TRENDS IN ANDROID MALWARE DETECTION

Kaveh Shaerpour
kavehshaerpour@aol.com
Faculty of Computer Science and Information Technology
University Putra Malaysia

Ali Dehghantanha*
alid@upm.edu.my
Faculty of Computer Science and Information Technology
University Putra Malaysia

Ramlan Mahmod
ramlan@upm.edu.my
Faculty of Computer Science and Information Technology
University Putra Malaysia

*Corresponding Author

ABSTRACT

This paper analyzes different Android malware detection techniques from several research papers, some of these techniques are novel while others bring a new perspective to the research work done in the past. The techniques are of various kinds ranging from detection using host based frameworks and static analysis of executable to feature extraction and behavioral patterns. Each paper is reviewed extensively and the core features of each technique are highlighted and contrasted with the others. The challenges faced during the development of such techniques are also discussed along with the future prospects for Android malware detection. The findings of the review have been well documented in this paper to aid those making an effort to research in the area of Android malware detection by understanding the current scenario and developments that have happened in the field thus far.

Keywords: Android, Malware Detection, Static Analysis, Malware Behavior

1. INTRODUCTION

Mobile devices are no longer used merely for facilitating wireless communication in the form of phone calls, or sending brief text messages using the Short Message Service (SMS), they have come a long way from that and now have become much more advanced devices commonly known as smart phones. These are devices that can connect to a wide range of networks including the internet via 3G or Wireless Fidelity (Wi-Fi), and have numerous applications that are developed to entertain and enhance the experience of the user. But because these devices have the ability to browse the internet and can exchange information with various devices and networks they become susceptible to attacks and threats by malware, and are the primary target these days (Daryabar et al., 2012). Previously one would find instances of malware only on personal computers, but since people rely on their smart phones ever so often for even sensitive interactions like online banking and these devices are becoming ubiquitous, the risk posed by malware on smart phones is very high. The extent of damage however is varied; a malware might render the victim's phone unusable, steal personal data like the victim's location or other confidential information, or access premium numbers from the victim's phone causing undesired billing (Daryabar et al., 2011; Mohtasebi et al., 2011).

There are several complexities that arise when it comes to antivirus solutions for smart phones, unlike personal computers, mobile devices do not have extensive computational resources, running an antivirus would consume a lot of RAM and CPU resources which would result in a sluggish operation of the phone along with accelerated depletion of the battery charge. For anti-virus solutions to be effective, a constantly updated virus signature database is needed so that all viruses can be detected with the matching of virus signature in the database. Nevertheless, the database itself should be added for new signatures regularly in order to detect previously unknown malware.

Smart Phones are available on several different platforms but Android platform has the most impressive growth and global spread. However, Android has become the breeding ground for new mobile malware. Although the platform itself is developing and releasing newer versions at quite a fast pace, the mobile malware is still spreading through the smart phone applications at a rapid rate.

Sahs and Khan (2012) described how the currently existent security features of the Android platform are largely insufficient. Besides, they mentioned that even applications that are not malicious, they bring coded in such a way that a lot of confidential information is collected. There has been a revolution in the services that are being offered via mobile applications. There are so many in numbers that there exist marketplaces for each mobile OS to centralize the distribution of these applications. They have their own mechanisms to prevent malicious applications from being distributed through the official channels. The problem however does not stop there since there are various other unofficial application marketplaces that are capable of similar distribution of genuine and malicious applications alike except these have no review process, and a developer may publish his Apps directly to these marketplaces.

In second paper, Zhou et al. (2012) described how most of the common malware are Trojan horses that usually camouflage themselves as useful applications like task managers, games, etc. Free applications, on Android markets are able to sustain themselves via advertising revenues. In fact, malware developers would tend to design and develop their Apps as a desirable application which would be downloaded often. As users are more inclined to download free versions of Apps instead of paying for them, malware developers can get their malicious Apps spread even wider. While these applications do provide the features and functionality of the legitimate app, they may have some hidden secret features that perform malicious activities without the user noticing it in the background. These kinds of applications are usually spyware, which track the user's personal data like GPS location and preferences and often sell them to advertisers so that they may generate targeted advertisements. Another benefit for the malware developer lies in his ability to make premium phone calls and send premium SMS silently from the victims phone, which causes unwanted billing for the victims.

Google tried to prevent these kinds of security breaches by making applications to declare permissions they would require on the phone, and to use only the required permissions for the functioning of the application. By right the victim approves access to the malware and lets the installation continue even though he sees that there are unnecessary permissions being requested in the manifesto. Asking users to only install applications does not seem an effective approach for deterring the users from downloading malicious applications. The users rarely know the extent to which a particular permission affects their personal data, and after installing several applications they get accustomed to ignoring all such warnings that come prior to installation. A similar ineffectiveness can be seen in the case of User Account Control that was implemented by Microsoft in Windows 7 to prompt the user to allow or deny any changes that was being made by an application on the computer, by fading out the screen and only focusing on a dialog box. This approach became annoying, and 69% of the users merely disabled the pop up account control feature in Windows 7 as stated by Zhou et al. (2012).

Schmidt et al. (2009) also did a study of third party Android market. It showed that there was a common practice of repackaging genuine applications acquired Google Play from and uploading them to third party marketplaces. Repackaging can have dangerous purpose which is to attach malware to it, plant backdoors or hide malicious payloads. Legitimate application developers are also affected by this repackaging since their reputation becomes impaired, as the word spreads that their app is malicious, even though it was not their intention and they become victims of intellectual property theft, and theft of their financial capital generated via the malicious apps.

