A Study of Android Malware Detection Techniques and Machine Learning

Balaji Baskaran and Anca Ralescu
EECS Department
University of Cincinnati
Cincinnati, OH 45221 - 0030
baskarbi@mail.uc.edu, anca.ralescu@uc.edu

Abstract

Android OS is one of the widely used mobile Operating Systems. The number of malicious applications and adwares are increasing constantly on par with the number of mobile devices. A great number of commercial signature based tools are available on the market which prevent to an extent the penetration and distribution of malicious applications. Numerous researches have been conducted which claims that traditional signature based detection system work well up to certain level and malware authors use numerous techniques to evade these tools. So given this state of affairs, there is an increasing need for an alternative, really tough malware detection system to complement and rectify the signature based system. Recent substantial research focused on machine learning algorithms that analyze features from malicious application and use those features to classify and detect unknown malicious applications. This study summarizes the evolution of malware detection techniques based on machine learning algorithms focused on the Android OS.

Introduction

According to a 2014 research study (RiskIQ(2014)), malicious applications in Google Play Store have increased 388% between 2011 and 2013.

As the initial part of our research, we conducted an extensive study where we analyze the current trends and approaches on detecting the malwares on Android Systems using Machine Learning techniques. The overall goal of this study is to identify the research so far on Android Malware detections using Machine Leaning Techniques. With this analysis we can formulate a defense mechanism specifically to counteract the Update attack, the most difficult intrusion technique to detect and eliminate.

Update Attack: In Android update attack is defined as the benign application installed in the system downloads malicious payloads while updating itself or downloads third party malicious applications and installs in the system. This type of attack is very hard to detect because the original application is benign. Unless we track the installed previous versions and the application after the update we cannot detect the malicious activity. We aim to give a brief approach on counteracting the update attack with the survey on recent trends on Malware detection.

Based on the current attack trends and analysis of the present literatures, (Raveendranath et al.(2014)Raveendranath, Rajamani, Babu, and Datta) lists the types of malwares as follows:

1. **Information Extraction**
   Compromises the device and steals personal information such as IMEI number, user's personal information, etc.
2. **Automatic Calls and SMS**
   User's phone bill is increased by making calls and sending SMS to some premium numbers
3. **Root Exploits**
   The malware will gain system root privileges and takes control of the system and modifies the information.
4. **Search Engine Optimizations**
   Artificially search for a term and simulates clicks on targeted websites in order to increase the revenue of a search engine or increase the traffic on a website.
5. **Dynamically Downloaded code**
   An installed benign application downloads a malicious code and deploys it in the mobile devices.
6. **Covert channel**
   A vulnerability in the devices that facilitates the information leak between the processes that are not supposed to share the information.
7. **Botnets**
   A network of compromised mobile devices with a BotMaster which is controlled by Command and Control servers (C&C). Carry out Spam delivery, DDDos attacks on the host devices.

From this point on, the structure of the paper is as follows. Section is a general overview of current security deployed by play-store. Classification of various methods used in detecting malwares in Android systems is presented in Section . The paper concludes in Section

Overview of Android System Security

Google Play Store uses an in-house malicious application detection system called Bouncer. But researchers have proved that Bouncer's ability to detect the malicious application is minimal and they could successfully publish a prototype malicious application in play-store. Android Play-Store uses application's meta data such as user's rating and user's comments to flag a malicious application. But by the time the malicious application is detected, it could have made enough damage to the affected mobile system.

Malware authors use many techniques to evade the detection such as (a) code obfuscation technique, (b) encryption, (c) including permissions which are not needed by the application, (d) requesting for unwanted hardwares, (e) download or update attack in which a benign application updates itself or another application now with malicious payload, which is very tough to detect. This also encourages the need for new researches on the detection techniques, including machine learning based techniques. Many studies have shown that machine learning algorithms to detect the malicious activities are successful in detecting them with very high accuracy.

Android Malware Detection

Based on the features used to classify an application, we can categorize the analysis as Static and Dynamic. Static analysis is done without running an application. Examples of static features include, (a) permissions, (b) API calls which can be extracted from the AndroidManifest.xml file. Dynamic analysis deals with features that were extracted from the application while running, including (a) network traffic, (b) battery usage, (c) IP address, etc. The third type of analysis is hybrid analysis which combines the features from static and dynamic techniques. The rest of this section describes the features extracted from the application and machine learning algorithm used.

Static Analysis

In static analysis, the features are extracted from the application file without executing the application. This methodology is resource and time efficient as the application is not executed. But at the same time, this analysis suffers from code obfuscation techniques the Malware authors employ to evade from static detection techniques. One of very popular evasion technique is the Update Attack: a benign application is installed on the mobile device and when the application gets an update, the malicious content is downloaded and installed as part of the update. This cannot be detected by static analysis techniques which will scan only the benign application.

