Hindawi
Security and Communication Networks
Volume 2018, Article ID 4947695, 8 pages
https://doi.org/10.1155/2018/4947695

Research Article

Automatic Benchmark Generation Framework for Malware Detection

Guanghui Liang,[1] **Jianmin Pang**,[1] **Zheng Shan**,[1] **Runqing Yang**,[2] **and Yihang Chen**[1]

[1]*State key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China*
[2]*Zhejiang University, Hangzhou, China*

Correspondence should be addressed to Jianmin Pang; jianmin_pang@126.com

Received 31 December 2017; Accepted 16 July 2018; Published 6 September 2018

Academic Editor: Alessandro Cilardo

To address emerging security threats, various malware detection methods have been proposed every year. Therefore, a small but representative set of malware samples are usually needed for detection model, especially for machine-learning-based malware detection models. However, current manual selection of representative samples from large unknown file collection is labor intensive and not scalable. In this paper, we firstly propose a framework that can automatically generate a small data set for malware detection. With this framework, we extract behavior features from a large initial data set and then use a hierarchical clustering technique to identify different types of malware. An improved genetic algorithm based on roulette wheel sampling is implemented to generate final test data set. The final data set is only one-eighteenth the volume of the initial data set, and evaluations show that the data set selected by the proposed framework is much smaller than the original one but does not lose nearly any semantics.

1. Introduction

Malicious code, or malware, is one of the most pressing security problems on the Internet. Over 100 million of new malware samples are identified every year. The malware research community has conducted exhaustive studies on malware analysis to improve detection accuracy [1–7].

Proposed approaches are expected to achieve the highest possible detection accuracy. However, we cannot determine that Approach A with 97.2% detection accuracy is definitively better than Approach B with 89% accuracy because the two approaches are evaluated based on completely different test data sets. The number of samples in the experimental stage ranges from hundreds to the thousands, and the composition of the data set is rarely public.

Several benchmarks, such as Standard Performance Evaluation Corporation (SPEC) and NAS Parallel Benchmarks (NPB), are widely used in high-performance computing to evaluate the performance of supercomputers [8, 9]. These benchmarks are continuously developed by organizations encompassing diverse groups, and the benchmarks are updated with advances in supercomputer technology. However, in the malware research field, malware writers are known to develop and employ new obfuscation and anti-analysis techniques, such as packers, self-modification, and environment check, and standard benchmarks based on labor-intensive manual selection may not be able to keep pace with malware development. In this environment, older malware data sets cannot be used to assess the capability of modern malware.

A few types of malware data sets are frequently used. The first type is used by some third-party evaluation organizations; this type includes AV-TEST and AV-Comparatives [10, 11]. These data sets test the performance of antivirus software by providing a large number of malware samples. According to published reports, these samples are selected by their analysts, who claim that these malware samples can be considered representative of modern malware. Unfortunately, these malware samples are not publicly available for academic research. Another category of data set is used in academic fields, usually presented as a small collection of malware samples for research purposes. These samples may be downloaded online or shared by other people. Some researchers apply the data sets during experiments to evaluate their detection technique, although the data set may contain only one or two malicious categories that may

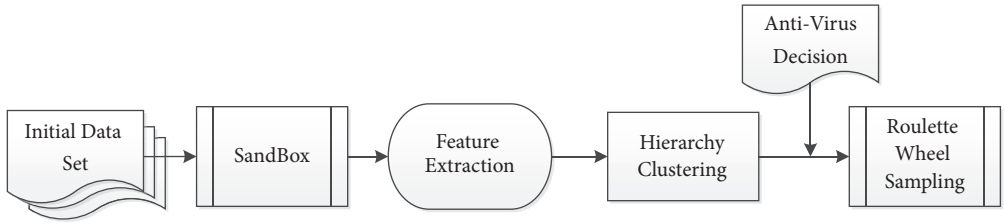

Figure 1: Overview of the proposed framework.

not represent mainstream malware. Many websites provide malware sample downloads for research, such as VX Heaven and VirusShare [12, 13]. However, generating a ground truth data set for malware detection remains challenging.