This serious situation calls for an effective manner to detect malware on the Android platform; this paper will review few of the newer Android malware detection techniques that have surfaced. This paper comprises of five sections and will be presented as such: Section 2 describes the Android platform. Section 3 identifies the current developments in Android malware detection techniques. Section 4 discusses challenges encountered during Android malware detection research. Section 5 provides some statistical data. Section 6 discusses the future prospects for Android malware detection and Section 7 provides a conclusion for this paper. All the figures in this paper are taken from referenced papers to support the explanations.

2. ANDROID PLATFORM

Android platform is the first fully featured open source mobile operating system that was created to serve the consumer market. The open nature of Android and its attractive features led to its global acceptance, and rapid growth which has given a huge boost to the market of Android applications.

There are four layers in the Android stack, the uppermost layer is the Android Application layer, and the three layers below it are the application framework, Android runtime and the Linux Kernel. The Linux Kernel is what acts as an abstraction layer between the software stack of Android and the remaining hardware of the device. Since Android is an open source OS, it allows security researchers to focus on interactions at the kernel level and obtain useful information. Android applications are developed using Java programming language along with the tools and API's provided by the Android Software Development Kit (SDK).

The Android security model relies on its operating system, and is enforced by making application developers declare permissions in the "Android Manifest.xml" file in their application. The user is asked before installation whether he would like to grant the permissions requested by the applications, and has no control over micromanaging which permissions can be allowed and which cannot since according to the developer all those permissions would be required to ensure the proper functionality of the application. But often we see permissions which are unnecessary being requested by applications, the users must be wary of this, but sadly the only choice the user has is to install the applications, or not install it if the permissions seems to infringing to the users data and it is against his interest. The developer cannot remotely modify the permissions, unless he releases a new version which upon installation would prompt the user to once again review and accept the permissions being requested by the application.

3. CURRENT DEVELOPMENTS IN ANDROID MALWARE DETECTION TECHNIQUES

There have been various different approaches towards detecting Android Malware and there have been several publications documenting them but not particular technique has prevailed over the other, this section will review current developments and contributions in the field of Android malware detection.

Shabtai et al. (2012) proposed a behavioral framework to detect malware on the Android platform called “Andromaly”, as shown in Figure 1, the proposed framework is able to perform as a malware detection system by continuously monitoring the events and features of the mobile device and passing the data through anomaly detectors that use Machine Learning, the data that is collected can then either be classified as safe or malicious. The framework is implemented using small application which once installed samples various pieces of system data like CPU usage, the bandwidth of data being used, the intensity of packets sent via cellular or Wi-Fi networks, total number of processes running, battery consumption etc. and then analyses if the phone is functioning normally, or there are some anomalies in the collected data. The framework utilizes the idea that malware that have not yet been encountered can be detected by analyzing the similarities shown in the fluctuation of above mentioned system data with the introduction of already known malware. This would help users detect any suspicious activity on their devices.

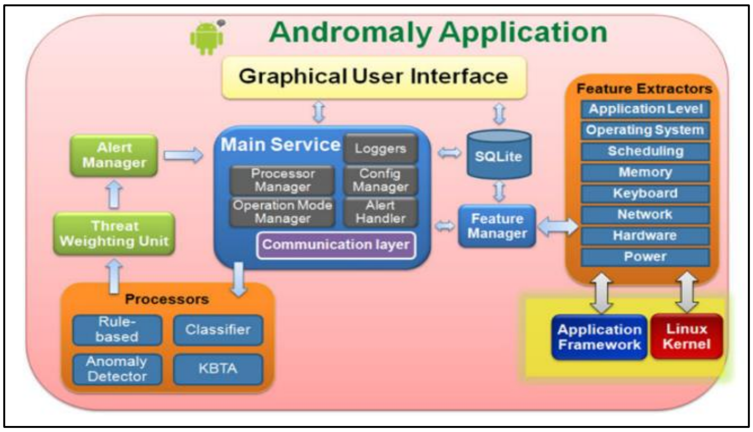

Figure 1 Andromaly Architecture (Shabtai et al., 2012)

The framework is modular and can utilize various malware detection techniques using rules and algorithms besides its behavioral approach. In order to improve the performance of the framework and make it less resource intensive they utilize a small number of features for the purpose of detection. The framework to be effective however requires a certain amount of training over a wide range of malware induced environments and safe environments to improve its rate of detection and reduce false positives. This however might put the user at a disadvantage should he not have ample malicious samples to train the framework to perform in a favorable manner.

Isohara et al. (2011) however discuss how one of the effective methods of detecting hidden threats in applications is by using some means to dynamically analyze them and the technique utilized to detect malware in their system is behavior based and their detection system consists of log analysis application on a server machine which would be used for analysis and at the Kernel layer they have a log collector, as shown in Figure 2, which would record all the system calls that were made since most applications have to go through the kernel layer, however since the logs generated would be huge and contain data from all applications, it would make the detection mechanism less efficient due to all the additional data, hence it implements a feature that allows it to filter the events with the perspective of a target application. The log analyzer in turn is able to pick up the process tree of the target application's activity and compares the activities with signatures created using regular expressions and is able to detect malicious activity if anything out of the ordinary takes place. Signatures of information leakage are also automatically generated using the data stored on the phone like Google account details and credentials, phone number, SIM info and IMEI numbers. And by using this technique they are able to identify and successfully detect the malicious behavior of applications and malware that have not been identified yet.