The most commonly used static features are the Permission and API calls. Since these are extracted from the application *AndroidManifest.xml* and influence the malware detection rate to a high extent, extensive research has been made with these as features as well as combined with other features extracted from meta-data available in Google PlayStore such as version name, version no., author's name, last updated time, etc.,

(Sahs and Khan(2012)) used permissions and Control Flow Graphs(CFG) as features and used One-class Support Vector Machine(SVM). The most of training data are benign applications and the classifier will classify a sample as malicious only if it is sufficiently different from the benign class.

(Shabtai et al.(2010)Shabtai, Fledel, and Elovici) used permission, framework methods and framework classes for their classification system.

(Sanz et al.(2012)Sanz, Santos, Laorden, Ugarte-Pedrero, and Bringas) extracted the strings in the application, permissions, user rating, number of ratings, size of the application and used Bayesian Networks, J48 Decision Tree and Random Forest, SVM with SMO kernel. A total of 820 samples were used to test and the authors concluded that they could achieve a very high accuracy with less false positive rate.

(Ghorbanzadeh et al.(2013)Ghorbanzadeh, Chen, Ma, Clancy, and McGwier) used Neural Networks to detect an application's category from permissions by means of multi layered feed forward networks. A feed forward Neural Network is built with two layers each containing 10 neurons. The hidden layer contains sigmoid transfer function and the output linear transfer functions was deployed. The suthoud assumed that the permissions declared in the manifest file may be manipulated by the malware authors and they may misrepresent the categories declared in the manifest. So to simulate this property, the authors permuted permissions of 50% of the test data and fed into the network.

(Yerima et al.(2013)Yerima, Sezer, McWilliams, and Muttik) used 2000 applications with 1000 malicious and 1000 benign applications. They extracted features like Permissions, API calls, Native Linux System commands and various features from manifest and class files. The malware authors embed native linux commads such as chnown, mount, remount, etc., and run them in the Android system when the application is launched. Mutual information (entropy) is used to rank the features and then a Bayesian Classifier is used for classification.

(Samra et al.(2013)Samra, Yim, and Ghanem) extracted features from AndroidManifest.xml such as count of xml elements, application specific information such as name, category, description, rating, package info, description, rating values, rating counts and price. The information from 18174 android application with 4612 business category and 13535 tools were extracted by using web crawlers. They were clustered using K-Means clustering.

(Peiravian and Zhu(2013)) utilized permission, API calls and the combination of both as features. The two types of permissions in Android, requested permission and required permission are used to express an application as a binary vector where $P_i = 1$ iff the Manifest.xml has the i^{th} permission. Same as permission, API calls are also expressed as a binary vector with $API_i = 1$ iff there is the API call made in the application. These two features are concatenated and the third feature is formed. A total of 2510 samples including 1260 are malicious and 1250 benign are used. The authors concluded that Bagging, an ensemble classification method has the best performance in classifying all created datasets.

(Liu(2013)) investigated three specific types of malware: *SMS-related, control-related and spy-related.* An application's permission and ¡uses-feature¿ xml tag which requests the necessary hardware devices needed to run the application, is extracted and used as features. Information Gain is used to select important features and SVM with the basic classifier is used to detect the malicious application. The authors could detect Spy-related applications with an accuracy of 81%, SMS-related with malicious applications with an accuracy of 97% and Control-related malicious applications with an 100% accuracy and could detect benign applications with an accuracy of 88%.

(Glodek and Harang(2013)) constructed five Random Forests with 5-fold cross validation and compared their performance in detecting malicious applications. They have used 500 malicious and 500 benign from North Carolina State University's malware project. Permission, broadcast receivers and native code embedded in the application are used as features and they concluded that their method outperforms a lot of commercial anti virus detection tools.

(Jerome et al.(2014)Jerome, Allix, State, and Engel) extracted the opcodes from class.dex file and translated into opcode sequences, binary sequences of k-grams that characterize the least functionalities required by a program. They trained their model with Gnome Project dataset and randomly picked 1246 applications from the Google Play Store. The test dataset consists of 25,476 malware samples, 15670 benign applications from VirusTotal. Information Gain was used to select important features among the available ones. The author used a linear implementation of SVM to classify application samples. The results were compared with the detection rate of 25 anti virus tools. The study release an interesting signature patterns of Malware, Goodware, False Positives and False Negatives of their classifier. The false negatives were found out to be adwares and they were also considered a threat by the tool.

