Application of Machine Learning Algorithms for Android Malware Detection

Mohsen Kakavand
School of Science & Technology
Sunway University
Kuala Lumpur, Malaysia
mohsenk@sunway.edu.my

Mohammad Dabbagh
School of Science & Technology
Sunway University
Kuala Lumpur, Malaysia
mdabbagh@sunway.edu.my

Ali Dehghantanha
School of Computer Science
University of Guelph
Ontario, Canada
adehghan@uoguelph.ca

ABSTRACT
As the popularity of Android smart devises increases, the battle of alleviating Android malware has been considered as a crucial activity with the advent of new attacks including progressively complicated evasion techniques, consequently entailing more cutting-edge detection techniques. Hence, in this paper, two Machine Learning (ML) algorithms, called Support Vector Machine (SVM) and K-Nearest Neighbors (KNN), are applied and evaluated to perform classification of the feature set into either benign or malicious applications (apps) through supervised learning process. This work involves in static analysis of apps, which checks for the presence and frequency of keywords in the Android apps' manifest file and derives the static feature sets from a 400-app dataset to produce better malware detection results. The classification performance of the ML algorithms is measured in terms of accuracy and true positive rate and interpreted to determine which algorithm is more applicable for the Android malware detection. The experimental results for a dataset of real malware and benign apps indicate the average accuracy rate of 79.08% and 80.50% with average true positive rate of over 67.00% and 80.00% using SVM and KNN, respectively.

CCS Concepts
• Security and privacy → Mobile and wireless security

Keywords
Machine Learning; Static Analysis; Android Malware Detection; SVM; KNN

1. INTRODUCTION
Android has been recognizing as one of the most prevailing operating systems for mobile phones, tablets and other devices. Like every other system, Android is prone to malware attacks which hackers use to obtain valuable information [1]. Because of this, most of hackers target on Android, as an example Kaspersky, which is one of the leading security solution company, stated that 5,730,916 malicious packages were detected by their lab in 2017 or scientists from Zimperium Labs have shown that a single text message could infect 95 percentage of Android smart devices while University of Cambridge's scholars figured out that 87 percentage of all Android devices could be suffered from at least one critical vulnerability [2]. So protecting the Android devices from misuse or any malicious application is one of the necessity in this field. To protect Android systems from such attacks, antivirus tools are used to prevent and detect them. However, existing commercial tools can only prevent and detect malware that are known to them [3], which means newly developed malware could still remain undetected for months or even years. To overcome this issue, researchers have proposed the utilization of Machine Learning (ML) algorithms in Android malware detection systems to identify malware through behavior-based, change-based and anomaly-based software analysis [4].

Application of machine learning has been widely used in order to detect and classify several Android applications. Basically, a learning classifier is trained on a labeled set of samples, using a feature vectors that represent syntactic or semantic properties of the applications. The structure of Android applications (apps) gives valuable syntactic information such as, method and package names, permissions, and configuration files. Consequently, statically derived features allow to detect and classify different Android applications [5]. In this paper, analyze both the user permissions and intent filters requested in an Android app's manifest file, using the two most promising ML classifier algorithms shortlisted from previous research, namely the Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) algorithms. The main objective of this paper is to investigate whether analyzing keywords in both the permissions requested in an app's manifest file and system call logs can provide us with better results in detecting the malware apps while using the two mentioned ML algorithms.

The remainder of this paper is organized as follows. Section 2 presents an overview on the existing researches which have been conducted by the other researchers to detect Android malware using ML algorithms. Section 3 elaborates our methodology that has been performed to conduct this research while the results and discussion are demonstrated in Section 4. Ultimately, Section 5 concludes our findings and introduces future research directions.

2. RELATED WORK
Over the recent years a substantial amount of research has been invested into proposing different Android malware anomaly detection approaches using machine learning techniques [6], [7], [8]. These approaches can be mainly split into static, dynamic, and hybrid malware analyses. Among the researchers who apply static analysis are Kumaran and Li [9]. Their selection of ML algorithms initially analyzes both the intent filters and permissions requested in an app's manifest.xml file individually, which resulted in a drastically huge difference in detection accuracy due to the relatively weak relevance of the intent filters alone.