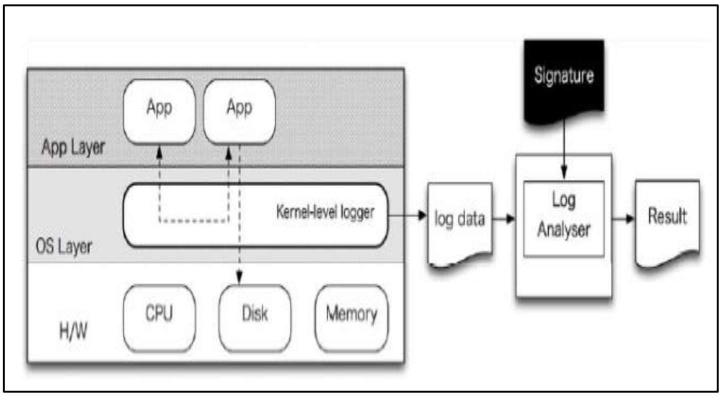

Figure 2 F System Architecture for Kernel-based Behavioral Analysis of Android Malware (Isohara et al., 2011)

This technique shows a lot of promises, and it can be greatly improved with a more sophisticated set of signatures to detect malicious activity which would help reduce error rates of false positive and false negative in the detection phase, also the efficiency can be improved by implementing techniques that would lessen the clutter in the logs being collected, and using more efficient means to reduce the log size.

Yang et al. (2012) focuses on detecting a particular kind of malicious Android application, the kind that attempts to steal money and cause unintended purchases from an unsuspecting user by pretending to be a legitimate application, but instead dialing premium numbers or sending premium SMS, from the victim's phone. They propose a system called "MoneyGuard", shown in Figure 3, follows a systematic approach towards detecting these malicious Android applications that steal money. Their technique also uses a behavioral approach to detect malware. They have identified two main behavioral patterns that seem to present in most of the money stealing malicious Android apps. The particular application behaviors they mention are the suppression of notifications, which allows the app to continue stealing money and charge the victim's by not letting the notifications surface and keeping the victim in the dark, and then the second behavior is hardcoded exfiltration, which makes the app send messages to hardcoded third party premium numbers or dialing to premium numbers in the background.

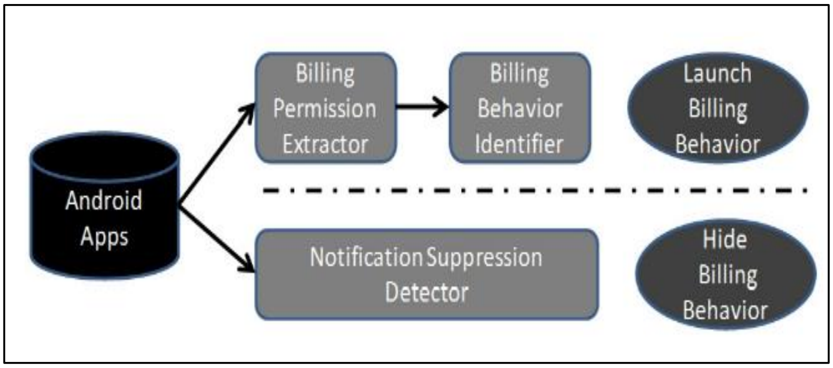
Android Apps
Billing Permission Extractor
Billing Behavior Identifier
Launch Billing Behavior
Notification Suppression Detector
Hide Billing Behavior

Figure 3 System Architecture for *MoneyGuard* (Yang et al., 2012)

Yang et al. (2012) present a light weight detection approach that will help in identifying this behavioral pattern. The system contains three components, the Billing Permission Extractor, Billing Behavior Identifier and Notification Suppression Detector. The first component extracts the permissions from the *Android Manifest.xml* file of the applications and analyses it for any permissions that allows the application to perform actions that are billable, like sending and receiving SMS, making phone calls and access the Internet. The second component decompiles the APK file of the application, and examines the Dalvik byte code for each activity performed by the application along with the system calls that are made, additionally it also searches for any hard-coded phone numbers that the application might try to communicate with. Lastly the third component, is used to detect if the application is utilizing any form of notification suppression, and if it already had certain suspicious billing behavior detected in the first two components, it is considered a money stealing application if in the third component it is found to have notification suppression features in it. This technique seems rather useful and relevant to mitigate the practical threat of money stealing apps that many Android users are a victim to, but are not yet aware of it. This platform can perhaps be extended to include other kinds of Android malware in a modular fashion to improve the usability of this system, in day to day detection of Android Malware in this ever growing market.

Schmidt et al. (2009) claims that commercially available techniques to counter malware on smartphones are largely inadequate, and that since these techniques rely on malware signatures alone, the users relying on these malware detection solutions are left exposed to new malware until the new signature is developed and the system is updated with the latest malware signatures. The extent of damage that can be done till then and the amount of other phones the malware can spread to during that time is tremendous. Their proposed system contains three main components, analysis on the device itself, collaboration module and a client-server feature for remote analysis and this system is shown in Figure 4. To communicate between the kernel layer and the software stack they wrote a tool called “Interconnect Daemon” which is a Linux server daemon that contains modules that perform various functions such as monitoring the system, scanning the files and creating hashes for the important ones, waiting for specific system signals that trigger certain events, one specific module is used to extract all Executable and Linking Format (ELF) object files excluding the shared library, and inspected them for a number of system commands, they performed similar analysis for malicious samples that were acquired from the internet and identified the static list of function calls made by them. This finding from the static analysis of system commands and malicious applications forms the training set for the system to help determine safe and malicious applications. The server component is utilized here to analyze the interdependencies between attributes of executable to help in classifying them better, rules are synthesized after this stage and pushed to the mobile devices, this helps in reduce false positives. The actual performance of the malware intrusion detection of the system on the mobile device would be possible after acquiring the rules. When a mobile device notices an anomaly if properly implemented, it should be able to utilize the other mobile devices around it running the system through collaboration and obtain computational resources for faster detection, if the computation cannot be performed on the device itself then it is sent to a remote server, and the results are obtained. Experimentation using real devices and malware is yet to be done using this technique, but it still seems like a viable and interesting solution to malware detection.