(Pehlivan et al.(2014)Pehlivan, Baltaci, Acarturk, and Baykal) used 3748 application packages, developed C# scripts to automatically extract about 182 attributes that include Permissions, version no and version name of the applications. The study compared feature selection methods such as Gain Ratio Attribute Evaluator, Relief Attribute Evaluator, Control Flow Subset Evaluator, and Consistency Subset Evaluator and machine learning algorithms Bayesian classification, Classification and Regression Tree (CART), J48 DT, RF, SMO. Using the feature selection methods, they came up with 97 features that could represent the whole dataset. Finally the authors conclude that, with just 25 features, the Control Flow Subset Evaluator selection gave a good performance and Random forest and J48 performed better than Bayesian classifier.

(Chan and Song(2014)) analyzed 796 benign and 175 malicious applications for their study. Permissions used from the manifest.xml file and API call info from the classes.dex file are extracted and with Information Gain they selected a set of 19 relevant API calls. They compared the results obtained by machine learning algorithms such as Naive Bayes, SVM with SMO algorithm, RBF Network, Multi Layer Perceptron, Liblinear, J48 decision tree and Random Forests.The authors concluded that the were able to get 90% of the accuracy by using the API calls and permission combined than using the individual features alone.

(Liu and Liu(2014)) combined the two types of permissions, required permission and requested permission and designed a two layer approach with these features and employed machine learning algorithms to detect the malicious applications. A total of 28,548 benign applications and 1,536 malicious applications and permission pairs i.e., combination of any two requested permission are analyzed. The two layered approach helped to balance the detection accuracy and detection speed of the classifier. In Phase 1, requested permissions and the J48 Decision Tree algorithm is used in detection and in Phase 2, requested permission pairs and the J48 Decision Tree is used for detection. If there is any contradiction in the results obtained from both the phases, used permission pairs and J48 is used to classify again. The authors achieved a good result with this approach and recommended using the permission in component level than the application level for better detection of malicious activities.

(Ideses and Neuberger(2014)) used permission, broadcast receivers and activities, byte code fragments, system-calls as features and trained SVM with the training dataset. The researchers tested their proposed Malware detection system with a security tester for benchmarking where their system was tested with 7,000 samples. They conclude that their system could achieve about 99.3% positive rate with just 0.14% false alarm rate.

(Yerima et al.(2014a)Yerima, Sezer, and McWilliams) presented and analyzed three Bayesian classification approaches for detecting Android malwares. Permissions and code based properties such as API calls, both Java system based and Android system based, Linux and Android system commands are also extracted from the sample applications. A list of top 20 permissions and top 25 API calls used by benign and malicious applications are presented.

(Fazeen and Dantu(2014)) used combines Intentions of the applications esp., Task Intentions with permission as feature in developing their model. At first the requested permissions are extracted and a histogram is constructed for that task-intention category. Normalizing this results in an I shaped PMF. This shape is used to compare and detect the unknown applications as benign or malicious based on their Task Intentions. The system works as follows:

- **Phase I** trains and uses machine learning algorithms to find the task intentions of the sample applications.
- **Phase II** uses the knowledge from Phase I to find the task intention of an unknown application and classify as benign or malicious. The I shape is compared with the requested permission by using a using a matching ratio, that is generated by a machine learning algorithm. If the ratio is in a threshold, then the application is potentially safe. The authors used Naive Bayes, Multi Layered Perceptron and Random Forests and compared their performances.

(Xiaoyan et al.(2014)Xiaoyan, Juan, and Xiujuan) extracted permissions from the manifest and represented as a binary vector. Then Principle Component Analysis (PCA) is performed to select the best features. A linear SVM is trained to classify the app samples. The author compares the result with other classifiers such as J48 Decision Tree, Naive Bayes, BayesNet, CART, RandomForest and concludes that SVM gives a better performance.

(Yerima et al.(2014b)Yerima, Sezer, and Muttik) came up with a parallel implementation of their system to detect malicious android applications. They used application related feature such as permissions, Standard OS and android framework commands. They developed parallel implementation of Logistic function based classifier, Naive Bayes - probabilistic method and PART, RIDOR which are rule based classifier. with the features extracted, the classification is performed with the individual algorithms and then parallel implementation is carried out. The maximum probability scheme fetched an accuracy of 97.5%.

(Idrees and Rajarajan(2014)) combines permissions and Intents and used 292 applications for training and 340 for testing their model. The study describes some usage statistics of benign and malicious applications with regards to intents and permissions and developed Naive Bayes, Kstar, Prism to detect the maliciuos applications from benign applications.

(Munoz et al.(2015)Munoz, Martin, Guzman, and Hernandez) The authors collected the information from Google Play meta-data such as intrinsic application features, Application category, Developer related feature, certificate related feature, social related feature. They concluded that certificate and developer information, intrinsic application feature are the most promising feature to determine a malware with just meta data.

