# Android Malware Detection & Protection: A Survey

Saba Arshad
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

Abid Khan
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

Munam Ali Shah
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

Mansoor Ahmed
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

***Abstract*—Android has become the most popular smartphone operating system. This rapidly increasing adoption of Android has resulted in significant increase in the number of malwares when compared with previous years. There exist lots of antimalware programs which are designed to effectively protect the users' sensitive data in mobile systems from such attacks. In this paper, our contribution is twofold. Firstly, we have analyzed the Android malwares and their penetration techniques used for attacking the systems and antivirus programs that act against malwares to protect Android systems. We categorize many of the most recent antimalware techniques on the basis of their detection methods. We aim to provide an easy and concise view of the malware detection and protection mechanisms and deduce their benefits and limitations. Secondly, we have forecast Android market trends for the year up to 2018 and provide a unique hybrid security solution and take into account both the static and dynamic analysis an android application.**

***Keywords*—Android; Permissions; Signature**

## I. INTRODUCTION

Since 2008, the rate of smartphone adoption has increased tremendously. Smartphones provide different connectivity options such as Wi-Fi, GSM, GPS, CDMA and Bluetooth etc. which make them a ubiquitous device. Google says, 1.3 million Android devices are being activated each day [1]. Android operating system left its competitors far behind by capturing more than 78% of total market share in 2013 [2]. Gartner report 2013 of smartphone sales shows that there is 42.3% increase in sales of smartphones in comparison with 2012. According to International data corporation IDC, Android OS dominates with 82.8% of total market shares in 2Q 2015 [3]. Figure 1 shows the market shares of Android operating system on yearly basis. It could be observed that Android has become the most widely used operating system over the years.

Android platform offers sophisticated functionalities at very low cost and has become the most popular operating system for handheld devices. Apart from the Android popularity, it has become the main target for attackers and malware developers. The official Android market hosts millions of applications that are being downloaded by the users in a large number everyday [4]. Android offers an open market model where no any application is verified by any security expert and this makes Android an easy target for developers to embed malicious content into their applications. The users' sensitive data can be easily compromised and can be transferred to other servers. Furthermore, the existence of third party application stores contribute in spreading malwares for Android because Google Play also hosts the applications of third-party developers. Android official market uses Bouncer for protection of marketplace against malwares [5]. However, Bouncer does not analyze the vulnerabilities of the uploaded apps. Malware developers take advantage of vulnerabilities among apps by repackaging the popular apps of Google Play and distributing them on other third-party app-stores. This degrades the reputation of the app-store and of the reputation of the developer. Malwares includes computer viruses, Trojan horses, adware, backdoors, spywares and other malicious programs which are designed to disrupt or damage the operating system and to steal personal, financial, or business information. Malware developers use code obfuscation methods, dynamic execution, stealth techniques, encryption and repackaging to bypass the existing antimalware techniques provided by Android platform.

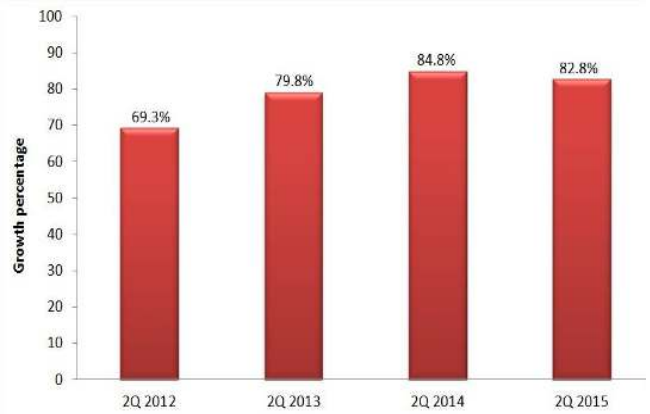



Fig. 1. Android Market Shares

In order to prevent such malwares, it is important to have accurate and deep understanding of them so that security measures to protect users' data could be taken accordingly. There are large numbers of attack scenarios where an attacker can compromise a user's data by taking advantage of the vulnerabilities of Android operating system. For example, a

Trojan app downloads some HD wallpapers with user's permission but this permission may allow this app to access the user's contacts or other personal information and it leaks user's confidential data to some other server from the device secretly. In such a case, the wallpapers app will have Internet permissions for download purpose. The user might not give much attention towards other requested access permissions and might grant READ_CONTATCS permission accidentally. As a result, the app may modify the device settings, corrupt the user's data and can transfer private data to some unknown remote servers. This results in user's business data loss and other personal information. The attackers can use the stolen data for kidnapping, blackmailing or business loss purposes. In an another attack scenario, attackers distribute the malicious apps as a repackaged version of some popular apps which may offer location-based services so in that scenario malicious app kill the victim device by draining its battery with the excessive use of GPS and radio etc. Some of the malicious programs get the user's device IMEI numbers and send it to remote server. These IMEI numbers have significant worth in black markets where IMEI numbers of stolen devices can be altered with user's IMEI [6].

There are hundreds of malware techniques identified which attack the Android platforms in several ways such as sending messages without the victim's knowledge and deleting them by itself, sending user's private information to some other server and many more. So there is a great need to protect user's data from these malwares.

This ever increasing malware threats have forced the Android antimalware industry to develop the solutions for mitigating malicious app threat on Android smartphones and other Android devices. Two main approaches are used for this purpose: *Static approach* and *Dynamic approach*. Antivirus programs use any of these approaches to protect the mobile systems from the malware attacks. They detect the malicious apps and notify the user about such apps and take measures to remove these malwares**.** With the increasing number of threat level, the antivirus detection rate has also increased. As a result of threat & malware, and protection mechanism offered by Android antimalware programs, the overall risk situation of Android users is difficult to assess [7].

In this paper, we have analyzed different malwares, their behaviors and techniques used by different malware types to attack Android devices. Furthermore, the paper provides detailed review on different antimalware techniques, their advantages and limitations. On the basis of this review, a hybrid solution for Android security has been proposed. The rest of the paper is organized as follow. Section II classifies the existing malwares on the basis of their behavior. Section III consists of malware penetration techniques employed by the attackers. In Section IV, a detailed analysis on the malware detection and removal methods for the protection of Android devices has been performed. Section V consists of performance evaluation of antimalware mechanisms. The future trends for Android market shares and malware growth and limitations for existing antimalware approaches are provided in Section VI. A solution has also been proposed in this section which is aimed at providing better security mechanism. The paper is concluded in Section VII.

