Malware Detection Using System Logs

Nhu T. Nguyen[*]
Thuy T. Pham[*]
AISIA Research Lab
Ho Chi Minh, Vietnam

Tien X. Dang
AISIA Research Lab
Ho Chi Minh, Vietnam

Minh-Son Dao
National Institute of Information and Communications Technology
Tokyo, Japan

Duc-Tien Dang-Nguyen[†]
Department of Information Science and Media Studies
University of Bergen
Bergen, Norway

Cathal Gurrin
Dublin City University
Dublin, Ireland

Binh T. Nguyen[‡]
AISIA Research Lab
VNU HCM - University of Science
Ho Chi Minh, Vietnam

ABSTRACT

Malware detection is one of the most critical features in many real applications, especially for the mobile platform and the Internet of Things (IoT) technology. Due to the proliferation of mobile devices and the associated app-stores, the volume of new applications growing extremely fast requires a better way to analyze all possible malicious behaviors. In this paper, we investigate the malware prediction problem using system log files that contain numbers of sequences of system calls recorded from IoT devices. We construct a suitable multi-class classification model by using the combination of hand-crafted features, (including Bag-of-Ngrams, TF-IDF, and the statistical metrics computed from the consecutive repeated system calls in each log file). Also, we consider different machine learning models, including Random Forest, Support Vector Machines, and Extreme Gradient Boosting, and measure the performance of each method in terms of precision, recall, and F1-score. The experimental results show that a combination of different features, as well as using the Extreme Gradient Boosting technique, can help us to achieve promising performance in the dataset provided by the organizers of the competition CMDC 2019.

CCS CONCEPTS

• **Computing methodologies → Machine learning algorithms**; • **Information systems → Expert systems**; • **Security and privacy → File system security**; **Mobile platform security**.

KEYWORDS

Malware Detection, IoT, SVMs, Random Forest, XGBoost

[*]Both authors contributed equally to this research.
[†]Senior author
[‡]Corresponding author

ICDAR '20, June 8, 2020, Dublin, Ireland
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7509-2/20/06…$15.00
https://doi.org/10.1145/3379174.3392318

ACM Reference Format:
Nhu T. Nguyen, Thuy T. Pham, Tien X. Dang, Minh-Son Dao, Duc-Tien Dang-Nguyen, Cathal Gurrin, and Binh T. Nguyen. 2020. Malware Detection Using System Logs. In *Proceedings of the 2020 Intelligent Cross-Data Analysis and Retrieval Workshop (ICDAR '20), June 8, 2020, Dublin, Ireland.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3379174.3392318

1 INTRODUCTION

Malware or malicious software refers to any software designed to infect and cause damages to individual computers, the entire organization's network, servers, clients, or mobile phones (iOS, Android, and Windows phones). Typically, it exploits the vulnerabilities of each target system. For instance, some hackers can utilize a bug in legitimate software (e.g., a browser or web application plugin) to illegally hijack a company server for stealing valuable information or money. Subsequently, the malware infiltration is disastrous, and it becomes one of the worldwide epidemics due to the rapid increase in the number of computer security incidents [6]. There are various malware programs, including computer viruses, worms, Trojan horses, ransomware, spyware, adware, and scareware [17].

Nowadays, the number of smart devices has been increasing significantly due to the rapid development of the Internet of Things (IoT) technology. There have been different types of applications related to IoT devices, including intelligent transportation, smart factories, smart homes, and smart laboratories. In general, the term "Internet of Things" represents a network of multiple sensing devices with limited resources that can do wired/wireless communications with cloud services or others. Given billions of IoT devices currently using in many aspects of daily lives, IoT has been gradually becoming one of the weakest parts of the computer network. Hackers more and more target IoT devices as they can quickly attack them and create more massive damages than conventional computers [9]. It turns out that malware detection has been playing a vital role in many systems, especially in mobile platforms that have a massive volume of new applications submitted to the apps stores. Traditionally, for building a malware detection algorithm, people can manually check malicious behaviors and de-compiled codes of known malware programs to determine the most critical signatures for identifying each of them. However, due to the massive number of new applications, signature-based malware detection is not easy to scale up. As a result, there has recently been a lot of studies on automatic malware detection by applying machine learning techniques.