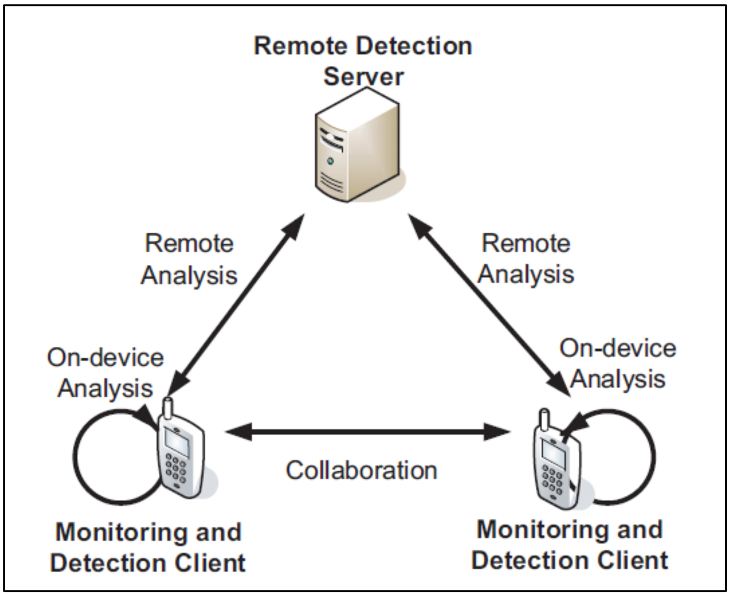

Figure 4 Collaborative Malware Detection Overall System Architecture (Schmidt et al., 2009)

The proposed method by Apvrille and Strazzere (2012) for detecting Android malware is by implementing a market wide scanner as shown in Figure 5. Their system is devised in such a way that it crawls the entire Google Play store using a combination of parameters based on various classifiers such as country, genre, language etc., this way their method is able to acquire all existing applications on the market and not only those which are displayed as available to the particular device that is accessing the market.

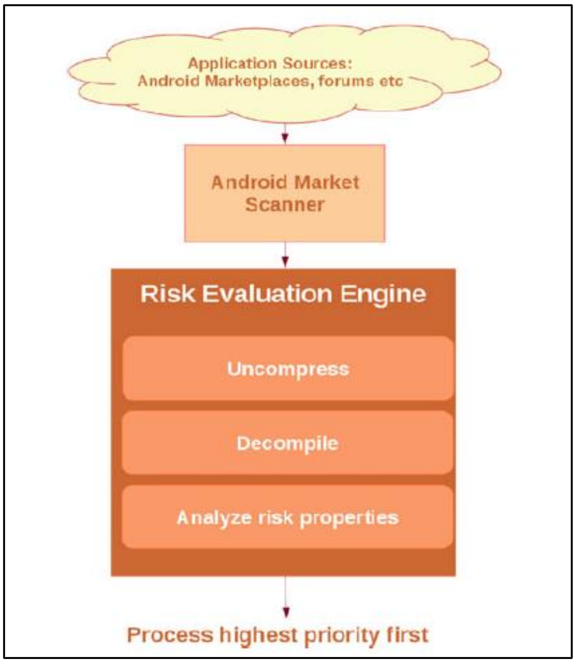

Figure 5 Risk Evaluation Heuristics Engine (Apvrille et al., 2012)

Once the application samples are collected from the market, a heuristics engine is run in order to pre-process the samples in order to gauge which samples possess malicious characteristics and prioritize the analysis on them. The heuristics engine used in this method is a kind of static analysis tool that would check for 39 different preset properties, which are broadly categorized as the permissions requested by the app, peculiar calls to Java classes or methods, code size, geodata indicators, executable files, URL's present in code etc. This technique was used to minimize the time consumed in doing a large scale scan and also appropriate filtering mechanisms were implemented where required to ignore legitimate applications, which host a guest application to facilitate in app advertising. Each attribute was assigned a particular risk score, and the risk would incremented for the application as the scan progresses and give a final risk score that would help deduce it's malicious or benign nature if the score is beyond a particular limit. The main objective of this technique is to sift out and prioritize which samples should be subjected to further analysis give the number of samples being scanned. An added benefit to this approach is that it serves as a repository of metadata which can be used to compare, any rogue apps in the market that are pretending to be popular legitimate applications but have their code altered to perform malicious functions.

Zhou et al. (2012) present another systematic approach to detect malicious Android applications on both official and unofficial Android marketplaces, shown in Figure 6. Unlike Apvrille and Strazzere (2012) that crawled only the official market, they attempt to detect both known malware variants as well the ones that are yet to be detected. They developed a prototype, tool called "DroidRanger" in order to implement this detection approach. After the market crawler collects the apps from various markets it stores them in repository. "DroidRanger" then extracts the basic properties of the app like information about the author and the permissions requested, and then it stores this information besides the actual application in a database for the purpose of efficiency. Two separate malware detection engines are utilized, to use the first type of detection engine, a behavioral and permission oriented footprint is generated of the known malware samples, this helps in detecting malicious applications that have footprints similar to the known malware by seeing which permissions are being abused for malicious purposes. The second detection engine utilizes a heuristics based filtering mechanism that helps detect zero day malware samples. They use two kinds of heuristic approaches the first analyzes if any binary code is being loaded dynamically from a remote sure, and the second analyzes if there is any native code being loaded dynamically, this is usually the case when it comes to kernel level exploits, and this method aims at finding malware that try to "root" the device.