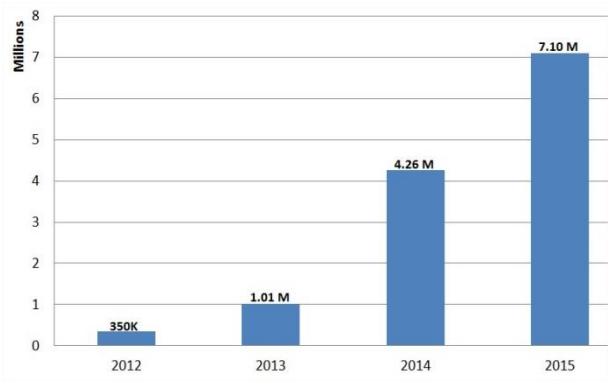



Fig. 2. Android Malware Growth

## II. Android Malware Analysis

Wide range of malwares has been detected and the number of malwares are increasing every year. According to TrendMicro, malwares have increased to 7.10 million in first half (1H) of 2015 [8][9]. Figure 2 shows the increased number of Android malwares over the years. The behavior of different malware families is provided in subsequent sections.

### A. Trojans

Trojans appear to a user as a Benign app [5]. In fact, they actually steal the user's confidential information without the user's knowledge. Such apps can easily get access to the browsing history, messages, contacts and device IMEI numbers etc. of victim's device and steal this information without the consent of user. *FakeNetflix* [10] is an example of such malwares that provide user interface identical to original Netflix app and collect the user's login credentials. SMS Trojans exploit the premium services to incur financial loss to the victim. *Fakeplayer* is a well-known SMS Trojan that sends messages to premium rate numbers without user awareness [11]. *Zsone* [12] and *Android.foney* are also the examples of such SMS Trojan apps. Malwares also capture the user's banking information such as account number and password. *Zitmo* and *Spitmo* Trojans are designed to steal the user's mTANs (Mobile Transaction Authentication Number) which then complete the transactions silently [13].

### B. Backdoors

Backdoors employ the root exploits to grant root privileges to the malwares and facilitate them to hide from antiviruses. *Exploid, Rageagainstthecage (RATC)* and *Zimperlich* are the top three root exploits which gain full- control of device [14]. *DroidKungFu* [15] uses root exploits, *Exploid and Rageagainstthecage,* in an encrypted form. When *DroidKungFu* executes, it first decrypts and launches the root exploits. If the root exploit succeed to gain control over device and root privilege, the malware become able to perform any operation on the device even the installation of applications keeping the user unaware of this act [16].

### C. Worms

Such malwares create copies of it and distribute them over the network. For example, Bluetooth worms spread malware through the Bluetooth network by sending copies of it to the

paired devices. *Android.Obad.OS* is the example of Bluetooth worm [17].

### D. Spyware

*Nickspy* [11] and *GPSSpy* [18] are the examples of spyware apps which appear as benign app, but it actually monitors the user's confidential information such as messages, contacts, bank mTANs, location etc. for some undesirable consequences. Personal spywares can install the malicious payload without the victim's knowledge. It sends the user's information such as text messages, contacts etc. to the attacker who installed that software on victim's device [6].

### E. Botnets

Botnet is a network of compromised Android devices. Botmaster, a remote server, controls the botnet through the C&C network. Geinimi [11] is one of the Android botnets.

### F. Ransomwares

Ransomware prevent the user from accessing their data on device by locking the device, until ransom amount is paid. FakeDefender.B [19] is a malware that masquerades itself as avast!, an antivirus. It locks the victim's device and force the user to pay ransom amount to unlock the device.

### G. Riskwares

Riskwares are the legitimate software exploited by the malicious authors to reduce the performance of device or harm the data e.g., delete, copy or modify etc. [20]. Table 1 below shows the top malware types detected in 2015 by TrendMicro [21].

TABLE I. TOP ANDROID MALWARE TYPES IN 2015

| Malware types | Threat percentage |
|---|---|
| PUAs | 50% |
| Adware | 27% |
| Trojans | 22% |
| Riskwares | 11% |
| SMSsenders | 7% |
| Downloaders | 3% |

The statistical data obtained from [21] has been computed and plotted in Figure 3 which presents the top Android malware families recorded by TrendMicro in second quarter (2Q) of 2015. According to the report, 24% of the total malwares were guided variants, which do not have any GUIs and silently run at the background without the user's knowledge.

## III. MALWARE PENETRATION TECHNIQUES

### A. Repackaging

Malware authors repackage the popular applications of Android official market, Google Play, and distribute them on other less monitored third party app-store. Repackaging includes the disassembling of the popular benign apps, both free and paid; append the malicious content and reassembling of app .This process of repackaging is done by reverse-engineering tools. During repackaging, malicious authors change the signature of repackaged app and so the app seems new to the antimalware. TrendMicro report have shown that 77% of the top 50 free apps available in Google Play are repackaged [22].

### B. Drive By Download

It refers to an unintentional download of malware in the background. Drive by download attacks occur when a user visit a website that contains malicious content and injects malware into the victim's device without the user's knowledge. Malware developers use *Android/NotCompatible* [23] which is one of the drive-by download app.

### C. Dynamic Payloads

Malwares also penetrate into Android devices through dynamic payload technique. They encrypt the malicious content and embed it within APK resources. After installation, the app decrypts the encrypted malicious payload and executes the malicious code. Some malwares, instead of embedding payload as resource, download the malicious content from remote servers dynamically and are not detected by static analysis approach [24].

### D. Stealth Malware Techniques

On Android device malware scanners cannot perform deep analysis because of the availability of limited resources such as battery. Malware developers exploit these hardware vulnerabilities and obfuscate the malicious code to easily bypass the antimalware. Different stealth techniques such as key permutation, dynamic loading, native code execution, code encryption and java reflection are used to attack the victim's device.

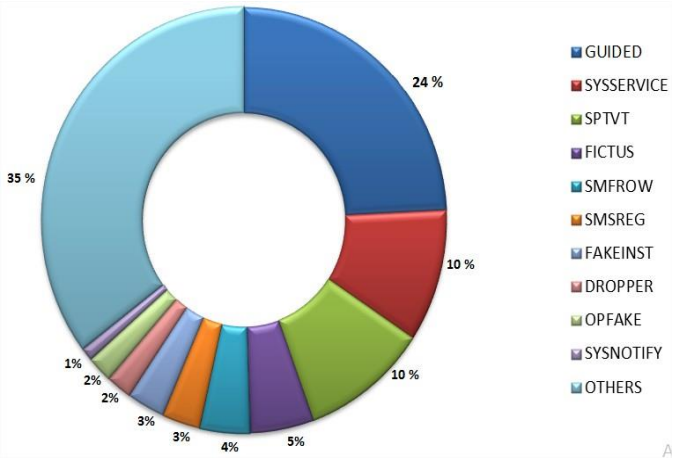



Fig. 3. Malware families seen in 2015

## IV. ANDROID MALWARE DETECTION

There are mainly two approaches to analyze the Android malwares: Static and Dynamic Approach. We have further categorized the antimalware using static and dynamic approaches. Figure 4 shows the taxonomy of existing antimalware techniques based on our study.

