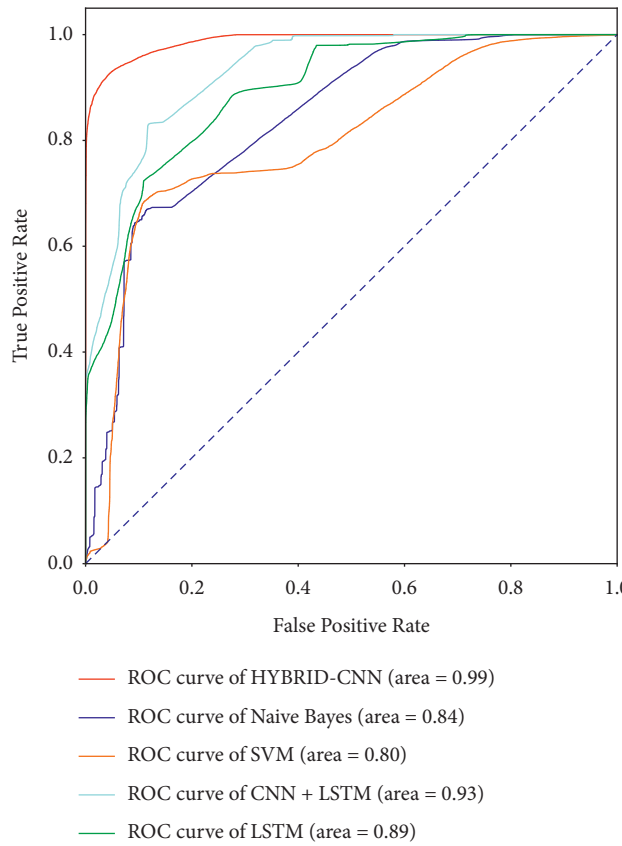


Figure 10: ROC curves of HYBRID-CNN and state-of-the-art methods on UNSW_NB15 dataset.

Table 5: The comparison of the computational complexity of the proposed and state-of-the-art methods.

Method	Trainable parameters	Training time (s)	Testing time (s)
LSTM	0.1391	402.58	1.77
CNN-LSTM	0.1404	526.31	8.99
Proposed	0.0951	271.26	0.75

CNN model. In Figure 7, the parameters of the hybrid CNN model proposed by us are $\alpha = 512$, $\beta = 4$, $\gamma = 1 \times 3$, and $\varepsilon =$ Adam. The model training results for these parameters are as follows.

6.3.1. Effect of Batch_size α. As shown in Figure 11, we set α to 128, 256, and 512 for experiments. When $\alpha = 128$, the training and validation loss converge faster in the same period and finally reach the set number of iterations. The best effect is 0.9477. We can know that a smaller Batch_size can speed up the optimization in the same period, but it means that more calculation time is needed to optimize. Increasing the Batch_size properly can improve the running speed and gradient descent direction. With accuracy increasing, the amplitude of training vibration decreases.

6.3.2. Effect of Number of Convolution Kernels β. As shown in Figure 12, we set the number of convolution kernels β as 1, 2, and 4 for experiments. When the number of convolution kernels is 1, we can get an accuracy of 0.9403. When the number of convolution kernels increases to 2, the loss convergence rate also increases. At 4, the speed of loss convergence is significantly accelerated. Generally, when the network is deeper, more convolution kernels are often required to fully extract key features.

6.3.3. Effect of Convolution Kernel Size γ. As shown in Figure 13, we set the size γ of the convolution kernel to 1×2, 1×3, and 1×4 for experiments. When the size of the convolution kernel is 1×2, the training loss and accuracy rate will jitter sharply. It is not conducive to convergence. When the size of the convolution kernel is increasing, the loss converges a little faster and the fluctuation range becomes smaller, so it should be better to choose a 1×3 or 1×4 size convolution kernel.

6.3.4. Effect of Optimizer ε. As shown in Figure 14, we have selected several commonly used optimizers SGD, RMSprop, Adam, and Adagrad for experimental comparison. When

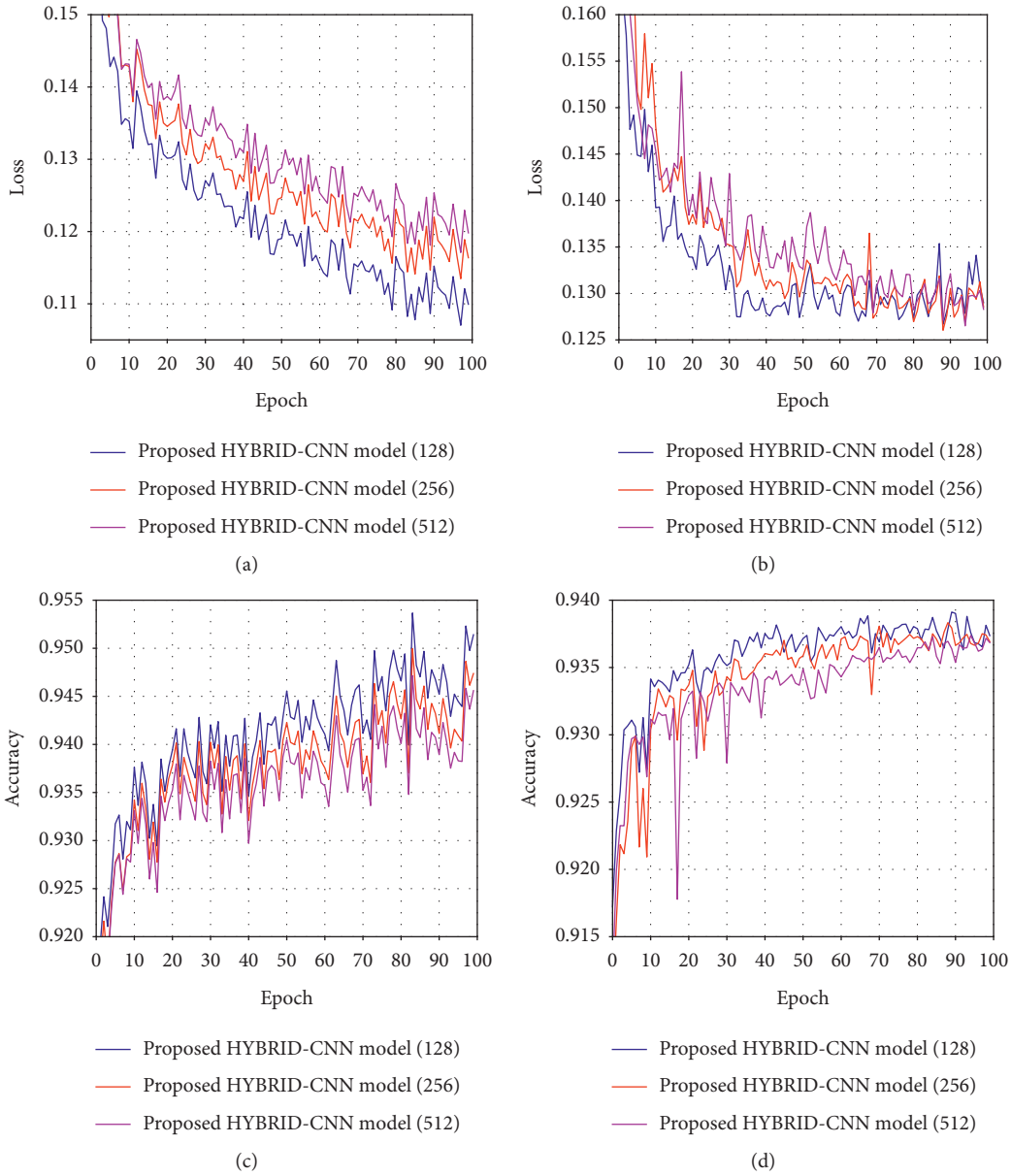


Figure 11: Parameter study of α. (a) Training loss. (b) Validation loss. (c) Training accuracy. (d) Validation accuracy.

SGD is used as an optimizer, the effect is not ideal. It can only achieve an accuracy of 0.9259. There was a large shock at around 40. We can see that when Adam optimizer is used, the initial loss convergence is like other optimizers. In the medium term, the Adam optimizer loss convergence is significantly faster and finally achieves the best. The accuracy is 0.9483.

6.4. Ablation Study. For a thorough analysis, we conduct an ablation study on HYBRID-CNN to analyze the effectiveness of each module. The details of the ablation study based on UNSW_NB15 are listed as follows:

(1) w/o attention: we remove the self-attention module from HYBRID-CNN but keep the DNN module and the CNN module

(2) w/o DNN: the DNN module is removed from HYBRID-CNN

(3) w/o CNN: the CNN module is removed from HYBRID-CNN

We further analyzed the detailed performance of HYBRID-CNN in the ablation study, and the results of the ablation studies are shown in Table 6. Comparing HYBRID-CNN with model (1), we can conclude that the self-attention module can help detect abnormal flow, because attention can capture key features more comprehensively. The effectiveness of DNN can also be demonstrated by comparing HYBRID-CNN with model (2). When we removed the DNN module, accuracy declined because the model could not extract high-dimensional global features. However, when the CNN module was removed, it could be found that the accuracy was greatly reduced,

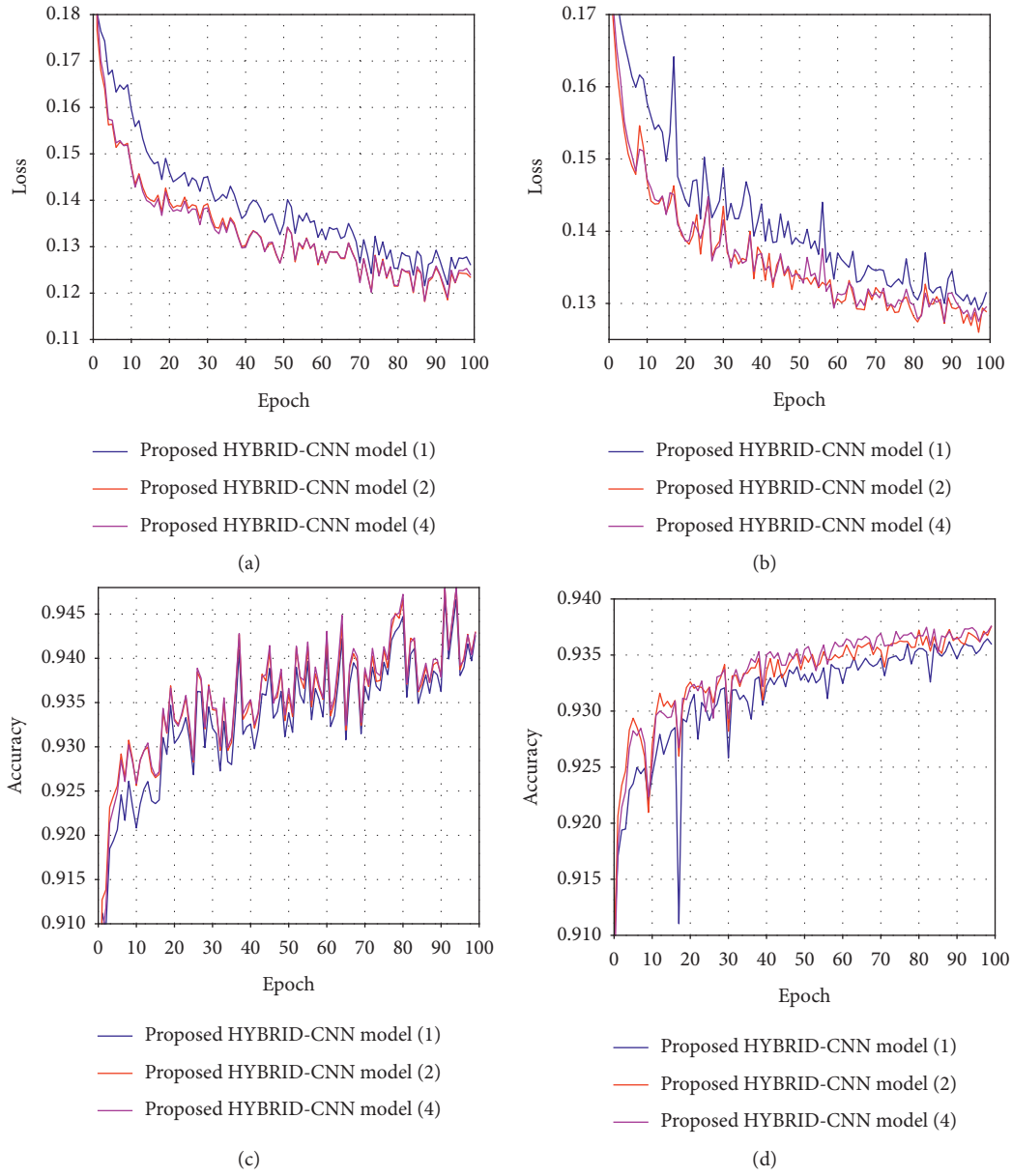


Figure 12: Parameter study of β. (a) Training loss. (b) Validation loss. (c) Training accuracy. (d) Validation accuracy.

because the model could not extract the local features of the flow, and CNN has a great impact on the results.

6.5. Attack Detection. In order to detect the attack type of abnormal flow, the dataset we used to evaluate the model was KDDCup 99 [38]. The entire dataset has approximately 5 million flow records, each of which has 41 features (the 1–9 features are the basic attributes of the packet, the 10–22 features are the packet content, and the 23–31 features are flow function and 32–41 are host-based features). As shown in Table 7, these attack flow instances can be further divided into DoS, U2R, R2L, and Probe. For the KDDCup 99 dataset, the flow sample has 41 features and a label. We cannot directly reshape a one-dimensional flow dataset into a two-dimensional matrix, so a zero feature is used here to add a dummy feature. It does not affect the result and is just for data reshaping.

We made comparisons with the current latest technology, and Figure 15 illustrates the relative comparison of our proposed abnormal flow detection algorithm with the current latest technology model. It is obvious from the obtained results that the proposed model performs better on the KDDCup 99 dataset than the existing scheme in terms of Accuracy, Detection Rate, and $F1$ score. Figure 15(a) shows the Precision evaluation of the proposed method corresponding to Normal, PROBE, DoS, U2R, and R2L data examples (99.92%, 98.11%, 99.98%, 93.81%, and 93.16%, respectively). Figure 15(b) shows the

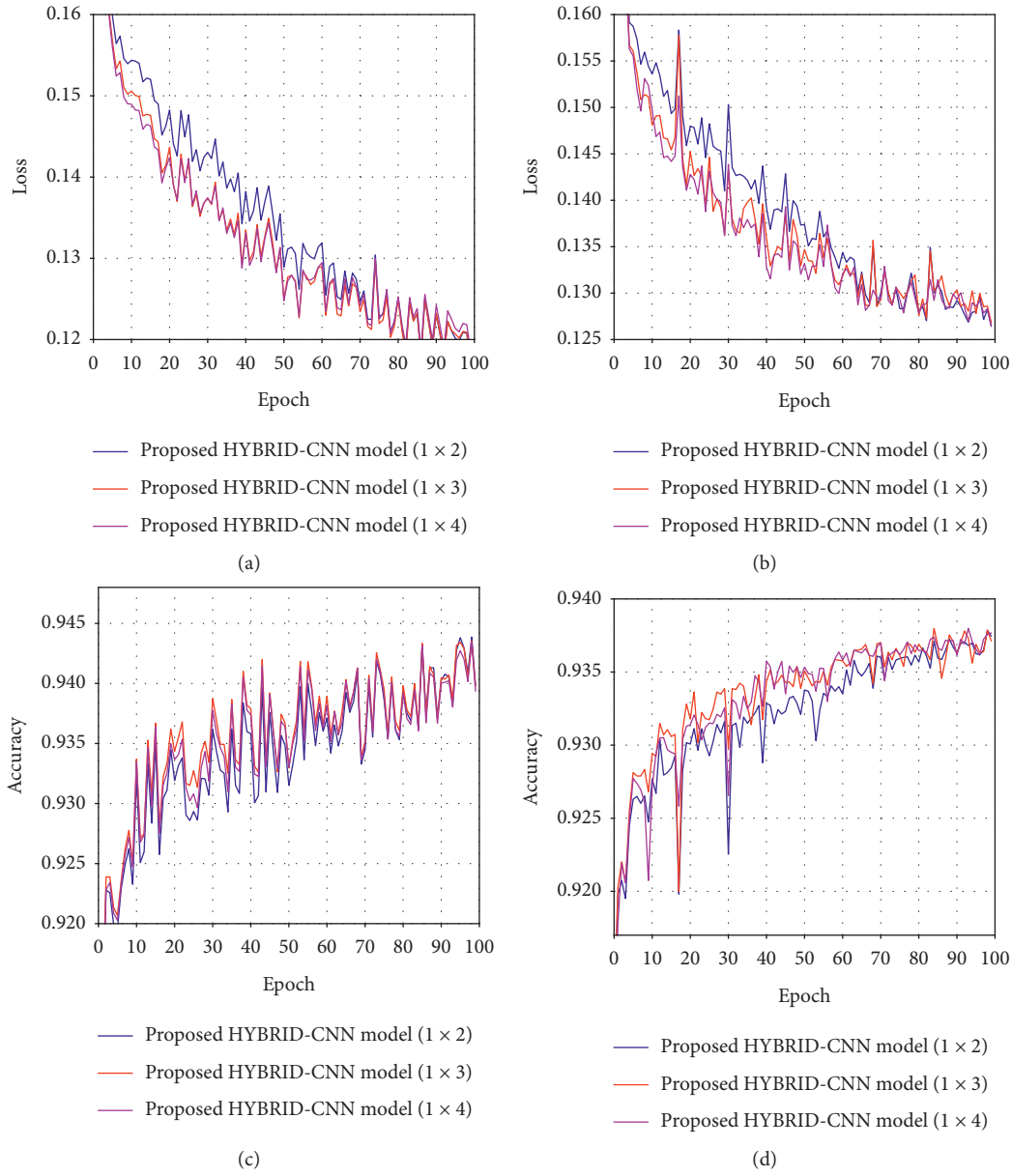


Figure 13: Parameter study of γ. (a) Training loss. (b) Validation loss. (c) Training accuracy. (d) Validation accuracy.

Detection Rate evaluation of the proposed method corresponding to successful detection of Normal, PROBE, DoS, U2R, and R2L data examples (98.21%, 93.62%, 98.89%, 92.59%, and 87.76%, respectively). Figure 15(c) shows the $F1$ score evaluation of the proposed method corresponding to Normal, PROBE, DoS, U2R, and R2L data examples (96.74%, 94.02%, 98.51%, 91.92%, and 89.37%, respectively).

It can be clearly seen from the obtained results that, for normal flow, DoS attacks and PROBE attacks have reached the maximum detection level, while detection effects for U2R and R2L attacks are slightly lower. In the real network, normal activity flow dominates while U2R and R2L are very few classes. Dataset imbalance is a quite common problem in intrusion detection. The detection model is biased towards most classes and neglects a few classes. For U2R and R2L, although the detection rate of the proposed model is lower than that of other classes, overall, it still achieves better results compared with other methods.

7. Discussion

Evaluation of the UNSW_NB15 dataset shows that our model can provide 95.64% accuracy, which is a major improvement over other deep learning methods. However, it should be noted that the results of the "R2L" and "U2L" attack classes are lower than those of other classes, because the model needs more data to learn. Unfortunately, due to the severe imbalance in the training data of such attacks, the results obtained are not stable. Hybrid detection methods are mainly combined with deep

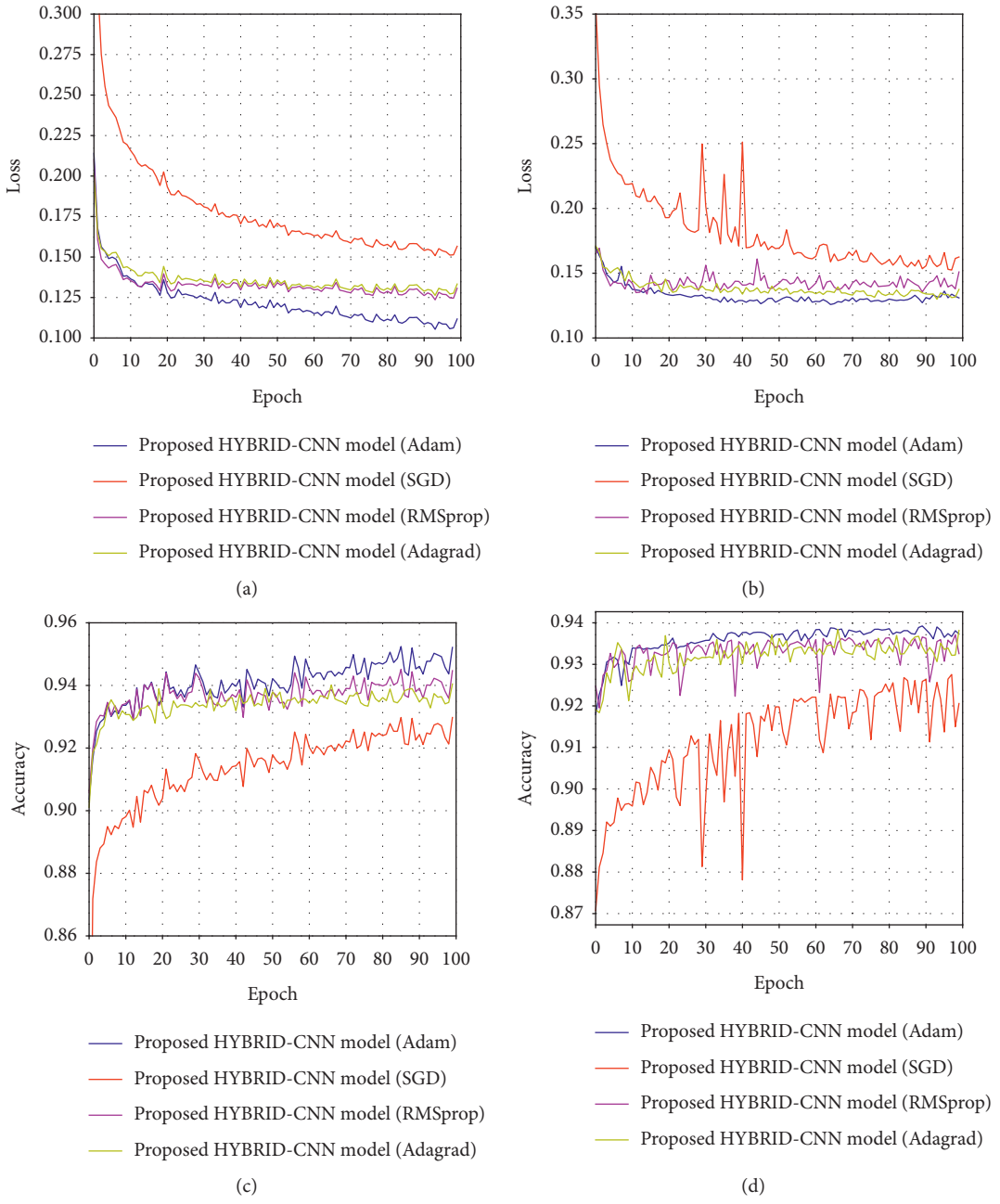


Figure 14: Parameter study of ϵ. (a) Training loss. (b) Validation loss. (c) Training accuracy. (d) Validation accuracy.

Table 6: Detailed performance (%) of HYBRID-CNN in ablation study.

Model	Acc	DR	Pre	FPR
HYBRID-CNN	**95.64**	**98.56**	**96.13**	**4.42**
(1) w/o attention	94.88	98.29	95.77	4.69
(2) w/o DNN	93.57	93.94	93.16	5.89
(3) w/o CNN	91.85	92.47	92.43	7.36

learning models, which can usually achieve higher detection accuracy. Considering the complexity of the deep learning algorithm, the algorithm can use less running time. Of course, our proposed model will spend more time on training, but using GPU acceleration can reduce training time.

Table 7: Attacks in the KDDCup 99 dataset.

Category	Training dataset	Testing dataset
DoS	Back, land, Neptune, pod, smurf, teardrop	Back, land, Neptune, pod, smurf, teardrop, mailbomb, processtable, udpstorm, apache2, worm
U2R	Buffer-overflow, loadmodule, perl, rootkit	Buffer-overflow, loadmodule, perl, rootkit, sqlattack, xterm, ps
R2L	fpt-write, guess-passwd, imap, multihop, phf, spy, warezclient, warezmaster	fpt-write, guess-passwd, imap, multihop, phf, spy, warezmaster, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named
Probe	ipsweep, nmap, portsweep, Satan	ipsweep, nmap, portsweep, Satan, mscan, saint

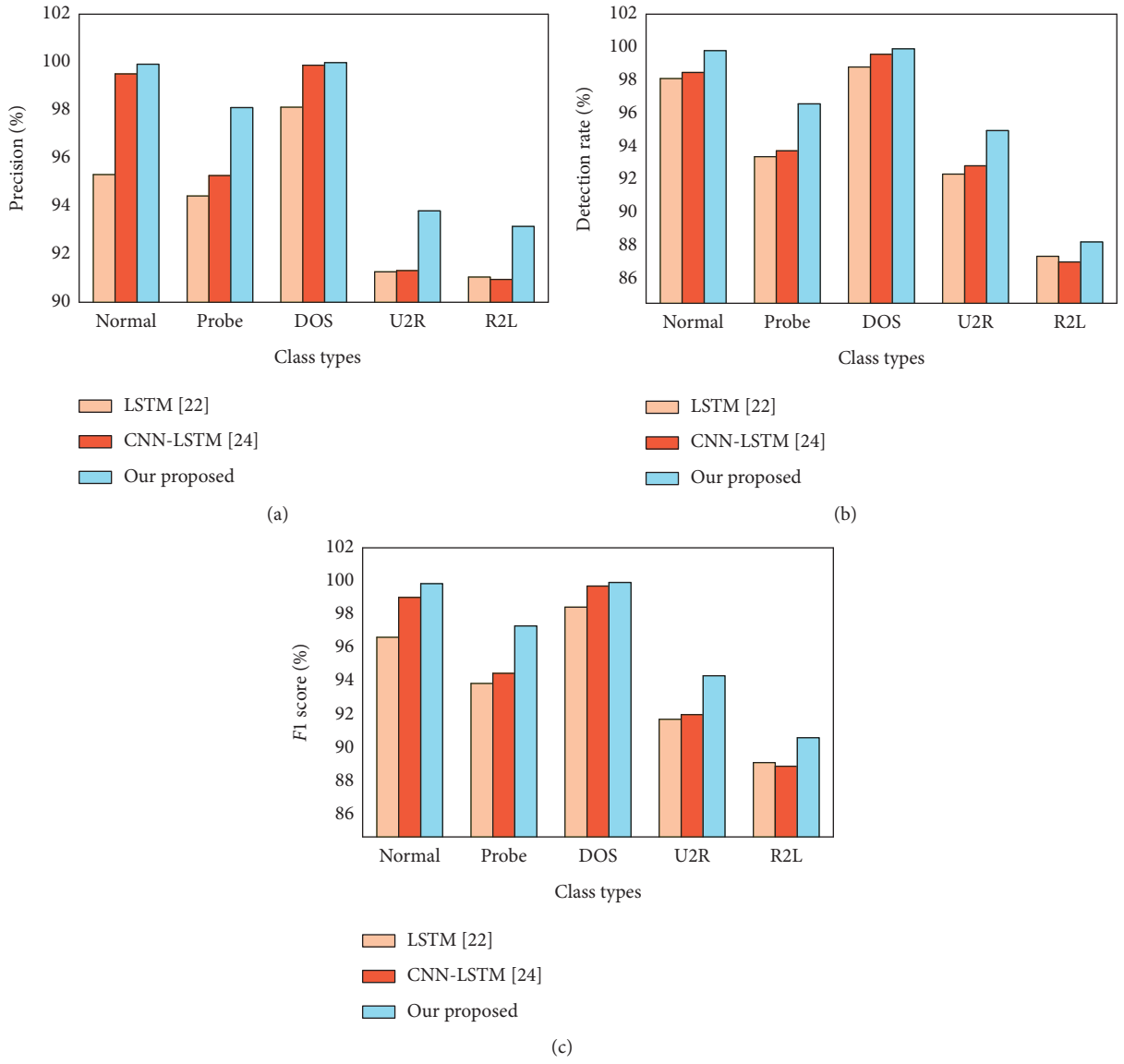


Figure 15: Experimental evaluation of the proposed method on the KDDCup 99 dataset. (a) Precision evaluation. (b) Detection Rate evaluation. (c) $F1$ score evaluation.