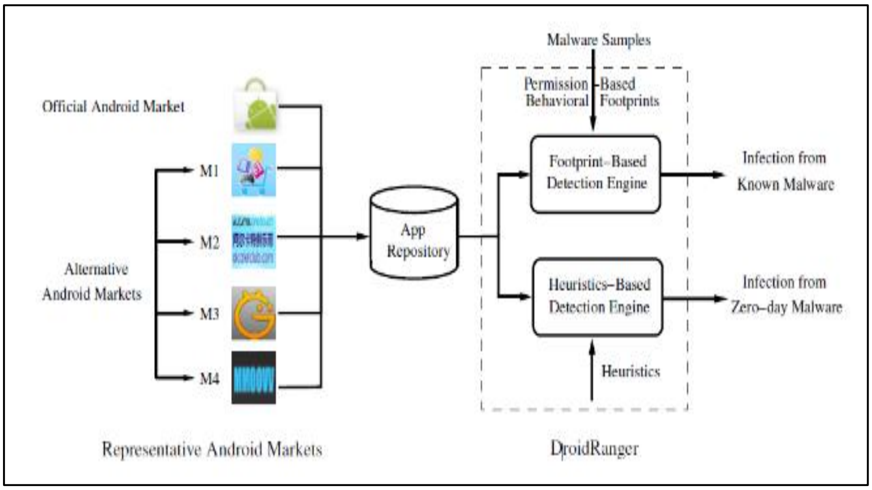
Malware Samples
Permission-Based Behavioral Footprints
Official Android Market
M1
M2
M3
M4
Alternative Android Markets
App Repository
Footprint-Based Detection Engine
Heuristics-Based Detection Engine
Infection from Known Malware
Infection from Zero-day Malware
Heuristics
Representative Android Markets
DroidRanger

Figure 6 DroidRanger System Architecture (Zhou et al., 2012)

Grace et al. (2012) also take a proactive stance towards detecting zero-day Android malware by analyzing a large number of applications from both official and unofficial Android marketplaces. Their technique does not rely on malware samples and signatures, and categorizes the potential risk an application can pose as high, medium and low risks as shown in Figure 7. Applications classified as high risk are the ones that exploit platform vulnerabilities and get unauthorized access to the devices, those classified as medium do not perform any exploits but are capable of causing financial damages to the user or steal personal information, those classified as low also steal information but cause less impact, as it is only device specific information like IMEI number etc. They developed an automated system called *RiskRanker* to assess and assign this risk classifications. The analysis happens in two fold, for the first phase to detect high risk apps that contain attack code, the code of the well-known exploits is distilled and signatures are created to compare the common characteristics between the exploits and the application being analyzed. To detect medium risk apps, certain permissions that can potentially harm the user if abused like "android.permission-group.COST_MONEY" along with certain system calls which indicate the functioning of the application silently without notifying the user are searched for by performing static analysis. The first phase would typically work with non-obfuscated applications. The second phase was designed malware that might be designed to be hidden from the analysis performed in the first phase or might be encrypted. The first part of this phase includes capturing certain behavior which can be commonly abused but is usually legitimate, like a child application inside a host application, utilization of Java encryption APIs to encrypt communication, dynamic code being loaded in the background, native code execution and hard coded access to internal directories of the device.

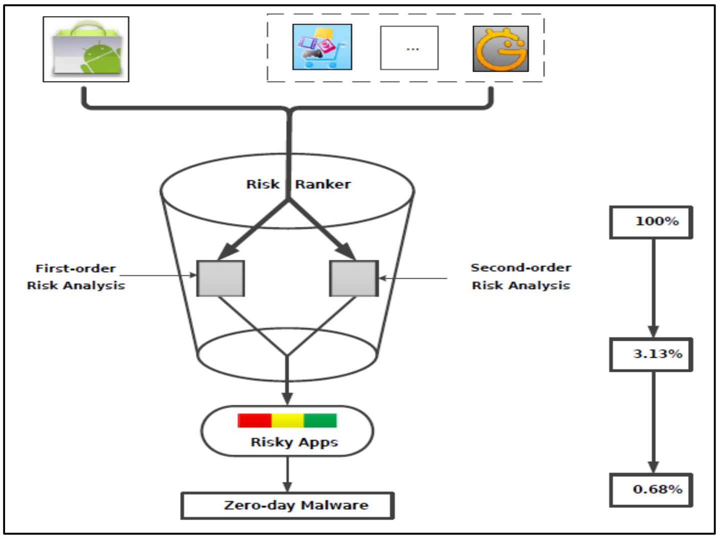

Figure 7 RiskRanker System Design (Grace et al., 2012)

Burguera et al. (2011) chose to use dynamic analysis of an application's behavior as the basis of detecting malicious apps in their framework. The framework itself is comprised of many components which are able to complement one another and provide the mechanism and the resources to detect Android Malware. The first component is a client application called *Crowdroid*, which is shown in Figure 8, is an application that can be downloaded by Android users from the Google Play store. This application allows the system calls made at the kernel level to be monitored. Users can aid in malware detection by providing certain data that is related to the behavior of the applications they use regardless of where they acquired their applications from, this method of crowdsourcing data does not collect any personal information from the users. The output logs of the application behavior which are created using a tool called *Strace* which enables the collection of system calls are then forwarded to the remote server which is in charge of parsing the data and creating vectors that correspond to various system calls which also represent the number of times each system call was used, this helps in generating a dataset of benign application behavior. For the success of this technique it requires multiple users to use the *Crowdroid* application which will increase the quality of the dataset and help detect anomalies in the application behavior to alert all users of a malicious application. The detection component works by using K-means clustering algorithm over the system call vector dataset to detect any anomalous behavior.