CIIS 2018, November 17-19, 2018, Phuket, Thailand

ACM ISBN 978-1-4503-6595-6/18/11…$15.00
DOI: https://doi.org/10.1145/3293475.3293489

However, analyzing a combination of both static features returned a result of 91.7% (SVM) and 91.4% (KNN) for detection accuracy, which is an improvement over the classification performance of both individual feature sets.

Most research on Android malware detection focused on dynamic analysis due to their resilience to code obfuscation [10], [11], and [9] and rapid application updates, as well as the open-source nature of Android that allows increased disclosure of device features [11]. As for dynamic analysis, Singh and Hofmann [10] concluded that applying an SVM classifier with correlation analysis on system call behaviors provided the best performance with 97.16% accuracy and 99.54% recall. They insist that system calls are suitable features for malware detection as all device resources are allocated to apps by system calls to the Linux kernel [10]. They also presented their findings about the system call behaviors relevant to malware detection in their paper. Tan et al [12] use a combination of feature weightage, feature correlation, and the Naïve Bayes ML classification model to analyze Android API calls. Compared to their result from using Naïve Bayes classification model (recall rate 0.3255, precision rate 0.4266), the proposed model achieves a result of 0.6678 for recall and 0.6894 for precision.

Meanwhile, Garcia et al [13] focus on analyzing Android API usage and system calls by native binaries to obtain high classification accuracy, obfuscation resilience, and system scalability. Their proposed approach manages to achieve a 98% F-measure for detection and an 84% malware family classification rate. The work of Al Ali et al [8] is also focused on the anomaly-based detection of malicious software. However, their proposed framework obtains feature vectors from system metrics (e.g. battery or network data) and then classified them with machine learning algorithms, implemented in the Waikato Environment for Knowledge Analysis (Weka) [14] running on an external server, thus conserving device resources. Top 3 algorithms that performed better than others are: Random Forest (RF), SVM, and Naïve Bayes. The RF algorithm they use offers the best true positive rate, precision, recall, F-measure and AUROC (all more than 0.9), while their SVM algorithm (Sequential Minimal Optimization, SMO, using the PUK kernel) performs second best with results closely matching that of the RF algorithm. The KNN algorithm they use (Instance Based Learner, IBk, using k = 10 neighbors) surpasses SVM on precision and recall criteria, but produced as many false positives as RF. As such, the SVM and KNN algorithms deserve further investigation to determine their applicability.

Lastly, regarding hybrid analysis, Rehman et al [11] proposed that performing a keyword-based analysis on both the constant strings (binaries) and the AndroidManifest.xml files of an Android app would assist in recognizing malicious apps more effectively. Optimal classification performance was achieved using SVM on constant string analysis and KNN on manifest file analysis, while KNN achieved an impressive 99.81% accuracy and 0.38% false positive rate when evaluating both malicious manifests and constant strings in the same feature vector. A list of relevant permission requests to be considered is included in their paper [11]. Their results suggest that a feature vector comprised of both an app's manifest declarations and some detected behaviors of said app would improve malware detection sensitivity, compared to classification purely based on either static or dynamic features.

Android OS system call logs performed by apps are proven by [10] to be reliable indicators of potential malware, while [11] and [9] have demonstrated the utility of permissions and intent filters declared in the manifest.xml file to achieve the same objective. Therefore, we hypothesize that a combined analysis of both feature sets improves Android malware detection accuracy as compared to purely static or dynamic feature sets. Additionally, the capabilities of SVM and KNN ML classifier algorithms in performing Android malware detection are prominently mentioned in the literature. Hence, we also intend to measure and compare the performance of these two ML algorithms in the hypothesis scenario. Our initial assumption in that regard is that there is no significant difference in the accuracy and true positive metrics of the two ML algorithms.

3. RESEARCH METHODOLOGY

One of the objectives of this paper is to determine if analyzing keywords in both the permissions requested in an Android app's manifest file and system call logs performed by said app can improve results compared to individual feature analysis. The other objective is to compare and evaluate the effectiveness of SVM and KNN algorithms in detecting malware, as they are proven in previous research to be successful algorithms due to the high detection accuracy that they provide[4].

We used a dataset, called M0Droid [15], which consists of 200 benign apps and 200 malware apps. The dataset was prepared and processed, according to the following subsections, to produce three feature vectors. These feature vectors consist of static (manifest keywords only), dynamic (system call logs only), and a hybrid of both feature sets. String manipulation scripts were coded and used to compile comma-separated value (CSV) files consisting of the integer number of times each feature appears in an app's manifest and/or the app's runtime behavior, with one CSV record per app.