Shabtai and co-workers [13] present a new framework, namely Andromaly, to detect malware on Android mobile devices. In this framework, the authors propose a host-based malware detection system that can continuously monitor different features and events obtained from the mobile device. After that, they formulate the malware detection problem into a classification problem and apply machine learning techniques to detect abnormal behaviors for identifying the corresponding malware. Su and colleagues [14] illustrate a new smart-phone dual defense protection framework allowing Official and Alternative Android Markets to detect all possible malicious applications among those new applications submitted for public release. These authors also transform the malware detection problem into a multi-class classification problem. They then apply Random Forest for constructing the corresponding classifier with a promising performance in terms of accuracy. Recently, Liu et al. [9] address the malware detection problem for the IoT technology and present an architecture for extracting and analyzing IoT malware behaviors as well as detecting anomalous events from IoT devices. More studies can be found at [3, 8, 11].

In this paper, we aim at studying the IoT malware classification problem by using the sequence of system calls during the runtime of malware and one dataset provided in the 10th international Cybersecurity Data Mining Competition (CDMC 2019)[1]. This IoT malware classification 2019 dataset is provided by the Taiwan Information Security Center (TWISC), including two datasets: one for training and another one for testing. One of the main challenges of these datasets is during the running time of a given Linux program, only all interactions among processes initialized by the program and the Linux kernel are captured and monitored. Subsequently, each procedure yields a log file containing multiple lines of system calls. On each line, only the time stamp, the invoked system call, as well as parameters and results of the calls are recorded by using the "strace" command with the lack of other useful information. To overcome these challenges in this problem, we propose a novel feature engineering approach by using bags-of-ngrams (BoN) features, Term Frequency–Inverse Document Frequency (TF-IDF), and all possible features that can be extracted from each log file. After that, we apply different machine learning models (Random Forest, SVM, and XGBoost) to construct a suitable algorithm for the malware classification step. The experimental results show that our proposed method can achieve promising performance in terms of precision, recall, and F1-score.

2 PROPOSED METHODS

In this section, we first give a brief of problem formulation for the malware identification based on system logs. We introduce our proposed approach to construct an appropriate multi-class classification model for the problem by using the combination of handcrafted features such as Bag-of-Ngrams [10], TF-IDF [4], and the statistical analytics related to the consecutive repeated system calls in each log file. After that, we consider various machine learning algorithms, including Random Forest [1], Support Vector Machines [15, 16], and Extreme Gradient Boosting [2], and measure the corresponding performance of each algorithm in terms of precision, recall, and F1-score.

[1]http://www.csmining.org/cdmc2019/index.php?id=5

2.1 Problem Formulation

As mentioned previously, we aim at studying the malware classification problem based on each log file, which contains a sequence of system calls captured during the running time of malware in a Sandbox environment. Datasets provided by the competition organizers include one training dataset and one testing dataset. In the training dataset, there are 4167 formatted sequences of system calls, labeled by the malware type. Meanwhile, there are 4275 files with unknown labels for the final submission.

Noticeably, there exist 8442 samples for both training and testing datasets provided by the following procedure. First, the competition organizers collects a list of potentially malicious Linux programs in CEF format from different resources. After that, they let each such program execute in a Sandboxed environment hosted by an emulator providing the required runtime environment for it. During the running time, the "strace" command is used to monitor and record all interactions among processes initialized by the program and the Linux kernel. Each procedure yields one separated log file, which contains multiple lines of system calls. On each line, "strace" records the time stamp, the invoked system call, parameters, and results of the calls.

These log files are later parsed and reformatted in a simplified format, as shown in the ".seq" files. There might be multiples lines in these files, where each line stands for the sequence of system calls invoked by a particular process initialized by the malware. The system calls in each line are presented in the ascending order of the function call time. The processes are presented in ascending order of the creation time. For instance, here is a sequence of system calls at a specific timestamp in one log file provided:
"execve ioctl ioctl time getpid time getpid socket connect getsockname open read read read read . . . read read read read read close ioctl close fork wait4 SIGCHLD exit EXIT".
One can see that the first process in this line is "execve", which is usually used for executing a program referred to by a given pathname. After that, the program has two executions for the process "ioctl", which is a system call for device-specific input/output operations and other operations which can not be expressed by regular system calls. Then, the program continues executing other processes, and especially, it has multiple "read" processes. Finally, the program ends by the "exit" command.

Typically, there are 123 system calls in total among both training and testing datasets. It is worth noting that each malicious program records many log files, and each log file has a different number of lines, even with the same malware. The main goal of this problem is to find an efficient classification model for identifying what type of malware programs a given logfile belongs to.

2.2 Feature Extraction

Feature extraction plays an important role in each machine learning project, particularly for the context of natural language processing. In this work, we select the following features for the malware classification problem.