The absence of an open-source standard ground truth data set continues to hinder the comparison of malware detection and analysis methods. Ideally, we should use as many malware samples as possible in training and testing experiments. However, applying a huge number of malware samples in training and testing is too complicated to implement in practice. Therefore, a standard malware test data set is essential to evaluate different malware detection techniques.

One approach to this problem is to extract a small portion of malware samples that could be considered representative of the vast number of existing malware samples. Manual selection is precise but labor intensive. Therefore, an automatic framework for malware benchmark generation is crucial. The diversity of malware samples in the benchmark data set should be guaranteed. The input of the framework is a large initial raw data set that includes a large number of malware samples, as detailed in Section 5. The initial data set is then sent to a virtual environment called a sandbox for behavior monitoring. The system calls are extracted during malware running, followed by sequence purity and behavior feature extraction to construct behavior profile. The distance between every malware pair is measured by Jaccard similarity. Hierarchical clustering then produces many clusters based on the malware behavior profiles. Every cluster is a set of samples that own similar malicious action. Finally, an improved roulette wheel sampling method selects valuable malware samples from clusters to generate the final data set, which we refer to as a benchmark data set in this paper. An overview of our framework is presented in Figure 1.

In particular, our paper makes the following contributions:

(1) We firstly propose a framework for generating a benchmark data set for malware detection research. With this framework, a more appropriate test data set can be obtained from raw data, compared with manual selection.

(2) We propose an improved roulette wheel sampling algorithm. Malware cluster labels and maliciousness degrees (MDs) are fused to determine whether or not a given sample should be selected. This algorithm can adapt to malware evolution.

(3) We propose the average entropy of data set to evaluate the diversity of the generated benchmark. Other metrics, such as detection accuracy and performance overhead, are also applied to test the effectiveness and efficiency of the proposed method.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 describes how to purify the system call sequence and obtain malware behavior features. Section 4 introduces the hierarchical clustering and roulette wheel sampling method to generate the benchmark. We evaluate the benchmark in Section 5 based on the entropy of the data set and other standard performance metrics.

2. Related Work

Sommer and Paxson tried to explain the gap between success in academia and actual deployments of intrusion detection systems. One of five reasons found in their study was the enormous variability of training data [14]. They explained that some academic detection systems could not get ideal accuracy rate in real deployments because the training data is not good enough. Rossow studied 36 academic publications from 2006 to 2011 and proposed that every researcher should adopt a prudent attitude toward designing malware experiments, especially with regard to the balance of data set collection [15]. The basis of a prudent experiment is a ground truth data set that can be accepted by other researchers. In 2015, Microsoft sponsored an online malware classification competition. 2,000 malware samples were provided in hexadecimal representation and binary content, without portable executable (PE) headers (to ensure sterility). All the samples comprised nine malware families and many research papers were published based on this data set. Unfortunately, this data set was static and could not be updated to accommodate new trends in malware development [16].

There are three approaches to generate data set for malware analysis. The first approach divides malware into different types according to its ability. Das downloaded 472 samples from VX Heaven and VirusShare to evaluate the Guardol system and classified these samples into backdoor, exploit, Trojan, virus, worm, rootkit, and flooder, according to labels returned from specific antivirus software [5]. It should be noted that there is no strict standard to distinguish whether malware is Trojan or backdoor in many cases. A Trojan may also own the ability of backdoor or rootkit.

The second approach collects samples with the help of antivirus labels. Shabtai used assembly opcode to detect unknown malware with a data set containing 7668 samples that were confirmed by Kaspersky antivirus software [19]. Naval realized the importance of data sets in the experimental

Table 1: Malware detection data sets used since 2010.