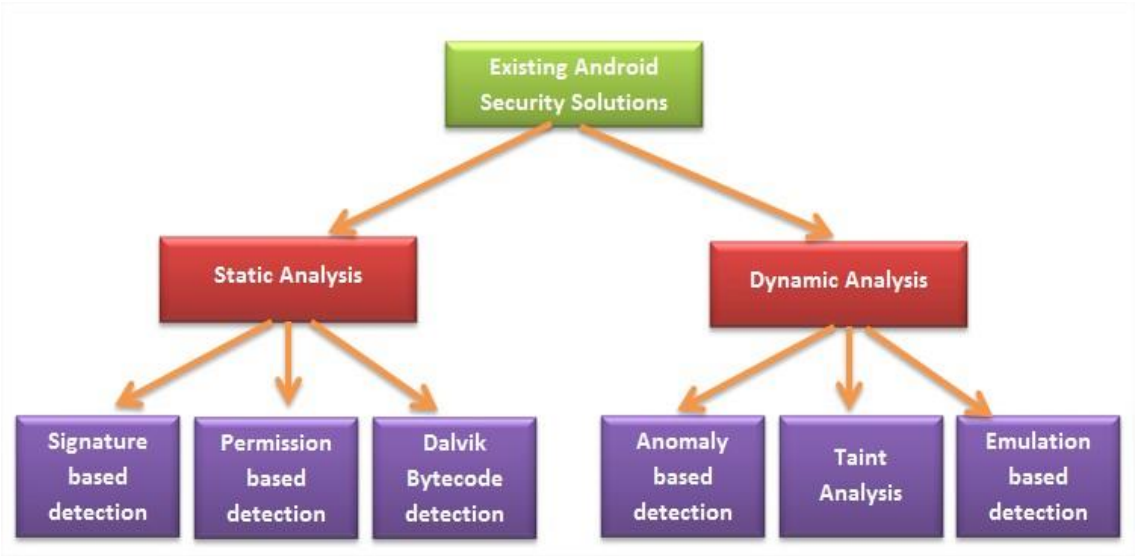



Fig. 4. Taxonomy of Existing Android Antimalwares

### A. *Static Approach*

Static approach is a way to check functionalities and maliciousness of an application by disassembling and analyzing its source code, without executing the application. It is useful for finding malicious behaviors that may not operate until the particular condition occurs.

#### 1) *Signature Based Approach*

Signature based malware detection methods are commonly used by commercial antimalware products. This method extracts the semantic patterns and creates a unique signature [25]. A program is classified as a malware if its signature matches with existing malware families' signatures. The major drawback of signature based detection is that it can be easily circumvented by code obfuscation because it can only identify the existing malwares and fails against the unseen variants of malwares. It needs immediate update of malware variants as they are detected.

Faruki *et al*. [26] proposed *AndroSimilar*, a robust statistical signature method to detect the unknown variants of existing malwares that are usually generated by using repackaging and code obfuscation techniques. It generates the variable length signature for the application under test and compares it with the signatures in AndroSimilar malware database and identify the app as malware and benign on the basis of similarity percentage. Authors tested the AndroSimilar against 1260 apps among which 6779 apps were Google Play apps and 545 apps were from third party app store. They also used code obfuscation techniques such as method renaming, string encryption, control flow obfuscation and junk method insertion techniques to change the signature of the code and tested the effectiveness of AndroSimilar against 426 samples. The solution detected more than 60% samples correctly. AndroSimilar compares the signatures of the applications in order to distinct between the malwares and benign apps but it has limited signature database as compared to the other antivirus solutions. So any unseen malwares will remain undetected. Also the similarity percentage creates the false positives as it may classify the clean apps as malicious on the basis of percentage.

*DroidAnalytics* [27] is a signature based analytic system which extract and analyze the apps at op-code level. It not only generates the signature but also associate the malware with existing malwares after identifying the malicious content. It generates 3 level signatures. First it generates signature at method level by API call tracing then combining all the signatures of methods in a class it generates the class level signatures and at third level it generates the application signature by combining the signatures of the classes in the application. Authors have used DroidAnalytics to detect 2,494 malware samples from 102 malware families and 342 repackaged malwares from other six malware families. The limitations of this method includes, it classifies the apps as malware on the basis of classes mostly used by malware families but during experiment they found some signatures that are used by both the legitimate apps and malwares. Also the similarity score used for detection of repackaged malwares do not provide 100% solution or it may also provide false positive, classify the legitimate app as malware.

- *Limitation of Signature Based Detection:* Although signature based detection is very efficient for known malwares but it cannot detect the unknown malware types. Also because of limited signature database most of the malwares remain undetected.

#### 2) *Permission Based Analysis:*

In Android system, permissions requested by the app plays a vital role in governing the access rights. By default, apps have no permission to access the user' data and effect the system security. During installation, user must allow the app to access all the resources requested by the app. Developers must mention the permissions requested for the resources in the AndroidManifest.xml file. But all declared permissions are not necessarily the required permissions for that specific application.

Ref. [28] has shown that most of the time developers have declared the permissions that are not actually required by the application which makes it difficult to detect the malicious behavior of application. Antimalware analyzes the Android Manifest.xml file where all the permissions for the resources required by the app are mentioned. *Stowaway* [28] exposes the permission over privilege problem in Android where an app requests more permissions than it actually uses. *Stowaway* performs static analysis to determine the API calls invoked by the application and then it maps the permissions required by the API calls. They found that one third applications are over privileged among 940 Android application samples. It cannot resolve the API calls invoked by applications with the use of java reflections.

In [29], authors have proposed a light weight malware detection mechanism which only analyze the manifest file and extract the information such as permissions, intent filters ( action, category and priority), process name and number of redefined permissions to detect the malicious behavior of an application. After extracting such information, they compare it with the keyword list provide in the proposed method and then calculate the malignancy score. They used *Weka* [30] which is a data mining tool for calculation of threshold value. At last they compare the malignancy score with threshold value and classify the app as malware if malignancy score exceeds threshold value. They have used 365 samples to test the efficiency of proposed solution and the solution provides 90% accurate detection. It is cost saving mechanism as it only includes the analysis of manifest file and can be implemented in other detection architectures easily to detect malwares efficiently. Also it can detect even those malwares that remain undetected by signature based detection method. This proposed solution is limited to manifest file information. Also it cannot detect the adware samples.