Table 1: All system calls existed in the training and testing sets provided.

1	_llseek	32	getdents	63	poll	94	shmdt
2	_newselect	33	getdents64	64	prctl	95	shmdt
3	access	34	geteuid	65	prlimit64	96	shutdown
4	bind	35	geteuid32	66	read	97	sigbus
5	brk	36	getpid	67	readlink	98	sigchld
6	cacheflush	37	getppid	68	readv	99	sigfpe
7	chdir	38	getrlimit	69	recv	100	sigill
8	chmod	39	getsockname	70	recvfrom	101	sigpipe
9	chroot	40	getsockopt	71	rename	102	sigreturn
10	clock_gettime	41	gettid	72	restart_syscall	103	sigrtmin
11	clone	42	gettimeofday	73	rmdir	104	sigsegv
12	close	43	getuid	74	rt_sigaction	105	sigtrap
13	connect	44	getuid32	75	rt_sigprocmask	106	socket
14	dup2	45	ioctl	76	rt_sigsuspend	107	stat
15	epoll_create1	46	kill	77	send	108	stat64
16	epoll_ctl	47	listen	78	sendfile64	109	sysinfo
17	epoll_pwait	48	lseek	79	sendto	110	tgkill
18	execve	49	lstat64	80	set_robust_list	111	time
19	exit	50	mkdir	81	set_thread_area	112	times
20	exit_group	51	mknod	82	set_tid_address	113	tkill
21	fchmod	52	mmap	83	set_tls	114	ugetrlimit
22	fchown	53	mmap2	84	setpgid	115	umask
23	fchown32	54	mprotect	85	setpriority	116	uname
24	fcntl	55	msgctl	86	setresuid	117	unlink
25	fcntl64	56	msgget	87	setresuid32	118	utimensat
26	flock	57	msgrcv	88	setrlimit	119	vfork
27	fork	58	munmap	89	setsid	120	wait4
28	fstat	59	nanosleep	90	setsockopt	121	waitpid
29	fstat64	60	open	91	setuid	122	write
30	futex	61	pipe	92	setuid32	123	writev
31	getcwd	62	pipe2	93	shmat		

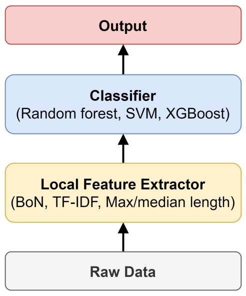
Output
Classifier
(Random forest, SVM, XGBoost)
Local Feature Extractor
(BoN, TF-IDF, Max/median length)
Raw Data

Figure 1: The workflow of our proposed malware classification method.

2.2.1 BoN features. A bags-of-ngrams [10], which stands for BoN, is a method to extract features from documents, similar to a bag-of-word [18] approach. The BoN model records the number of times that each n-gram appears in each document of one collection. This one-of-the-most-popular approach is very simple and flexible as it is a collection of all words (both from training and testing data) to represent a text document as a collection of n successive words.

In this study, we represent the frequency of each group including from one to three system calls among log files. For example, let us use the following sequence of system calls, which is "execve ioctl ioctl prctl gettimeofday getpid exit". We can divide it into bi-grams ($n = 2$) as follows:

- execve ioctl
- ioctl ioctl
- ioctl prctl
- prctl gettimeofday
- gettimeofday getpid
- getpid exit

2.2.2 TF-IDF features. Term Frequency–Inverse Document Frequency [4], TF-IDF for short, is a numerical measure that aims to emphasize the role of each word in one text document. This scheme does not consider the connection among words, whereas it highlights the similarity of documents if they have the same words. Consequently, the higher the TF-IDF scores, the rarer the term is.

$$TFIDF = TF * IDF \quad (1)$$

Table 2: Our proposed hand-crafted features.

Name	Size
TF-IDF	5000
Bag-of-Words	6969
Max length	123
Median length	123

In particular, TF, Term-Frequency, shows how frequently a term occurs in a document.

$$TF = \frac{\text{Number of times term } t \text{ appears in a doc}}{\text{Number of total terms in doc}} \quad (2)$$

Moreover, IDF, Inverse Document Frequency, is a measure of determining the frequency of a word in texts. If a word occurs many times not only in a document but also along many others, maybe it is just a frequent word, not relevant or meaningful.

$$IDF = \log\left(\frac{\text{Number of docs}}{\text{Number of docs terms } t \text{ appears in}}\right) \quad (3)$$

In this task, we are taking into account the relation among phrases having the length from one to three words for each.