Author	Year	Quantity	Selection Standard
Pengli [17]	2010	400	Manual
Jason [7]	2015	85	Manual
Smita Naval [18]	2015	2525	Manual+anti-virus
Sanjeev Das [5]	2016	472	Malware type
Shabtai [19]	2012	7668	Kaspersky label
Zhao Z [20]	2013	4701	Manual
Youngjoon Ki [21]	2015	6000	Anti-virus label

stage; therefore, he used two different data sets and labeled them as Dold and Dnew, including 1209 and 1316 samples, respectively [18]. However, there is generally a lack of consensus in antivirus engines' decisions on a given sample which may get various labels by different antivirus [22].

The third approach is a manual selection which should be operated by some experienced experts. Json proposed a premium data set composed of only 85 samples that were selected manually by analysts and divided into eight categories, and subsequently used TLSH, BiSherd, and First Byte algorithms to evaluate this test set, with results that showed that every algorithm had a lower accuracy on the premium test set than the algorithm had previously exhibited [7]. Manual selection may be accurate but is labor extensive and not scalable

In Table 1, we present statistics on some data sets for malware detection in recent years. The table includes the author of the paper and the published year, the number of samples in the experimental data set, and the method used to select samples. The number of samples ranged from dozens to thousands, selected manually or by labels from antivirus software.

Wei and Li thought that some data sets not only were outdated and no longer represented the current malware landscape, but also did not provide the detailed information on the malware's behavior that was needed for research [23]. Our automatic benchmark generation framework is proposed based on the behavior of malware. The selection method could ensure the diversity of the malware data set and maintain it at a reasonable size.

3. Feature Extraction

Malware could be characterized in many ways by various malware features, such as disassembly instructions, system calls, and system log after running the malware. Dynamic analysis based on the virtual sandbox has become more popular because such methods are immune to malware packers and some types of obfuscation. In this section, we provide more details on the components of feature extraction. In particular, we first discuss how we monitor the behavior of program activity. Then, to cope with some potential confusion, sequence purification is applied on the dynamic trace. Finally, behavior feature is extracted by classical n-gram algorithm.

3.1. Behavior Monitoring. The behavior of malware can be characterized by its system calls, including process injection, memory allocation, network connection, and file operation. To monitor the behavior of executable binaries, many different methods could be applied, a majority of which are based on the interception of Windows API or native system calls. The binary is executed in a controlled environment, which is usually called a sandbox.

We construct a new dynamic malware analysis sandbox on top of the well-known dynamic instrumentation tool platform DynamoRIO, facilitating a customized analysis tool with fine-grained instrumentation support as well as high-level interfaces [24].

With the development of anti-analysis technology, malware writers are aware that almost all of the detection tools have been equipped with the capability to monitor API call paths via the API hooking technique, and therefore malware writers now use the native system calls rather than API calls to bypass API hooking. Our sandbox tracks and stores all system calls while the malware is running. Native system calls are closer to the system kernel than API and are more complicated. The output of our sandbox is a log of the program's dynamic execution trace, which includes all system calls placed during malware operation.

3.2. Sequence Purification. In this step, we process the execution trace provided by the previous step. More precisely, for each execution trace, we purify the system calls to purge the many duplicate sequences in a malware dynamic trace. Most of these duplicate sequences are made deliberately to escape active defense techniques. For example, packed malware can release its kernel code during the unpacking stage, and when different malware uses the same packing tool, they may execute similar sequences during the unpacking stage. In the meantime, a file or network operation could also produce duplicate sequences. For example, consider the different methods of reading from a file: Program A might read 256 bytes at once, whereas Program B calls 256 separate times, reading one byte with each call. That is to say, the same behavior may produce different system call sequences. Both behavioral analysis and malware profiling can benefit from sequence purification [21].

The process of malware profiling reveals interesting findings on duplicated sequences. For instance, duplicate sequences, e.g., loadlibrary() and getprocess(), used to obtain an API address in virtual memory usually contain a shorter sequence length than others. We found that such duplicate sequences tend to appear together, emphasizing the locality of the malware behavior.