(Westyarian et al.(2015)Westyarian, Rosmansyah, and Dabarsyah) used 205 benign and 207 malicious application files and extracted API calls that are only related to the permission declared in ¡used-permission¿ label in manifest.xml file. The study concluded that 97% of the malware requests *telephonyManager* and *connectivityManager* are the most important features. Random forest classification obtains 92.4% with cross validation as feature selection algorithm and SVM obtain 91.4% with percentage split as feature selection algorithm.

(Chuang and Wang(2015)) collected API calls from benign application separately and API calls from malicious applications separately and used these as features for classifying an unknown sample. The APIs in the unknown are ranked according to their difference in the number of occurrences in benign and number of occurrences in malicious applications. Then they deploy single a model approach where they will combine the two feature sets into a single vector. In Malicious model approach only the hypothesis from Malicious tended APIs is used for classification. The Hybrid approach combines two separately trained SVM models. These results are then compared to predict whether the unknown sample is malicious. The Hybrid model behaved much better than the Malicious model but the single model obtained from combined features outperformed the Malicious model.

Table 1 shows the top frequent used features in static analysis. Table 2 summarizes the top features that are combined with other features to produce better detection rate. By observing the table 1 and table 2, it can be clearly seen that Permission and API calls, the two features extracted from Manifest file and .dex file produces higher detection rate and inorder to make them more fail safe these can be combined with other features such as mate-data collected from Google Play Store or the features extracted from the XML elements.

Table 1: Topmost used features in static analysis

Sl. No.	Feature
1	Permission
2	API calls
3	Strings extracted
4	Native commands
5	XML elements
6	Meta data
7	Opcodes from .dex file
8	Task Intents

Table 2: Top features combined with other features in static analysis

Feature	Combined With
Permissions	Broadcast receivers Uses-feature tag Android OS commands API calls meta-data opcodes
Features extracted from manifest files and class files	API calls

Dynamic Analysis

(Wei et al.(2012)Wei, Mao, Jeng, Lee, Wang, and Wu) used Droidbox, a tool to monitor the application real time, to dynamically analyze the behavior of android applications. IP address of the source is extracted from the network traffic after then application is run in a sandbox environment. The research concentrated only on the network characteristics of the malwares leveraging the fact that they will find their next target soon. The extracted IP address is used to find the spatial address using external services and to determine the uniformity of geographic distribution of the hosts because infected hosts will be distributed worldwide. After extracting the features, a MxN APP-GEO Matrix is constructed with M representing the android applications(rows) and N network features. ICA (Independent Component Analysis) to extract the latent concept or sparse from the noisy spamming data. The researchers used Weka and FastICA, the two open source libraries to evaluate their model. A total of 310 malware samples were used and they could achieve about 93% accuracy rate.

(Ham and Choi(2013)) used 30 normal apps and 5 malware samples (GoldDream, PJApps, DroidKungFu2, Snake and Angry Birds Rio Unlocker) in this study. The allocated resources when the app starts are monitored and the behavioral pattern is extracted. hese resource data are stored within the device and are converted into feature vectors. Each feature is subdivided to 7 categories, 1. Network, SMS, CPU, and power usage, Process (like ID, Name , running process), memory Native, Dalvik and other and Virtual Memory.32 features are related to malware detection and applied

Information Gain to select features. They used Naive Bayes, Random Forest LR, SVM with 10 fold cross validation.

The authors concluded that Naive Bayes/LRs confusion matrix are irregular in distribution with these features. SVM correctly classified normal type data almost 100% but falsely detected malicious applications as benign. Random Forests outperformed all the algorithms and correctly classifies the majority of normal and malware applications.

(Lu et al.(2013)Lu, Zulie, Jingju, and Yi) compared Bayesian method alone and Bayesian method combined with Chi Square feature selection method results are compared to evaluate the performance of the two ML algorithms. The study concluded that Bayesian method with Chi Squared yielded an accuracy of 89% while Bayesian method alone yielded 80%.

(Tenenboim-Chekina et al.(2013)Tenenboim-Chekina, Barad, Shabtai, Mimran, Rokach, Shapira, and Elovici) used 5 to 10 self-written Trojan malware with two versions of the malware, one benign and other malicious which is repacked version of the benign with malicious code. While the application is running, Many network based features are extracted. The self-written applications are installed in the devices and their behavior was collected and analyzed. This helps the traffic patterns distinguishable from benign and malicious. Feature measurements are performed at fixed time intervals and then aggregation functions are computed over these measurements. Cross feature analysis is used to explore the correlation between features. The deviations caused by abnormal activities from normal activities are observed. With labeled samples a threshold of deviation is obtained during the algorithm formulation. The study could successfully detect the repacked malicious applications using the network features learned.