2.2.3 Maximum length features. We individually consider the maximum length of the frequency of the consecutive repeated system calls in one file. As one can see, something that repeats overtimes has a high risk of abnormal behaviors, so it is such a piece of valuable information. First, we calculate how many times each term continually occurs, whereas the non-repeated cases are denoted by 0. Then we get the maximum value corresponding to a given system call.

2.2.4 Median length features. The consecutive repetitions are similarly calculated similar to what we do for the maximum length features. However, in this case, we want to get the median value instead of the maximum value. The details of different features computed can be showed in Table 2.

2.2.5 Fusion features. TF-IDF features provide information about essential words in a text document. Besides, BoN and the length features (maximum and median ones) give deeper information about abnormal behaviors via the consecutive repeated system calls in one file. Accordingly, combination of these features, $\mathbf{F}_f$, can gets more advantages of them to enhance the performance of our proposed approach. $\mathbf{F}_f$ can be created by concatenating the TF-IDF features ($\mathbf{F}_{TFIDF}$), BoN features ($\mathbf{F}_{BoN}$), the maximum length features ($\mathbf{F}_{max}$), and the median length features ($\mathbf{F}_{med}$), as depicted in the following equation:

$$\mathbf{F}_f = \mathbf{F}_{TFIDF} \oplus \mathbf{F}_{BoN} \oplus \mathbf{F}_{max} \oplus \mathbf{F}_{med} \quad (4)$$

2.3 Machine Learning Algorithms

As there are not so many records in the training data, using a deep learning model somehow is not the right choice in this work. Consequently, we decide choosing a traditional machine learning approach and firmly believe that it can work well to learn useful features from the initial raw data. Once we complete finding appropriate features, we can apply different learning models, such as Random Forest [1], Support Vector Machines [15, 16], and Extreme Gradient Boosting [2], for learning the optimal classifier.

2.3.1 Random Forest. It is one of the most well-known supervised learning algorithms and can be used for both classification and regression issues. Random forest is a combination of decision trees (DT) [12] by emploiting ensemble methods. For the training process, the random forest uses the bagging method. Figure 2 illustrates a typical structure of the random forest and how it works. In summary, a random forest builds multiple decision trees and merges them to get a more accurate and stable prediction.

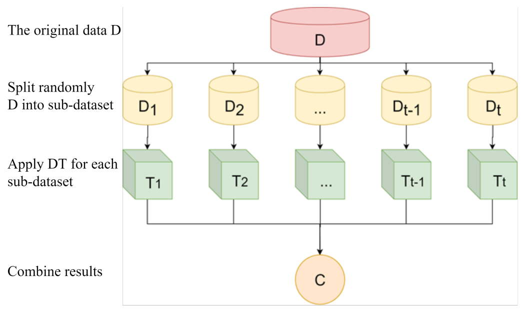
The original data D
D
Split randomly D into sub-dataset
D1
D2
...
Dt-1
Dt
Apply DT for each sub-dataset
T1
T2
...
Tt-1
Tt
Combine results
C

Figure 2: A typical structure of a random forest algorithm.

2.3.2 Support Vector Machines (SVMs). Vladimir Vapnik introduced Support Vector Machines (SVMs) in 1995 [15, 16], which have had a vast number of useful applications in many fields of research. SVMs are supervised learning models hvaing associated learning algorithms that analyze data used for classification and regression analysis. The fundamental problem of SVMs is to classify two classes by finding an optimal hyperplane, which divides data into two parts with the same label, as shown in Figure 3. However, in real-world problems often have more than two classes. Therefore, SVMs are upgraded to solve the multi-class problem. In this study, we use a one-versus-rest method (one-vs-all), which can split m classes into m binary classifications and solve each separately.

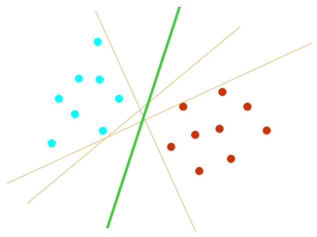

Figure 3: The optimal hyperplane (the green line) generated by SVMs.

2.3.3 Extreme Gradient Boosting. XGBoost is a short term for Extreme Gradient Boosting [2], which is a supervised learning algorithm. Its main idea is based on the Gradient Boosting algorithm [5]. In real-world problems, missing values in data one of the possible cause for sparsity, which leads to low performances. Therefore, it is essential to make the algorithm aware of the sparsity pattern in the data. To solve this problem, Chen and co-workers [2] propose to add a default direction in each tree node. Also, applying parallel computation makes its performance improve significantly. XGBoost has later become very popular in recent years due to its versatility, scalability, and efficiency.