8. Conclusion

In this paper, we consider the problem of abnormal network flow detection of the Smart Grid integrated with the SDN. For the pursuit of accurate detection and guaranteeing network performance, we formulate a deep learning detection algorithm based on the HYBRID-CNN. In particular, our HYBRID-CNN model consists of the double channel feature extraction, key feature fusion, and classification. It gains the benefits of global memorization and local generalization brought by the DNN and the CNN, respectively. Besides, to measure the performance of the proposed algorithm, we analyze the hyperparameters of the HYBRID-CNN. Compared with other existing detection

algorithms, the experiment results show that the HYBRID-CNN has a higher detection accuracy and a lower false alarm rate.

In our future work, a problem to be solved is to improve the performance of the model through network structure optimization and automatic hyperparameter tuning. The swarm intelligent optimization algorithm, such as Particle Swarm Optimization (PSO) algorithm and Artificial Bee Colony (ABC) algorithm, can be used to automatically tune hyperparameters, which is an efficient method to improve the detection accuracy. Another problem to be solved is the unbalanced dataset. The detection accuracy of a few types of attacks needs to be improved. We hope to use data augmentation in future work to reduce the impact of the dataset.

Abbreviations

ABC: Artificial Bee Colony
CNN: Convolutional Neural Network
DNN: Deep Neural Network
FPR: False Positive Rate
IoT: Internet of Things
LSTM: Long Short-Term Memory
MLP: Multilayer Perceptron
PMU: Power Management Unit
PSO: Particle Swarm Optimization
ReLU: Rectified Linear Unit
RNN: Recurrent Neural Network
ROC: Receiver Operating Characteristic
SAE: Stacked Autoencoder
SDN: Software-Defined Network
SVM: Support Vector Machine.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61702321, 61872230, 61802249, 61802248, and U1936213).

References

[1] M. L. Tuballa and M. L. Abundo, "A review of the development of smart grid technologies," *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 710–725, 2016.

[2] X. Yao, Y. Zou, Z. Chen, M. Zhao, and Q. Liu, "Topic-based rank search with verifiable social data outsourcing," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 1–12, 2019.

[3] Y. Zou, X. Yao, Z. Chen, and M. Zhao, "Verifiable keyword-based semantic similarity search on social data outsourcing," *IEEE Access*, vol. 7, pp. 5616–5625, 2018.

[4] I. Colak, S. Sagiroglu, G. Fulli, M. Yesilbudak, and C.-F. Covrig, "A survey on the critical issues in smart grid technologies," *Renewable and Sustainable Energy Reviews*, vol. 54, pp. 396–405, 2016.

[5] A. Feghali, R. Kilany, and M. Chamoun, "SDN security problems and solutions analysis," in *Proceedings of the 2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, Paris, France, October 2015.

[6] R. Chaudhary, G. S. Aujla, S. Garg, N. Kumar, and J. J. P. C. Rodrigues, "SDN-enabled multi-attribute-based secure communication for smart grid in IoT environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2629–2640, 2018.

[7] M. C. Dacier, H. König, R. Cwalinski, F. Kargl, and S. Dietrich, "Security challenges and opportunities of software-defined networking," *IEEE Security & Privacy*, vol. 15, no. 2, pp. 96–100, 2017.

[8] R. L. Sahita, "State-transition based network intrusion detection," US20050111460A1, 2016.

[9] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[10] V. Vaidya, "Dynamic signature inspection-based network intrusion detection," US6279113B1, 2001.

[11] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[12] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, no. S1, pp. 949–961, 2017.

[13] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[14] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: mobile encrypted traffic classification using multimodal deep learning," *Computer Networks*, vol. 165, Article ID 106944, 2019.

[15] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[16] S. Aljawarneh, M. Aldwairi, and M. B. Yassein, "Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model," *Journal of Computational Science*, vol. 25, pp. 152–160, 2018.

[17] N. Ashraf, W. Ahmad, and R. Ashraf, "A comparative study of data mining algorithms for high detection rate in intrusion detection system," *Annals of Emerging Technologies in Computing (AETiC)*, vol. 2, no. 1, 2018.

[18] K. Rai, M. S. Devi, and A. Guleria, "Decision tree based algorithm for intrusion detection," *International Journal of Advanced Networking and Applications*, vol. 7, no. 4, p. 2828, 2016.

[19] R. R. Reddy, Y. Ramadevi, and K. N. Sunitha, "Effective discriminant function for intrusion detection using SVM," in *Proceedings of the 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, India, September 2016.

[20] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion

detection in software defined networking," in *Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Fez, Morocco, October 2016.

[21] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[22] Y. Xin, L. Kong, Z. Liu et al., "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[23] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proceedings of the 2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, Beijing, China, July 2017.

[24] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[25] W. Wang, Y. Sheng, J. Wang et al., "Hast-ids: learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[26] S. Demirci and S. Sagiroglu, "Software-defined networking for improving security in smart grid systems," in *Proceedings of the 2018 7th International Conference on Renewable Energy Research and Applications (ICRERA)*, Paris, France, 2018.

[27] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[28] C. Gonzalez, S. M. Charfadine, O. Flauzac, and F. Nolot, "SDN-based security framework for the iot in distributed grid," in *Proceedings of the 2016 International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, Split, Croatia, July 2016.

[29] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, https://arxiv.org/abs/1803.08375.

[30] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," 2014, https://arxiv.org/pdf/1412.6980.pdf.

[31] E. Bisong, "Introduction to scikit-learn," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 215–229, Springer, Berlin, Germany, 2019.

[32] S. Sharma, A. Soni, and V. Malviya, "Face recognition based on convolution neural network (CNN) applications in image processing: a survey," in *Proceedings of the Recent Advances in Interdisciplinary Trends in Engineering & Applications (RAITEA)*, 2019.

[33] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The Handbook of Brain Theory and Neural Networks*, vol. 3361, no. 10, 1995.

[34] A. Vaswani, N. Shazeer, N. Parmar et al., "Attention is all you need," in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pp. 5998–6008, Long Beach, CA, USA, 2017.

[35] K. Adem, S. Kiliçarslan, and O. Cömert, "Classification and diagnosis of cervical cancer with stacked autoencoder and softmax classification," *Expert Systems with Applications*, vol. 115, pp. 557–564, 2019.

[36] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, Australia, November 2015.

[37] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, 2018.

[38] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDDCup 99 data set," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, Canada, July 2009.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 1924140, 11 pages
https://doi.org/10.1155/2020/1924140

Research Article

Group Recommender Systems Based on Members' Preference for Trusted Social Networks

Xiangshi Wang, Lei Su, Qihang Zhou, and Liping Wu

School of Information and Automation, Kunming University of Science and Technology, Kunming, China

Correspondence should be addressed to Lei Su; s28341@hotmail.com

Received 3 December 2019; Revised 15 January 2020; Accepted 1 May 2020; Published 19 May 2020

Academic Editor: Yin Zhang

With the development of the Internet of Things (IoT), the group recommender system has also been extended to the field of IoT. The entities in the IoT are linked through social networks, which constitute massive amounts of data. In group activities such as group purchases and group tours, user groups often exhibit common interests and hobbies, and it is necessary to make recommendations for certain user groups. This idea constitutes the group recommender system. However, group members' preferences are not fully considered in group recommendations, and how to use trusted social networks based on their preferences remains unclear. The focus of this paper is group recommendation based on an average strategy, where group members have preferential differences and use trusted social networks to correct for their preferences. Thus, the accuracy of the group recommender system in the IoT and big data environment is improved.

1. Introduction

With the advent of the 5G era [1], which promotes the development of mobile Internet and big data, users are faced with massive amounts of data on the Internet. It is difficult for search engines to accurately obtain information resources that meet their own needs and personalized preferences. Information overload issues are increasingly prominent [2]. The recommender system [3] is considered to be effective in dealing with this problem. In recent years, research on recommender systems has developed very rapidly. There are several types of recommender systems, such as mobile recommender systems, context-aware recommender systems, and social network recommendation systems. However, these recommender systems can only be recommended for a single user. In real life, there are many situations where we interact mostly with groups, such as while watching a movie, having dinner, and planning a vacation with friends. As such, the recommender system must consider the preferences of each user in the group. This recommender system is called a group recommender system [4]. However, most of these systems deal with every individual preference in the same way, ignoring the personality of each member and the relationships among group members. People have linked entities in the IoT, which has developed rapidly through social networks. Personal social networks, where users imple-ment network interactions, are mainly divided into strong relationship networks and weak relationship networks. A strong relationship network mainly contains applications such as QQ and WeChat. Weak relationship networks include Weibo and various forums. Network members with strong connections have a high degree of relationship with each other, and they are more willing to share their views and experiences without reservation, while weak connections are the opposite. A previous study found that the higher the degree of the relationship between users is, the more the trust exists between users [5]. According to Mui, trust is the subjective expectation of one subject for its future behavior decisions based on its historical interaction experience with another subject [6].

The social network relationship of group members is an essential factor in a group recommender system. Studies have shown that users are more willing to accept

recommendations from trusted users than recommendations from anonymous users [7]. For example, users are more likely to accept recommendations from friends rather than from strangers. Group-recommended social network factors mainly include personality and trust [8, 9]. The higher users' personality leads to less effect by others. Users' trust in another user influences their preferences. The group recommendation studies bases on social networks where users may need to change their preferences to reach group consensus. In social network-based group recommendations, users may need to change their preference to reach a group consensus. At this point, the trust between the members of the group becomes the main influencing factor. However, changing your preference can easily lead to preference differences within the group. Chen e al. [10] uses the similarity of preferences and the relevance of trust, calculating the trust and influence of group members according to the similarity of group members. And finally it gets the final rating through the weighted mean fusion strategy. At the same time, not all group members have significant social networking relationships. Furthermore, according to the users' social network, closely connected users naturally come together to form a user group, and the connections between different user groups are sparse, thus forming a community structure [11]. So this community structure formed by natural social network relationship is a natural grouping method with good interpretability. However, this kind of social network relationship can only achieve a better group recommendation effect when the user relationship information is dense. In the case of sparse user data in the big data environment, the group recommendation is easy to generate a cold start and recommend the group. The improvement associated with this effect is not obvious. It is a challenge in group recommendation problem to address member preferences, random groupings, user relationship sparseness, and cold starts in a social network group recommendation in a big data environment. This paper proposes members' preference for trusted social networks. The system analyzes the group recommendations of the average strategy. When the group recommendations of the average strategy differ greatly in preferences, the trusted social network of the group members is introduced to modify the preferences in the group, to obtain better recommendation results.

The structure of this paper is as follows. Section 2 introduces related work on the social network recommendation and group recommendation methods, and Section 3 elaborates the method based on a preference for trusted social networks proposed in this paper. Section 4 introduces the experimental results. Section 5 summarizes the full text and discusses future work.

2. Related Work

2.1. Group-Recommended User Preferences. The recommended method for groups is usually to obtain the preferences of each user in the group [4]. It is generally believed that preferences are used to describe the ordering relationship of decision-makers to two or more items [12]. In the group recommendation, the user's preference acquisition methods are mainly divided into explicit preference acquisitions [13] and implicit preference acquisitions [14]. In the display of preference acquisitions, the user is required to explicitly provide preference information, which is usually defined by a rating on a given interval. MusicFX [15] requires users to rate different music style gen-res and group recommendations based on user ratings. Literature [16] requires users to rate the location, amount of food, and taste of the restaurant. In the implicit preference acquisition method, the user does not need to explicitly provide preference information, but users use historical behavior data to mine user preferences. Crossen et al. [13] learn users' musical preferences from users' music listening behavior data. Let's Browse [17] proposes to use the term frequency-inverse document frequency (TF-IDF) algorithm to learn users' preferences for news topics based on the keywords on users' homepages. As long as there is enough user behavior data, the implicit preference acquisition method can accurately extract users' preference characteristics, which is beneficial to protect the users' privacy.

2.2. Group Recommendation Preference Fusion Method. Preference fusion occurs at different stages of the group recommendation process, such that the content of the fusion is different. Literature [18] analyzed 10 preference fusion strategies in detail. The different manifestations of the four most commonly used average strategies are listed below. Assuming that each member's weight is the same, the average of the scores of all group members is used as the recommended score for the group.

Assume that each member's weight is the same in the average strategy group recommendation based on item similarity. The ratings of all group members are averaged as the recommended rating for the group:

$$\mathrm{ISpre}(G,i) = \frac{1}{|G|}\sum_{u\in G}\mathrm{ItemsPro}(u,i), \tag{1}$$

where $|G|$ indicates the size of the group; $\mathrm{ItemsPro}(u,i)$ indicates the rating of item i by user u in the recommendation method based on the user similarity; and $\mathrm{ISpre}(G,i)$ represents the final forecast score formed by the group recommendations based on the item similarity average strategy.

It is assumed that each member has the same weight in the average strategy group recommendation based on matrix factorization. The average of the ratings of all group members is used as the recommended rating for the group:

$$\mathrm{MFpre}(G,i) = \frac{1}{|G|}\sum_{u\in G}\mathrm{MatrixfPro}(u,i), \tag{2}$$

where $|G|$ indicates the size of the group; $\mathrm{MatrixfPro}(u,i)$ represents the rating of item i by user u in the recommendation method based on matrix factorization; and $\mathrm{MFpre}(G,i)$ represents the final prediction rating formed by the group recommendation based on the matrix factorization average strategy.

In the average strategy group recommendation based on the popularity of the item, it is assumed that each member has the same weight and the average of the ratings of all group members is used as the recommended rating of the group:

$$\mathrm{IPpre}(G,i) = \frac{1}{|G|}\sum_{u\in G}\mathrm{ItempPro}(u,i), \tag{3}$$

where $|G|$ indicates the size of the group; $\mathrm{ItempPro}(u,i)$ represents the rating of item i by user u in the recommendation method based on the popularity of the item; $\mathrm{IPpre}(G,i)$ represents the final predicted rating formed by the group recommendation based on the popularity of the item average strategy.

In the average strategy group recommendation based on implicit feedback, it is assumed that each member has the same weight. In addition, the average of the ratings of all group members is used as the group recommendation rating:

$$\mathrm{IFpre}(G,i) = \frac{1}{|G|}\sum_{u\in G}\mathrm{ImplicitfPro}(u,i), \tag{4}$$

where $|G|$ indicates the size of the group; $\mathrm{ImplicitfPro}(u,i)$ represents the rating of item i by user u in the recommendation method based on the implicit feedback dataset; and $\mathrm{IFpre}(G,i)$ represents the final predicted rating based on group recommendations for implicit feedback dataset average strategy.

2.3. Degree of Preference Divergence in the Recommendation of the Average Strategy Group Recommendation. Recommendations generated in the group recommendation of the average strategy [19] may cause dissatisfaction among individual group members, which is known as pain problems. To measure this "pain," the preference divergence is introduced in this paper. Because the chosen group recommendation algorithms are different, the degree of divergence in the group recommendation is different:

$$\mathrm{ISdis}(G,i) = \frac{1}{|G|}[\mathrm{ISpro}(u,i) - \mathrm{ISmean}(G,i)]^2, \tag{5}$$

where $\mathrm{ISpro}(u,i)$ indicates the rating of each user in the group calculated by the recommendation method based on the item similarity; $\mathrm{ISmean}(G,i)$ represents the average of the user ratings in the group; and $\mathrm{ISdis}(G,i)$ represents the extent to which members of the group disagree with item i in group G.

$$\mathrm{MFdis}(G,i) = \frac{1}{|G|}[\mathrm{MFpro}(u,i) - \mathrm{MFmean}(G,i)]^2. \tag{6}$$

where $\mathrm{MFpro}(u,i)$ represents the rating for each user in the group calculated by the recommendation method based matrix factorization; $\mathrm{MFmean}(G,i)$ represents the average of the user ratings in the group; and $\mathrm{MFdis}(G,i)$ represents the extent to which members of the group disagree with item i in group G:

$$\mathrm{IPdis}(G,i) = \frac{1}{|G|}[\mathrm{IPpro}(u,i) - \mathrm{IPmean}(G,i)]^2, \tag{7}$$

where $\mathrm{IPpro}(u,i)$ represents the rating of each user on the item, which can be calculated by the recommendation method of the item popularity; $\mathrm{IPmean}(G,i)$ represents the average of user ratings in the group; and $\mathrm{IPdis}(G,i)$ represents the extent to which members of the group disagree with item i in group G:

$$\mathrm{IFdis}(G,i) = \frac{1}{|G|}[\mathrm{IFbPro}(u,i) - \mathrm{IFmean}(G,i)]^2, \tag{8}$$

where $\mathrm{IFbPro}(u,i)$ represents the rating of each user on the item, which can be calculated by the recommendation method of the implicit feedback dataset; $\mathrm{IFmean}(G,i)$ represents the average of the user ratings in the group; and $\mathrm{IFdis}(G,i)$ represents the extent to which members of the group disagree with item i in group G.

3. Group Recommender System-Based Members' Preference for Trusted Social Networks

In group recommender system-based members' preference for trusted social networks, in order to alleviate the problems of data sparsity and cold starts, first, the recommendation methods based on item similarity [3], matrix factorization [20, 21], item popularity, and implicit feedback datasets [22] are adopted to form personalized recommendations for users. The purpose of personalized recommendations is to alleviate data sparsity and cold start problems in group recommendations. Through personalized recommendation, the preference scoring data of trusted members are supplemented. Then, a group recommendation based on the average strategy is adopted for the above recommendation methods. Average strategy is the most commonly used preference fusion strategy in a group recommendation system, which takes the average score of the group members as the score of the group [4]. However, recommendations generated by the average strategy may cause dissatisfaction among individual group members, namely, the so-called preference divergence problem [4]. To avoid this "preference divergence" problem, the degree of preference divergence is calculated by preference differences. Taking preference divergence as a criterion, trust-based social networks are introduced when the preference divergence of group members is greater than that of the group as a whole. In the trusted social network of group members, each group member has several trust members. When the preferences of group members are quite different, the group members can appropriately modify the ratings of group members through the preference ratings of trust members.

3.1. Calculation of Correction Factors in Group Preference

3.1.1. Calculation of Correction Factors According to the Divergence Degree. After the group recommendation system based on the average strategy [19] presented in this

paper, we take the divergence degree as a measure and obtain μ:

$$\mu = \frac{1}{|G|} F\left[\sum_{u \epsilon G}\left(\text{somepro}(u,i) - \frac{\sum_{v \epsilon G}\text{somepro}(v,i)}{|G|}\right) \geq \sum_{i=3}^{i=11}\text{somedis}(G,i)\right], \tag{9}$$

where|G| represents the number of members of the group and introduces the function $F(x)$. If the expression of x is true, it is counted as 1. The expression of x does not hold; it is recorded as 0. somepro(u,i) indicates the user u on i in the above four personalized recommendation methods. somedis(G,i) indicates the divergence degrees obtained in the group recommendations of different average strategies. μ is a measure of divergence. The larger the value of μ is, the greater the difference between the preference in the group and the preference recommended by the group of average strategy, and the more need there is to revise the preference of the group members based on the trusted members of the trusted social network. The smaller the value of μ is, the smaller the difference between the preference in the group and the preference recommended by the group of Average Strategy. When μis sufficiently small, it shows that there is no preference divergence in the group recommendation based on the average strategy. As such, it follows the group recommendation based on the average strategy.

3.1.2. Calculation of Correction Factors According to the Standard Value. Through the above analysis of the divergence degree, the divergence degree between the group members and the group recommendation of the average strategy is reduced, and the group recommendation is further optimized. To further study the correction factor μ, finding the rating of the group members for a certain item is proposed. In this rating, we use the sample standard deviation of the scored item, the median of the evaluation range, and the median of the scored item. We, respectively, use $\text{standard}_{\text{mean}}$, $\text{standard}_{\text{middle}}$, and $\text{standard}_{\text{median}}$ to express these three standard values:

$$\mu_{\text{mean}} = \frac{(\text{higher}_{\text{sdui}} - \text{fewer}_{\text{sdui}})}{w\,\text{standard}_{\text{mean}}}, \tag{10}$$

where $\text{standard}_{\text{mean}}$ means that the standard deviation of the sample is taken as the standard value. The standard deviation of the sample is calculated through the calculation of the rating. In the item being scored, the score must have a portion larger than the sample standard and a portion smaller than the sample standard deviation. The sample variance is taken as a measure. There are a large number of parts and a small number of parts that are greater than or less than the standard deviation of the sample. The large number of parts is written as $\text{higher}_{\text{sdui}}$, and the small number is written as $\text{fewer}_{\text{sdui}}$. In this experiment, in order to better combine the recommendation system of the average strategy with trusted social networks, the parameter w is taken as 2:

$$\mu_{\text{middle}} = \frac{(\text{higher}_{\text{middleui}} - \text{fewer}_{\text{middleui}})}{w\,\text{standard}_{\text{middle}}}, \tag{11}$$

where $\text{standard}_{\text{middle}}$ means that the median of the scoring range is the standard value. Through the calculation of the scoring range, the median of the scoring range is calculated. $\text{higher}_{\text{middleui}}$ indicates a set larger than the median of the scoring range. $\text{fewer}_{\text{middleui}}$ indicates a set smaller than the median of the scoring range. To better combine the recommendation system of average strategy with the trusted members in trusted social networks, w is herein taken as 2:

$$\mu_{\text{median}} = \frac{(\text{higher}_{\text{medianui}} - \text{fewer}_{\text{medianui}})}{wf\,(w\,\text{standard}_{\text{median}})}, \tag{12}$$

where $\text{standard}_{\text{median}}$ indicates that the median of the items to be scored is taken as the standard value. Through the calculation of the median score, the median of the items to be scored is calculated. $\text{higher}_{\text{medianui}}$ indicates the number of sets greater than the median of the item being scored. $\text{fewer}_{\text{medianui}}$ represents the number of sets that are less than the median of the item being scored. To better combine the recommendation system of average strategy with the preferences of trusted members in trusted social networks, w is herein taken as 2.

3.1.3. Calculation of Preference Rating for Trusted Network Members. By trusting the social network, the group members are connected with the members who trust the social network. Each user has one or more trust objects. There are two points to consider for trusting social networks: (a) the trust degree and real preference evaluation of members in trust social networks; and (b) the number of trusted social network members. The formulae are as follows:

$$\begin{aligned}
\text{ISTR}(\text{TR},i) &= \frac{\sum_{v \epsilon R} X_{u,v} \cdot \text{ISrating}(v,i)}{\sum_{v \epsilon R} G_{\text{Istrust}}},\\
\text{MFTR}(\text{TR},i) &= \frac{\sum_{v \epsilon R} X_{u,v} \cdot \text{MFrating}(v,i)}{\sum_{v \epsilon R} G_{\text{Mftrust}}},\\
\text{IPTR}(\text{TR},i) &= \frac{\sum_{v \epsilon R} X_{u,v} \cdot \text{IPrating}(v,i)}{\sum_{v \epsilon R} G_{\text{Iptrust}}},\\
\text{IFTR}(\text{TR},i) &= \frac{\sum_{v \epsilon R} X_{u,v} \cdot \text{IFrating}(v,i)}{\sum_{v \epsilon R} G_{\text{IFtrust}}},
\end{aligned} \tag{13}$$

where $X_{u,v}$ indicates the trust degree of user u to v. $X_{u,v} \in [0, 1]$, 0 means distrust and 1 means trust; ISrating (v, i) indicates the true rating of item i by user v in the

recommendation method based on item similarity in the trust network; MFrating (v, i) indicates the true rating of item i by user v in the matrix factorization based recommendation method in the trust network; IPrating (v, i) indicates the real rating of item i by user v in the recommendation method based on the popularity of the item in the trusted social network; IFrating (v, i) indicates the real rating of item i by user v in the recommendation method based on implicit feedback dataset in the trusted social network; G_{Istrust} indicates a single trusted user of a group member in a recommendation method based on item similarity; G_{Mftrust} denotes a single trusted user of a group member in a matrix factorization based recommendation method; G_{Iptrust} indicates a single trusted user of a group member in a recommendation method based on the popularity of an item; and G_{IFtrust} represents a single trusted user of a group member in a recommendation method based on an implicit feedback dataset. These parameters are introduced into the group recommendations based on group recommender systems based on members' preference for trusted in social networks. Even in group recommendation without the average strategy, data sparsity and cold start problems in group recommendations can be alleviated to some extent by trusting ISTR (TR, i), MFTR (TR, i), IPTR (TR, i), and IFTR (TR, i) in social network.