NtOpenKey, NtQueryValueKey, NtClose, NtUserSetCursor, NtAllocateVirtualMemory, NtGdiHfontCreate
(a) System Call Sequence
(NtOpenKey, NtQueryValueKey, NtClose)
(NtQueryValueKey, NtClose, NtUserSetCursor)
(NtClose, NtUserSetCursor, NtAllocateVirtualMemory)
(NtUserSetCursor, NtAllocateVirtualMemory, NtGdiHfontCreate)
(b) 3-gram of Given Sequence

Box 1: 3-gram of system call sequence.

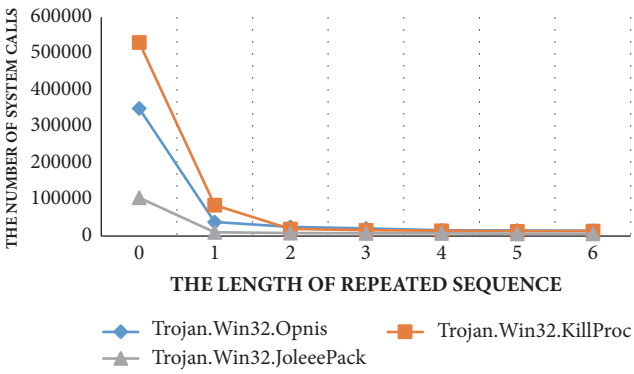

Figure 2: Sequence purification of different malware.

Figure 2 shows three representative processes of system call sequence purifications. We use recursive methods to remove duplicate sequences. We first removed the single repeated system call and found that the result became stable when the length of the duplicate sequence was approximately four. This means that the length of most duplicate system calls is below four.

3.3. N-Gram Feature. The track path of malware behavior is the sequence of its instructions. To characterize the contents of this sequence more specifically, we performed malware profiling by the n-gram method, which enables mapping between the original system call sequences and a characteristic vector by applying a window to the flow of system calls. The n-gram method is widely applied in malware feature analysis [25, 26].

The fragments of instructions, referred to as the n-gram, reflect short behavioral patterns and thus implicitly capture some of the underlying program semantics. As we slide the window across the malware system call trace, we record every unique n-gram. Box 1 shows an example when n = 3. A short system call sequence which contains six system calls will be divided into four different 3-gram snippets. The first 3-gram snippet, NtOpenKey, NtQueryValueKey, NtClose, is a complete register query operation. These small behavioral features are then generalized into behavioral profiles.

In the malware analysis field, n is the length of the sliding window and is usually set at three or four. We choose to use a 3-gram method and 232 system calls involving registry, file, process, network, and other behavior. Every dynamic malware system's call traces are thus transformed into a long vector that represents the capability of the malware behavior.

4. Generating a Benchmark

In this section, hierarchical clustering is performed on the initial data set and malware with similar behavior being gathered into the same cluster. Then, an improved sampling algorithm will select representative samples from cluster to form the final benchmark.

4.1. Hierarchical Clustering. Several methods have been proposed to measure distances between different elements, such as Euclidean distance, Manhattan distance, and Jaccard distance [27]. We chose the Jaccard index as a measure of similarity between different samples [28]. Given the initial malware data set M, Jaccard similarity is defined as

$$S\left(M_i, M_j\right) = \frac{M_i \cap M_j}{M_i \cup M_j} \quad \left(M_i, M_j \in M\right). \tag{1}$$

This equation represents the similarity between two samples M_i and M_j: they are considered to exhibit identical behavior patterns when $S(M_i, M_j)$ is equal to 1. The distance between M_i and M_j can be denoted as D_{ij}, and we can infer that $D_{ij} = 1 - S(M_i, M_j)$. D is a distance matrix of which the elements D_{ij} represent the distance between two samples. For each malware sample in the data set, we compute its distance to other samples and store the result in D.