C. Y. Haung *et al*. [31] proposed a method for better detection of permission based malware detection which includes the analysis of both requested and required permissions as most of the time malware authors declare more permissions in the manifest file than they actually require for the application. Also it analyses the easy to retrieve features and then labels the application as benign or malware. Three different labeling types are used for this purpose which includes site based labeling; scanner based labeling and mixed labeling. In site based labeling it labels the app as benign if it is downloaded from Google official app market and if it is downloaded from some malicious source then the app is labeled as malicious. In the second labeling scheme, if the antivirus scanner declares the app as benign the app is label as benign and same for the malware case. In the mixed labeling the app is labeled on the basis of both site based and scanner based labels. After labeling all the samples are divided into three datasets and requested permissions of these datasets are analyzed by the machine learning algorithms such as *Naive Bayes, AdaBoost, Support Vector Machine and Decision Tree* [32]. On the basis of results generated by these classifiers we can evaluate the performance of permission based detection method. in [31] authors have performed experiment on data set of 124,769 benign and 480 malicious apps. They analyzed the performance of permission based detection of malware and showed that more than 81% of malicious apps samples can be detected by the permission based detection method. Proposed method provides the quick filter for malware detection but the performance values generated by the classifiers are not perfect and we cannot completely rely on those results.

Sanz Borja *et al*. [33] presented *PUMA* for detection of malicious apps by analyzing the requested permissions for application. They used permission tags such as <uses-permission> and <uses-features> present in AndroidManifest.xml file to analyze the malicious behavior of apps and applied different classifier algorithms on dataset of 357 benign apps and 249 malicious apps. The solution provides high detection rate but results generated have high false positives rate also it is not adequate for efficient detection of malware it still requires information related to other features and dynamic analysis.

Shin *et al*. [34] used a state machine based approach and formally analyze the permission based Android security model. They also verified that the specified system satisfy the security property.

Tang, Wei *et al*. [35] proposed a Security Distance Model for mitigation of Android malware. Security Distance Model is based on the concept that not a single permission is enough for an application to threaten the security of Android devices. For example an application requesting permission READ_PHONE_STATE can access the phone number and IMEI but it cannot move data out of the device. There must be a combination of permissions to affect the security model of device such as INTERNET permission allows to concept the device with the network and will be needed to move data to some remote server. The SD measure the dangerous level of application on the basis of permissions requested by the app. Authors classify the combinations of permissions into four groups and assigned threat points (TP) to each group such as TP-0, 1, 5 and 25 to Safe SD, Normal SD, Dangerous SD and Severe SD. Before the installation of new application it calculates the threat point from the combination of permissions requested by the application. That helps the user to get aware of more dangerous permissions while installation of app. It can easily detect the unknown malwares with very high threat points. They found 500 threat points for the Geinimi malware which is a very clear variation from benign apps. A limitation of this solution includes that applications with threat points between 50 and 100 are not easy to identify as benign and malware. They could be the benign apps with such permission combinations or malwares.

Enck *et al*. [36] developed *KIRIN*, a tool that provides light weight certification at installation time. It defines the security rules and simply compares the requested permissions of app with its security rules and certifies the app as malware if it fails to pass all the security rules. The installation of app is aborted if the app is attributed as malware. Authors have tested 311 applications downloaded from official Android market and found that 5 applications failed to pass the specified rules. Proposed solution is light weight as it only analyzes the Menifest.xml file. The limitation of *KIRIN* includes that it may

also declare some legitimate applications as malware because the information provided for application certification is not adequate for detection of malware.

DroidMat [37] is a tool that extracts the information from manifest file such as permissions, message passing through intents and API call tracing to analyze the behavior of application. It applies K-means clustering that increases the malware detection capability and classify the applications as benign or malware by using KNN algorithm [38]. It is more efficient than Androgaurd [39] as it takes lesser time to identify the 1,738 apps as malware or benign. Also it is cost saving as it doesn't require dynamic simulation and manual efforts. But as a static based detection method it cannot detect the malwares which dynamically load the malicious content such as *DroidKngFu* and *BaseBridge*.

- *Limitation of Permission Based Detection:* Permission based detection is a quick filter for the application scanning and identifying that whether the application is benign or malware but it only analyses the manifest file it do not analyze other files which contain the malicious code. Also there is very small difference in permissions used by the malicious and benign apps. Permission based methods require second pass to provide efficient malware detection.

*3) Dalvik Bytecode Analysis:*

In Android, Dalvik is a register-based VM. Android apps are developed in java language, compiled in java bytecode and then translated to dalvik byte code. Bytecode analysis helps us to analyze the app behavior. Control and data flow analysis detect the dangerous functionalities performed by malicious apps.

Jinyung Kim *et al.* [40] developed *SCANDAL*, a static analyzer that analyze the dalvik byte code of applications and detects the privacy leakage in applications. It determines the data flow from information source to any remote server. Dalvik bytecode contains branch, method invocation and jump instructions which alters the order of execution of code and obfuscates the code. During execution, the possible paths that an application can take can be identified by the Bytecode analysis. In [40] Authors have examined 90 applications from Android official market and 8 malicious applications from third party market place. They found privacy leakage in 11 Google market applications and 8 third party market applications. There is a need of performance optimization techniques to implement as SCANDAL consumes more time and memory for analysis of application. Also it does not support the applications which use reflections for data leakage. In the SCANDAL authors have implemented reflection semantics manually to detect the privacy leakage in malicious apps taken from black market.

Karlsen *et al.* [41] presented the first formalization of Dalvik Bytecode along with java reflective features. They examined 1700 popular Android Apps to determine what Dalvik Bytecode instructions and features are mostly used by the Android Apps. Such formalization helps to perform control and data flow analysis in order to detect the malicious apps or to identify the sensitive API calls invoked during execution. It supports the dynamic dispatch and reflective features. But it requires extension in analysis of concurrency and reflection handling.

Zhou *et al.* [42] implemented *DroidMOSS* that extract the Dalvik Byte code sequence and developer information of application by using baksmali tool [43] and generate finger prints for each app by using fuzzy hashing techniques to create the fixed sized 80 byte signature to detect the repackaged applications. On the basis of similarity score it identifies the repackaged apps. Authors have applied *DroidMOSS* to test 200 samples from six different third party market places and detected that 5% to 13% apps were repackaged. The proposed solution cannot detect the repackaged apps if the original app is not present in database. Also because of limited database most of the malwares remains undetected. Google play store may also contain malwares. The limitation of this solution also includes that they have assumed all the Google Play apps as legitimate apps and then matched the signature of the apps taken from other app store to detect the repacked apps.