3.2. Group Recommendation Method Based on Members' Preference for Trusted Social Network Item Similarity. In the group recommendation based on the preference of members who trust social network item similarity, first, the recommendation method based on item similarity is used to generate personalized recommendations for users, and groups are randomly divided to make group recommendations based on the average strategy of item similarity. Through the group recommendation, the following methods are proposed to solve the problem of preference divergence in group recommendation of the average strategy:

$$\text{ISpre}(G,i) = \begin{cases} \left|\frac{1}{2} - \mu_i^2\right| \cdot \text{ISGP}(G,i) + \mu_i \cdot \text{ISTR}(\text{TR},i) \\ \text{ISGP}(G,i), \quad \text{if ISTR}(\text{TR},i) = \varnothing, \\ \text{ISTR}(\text{TR},i), \quad \text{if ISGP}(G,i) = \varnothing, \end{cases} \tag{14}$$

where ISGP (G, i) (item similar group prediction) indicates a group recommendation based on item similarity average strategy; ISTR (TR, i) (item similarity trust rating) indicates the rating of recommendation methods that trust the similarity of items of social network members; and μ_i indicates that the divergence degree or standard value is obtained as correction factors. The function of the correction factor is to reduce the problem of preference divergence among group members in the average strategy. ISpre (G, i) (item similarity prediction) indicates a group recommendation based on the preference of members who trust the similarity of social network items. If $\text{ISTR}(\text{TR},i) = \varnothing$, the members of the group do not trust the network. At this time, we adopt the group recommendation based on the average strategy of item similarity. If $\text{ISGP}(G,i) = \varnothing$, there is no exact group recommendation for the group recommendation of average strategy; then, a recommendation based on the trusted social network is adopted.

3.3. Group Recommendation Method Based on Members' Preference for Trusted Social Network Matrix Factorization. In the group recommendation based on the trust social network matrix factorization preference, the recommendation method based on matrix decomposition is used to generate personalized recommendations for the user. The group is randomly divided, and the group is recommended based on the matrix factorization average strategy. After the group recommendation, for the divergence problem in the group recommendation of the average strategy, the following methods are proposed:

$$\text{MFpre}(G,i) = \begin{cases} \left|\frac{1}{2} - \mu_i^2\right| \cdot \text{MFGP}(G,i) + \mu_i \cdot \text{MFTR}(\text{TR},i) \\ \text{MFGP}(G,i), \quad \text{if MFTR}(\text{TR},i) = \varnothing, \\ \text{MFTR}(\text{TR},i), \quad \text{if MFGP}(G,i) = \varnothing, \end{cases} \tag{15}$$

where MFGP (G, i) (matrix factorization group prediction) indicates a group recommendation based on the matrix factorization average strategy; MFTR (TR, i) (matrix factorization trust rating) indicates the score of the recommendation method based on matrix factorization of trusted social network member preferences; and μ_i indicates that the divergence degree or the standard value is obtained as a correction factor. The function of the correction factor is to correct the preference differences among the group members in the group recommendation of the average strategy. MFpre (G, i) (matrix factorization prediction) indicates a group recommendation based on trusted social network matrix factorization member preferences. If $\text{MFTR}(\text{TR},i) = \varnothing$, the group members do not trust the social network. In this case, the average strategy group recommendation based on matrix factorization is adopted. If $\text{MFGP}(G,i) = \varnothing$, no group recommendation is formed in the random group. At this time, the recommendation based on the trusted social network is adopted.

3.4. Group Recommendation Method Based on Members' Preference for Trusted Social Network Item Popularity. In group recommendations based on trusted social network item popularity membership preferences, personalized recommendations are generated for users by the recommendation method based on the popularity of the item. Then, the groups are randomly divided and group recommendations are made based on the average strategy of the popularity of the item. After the group recommendation, the following methods are proposed to solve the problem of preference divergence in the group recommendation based

on the average strategy of trusted social network item popularity:

$$\mathrm{IPpre}(G,i)=\begin{cases}\left|\frac{1}{2}-\mu_i^2\right|\cdot \mathrm{IPGP}(G,i)+\mu_i\cdot \mathrm{IPTR}(\mathrm{TR},i)\\ \mathrm{IPGP}(G,i), \quad \text{if } \mathrm{IPTR}(\mathrm{TR},i)=\varnothing,\\ \mathrm{IPTR}(\mathrm{TR},i), \quad \text{if } \mathrm{IPGP}(G,i)=\varnothing.\end{cases} \tag{16}$$

where IPGP (G, i) (item popular group prediction) indicates the average strategy group recommendation based on the popularity of the item; IPTR (TR, i) (item popular trust rating) indicates a recommended method for trusting the popularity of the social networking item; μ_i is the divergence degree or standard value obtained and used as a correction factor to correct the preference divergence problem among group members in group recommendation based on the average strategy of item popularity; and IPpre (G, i) (item popular prediction) indicates a group recommendation based on member preference of trusted social network item popularity. If $\mathrm{IPTR}(\mathrm{TR},i)=\varnothing$, the members of the group do not trust the social network, and the average strategy group recommendation based on the popularity of the item is adopted. This means that no group recommendation based on the average strategy of the popularity of the item has been formed; thus, the recommendation based on the trusted social network will be adopted.

3.5. Group Recommendation Method Based on Implicit Feedback of the Dataset of Members' Preference by Trusted Social Networks. In group recommendations based on implicit feedback of member preferences by trusted social networks, a recommendation method based on implicit feedback dataset is used to generate personalized recommendations for users, groups are randomly divided, and group recommendations based on the average strategy of implicit feedback datasets are carried out on the groups. After group recommendation, the following methods are proposed to solve the preference divergence problem in the group recommendation based on the implicit feedback dataset average strategy:

$$\mathrm{IFpre}(G,i)=\begin{cases}\left|\frac{1}{2}-\mu_i^2\right|\cdot \mathrm{IFGP}(G,i)+\mu_i\cdot \mathrm{IFTR}(\mathrm{TR},i)\\ \mathrm{IFGP}(G,i), \quad \text{if } \mathrm{IFTR}(\mathrm{TR},i)=\varnothing,\\ \mathrm{IFTR}(\mathrm{TR},i), \quad \text{if } \mathrm{IFGP}(G,i)=\varnothing,\end{cases} \tag{17}$$

where IFGP (G, i) (implicit feedback group prediction) indicates the group recommendation of average strategy based on the implicit feedback dataset; IFTR (TR, i) (implicit feedback trust rating) represents recommendations based on implicit feedback datasets of trusted social network members; μ_i is used to find the divergence degree and standard value as correction factors to correct the preference divergence problem in group recommendation based on the implicit feedback dataset average strategy; and IFpre (G, i) (implicit feedback prediction) indicates a group recommendation based on the implicit feedback of dataset member preferences by trusted social networks. If $\mathrm{IFTR}(\mathrm{TR},i)=\varnothing$, the members of the group do not trust the social network, and the group recommendation based on the implicit feedback dataset average strategy is adopted. If $\mathrm{IFGP}(G,i)=\varnothing$, the average strategy group recommendation based on the implicit feedback dataset has not formed an effective group recommendation; at this time, the recommendation based on the implicit feedback dataset and the trusted social network is adopted.

3.6. Algorithm Based on Members' Preference for Trusted Social Networks. The evaluation dataset is divided into a training set and a test set. After preprocessing the data, all trust-based social networks will be used. At the same time, the following group member preference algorithm is proposed to reduce the preference divergence problem in the group recommendation of average strategy. The basic idea behind the experiment is as follows: using recommendation methods based on item similarity, matrix factorization, item popularity, and implicit feedback dataset, personalized recommendations are generated for users. Groups are divided randomly and μ is calculated according to the divergence and standard value. Through trusting social networks, trust members are found for team members, after which group recommendations are generated and the MAE is calculated by correcting for the preference differences of trusted social networks. Through trusted social networks, we can find trusted members for team members and can generate group recommendations through correcting the differences in the preferences of trusted social networks. Finally, we calculate MAE. The algorithm is named GRIMPFTSN (group recommendation involves members preference for trusted social networks) Algorithm 1 was proposed and verified by experiments.

4. Experiment and Analysis

4.1. Experimental Dataset. The dataset uses the FilmTrust dataset (https://www.librec.net/datasets/filmtrust.zip) [23]. The dataset includes 1508 users, 2071 movies, and 35497 reviews. The density of data is 1.14%. The dataset is preprocessed as follows: users are randomly divided into training sets and test sets according to a certain proportion, 80% for training sets and 20% for test sets. Assuming that a user has 10 scores on items, we select 8 of the users as training sets and 2 as test sets. To divide the data, we select users who score items more than 5 times. Through data preprocessing, it can be obtained that there are 1478 training set users and 1420 testing set users. Further, 607 users in the trusted social network have trusted objects through the trusted social network, and 41% of users in the training set will use trusted social networks. In this paper, a randomized group approach is used for experimental research. The goal is to alleviate the computational cost of the experiment and

Input: Ratingmatrix, Trustmatrix
Output: MAE
Initial: Pro (u, i), TR (u, i) by RS methods
Repeat
  Gpro (G, i) by GRS for the average strategy
  If TR $(u, i) = =$ null **then**
    Pre $(G, i) =$ Gpro (G, i)
  else
    Gpro $(G, i) = =$ null,
    Pre $(G, i) =$ TR (u, i)
  else
    Pre $(G, i) = |(1/2) - \mu^2| + \mu_i \cdot \mathrm{TR}(\mathrm{TR}, i)$
    μ_i from dis and standard value
  End if
Until MAE

Algorithm 1: GRIMPFTSN.

to verify the validity of the group preference group recommendations of the social networks that trust the randomly generated team.

4.2. Evaluation Method MAE. To evaluate the effectiveness of the group recommendation based on trusted social network preferences, the offline data evaluation method, mean absolute error (MAE) [24], is used to measure the prediction accuracy:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{N} (\mathrm{pre}(G, i) - \mathrm{real}(G, i))}{N}, \tag{18}$$

where pre (G, i) represents a group recommendation based on trusted social network preferences; real (G, i) represents the real recommendation in the group; and N represents the number of items in the group recommendation.

4.3. Group Recommendation Experiment Based on Members' Preference for Trusted Social Network Item Similarity. Experimental description: in Table 1, IS_average represents the group recommendation based on item similarity average strategy (GRBOISAS) and μ_{ISdis} indicates the method calculated according to the degree of divergence, named IStrust_dis. Other methods are not applicable to group recommendation based on members' preference for trusted social network item similarity.

In this experiment, we randomly selected 100 times according to the number of the group. The group size is 3–11 people. Each group sampling is relatively independent. We obtained MAE through the experiment. The MAE shows the difference between group recommendation based on member preference for trusted social network item popularity and group recommendations based on the average strategy. As shown in this experiment, we made 100 random selections according to the number in the group. Each group sampling is relatively independent, including 3–11 people. We obtained the MAE through the experiment. MAE is the mean absolute error. The mean absolute error shows the difference between group recommendations based on the preference of similar members of trusted social network items and group recommendations based on the average strategy of the trusted social network. The results are shown in Figure 1.

Table 1: IS experimental description.

Name	Description
IS_average	GRBOISAS
IStrust_dis	μ_{ISdis}

It is apparent in Figure 1 that under the average strategy based on item similarity, the MAE of IS_average decreases as the group increases in size. In the group recommendation based on the item similarity average strategy, there is no accumulation of the user's historical behavior, and it is not a personalized recommendation. Thus, the differences in the recommendation results of all users are very small. When the difference in the users' recommendation results is small, the difference in the group recommendation based on the average strategy formed by the user is also small. In this experiment, the group recommendation based on the average strategy of item similarity is also verified to have a good recommendation effect by randomly dividing the groups. IStrust_dis corrects the preferences of group members in the group recommendation based on the average strategy of item similarity. By revising the preferences of group members, the effect of group recommendation based on the item similarity average strategy is further improved.

4.4. Group Recommendation Experiment Based on Members' Preference for Trusted Social Network Matrix Factorization. The variables in Table 2 are as follows: MF_average is the group recommendation based on the matrix factorization average strategy (GRBOMFAS).μ_{MFdis} denotes a method for calculating the correction factor according to the divergence degree, which is named MFtrust_ dis;μ_{MFssd} means that in the calculation of the standard value, the sample standard variance is used to calculate the correction factor, named MFtrust_sd; $\mu_{\mathrm{MFsmiddle}}$ means that the standard variance

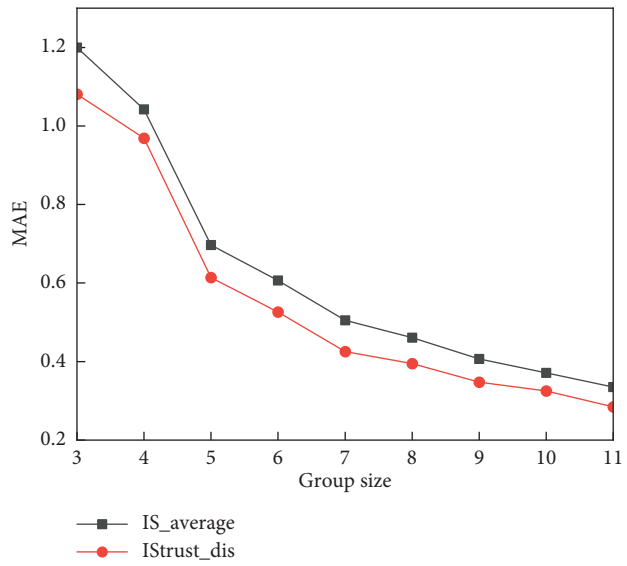


Figure 1: MAE comparison of different group sizes based on the item similarity average strategy.

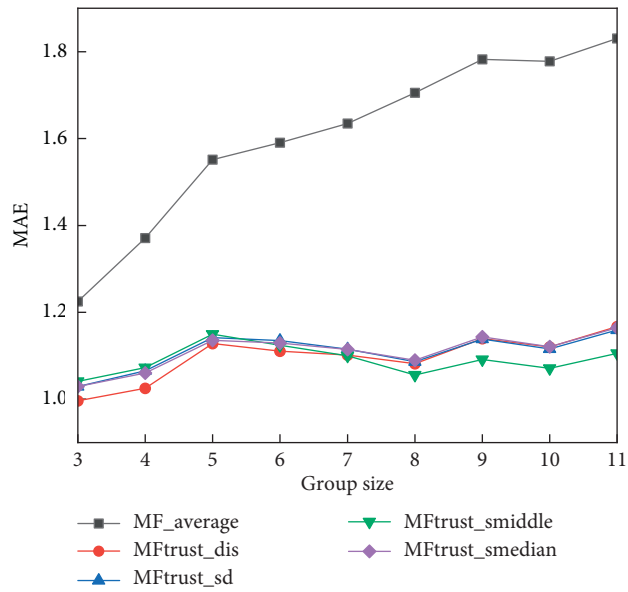


Figure 2: MAE comparison of different group sizes based on the matrix factorization average strategy.

Table 2: MF experimental description.

Name	Description
MF_average	GRBOMFAS
MFtrust_dis	μ_{MFdis}
MFtrust_sd	μ_{MFssd}
MFtrust_smiddle	$\mu_{\text{MFsmiddle}}$
MFtrust_smedian	$\mu_{\text{MFsmedian}}$

calculation method of samples is adopted in the calculation of the standard value, named MFtrust_smiddle; and $\mu_{\text{MFsmedian}}$ means that in the calculation of the standard value, the calculation method of the median value of the evaluated item is adopted and is named MFtrust_smedian.

In this experiment, we made 100 random selections according to the number of the group, and the group size was 3–11. Each group sampling is relatively independent, and Figure 2 is obtained through experiments. MAE is the mean absolute error. In this experiment, the root mean square error is used to represent the performance difference between the group recommendations based on trusted social network matrix factorization member preferences and group recommendations based on the matrix factorization average strategy.

As shown in Figure 2, when the group size is 3–6, the experimental effect of MFtrust_dis is obviously better than other experimental effects. The performance of the group recommendation based on the average strategy of matrix factorization is greatly improved. When the group size is 7–11, the MAE obtained by the MFtrust_smiddle method is lower, and the group recommendation effect is better.

4.5. Group Recommendation Based on Members' Preference for Trusted Social Network Item Popularity. The variables given in Table 3 are defined as follows: IP_average is a group recommendation based on item popularity average strategy (GRBOIPAS); μ_{IPdis} refers to the method of calculating the correction factor according to the divergence degree, named IPtrust_dis; μ_{IPssd} means that in the calculation of the standard value, the sample standard variance is used to calculate the correction factor, named IPtrust_sd; $\mu_{\text{IPsmiddle}}$ means that in the calculation of the standard value, the middle value of the evaluable range is adopted as the calculation method of the standard value, named IPtrust_smiddle; and $\mu_{\text{IPsmedian}}$ indicates that in the calculation of the standard value, the calculation method of the median value of the evaluated article is adopted, named IPtrust_ smedian.

In this experiment, we made 100 random selections according to the number of groups, with a group size of 3–11, for which each group sampling was relatively independent. Figure 3 is obtained through the experiment. In this experiment, MAE represents the effect of group recommendation based on the preference of members of the popularity of the item and the group recommendation based on the average popularity strategy of the item.

As shown in Figure 3, the experimental effect of IPtrust_dis is obviously better than other experimental effects when the group size is 3-4. When the group size is 5–7, the MAE of the IPtrust_smedian method is lower than those of other methods, and the group recommendation effect is better. When the group size is 8–11, the MAE of IPtrust_smiddle is lower than those of other methods, and the group recommendation effect is better.

4.6. Group Recommendation Based on Members' Preference for Trusted Social Network Implicit Feedback Datasets. The variables in Table 4 are defined as follows: IF_average is a group recommendation based on an implicit feedback dataset average strategy (GRBOAIFDAS); μ_{IFdis} indicates the method of calculating the correction factor according to the

Table 3: IP experimental description.

Name	Description
IP_average	GRBOIPAS
IPtrust_dis	μ_{IPdis}
IPtrust_sd	μ_{IPssd}
IPtrust_smiddle	$\mu_{\text{IPsmiddle}}$
IPtrust_smedian	$\mu_{\text{IPsmedian}}$

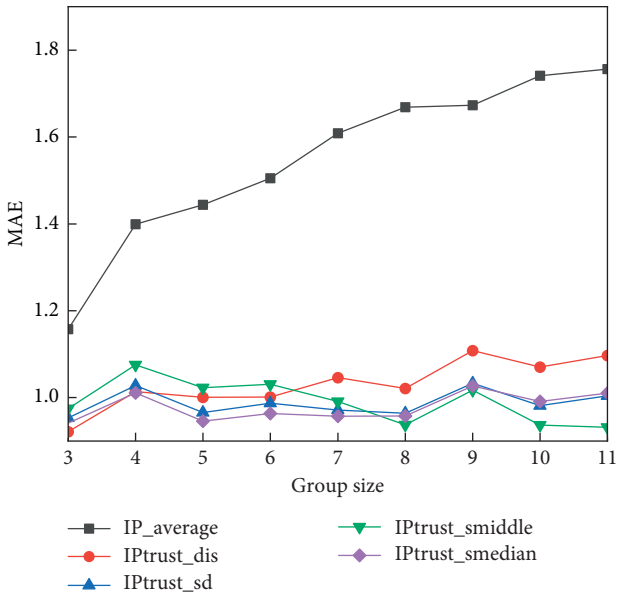


Figure 3: MAE comparison of different group sizes based on the item popularity average strategy.

Table 4: IF experimental description.

Name	Description
IF_average	GRBOIPAS
IFtrust_dis	μ_{IFdis}
IFtrust_sd	μ_{IFssd}
IFtrust_smiddle	$\mu_{\text{IFsmiddle}}$
IFtrust_smedian	$\mu_{\text{IFsmedian}}$

divergence degree, named IFtrust_dis; μ_{IFssd} means that in the calculation of the standard value, the sample standard variance is used to calculate the correction factor, named IFtrust_sd; μ_{IFmiddle} means that in the calculation of the standard value, the middle value of the evaluable range is used as the calculation method of the standard value, named IFtrust_smiddle; and $\mu_{\text{IF median}}$ means that in the calculation of the standard value, the method of calculating the median of the evaluated items is adopted, named IFtrust_smedian.

In this experiment, we made 100 random selections according to the number of groups, with a group size of 3–11. Each group sampling was relatively independent. Figure 4 is obtained through the experiment. MAE represents mean absolute error, and MAE represents group recommendation based on the trusted social network implicit feedback dataset member preference and group

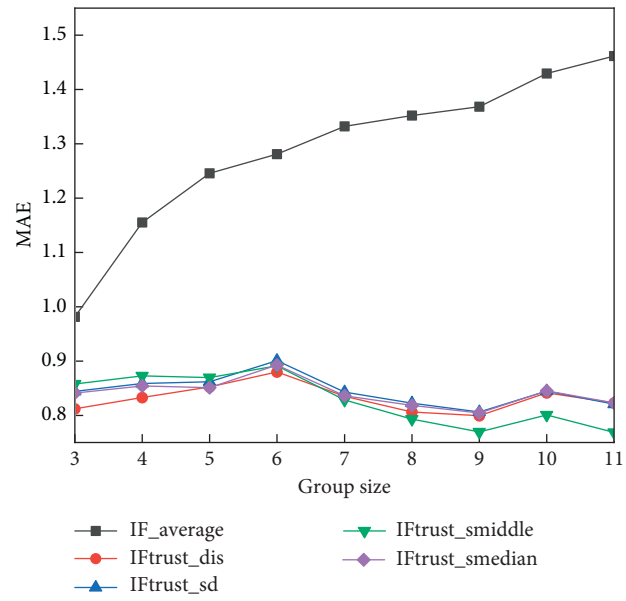


Figure 4: MAE comparison of different group sizes based on the implicit feedback dataset average strategy.

Table 5: UR experimental description.

Name	Description
IS_rate	SNUBOIS
MF_rate	SNUBOMF
IP_rate	SNUBOTPOI
IF_rate	SNUBOIFD

recommendation effect based on the implicit feedback dataset average strategy.

From Figure 4, it is apparent that the experimental effect of IFtrust_dis is obviously better than other experimental effects when the group size is 3–6. When the group size is 7–11, the MAE of IFtrust_smiddle is lower than that of other methods.

4.7. Utilization Rate of Trusted Social Networks. The variables in Table 5 are defined as follows: IS_rate is the social network utilization based on item similarity (SNUBOIS); MF_rate is the social network utilization based on matrix factorization (SNUBOMF); IP_rate is the social network utilization based on item popularity (SNUBOIP); and IF_rate is the social network utilization based on implicit feedback dataset (SNUBOIFD).

In this experiment, we made 100 random selections according to the number of groups, with a group size of 3–11. Each group sampling was relatively independent. Figure 5 is obtained through the experiment. Percent indicates the utilization rate of social networks, and group size indicates the size of groups.

Figure 5 shows that the utilization rate of trusted social networks is higher when the group is larger, showing a relatively stable trend. Through the above experiments, it is apparent that when the group is bigger, the higher the social network utilization rate. The utilization rate of social

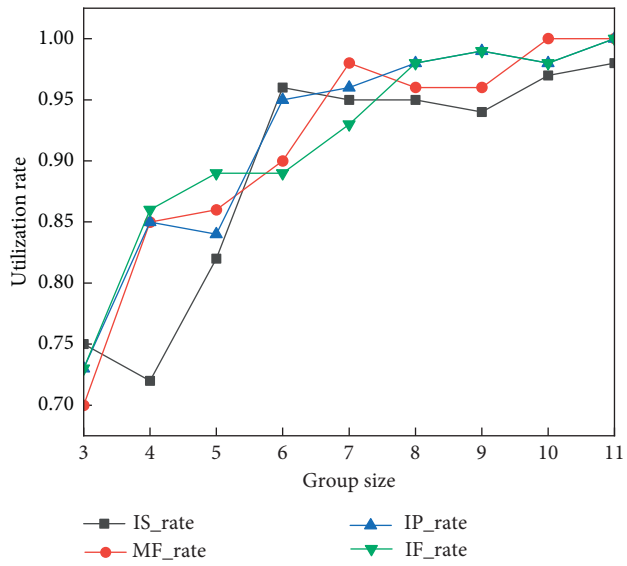


Figure 5: Social network utilization rate.

networks shows that the utilization rate of social networks recommended by groups based on the preference of similar members of projects is relatively low. Thus, the average strategy group recommendation approach is more suitable.

5. Conclusion

This paper introduces group recommendation based on members' preferences of trusted social networks. Compared with previous experiments, this paper proposes that group recommendation based on trusted social network preferences greatly reduced the MAE and produced an optimal model in the face of different recommendation methods. In this paper, the preferences of group members are modified to reduce the divergence of group members and further optimize the group recommendation of average strategy. By comparing four types of group recommendations based on trusted social networks, it was found that group recommendations based on the preferences of similar members of trusted social networks produced superior results. In this experiment, the item-based recommendation system that has been used in previous experiments is further verified to be suitable for recommendation fusion. Therefore, it is also discovered that the recommendation effect of the members' preference group based on trusted social network project similarity is also optimal. By comparing four types of group recommendation systems based on trusted social network member preferences, it is found that the recommendation system based on trusted social network item popularity has the best interpretability. The effect of this experiment has been greatly improved. And different types of group recommendation systems based on social networks are also compared. However, for each group recommendation system based on trusted social network member preferences, there is no in-depth study, nor does it exactly indicate which exact factors of social network affect the results of group recommendation. It needs to be extended to use in different datasets.

Data Availability

The FilmTrust dataset used to support the study is available at https://www.librec.net/datasets/filmtrust.zip [23].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work was supported by the National Science Foundation of China under Grant no. 61365010.

References

[1] K. Zheng, L. Zhao, J. Mei, M. Dohler, W. Xiang, and Y. Peng, "10 Gb/s hetsnets with millimeter-wave communications: access and networking—challenges and protocols," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 222–231, 2015.

[2] H.-L. Xu, X. Wu, X.-D. Li, and B.-P. Yan, "Comparison study of Internet recommendation system," *Journal of Software*, vol. 20, no. 2, pp. 350–362, 2009.

[3] P. Resnick and H. R. Varian, "Recommender systems," *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.

[4] I. Garcia, S. Pajares, L. Sebastia, and E. Onaindia, "Preference elicitation techniques for group recommender systems," *Information Sciences*, vol. 189, pp. 155–175, 2012.