Hierarchical clustering generates the final cluster results by iterating over the matrix D. One of three linkage criteria must be selected: single linkage, average linkage, and complete linkage. We selected the complete linkage criteria to ensure that the maximum distance between elements of each cluster was under the threshold t. Complete linkage clustering avoids a drawback of the alternative single linkage method, the so-called chaining phenomenon, where clusters formed via single linkage clustering may be forced together because single elements are too close to each other, even though many of the elements in each cluster may be very distant from each other. Complete linkage tends to find compact clusters of approximately equal diameters.

A threshold value t = 0.3 was set, meaning that if the distance between M_i and M_j was above 0.3, those elements could not be assigned to the same cluster. The value of this threshold was the same as that of the Bayer clustering system, which also used dynamic system call features [27].

4.2. Sampling Based on Malice Degree. Sampling on clusters that came from the very first data set was a fundamental step to obtain the malware benchmark. Sampling was expected to meet two requirements: the samples should be representative of the richness of malware varieties and they should include an appropriate number of malware samples. These requirements were set so that the benchmark would maintain its functional effectiveness at a suitable scale. The sampling process presented in this paper was based on a genetic algorithm.

A genetic algorithm is an algorithm that naturally evolves, and such algorithms are widely used to solve optimization and search problems. Several bioinspired operations such as mutation, crossover, and selection are also involved to aid the algorithm's performance. The evolution is an iterative process: in each generation, the fitness of every individual in the population is evaluated by a fitness function, and the fitness values are then normalized. In this work, the fitness values are represented by the maliciousness degree (MD).

Malicious behavior is difficult to quantify, especially for some potentially unwanted program software. Although one analyst may consider a program to be infringing on privacy rights, another may treat the same program as normal data collection. Therefore, we calculate the MD based on antivirus vendors' decisions. We selected 20 top antivirus software programs from the 2016 AV-TEST report and scanned all the samples with those programs [11]. Malware is considered to have a higher MD if it is labeled positive by more antivirus tools. With MDs ranging from 0 to 1, a MD equal to 0.6 means the malware is identified positively by 12 of the antivirus tools used in this assessment.

For cluster k, covering n malware samples, the fitness value of each malware is denoted as F_i:

$$F_i = \frac{MDi}{\sum_{i=1}^{n} MDi}. \quad (2)$$

A roulette wheel result of one cluster that contains 8 malware samples is shown in Figure 3. The higher the fitness value is, the more likely the sample is to be selected in this step. If we need to select m samples in one cluster, m selection iterations will be performed. During each iteration, a number ranging between 0 and 1 is randomly generated and will definitely fall into a region on the roulette wheel. In case the region has been reached before, another random number will be generated instead.

The sampling algorithm is implemented on each cluster, one by one. The detail of algorithm is shown in Algorithm 1. In this study, we chose three samples from every cluster; i.e., k = 3. In case a cluster contains only one or two samples, this cluster would be selected or abandoned. A compromise method is that we select malware from such small clusters only if the sample's MD is above 0.6.

5. Evaluation

The initial data set is composed of two parts: the first part is a collection of 9143 latest incoming samples that might exhibit unknown behaviors, and the second part is a random collection of 9000 samples from the known library. This composition makes sure that the initial data set does not only have a large volume of malware but also own various malware. After the hierarchical clustering, 18,143 samples fell into 925 clusters, 663 of which were small clusters containing less than three samples. Finally, the generated benchmark contained 941 samples which are listed in the Supplementary Material (available here). We have released the generated benchmark's malware md5 list to the malware research community.