(Alam and Vuong(2013)) rooted the mobile device to get the details such as, data being sent by applications, IP address being communicated, number of active communications, the system calls and used Random forest with 1330 malicious and 407 benign applications. The authors concluded that with more trees and less feature per tree in the Random Forest, they could achieve an accuracy of 99%.

(Mas'ud et al.(2014)Mas'ud, Sahib, Abdollah, Selamat, and Yusof) monitored the system call of 30 normal applications and 30 malicious applications. The study compares 5 feature selection methods and 5 Machine Learning classifiers KNN, Decision Tree, Multi Layer Perceptron (MLP), Random Forests, Naive Bayes. The applications are run in real devices and are monitored for system calls generated by Strace, an application used to log various system activities in android systems. Then the features are selected by Information Gain and Chi-Square. A set of 5 feature sets are devised and used to compare the efficiency of 5 Machine Learning algorithms. The study concluded that the MLP achieves a highest accuracy and True Positive rate for one feature set while J48 Decision Tree achieves high performance rate for another feature set.

(Ng and Hwang(2014)) also used Strace to monitor the application for 60 secs. The features taken into account were Strace logged ProcessID, system calls, returned values and times between consecutive system calls. The no of times each invoked call is counted. PCA is used in selecting the important feature and then the classifier classifies the application sample malicious or benign based on anomaly score obtained by the input. The author compared their system's performance with classifiers such as Naive Bayes, J48 Decision Tree and SVM and claims that they could achieve 98.4% detection rate.

(Kim and Choi(2014)) Linux based features are extracted from the Android Os and used as feature to detect malicious applications. There were 59 features obtained like, Memory, CPU, Network, etc. 6 malwares were run on the system and the system is monitored to collect the above said features. Every 10 seconds the data is collected and sent over to a server and the server does the classification. Out of 59 features, 36 are selected and the results are compared before and after applying feature selection. It has been said that the feature selection improves the accuracy and reduces the False Positive Rate of the classification.

(Kurniawan et al.(2015)Kurniawan, Rosmansyah, and Dabarsyah) used Logger, a default application which is inbuilt in Android was used to extract the sum of Internet traffic, percentage of battery used and battery temperature for every minute. These information collected as set of features and is fed into weka, a open source learning library for testing and training with Naive Bayes, J48 decision tree and Random Forest algorithms. The author concluded that Random Forest has high accuracy of 85.6% with these features and proposes other features that can be combined with existing system to improve the accuracy.

Table 3 summarizes the most frequently used features in Dynamic analysis. As seen, Network traffic which includes data packets sent, and other behavioral patterns can lead to quick detection of malicious activity. Tracing the IP address can help us to get the geographical landscape of the attack surface. Other than this, SMS, information logged by Logger and Strace is very much helpful in achieving a higher detection rate.

Hybrid Analysis

The hybrid methodology involves combining static and dynamic features collected from analyzing the application and extracting information while the application is running, respectively. Though it could increase the accuracy of the detection rate, it makes the system cumbersome and the analysis process time consuming.

(Shabtai(2010)) extracted opcodes from the executable and proposed a framework that monitors the device state at every instant such as CPU usage, number of packets sent over network, number of running process, battery level. Applications are downloaded from play store. The authors examine the applicability of Knowledge Based Temporal Abstraction (KBTA) which helps continuously monitor and measure events on a mobile system. The study was concluded with 94% detection rate with the feasibility of running such a system with just 3% power consumption. The authors also recommend the implementation of SELinux to enhance the security mechanisms of Android. Efficiency of Machine Learning algorithms such as Decision Trees, Naive

Table 3: Top features used in Dynamic analysis

Sl. No.	Feature	Machine Learning Algorithm
1	Network, SMS, Power Usage, CPU, Process info, Native and Dalvik Memory	Naive Bayes, Random Forest, SVM with SMO algorithm
2	Data packets being sent, IP address, No. of active communications, System calls	Random Forest
3	Process id, System calls collected by Strace, Returned values, Times between consecutive calls	Naive Bayes, Decision Trees, SVM
4	Network Traffic - Destination IP address	Classification
5	System calls collected Strace, Logs of System activities	J48 Decision Trees, KNN, ST, Multi Layer Perceptron
6	Data collected by Logger, Internet traffic, Battery percentage, Temperature collected every minute	Naive Bayes, J48 Decision Trees

Bayes, BayesNet, K-Means, Histogram and Logistic Regression are compared and evaluated.