[5] W. Wang and I. Benbasat, "Attributions of trust in decision support technologies: a study of recommendation agents for e-commerce," *Journal of Management Information Systems*, vol. 24, no. 4, pp. 249–273, 2008.

[6] L. Mui, *Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks*, Massachusetts Institute of Technology, Cambridge, MA, USA, 2002.

[7] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl, "PolyLens: a recommender system for groups of users," in *ECSCW*, pp. 199–218, Springer, Berlin, Germany, 2001.

[8] H. D. Ambulkar and A. Pathan, "Recommender system challenges and methodologies in social network: survey," *International Journal of Science and Research (IJSR)*, vol. 4, no. 11, pp. 286–289, 2015.

[9] M. Tavakolifard and K. Almeroth, "Social computing: an intersection of recommender systems, trust/reputation systems, and social networks," *IEEE Network*, vol. 26, no. 4, pp. 53–58, 2012.

[10] Y.-L. Chen, L.-C. Cheng, and C.-N. Chuang, "A group recommendation system with consideration of interactions among group members," *Expert Systems with Applications*, vol. 34, no. 3, pp. 2082–2090, 2008.

[11] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[12] L.-C. Wang, X.-W. Meng, and Y.-J. Zhang, "Context-aware recommender systems," *Journal of Software*, vol. 23, no. 1, pp. 1–20, 2012.

[13] A. Crossen, B. Jay, and K. J. Hammond, "Flytrap: intelligent group music recommendation," in *Proceedings of the 7th International Conference on Intelligent User Interfaces*, pp. 184-185, San Francisco, CA, USA, January 2002.

[14] Y. Dunham, A. S. Baron, and M. R. Banaji, "The development of implicit intergroup cognition," *Trends in Cognitive Sciences*, vol. 12, no. 7, pp. 248–253, 2008.

[15] J. F. McCarthy and T. D. Anagnost, "MUSICFX: an arbiter of group preferences," in *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, Menlo Park, CA, USA, 1998.

[16] D. Ribeiro Soriano, "Customers' expectations factors in restaurants," *International Journal of Quality & Reliability Management*, vol. 19, no. 8-9, pp. 1055–1067, 2002.

[17] H. Lieberman, N. W. Van Dyke, and A. S. Vivacqua, "Let's browse: a collaborative web browsing agent," *IUI*, vol. 99, pp. 65–68, 1999.

[18] S. Amer-Yahia, S. B. Roy, A. Chawlat, G. Das, and C. Yu, "Group recommendation," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 754–765, 2009.

[19] S. Basu Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu, "Space efficiency in group recommendation," *The VLDB Journal*, vol. 19, no. 6, pp. 877–900, 2010.

[20] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[21] D. Bokde, S. Girase, and D. Mukhopadhyay, "Matrix factorization model in collaborative filtering algorithms: a survey," *Procedia Computer Science*, vol. 49, pp. 136–146, 2015.

[22] G. Li and W. Ou, "Pairwise probabilistic matrix factorization for implicit feedback collaborative filtering," *Neurocomputing*, vol. 204, pp. 17–25, 2016.

[23] G. Guo, J. Zhang, and N. Yorke-Smith, "A novel bayesian similarity measure for recommender systems," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pp. 2619–2625, Beijing, China, August 2013.

[24] J. Castro, J. Lu, G. Zhang, Y. Dong, and L. Martinez, "Opinion dynamics-based group recommender systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 12, pp. 2394–2406, 2018.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 7913061, 12 pages
https://doi.org/10.1155/2020/7913061

Research Article

Design and Analysis of a Novel Chaos-Based Image Encryption Algorithm via Switch Control Mechanism

Shenyong Xiao, ZhiJun Yu, and YaShuang Deng

The School of Information and Safety Engineering, Zhongnan University of Economics and Law, Wuhan 430073, China

Correspondence should be addressed to YaShuang Deng; dys0377@163.com

Received 18 December 2019; Accepted 13 February 2020; Published 16 March 2020

Guest Editor: Farrukh Aslam Khan

Chaos has been widely used in image encryption due to its rich properties. However, it remains an irreconcilable contradiction for security and implementation efficiency for image encryption schemes. In this paper, a novel chaos-based image encryption scheme has been proposed, where the Lorenz chaotic system is applied to generate pseudorandom sequences with good randomness, and a random switch control mechanism is introduced to ensure the security of the encryption scheme. Experimental results demonstrate the effectiveness and superiority of the algorithm.

1. Introduction

Mass data transmission on various communication networks has led to a security risk in multimedia data. Digital images have become an important expression in the network of information transmission due to its intuitive and vivid attribute; meanwhile, a great deal of researches on image processing has emerged [1–4]. The increasingly rampant network crime makes the digital image security particularly important. In the past few decades, many encryption algorithms such as Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA), and Advanced Encryption Standard (AES) have been put forward. However, those methods are more suitable for text encryption rather than image encryption owing to the special properties of images including large amount of data, high redundancy, and strong correlation between pixels. Chaotic system has features such as sensitivity to initial conditions and control parameters, dense periodic points, and topological transitivity, which make it especially suitable for image encryption. A chaotic image encryption algorithm was firstly proposed in 1989 [5]. Ever since then, a variety of chaos-based image encryption algorithms have been put forward [6–16].

Early chaotic image encryption schemes were mostly based on simple low-dimensional chaotic systems, such as Logistic chaotic map [17], tent chaotic map [18], cat chaotic map [11], Baker chaotic map [19], and so on [20]. Specific can elaborate as: El Assad and Farajallah [11] proposed an image encryption system based on 2D cat map, where a diffusion layer along with a bit-permutation layer was contained. Li et al. [18] proposed a novel image encryption scheme based on the tent map, which had been proved to perform well. Zhang and Wang [21] proposed a new multiple-image encryption algorithm based on the mixed image element and piecewise linear chaotic map, which is a quite fast way to encrypt, and the like. However, some existing schemes have been revealed to be security risk owing to the simple structure of the applied chaotic maps [22]. Then, researchers tried to design image encryption schemes by using various deformation or combinations of these well-known chaotic maps and other mathematical manipulation [23–26], such as the composition of logistic and tent map [27], the Logistic-Sine-coupling map [28], and baker map and logistic map [29]. These solutions have enhanced the security of algorithms and were effective to some certain extent. With the further study of chaotic systems, more and more image encryption schemes based on higher-dimensional chaotic systems, especially for hyperchaos and spatiotemporal chaos, emerge gradually [13, 30–32]. Actually, most of these schemes have a high level of security, as opposed to a high implementation cost. Moreover, some new technologies have been

introduced to the design and security analysis of image encryption schemes such as neural network [33], DNA coding [34], genetic recombination [31], compressed sensing [35], and machine leaning [36]. In general, there exists an irreconcilable contradiction between the security and implementation complexity of cryptographic algorithms.

Switch control technology has been addressed in many fields such as biological and medical systems [37], electric power systems [38], and others. It is worth mentioning that switch control can be used to realize the chaotification of given dynamical systems or make an original simple system become complex, etc. Motivated by the above discussion, in this paper, we introduce the switch control mechanism into the chaos-based image encryption scheme, where the required pseudorandom numbers for encryption are still generated by chaos, and substitutions of rows or columns of image in the permutation of plain images are determined by the designed random switch control mechanism. Moreover, the confusion of the permuted image is completed to ensure the security of the whole image encryption. Finally, experimental results are carried out to show the effect of the scheme. Given that the process of image encryption can hide any information about the original image as much as possible, the whole process can be regarded as the process of decreasing entropy. Then, performance comparisons with some existing image schemes are carried out by using information entropy along with other indicators to show the superiority of the proposed image encryption algorithm.

The rest of this paper is organized as follows. Some preliminaries are given in Section 2. In Section 3, we present a novel image encryption scheme via switch control mechanism. In Section 4, some numerical examples are given to illustrate the validity and superiority of image encryption algorithm. Section 5 concludes this paper.

2. Pseudorandom Number Generator Based on Lorenz Chaotic System

For a given plain image, the whole encryption requires a series of random numbers to produce secret image. Therein, this paper exploits the effectiveness of the Lorenz chaotic system to generate pseudorandom numbers. The Lorenz system is formally defined as

$$\begin{cases} \dot{x} = a(y - x), \\ \dot{y} = cx - y - xz, \\ \dot{z} = xy - bz, \end{cases} \tag{1}$$

where a, b, and c are system parameters. It is well known that the system has a strange chaotic attractor over the parameters $a = 10$, $b = 8/3$, and $c = 28$, as depicted in Figure 1.

Motivated by the idea in [39] that proper stretch transformation along with modular operation can make the chaotic system generate pseudorandom numbers with good randomness, two new pseudorandom number generators are designed as

$$S1_i = \mathrm{mod}\left(\mathrm{round}\left(\left(x_i + y_i\right) * 10^{12}, 2\right)\right); \quad i = 1, 2, \ldots, \tag{2}$$

$$S2_i = \mathrm{mod}\left(\mathrm{round}\left(z_i * 10^{12}\right), 256\right); \quad i = 1, 2, \ldots, \tag{3}$$

where $\{x_i\}$, $\{y_i\}$, and $\{z_i\}$ are sample sequences of the Lorenz chaotic system over the sampling interval $T = 0.1$. Actually, the chaotic signal can be completely described by its samples for this sampling interval [39].

The standard NIST SP800-22 test is applied to evaluate the performance of two pseudorandom number generators, and the test results are summarized in Table 1. As shown in the table, two pseudorandom number generators have passed all the tests, which indicate that both of them have good randomness, thus can be used in the next image encryption process.

3. Proposed Image Encryption Scheme

This section presents the proposed image encryption in detail. The encryption scheme consists of two main parts: image confusion and image diffusion. The confusion process generates scrambled images from a series of plain images by relocating image pixels, where it is determined by a switch control rule. The diffusion increases the security of permutated images by using mixed operation on relocated pixel values of images.

3.1. Image Encryption Process. Let I be an image with the size of $M \times N$, which can be turned into the vector form as follows:

$$I = \{I_1, I_2, \ldots, I_{MN}\}, \tag{4}$$

where I_i denotes the image pixel at i-th position, for $i = 1, 2, \ldots, M \cdot N$.

3.1.1. Image Confusion via Switch Control Mechanism. To perform the image confusion, the proposed method exploits the Lorenz chaotic system to generate two chaotic sequences denoted as follows:

$$\begin{aligned} R &= \{R_1, R_2, \ldots, R_M\}, \\ L &= \{L_1, L_2, \ldots, L_N\}. \end{aligned} \tag{5}$$

The proposed method then sort two these chaotic sequences R and L to yield the following sets:

$$\begin{aligned} SR &= \{SR_1, SR_2, \ldots, SR_M\}, \\ SL &= \{SL_1, SL_2, \ldots, SL_N\}. \end{aligned} \tag{6}$$

Finally, marking the positions of each point in the sequences SR and SL in the original sequences R and L, we can get two random permutations denoted as follows:

$$\begin{aligned} TR &= \{TR_1, TR_2, \ldots, TR_M\}, \\ TL &= \{TL_1, TL_2, \ldots, TL_N\}. \end{aligned} \tag{7}$$

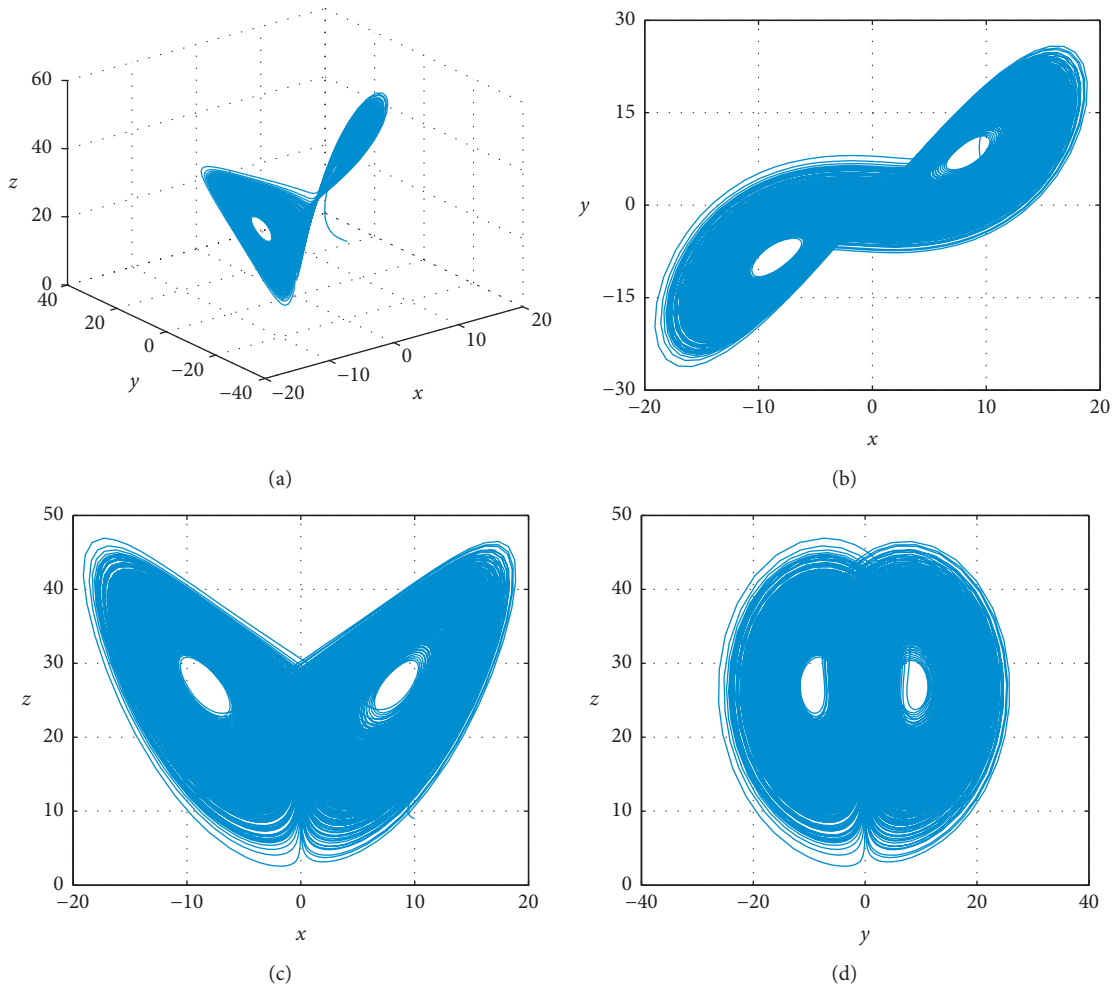


Figure 1: Lorenz chaotic attractor. (a) All directions. (b) x-y direction. (c) x-z direction. (d) y-z direction.

Table 1: NIST SP800-22 test results of two generators.

Test name	S1		S2	
	p value	Results	p value	Results
Frequency	0.115026	SUCCESS	0.852445	SUCCESS
Block frequency	0.479345	SUCCESS	0.335341	SUCCESS
Cumulative sums	0.133011	SUCCESS	0.739284	SUCCESS
Runs	0.628042	SUCCESS	0.648365	SUCCESS
Longest runs of ones	0.746332	SUCCESS	0.269936	SUCCESS
Rank	0.955981	SUCCESS	0.057146	SUCCESS
FFT	0.713570	SUCCESS	0.818546	SUCCESS
Overlapping template matching	0.360195	SUCCESS	0.434233	SUCCESS
Universal statistical	0.689639	SUCCESS	0.693656	SUCCESS
Random excursions	0.364557	SUCCESS	0.504450	SUCCESS
Random excursions variant	0.490487	SUCCESS	0.490322	SUCCESS
Serial	0.880692	SUCCESS	0.157533	SUCCESS
Nonperiodic template	0.541996	SUCCESS	0.474985	SUCCESS
Linear complexity	0.519593	SUCCESS	0.736412	SUCCESS
Apen	0.909288	SUCCESS	0.401933	SUCCESS

To increase the randomness of rearrangement of image pixels, a switch control mechanism is injected into the image confusion step, which can be used to determine whether a row or column transformation will be performed on plain images. The switch control mechanism can be designed as follows:

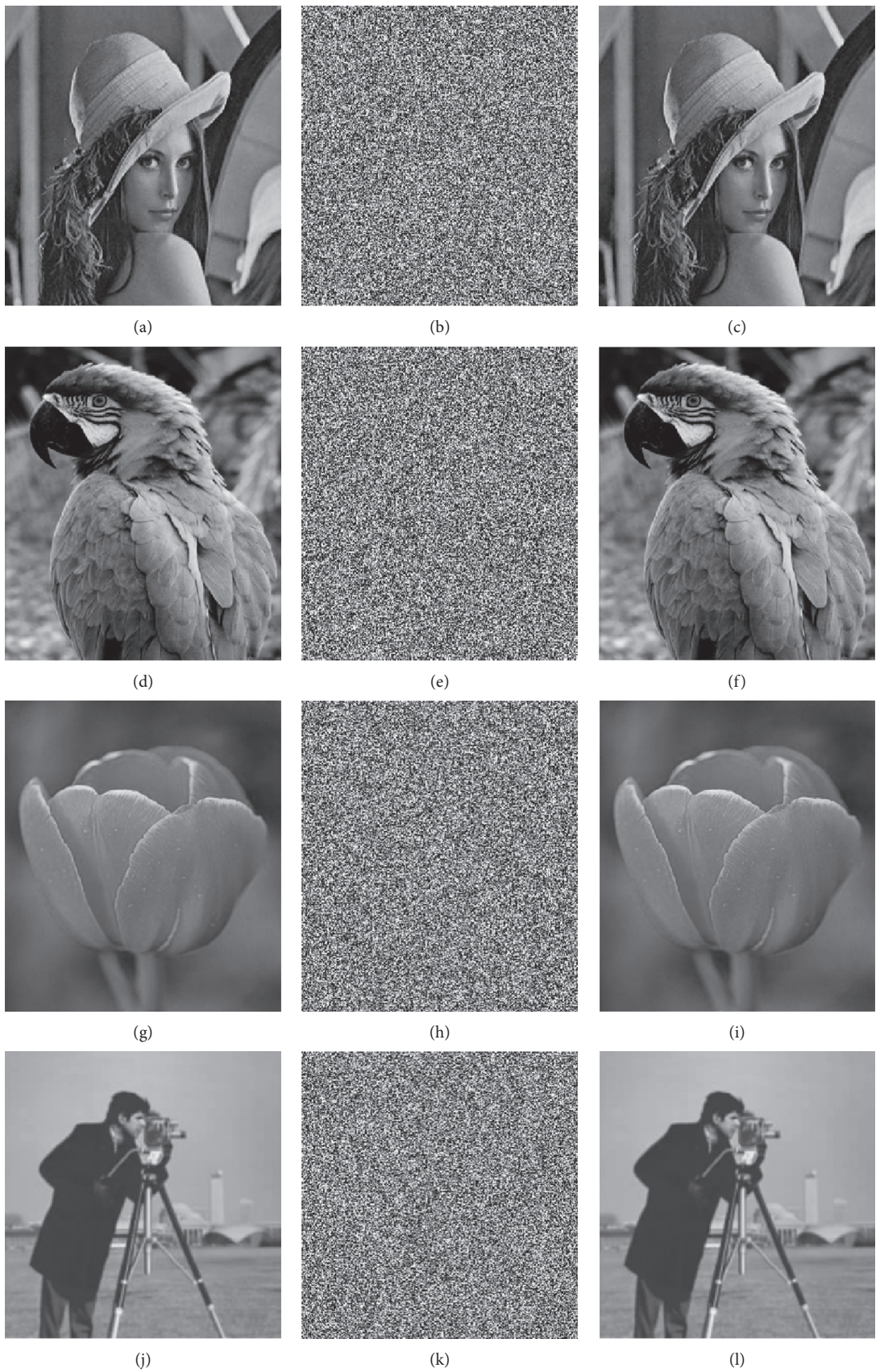


Figure 2: Encryption and decryption of images. (a) Plaintext of Lena. (b) Encryption of Lena. (c) Decrypion of Lena. (d) Plaintext of bird. (e) Encryption of bird. (f) Decryption of bird. (g) Plaintext of flower. (h) Encryption of flower. (i) Decryption of flower. (j) Plaintext of photographer. (k) Encryption of photographer. (l) Decryption of photographer.

$$\overline{I} = \begin{cases} f_1(I), & if\ S1_i = 0, \\ f_2(I), & if\ S1_i = 1, \end{cases} \tag{8}$$

where I is the plain image, $\overline{I}$ is the scrambled image, and $S1$ is the pseudorandom number generator expressed by equation (2), whose randomness determines that the proposed switch control law is also random. $f1$ and $f2$ represent row and column transformation described as follows.

Row transformation (f_1): rearrange the positions of the rows in I according to the order of TR, e.g., move the TR_1 row to the first row, the TR_2 row to the second row, . . ., the TR_M row to the M th row.

Column transformation (f_2): rearrange the positions of the columns in I according to the order of TL, e.g., move the TL_1 column to the first column, the TL_2 column to the second column, . . ., the TL_N column to the Nth column.

Based on the above switch control rule, the confusion process can be described as follows:

Step 1: select the first K points from $S1$ and let $\theta = S1_i$ $i = 1, 2, \ldots, K$, where K represents the maximum value of M and N.

Step 2: relocate the pixels of image I according to the switch control rule represented by equation (8) along with the pseudorandom sequence $S1$. That is, if $\theta = 0$, a row transformation will be performed on the plain image; otherwise, column transformation works.

Step 3: a new matrix $\overline{I}$ can be got after MN times transformations. If the pixel in the image has not been permutated completely, discard the first MN points in $S1$, and repeat Step 1-Step 2 until the results perform well.

3.1.2. Image Diffusion Process. To further increase the security level of the proposed image encryption, the scrambled image $I1'$ can be made more confusing via the pseudorandom numbers $S2$ designed previously, where the whole image can be processed as a sequence with length of MN.

The image diffusion can simply perform by utilizing the following computation:

$$C_i = \left(\mathrm{mod}\left(C_{i-1} + \overline{I}_i + \overline{I}_{i-1}\right), 256\right) \oplus S2, \tag{9}$$

where C_i is the current ciphered value, C_{i-1} is the previous ciphered value, $\overline{I}_i$ is the current scrambled image value, and $\overline{I}_{i-1}$ is the previous scrambled image value, and $S2$ has been calculated by equation (3). Set the initial value $C_0 = 0$.

3.2. The Decryption Process. The inverse process of image diffusion aims to recover back the diffused image into its original value, which can be viewed as the reverse of the encryption part. The same keys as used in the encryption process are introduced into the Lorenz chaotic system to obtain three output sequences $\{x_i\}$, $\{y_i\}$, and $\{z_i\}$, $i = 1, 2, \ldots, N$. Then, the same method employed above to calculate $S1$ and $S2$ was used.

The formula of the decryption is given as

$$\overline{I}_i = \mathrm{mod}\left(C_i \oplus S2 - C_{i-1} - \overline{I}_{i-1}, 256\right), \tag{10}$$

where C_i is the current ciphered value, C_{i-1} is the previous ciphered value, $\overline{I}_i$ is the current scrambled image value, and $\overline{I}_{i-1}$ is the previous scrambled image value, and $S2$ has been calculated by equation (3) . Without loss of generality, set initial value $\overline{I}_0 = 0$.

The confusion of decryption: extract K points from $S1$ to get θ. If $\theta = 0$, the corresponding part will be performed by row transformation; else, the column transformation works. And it is determined by the random permutation TR and TL how the image transforms. It is worth noting that the round of permutation part used here should be the same as the one designed in the encryption process. In this way, the plain image I can be recovered.

4. Experimental Results and Performance Analysis

Series of images are chosen here to verify the performance of the proposed image encryption algorithm, where all image sizes are normalized to 256×256 for convenience. Setting the parameters and initial values of the Lorenz chaotic system as $a = 10$, $b = 8/3$, $c = 28$, $x_0 = 10$, $y_0 = 5$, and $z_0 = 9$, we carry out the encryption scheme.

We first invest the performance of the proposed encryption algorithm for different images. It can be shown from Figure 2 that the algorithm destroys the obvious pattern of the plain image and makes the ciphered image display a space filling with a noise-like pattern. The shuffling process of pixels of the image hides the information of the original plain image and makes the ciphered image seem random to the intruder. Thus, the encryption scheme is effective.

Then, we analyze the security of the proposed encryption scheme. Generally, a good encryption scheme not only can hide any information of the plain image but also can resist some attacks. Some commonly used test indicators have been applied to analyse the security of the proposed image encryption scheme, which include key space and key sensitivity analysis, histogram analysis, NPCR (number of pixel change rate) and UACI (unified average changing intensity) analysis, entropy analysis, and correlation analysis.

4.1. Key Space and Sensitivity Analysis. The size of key space is an important indicator to measure the ability of resistance to exhaustion attack. In general, the smaller the key space, the more vulnerable the scheme is to attack. From the cryptographic point of view, the size of key space should be no smaller than 2^{128} to make brute force attack ineffective. Given that the secret keys include the initial value x_0, y_0, and z_0 and system parameters a, b, and c, the size of the key space can reach 10^{6L} with computing precision 10^L. In the case, the key space is far more than $10^{84}(\approx 2^{128})$ if the precision $L \geq 14$. Therefore, the key space is large enough to resist the exhaustion attack.

A good image encryption scheme should also be sensitive to tiny changes of keys, which means any tiny changes of the keys can induce huge changes of the encrypted images.