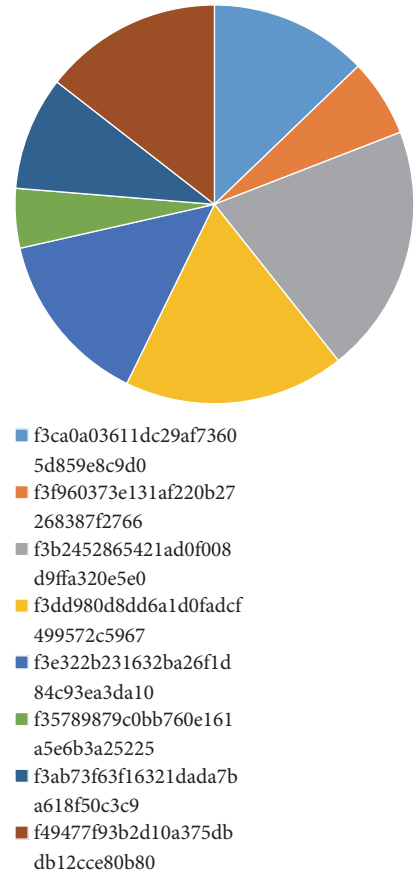

Figure 3: Roulette wheel sampling of one cluster.

5.1. Entropy of Data Set. Entropy is used here to represent the disorder or uncertainty of information, following the work of Shannon [29]. Entropy is a measure of the state's unpredictability or, equivalently, of its average information content. The amount of information of every event forms a random variable, the expected value, or average, of which is the Shannon entropy [29]. Entropy is also used in binary file packer detection: the entropy of a normal binary is usually below 6.5. Malware packers are based on an anti-static analysis technology and have been widely used since recent years. The entropy of a binary file increases dramatically after being packed because the packer changes the original byte type composition, rendering it more diverse.

For a malware data set, the more diverse it is, the more information it contains. A malware benchmark's capability is determined according to how many categories of malware it

```
1: Input: cluster i which contained n samples
2: Output: k selected samples
3: Step 1:
4: for sample in cluster:
5:     calculate MD of each sample
6: end for
7: for sample in cluster:
8:     calculate F_i of each sample
9:     map F_i on a roulette wheel
10: end for
11: while selected samples < k:
12:     generate random number i between 0 and 1
13:     map i to the corresponding range on roulette wheel
14:     select corresponding sample
15: end while
16: return k selected samples
```

Algorithm 1: Sampling on one cluster.

covers rather than the number of malware samples. A data set of 1000 malware samples that came from one category contains less representative information than a data set of 100 malware samples that came from 100 categories. Entropy could indicate the average information content of one set. Apparently, the capability of a malware data set could be represented by the information it contains. A malware data set which contains more capability is preferred to be used as a benchmark for malware detection. In addition, the significativeness of benchmark was further validated in Section 5.2; that is, benchmark's capability of training malware detection model is identical to the initial data set.

For a data set X, the entropy can explicitly be written as

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i). \quad (3)$$

Here, H(X) is the final information entropy, n represents the total number of categories of the malware in the data set, and $p(x_i)$ represents the proportion of the malware in category i. A common value of b, the base of the logarithm used, is two. If there are 1024 malware samples in a data set that belong to only one category, the information entropy is 0, whereas the maximum information entropy is 10 if every malware sample belongs to a different category.

We propose the following average information entropy for different data sets with different volumes of malware samples.

$$AVE_{H(X)} = \frac{-\sum_{i=1}^{n} p(x_i) \log_b p(x_i)}{\log_b K} \quad (4)$$

Here, K is the number of malware samples in the data set and $AVE_{H(X)}$ ranges from 0 to 1. We used antivirus software to determine the category of every malware sample in the benchmark. As described in Section 4.2, we scanned all samples with 20 antivirus software programs and applied a voting algorithm similar to AVclass [30] to determine to which category each malware sample belonged.

APIMDS is a malware detection system proposed by Ki, who released a training data set containing 17,267 records to the research community [22]. This data set provided a full list of md5 hash, with name labels from Kaspersky, and API call sequences, but not the malware files. We downloaded 20 antivirus software labels for these 17,267 records from VirusTotal and performed the voting algorithm mentioned in the previous paragraph.