*DroidAPIMiner* [44], build upon Androgaurd [39], identifies the malware by tracking the sensitive API calls , dangerous parameters invoked and package level information within the bytecode. To classify the application as benign or malware it implements KNN algorithm [38] and detected up to 99 % accuracy and 2.2% false positive rate.

Fuchs *et al.* [45] presented *SCandroid* which analyze the Android application statically as they are installed and performs data flow analysis to checks whether the data flow through the applications is consistent or not. On the basis of data flows it declares the application as safe to be run with requested permissions. Authors use it as a security certification tool for Android apps.

Many researchers worked on conversion of Dalvik bytecode to Java bytecode and then performed static analysis on java code to detect the malicious behavior of the app. *ded* [46] and *Dare* [47] are the tools used for conversion of dalvik bytecode into java bytecode. These tools are also useful when developers don't distribute the java source code, in such case one must analyze the source code to detect the malware through static analysis. *Dexpler* tool [48] converts the Dalvik bytecode into Jimple code which is used by static analysis framework named Soot [49]. It makes the Soot to read the Dalvik Bytecode directly and perform the static analysis without converting Dalvik bytecode into java bytecode. Well known static analysis framework used by researchers is WALA which perform static analysis on java bytecode to detect privacy leakage within malicious apps [50].

Chin *et al.* [51] presented a tool named *ComDroid* that detect the communication based vulnerabilities among Android apps. They have analyzed 20 samples and detected 34 exploitable vulnerabilities among 12 applications. It uses *Dedexer* tool [52] to disassemble the dex files in the app. It performs the static analysis on Dalvik files, analyzes the permissions listed in the manifest.xml file of the app, performs intraprocedural analysis and examines the Intents of the apps to detect the communication vulnerabilities

- *Limitations of Dalvik Bytecode Detection*: In this method analysis is performed at instruction level and

consumes more power and storage space. As the android devices are resource poor so they limits this detection approach.

### B. Dynamic Approach

Dynamic analysis examines the application during execution. It may miss some of the code sections that are not executed but it can easily identify the malicious behaviors that are not detected by static analysis methods. Although static analysis methods are faster to malware detection but they fail against the code obfuscation and encryption malwares.

In [53] , Egele provided a detailed overview of different dynamic analysis methods used for discrimination between malware and benign apps. Dynamic analysis approach is effective against polymorphic and metamorphic code obfuscation techniques employed by the malwares [54] but it requires more resources.

#### 1) Anomaly Based Detection

Iker *et al*. [55] proposed *CrowDroid* to detect the behavior of applications dynamically. Details of system calls invoked by the app are collected by the *Strace* tool [56] and then crowdsourcing app, which is installed on the device, creates a log file and sends it to remote server. Log file may include the following information: Device information, apps installed on device and system calls. 2-mean clustering algorithm is applied at server side to classify the application as malware or benign. Results are stored at server database. The solution provides deep analysis and thus require large amount of resources. The solution requires client app to be installed on the user' device and may classify the legitimate app as malware if it invoke more system calls.

Shabtai *et al*. [57] proposed *Andromly* , a behavior based Android malware detection system. In order to classify the application as benign or malware it continuously monitor the different features and patterns that indicate the device state such as battery level, CPU consumption etc. while it is running and then apply the machine learning algorithms to discriminate between malicious and Benign apps. the solution can detect continuous attacks and can notify the user about these attacks.

*AntiMalDroid* [58], a malware detection framework using SVM algorithm is proposed by Zhao, can identify the malicious apps and their variants during execution. First it monitors the behavior of applications and their characteristics then it categorize these characteristics as normal and malicious behavior. Then it puts the two types of characteristics into learning module and generates the signatures for the behavior characteristics, produced by learning module. Then it store the signature in database and compare it with the already existing malware and benign app signatures. It classify the app as benign if the signature matches with already existing benign app' signatures. The solution can extend the signature database dynamically and can provide high detection rate. But it consumes more time while detection process.

#### 2) Taint Analysis

Enck *et al*. [59] proposed *TaintDroid* which provides system-wide information flow tracking for Android. It can simultaneously track multiple sources of sensitive data such as camera, GPS and microphone etc. and identify the data leakage in third party developer apps. It labels the sensitive data and keeps track of that data and app when tainted data leaves moves from the device. It provides efficient tracking of sensitive information but it do not perform control flow tracking. Also it cannot track information that leaves deice and returns in network reply.

#### 3) Emulation Based Detection

Yan *et al*. [60] present Android dynamic analysis platform *DroidScope*, based on Virtual Machine Introspection. As the antimalware detect the presence of malwares because both of them reside in the same execution environment so the malwares also can detect the presence of antimalware. *DroidScope* monitors the whole operating system by staying out of the execution environment and thus have more privileges than the malware programs. It also monitors the Dalvik semantics thus the privilege escalation attacks on kernel can also be detected. It is built upon QEMU. *DroidDream* and *DroidKungFu* [61] were detected with this technique.

Blaising *et al*. [62] proposed Android Application Sandbox (*AASandbox*) which detect the suspicious applications by performing both static and dynamic analysis on them. It first extracts the .dex file into human readable form and then performs static analysis on application. Then it analyzes the low level interactions with system by execution of application in isolated sandbox environment. Actions of application are limited to sandbox due to security policy and do not affect the data on device. It uses Money tool to dynamically analyze the application behavior which randomly generates the user events like touches, clicks and gestures etc. it cannot detect the new malware types.

## V. Performance Evaluation & Analysis

In this section, we evaluate the performance of different parameters and provide a comprehensive comparison of different attributes. Table 2 provides the limitations of the static and dynamic approach of the malware detection. The malware detection through static analysis and dynamic analysis is provided in Table 3 and Table 4 respectively.

TABLE II. Limitations of Static and Dynamic Approaches

| | Mechanism | Limitations |
|---|---|---|
| Static | Signature based detection | Cannot detect unknown malware types. |
| | Permission based detection | May consider benign app as malicious because of very small difference between permissions requested by both types. |
| | Dalvik bytecode detection | More power and memory consumption. |
| Dynamic | Anomaly detection | Incorrect if a benign app shows same behaviors e.g., invoke more API calls or consumes more battery and memory. |
| | Taint Analysis | Not suitable for real time analysis<br>Reduce performance. 20 times slowdown system |
| | Emulation based detection | More resource consumption. |

On the basis of their working techniques we have deduced major limitations and benefits for each detection mechanism.