3 EXPERIMENTS AND RESULTS

3.1 Dataset

The given IoT malware dataset from the competition CDMC 2019 consists of the training set (4167 log files having formatted sequences of system calls and labeled by the malware type) and the testing set (4275 log files). Importantly, there are five labels (types of malware programs) selected in this dataset. As depicted in Figure 4, one can see that the training set is imbalanced, where most of the samples belong to the first and the second classes. The first class has 1541 samples, while the second class has 2265 ones. As the imbalance can create more challenges and sometimes create poor performance, one can set the class weights for the imbalanced data [7] to overcome this issue. In our work, we alter the corresponding weight of each class during the experiment for improving the learning model, instead of choosing equal weights for all categories. Higher class weight is more emphasis on a particular label.

Tables 3 and 4 show the top 5 highest-frequency system calls as well as the top 5 lowest-frequency system calls among different classes in the initial training set provided by the organizers.

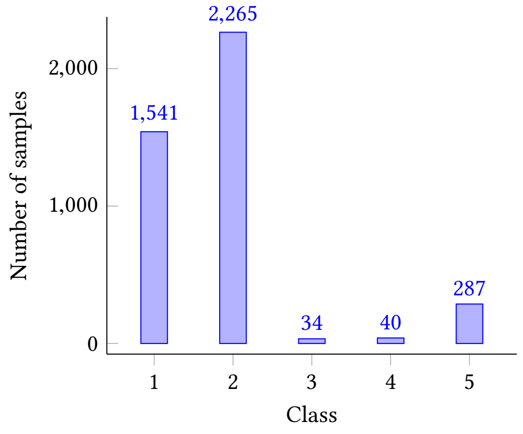

Figure 4: The distribution among different malware labels in the training set provided.

3.2 Experiments and Results

In this study, we apply traditional machine learning algorithms for building an appropriate classification model for the problem, such as Random Forest [1], SVM [15, 16], and XGBoost[2]). During the training process, we use the training data provided to learn the most suitable hyperparameters of each model and compare them directly by using precision, recall, and F1-score.

For tuning hyperparameters, a grid search method is used to learn the best setting for each classifier. In detail, the random forest model is trained by selecting the number of trees in the set [10, 100, 1000], respectively. Also, SVM models are trained by choosing the radial basis function (RBF) kernel, γ in the range [0, 10], and C in the range [0, 10]. XGBoost with SoftMax as the objective function is measured the corresponding performance by using the number of trees in set [10, 100, 1000], and the depth in the range [3, 12], consecutively. Moreover, we consider two different scenarios in which we separate the initial training dataset into two groups (one for training and one for validation) with the following ratios: (a) 8 by 2 and (b) 7 by 3 to measure the stability of various approaches.

Tables 5 and 6 illustrate the experimental results on both cases. There is fluctuation in the results of SVMs, especially in the case of the fourth class. For instance, the results with respect to the recall and F1 score reduce significantly compared with the results associated with the precision. While the stability results are achieved in both Random Forest and XGBoost, XGBoost has the best performance in comparison with the others in two different scenarios. Finally, these experimental results indicate that the more data are, the better the performance is.

4 CONCLUSION

In this paper, we have presented an efficient approach for malware detection based on system logs. We have conducted extensive experiments using a number of features (such as BoN [10], TF-IDF [4], the maximum length, and the median length) to build the classification system. Experiment results show the capability of our approach, especially using XGBoost [2], can achieve superior performance among the other methods. Our work is the first step for building an autonomous system that aims to detect malware in IoT systems. In the future, we plan to explore other features of the text documents for a more robust prediction.

ACKNOWLEDGEMENT

This research is conducted under the Collaborative Research Agreement between National Institute of Information and Communications Technology and University of Science, Vietnam National University at Ho Chi Minh City. We acknowledge the support of Science Foundation Ireland under grant number SFI/13/RC/2106 and L. Meltzers Høyskolefonds, UiB 2019/2259-NILSO.