We generated six benchmarks from the initial data set and selected six data sets with the same volume as the benchmarks from APIMDS. The comparative results are shown in Table 2, where the average entropy of our generated benchmark can be seen to be higher than that of APIMDS, which means that the proposed benchmark contains more malware categories than APIMDS.

5.2. Detection Accuracy Comparison. We chose a machine learning algorithm called random forest, which achieved the best results over other classification algorithms in the Kaggle 2015 Malware Classification Challenge [16]. We designed a rough malware detection model based on the random forest algorithm, which uses dynamic behavior feature inputs, without adjusting any parameters.

We conducted two rounds of experiments. In the first round, we used the initial data set that contained 18,143 malware samples and 6000 benign software samples. To assess the proposed detection model, 10-fold cross-validations were applied in the experimental stage. In the second round, we selected different data for the training and test stages. The generated benchmark and 1000 benign software samples were used as the training data, and 1800 malware samples randomly selected from the initial data set and 600 samples from 6000 benign software samples were used as the test data. The results of these two experiments are shown in Table 3.

We found that the precision of both data sets differed by only 0.33%, which demonstrated that the benchmark's ability of training malware detection model is identical to the initial data set. The higher recall result for the benchmark indicates that the detection model becomes more effective

Table 2: Entropy of APIMDS and benchmark.

	APIMDS	Benchmark
1	0.47	0.88
2	0.46	0.84
3	0.45	0.87
4	0.47	0.84
5	0.47	0.83
6	0.45	0.85

Table 3: Detection accuracy results.

Data Set	Precision	Recall	FPR	Accuracy
Initial Data Set	0.7234	0.7225	0.2762	0.7231
Benchmark	0.7267	0.8012	0.3012	0.7500

in detecting malware samples. The initial data set exhibited a better false positive rate because it contained more benign software than the benchmark. The benchmark also exhibited a higher accuracy than the initial data set.

5.3. Performance Analysis. We generated the benchmark on top of a standard desktop system equipped with 8 GB of memory, an i7 CPU core, and a 1-TB disk. We generated benchmarks covering 941 samples in 12014 seconds (excluding the time when the malware was running in the sandbox). Thus, picking a sample for the benchmark requires an average of 12 seconds, which is substantially less time than is required for manual selection. Therefore, the framework enables malware researchers to generate their own benchmark easily and automatically, and the volume of the benchmark is only one-eighteenth of the initial data set so that the processing time in the training and testing stages is only one-eighteenth of that required for the initial data set.

6. Conclusion

In this paper, we firstly proposed a framework for generating benchmarks for malware detection. The framework has three components: dynamic feature extraction, hierarchical clustering, and a genetic algorithm based on roulette wheel sampling. We validated the proposed framework's ability to generate a diverse benchmark data set. The final data set is only one-eighteenth the volume of the initial data set, comparing the average entropy of the generated benchmark with other data sets. We compared the detection accuracy of the same learning model trained by benchmark and initial data set, and the result showed that benchmark could make the learning model own the same detection ability even though its volume is only one-eighteenth of the initial data set

Finally, we have released the benchmark data set on our website www.malwarebenchmark.org, where researchers can download the md5 list. For security, if someone needs to use binary files, we will provide those separately. Every two or three months, we will update the website with the newest benchmark.

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

We are grateful to Pros.S for providing us with the initial raw data. We also thank VirusTotal for their support. This research was supported by a project of National Science Foundation of China under grant no. 61472447.

Supplementary Materials

Supplementary material is the md5 list of benchmark which we selected from initial data set. *(Supplementary Materials)*

References

[1] S. Cesare, Y. Xiang, and W. Zhou, "Malwise—an effective and efficient classification system for packed and polymorphic malware," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 62, no. 6, pp. 1193–1206, 2013.

[2] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proceedings of the USENIX Security Symposium*, pp. 807–822, 2016.

[3] J. Drew, T. Moore, and M. Hahsler, "Polymorphic Malware Detection Using Sequence Classification Methods," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy Workshops, SPW 2016*, pp. 81–87, USA, May 2016.