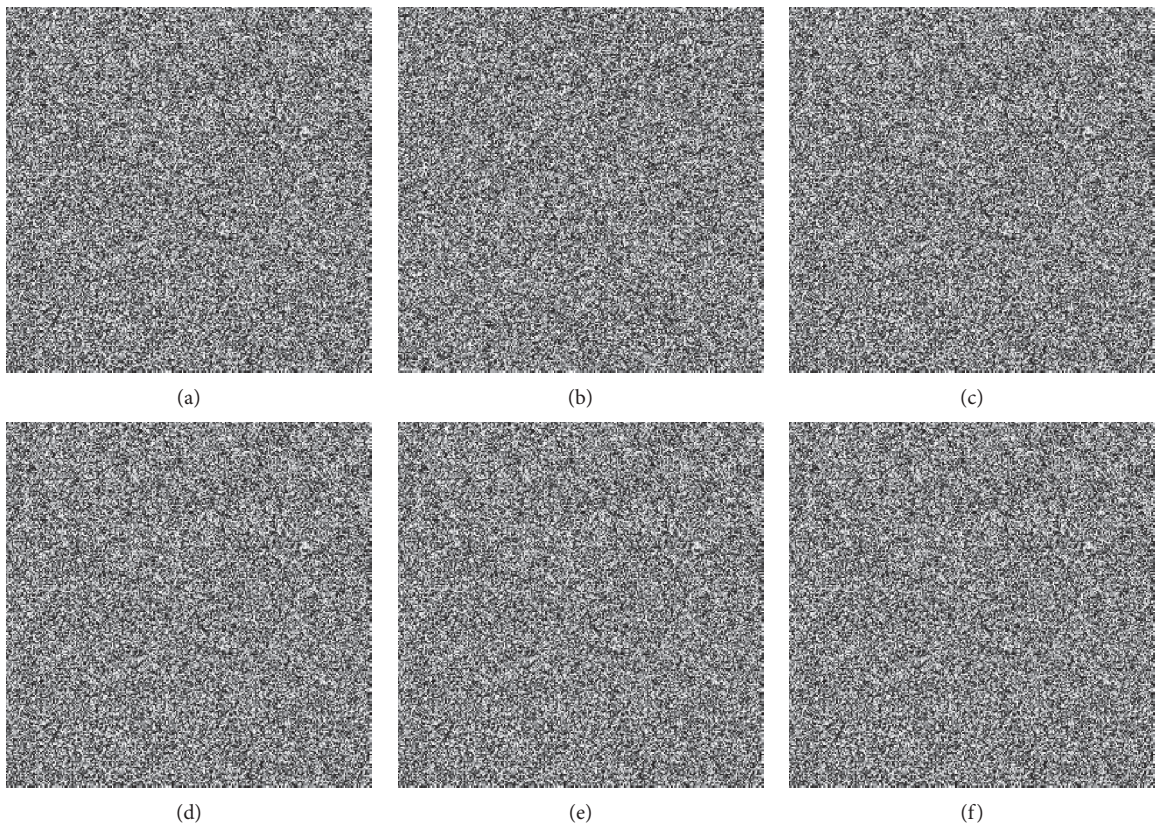


Figure 3: Key Sensitivity Analysis. (a) Change a restoring diagram. (b) Change b restoring diagram. (c) Change c restoring diagram. (d) Change x_0 restoring diagram. (e) Change y_0 restoring diagram. (f) Change z_0 restoring diagram.

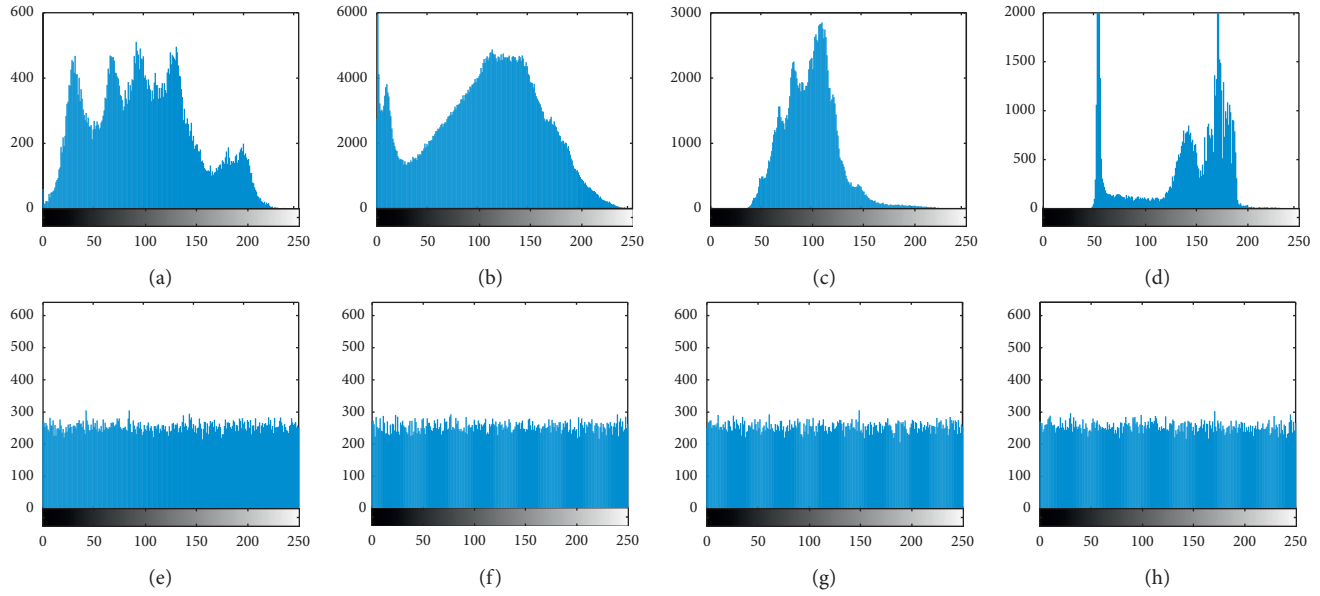


Figure 4: Histograms of plain and ciphered images. (a–d) Histograms of Lena, bird, flower, and photographer. (e–h) Histograms of ciphered Lena, bird, flower, and photographer.

In this way, the attacker cannot decode the original image by using the keys similar to the real ones.

Without loss of generality, we choose randomly system parameters and initial values to carry out the process of encryption and decryption and observe the influence of tiny changes on the decryption. For each secret key, suppose that the last one bit is changed in it and other keys are unchanged and then investigate if the original images can be restored

Table 2: NPCR and UACI of different positions.

Position	(1, 1)	(64, 64)	(128, 128)	(1, 256)	(256, 1)	(256, 256)
NPCR (%)	99.6063	99.6048	99.6201	99.5880	99.6048	99.5926
UACI (%)	33.4621	33.2419	33.4538	33.3603	33.3034	33.3788

Table 3: NPCR and UACI of proposed algorithm.

Image	Lena	Bird	Flower	Photographer
NPCR (%)	99.6063	99.6002	99.5834	99.6086
UACI (%)	33.4621	33.4665	33.4696	33.4434

Table 4: Comparison of NPCR and UACI with other methods.

Scheme	Proposed	Ref. [26]	Ref. [34]	Ref. [12]	Ref. [16]
NPCR (%)	99.6063	99.62000	99.6173	99.6552	99.6094
UACI (%)	33.4621	33.46000	29.5664	33.4846	28.6181

Table 5: Correlation coefficients of images.

Image	Plain image			Ciphered image		
	Horizontal	Vertical	Diagonal	Horizontal	Vertical	Diagonal
Lena	0.9728	0.9281	0.9050	−0.0011	0.0014	0.0005
Bird	0.9687	0.9596	0.9298	0.0004	−0.0020	0.0028
Flower	0.9694	0.9528	0.9301	0.0026	−0.0041	−0.0023
Photographer	0.9626	0.9231	0.9496	0.0029	−0.0024	−0.0008

using the changed key. Setting the parameters $a = 10$, $b = 8/3$, $c = 28$, $x_0 = 10$, $y_0 = 5$, and $z_0 = 9$, the decrypted images using the same settings can be shown in Figure 2 in front. Meanwhile, the decrypted image with the keys a, b, c, x_0, y_0, and z_0 changed to 0.00000000001 is shown in Figure 3. To be exact, the key a is changed to 10.00000000001. Similar conclusions can be got for other keys. As shown in Figure 3, the encrypted image cannot be cracked by using a similar key ($x_0 = 10.00000000000001$). Hence, the algorithm is sensitive to tiny changes of keys.

4.2. Histogram Analysis. The histogram of an image is an important statistical property which can reflect the relation between gray level and its corresponding frequency. For a good image encryption algorithm, its encrypted image should have a histogram with uniform distribution to hide the statistical characteristic. The images before and after encryption are shown in Figure 4. It can be seen from Figures 4(a)–4(d) that the frequency distributions of given images are not uniform for different gray levels, which makes attackers often easily get information from them. It can be found from Figures 4(e)–4(h) that the frequency distribution becomes quite uniform after encryption, which indicates that the statistical characteristic has been hidden and will not leak any information of the plain images, thus enhance the security of the images.

4.3. NPCR and UACI Analysis. NPCR and UACI are two measures to examine the performance of an image encryption algorithm to resist differential attack. Actually, NPCR depicts the number of pixels change rate while one pixels of plain image changed, while UACI stands for the average intensity of difference between the plain image and the ciphered image. Two these indicators can be defined as follows:

$$\begin{aligned} \mathrm{NPCR} &= \frac{\sum_{i,j} D(i,j)}{W \times H} \times 100\%, \\ \mathrm{UACI} &= \frac{1}{W \times H} \sum_{i,j} \frac{\left| C_1(i,j) - C_2(i,j) \right|}{255} \times 100\%, \end{aligned} \tag{11}$$

where W and H are the width and height of C_1 or C_2. C_1 or C_2 is the encrypted image before and after one pixel of the plain image is changed. $D(i, j)$ can be defined as follows: if $C_1 \neq C_2$, $D(i, j) = 1$; else, $D(i, j) = 0$.

And the expectation of the NPCR and UACI with 8 bits representation can be described as [40]

$$\begin{aligned} \mathrm{NPCR}_E &= \left(1 - 2^{-n}\right) \times 100\% \\ \mathrm{UACI}_E &= \frac{1}{2^{2n}} \frac{\sum_{i=1}^{2^n - 1} i(i+1)}{2^n - 1} \times 100\%, \end{aligned} \tag{12}$$

where n denotes the digit. It can be calculated that NPCR_E and UACI_E are close to 99.6094070 and 33.4635070 for 8 bits digits.

We invest the NPCR and UACI results of the encrypted Lena images for different positions and different images separately, and the results are shown in Tables 2 and 3, respectively. As shown in two these tables, the values of NPCR are close to the ideal value, which means the encryption scheme is very sensitive to small changes in the plain image. With respect to UACI, the values for different

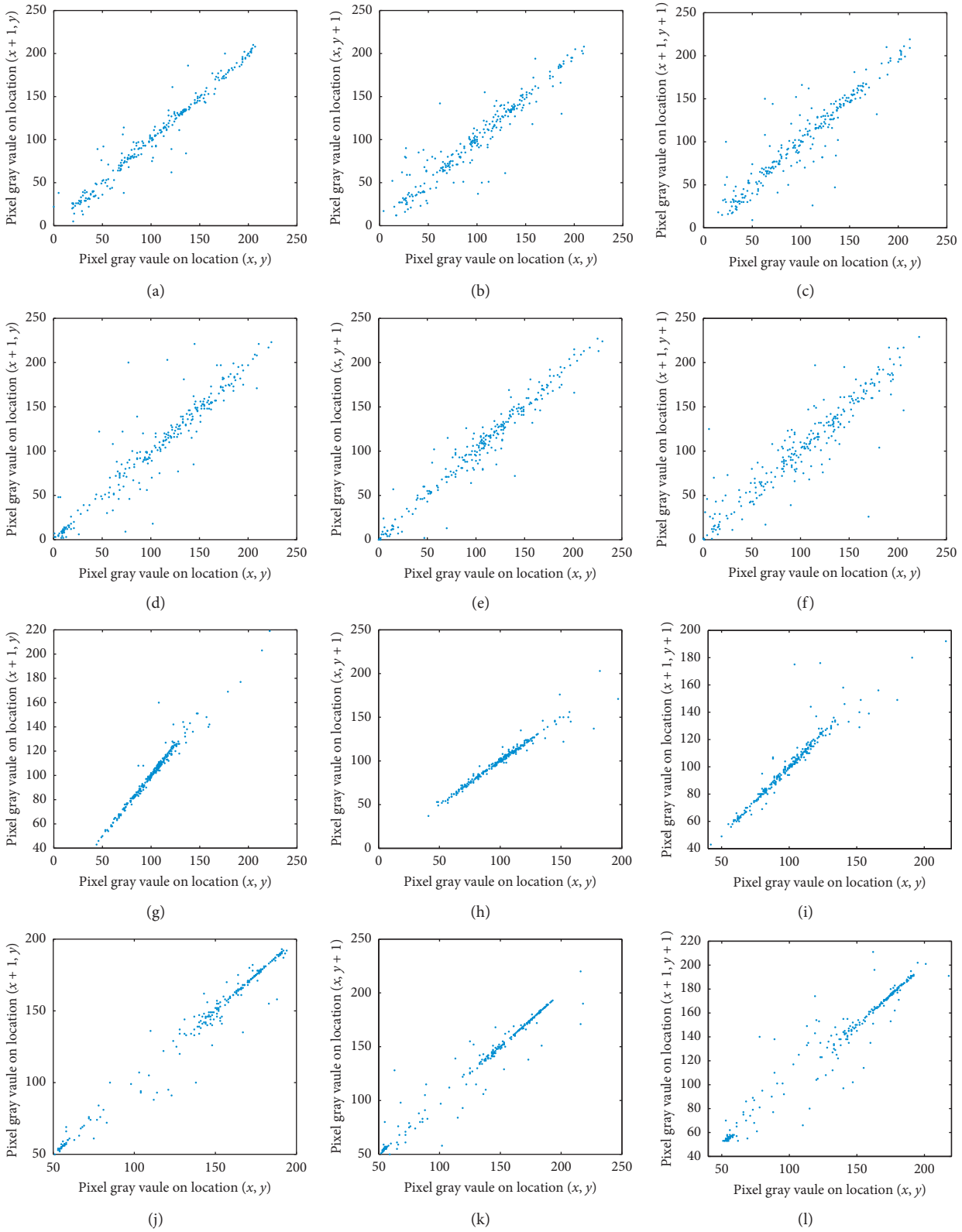


Figure 5: Correlations of plain images in horizontal, vertical, and diagonal adjacent pixels. (a–c) Lena. (d–f) Bird. (g–i) Flower. (j–l) Photographer.

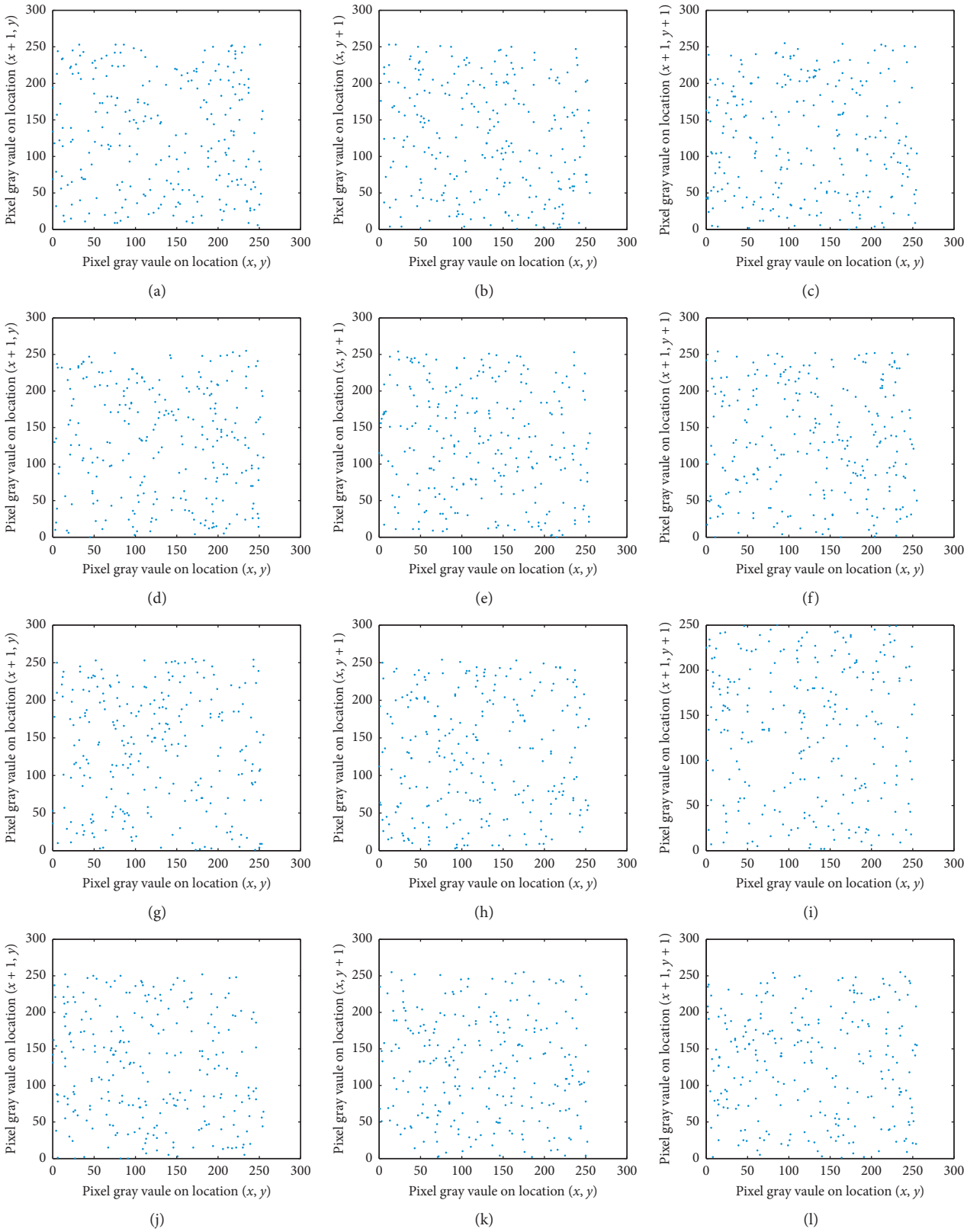


Figure 6: Correlations of ciphered images in horizontal, vertical, and diagonal adjacent pixels. (a–c) Lena. (d–f) Bird. (g–i) Flower. (j–l) Photographer.

Table 6: Comparison of correlation coefficients with other methods.

Scheme	Lena	Proposed	Ref. [26]	Ref. [34]	Ref. [12]	Ref. [16]
Horizontal	0.9728	−0.0011	−0.0285	0.0027	−0.0038	−0.0245
Vertical	0.9281	0.0014	0.0014	0.0005	−0.0026	−0.0226
Diagonal	0.9050	0.0005	0.0013	−0.0045	0.0017	−0.0193

Table 7: Information entropy of images.

Image	Lena	Bird	Flower	Photographer
Plain	7.5545	7.6515	6.6792	6.5786
Ciphered	7.9974	7.9973	7.9970	7.9971

Table 8: Approximate entropy of images.

Image	Lena	Bird	Flower	Photographer
Plain	0.6434	0.7828	0.4613	0.2723
Ciphered	2.1733	2.1785	2.1815	2.1734

Table 9: Comparison of information entropy with other methods.

Scheme	Plain image	Encrypted image	Ref. [26]	Ref. [34]	Ref. [12]	Ref. [16]
Entropy	7.5545	7.9997	7.9993	7.9972	7.9874	7.9975

images are also close to the ideal value, which indicates that the rate of influence due to one pixel change is very large. In this way, the algorithm has strong ability to resist differential attacks.

To further show the superiority of the algorithm, comparisons with other existing schemes have been made here, as depicted in Table 4. It is clear from the analysis result that the algorithm has higher ability to resist differential attack.

4.4. Correlation Analysis. The correlation between image pixels is an important indicator to measure whether the ciphered image can resist the chosen-plaintext attack. The correlation between adjacent pixels can be characterized by correlation coefficients, which can be defined as follows:

$$r_{x,y} = \frac{\mathrm{Cov}(x, y)}{\sqrt{D(x) * D(y)}}, \tag{13}$$

where x and y are the grayscale values of two adjacent pixels in the given image. $D(x)$ and $D(y)$ are the variance of x and y, respectively. $\mathrm{Cov}(x, y)$ shows the covariance of x and y.

To measure the correlation of adjacent pixels, we first select 2000 pairs of adjacent pixels (in vertical, horizontal, and diagonal directions) randomly from plain images and ciphered images, respectively, and calculate their correlation coefficients. The mean value of the correlation coefficients for different images is shown in Table 5. Obviously, the correlation between two adjacent pixels can be greatly reduced after encryption. Figures 5 and 6 also depict the correlations of plain and ciphered images in horizontal, vertical and diagonal adjacent pixels, respectively. In addition, correlation performance comparison with other existing image encryption algorithms is made, as shown in Table 6. It can be observed that the correlation between two adjacent pixels of plain images has been eliminated to a large extent and then enhances the ability to resist the chosen-plaintext attack. All these results indicate the proposed image encryption algorithm is effective and has higher security.

4.5. Entropy Analysis. For a given image, it is ideal that the character information of the image can be hidden entirely after being encrypted, and thus the intruder cannot carry an effective attack on it. To measure the complexity or uncertainty of given images, two entropy indicators are introduced here: information entropy and approximate entropy (ApEn) [41]. The former mainly measures the uncertainty of an information source, while the latter depicts the probability of new patterns appearing in the information source. Normally, the more complicated the sequence, the higher the entropy, and the less likely it is to leak information.

The information entropy $H(x)$ of an information source x can be calculated as

$$H(x) = -\sum p(x)\log_2 p(x), \tag{14}$$

where $p(x)$ represents the probability of source x. Normally, the greater the uncertainty of source x, the higher the entropy. A source with uniform distribution has the greatest uncertainty, and the information entropy is at its maximum. That is, the more information entropy closes to 8, the more uncertainty there is, and less information the system may leak. As shown in Table 7, the information entropy of images is enhanced to be in close proximity to the maximum value 8 under the current digit after encryption. That is, the probability of information leakage is very small, which means the encryption scheme is effective.

Generally, the more evenly distributed, the less likely new patterns are to emerge and the greater the ApEn value is. As shown in Table 8, the ApEn values of the ciphered images are enhanced to be more than 2.5 times than the ones of the original images and even be close to the mean value (about 2.1773nat) of the random images with uniform distribution, which further ensure the validity of the proposed scheme.

Furthermore, we compare the entropy performance of the proposed algorithm with other existing chaos-based image encryption schemes. As shown in Table 9, the information entropy of the proposed scheme is not only larger than that of other schemes but also closer to the maximum value of 8. In this way, the proposed scheme has higher security.

5. Conclusion

Given that there exists a contradiction between the security and implementation for most existing chaotic image encryption schemes, a novel chaos-based encryption scheme via switch control technology has been proposed. In this scheme, the three-dimensional Lorenz chaotic system is introduced to generate pseudorandom sequences with good randomness, and a switch control law is designed to realize the random permutation of the given images. The simulation results show that the proposed algorithm has a good performance, and the comparisons of entropy and other indicators also show its superiority.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work described in this paper was supported by the National Natural Science Foundation of China (no. 61702554) and the National Crypto Development Fund of China (no. MMJJ20170109). Thanks are due to the National Natural Science Foundation of China and the State Encryption Administration of China.

References

[1] T. Liu, H. Liu, Z. Chen, and A. M. Lesgold, "Fast blind instrument function estimation method for industrial infrared spectrometers," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 12, pp. 5268–5277, 2018.

[2] T. Liu, H. Liu, Y. Li, Z. Zhang, and S. Liu, "Efficient blind signal reconstruction with wavelet transforms regularization for educational robot infrared vision sensing," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 1, pp. 384–394, 2018.

[3] T. Liu, Y. Fu Li, H. Liu, Z. Zhang, and S. L. Risir, "Rapid infrared spectral imaging restoration model for industrial material detection in intelligent video systems," *IEEE Transactions on Industrial Informatics*, 2019.

[4] T. Liu, H. Liu, Y.-F. Li, Z. Chen, Z. Zhang, and S. Liu, "Flexible ftir spectral imaging enhancement for industrial robot infrared vision sensing," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 544–554, 2020.

[5] R. Matthews, "On the derivation of a "chaotic" encryption algorithm," *Cryptologia*, vol. 13, no. 1, pp. 29–42, 1989.

[6] G. Chen, Y. Mao, and C. K. Chui, "A symmetric image encryption scheme based on 3d chaotic cat maps," *Chaos, Solitons & Fractals*, vol. 21, no. 3, pp. 749–761, 2004.

[7] L. Zhang, X. Liao, and X. Wang, "An image encryption approach based on chaotic maps," *Chaos, Solitons & Fractals*, vol. 24, no. 3, pp. 759–765, 2005.

[8] N. K. Pareek, V. Patidar, and K. K. Sud, "Image encryption using chaotic logistic map," *Image and Vision Computing*, vol. 24, no. 9, pp. 926–934, 2006.

[9] Y. Zhou, L. Bao, and C. L. P. Chen, "A new 1d chaotic system for image encryption," *Signal Processing*, vol. 97, pp. 172–182, 2014.

[10] C. Song and Y. Qiao, "A novel image encryption algorithm based on dna encoding and spatiotemporal chaos," *Entropy*, vol. 17, no. 10, p. 6954, 2015.

[11] S. El Assad and M. Farajallah, "A new chaos-based image encryption system," *Signal Processing: Image Communication*, vol. 41, pp. 144–157, 2016.

[12] C. Pak and L. Huang, "A new color image encryption using combination of the 1d chaotic map," *Signal Processing*, vol. 138, pp. 129–137, 2017.

[13] C. Cao, K. Sun, and W. Liu, "A novel bit-level image encryption algorithm based on 2d-licm hyperchaotic map," *Signal Processing*, vol. 143, pp. 122–133, 2018.

[14] S. S. Moafimadani, Y. Chen, and C. Tang, "A new algorithm for medical color images encryption using chaotic systems," *Entropy*, vol. 21, no. 6, p. 577, 2019.

[15] M. Alawida, A. Samsudin, J. S. Teh, and R. S. Alkhawaldeh, "A new hybrid digital chaotic system with applications in image encryption," *Signal Processing*, vol. 160, pp. 45–58, 2019.

[16] Y. Zhang and Y. Tang, "A plaintext-related image encryption algorithm based on chaos," *Multimedia Tools and Applications*, vol. 77, no. 6, pp. 6647–6669, 2018.

[17] A. Akhshani, A. Akhavan, S.-C. Lim, and Z. Hassan, "An image encryption scheme based on quantum logistic map," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 12, pp. 4653–4661, 2012.

[18] C. Li, G. Luo, K. Qin, and C. Li, "An image encryption scheme based on chaotic tent map," *Nonlinear Dynamics*, vol. 87, no. 1, pp. 127–133, 2017.

[19] Y. Mao, G. Chen, and S. Lian, "A novel fast image encryption scheme based on 3d chaotic baker maps," *International Journal of Bifurcation and Chaos*, vol. 14, no. 10, pp. 3613–3624, 2004.

[20] H. M. Ghadirli, N. Ali, and R. Enayatifar, "An overview of encryption algorithms in color images," *Signal Processing*, vol. 164, pp. 163–185, 2019.