3.1 Static Analysis

To compile the static feature vector, the APKTool was first used to decompile all the 400 Android package (APK) files of the dataset, and then a Windows PowerShell script was written to extract the decompiled AndroidManifest.xml file from all the dataset apps and isolate them into a separate folder. Next, a Python script was written to analyze each manifest, browsing for specific keywords that were known to signal the possible existence of malware, and log the number of times each keyword appeared in the manifest. The list of relevant manifest keywords (Table 1) is adapted from [11], but with some keywords that are absent from all manifests in this dataset removed from consideration. Thus, the static analysis vector comprises of the apps' individual histogram of relevant keywords detected in their AndroidManifest.xml file.

Table 1. Relevant manifest keywords (adapted from [11]**)**

Manifest element	Keyword
Permissions	READ_SMS, SEND_SMS, RECEIVE_SMS, WRITE_SMS, PROCESS_OUTGOING_CALLS, MOUNT_UNMOUNT_FILESYSTEMS, READ_HISTORY_BOOKMARKS, WRITE_HISTORY_BOOKMARKS, READ_LOGS, INSTALL_PACKAGES, MODIFY_PHONE_STATE, READ_CONTACTS, ACCESS_WIFI_STATE, DISABLE_KEYGUARD, CHANGE_WIFI_STATE,

Manifest element	Keyword
	SYSTEM_ALERT_WINDOW, CHANGE_CONFIGURATION
Intent Filters (Action)	BOOT_COMPLETED, SMS_RECEIVED, CONNECTIVITY_CHANGE, NEW_OUTGOING_CALL, UNINSTALL_SHORTCUT, INSTALL_SHORTCUT, VIEW, MAIN, CALL
Intent Filters (Category)	HOME, BROWSABLE, LAUNCHER
Intent Filters (Scheme)	sms, smsto, file, content
Intent Filters (Priority)	1000, 999, 2147483647, 100
Receivers	OnBootReceiver, AutorunBroadcastReceiver, SMSReceiver, SecurityReceiver, RepeatingAlarmService, AdNotification, GCMBroadcastReceiver, MessageReceiver, ActionReceiver

3.2 Dynamic Analysis

To compile the dynamic vector, system log files were obtained from the dataset apps. Virtual machines with the Android system installed were used to run the apps one by one. To extract the logs from the dataset files, 2 additional apps were installed to the environment. "Logcat Extreme" was used to save the log files of current applications, as this application allows to monitor the logs of one application by time. To run the application, "ES File Explorer" was used. Malware apps were individually installed and deleted after extracting the logs. After that, log files were extracted from the virtual machine and put in separate folders, namely "Malware" and "Goodware". As with static analysis, a custom script was coded to scan for the system calls of each application from the log files and put them in a CSV file that denotes their frequency of being called during the execution of the apps. The list of relevant system calls is adopted from [10] and shown in Table 2 below; as with static analysis, calls that are not present in the obtained feature vector are to be eliminated.

Table 2. Relevant system calls (adapted from [10]**)**

System call	Description
Read	Read data from files/device
Write	Write data to device/files
Open	Open file
Close	Close file
Unlink	Delete files
Chmode	Change permission
Lseek	Change location of read/write pointer
Getpid	Get process identifier
Access	Check access to a file
Rename	Rename a file
Dup	Creates copy of file descriptor
Brk	Change the location of program break
Ioctl	Manipulate device parameters of special files
Umask	sets the calling process's file mode creation mask (umask) to mask & 0777
Uname	returns system information
Munmap	deletes the mappings for the specified address range
Fsync	synchronize a file's in-core state with storage
Clone	create a child process
Mprotect	set protection on a region of memory
Sigprocmask	examine and change blocked signals
Select	synchronous I/O multiplexing
Writev	write data into multiple buffers
Sched_yield	yield the processor
Nanosleep	High-resolution sleep
Pread64	read from a file descriptor at a given offset
Stat64	get file status
Madvise	give advice about use of memory
Getdents64	get directory entries
Fentl64	manipulate file descriptor
Epoll_wait	wait for an I/O event on an epoll file descriptor
Clock_gettime	retrieve and set the time of the specified clock clk_id.