REFERENCES

[1] Leo Breiman. 2001. Random Forests. *Machine Learning* 45 (2001), 5–32.
[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*.
[3] Andrei Costin. 2018. IoT Malware : Comprehensive Survey , Analysis Framework and Case Studies. In *Black Hat Conference 2018*.
[4] Bijoyan Das and Sarit Chakraborty. 2018. An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation. *ArXiv* abs/1806.06407 (2018).
[5] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.
[6] Nwokedi Idika and Aditya P Mathur. 2007. A survey of malware detection techniques. *Purdue University* 48 (2007), 2007–2.
[7] Gary King and Langche Zeng. 2001. Logistic regression in rare events data. *Political analysis* 9, 2 (2001), 137–163.
[8] A. Kumar and T. J. Lim. 2019. EDIMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. 289–294. https://doi.org/10.1109/WF-IoT.2019.8767194
[9] Zhongjin Liu, Le Zhang, Qiuying Ni, Juntai Chen, Ru Wang, Ye Li, and Yueying He. 2019. An Integrated Architecture for IoT Malware Analysis and Detection. In *IoT as a Service*, Bo Li, Mao Yang, Hui Yuan, and Zhongjiang Yan (Eds.). Springer International Publishing, Cham, 127–137.

Table 3: Top-5 system calls frequently appear in each class of training data.

Ranking	Class				
	1	2	3	4	5
1	sendto	recvfrom	recvfrom	sendto	recvfrom
2	recvfrom	sendto	sendto	read	read
3	read	read	read	recvfrom	sendto
4	fcntl	fcntl	fcntl	time	fcntl
5	close	close	close	readlink	close

Table 4: Top-5 system calls rarely appear in each class of training data.

Ranking	Class				
	1	2	3	4	5
1	tkill	tkill	chmod	pipe	setpgid
2	pipe2	setpriority	epoll_pwait	shmget	sigpipe
3	sigill	umask	pipe	geteuid32	mkdir
4	shmget	sigfpe	epoll_create1	geteuid	mknod
5	restart_syscall	pipe2	set_thread_area	umask	msgget

Table 5: The performance of our models where the ratio of samples between the training dataset and the testing dataset is 8:2.

Class	Random Forest			SVM			XGBoost			Number of samples
	P	R	F	P	R	F	P	R	F	
1	0.99	0.97	0.98	0.93	0.97	0.95	0.99	0.97	0.98	313
2	0.98	1.00	0.99	0.98	0.96	0.97	0.98	0.99	0.99	466
3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	6
4	1.00	0.88	0.93	1.00	0.62	0.77	1.00	1.00	1.00	8
5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	41
AVERAGE	0.99	0.99	0.99	0.96	0.96	0.96	**0.99**	**0.99**	**0.99**	834

Table 6: The performance of our models where the ratio of samples between the training dataset and the testing dataset is 7:3.

Class	Random Forest			SVM			XGBoost			Number of samples
	P	R	F	P	R	F	P	R	F	
1	0.99	0.97	0.98	0.94	0.97	0.95	0.99	0.97	0.98	479
2	0.97	0.99	0.98	0.98	0.96	0.97	0.98	0.99	0.99	684
3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	6
4	1.00	0.85	0.92	1.00	0.77	0.87	1.00	0.92	0.96	13
5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	69
AVERAGE	0.98	0.98	0.98	0.96	0.96	0.96	**0.98**	**0.98**	**0.98**	1251

[10] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. 1999. *Foundations of statistical natural language processing.* MIT press.

[11] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and et al. 2017. Deep Android Malware Detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY ’17).* Association for Computing Machinery, New York, NY, USA, 301–308. https://doi.org/10.1145/3029806.3029823

[12] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1, 1 (1986), 81–106.

[13] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. 2012. “Andromaly”: A Behavioral Malware Detection Framework for Android Devices. *J. Intell. Inf. Syst.* 38, 1 (Feb. 2012), 161–190. https://doi.org/10.1007/s10844-010-0148-x

[14] X. Su, M. Chuah, and G. Tan. 2012. Smartphone Dual Defense Protection Framework: Detecting Malicious Applications in Android Markets. In *2012 8th International Conference on Mobile Ad-hoc and Sensor Networks (MSN).* 153–160. https://doi.org/10.1109/MSN.2012.43

[15] Simon Tong and Daphne Koller. 2001. Support Vector Machine Active Learning with Applications to Text Classification. *J. Mach. Learn. Res.* 2 (2001), 45–66.

[16] Vladimir Vapnik. 2013. *The nature of statistical learning theory.* Springer science & business media.

[17] Amit Vasudevan and Ramesh Yerraballi. 2006. Spike: engineering malware analysis tools using unobtrusive binary-instrumentation. In *Proceedings of the 29th Australasian Computer Science Conference-Volume 48.* 311–320.

[18] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1 (2010), 43–52.