[21] X. Zhang and X. Wang, "Multiple-image encryption algorithm based on mixed image element and chaos," *Computers & Electrical Engineering*, vol. 92, pp. 6–16, 2017.

[22] C. Li, D. Lin, B. Feng, J. Lu, and F. Hao, "Cryptanalysis of a chaotic image encryption algorithm based on information entropy," *IEEE Access*, vol. 6, pp. 75834–75842, 2018.

[23] C. Zhu, G. Wang, and K. Sun, "Improved cryptanalysis and enhancements of an image encryption scheme using combined 1d chaotic maps," *Entropy*, vol. 20, no. 11, p. 843, 2018.

[24] Y. Xiong, C. Quan, and C. J. Tay, "Multiple image encryption scheme based on pixel exchange operation and vector decomposition," *Optics and Lasers in Engineering*, vol. 101, pp. 113–121, 2018.

[25] D. C. Mishra, R. K. Sharma, S. Suman, and A. Prasad, "Multi-layer security of color image based on chaotic system combined with RP2DFRFT and Arnold transform," *Journal of Information Security and Applications*, vol. 37, pp. 65–90, 2017.

[26] X. Chai, "An image encryption algorithm based on bit level brownian motion and new chaotic systems," *Multimedia Tools and Applications*, vol. 76, no. 1, pp. 1159–1175, 2017.

[27] P. Ramasamy, V. Ranganathan, S. Kadry, R. Damaševičius, and T. Blažauskas, "An image encryption scheme based on block scrambling, modified zigzag transformation and key generation using enhanced logistic-tent map," *Entropy*, vol. 21, no. 7, p. 656, 2019.

[28] Z. Hua, F. Jin, B. Xu, and H. Huang, "2d logistic-sine-coupling map for image encryption," *Signal Processing*, vol. 149, pp. 148–161, 2018.

[29] Y. Luo, J. Yu, W. Lai, and L. Liu, "A novel chaotic image encryption algorithm based on improved baker map and logistic map," *Multimedia Tools and Applications*, vol. 78, no. 15, pp. 22023–22043, 2019.

[30] Y.-Q. Zhang and X.-Y. Wang, "A symmetric image encryption algorithm based on mixed linear-nonlinear coupled map lattice," *Information Sciences*, vol. 273, no. 8, pp. 329–351, 2014.

[31] X. Wang and H. Li Zhang, "A novel image encryption algorithm based on genetic recombination and hyper-chaotic systems," *Nonlinear Dynamics*, vol. 83, no. 1-2, pp. 333–346, 2016.

[32] H. Zhu, X. Zhang, H. Yu, C. Zhao, and Z. Zhu, "An image encryption algorithm based on compound homogeneous hyper-chaotic system," *Nonlinear Dynamics*, vol. 89, no. 1, pp. 61–79, 2017.

[33] A. Kassem, H. Al Haj Hassan, Y. Harkouss, and R. Assaf, "Efficient neural chaotic generator for image encryption," *Digital Signal Processing*, vol. 25, no. 2, pp. 266–274, 2014.

[34] A. Kulsoom, D. Xiao, S. A. Aqeel-ur-Rehman, and S. A. Abbas, "An efficient and noise resistive selective image encryption scheme for gray images based on chaotic maps and dna complementary rules," *Multimedia Tools and Applications*, vol. 75, no. 1, pp. 1–23, 2016.

[35] Y. Xie, J. Yu, S. Guo, Q. Ding, and E. Wang, "Image encryption scheme with compressed sensing based on new three-dimensional chaotic system," *Entropy*, vol. 21, no. 9, p. 819, 2019.

[36] X. Li, Y. Jiang, M. Chen, and F. Li, "Research on iris image encryption based on deep learning," *EURASIP Journal on Image and Video Processing*, vol. 2018, no. 1, p. 126, 2018.

[37] J. Schuijers, J. P. Junker, M. Mokry et al., "Ascl2 acts as an R-spondin/wnt-responsive switch to control stemness in intestinal crypts," *Cell Stem Cell*, vol. 16, no. 2, pp. 158–170, 2015.

[38] N. Lu, P. Zhang, Q. Zhang et al., "Electric-field control of tri-state phase transformation with a selective dual-ion switch," *Nature*, vol. 546, no. 7656, pp. 124–128, 2017.

[39] H. Hu, L. Liu, and N. Ding, "Pseudorandom sequence generator based on the chen chaotic system," *Computer Physics Communications*, vol. 184, no. 3, pp. 765–768, 2013.

[40] Y. Luo, L. Cao, S. Qiu, H. Lin, J. Harkin, and J. Liu, "A chaotic map-control-based and the plain image-related cryptosystem," *Nonlinear Dynamics*, vol. 83, no. 4, pp. 2293–2310, 2016.

[41] S. M. Pincus, "Approximate entropy as a measure of system complexity," *Proceedings of the National Academy of Sciences*, vol. 88, no. 6, pp. 2297–2301, 1991.

Hindawi
Security and Communication Networks
Volume 2020, Article ID 7501894, 17 pages
https://doi.org/10.1155/2020/7501894

Research Article

Using a Subtractive Center Behavioral Model to Detect Malware

Ömer Aslan,[1,2] **Refik Samet**,[1] **and Ömer Özgür Tanrıöver**[1]

[1]*Ankara University, Computer Engineering Department, Ankara 06830, Turkey*
[2]*Siirt University, Computer Engineering Department, Siirt 56100, Turkey*

Correspondence should be addressed to Ömer Aslan; omer.aslan@siirt.edu.tr

Received 6 November 2019; Accepted 29 January 2020; Published 27 February 2020

Guest Editor: Hammad Afzal

In recent years, malware has evolved by using different obfuscation techniques; due to this evolution, the detection of malware has become problematic. Signature-based and traditional behavior-based malware detectors cannot effectively detect this new generation of malware. This paper proposes a subtractive center behavior model (SCBM) to create a malware dataset that captures semantically related behaviors from sample programs. In the proposed model, system paths, where malware behaviors are performed, and malware behaviors themselves are taken into consideration. This way malicious behavior patterns are differentiated from benign behavior patterns. Features that could not exceed the specified score are removed from the dataset. The datasets created using the proposed model contain far fewer features than the datasets created by *n*-gram and other models that have been used in other studies. The proposed model can handle both known and unknown malware, and the obtained detection rate and accuracy of the proposed model are higher than those of the known models. To show the effectiveness of the proposed model, 2 datasets with score and without score are created by using SCBM. In total, 6700 malware samples and 3000 benign samples are tested. The results are compared with those derived from *n*-gram and models from other studies in the literature. The test results show that, by combining the proposed model with an appropriate machine learning algorithm, the detection rate, false positive rate, and accuracy are measured as 99.9%, 0.2%, and 99.8%, respectively.

1. Introduction

Any software that performs malicious activities on victim machines is considered to be malware. Sophisticated malware uses packing and obfuscation techniques to make the analysis and detection processes more difficult [1]. Malware lies at the root of almost all cyber threats and attacks including global threats, advanced persistent threats (APTs), sensitive data theft, remote code execution, and distributed denial of service (DDoS) attacks. In recent years, the number, sophistication of malware attacks, and the economic damage caused by malware have been increasing exponentially. According to scientific and business reports, approximately 1 million malware files are created every day. According to cybersecurity ventures, cybercrime will cost the world economy approximately $6 trillion annually by 2021 [2]. According to the same report in 2019, ransomware malware costs around $11.5 billion globally [2].

Mobile malware is on the rise. According to the McAfee mobile threat report, there is a substantial increase in backdoors, fake applications, and banking Trojans for mobile devices [3]. The number of new mobile malware variants increased by 54% from 2016 to 2017 [4], and most types of unknown and mobile malware are evolved versions of known malware [5]. Moreover, malware attacks related to the healthcare industry, cloud computing, social media, Internet of Things, and cryptocurrencies are also on the rise [2, 6].

It is almost impossible to propose a method or system that can detect every new generation of sophisticated malware. The 4 main methods used to detect malware are based on signature, behavior, heuristic, and model checking detection. Each method has advantages and disadvantages.

Signature-based malware detector examines the features that encapsulate the program's structure and uniquely identify the malware. This method detects known malware efficiently, but it cannot detect unknown malware. Behavior-basedmalware

detector observes program behaviors using monitoring tools and determines whether the program is malware or benign. Although program codes change, the behavior of the program will remain relatively the same; thus, new malware can be detected with this method [7]. However, some malware does not run properly under the protected environment (e.g., virtual machine and sandbox environment), and thus, the malware sample may be incorrectly marked as benign.

In recent years, heuristic-based detection methods have been used frequently. These methods are complex detection methods that apply both experience and different techniques such as rules and machine learning techniques [8]. However, even if the heuristic technique can detect various forms of known and unknown malware [7], it cannot detect new malware that is quite different from existing malware. In model checking-based detection, malware behaviors are manually extracted, and behavior groups are coded using linear temporal logic (LTL) to display a specific feature [9]. Although model checking-based detection can successfully detect some unknown malware that could not be detected with the previous 3 methods, it is insufficient for detecting all new malware.

In this paper, the subtractive center behavior model (SCBM), which captures semantically associated behaviors when creating a dataset, is proposed. In this model, in addition to malware behaviors, system paths where malware behaviors are executed are also considered.

The proposed model makes the following contributions:

(i) SCBM is proposed to create a malware dataset with fewer features than known models.

(ii) Instead of directly using system calls as behaviors, system calls are mapped to relevant behaviors.

(iii) Behaviors are divided into groups, and risk scores are calculated based on the system path and active-passive behaviors.

(iv) Features are extracted from behaviors according to the type of resources and instances that have been used. This way malicious behavior patterns are segregated from benign behavior patterns.

(v) The proposed model can handle both known and unknown malware.

(vi) The obtained detection rate and accuracy of the proposed model are higher than those of the known models.

The rest of this paper is organized as follows. Section 2 defines malware and describes trends in malware technologies. Related work is summarized in Section 3. SCBM is explained in Section 4, and the case study is presented in Section 5. The results and discussion are provided in Section 6. Finally, the limitations and future works are given in Section 7, and the conclusion is given in Section 8.

2. Definition of Malware and Trends in Malware Technologies

Any software that intentionally executes malicious payloads on victim machines is considered to be malware [7]. There are different types of malware including viruses, worms, Trojan horses, rootkits, and ransomware. Common malware types and their primary characteristics can be seen in Table 1. The malware types and families are designed to affect the original victim machine in different ways (e.g., damaging the targeted system, allowing remote code execution, and stealing confidential data). Generally, hackers launch an attack by using malware, which exploits vulnerabilities in existing systems such as buffer overflow, injection, and sensitive data misconfiguration [10]. These days, the classification of malware is becoming more complex because some malware instances can present the characteristics of multiple classes at the same time [11].

Viruses, which are considered to be first malware that appeared in the wild, were defined as self-replicating automata by John von Neumann in the 1950s. However, practically the first virus called "the Creeper" was created in 1971 by Bob Thomas [12, 13]. In the early days, this software was written for simple purposes, but in time, it was replaced by a new generation of malware that targeted large companies and governments. Malware that runs in the kernel mode is more destructive and difficult to detect than traditional malware, and it can be defined as a new generation (next generation) of malware. The comparison between traditional and new generation malware can be seen in Table 2.

The inability to implement the operating system control features in the kernel mode makes the detection of new generation malware difficult. This malware can easily bypass protection software that is running in the kernel mode such as antivirus software and firewalls. In addition, by using this software, targeted and persistent cyberattacks that have never been seen before can be launched, and more than one type of malware can be used during the attacks. Examples of traditional versus new generation malware can be seen in Figures 1 and 2.

M represents malware, and (P_1, P_2, P_3, P_4) show the running processes that interact with the malware. First, M copies itself into different processes such as P_1, P_2, and P_3. Then, M deletes itself from the system to make itself invisible (Figure 2). In early days, rootkits were using similar techniques to hide themselves from the system. However, in process of time, many other kinds of malware (in some cases, rootkits are combining with viruses, worms, and Trojan horses) have started to use similar techniques to hide themselves as well. With the help of the processes, it has recently copied ($P_1 \longrightarrow P_2$; $P_1 \longrightarrow P_3$; $P_3 \longrightarrow P_4$) and it connects to remote system and makes changes on the victim's operating system. Even if the actual malware containing the malicious code has deleted itself from the system, the new version of the malware remains in and affects the system because the actual malware injected itself into different processes such as existing system files, third-party software, and newly created processes, which make the malware almost impossible to detect. To determine the malicious software mentioned in Figure 2, M and the P_1, P_2, P_3, and P_4 processes must be examined separately, and the relations among these processes should be determined.

Table 1: Common malware types and their primary characteristics.

Common malware types	Primary characteristics
Virus	Most common and well-known malware Attaches itself to other programs to replicate
Worm	Spreads by using computer network Allows unauthorized access Often opens backdoor in the victim system
Trojan Horse	Appears to be a normal program, but it is not Can open backdoors Can cause unauthorized access Can send critical information to the third party
Backdoor	Bypasses traditional security mechanisms Opens system to remote access Usually installed by using Trojans and worms Used by viruses and worms for complex attacks
Rootkit	Provides administrator-level access Hides their files from the operating system Can combine with other malware
Ransomware	Encrypts the data on infected system Victim needs to pay ransom to view the data
Spyware	Collects victim's sensitive information and sends them to third parties Commonly used to access credit card information or to identify user habits
Obfuscated malware	Can be any type of malware Uses obfuscation techniques to make detection process more difficult

Table 2: Comparison of traditional and new generation malware.

Comparison parameter	Traditional	New generation
Implementation level	Simple coded	Hard coded
State of behaviors	Static	Dynamic
Proliferation	Each copy is similar	Each copy is different
Through spreading	Uses .exe extension	Uses also different extensions
Permanence in the system	Temporal	Persistent
Interaction with processes	A few processes	Multiple processes
Using concealment techniques	None	Yes
Attack type	General	Targeted
Defensive challenge	Easy	Difficult
Targeted devices	A few devices	Many different devices

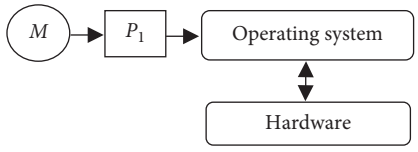


Figure 1: Traditional malware.

In addition, the new generation malware uses the common obfuscation techniques such as encryption, oligomorphic, polymorphic, metamorphic, stealth, and packing methods to make the detection process more difficult. This makes practically almost impossible to detect all malware with single detection approach. The well-known obfuscation techniques can be explained as follows:

(1) Encryption: malware uses encryption to hide malicious code block in its entire code [9]. Thus, malware becomes invisible in the host.

(2) Oligomorphic: a different key is used when encrypting and decrypting malware payload. Hence, it is more difficult to detect malware, which uses the oligomorphic method rather than encryption.

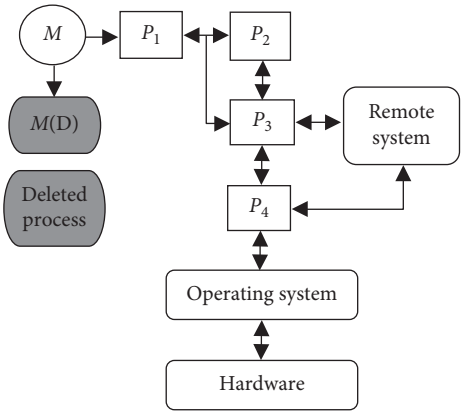


Figure 2: New generation malware.

(3) Polymorphic: malware uses a different key to encrypt and decrypt likewise the key used in the oligomorphic

and encryption method. However, the encrypted payload portion contains several copies of the decoder. Thus, it is more difficult to detect polymorphic malware when compared to oligomorphic malware.

(4) Metamorphic: metamorphic method does not use encryption. Instead, it uses dynamic code hiding which the opcode is changing on each iteration when the malicious process is executed [9]. It is very difficult to detect such malware because each new copy has a completely different signature.

(5) Stealth: the stealth method also called code protection implements a number of countertechniques to prevent it from being analyzed correctly. For example, it can make changes on the system and keep it hidden from detection systems.

(6) Packaging: packaging is an obfuscation technique to compress malware to prevent detection or hiding the actual code by using encryption. Due to this technique, malware can easily bypass firewall and antivirus software [7]. Packaged malware need to be unpacked before being analyzed.

3. Related Work

In recent years, there has been a rapid increase in the number of studies on malware analysis and detection. In the early years, signature-based detection was used widely. Over time, researchers have developed new techniques for detecting malware including detection techniques based on behavior, heuristics, and model checking. There is huge demand for methods that effectively detect complex and unknown malware. Thus, we present related research from the literature and examine the pros and cons of each study. The summary of related works can be seen in Table 3.

The similarities determined among features by using system calls were described in [14, 20]. Wagener et al. [14] proposed a flexible and automated approach that considered system calls to be program behaviors. They used an alignment technique to identify similarities and calculated the Hellinger distance to compute associated distances. The paper claimed that the classification process can be improved using a phylogenetic tree that represents the common functionalities of malware. They also claimed that obfuscated malware variants that show similar behaviors can be detected. The limitations of paper can be summarized as follows:

(1) Lack of knowledge is provided about the malware dataset.

(2) Statistical evaluation of performance is not provided.

(3) Comparison of proposed method against other methods is not given. Besides, it is not clear how phylogenetic tree can improve the performance.

Shan and Wang proposed a behavior-based clustering method to classify malware [20]. Behaviors were generated using system calls, and features within a cluster were shown to be similar. According to paper, the proposed method can detect 71.1% of unknown malware samples without FPs, while the performance overhead is around 9.1%. The proposed method is complex, not scalable for large datasets, and there are some performance issues on servers. Eliminating these deficiencies will improve the model performance.

A graph-based detection schema was defined in [15, 17, 21]. Kolbitsch et al. [21] proposed a graph-based detection method in which system calls are converted into a behavior graph, where the nodes represent system calls and the edges indicate transitions among system calls, to show the data dependency. The program graph to be marked is extracted and compared with the existing graph to determine whether the given program is malware. Although the proposed model has performed well for the known malware, it has difficulties detecting unknown malware.

Park et al. proposed a graph method that specifies the common behaviors of malware and benign programs [15]. In this method, kernel objects are determined by system calls, and behaviors are determined according to these objects. According to the paper, the proposed method is scalable and can detect unknown malware with high detection rates (DRs) and false positive (FP) rates close to 0%. In addition, the proposed model is highly scalable regardless of new instances added and robust against system call attacks. However, the proposed method can observe only partial behavior of an executable. To explore more possible execution paths would improve the accuracy of this method.

Naval et al. [17] suggested a dynamic malware detection system that collects system calls and constructs a graph that finds semantically relevant paths among them. To find all semantically relevant paths in a graph is a NP-*complete* problem. Thus, to reduce the time complexity, the authors measured the most relevant paths, which specify malware behaviors that cannot be found in benign samples. The authors claim that the proposed approach outperforms its counterparts because, unlike similar approaches, the proposed approach can detect a high percentage of malware using system call injection attacks. Paper has some limitations such as performance overhead during path computation and vulnerable to call-injection attacks and cannot identify all semantically relevant paths efficiently. Eliminating these limitations may improve the performance.

Fukushima et al. proposed a behavior-based detection method that can detect unknown and encrypted malware on Windows OS [22]. The proposed framework not only checks for specific behaviors that malware performs but also checks normal behaviors that malware usually does not perform. The proposed scheme's malware DR was approximately 60% to 67% without any FP. The DR is very low; to increase the DR, more malicious behaviors can be identified, and to prove the effectiveness of new method, the test set will be extended.

Lanzi et al. [23] proposed a system-centric behavior model. According to the authors, the interaction of malware programs with system resources (directory, file, and registry) is different from that of benign programs. The behavioral sequences of the program to be marked are compared with the behavior sequences of the two groups (i.e., malware and benign). The paper claimed that the suggested system detects

Table 3: Summary of related works on malware detection methods.

Paper	Feature representation	Goal/success	Year
Wagener et al. [14]	System calls, Hellinger distance, phylogenetic tree	Identify new and different forms of malware	2008
Park et al. [15]	Creating system call diagrams	Identify different forms of malware	2013
Islam et al. [16]	Printable strings, API method frequencies	Identify malware with 97% accuracy	2013
Naval et al. [17]	Diagram of system calls and relations	Detect code insertion attacks	2015
Das et al. [18]	System call frequencies, *n*-gram	Identify new and different forms of malware	2016
Zhang et al. [19]	API calls sequence to construct a behavior chain	It achieved 98.64% accuracy with 2% FPR	2019

a significant fraction of malware with a few FP. The proposed method cannot detect all malicious activities such as malware which does not attempt to hide its presence or to gain control of the OS and which uses only computer network for transmission. To include network-related policies and rules for malware, which ignores to modify legitimate applications and the OS execution, can improve the performance.

Chandramohan et al. proposed BOFM (bounded feature space behavior modeling), which limits the number of features to detect malware [24]. First, system calls were transformed into high-level behaviors. Then, features were created using the behaviors. Finally, the feature vector is created and machine learning algorithms are applied to the feature vector to determine whether the program is malware or benign. This method ignored the frequency of system calls. Executing the same system call repeatedly can cause DoS attacks. Considering the frequency of system calls can improve DR and accuracy.

A hardware-enhanced architecture that uses a processor and an FPGA (field-programmable gate array) is proposed in [18]. The authors suggested using an FCM (frequency-centralized model) to extract the system calls and construct the features from the behaviors. Features obtained from the benign and malware samples are used to train the machine learning classifier to detect the malware. The paper claimed that the suggested system achieved a high classification accuracy, fast DR, low power consumption, and flexibility for easy functionality upgrades to adapt to new malware samples. However, malware can perform various behaviors, and there is no uniform policy to specify number of behaviors and features to be extracted before triggering the early prediction. Furthermore, the proposed method performance has only been compared with BOFM and *n*-gram which is not enough to determine the efficiency of the proposed model.

Ye et al. proposed associative classification post-processing techniques for malware detection [25]. The proposed system greatly reduces the number of generated rules by using rule pruning, rule ranking, and rule selection. Thus, the technique does not need to deal with a large database of rules, which accelerate the detection time and improve the accuracy rate. According to the paper, the proposed system outperformed popular antivirus software tools such as McAfee, VirusScan, and Norton Antivirus and data mining-based detection systems such as naive Bayes, support vector machine (SVM), and decision tree. To collect more API calls, which can provide more information about malware, and identify complex relationships among the API calls may improve the performance.

A supervised machine learning model is proposed in [26]. The model applied a kernel-based SVM that used weighting measure, which calculates the frequency of each library call to detect Mac OS X malware. The DR was 91% with an FP rate of 3.9%. Test results indicated that incrementing sample size increases the detection accuracy but decreases the FPR. Combining static and dynamic features, using other techniques such as fuzzy classification and deep learning can increase the performance.

The method of grouping system calls using MapReduce and detecting malware according to this grouping is described by Liu et al. [27]. According to the authors, most of the studies performed so far were process-oriented, which determines a process as a malware only by its invoked system calls. However, most current malware is module-based, which consists of several processes, and it is transmitted to the system via driver or DLL [28]. In such cases, malware performs actions on the victim's machine by using more than one process instead of its own process. When only one process is analyzed, malware can be marked as benign. However, there are some limitations of the proposed method. The limitations of this method can be addressed as follows: (1) some malware does not require persistent behavior ASEP; (2) persistent malware behaviors can be completed without using system calls; and (3) the cost of data transmission has not been measured. Besides, the proposed method results were not compared with other studies in the literature. Eliminating abovementioned limitations can improve the method performance.

A detection system that combines static and dynamic features was proposed in [16]. This system has three properties: the frequencies (in bytes) of the method, the string information, and the system calls and their parameters. By combining these properties, the feature vector was constructed and classified using classification algorithms. The paper claimed that the detection of the proposed system is reasonable and increases the probability of detecting unknown malware compared to their first study. However, the probability of detecting unknown malware is still low and FPR is high. Using more distinctive features and train model with more malware may improve the method performance for unknown malware.

Recent works on malware behaviors are represented in [19, 29–31]. Lightweight behavioral malware detection for windows platforms is explained in [29]. It extracts features from prefetch files and discriminates malware from benign applications using these features. To show the effectiveness of the malware detector on the prefetch datasets, they used LR (logistic regression) and SVM (support vector machine)

classifier. According to the authors, test results are promising especially TPR and FPR for practical malware detection. Choi et al. proposed metamorphic malicious code behavior detection using probabilistic inference methods [30]. It used FP-growth and Markov logic networks algorithm to detect metamorphic malware. FP-growth algorithm was used to find API patterns of malicious behaviors from among the various APIs. Markov logic networks algorithm was used to verify the proposed methodology based on inference rules. According to the test results, the proposed approach outperformed the Bayesian network by 8% higher category classification.

Karbab and Debbabi proposed MalDy (mal die), a portable (plug and play) malware detection, and family threat attribution framework using supervised ML techniques [31]. It uses behavioral reports into a sequence of words, along with advanced natural language processing (NLP) and ML techniques to extract relevant security features. According to the test results, MalDy achieved 94% success on Win32 malware reports. A depth detection method on behavior chains (MALDC) is proposed in [19]. The MALDC monitors behavior points based on API calls and uses the calling sequence of those behavior points at runtime to construct behavior chains. Then, it uses the depth detection method based on long short-term memory (LSTM) to detect malicious behaviors from the behavior chains. To verify the performance of the proposed model, 54.324 malware and 53.361 benign samples were collected from Windows systems and tested. MALDC achieved 98.64% accuracy with 2% FPR in the best case.

The malware detection schema landscape is changing from computers to mobile devices, and cloud-, deep learning-, and mobile-based detection techniques are becoming popular. However, these detection schemas have some problems, too. For instance, deep learning-based detection approach is effective to detect new malware and reduces features space sharply [32], but it is not resistant to some evasion attacks. On the other hand, cloud-based detection approach increases DR, decreases FPs, and provides bigger malware databases and powerful computational resources [33]. However, the overhead between client and server and lack of real monitoring is a still challenging task in cloud environment. Mobile- and IoT-based detection approaches can use both static and dynamic features and improve detection rates on traditional and new generation of malware [34]. But, they have difficulties to detect complex malware and are not scalable for large bundle of apps.

