Research Article

Using a Subtractive Center Behavioral Model to Detect Malware

Ömer Aslan ^{1,2}, Refik Samet ¹, and Ömer Özgür Tanrıöver ¹

¹Ankara University, Computer Engineering Department, Ankara 06830, Turkey

²Siirt 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

Copyright © 2020 Ömer Aslan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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-based malware

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 $(P_1 \rightarrow P_2; P_1 \rightarrow P_3; P_3 \rightarrow P_4)$ 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

FIGURE 2: New generation malware.

difficult to detect malware, which uses the oligomorphic method rather than encryption.

- (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 counter techniques 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 anti-virus 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, $M_{\text{pattern}} = \{ab, ce\}$, while $B_{\text{pattern}} = \{\}$, 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)  $d_1 \leftarrow \text{file}_1, d_2 \leftarrow \text{file}_2, n \leftarrow u(d_1)$ 
(2) for  $i \leftarrow 1$  to  $n$ 
(3)   if ( $d_{1[i][\text{state}]} == \text{'AE'}$ )
(4)      $\psi \leftarrow A$ 
(5)   else
(6)      $\psi \leftarrow P$ 
(7)   end if
(8)   if ( $P.\text{name} == d_1.\text{FileName}$ )
(9)      $\mu \leftarrow \text{self}$ 
(10)  elif ( $eST == \text{'ss'}$ )
(11)     $\mu \leftarrow \text{system}$ 
(12)  elif ( $eST == \text{'ts'}$ )
(13)     $\mu \leftarrow \text{thirdParty}$ 
(14)  else
(15)     $l = 1$ 
(16)  end if
(17)  if ( $d_{1[i-1][o]} != d_{1[i][o]}$ )
(18)    if ( $d_{1[i][o]} != \text{rcK} \ \&\& \ d_{1[i][o]} != \text{cF} \ \&\& \ d_{1[i][o]} != \text{tE} \ \&\& \ d_{1[i][o]} != \text{pE}$ )
(19)      if ( $d_{1[i-1][s]} == d_{1[i][s]}$ )
(20)         $\text{Write}.d_2()$ 
(21)      end if
(22)    end if
(23)  end if
(24)  if ( $d_{1[i-1][o]} = d_{1[i][o]}$ )
(25)    if ( $d_{1[i-1][sfp]} != d_{1[i][sfp]}$ )
(26)      if ( $d_{1[i][o]} != \text{rcK} \ \&\& \ d_{1[i][o]} != \text{cF} \ \&\& \ d_{1[i][o]} != \text{tE} \ \&\& \ d_{1[i][o]} != \text{pE}$ )
(27)         $\text{write}.d_2()$ 
(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 d_2 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 $S = \{a, b, c, d, \dots, n\}$, where S represents the system-call sequence and $a, b, c, \dots, n$ represent each system call. Only $s \in S$ 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 $y_1, y_2, \dots, y_n$ behaviors. The risk score for feature path (rsP) is calculated for x_i as follows:

$$x_i(\text{rsP}) = \frac{y_1(\text{rsP}) + y_2(\text{rsP}) + \dots + y_n(\text{rsP})}{n} \quad (1)$$