(Xu et al.(2013)Xu, Yu, Chen, Cao, Dong, Guo, and Cao) proposes a system, MobSafe that combines the dynamic (Android Security Evaluation Framework - ASEF) and static (Static Android Analysis Framework - SAAF) analysis methods. They used 100,000 active android applications from AppChina. Static features include the information from apk files and decoded smali files were analyzed to extract the permissions, heuristic patterns, and program slicing for functions of interest NO ML: analyzing takes within 2mins and For dynamic analysis ADB logging and TCP DUMP were used. The application is launched on a Virtual Machine and subjected to human level interaction simulation. This is then compared with a CVE library and its Internet activity with Google Safe Browser API to check the URLS the app requested is malicious or not.

(Wei et al.(2013)Wei, Zhang, Ge, and Hardy) analyzed 96 benign applications and 92 malware samples to extract static features such as software profiles. Strace is used to record system calls along with the process ID while the application is running for dynamic features. These information are collected and applied over Support Vector Machones and Naive Bayes.

(Feldman et al.(2014)Feldman, Stadther, and Wang) proposes a system, Manilyzer which uses requested permissions, High Prior receivers, Low version numbers and abused services as features and test their model with 617 applications 307 malicious 310 benign applications. Efficiency of Naive Bayes, SVM, K-Nearest Neighbours, J48 Decision Trees are compared and concluded with saying the most number of malware were labelled with 1.x application version number. And also that high priority intent filter were closely associated with SMS malware as 88% of the applications with this characteristics were malicious. Manilyzer is less effective but can be enhanced with other features associated with permissions such as API calls. Manilyzer is effectively used to detect adware spywae and SMS malware.

(Hsieh et al.(2015)Hsieh, Wu, and Kao) studies and summarizes the threat from malware on handheld devices, how malware writers evade the anti virus detection on mobile devices and the techniques that were used to deliver the malicious payload onto the mobile systems. The authors conclude the research by giving out the analysis methodologies in detecting the malwares.

(Lindorfer et al.(2015)Lindorfer, Neugschwandtner, and Platzer) proposes a system MARVIN with large-scale Android malware analysis sandbox ANDRUBIS to provide users with a risk assessment for an application. They developed an end user app into which users will submit their app and receive the score that tells the users how malicious the application is. MARVIN has 98.24% accuracy with less than 0.04% false positives. Static features such as permission, API Calls based on used-permission, reflection API, cryptographic API, dynamic loading of code are combined with dynamic features such as File operations, Network operations, Phone events, Data leaks, Dynamically loaded code, dynamically registered broadcast receivers. SVM with a linear classifier is used as a model of classification. The authors made use of labeled data set obtained from play-store, gnome project and used their system to classify samples from VirusTotal.

Table 4 summarizes the static an dynamic features combined and used as part of hybrid analysis. As seen from Table 1, Permissions is used mostly as a feature along with dynamic features like Logged information, API call traces and Network Traffics.

Future Goals on Counteracting the Update Attack

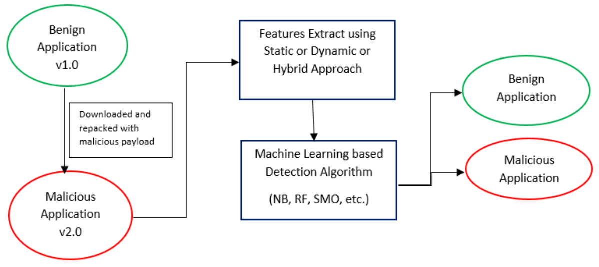
With this analysis, it can be seen that only very few researches have been conducted which deals with counteracting the update attack. As discussed in the previous section, the update attack is so hard to detect because with the previous version installed on the device is benign and it is not sure when the malicious activity os performed. The key to detect update attack is to keep track of the functions of the previous benign applications that are installed on the android devices. When the application is updated we can find the difference between the old and recent versions of the application and with combining the machine learning techniques and the acquired knowledge from malicious malware files, we can easily detect the update attack and the malicious in-

Table 4: Top features used in Hybrid analysis

Sl. No.	Feature	Machine Learning Algorithm
1	CPU Usage, No. of packets sent, No. of running process, Battery level	Naive Bayes, Decision trees, Random Forest,BayesNet, K-Means, Logistic Regression
2	Static: Information from apk, Decoded smali files Dynamic: ADB Logging, TCP Dump	Random Forest
3	Static: Static: Software profile Dynamic: Strace - system calls and process id,	Naive Bayes, SVM
4	Static: Permission, High priority receivers, version numbers	Naive Bayes, SVM, K-NN, J48 Decision Trees
5	Static: Permission, API Calls based on used-permission, reflection API, cryptographic API, Dynamic: loading, File operations, Network operations, Phone events, Data leaks, Dynamically loaded code, dynamically registered broadcast receivers	SVM with linear function

tent of the malware author.