[4] X. Hu and K. G. Shin, "DUET: integration of dynamic and static analyses for malware clustering with cluster ensembles," in *Proceedings of the the 29th Annual Computer Security Applications Conference*, pp. 79–88, New Orleans, Louisiana, December 2013.

[5] 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.

[6] L. Wu, M. Xu, J. Xu, N. Zheng, and H. Zhang, "A novel malware variants detection method based on function-call graph," in *Proceedings of the 2013 IEEE Conference Anthology, ANTHOLOGY 2013*, China, January 2013.

[7] J. Upchurch and X. Zhou, "Variant: A malware similarity testing framework," in *Proceedings of the 10th International Conference on Malicious and Unwanted Software, MALWARE 2015*, pp. 31–39, USA, October 2015.

[8] S. P. E. C. Benchmarks, *Standard Performance Evaluation Corporation*, 2000, Standard Performance Evaluation Corporation.

[9] D. Bailey, E. Barszcz, J. Barton et al., "The Nas Parallel Benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 2016.

[10] AV-Comparatives, Malware Protection Test March 2017, https://www.av-comparatives.org/wp-content/uploads/2017/04/avc_mpt_201703_en.pdf.

[11] AV-TEST, The Best Antivirus Software for Windows Home User, https://www.av-test.org/en/antivirus/home-windows/windows-7/february-2017/.

[12] V. X. Heaven, http://vxer.org/src.php?show=all.

[13] VirusShare, https://virusshare.com/.

[14] G. Wolf and M. Khoshgoftaar T, "Using Machine Learning for Network Intrusion Detection," *The Need for Representative Data*, 2016.

[15] C. Rossow, C. J. Dietrich, C. Grier et al., "Prudent practices for designing malware experiments: Status quo and outlook," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy, S and P 2012*, pp. 65–79, USA, May 2012.

[16] Malware classification, Microsoft Malware Classification Challenge (BIG 2015), https://www.kaggle.com/c/malware-classification.

[17] L. Peng and C. Wang R, "Research on Unknown Malicious Code Automatic Detection Based on Space Relevance Features," *JCRD*, 2010.

[18] 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.

[19] A. Shabtai, R. Moskovitch, C. Feher et al., "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, 2012.

[20] Z. Zhao, J. Wang, and C. Wang, "An unknown malware detection scheme based on the features of graph," *Security and Communication Networks*, vol. 6, no. 2, pp. 239–246, 2013.

[21] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 2015, Article ID 659101, 9 pages, 2015.

[22] M. Hurier, K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "On the Lack of Consensus in Anti-Virus Decisions: Metrics and Insights on Building Ground Truths of Android Malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 9721 of *Lecture Notes in Computer Science*, pp. 142–162, Springer International Publishing, Cham, 2016.

[23] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep Ground Truth Analysis of Current Android Malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 10327 of *Lecture Notes in Computer Science*, pp. 252–276, Springer International Publishing, Cham, 2017.

[24] DynamoRIO, Dynamic Instrumentation Tool Platform, http://dynamorio.org.

[25] C. Wressnegger, G. Schwenk, D. Arp et al., "A close look on n-grams in intrusion detection: anomaly detection vs. classification," in *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, pp. 67–76, ACM, 2013.

[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 604–613, ACM, 1998.

[27] U. Bayer, M. P. Comparetti, and C. Hlauschek, "Scalable, behavior-based malware clustering," *NDSS*, vol. 9, pp. 8–11, 2009.

[28] S. Dolev, M. Ghanayim, A. Binun, S. Frenkel, and Y. S. Sun, "Relationship of Jaccard and edit distance in malware clustering and online identification (Extended abstract)," in *Proceedings of the 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pp. 1–5, Cambridge, MA, October 2017.

[29] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[30] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 230–253, Springer International Publishing, 2016.