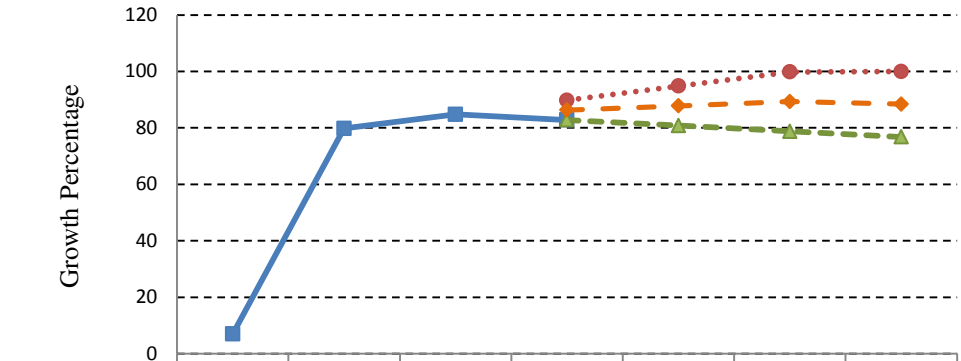
TABLE III. MALWARE DETECTION THROUGH STATIC ANALYSIS

| Approach | Name | Goal | Method | Year | Limitations | Benefits |
|---|---|---|---|---|---|---|
| **Signature Based Detection** | AndroSimilar [26] | Detect unseen and zero day samples of known malwares. | • Creates variable length signature and compares with signature database.<br>• Use fuzzy hashing technique<br>• Differentiates between benign and malicious apps on the basis of similarity percentage. | 2013 | • Limited signature database<br>• Similarity percentage may classify benign apps as malicious.<br>• Can only detect known malware variants | • Effective against code obfuscation and repackaging. |
| | DroidAnalytics [27] | Automatic collection, extraction, analysis and association of Android malwares. | • Create 3 level signatures for app on the basis of API calls.<br>• Perform Op-code level analysis (method, class, application).<br>• Correlate application with existing malwares in database via similarity score based on class level signature. | 2013 | • Similarity score may classify legitimate apps as malicious.<br>• Some level 2 signatures classified as malwares are also used by legitimate apps.<br>• Cannot detect unknown malware types. | • Effective against mutations and repackaged apps.<br>• Associates malware at op-code level<br>• Easy malware and dynamic payload tracking.<br>• Also detect dynamic malware payloads. |
| **Permission Based Detection** | Stowaway [28] | Application over privilege detection | • API call tracing through static analysis tool.<br>• Permission map to identify the permissions required by each API cal. | 2011 | • Cannot resolve complex reflective calls | • Notify about the over privileged applications. |
| | R.Sato [29] | Malware detection by manifest file analysis. | • Analyze manifest file<br>• Compare extracted information with keyword list.<br>• Calculate malignancy score<br>• Compare malignancy score with threshold values<br>• Classify the app as malware if malignancy score exceeds threshold values. | 2013 | • Cannot detect adware samples<br>• Generates results only on the basis of manifest file. | • Light weight approach<br>• Low cost<br>• Can detect the unknown malwares.<br>• Can detect the malwares that remain undetectable by signature based detection.<br>• Can be implemented in other security systems for better malware detection. |
| | C.Y.Haung [31] | Performance evaluation on permission based malware detection. | • Analyze the required and requested permissions for application<br>• Analyze easy to retrieve features<br>• Labels apps as benign or malware using site based, scanner based and mixed labeling<br>• Use machine learning algorithms on three data sets (on the basis of labels)<br>• Evaluate the permission based malware detection performance. | 2013 | • Performance numbers generated by classifiers are not perfect.<br>• Cannot completely rely on results generated by classifiers.<br>• Ada Boost identifies all apps as legitimate.<br>• Naïve Bayes also do not give précised results. | • Can use different classifiers for different scenarios.<br>• Quick filter for malware detection. |
| | PUMA [32] | Malware detection | • Analyze extracted permissions<br>• Use the <use permissions> and <use features> tags.<br>• Classify apps by using machine learning algorithms.<br>• Evaluate the performance by k-fold cross validation with k=10. | 2013 | • High false positive rate<br>• Not adequate for efficient malware detection | • High detection rate |
| | Tang Wei [34] | Application assessment and analysis to extend android security | • Uses Security Distance Model to measure dangerous level due to combination of requested permissions. | 2011 | • Applications with threat point between 50 and 100 are difficult to identify as malware or benign apps. | • Provide malware identification during installations.<br>• Can detect unknown malwares |

| | | | | | | |
|---|---|---|---|---|---|---|
| | Kirin [35] | Risk assessment and certification of applications at install time. | • Uses security rules<br>• Compares the security configuration of application with security rules<br>• Certifies the app as malware if app fails to satisfy all the security rules. | 2009 | • May declare benign app as malware because mostly similar permissions are requested by benign and malicious apps. | • Light weight certification of application at installation time.<br>• Low cost.<br>• Block the malicious applications. |
| **Dalvik Bytecode Detection** | SCANDAL [38] | Privacy leak detection | • Extracts bytecode of application as a dalvik executable file<br>• Translates dalvik executable into dalvik core, an intermediate language for efficient analysis | 2012 | • More time and memory consumption<br>• Needs performance improvement techniques to implement.<br>• Does not support applications that use reflections for privacy leakage<br>• Does not support java native interface libraries | • Saves the data from privacy leakage.<br>• Dalvik bytecode is always available.<br>• Does not need reverse engineering tools |
| | Karlsen [39] | Dalvik bytecode formalization and control flow analysis | • Provides formal control flow analysis.<br>• Formalizes dalvik bytecode language with reflection features. | 2013 | • Requires extension in analysis of reflection and concurrency handling. | • Supports reflection and dynamic dispatch features.<br>• Formal control flow analysis easily traces the API calls. |
| | DroidMOSS [40] | Repackaged malicious app detection | • Extract instructions in app and developer information.<br>• Uses baksmali tool for dalvik bytecode extraction.<br>• Generates fingerprint for each app by applying fuzzy hashing techniques<br>• Measures similarity between apps to detect repackaged apps | 2012 | • It assumes all the Google Play apps as legitimate apps.<br>• Limited database.<br>• Cannot detect repackaged apps if original app is not present in database. | • Effective detection of repackaged apps. |
| | DroidAPIMiner [42] | API level Malware detection | • Extract API level features<br>• Apply classifiers for evaluation | 2013 | • More occurrences of false positives<br>• May generate incorrect classification. | • Better accuracy. |
| | SCanDroid [43] | Application data flow analysis and security certification | • Analyze data flows in app.<br>• Make decision to classify app as benign or malware on the basis of data flow. | 2009 | • Cannot be applied to packaged applications. | • Provide security at install time. |
| | ComDroid [49] | Application communication vulnerability detection | • Extract dalvik executable files<br>• Disassemble DEX files using dedexer tool.<br>• Keep logs of the communication vulnerabilities | 2011 | • Does not verify the existence of malware<br>• Require users to manually investigate the warnings | • Issue warnings about threats. |