Figure 1: Counteracting the update attack


Benign Application v1.0
Downloaded and repacked with malicious payload
Malicious Application v2.0
Features Extract using Static or Dynamic or Hybrid Approach
Machine Learning based Detection Algorithm (NB, RF, SMO, etc.)
Benign Application
Malicious Application

Conclusion

This study summarizes recent developments in android malware detection using machine learning algorithms. Detection techniques and systems that uses static, dynamic and hybrid approaches are discussed and highlighted. A method that could lead to potential counteracting the update attack is discussed. The unavailability of a larger android malware dataset remains a great problem in evaluating various approaches. With a proper dataset shared among researchers, a system that learns a new malware and share that knowledge to all the mobile devices, so that they can protect themselves from future attacks, could be developed.

References

M.S. Alam and S.T. Vuong. Random forest classification for detecting android malware. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 663–669, Aug 2013. doi: 10.1109/GreenCom-iThings-CPSCom.2013.122.

P.P.K. Chan and Wen-Kai Song. Static detection of android malware by using permissions and api calls. In *Machine Learning and Cybernetics (ICMLC), 2014 International Conference on*, volume 1, pages 82–87, July 2014. doi: 10.1109/ICMLC.2014.7009096.

Hsin-Yu Chuang and Sheng-De Wang. Machine learning based hybrid behavior models for android malware analysis. In *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on*, pages 201–206, Aug 2015. doi: 10.1109/QRS.2015.37.

M. Fazeen and R. Dantu. Another free app: Does it have the right intentions? In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pages 282–289, July 2014. doi: 10.1109/PST.2014.6890950.

S. Feldman, D. Stadther, and Bing Wang. Manilyzer: Automated android malware detection through manifest analysis. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pages 767–772, Oct 2014. doi: 10.1109/MASS.2014.65.

M. Ghorbanzadeh, Yang Chen, Zhongmin Ma, T.C. Clancy, and R. McGwier. A neural network approach to category validation of android applications. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 740–744, Jan 2013. doi: 10.1109/ICNC.2013.6504180.

W. Glodek and R. Harang. Rapid permissions-based detection and analysis of mobile malware using random decision forests. In *Military Communications Conference, MILCOM 2013 - 2013 IEEE*, pages 980–985, Nov 2013. doi: 10.1109/MILCOM.2013.170.

Hyo-Sik Ham and Mi-Jung Choi. Analysis of android malware detection performance using machine learning classifiers. In *ICT Convergence (ICTC), 2013 International Conference on*, pages 490–495, Oct 2013. doi: 10.1109/ICTC.2013.6675404.

Wan-Chen Hsieh, Chuan-Chi Wu, and Yung-Wei Kao. A study of android malware detection technology evolution.

In *Security Technology (ICCST), 2015 International Carnahan Conference on*, pages 135–140, Sept 2015. doi: 10.1109/CCST.2015.7389671.

I. Ideses and A. Neuberger. Adware detection and privacy control in mobile devices. In *Electrical Electronics Engineers in Israel (IEEEI), 2014 IEEE 28th Convention of*, pages 1–5, Dec 2014. doi: 10.1109/EEEI.2014.7005849.

F. Idrees and M. Rajarajan. Investigating the android intents and permissions for malware detection. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*, pages 354–358, Oct 2014. doi: 10.1109/WiMOB.2014.6962194.

Q. Jerome, K. Allix, R. State, and T. Engel. Using opcode-sequences to detect malicious android applications. In *Communications (ICC), 2014 IEEE International Conference on*, pages 914–919, June 2014. doi: 10.1109/ICC.2014.6883436.

Hwan-Hee Kim and Mi-Jung Choi. Linux kernel-based feature selection for android malware detection. In *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, pages 1–4, Sept 2014. doi: 10.1109/APNOMS.2014.6996540.

H. Kurniawan, Y. Rosmansyah, and B. Dabarsyah. Android anomaly detection system using machine learning classification. In *Electrical Engineering and Informatics (ICEEI), 2015 International Conference on*, pages 288–293, Aug 2015. doi: 10.1109/ICEEI.2015.7352512.

M. Lindorfer, M. Neugschwandtner, and C. Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 2, pages 422–433, July 2015. doi: 10.1109/COMPSAC.2015.103.

Wen Liu. Mutiple classifier system based android malware detection. In *Machine Learning and Cybernetics (ICMLC), 2013 International Conference on*, volume 01, pages 57–62, July 2013. doi: 10.1109/ICMLC.2013.6890444.

Xing Liu and Jiqiang Liu. A two-layered permission-based android malware detection scheme. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 142–148, April 2014. doi: 10.1109/MobileCloud.2014.22.

Yu Lu, Pan Zulie, Liu Jingju, and Shen Yi. Android malware detection technology based on improved bayesian classification. In *Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2013 Third International Conference on*, pages 1338–1341, Sept 2013. doi: 10.1109/IMCCC.2013.297.