Let a be a specified threshold value, if $x_i(\text{rsP}) \geq a$, 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 (“tfile1.txt,” malware.exe → 4) and ReadFile (“tfile2.exe,” malware.exe → 5) create two different properties (Table 4), while ReadFile (13, “stfile1.dll,” tfile2.exe → 16) and ReadFile (14, “stfile2.dll,” tfile2.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, “tfile2.exe,” sfile1.exe → 9) and SetValue (10, “key1,” tfile2.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)  $d_2 \leftarrow \text{file}_2, d_3 \leftarrow \text{file}_3, n \leftarrow u(d_2)$ 
(2) for  $i \leftarrow 1$  to  $n$ 
(3)   if ( $\mu = \text{'self'}$ )
(4)     if ( $P_{\text{name}} = d_{2\text{-fileName}}$ )
(5)        $pRS \leftarrow 0$ 
(6)     elif ( $P_{\text{name}} \neq d_{2\text{-fileName}} \ \&\& \ d_{2\text{-fileName}} = rD$ )
(7)        $pRS \leftarrow 3$ 
(8)     else
(9)        $pRS \leftarrow 2$ 
(10)    end if
(11)  elif ( $\mu = \text{'ts'}$ )
(12)    if ( $d_{2[i][fp]} = tY$ )
(13)       $pRS \leftarrow 2$ 
(14)    # Registry Autostart Location
(15)    elif ( $d_{2[i][fp]} = aS$ )
(16)       $pRS \leftarrow 3$ 
(17)    else
(18)       $pRS \leftarrow 0$ 
(19)    end if
(20)  elif ( $\mu = \text{'ss'}$ )
(21)    if ( $P_{\text{name}} = d_{2\text{-fileName}}$ )
(22)       $pRS \leftarrow 0$ 
(23)    elif ( $d_{2\text{-fileName}} = \text{'*.exe'}$ )
(24)       $pRS \leftarrow 3$ 
(25)    elif ( $d_{2[i][sfp]} = sRY$ )
(26)       $pRS \leftarrow 3$ 
(27)    elif ( $d_{2[i][sfp]} = rD$ )
(28)       $pRS \leftarrow 3$ 
(29)    else
(30)       $pRS \leftarrow 0$ 
(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)  $d_2 \leftarrow \text{file}_2, d_4 \leftarrow \text{file}_4, n \leftarrow u(d_2)$ 
(2) for  $i \leftarrow 1$  to  $n$ 
(3)   if  $(i < n - 10)$ 
(4)     for  $j \leftarrow i + 1$  to  $i + 10$ 
(5)        $\text{fP}_2 = d_{2[j][\text{sfP}]}$ 
(6)       if  $(P_1.\text{as} == P_2.\text{as} \ \&\& \ P_1.\text{as} == 'A')$ 
(7)          $\psi \leftarrow 'AA'$ 
(8)       elif  $(P_1.\text{as} == P_2.\text{as} \ \&\& \ P_1.\text{as} == 'P')$ 
(9)          $\psi \leftarrow 'PP'$ 
(10)      else
(11)         $\psi \leftarrow 'AP' = 'PA'$ 
(12)      end if
(13)      if  $(d_{2[j][o]} == \text{rdF} \ \&\& \ d_{2[j+1][o]} == \text{weF})$ 
(14)         $\pi \leftarrow O_1 + ' ' + O_2$ 
(15)      if  $(d_{2[j][\mu]} == 'self')$ 
(16)         $\pi \leftarrow \pi + 'S'$ 
(17)      elif  $(d_{2[j][\mu]} == 'ts')$ 
(18)         $\pi \leftarrow \pi + 'TP'$ 
(19)      elif  $(d_{2[j][\mu]} == 'ss')$ 
(20)         $\pi \leftarrow \pi + 'ST'$ 
(21)      else
(22)         $2 \leftarrow 2$ 
(23)      end if
(24)       $\text{write}.d_4()$ 
(25)      if  $(\text{sfP}_1 == \text{sfP}_2 \ \&\& \ O_1 \neq O_2)$ 
(26)         $\pi \leftarrow O_1 + ' ' + O_2$ 
(27)        if  $(d_{2[j][\mu]} == 'self')$ 
(28)           $\pi \leftarrow \pi + 'S'$ 
(29)        elif  $(d_{2[j][\mu]} == 'ts')$ 
(30)           $\pi \leftarrow \pi + 'TP'$ 
(31)        elif  $(d_{2[j][\mu]} == 'ss')$ 
(32)           $\pi \leftarrow \pi + 'ST'$ 
(33)        else
(34)           $2 \leftarrow 2$ 
(35)        end if
(36)         $\text{write}.d_4()$ 
(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_12 , A can be evaluated as a related process trying to make changes on its files by using active behaviors, while in I_31 , 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_12, A	{CreateFileSF}	"c:\windows\...\sfile ₁ .exe"
2	I_12, A	{WriteFileSF }	"c:\windows\...\sfile ₁ .exe"
3	I_12, I_12, A, P	{WriteFileSF, ReadFileSF}	"c:\windows\...\sfile ₁ .exe" "c:\windows\...\sfile ₁ .exe"
4	I_12, I_22, P, A	{ReadFileSF, WriteFileTP}	"c:\windows\...\sfile ₁ .exe" "c:\programfiles\...\tfile ₂ .exe"
5	I_22, P	{SearchDirectoryTP}	"c:\programfiles\..."
6	I_22, I_22, P, P	{SearchDirectoryTP, ReadFileTP}	"c:\programfiles\..." "c:\programfiles\...\tfile ₁ .txt"
7	I_21, I_22, P, P	{SearchDirectoryTP, ReadFileTP}	"c:\programfiles\..." "c:\programfiles\...\tfile ₂ .exe"
8	I_22, I_22, A, A	{WriteFileTP, SetValueTP}	"c:\programfiles\...\tfile ₂ .exe" "hklm\Software\...\key ₁ "
9	I_31, P	{ReadFileST}	"c:\windows\...\stfile ₁ .dll" "c:\windows\...\stfile ₂ .dll"

TABLE 7: Feature vector.

Features	F1	F2	F3	F4	F5	F6	F7	F8	F9
Program 1	1	1	1	1	1	2	1	1	0
	3	3	3	3	2	3	5	1	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, RegSetValueKeyTP
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, f3ab8addce6730b1ee494e59ca88d70.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

$$\begin{aligned}
 DR = \text{recall} &= \frac{TP}{TP + FN}, \\
 FPR &= \frac{FP}{FP + TN}, \\
 F - \text{measure} &= \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}, \\
 \text{accuracy} &= \frac{TP + TN}{TP + TN + FP + FN}.
 \end{aligned} \tag{3}$$

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	—	14	90.7	2013
	K = 2	—	—	—	
	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
- (viii) 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/securitylabs/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, December 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, *Practical 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] "VirusSign, 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.