TABLE IV. MALWARE DETECTION THROUGH DYNAMIC ANALYSIS

| Approach | Name | Goal | Method | Year | Limitations | Benefits |
|---|---|---|---|---|---|---|
| Anomaly Detection | CrowDroid [53] | Detect anomalously behaving malicious applications | • CrowDroid client app installed on user' device.<br>• Strace tool perform system calls tracing.<br>• Creates a log file and send to remote server.<br>• Dynamic analysis is performed on the data at server side.<br>• Consider that malicious apps invoke more system calls. | 2011 | • Requires the installation of CrowDroid client application to perform detection.<br>• Results incorrect if legitimate app invokes more system calls. | • Provides deep analysis. |
| | Andromly [55] | Malware detection | • Continuously monitor the features and events e.g., battery level, data packets transferred through Internet, CPU consumption and running processes.<br>• Apply machine learning classifiers to discriminate between benign and malicious applications. | 2012 | • Only four artificially created malware instances were used for testing the system<br>• Battery drainage issue. | • Can detect the continuous attacks.<br>• Alerts the user about detected anomaly. |
| | AntiMalDroid [56] | Malware detection through characteristic learning and signature generation. | • Monitor the behavior of applications and their characteristics<br>• Categorize the characteristics into normal behavior and malicious behavior<br>• Put these characteristic types into learning module<br>• Generate behavioral characteristics.<br>• Generate the signatures for these behavioral characteristics<br>• Store these signatures to database.<br>• Compares a signature with the signatures in the database.<br>• Declares as a malware if signature matches with malware signature in database. | 2011 | • More time consumption. | • Can detect unknown malwares and their variants in runtime.<br>• Extends malware database dynamically.<br>• Higher detection rate<br>• Low cost and better performance. |
| Taint Analysis | TaintDroid [57] | Data flow analysis and leakage detection | • Automatically labels the data.<br>• Keeps track of the data.<br>• Records the label of the data, source and destination device if the data moves out of the device. | 2010 | • Only track data flows and do not track control flows.<br>• Cannot track information that leaves the device and return in network reply. | • Efficient tracking of sensitive information |
| Emulation Based Detection | DroidScope [58] | Android malware analysis | • System calls tracking<br>• Built upon QEMU (quick emulator)<br>• Monitors the OS and Dalvik semantics<br>• Perform virtual machine introspection based dynamic analysis | 2012 | • Limited code coverage | • Can detect privilege escalation attacks on the kernel. |
| | AASandbox [60] | Malware detection | • Extracts a class.dex file and decompiles it into human readable form.<br>• Performs static analysis on application.<br>• Executes the application in sandbox and perform dynamic analysis<br>• Uses Monkey tool to analyze the malicious behavior of app. | 2010 | Cannot detect new malwares | Can be used to improve the efficiency of the antimalware programs for Android OS |




| | 2012 | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 |
|---|---|---|---|---|---|---|---|
| Current | 6.93 | 79.8 | 84.8 | 82.8 | | | |
| Increase | | | | 89.8 | 94.8 | 99.8 | 99.99 |
| Decrease | | | | 82.8 | 80.8 | 78.8 | 76.8 |
| Average | | | | 86.3 | 87.8 | 89.3 | 88.4 |

Fig. 5. Expected future trends of android OS market share

## VI. Discussion

The popularity of Android operating system is increasing tremendously. The yearly records, presented by IDC [3], show that Android OS market shares in second quarter (2Q) of 2015 are 82.8%, which is 2% decrease from the 2Q 2014. If the value remains the same till the end of year and keep on decreasing every year with the same rate then we can expect that in 2018, the Android market shares will drop to 76.8%. According to same record, the Android shares have increased 5% in 2014 from previous year. If it keep on increasing with the same rate and increases up to 89.8% till the end of 2016 then we can say that the Android shares will grow up to 99.9% in 2018. Furthermore, it is predicted that the market shares of the Android will be on average 88.4% in 2018. The estimations and future predictions of the Android market are computed and plotted in Figure 5. It should be noted that with the increased usage of the Android based devices, the number of malwares attacking Android is increasing at an exponential rate. In 2015, number of Android malwares spiked to 7.10 million. This figure is 2.84 million more than the previous year [8][9]. If the malware growth keeps on increasing with the same ratio, it is expected that this number will be increased up to 15.8 million in 2018**.** The malware growth trends are predicted and estimated values are provided in Figure 6.

In contrast to malwares, the antimalware have been designed and developed in a wide range in order to protect the devices. It is inferred that an antimalware using static approach is less efficient in detecting the malicious contents that are loaded dynamically from remote servers. Although, the dynamic approach is efficient as it keeps on monitoring the application and able to detect the malicious content at execution time. However, the portions of malicious code that are not executed remain undetected. It is believed that any single security solution in Android cannot provide full protection against the vulnerabilities and malwares. It is better to deploy more than one solution simultaneously for example, a hybrid of two approaches, i.e. static and dynamic. The hybrid approach will first statically analyze the application and will then perform dynamic analysis. This hybrid solution may be an expensive method to apply because of the limited available resources such as battery, memory etc. However, the limitation of this hybrid solution can be addressed in twofold. Firstly, the static analysis can be performed locally on the Android device; and afterwards, the dynamic analysis could be performed in a distributed fashion by sending the malicious activity or event in the form of a log file to a remote server. The remote server can perform the dynamic analysis quickly and efficiently as the server will have enough resources to perform dynamic analysis and can generate rapid responses against the application behavior and the user can be instantly notified. However, this hybrid solution needs more investigation and is subject to the design tradeoffs. The future works will focus to develop such hybrid antimalware to provide better security for android devices.