M.Z. Mas'ud, S. Sahib, M.F. Abdollah, S.R. Selamat, and R. Yusof. Analysis of features selection and machine learning classifier in android malware detection. In *Information Science and Applications (ICISA), 2014 International Conference on*, pages 1–5, May 2014. doi: 10.1109/ICISA.2014.6847364.

A. Munoz, I. Martin, A. Guzman, and J.A. Hernandez. Android malware detection from google play meta-data: Selection of important features. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 701–702, Sept 2015. doi: 10.1109/CNS.2015.7346893.

D.V. Ng and J.-I.G. Hwang. Android malware detection using the dendritic cell algorithm. In *Machine Learning and Cybernetics (ICMLC), 2014 International Conference on*, volume 1, pages 257–262, July 2014. doi: 10.1109/ICMLC.2014.7009126.

U. Pehlivan, N. Baltaci, C. Acarturk, and N. Baykal. The analysis of feature selection methods and classification algorithms in permission based android malware detection. In *Computational Intelligence in Cyber Security (CICS), 2014 IEEE Symposium on*, pages 1–8, Dec 2014. doi: 10.1109/CICYBS.2014.7013371.

N. Peiravian and Xingquan Zhu. Machine learning for android malware detection using permission and api calls. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 300–305, Nov 2013. doi: 10.1109/ICTAI.2013.53.

R. Raveendranath, V. Rajamani, A.J. Babu, and S.K. Datta. Android malware attacks and countermeasures: Current and future directions. In *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on*, pages 137–143, July 2014. doi: 10.1109/ICCICCT.2014.6992944.

RiskIQ. Android malware attacks and countermeasures: Current and future directions. June 2014.

J. Sahs and L. Khan. A machine learning approach to android malware detection. In *Intelligence and Security Informatics Conference (EISIC), 2012 European*, pages 141–147, Aug 2012. doi: 10.1109/EISIC.2012.34.

A.A.A. Samra, Kangbin Yim, and O.A. Ghanem. Analysis of clustering technique in android malware detection. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, pages 729–733, July 2013. doi: 10.1109/IMIS.2013.111.

B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P.G. Bringas. On the automatic categorisation of android applications. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 149–153, Jan 2012. doi: 10.1109/CCNC.2012.6181075.

A. Shabtai. Malware detection on mobile devices. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, pages 289–290, May 2010. doi: 10.1109/MDM.2010.28.

A. Shabtai, Y. Fledel, and Y. Elovici. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 329–333, Dec 2010. doi: 10.1109/CIS.2010.77.

L. Tenenboim-Chekina, O. Barad, A. Shabtai, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici. Detecting application update attack on mobile devices through network featur. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 91–92, April 2013. doi: 10.1109/INFCOMW.2013.6970755.

Te-En Wei, Ching-Hao Mao, A.B. Jeng, Hahn-Ming Lee, Horng-Tzer Wang, and Dong-Jie Wu. Android malware detection via a latent network behavior analysis. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1251–1258, June 2012. doi: 10.1109/TrustCom.2012.91.

Yu Wei, Hanlin Zhang, Linqiang Ge, and R. Hardy. On behavior-based detection of malware on android platform. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 814–819, Dec 2013. doi: 10.1109/GLOCOM.2013.6831173.

Westyarian, Y. Rosmansyah, and B. Dabarsyah. Malware detection on android smartphones using api class and machine learning. In *Electrical Engineering and Informatics (ICEEI), 2015 International Conference on*, pages 294–297, Aug 2015. doi: 10.1109/ICEEI.2015.7352513.

Zhao Xiaoyan, Fang Juan, and Wang Xiujuan. Android malware detection based on permissions. In *Information and Communications Technologies (ICT 2014), 2014 International Conference on*, pages 1–5, May 2014. doi: 10.1049/cp.2014.0605.

J. Xu, Y. Yu, Z. Chen, B. Cao, W. Dong, Y. Guo, and J. Cao. Mobsafe: cloud computing based forensic analysis for massive mobile applications using data mining. *Tsinghua Science and Technology*, 18(4):418–427, August 2013. doi: 10.1109/TST.2013.6574680.

S.Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik. A new android malware detection approach using bayesian classification. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 121–128, March 2013. doi: 10.1109/AINA.2013.88.

S.Y. Yerima, S. Sezer, and G. McWilliams. Analysis of bayesian classification-based approaches for android malware detection. *Information Security, IET*, 8(1):25–36, Jan 2014a. ISSN 1751-8709. doi: 10.1049/iet-ifs.2013.0095.

S.Y. Yerima, S. Sezer, and I. Muttik. Android malware detection using parallel machine learning classifiers. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pages 37–42, Sept 2014b. doi: 10.1109/NGMAST.2014.23.