3.3 Hybrid Analysis

For the combined feature vector, the CSV files of the static and dynamic feature vectors were appended to each other, with the list of features extended accordingly. Only the dataset apps successfully preprocessed in both the static and dynamic analyses were included in the combined feature vector, with the unique identifiers of the apps serving as the reference to guide the association and attachment of the static and dynamic features for each considered app using a custom script.

The CSV files containing the feature vectors were then converted into the Attribute-Relation File Format (ARFF) and loaded into the Weka Explorer for classification. The Weka implementations of the SVM and KNN algorithms, which are SMO and IBk respectively, were used to classify the entries of the ARFF feature vector into malware and goodware. All the ML classifications were performed with 10-fold cross validation applied to the feature vector, in which the dataset was split up into several different (content-wise) but equally-sized subsets of goodware and malware. The rationale behind cross validation is to reduce the probability of overfitting while training the algorithm, and to reliably evaluate the generalizability of the trained algorithm towards detecting malware from unknown apps, without utilizing any additional apps other than what is already present in the experimental dataset. Each of the 10 feature vector subsets consisting of both goodware and malware was used as the training data 9 times and as the test data once (to evaluate the accuracy of

the proposed model), for a total of 10 iterations per round of ML classification.

The SMO (SVM) and IBk (KNN) algorithms were run for multiple rounds of classifying the same feature vector, with the configuration parameters (complexity c and kernel for SMO, neighbor count k and distance measure for IBk) changed at each round. The summary results of each round stating the various performance measures of the ML algorithm on detecting malware from the feature vector under the given algorithm configuration were recorded, and the best and worst result of all the tested configurations for each ML algorithm are tabulated in Table 3.

Table 3. ML configurations

ML Algorithm	Parameters	Label
SVM as SMO (Sequential Minimal Optimisation)	Default values (PolyKernel, c = 1)	SMO1
	PUK Kernel, c = 4	SMO2
	NormalizedPolyKernel, c = 4	SMO3
KNN as IBk (Instance Based Learner)	Default values (Euclidean distance, k = 1)	IBk1
	Euclidean distance, k = 5	IBk2
	Manhattan distance, k = 5	IBk3

The results were measured according to two key performance metrics: the detection accuracy (As shown in Table 4), which states the ratio between the number of correctly detected malwares and the total number of malwares; and the malware true positive rate (as shown in Table 4), which represents the percentage of successfully detected malwares. The latter is a particularly important measurement, as the ML algorithms were evaluated based on their generalizability of application, given limited training data.

Table 4. Performance metrics for classification evaluations

Evaluation metrics	Description	Formula
True Positive Rate	Android malwares that are successfully detected	$\frac{TP}{TP + FN}$
Accuracy	It is computed as the ratio between the number of correctly detected malwares and the total number of malwares	$\frac{TP + TN}{TP + TN + FP + FN}$

In this context, malware classification is considered as positive, and benign classification is taken as negative. The potential damage that may result from failing to flag actual malware is far greater than the annoyance of flagging benign apps as malware. Therefore, false positives may be tolerated to a certain extent depending on the real-world circumstances, but a significant number of false negatives would be hazardous and counterproductive to the purpose of implementing a malware detection system.

4. RESULTS AND DISCUSSION

Since a specialized malware testing environment is not required to compile the static feature vector from the dataset, a static ML analysis of the app manifest keywords was performed as an initial test of the proposed method. The preparation of the dynamic analysis and hybrid analysis feature vectors were proposed in this paper but left to be done in future experiments. This is because specialized scripts and emulation tools are required to properly log and extract the system call sequences while running the apps with full permissions and functionality for complete app coverage in a controlled environment. Additionally, there are inherent risks with executing apps known to be malware, which we lack the resources to adequately deal with at the time of writing. For instance, Singh and Hofmann [10] reported that their laptop crashed while running the apps, and that another device was compromised by ransomware.

Table 5 shows the measurements obtained by performing classification of the static feature vector through multiple configurations of the two ML algorithms in Weka.

Table 5. Static (manifest) vector classification results

ML Algorithm	Accuracy	True positive
SMO1	79.75%	64.00%
SMO2	79.50%	66.50%
SMO3	78.00%	70.50%
IBk1	83.00%	80.00%
IBk2	79.75%	80.00%
IBk3	78.75%	80.00%

The static analysis results find no significant difference in the detection accuracy of both SVM (SMO) and KNN (IBk) algorithms. However, the detection accuracy does not differentiate between false positives and false negatives, and merely suggests that approximately 80% of benign or malware apps are correctly predicted through static analysis only.