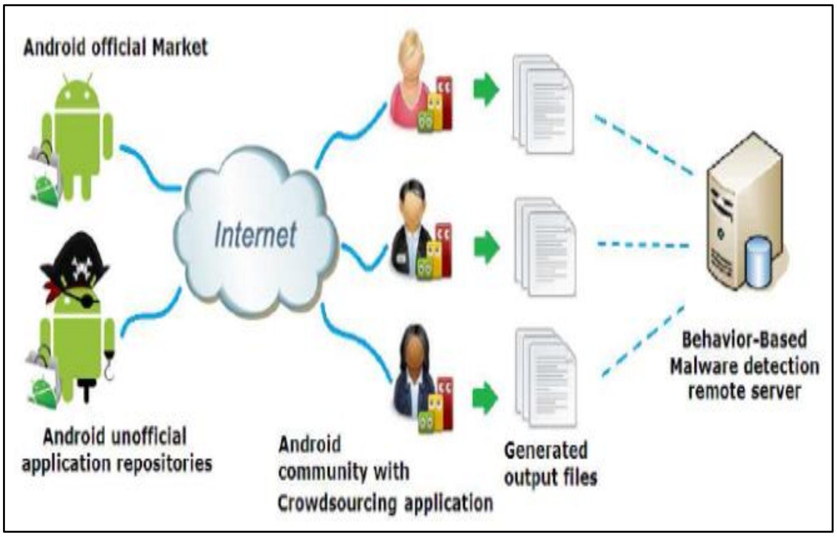
Android official Market
Internet
Android unofficial application repositories
Android community with Crowdsourcing application
Generated output files
Behavior-Based Malware detection remote server

Figure 8 Crowdroid System Architecture (Burguera et al., 2011)

Wu et al. (2012) proposes a similar technique of a feature based static analysis to detect malicious Android applications; it utilizes the static information like API calls, components being deployed and permissions to obtain characteristic features of the Android applications. They developed a proof of concept framework called *DroidMat*, which is shown in Figure 9 that extracts this information from the application's manifest file, the disassembled code is then further analyzed using "apktool", this approach also makes use of vectors for each application that indicates whether or not a malicious element is present in the application. The application's functional behavior is further classified using K-means and EM algorithm using the Singular Value Decomposition method; this analyzes the behavior of the application and its functionalities. Finally, K-

Nearest Neighbor (K-NN) algorithm is utilized to identify the application as benign or malicious.

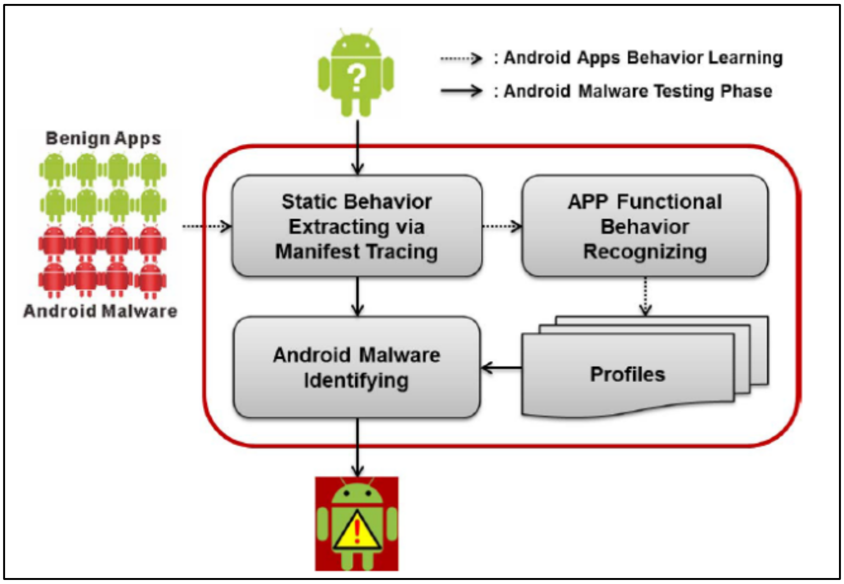
: Android Apps Behavior Learning
: Android Malware Testing Phase
Benign Apps
Android Malware
Static Behavior Extracting via Manifest Tracing
APP Functional Behavior Recognizing
Android Malware Identifying
Profiles

Figure 9 DroidMat System Architecture (Wu et al., 2012)

4. CHALLENGES ENCOUNTERED DURING ANDROID MALWARE DETECTION RESEARCH

One of the most common challenges when it comes to malware detection techniques is the occurrence of false negatives and false positives in the report. Which causes malicious applications to be classified as safe and legitimate applications that seem to need access to particular features in order to perform its function seems similar to a malware and is classified as one.

Sahs and Khan (2012) identified various challenges that they faced in utilizing their technique of anomaly detection, like the fact that the activities that malware perform are often very short and do not provide enough data for their framework to learn from the behavior or even detect it. Also that there was a shortage of a training set of malicious applications that can be used to improve the detection framework, and lastly the behavior of malware can be variable between each attack and be polymorphic in nature, which causes additional challenges to the researcher.

Isohara et al. (2011) used a dynamic analysis technique and relies heavily on the log data and the signature set the system contains, to be comprehensive it stores all the communication of applications with the kernel layer via system calls but the drawback with using this technique, is the constant monitoring of processes generates a large amount of log entries, and one would have to sift through a lot of log data to get to the relevant information , unless an efficient means of filtering the logs is used and also an efficient means of collecting the logs so that the log size is reduced.

The quality of the results would also depend on how well the signature set is developed to reduce errors in detection.

Grace et al. (2012) also faces similar challenges whereby their technique relies on known exploit signatures, if the exploit is encrypted or obfuscated the analysis technique used might give a false negative, the techniques utilized to curb the flow of malwares into the markets as well the techniques used to identify them might be studied by malware authors and countered, making the next generation of malware even harder to detect.

Techniques like by Apvrille and Strazzere (2012) where markets are crawled to download application to add to the repository might be banned from the markets for excessive downloading.