In the literature review, the malware detection methods have been summarized. Current studies can be divided into 2 major groups:

(1) Studies that apply certain rules directly to behaviors or features to group similar behaviors and extract the signature (no ML is required at this stage)

(2) Studies that determine behaviors, extract features from behaviors, and apply classification by using ML and data mining algorithms

In current studies, some new techniques and methods have been used widely for many years. These techniques and methods are can be listed as follows:

(i) Datamining and ML have been used widely for a decade, and cloud and deep learning have been used recently in malware detection

(ii) The n-gram, n-tuple, bag, and graph models have been used to determine the features from behaviors

(iii) Probability and statistical methods such as Hellinger distance, cosine coefficient, chi-square, and distance algorithms are used to specify similarities among features

Current studies which are explained above have some limitations and can be addressed as follows:

(i) Many detection methods produce high FPs and require complex and resource-intensive hardware

(ii) Detection rate and accuracies are low

(iii) Cannot effectively handle new and complex malware

(iv) Focused on specific malware type, family, or specific OS

(v) Prone to evasion techniques

(vi) Have difficulties to handle all malicious behaviors

(vii) Feature extraction methods are not effective, so the size of the features increases overtime

As a result, the difficulties in defining behaviors and identifying the similarities and differences among the extracted properties have prevented the creation of an effective detection system. The use of new methods and approaches along with the use of ML and data mining algorithms in malware detection has begun to play a major role in making the extracted features meaningfully.

On the contrary, the SCBM has a high detection rate and accuracy with low FP. It can handle new and complex malware to a certain degree, and it is resistant to evasion techniques. Besides, the feature extraction method is effective and only specifies the features which can discriminate malware from benign. During the feature extraction process, the SCBM assigns numbers to each feature, which shows the importance of the feature in the dataset. Thus, the model does not need feature selection techniques before ML, and this makes SCBM faster and less resource-intensive.

4. Subtractive Center Behavior Model

This section describes the system architecture and explains the proposed model in detail.

4.1. Architecture of the Proposed Model. The system architecture of the proposed malware detection model is summarized in Figure 3.

According to the proposed model, the program samples are first collected and analyzed by relevant dynamic tools. Then, the behavior is determined according to the results of the analysis. After that, behaviors are grouped according to the determined rules, and features are extracted. Finally, the most important features are selected, and the system is

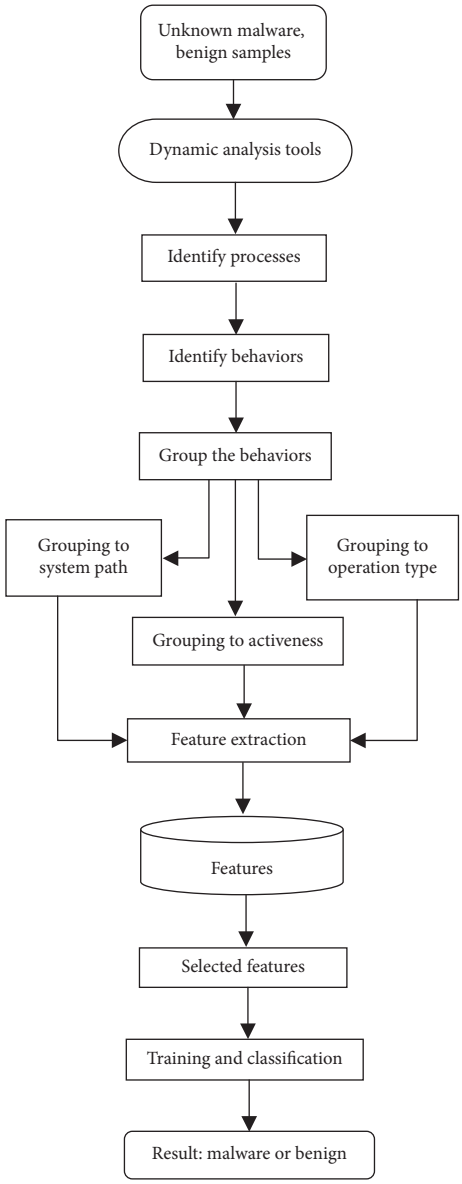


Figure 3: Proposed model architecture.

trained. Based on the training data, each sample is marked as malware or benign.

During the detection process, the SCBM specifies malicious behavioral patterns which can be seen in malware, but not seen or rarely seen in benign samples. Scoring system is used to determine the behavioral patterns. For instance, even if malware (M) and benign (B) samples system calls are the same (in real examples, this is not the case) $M = B = \{a, b, c, d, e\}$, the behavior patterns will be different. $M_{\text{pattern}}(\text{candidate}) = \{ab, ac, ce\}$, where $ab_{\text{score}} = 4$, $ac_{\text{score}} = 1$, $ce_{\text{score}} = 3$, while $B_{\text{pattern}}(\text{candidate}) = \{ab, ac, ce\}$, where $ab_{\text{score}} = 1$, $ac_{\text{score}} = 1$, and $ce_{\text{score}} = 0$. In this case, $M_{pattern} = \{ab, ce\}$, while $B_{pattern} = \{\ \}$, and we can easily differentiate malware from benign.

To collect the execution trace of each sample, both a process monitor and explorer are used in this study, but other dynamic tools such as API monitor and different sandboxes can be used as well. The proposed system is implemented using the Python scripting language, and classification is done on Weka. To prove the efficiency of the proposed model, different tools and programming language have been used. However, someone can use different tools and can get better results with proposed model. Thus, the implementation of proposed model does not put restriction on SCBM.

4.2. Proposed Model. In this study, the SCBM creates a dataset. When the SCBM and the n-gram model are compared, the SCBM contains far fewer features and determines the related processes more clearly than n-gram. In the proposed model, system paths, where malware behaviors are performed, and the malware behaviors themselves are taken into consideration. Based on each malware behavior and related system path, a score is assigned. Features that do not exceed the specified score are removed from the dataset. For example, to run properly, each process accesses certain system files and performs similar actions and behaviors. Those behaviors and the resulting properties are not included in the dataset. Therefore, the datasets created using the proposed model contain far fewer features than the datasets created by n-gram and the models used in other studies. The proposed SCBM model consists of following phases:

(i) Phase 1: convert the actions into behaviors
(ii) Phase 2: divide the behaviors into groups and calculate the risk scores
(iii) Phase 3: group the behaviors according to the types of resources
(iv) Phase 4: group the behaviors based on the same resources but different instances
(v) Phase 5: extract the features from repeated behaviors
(vi) Phase 6: extract the features from different data sources
(vii) Phase 7: calculate the risk scores for each behavior based on active/passive behaviors

The details of these phases are given below.

4.2.1. Phase 1: Convert Actions into Behaviors. In this phase, system calls such as Windows API and Windows Native API calls are converted into higher-level operations, and the associated behaviors are generated. For example, if the sequence of the running program's operations are in the order of NtCreateFile, NtWriteFile, and NtCloseFile, then the mapped behavior will be WriteFile. When we convert the action into a behavior, we drop the Nt and remove the NtCreateFile and NtCloseFile actions, which are not needed for real behavior. Similarly, if the system calls are order of NtCreateFile, NtQueryFile, and NtCloseFile, the mapped behavior will be SearchFile. In this way, low-level system calls are transformed into higher-level behaviors. The algorithm used to create behaviors is shown in Algorithm 1.

```
(1) d1 ⟵ file1, d2 ⟵ file2, n ⟵ u(d1)
(2) for i ⟵ 1 to n
(3)    if (d1[i][state] = = 'AE')
(4)       ψ ⟵ A
(5)    else
(6)       ψ ⟵ P
(7)    end if
(8)    if (P.name = = d1·FileName)
(9)       μ ⟵ self
(10)   elif (eST = = 'ss')
(11)      μ ⟵ system
(12)   elif (eST = = 'ts')
(13)      μ ⟵ thirdParty
(14)   else
(15)      1 = 1
(16)   end if
(17)   if (d1[i−1][o] ! = d1[i][o])
(18)      if (d1[i][o] ! = rcK && d1[i][o] ! = cF && d1[i][o] ! = tE && d1[i][o] ! = pE)
(19)         if (d1[i−1][s] = = d1[i][s])
(20)            Write.d2()
(21)         end if
(22)      end if
(23)   end if
(24)   if (d1[i−1][o] = d1[i][o])
(25)      if (d1[i−1][sfP] ! = d1[i][sfP])
(26)         if (d1[i][o]! = rcK&& d1[i][o] ! = cF && d1[i][o] ! = tE && d1[i][o] ! = pE)
(27)            write.d2()
(28)         end if
(29)      end if
(30)   end if
(31) end for
```

Algorithm 1: Malware behavior creation algorithm.

In Algorithm 1, d_1, d_2, and n represent the input action sequence, output behavior sequence, and input size, respectively. The algorithm takes d_1 as an input and generates d_2. During this process, AE (active) and PE (passive) behaviors are identified, and sfPs (system file paths) such as self, system, and third party's software are determined. On this basis, ψ and μ, which represent action state and action type, are calculated by using AE, PE, and eST (action state type). Finally, system calls, which cannot define new behaviors, such as rcK: "RegCloseKey," cF: "CloseFile," tE: "Thread Exit," and pE: "Process Exit," are eliminated from the action list, and the rest of the actions are written to the d2 file.

An example system-call sequence and corresponding behaviors are given in Table 4. The system calls that are produced by each sample are formulated as $S = \{a, b, c, d, \ldots, n\}$, where S represents the system-call sequence and $a, b, c, \ldots, n$ represent each system call. Only $s \subset S$ is taken into consideration when building behaviors. In this way, the behaviors that define the program are clarified, and the data to be analyzed are reduced significantly before feature extraction.

4.2.2. Phase 2: Divide the Behaviors into Groups and Calculate the Risk Scores. The behaviors identified in the previous phase are divided into three groups: self-generated behaviors, behaviors on third-party software, and behaviors on system software. In this section, the risk score is calculated for each behavior and its path (Table 5). The risk score is numbered from 0 to 4, where 0 means that related behavior is normal and can be seen in both malware and benign samples and 4 means that the related behavior is risky, likely to be seen for malware and rarely seen in benign samples (Table 5). The score is assigned based on the behavior path performed by the program sample. SGB1 shows the first type of behaviors from self-generated behaviors, TPB1 shows the first type of third-party behaviors, and SB1 shows the first type of system behaviors. Higher score is given to system behaviors because more differentiating malicious behaviors are performed on system files. In addition, a score is assigned for active and passive behaviors, as explained in phase 7. A threshold value was used when excluding behaviors. For instance, feature $x_i \in$ feature set X consists of $y_1, y_2, \ldots, y_n$ behaviors. The risk score for feature path (rsP) is calculated for x_i as follows:

$$x_i(\mathrm{rsP}) = \frac{y_1(\mathrm{rsP}) + y_2(\mathrm{rsP}) + \cdots + y_n(\mathrm{rsP})}{n}. \quad (1)$$

Let a be a specified threshold value, if $x_i(\mathrm{rsP}) \geq a$, x_i is in the feature set. Otherwise, x_i is not in the feature set.

Table 4: Sample malware execution trace and behaviors (list is abbreviated).

Action call	System path	Extracted behavior
NtCreateFile	"c:\windows\...\sfile$_1$.exe," malware.exe ⟶ 1	CreateFile (1)
NtCreateFile	"c:\programfiles\...\," malware.exe ⟶ 2	none
NtQueryDirectory	2, "c:\programfiles\...\," malware.exe ⟶ 3	SearchDirectory (2)
NtReadFile	3, "c:\...\tfile$_1$.txt," malware.exe ⟶ 4	ReadFile (3)
NtReadFile	3, "c:\...\tfile$_2$.exe," malware.exe ⟶ 5	ReadFile (3)
NtCloseFile	4, "tfile$_1$.txt," malware.exe ⟶ 6	none
NtWriteFile	1, "sfile$_1$.exe," malware.exe ⟶ 7	WriteFile (1)
NtReadFile	7, "sfile$_1$.exe," malware.exe ⟶ 8	ReadFile (7)
NtWriteFile	5, "tfile$_2$.exe," sfile$_1$.exe ⟶ 9	WriteFile (5)
NtCreateKey	"hklm\software\...\, key$_1$," tfile$_2$.exe ⟶ 10	none
NtSetValue	10, "key$_1$," tfile$_2$.exe ⟶ 11	SetValue (10)
NtRegCloseKey	11, "key$_1$," tfile$_2$.exe ⟶ 12	none
NtCreateFile	"c:\windows\...\stfile$_1$.dll," tfile$_2$.exe ⟶ 13	none
NtCreateFile	"c:\windows\...\stfile$_2$.dll," tfile$_2$.exe ⟶ 14	none
NtCloseFile	8, "sfile$_1$.exe," malware.exe ⟶ 15	none
NtReadFile	13, "stfile$_1$.dll," tfile$_2$.exe ⟶ 16	ReadFile (13)
NtReadFile	13, "stfile$_1$.dll," tfile$_2$.exe ⟶ 17	ReadFile (13)
NtReadFile	14, "stfile$_2$.dll," tfile$_2$.exe ⟶ 18	ReadFile (14)
NtCloseFile	17, "stfile$_1$.dll," tfile$_2$.exe ⟶ 19	none
NtCloseFile	18, "stfile$_2$.dll," tfile$_2$.exe ⟶ 20	none
NtCloseFile	9, "tfile$_2$.exe," tfile$_2$.exe ⟶ 21	none

Table 5: Risk score calculation.

System path	Path risk score (PRS)	Behavior risk score (BRS)	
		P	*A*
SGB1, TPB1, SB1	0	0	3
SGB2, SB2	1	0	3
SB3	2	0	3
TPB2, SB4	3	0	3
SB5	4	0	3

(1) Phase 2.1: *Self-generated behaviors (SGB)*. When an executed malware/benign sample performs behaviors on its own directory (SGB1), these behaviors are determined as the lowest dangerous behaviors and assigned a risk score of 0. In this case, because the program needs to retrieve some data from its own file to run properly, it generates normal behaviors that cannot be categorized as dangerous. However, when an executed malware/benign sample presents registry or network-related behaviors within some files (SGB2), this behavior group is considered to be slightly more dangerous and is assigned a risk score of 1. The behaviors marked with a risk score of 1 are likely to be included in the dataset according to the specified threshold. For instance, the behavior in which a file that creates another file and then copies its own file content to another file is more dangerous than the behavior that retrieves some data from its own file to run properly.

(2) Phase 2.2: Third-Party Behaviors (TPBs). Many programs require third-party software to run properly. For instance, in order to compile and run a program written in the Python language, the program will frequently perform behaviors for the file path (TPB1) where this language exists. Such behavior is considered harmless, and the behavior risk score is assigned as 0. However, behaviors related to directories and files that are not related to the performed sample (TPB2) are considered dangerous and the behavior risk score is assigned as 3.

(3) Phase 2.3: System behaviors (SBs). Programs are needed to interact with the operating system to work properly. Typically, this interaction is provided by system DLLs, background processes, Windows services, etc. on the Windows operating system. Most of these interactions are considered normal, while some of them are classified as malicious. If a program contains interactions that are necessary for the program to work properly, these type of behaviors (SB1) are evaluated not dangerous and lowest level risk score is assigned as 0. If the program uses "GDI32.dll" and "shell32.dll" [35] which can be used for both in malicious and benign behaviors (SB2), the risk score assigned as 1. If the program uses "User32.dll" and "kernel32.dll," which can be used frequently by malware and also sometimes used by benign (SB3), the risk score is assigned as 2. However, if the program frequently calls "Wininet.dll," "Advapi32.dll," and directly calls "Ntdll.dll" instead of "kernel32.dll" or uses high-level methods that are likely to be categorized as dangerous such as "ReadProcessMemory" and "AdjustTokenPrivileges" [35] (SB4), then a behavior risk score is assigned as 3.

In addition, if the program is attempting to interfere with system processes such as "svchost.exe" and "winlogon.exe" and to use these processes to access system databases that contain critical information, then these behaviors (SB5) are also considered malicious and behavior risk score is assigned as 4. Furthermore, if the same name as the system files in different system paths such as "svchost.exe," "winlogon.exe," and

"smss.exe" have been created or if the file is automatically initializing itself each time, the system is started (autostart locations such as "hklm\software\...\currentversion\run," "hklm\software\...\currentversion\runonce," "c:\users\...\startmenu\...\startup"), then these behaviors (SB4) are also considered malicious and behavior risk score is assigned as 3.

4.2.3. Phase 3: Group the Behaviors according to Types of Resources. Operating system resources are divided into groups such as file, registry, network, section, and thread; and the same types of resources are generally considered when determining property relationships. For instance, in Table 4, the behaviors of ReadFile (7, "sfile1.exe," malware.exe, 8) and WriteFile (5, "tfile2.exe," sfile1.exe, 9) are directly associated with each other. However, SetValue (10, "key1," tfile2.exe, 11) and ReadFile (13, "\...\stfile1.dll," tfile2.exe, 16) are not directly associated with each other. Thus, ReadFile and WriteFile can create a property, while SetValue and ReadFile cannot create a property.

4.2.4. Phase 4: Group the Behaviors on the Same Resources but Different Instances. While behaviors on the same resource (file and registry) and the same file format create the same properties, behaviors on the same resource on different file formats (exe, txt, sys, and dll) create different properties. For example, ReadFile ("tfile$_1$.txt," malware.exe ⟶ 4) and ReadFile ("tfile$_2$.exe," malware.exe ⟶ 5) create two different properties (Table 4), while ReadFile (13, "stfile$_1$.dll," tfile$_2$.exe ⟶ 16) and ReadFile (14, "stfile$_2$.dll," tfile$_2$.exe ⟶ 18) create the same property.

4.2.5. Phase 5: Extract the Features from Repeated Behaviors. The successive behaviors on the same resource and sample are set to a single property. Behaviors that occur in different locations and names are set to the same feature as well, but the importance of the feature increases.

4.2.6. Phase 6: Extract the Features from Different Data Resources. Behaviors that are on different resources but are indirectly determined as having a relationship also create a property. For example, although their behaviors take place in different resources, WriteFile (5, "tfile$_2$.exe," sfile$_1$.exe ⟶ 9) and SetValue (10, "key$_1$," tfile$_2$.exe ⟶ 11) (Table 4) create a property between them.

4.2.7. Phase 7: Calculate the Risk Scores for Each Behavior Based on Active/Passive Behaviors. Active behaviors are considered to be more dangerous than passive behaviors, and consequently, a higher level of danger is assigned. For example, while the danger level for ReadFile is set to 0, the danger level of WriteFile is set to 3. The feature creation algorithms are shown in Algorithms 2 and 3.

In Algorithms 2 and 3, the first algorithm contains abbreviations d_2, d_3, *(rD, tY, aS,* and *sRY)*, and *pRS,* which define input file, output file, related file paths, and each file path risk score and the second algorithm contains

```
(1) d2 ⟵ file2, d3 ⟵ file3, n ⟵ u(d2)
(2) for i ⟵ 1 to n
(3)   if (μ = = 'self')
(4)     if ( P.name = = d2.fileName)
(5)       pRS ⟵ 0
(6)     elif (P.name! = d2.fileName && d2.fileName = = rD)
(7)       pRS ⟵ 3
(8)     else
(9)       pRS ⟵ 2
(10)    end if
(11)  elif (μ = = 'ts')
(12)    if (d2[i][fP] = = tY)
(13)      pRS ⟵ 2
(14)    # Registry Autostart Location
(15)    elif (d2[i][fP] = = aS)
(16)      pRS ⟵ 3
(17)    else
(18)      pRS ⟵ 0
(19)    end if
(20)  elif (μ = = 'ss')
(21)    if (P.name = = d2.fileName)
(22)      pRS ⟵ 0
(23)    elif ( d2.fileName = = '*.exe')
(24)      pRS = 3
(25)    elif (d2[i][sfP] = = sRY)
(26)      pRS ⟵ 3
(27)    elif (d2[i][sfP] = = rD)
(28)      pRS ⟵ 3
(29)    else
(30)      pRS ⟵ 0
(31)    end if
(32)  end if
(33) end for
```

Algorithm 2: Feature creation algorithm I.

abbreviations d_2, d_4, *as, (O,* O_1, and O_2*), π, rdF,* and *weF,* which define input file, output file, action state, action values, operation value, "ReadFile," and "WriteFile." In Algorithms 2 and 3, the risk score is first calculated for each behavior, and the features from the related behaviors are constructed. For example, let $B = \{a, b, c, d\}$ be a behavior sequence, where a and c are active behaviors while b and d are passive behaviors. In addition, behavior a is related to behaviors b *and* c, and behavior b is related to behavior d. In this case, features (F) and their risk scores (rS) are calculated as

$$F = \{a, ab, ac, b, bd, c, d\},$$
$$rS = \{3 + rS_a; 3 + rS_{ab}; 4 + rS_{ac}; 0 + rS_b; 0 + rS_{bd}; 3 + rS_c; 0 + rS_d\}, \quad (2)$$

where the first score represents the active-passive risk score and the second score represents the path score. After the feature sequences have been generated, the frequency of each feature is calculated. The features that have a risk score above a certain threshold are considered during classification. In this case, the number of features decreases significantly, and classification algorithms produce better results without the use of feature selection algorithms.

```
(1) d2 ⟵ file2, d4 ⟵ file4, n ⟵ u(d2)
(2) for i ⟵ 1 to n
(3)   if (i < n − 10)
(4)     for j ⟵ i + 1 to i + 10
(5)       fP2 = d2[j][sfP]
(6)       if (P1.as = = P2.as && P1.as = = 'A')
(7)         ψ ⟵ 'AA'
(8)       elif (P1.as = = P2.as && P1.as = = 'P')
(9)         ψ ⟵ 'PP'
(10)      else
(11)        ψ ⟵ 'AP' = 'PA'
(12)      end if
(13)      if (d2[j][o] = = rdF && d2[j+1][o] = = weF)
(14)        π ⟵ O1 + ' ' + O2
(15)      if (d2[j]['μ'] = = 'self')
(16)        π ⟵ π + 'S'
(17)      elif (d2[j]['μ'] = = 'ts')
(18)        π ⟵ π + 'TP'
(19)      elif (d2[j]['μ'] = = 'ss')
(20)        π ⟵ π + 'ST'
(21)      else
(22)        2 ⟵ 2
(23)      end if
(24)      write.d4()
(25)      if (sfP1 = = sfP2 && O1! = O2)
(26)        π ⟵ O1 + ' ' + O2
(27)        if (d2[j]['μ'] = = 'self')
(28)          π ⟵ + 'S'
(29)        elif (d2[j]['μ'] = = 'ts')
(30)          π ⟵ π + 'TP'
(31)        elif (d2[j]['μ'] = = 'ss')
(32)          π ⟵ π + 'ST'
(33)        else
(34)          2 ⟵ 2
(35)        end if
(36)      write.d4()
(37)    end for
(38)  end if
(39) end for
```

Algorithm 3: Feature creation algorithm II.

Using the SCBM, Table 4 malware behaviors, Table 6 malware features, and Table 7 feature vector are generated. In Table 6, the Risk IDs column provides information about features. By looking at the Risk IDs column, the importance of each feature and risk score can be understood. In the Risk IDs, column I_a represents property types such as self, third party, and system; b represents the level of property; and A and P represent active and passive, respectively. For example, in $I_1 2$, A can be evaluated as a related process trying to make changes on its files by using active behaviors, while in $I_3 1$, P can be evaluated as a related process trying to perform operations on system files by using passive behaviors. When the values for Table 7 are obtained by using Table 6, a value of 0 is assigned for missing properties, 1 is assigned for one-time repeated properties, and x is assigned for x-time repeated properties. In addition, risk scores are assigned as a subfeature of the feature, considering behavioral groups and danger levels.

When comparing SCBM and the n-gram model, the test results showed that the number of created features decreases rapidly while the remaining features are more closely related one another. The dataset constructed by n-gram contained approximately 37-folds more features than the proposed model's dataset, which shows that machine learning algorithms likely perform better on dataset that is generated by the proposed model.

5. Case Study

This section describes the case study and experiments. Test cases were performed on different versions of Windows such as Windows 7 virtual machines, Windows 8 virtual machines, and Windows 10. For malware analysis, a process explorer and process monitor were used. To show the effectiveness of the proposed model, 2 datasets with score and without score by using SCBM have been created, and the results are compared with those of n-gram and other methods from the literature. A dataset with score is a modification of a dataset without score, which takes the features that can precisely represent each sample. In total, 6700 malware and 3000 benign samples have been analyzed. This section consists of 5 parts: data collection, representation, differentiate malicious patterns, ML and detection, and model performance and evaluation.

5.1. Data Collection. Malware samples were collected from a variety of sources such as Malware Benchmark [36], ViruSign [37], Malshare [38], Malware [39], KerelMode [40], and Tekdefense [41]. The malware was labeled using Virustotal [42], which uses approximately 70 antivirus scanners online and 10 antivirus scanners locally such as Avast, AVG, ClamAV, Kaspersky, McAfee, and Symantec. For this purpose, 6700 malware samples were randomly selected among 10,000 malware samples and analyzed. The dataset contains different malware types including viruses, Trojans, worms, backdoor, rootkit, ransomware, and packed malware (Figure 4) and contains different malware families such as agent, rooter, generic, ransomlock, cryptolocker, sality, snoopy, win32, and CTB-Locker. Analyzed malware is created from year 2000 to 2019 and can be categorized as regular known malware, packed malware, complicated malware, and some zero-day malware. The dataset contains 3000 benign samples from several categories including system tools, games, office documents, sound, multimedia, and other third-party software.

The malware signature was used for each scanner, and each malware was marked at the deepest level as possible. For example, a Trojan downloader and a virus downloader were marked as downloader, and key logger was marked as keylogger instead of spyware. Some of the malware could not be categorized; those malware files were marked as malware. The majority of the malware tested were Trojan horses, viruses, adware, worms, downloader, and backdoor. Other types of malware tested were rootkit, ransomware, dropper, injector, spyware, and packed malware (Figure 4).

5.2. Data Representation. As discussed in Section 4.2, the proposed model takes each malware sample as an input and

Table 6: Extracted features.