The malware true positive rates, however, provide a better insight into the performance of the two ML algorithms. In Table 5, KNN (IBk) is shown to outperform SVM (SMO) by around 10% of the tested apps in terms of the percentage of malware successfully detected by the algorithm. Despite that, the detection accuracies remain similar for both ML algorithms. This implies that the KNN (IBk) algorithm gives more false positives (benign apps flagged as malware) as a tradeoff for flagging malware more comprehensively. Overall, when the manifest keywords of apps are considered as features, there is minimal difference in the detection performance of SVM and KNN algorithms with respect to accuracy and true positive rate. However, this conjecture may or may not hold true for dynamic and hybrid feature analyses once they are performed.

5. CONCLUSION

In this paper, we used keywords as the primary malware detection features with the aim of determining if analyzing keywords in both the permissions requested in an app's manifest file and system call logs could lead to better results compared to individual feature analysis. We applied and compared two machine learning algorithms, called SVM and KNN, to classify the Android apps into benign or malware. The evaluation of

results was performed based on measuring the accuracy and true positive rates of Android malware detection. Our results show the accuracy rate of 79.08% and 80.50% and true positive rate of 67.00% and 80.00% for SVM and KNN, respectively.

As a future work, it would be of interest to expand our methodology by considering two categories of dynamic and hybrid malware analysis and compare the results with our findings in this research. Furthermore, another potential research direction would be to investigate the application of unsupervised learning algorithms on the same or different dataset in order to detect Android malware.

6. REFERENCES

[1] Ericsson, "Ericsson Mobility Report," 2018. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2018/emr-q2-update-2018.pdf.

[2] R. Unuchek, "Mobile malware evolution 2017," *Kaspersky Lab*, 2018. [Online]. Available: https://securelist.com/mobile-malware-review-2017/84139/.

[3] M. Kakavand, N. Mustapha, A. Mustapha, M. T. Abdullah, and H. Riahi, "Issues and Challenges in Anomaly Intrusion Detection for HTTP Web Services," *J. Comput. Sci.*, vol. 11, no. 11, pp. 1041–1053, 2015.

[4] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, "Machine learning aided Android malware classification," *Comput. Electr. Eng.*, vol. 0, pp. 1–9, 2017.

[5] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs," in *Proceeding CCS '14 Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[6] B. Baskaran and A. Ralescu, "A Study of Android Malware Detection Techniques and Machine Learning," *MAICS*, pp. 15–23, 2016.

[7] S. Y. Yerima, S. Sezer, and I. Muttik, "Android Malware Detection Using Parallel Machine Learning Classifiers," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, 2014.

[8] M. Al Ali, D. Svetinovic, Z. Aung, and S. Lukman, "Malware Detection in Android Mobile Platform using Machine Learning Algorithms," in *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, 2017, pp. 4–9.

[9] M. Kumaran, "Lightweight Malware Detection based on Machine Learning Algorithms and the Android Manifest File," in *2016 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2016, pp. 1–3.

[10] L. Singh and M. Hofmann, "Dynamic Behavior Analysis of Android Applications for Malware Detection," in *2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)*, 2017, no. 2013, pp. 1–7.

[11] Z. Rehman, S. Nasim, K. Muhammad, and J. Weon, "Machine learning-assisted signature and heuristic-based detection of malwares in Android devices," *Comput. Electr. Eng.*, vol. 69, pp. 828–841, 2018.

[12] M. Tan, M. Yu, Y. Wang, S. Li, and C. Liu, "Android Malware Detection Combining Feature Correlation and Bayes Classification Model," in *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, 2017, pp. 1–5.

[13] J. Garcia, M. Hammad, and S. A. M. Malek, "Lightweight , Obfuscation-Resilient Detection and Family Identification of Android Malware," *J. ACM Trans. Softw. Eng. Methodol.*, vol. 9, no. 4, pp. 1–27, 2016.

[14] M. Hall *et al.*, "The WEKA Data Mining Software : An Update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18.

[15] M. Damshenas, A. Dehghantanha, K. R. Choo, M. Damshenas, A. Dehghantanha, and K. R. Choo, "M0Droid : An Android Behavioral-Based Malware Detection Model," *J. Inf. Priv. Secur.*, vol. 6548, no. September, 2015.