Zhou et al. (2012) states that the current model of the Google Play store where users need to rate and alert if a particular application is malicious is not effective and does not work, the analysis they performed was done only on free applications and they are of the opinion that paid apps might have certain obvious differences which might not let their technique work the same way. Only five marketplaces were utilized for the experiment, and only two behavioral characteristics were utilized for the identification of zero day malware using the heuristic search, a much more effective analysis can be performed by adding more parameters to the heuristic search like checking for a behavior where an SMS is being sent to a premium number.

For the adoption of the crowd sourced technique proposed by Burguera et al. (2011), they face a challenge in convincing Android users to download and install the *Crowdroid* application, and moreover to not feel like it is a loss of privacy when they support the researchers with behavioral information of their application since this only benefits them by providing updated statistics of malwares being detected.

5. STATISTICALLY RELEVANT DATA

According to Apvrille and Strazzere (2012), the time taken between the detection of new Android malware by an anti-virus company and its actual release in the market is approximately 80 days as is seen in Figure 10, the dates of detection in the graph above are certain since they are determined by the malware detection announcements made by the vendors. However the release date of the malware is determined using the date on the signed certificate which may contain erroneous data. The reason behind is that developers might reuse certificates. Regardless, this only goes to show that the need for a better and faster detection mechanism still exists and is extremely relevant to the status quo.

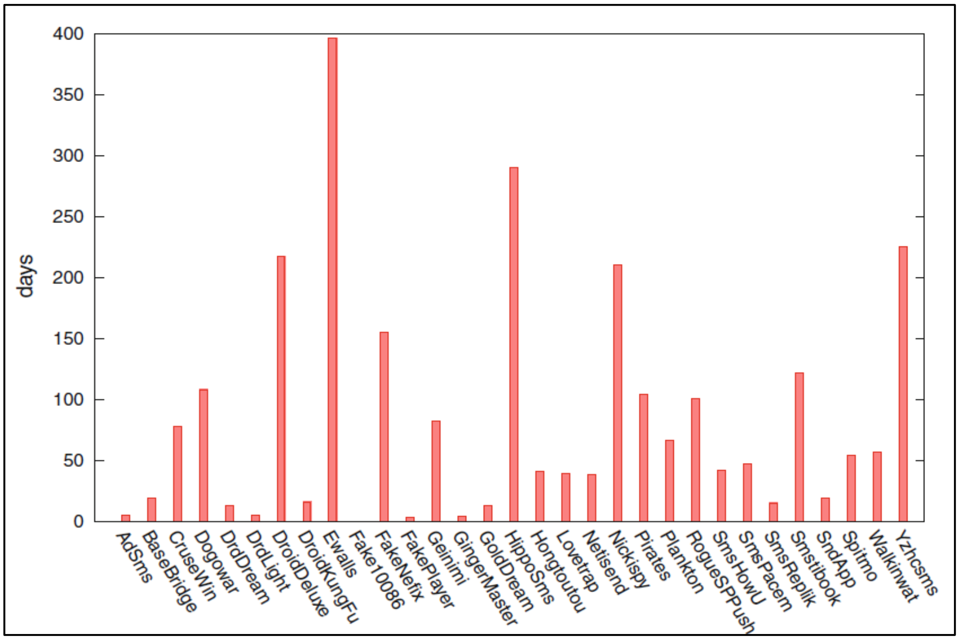

Figure 10 Number of Days Taken Between Release and Detection of Android Malware by Anti-virus Vendors

From the techniques reviewed the trends in their approaches have been quantified in the Figure 11 below

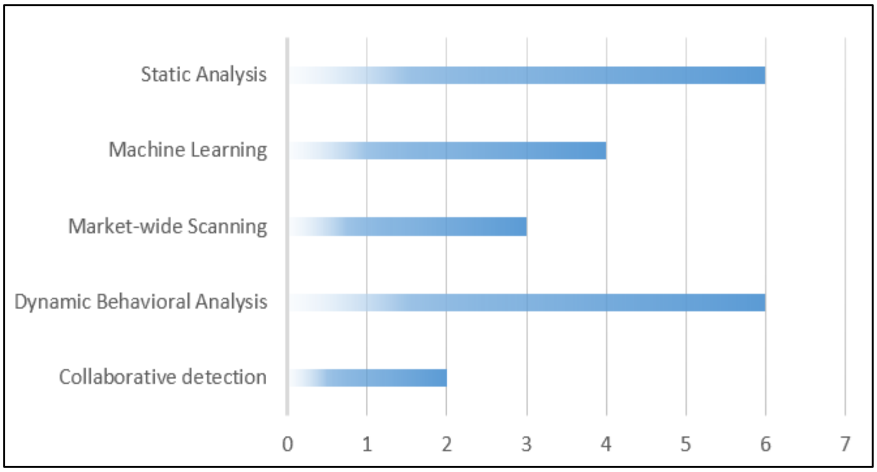

Figure 11 Trends Used in Detection Techniques

There has been, equal emphasis on detecting android malware using static and dynamic analysis techniques, some papers documented using more than one technique in a multi-phase framework which seemed to yield good results, quite a few papers utilize Machine Learning Algorithms in order to amplify the effectiveness of their heuristics engines. A lot of work can still be done in the area of collaborative detection as this technique has not been utilized at large mostly due to its drawback of needing user participation. Lastly, the idea of establishing a repository of known applications via market scanning with the support of the market vendors could provide useful in detecting rogue apps with modified code holding payloads and prove less cumbersome on the researchers.