No	Risk IDs	Features	Related sources
1	$I_1$2, A	{CreateFileSF}	"c:\windows\...\sfile$_1$.exe"
2	$I_1$2, A	{WriteFileSF }	"c:\windows\...\sfile$_1$.exe"
3	$I_1$2, $I_1$2, A, P	{WriteFileSF, ReadFileSF}	"c:\windows\...\sfile$_1$.exe" "c:\windows\...\sfile$_1$.exe"
4	$I_1$2, $I_2$2, P, A	{ReadFileSF, WriteFileTP}	"c:\windows\...\sfile$_1$.exe" "c:\programfiles\...\tfile$_2$.exe"
5	$I_2$2, P	{SearchDirectoryTP}	"c:\programfiles\...\"
6	$I_2$2, $I_2$2, P, P	{SearchDirectoryTP, ReadFileTP}	"c:\programfiles\...\" "c:\programfiles\...\tfile$_1$.txt"
7	$I_2$1, $I_2$2, P, P	{SearchDirectoryTP, ReadFileTP}	"c:\programfiles\...\" "c:\programfiles\...\tfile$_2$.exe"
8	$I_2$2, $I_2$2, A, A	{WriteFileTP, SetValueTP}	"c:\programFiles\...\tfile$_2$.exe" "hklm\Software\...\...\key$_1$"
9	$I_3$1, P	{ReadFileST}	"c:\windows\...\stfile$_1$.dll" "c:\windows\...\stfile$_2$.dll"

Table 7: Feature vector.

Features	*F*1	*F*2	*F*3	*F*4	*F*5	*F*6	*F*7	*F*8	*F*9
Program 1	1 3	1 3	1 3	1 3	1 2	2 3	1 5	1 1	0 0
...	–	–	–	–	–	–	–	–	–
Program *n*	—	—	—	—	—	—	—	—	—

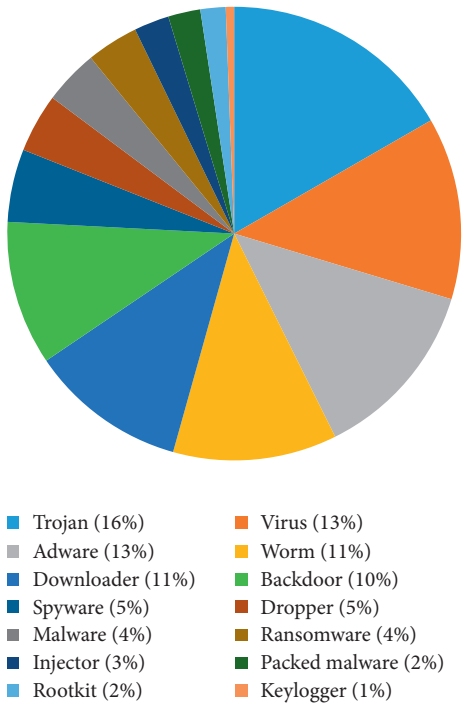


Figure 4: Distribution of analyzed malware.

generates a vector consisting of a set of features uniquely identifying the malware. Each feature is a combination of malware behaviors that have been determined by system calls to the operating system. Our model differentiates each system call and where the system call has occurred. The proposed model considers only features that can discriminate malware from benign samples.

5.3. Differentiate Malicious Behavior Patterns from Benign. During the detection process, the SCBM specifies malicious behavioral patterns, which can be seen frequently in malware but rarely seen in benign samples. To do that the algorithms in Section 4 have been used. To specify the malicious behavior patterns, following procedures are taken into consideration:

(1) The behaviors and the system paths where sample program performed are identified

(2) Scores are calculated for each behavior

(3) Behavior that could not exceed the specified score is removed from the list

(4) Behavior groups are determined according to the order of the selected behaviors

(5) Classification is performed according to the frequency of selected behaviors

By using these procedures, someone can easily separate malicious behavior patterns from benign even if malware and benign samples system calls are the same (in real examples, this is not the case). Example real features from our dataset and their frequencies are shown in Table 8. It can be clearly seen in Table 8 that someone can easily differentiate malware and benign samples by grouping to frequencies and level of frequencies. One way to do that is group to frequencies by numbering {0}, {1 to 20}, {21 to 100}, {101 to 200}, {201 to 300}, and {300+} and using decision tree for classification.

5.4. Machine Learning and Detection. Machine learning (ML) algorithms have been used to discriminate malware from benign samples. Even though ML algorithms have been used in many different areas for a long time, they have not been used sufficiently in malware detection. Thus, in this study, the most appropriate algorithms were used including Bayesian network (BN), naive Bayes (NB), decision tree variant (C4.5-J48), logistic model trees (LMT), random forest (RF), k-nearest neighbor (KNN), multilayer perceptron (MLP), simple logistic regression (SLR), and sequential minimal optimization (SMO). It cannot be concluded that one algorithm is more efficient than the others because each algorithm has its own advantages and disadvantages. Each algorithm can perform better than other algorithms under certain distributions of data, numbers of features, and dependencies between properties.

NB does not return good results due to calculation on assumptions that are not very related to each other, and BN is not practically applicable for data sets with many features. On our dataset, performance of these two algorithms was lower

Table 8: Created dataset and its features.

Class, name, RegOpenKeyTP, RegQueryValueTP, RegSetInfoKeyTP
Malware, f2ec3cbe4d3840b9b11d3b4052ee2dc7.exe, 760,0,508
Benign, cmd.exe,15,14,0
Malware, f2f72360bada04cb04a148334fb9b4f0.exe,67,48,48
Benign, calc.exe,62,59,9
Malware, f3a61848058d68097e7948cc3662963f.exe,546,701,305
Benign, notepad.exe,31,48,4
Malware, f3ab8adddce6730b1ee494e59ca88d70.exe,321,312,213
Benign, services.exe, 103,84,0
Malware, f3aa954ad390fc6be0be4c89120138e0.exe,498,557,342
Benign, taskhost.exe,0,0,6

Features and number of rows are shortened.

than other ML algorithms. However, some satisfying results have been measured in the literature. SVM and SMO work well in both linear separation and nonlinear boundary situations depending on the kernel used and performs well on high-dimensional data, but the desired performance measurements could not gather on the data sets generated. However, the SVM and SMO perform better than NB and BN. KNN algorithm requires a lot of storage space, and MLP algorithm requires long calculation time during the learning phase. These 2 deficiencies reduce the efficiency of these 2 algorithms. However, KNN performance was much higher than NB and BN performance. Although the fact that the SLR algorithm is inadequate to solve nonlinear problems and contains high bias decreases the efficiency of the algorithm, it has returned good results on the data sets created with the proposed model. On the contrary, decision trees produce scalable and highly accurate results, and they are the best performing classifiers according to test results on our dataset makes these classifiers more prominent than other classifiers. In the literature, except in some cases, they have returned satisfying results as well.

5.5. Model Performance and Evaluation. To evaluate the performance of the ML algorithms, DR, FP rate, f-measure, and accuracy were used. These values are calculated using the confusion matrix (Table 9).

These values are represented by the TP (the number of malicious software being marked as malicious), TN (the number of benign software being marked as normal), FP (the number of benign software being mistakenly marked as malicious), and FN (the number of malicious software accidentally being marked as benign). By using these values, DR, FPR, f-measure, and accuracy are calculated as

$$\begin{aligned} \mathrm{DR} = \mathrm{recall} &= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \\ \mathrm{FPR} &= \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}, \\ F - \mathrm{measure} &= \frac{2 * \mathrm{precision} * \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}, \\ \mathrm{accuracy} &= \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \end{aligned} \tag{3}$$

To evaluate the model and ML performance, holdout, cross-validation, and bootstrap have been used widely. For small datasets, cross-validation is a preferable method because the model performs better on previously unknown data, while the holdout method is useful for large datasets because the system can be trained with enough instances.

In this study, both the holdout and cross-validation methods were used to evaluate performance. At the beginning, when the dataset was small, cross-validation returned better results. However, when the dataset had grown, the holdout method also generated favorable results.

6. Results and Discussion

The summarized test results can be seen in Tables 10–14 and Figures 5 and 6. The test results show the DR, FPR, and accuracy on *n*-gram and proposed models. The both holdout and cross-validation methods perform well on the proposed model. Thus, when evaluating a model performance, the combination of 10-fold cross validation and percentage split (75% training and 25% testing) for holdout results are used. Similar results were obtained when parameters are changed. Table 10 shows the comparison of the classification algorithms on the SCBM and *n*-gram model that were used to build the dataset.

In Table 10, 400 malware and 300 benign portable executables are tested. In almost all cases, the proposed model achieved better results than 4-gram; similar results were obtained using 2-gram, 3-gram, and 6-gram. For instance, the SLR algorithm performance on 4-gram is measured as 94.6% for DR, 6.3% for FPR, and 94.5% for accuracy; versus SCBM performance is measured as 98.5% for DR, 4% for FPR, and 97.2% for accuracy. In the same way, J48 algorithm achieved 91.4% for DR, 9.1% for FPR, and 91% for accuracy when using 4-gram; and versus 99.5% for DR, 0.7% for FPR, and 99.4% for accuracy when using SCBM. Other classification algorithms achieved similar results on the *n*-gram and SCBM datasets, which shows that the proposed model's results are much better than those of the *n*-gram models. The *n*-gram uses consecutive system calls whether related or not from properties. This causes malware features to grow significantly, which increases the training time and makes the detection processes challenging.

The test results with and without scores can be seen in Tables 11 and 12 when 1000 program samples have been analyzed. The both datasets without score and with score have been created by using the proposed model. However, the dataset with score contains far less features than dataset without score. Thus, after 1000 programs have been analyzed, we have only continued to analyze programs for dataset with score.

Decision tree classifiers (J48, LMT, and RF) give better results than other classifiers such as SMO, KNN, BN, and NB (Tables 11–13). For example, in J48, DR, FPR, and accuracy were measured as 99.1%, 1.2%, and 99.2%, respectively (Table 12). The test results also indicate that SLR performs better than SMO, KNN, BN, and NB. However, KNN is slightly better than SMO in terms of FPR and accuracy. SMO

Table 9: Confusion matrix.

Actual class	Predicted class	
	Yes	No
Yes	TP	FN
No	FP	TN

Table 10: Comparison of n-gram and the proposed model (400 malware and 300 benign).

Model	Classifier	DR (%)	FPR (%)	Acc. (%)
4-gram	J48	91.4	9.1	91
	LMT	97.7	2.4	97.4
	RF	85.1	18.8	85
	SLR	94.6	6.3	94.5
	SMO	92	9.6	92.1
	KNN	87	16.2	87.3
	BN	—	—	—
	NB	86.7	16.4	87
Proposed model	J48	99.5	0.7	99.4
	LMT	98.6	1.5	98.4
	RF	96.1	4.9	96
	SLR	98.5	4	97.2
	SMO	97.4	2.4	97.3
	KNN	87.4	13.6	87.7
	BN	86.6	12.8	86.5
	NB	75.8	20	75.5

Table 11: Classifiers results on the proposed model without score (700 malware and 300 benign).

Classifier	DR (%)	FPR (%)	F-score (%)	Acc. (%)
J48	98.9	1.6	98.9	99
LMT	97.4	1.5	97.4	97.4
RF	93.9	8.2	92.2	94
SLR	97.3	1.4	97.3	97.3
SMO	89.8	12.1	89.9	90
KNN	88.3	7.3	88.5	88.4
BN	85.1	12.9	85.5	85
NB	78.3	14.1	79	78.4

Table 12: Classifiers results on the proposed model with score (700 malware and 300 benign).

Classifier	DR (%)	FPR (%)	F-score (%)	Acc. (%)
J48	99.1	1.2	99.1	99.2
LMT	98.1	1.8	98.1	98
RF	95.7	6.5	95.7	96
SLR	97.3	2.2	97.4	97.4
SMO	92	7.9	92.1	92
KNN	92	9.7	92.1	92.2
BN	88.3	9	88.6	88.4
NB	73.5	16.8	74.5	74

performs better than BN and NB. NB shows lower performance than other classifiers. Thus, NB is not an appropriate classifier for our dataset. MLP was too slow to classify malware and benign samples in both the n-gram dataset and the proposed method. Thus, it was not included in the test results.

Table 13: Classifiers results on the proposed model with score (6700 malware and 3000 benign).

Classifier	DR (%)	FPR (%)	F-score (%)	Acc. (%)
J48	99.9	0.2	99.8	99.8
LMT	99.9	0.1	99.9	99.87
RF	99.8	0.4	99.8	99.82
SLR	97.4	2	97.5	97.4
SMO	93.2	6.8	93.2	93.1
KNN	99.6	0.8	99.6	99.62
BN	89	8.6	89.3	89
NB	75.6	15.3	76.5	75.62

Table 14: Comparison of classifiers from different studies.

Paper	Classifier	DR (%)	FPR (%)	Acc. (%)	Year
Firdausi et al. [43]	NB	58.1	12.8	65.4	2010
	J48	90.9	3.8	93.6	
Ye et al. [25]	NB	63.3	—	50.2	2010
	SVM	84.5	—	83.4	
	J48	56.8	—	57.3	
Islam et al. [16]	SVM	—	14	84.3	2013
	RF	—	10.4	87.8	
Santos et al. [44]	KNN K = 2	—	14	90.7	2013
	J48	—	9	91.2	
	NB	—	31	79.6	
Yousefi-Azar et al. [45]	SVM	95	5.07	93.4	2018
	RF	93.2	6.82	90.1	
	KNN	90	10	91.2	
Proposed method	J48	99.9	0.2	99.8	2019
	SLR	97.4	2	97.4	
	SMO	93.2	6.8	93.1	
	KNN	99.6	0.8	99.62	
	NB	75.6	15.3	75.62	

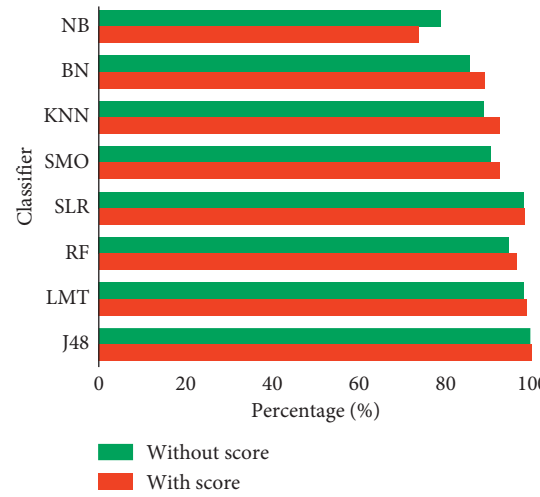


Figure 5: Comparison of classification algorithms accuracy on the suggested model with and without scores.

The DRs and accuracies are increased when the number of analyzed programs are increased, while FPRs are decreased (Tables 12 and 13). This shows that the proposed

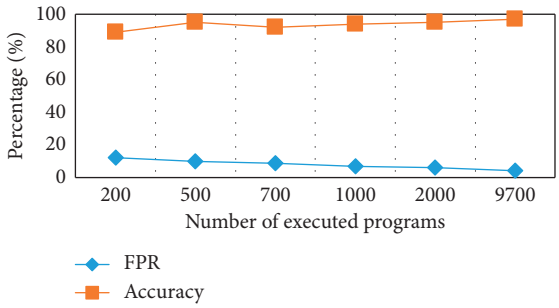


Figure 6: Average accuracy and FPR of classifiers versus the number of analyzed programs.

model successfully differentiates malicious from benign patterns. However, the n-gram was too slow when the analyzed programs increase. Hence, we stopped to analyze more programs to create dataset with n-gram.

The test results also indicate that the proposed model with score-specified malware properties is better than the proposed model without score (Figure 5). The average classification accuracy (cross-validation and holdout split by 75/25%) can be seen in Figure 5, which shows the accuracy of the classifiers on the dataset with and without scores. It can be clearly seen that, with the exception of the NB, all classifiers performed much better when the scoring system was used.

We have concluded that using the scoring schema for our dataset eliminated less important features for discriminating malware from benign samples. This is because the SCBM model with score also works as a feature selection algorithm and metric which produce better performance. Feature selection algorithms use dependency, accuracy, distance, and information measures such as information gain and gain ratio to select more important features from the dataset. The dataset with score outperformed the dataset without score, which uses feature selection algorithms and metrics. Thus, there is no need to use a feature selection algorithm for most of the classifiers before classification. Since decision tree classifiers use a feature selection algorithm by default (feature selection and tree pruning), the classification algorithm difference is low (Figure 5). For example, J48 accuracy is 99.2% with score and 99% without score, LMT accuracy is 98% with score and 97.4% without score, and RF accuracy is 96% with score and 94% without score. However, SMO accuracy is 92% with score and 90% without score and KNN accuracy is 92.2% with score and 88.4% without score. Thus, providing fewer but more meaningful features for classification produces better results. It can also be concluded that using the feature selection algorithm for the dataset without scoring for some classifiers may increase the detection and accuracy rates.

To evaluate the proposed model more accurately, different numbers of malware and benign samples were tested. Figure 6 shows the average accuracy rate and FPR when the number of analyzed programs increase. The classification accuracy increases when the number of analyzed programs increase while FPR decreases for all ML algorithms that have been used including J48, LMT, RF, SLR, SMO, KNN, BN, and NB. For example, when 200 programs were analyzed, the accuracy rate was 89%. This accuracy increases over time when more programs are analyzed, up to 94%, 95.3%, and 97% (Figure 6). However, FPR decreased sharply when more programs were analyzed. FPR was 12% at the beginning, but overtime, it decreased to 9.7%, 5.9%, and 4.1%. Based on the test results, it can be concluded that the classifier results improve when more programs are analyzed.

To evaluate the efficiency of the proposed model, DR, FPR, and accuracies are also compared with different models from the literature (Table 14). The proposed model produces considerably better results than other models [16, 43, 45] when the same classifier is used for evaluation. For instance, when J48 is used as a classifier; the DR, FPR, and accuracies are measured as 99.9%, 0.2%, and 99.8%, respectively, for the proposed model, while 90.9%, 3.8%, and 93.6% for the model from [43] (Table 14). For other classifiers, the proposed model also performed better than other models. The worst result was obtained for NB (75.6% DR, 15.3% FPR, and 75.62% accuracy for the proposed model), while DR of 58.1% was obtained for the model in [43] and an FPR of 31% was obtained for the model in [44]. Even if our result was fairly low when using the NB classifier, it was still better than those of other works in the literature.

Furthermore, some important findings were found during analysis. These findings should be considered when creating an effective detection system. The key findings of the analysis are listed as follows:

(i) Most of the new generation malware uses existing processes or newly created processes for malicious purposes

(ii) New generation malware tries to hide itself by creating similar systems and third-party software files

(iii) Most malware creates malicious behaviors in temporary file paths

(iv) Malware usually tries to become permanent in the system by locating itself within Windows automatic startup locations

(v) Some malware displays the actual behaviors only when it runs with administrator-level authority

(vi) Most malware creates random files (using meaningless file names)

(vii) Most new generation malware injects itself into Windows system files ("svchost.exe," "winlogon.exe," and "conhost.exe") or copies itself into different file paths with the same or similar names

(vii) Some malware tries to find and disable existing security software (firewall and antivirus program) as soon as it is performed

7. Limitations and Future Works

Even though SCBM is fast and efficient to detect malware, there are some limitations needed to be mentioned. The proposed model has been tested on uniformly distributed

dataset, more zero-day malware need to be tested. The test cases for malware is performed on virtual machines which can represent limited behaviors of malware [46]. Thus, running malware on real machine can improve the performance. Besides, suggested schema only tested on our dataset, if raw data of other datasets will be gathered, in the future suggested schema will be tested on other datasets as well. The suggested schema will be integrated with other technologies such as cloud, blockchain, and deep learning to build more powerful detection system [46].

8. Conclusion

The SCBM is presented. In the SCBM, malware behaviors and system paths, where malware behaviors are performed, are considered. Features that could not exceed the specified score are removed from the dataset. This way malicious behavior patterns were differentiated from benign behavior patterns. Therefore, datasets created using the proposed model contained far fewer features than datasets created by *n*-gram. To evaluate the performance, the proposed model was combined with an appropriate ML algorithm. The test results showed that the proposed model outperformed *n*-gram and some models used in other studies. For the proposed model, DR, FPR, and accuracies were 99.9%, 0.2%, and 99.8%, respectively, which are higher than those of *n*-gram and other methods.

The test results also indicated that decision tree classifiers (J48, LMT, and RF) and SLR yield better results than classifiers such as SMO, KNN, BN, and NB. BN and NB show lower performance than other classifiers, which show that BN and NB are not appropriate classifiers. It can be concluded that the proposed method combined with appropriate ML algorithms has outperformed signature-based detection method, *n*-gram model, and other behavior-based detection methods. The proposed model has performed effectively for known and unknown malware.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this article.

References

[1] E. Masabo, K. S. Kaawaase, J. Sansa-Otim, J. Ngubiri, and D. Hanyurwimfura, "A state of the art survey on polymorphic malware analysis and detection techniques," *ICTACT Journal of Soft Computing*, vol. 8, no. 4, 2018.

[2] S. Morgan, "Cybersecurity almanac: 100 facts, figures, predictions and statistics. Cisco and cybersecurity ventures," 2019, https://cybersecurityventures.com/cybersecurity-almanac-2019.

[3] R. Samani and G. Davis, "McAfee mobile threat report Q1," 2019, https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf.

[4] Symantec, *Internet Security Threat Report (ISTR)*, Vol. 23, Symantec, Mountain View, CA, USA, 2018.

[5] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: a user-oriented behavior-based malware variants detection system for android," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1103–1112, 2017.

[6] D. Emre and R. Samet, "A new model for secure joining to ZigBee 3.0 networks in the internet of things," in *Proceedings of the 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, pp. 3-4, IEEE, Ankara, Turkey, December 2018.

[7] O. Aslan and R. Samet, "Investigation of possibilities to detect malware using existing tools," in *Proceedings of the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications*, pp. 1277–1284, IEEE, Hammamet, Tunisia, October 2017.

[8] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *Proceedings of the 5th Conference on Information and Knowledge Technology*, May 2013.

[9] K. M. Alzarooni, "Malware variant detection," *Doctoral dissertation*, University College London, London, UK, 2012.

[10] O. Aslan and R. Samet, "Mitigating cyber security attacks by being aware of vulnerabilities and bugs," in *Proceedings of the 2017 International Conference on Cyberworlds (CW)*, pp. 222–225, Chester, UK, September 2017.

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

[12] S. Spencer, "Timeline of computer viruses," 2019, https://www.mapcon.com/us-en/timeline-of-computer-viruses.

[13] "History of Malware," 2019, https://www.gdatasoftware.com/seccuritylabs/information/history-of-malware.

[14] G. Wagener, R. State, and A. Dulaunoy, "Malware behaviour analysis," *Journal in Computer Virology*, vol. 4, no. 4, pp. 279–287, 2008.

[15] Y. Park, D. S. Reeves, and M. Stamp, "Deriving common malware behavior through graph clustering," *Computers & Security*, vol. 39, pp. 419–430, 2013.

[16] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[17] S. Naval, V. Laxmi, M. Rajarajan, M. S. Gaur, and M. Conti, "Employing program semantics for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2591–2604, 2015.

[18] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: towards efficient real-time protection against malware," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 289–302, 2016.

[19] H. Zhang, W. Zhang, Z. Lv, A. K. Sangaiah, T. Huang, and N. Chilamkurti, "MALDC: a depth detection method for malware based on behavior chains," *World Wide Web*, pp. 1–20, 2019.

[20] Z. Shan and X. Wang, "Growing grapes in your computer to defend against malware," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 2, pp. 196–207, 2014.

[21] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-Y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proceedings of the 2009 USENIX Security Symposium*, vol. 4, no. 1, pp. 351–366, Montreal, Canada, August 2009.

[22] Y. Fukushima, A. Sakai, Y. Hori, and K. Sakurai, "A behavior based malware detection scheme for avoiding false positive," in *Proceedings of the 2010 6th IEEE Workshop on Secure Network Protocols*, pp. 79–84, IEEE, Kyoto, Japan, October 2010.

[23] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "AccessMiner: using system-centric models for malware protection," in *Proceedings of the 2010 ACM Symposium on Information, Computers and Communications Security*, pp. 399–412, Chicago, IL, USA, October 2010.

[24] M. Chandramohan, H. B. K. Tan, L. C. Briand, L. K. Shar, and B. M. Padmanabhuni, "A scalable approach for malware detection through bounded feature space behavior modeling," in *Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 312–322, IEEE, Silicon Valley, CA, USA, November 2013.

[25] Y. Ye, T. Li, Q. Jiang, and Y. Wang, "CIMDS: adapting postprocessing techniques of associative classification for malware detection," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 3, pp. 298–307, 2010.

[26] H. H. Pajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "Intelligent OS X malware threat detection with code inspection," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 213–223, 2018.

[27] S. T. Liu, H. C. Huang, and Y. M. Chen, "A system call analysis method with MapReduce for malware detection," in *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pp. 631–637, IEEE, Tainan, Taiwan, Decemeber 2011.

[28] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," 2009, http://www.eurecom.fr/seminar/3832.

[29] B. Alsulami, A. Srinivasan, H. Dong, and S. Mancoridis, "Light-weight behavioral malware detection for windows platforms," US Patent Application No. 16112825, 2019.

[30] C. Choi, C. Esposito, M. Lee, and J. Choi, "Metamorphic malicious code behavior detection using probabilistic inference methods," *Cognitive Systems Research*, vol. 56, pp. 142–150, 2019.

[31] E. B. Karbab and M. Debbabi, "MalDy: portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports," *Digital Investigation*, vol. 28, pp. S77–S87, 2019.

[32] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep auto encoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[33] Q. K. Ali Mirza, I. Awan, and M. Younas, "CloudIntell: an intelligent malware detection system," *Future Generation Computer Systems*, vol. 86, pp. 1042–1053, 2018.

[34] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[35] M. Sikorski and A. Honig, *Pratical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, No Starch Press, San Francisco, CA, USA, 2012.

[36] "Open malware benchmark, malware downloading website," 2019, http://malwarebenchmark.org/.

[37] "ViruSign, malware downloading website," 2019, http://www.virusign.com/.

[38] "Mal share, malware downloading website," 2019, https://malshare.com.

[39] D. B. Malware, "Open malware, malware downloading website," 2019, https://www.openmalware.org.

[40] "KernelMode, malware downloading website," 2019, https://www.kernelmode.info.

[41] "Tekdefense, malware downloading website," 2019, http://www.tekdefense.com/downloads/.

[42] "Virustotal, malware scanning service website," 2019, https://www.virustotal.com.

[43] I. Firdausi, C. Lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *Proceedings of the 2010 2nd International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 201–203, Jakarta, Indonesia, December 2010.

[44] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64–82, 2013.

[45] M. Yousefi-Azar, L. G. C. Hamey, V. Varadharajan, and S. Chen, "Malytics: a malware detection scheme," *IEEE Access*, vol. 6, pp. 49418–49431, 2018.

[46] O. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.