6. FUTURE PROSPECTS FOR ANDROID MALWARE DETECTION

The security model of the Android platform might change steeply, and the developers contributing to the Android Operating System probably will start taking security seriously making it less easy for malware to reside on their app stores, or the mobile devices running the Android OS. Until then however malware developers would start developing malware and packing them in such a way that they evade the existing techniques of detecting malware. Be it static, dynamic or intrusion detection based, Android malware techniques seem to be greatly improving upon the work of the predecessors, with more malware samples available in the future, it would be easier to conduct research and build more efficient detection techniques that would keep Android users safe. With well-established techniques, the next step would be to run comparisons between them side by side, to see which technique would be best suited for today’s generation of Android users, and if a single technique is insufficient, then perhaps a combination of more than one techniques would be the way to go. In the future, when phones with higher specifications become more readily available and more widely used the capability of the malware detection techniques can also be greatly increased by leveraging the higher computational capacities and memory modules of the future devices. It would be in the best interest for Android security if the future frameworks developed for malware detection, are open and modular, this way the community can contribute as a whole to help make it better, and the ability to crowd source statistical data would be much beneficial to the research towards developing better malware detection techniques.

7. CONCLUSIONS

In this review paper, we analyzed various Android Malware detection techniques that have been proposed, and observed the trends in the research geared towards finding better detection techniques. We were able analyze and understand the various challenges that were faced in building efficient techniques and implementing them. Just like the computer counterpart there is not yet an All-in-one solution, that can be a remedy for all malware and detect the ones yet to be discovered, but considerable progress has been made. A common obstacle when most of the research in this field was conducted was the lack of malware samples to experiment with. But today that is not the case there are sufficient amount of samples that are available and have been studied and well documented and this study of Android malware can greatly aid in the development of better Android Malware detection techniques, by knowing how the known malware behave and affect Android devices.

ACKNOWLEDGMENTS

This work has been supported by the MOSTI-Science Fund project 01-01-04-SF1677. The views and opinions expressed in this article are those of authors alone and not the organizations with whom authors are or have been associated/supported.

REFERENCES

Apvrille, A., & Strazzere, T. (2012). Reducing the window of opportunity for Android malware gotta catch ’em all. *Journal in Computer Virology*, *8*(1-2): 61-71.

Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). Crowdroid: Behavior-based malware detection system for Android. *2011 ACM CCS Workshops on Security and Privacy in Smartphones and Mobile Devices (SPSM’11)*, 17-21 October 2011, Chicago, Illinois, USA.

Daryabar, F., Dehghantanha, A., & Broujerdi, H. G. (2012). Investigation of malware defense and detection techniques. *International Journal of Digital Information and Wireless Communications (IJDIWC)*, *1*(3): 645-650.

Daryabar, F., Dehghantanha, A., & Udzir, N. (2011). Investigation of bypassing malware defenses and malware detections. *7th International Conference on Information Assurance and Security (IAS)*, 5-8 December 2011, Malacca, Malaysia.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012). RiskRanker: scalable and accurate zero-day Android malware detection. *The 10th International Conference on Mobile Systems, Applications, and Services (MobiSys’12)*, Low Wood Bay, Lake District, United Kingdom.

Isohara, T., Takemori, K., & Kubota, A. (2011). Kernel-based behavior analysis for Android malware detection. *2011 Seventh International Conference on Computational Intelligence and Security*, 3-4 December 2011, Sanya, Hainan Province, China.

Mohtasebi, S. H., & Dehghantanha, A. (2011). A mitigation approach to the privacy and malware threats of social network services. *Digital Information Processing and Communications*, Springer Berlin Heidelberg.

Sahs, J., & Khan, L. (2012). A machine learning approach to Android malware detection. *2012 European Intelligence and Security Informatics Conference*, 22-24 August 2012, Odense, Denmark.

Schmidt, A., Bye, R., Schmidt, H., Clausen, J., Kiraz, O., Yuksel, K., … Albayrak, S. (2009). Static analysis of executables for collaborative malware detection on Android. *IEEE International Conference on Communications Workshops (IEEE ICC 2009)*, 14-18 June 2009, Dresden, Germany.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, G., & Weiss, Y. (2012), "Andromaly": A behavioral malware detection framework for Android devices. *Journal of Intelligent Information Systems*, *38*(1): 161-190.

Wu, D., Mao, C., Wei, T., Lee, H., & Wu, K. (2012). DroidMat: Android malware detection through manifest and API calls tracing. *2012 Seventh Asia Joint Conference on Information Security*, 9-10 August 2012, Tokyo, Japan.

Yang, C., Yegneswaran, V., Porras, P., & Gu, G. (2012). POSTER: Detecting money-stealing apps in alternative Android markets. *CCS '12 Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 16-18 October 2012, Raleigh, North Carolina, USA.

Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012). Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 5-8 February 2012, San Diego, California, USA.

ABOUT THE AUTHORS

Kaveh Shaerpour is a master by research student at the faculty of Computer Science and Information Technology, University Putra Malaysia. He received his BSc in Computing from Staffordshire University. His research interest includes digital forensics, computer security and cyber physical systems forensics, and malware analysis. He can be contacted at kavehshaerpour@aol.com.

Ali Dehghantanha obtained his PhD in Computer Science with a specialization in Security in Computing from University Putra Malaysia (UPM) and serves as a senior lecturer at the same school. His research interests include digital forensics, penetration testing, and game-theory applications in security, malware analysis, and cyber physical systems forensics. He can be contacted at AliD@upm.edu.my.

Ramlan Mahmod is a professor at the faculty of Computer Science and Information Technology, University Putra Malaysia. He received his BSc in Computer Science from Western Michigan University, USA, MSc from Central Michigan University, USA, and his PhD in Artificial Intelligence from Bradford University, UK. His research interest includes artificial intelligence, cryptography, trusted computing, and computer vision. He can be contacted at ramlan@fsktm.upm.edu.my.