## VII. Conclusion

In this paper, the malwares and their penetrations techniques have been thoroughly analyzed. The antimalware are categorized on the basis of detection methods they use. A detailed performance evaluation of these antimalware

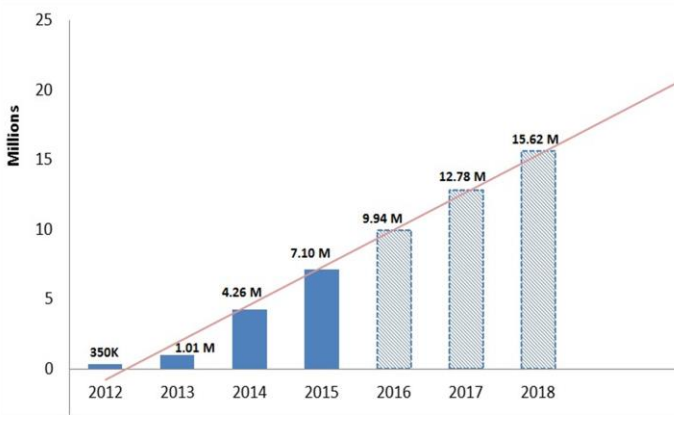



Figure 6: Future Trends of Android Malware Growth

techniques is also provided and the benefits and limitations of these antimalware are deduced comprehensively. At the end, a concept of hybrid antimalware is presented which will address the limitations of existing static and dynamic approaches. In future, it is aimed to implement the proposed hybrid solution which will be a generic antimalware that will provide better security for Android devices by firstly statically analyzing the Android applications on local device and then it will perform dynamic analysis on a remote antimalware server. This will consume very small amount of memory space on the device and the battery consumption will also be low as all dynamic analysis will be performed at the remote server.

## REFERENCES

[1] "Eric Schmidt: 'There Are Now 1.3 Million Android Device Activations Per Day.'" [Online]. Available: http://techcrunch.com/2012/09/05/eric-schmidt-there-are-now-1-3-million-android-device-activations-per-day/. [Accessed: 28-Oct-2015].

[2] "Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013." [Online]. Available: http://www.gartner.com/newsroom/id/2665715. [Accessed: 28-Oct-2015].

[3] "IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012." [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed: 08-Dec-2015].

[4] "Number of available Android applications - AppBrain." [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps. [Accessed: 28-Oct-2015].

[5] "Android and Security - Official Google Mobile Blog." [Online]. Available: http://googlemobile.blogspot.in/2012/02/android-and-security.html. [Accessed: 28-Oct-2015].

[6] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," Proc. 1st ACM Work. Secur. Priv. smartphones Mob. devices - SPSM '11, pp. 3 – 14, 2011.

[7] R. Fedler, J. Schütte, and M. Kulicke, "On the Effectiveness of Malware Protection on Android," p. 36, 2013.

[8] "Mind the (Security) Gaps: The 1H 2015 Mobile Threat Landscape - Security News - Trend Micro USA." [Online]. Available: http://www.trendmicro.com/vinfo/us/security/news/mobile-safety/mind-the-security-gaps-1h-2015-mobile-threat-landscape. [Accessed: 08-Dec-2015].

[9] "The Mobile Landscape Roundup: 1H 2014 - Security News - Trend Micro USA." [Online]. Available: http://www.trendmicro.com/vinfo/us/security/news/mobile-safety/the-mobile-landscape-roundup-1h-2014. [Accessed: 08-Dec-2015].

[10] R. Raveendranath, V. Rajamani, A. J. Babu, and S. K. Datta, "Android malware attacks and countermeasures: Current and future directions," 2014 Int. Conf. Control. Instrumentation, Commun. Comput. Technol., pp. 137–143, 2014.

[11] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," 2012 IEEE Symp. Secur. Priv., no. 4, pp. 95–109, 2012.

[12] "Security Alert: Zsone Trojan found in Android Market | Lookout Blog." [Online]. Available: https://blog.lookout.com/blog/2011/05/11/security-alert-zsone-trojan-found-in-android-market/. [Accessed: 15-Dec-2015].

[13] L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Over-the-Air Cross-platform Infection for Breaking mTAN-based Online Banking Authentication," Black Hat Abu Dhabi, pp. 1–12, 2012.

[14] "root exploits." [Online]. Available: http://www.selinuxproject.org/~jmorris/lss2011_slides/caseforseandroid.pdf. [Accessed: 15-Dec-2015].

[15] "Trojan: Android/DroidKungFu.C Description | F-Secure Labs." [Online]. Available: https://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml. [Accessed: 15-Dec-2015].

[16] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp., no. 2, pp. 5–8, 2012.

[17] "contagio mobile: Backdoor.AndroidOS.Obad.a." [Online]. Available: http://contagiominidump.blogspot.in/2013/06/backdoorandroidosobada.html. [Accessed: 28-Oct-2015].

[18] C. a Castillo, "Android Malware Past , Present , and Future," McAfee White Pap. Mob. Secur. Work. Gr., pp. 1–28, 2011

[19] "Android.Fakedefender.B | Symantec." [Online]. Available: https://www.symantec.com/security_response/writeup.jsp?docid=2013-091013-3953-99. [Accessed: 15-Dec-2015].

[20] "Riskware | Internet Security Threats." [Online]. Available: http://usa.kaspersky.com/internet-security-center/threats/riskware#.Vm-5IUp97IU. [Accessed: 15-Dec-2015].

[21] "Trend Micro Q2 Security Roundup Report | Androidheadlines.com." [Online]. Available: http://www.androidheadlines.com/2015/08/trend-micro-q2-security-roundup-report.html. [Accessed: 08-Dec-2015].

[22] "A Look at Repackaged Apps and their Effect on the Mobile Threat Landscape." [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-into-repackaged-apps-and-its-role-in-the-mobile-threat-landscape/. [Accessed: 15-Dec-2015].

[23] "NotCompatible Android Trojan: What You Need to Know | PCWorld." [Online]. Available: http://www.pcworld.com/article/254918/notcompatible_android_trojan_what_you_need_to_know.html. [Accessed: 15-Dec-2015].

[24] New Threats and Countermeasures in Digital Crime and Cyber Terrorism. IGI Global, 2015.

[25] A. Aiken, "Apposcopy : Semantics-Based Detection of Android Malware Through Static Analysis," Fse 2014, pp. 576–587, 2014.

[26] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "AndroSimilar: Robust Statistical Feature Signature for Android Malware Detection," Proc. 6th Int. Conf. Secur. Inf. Networks, pp. 152–159, 2013.

[27] M. Zheng, M. Sun, and J. C. S. Lui, "DroidAnalytics : A Signature Based Analytic System to Collect , Extract , Analyze and Associate Android Malware," 2013.

[28] Android Permissions Demystified." [Online]. Available: https://www.truststc.org/pubs/848.html. [Accessed: 06-Nov-2015].

[29] R. Sato, D. Chiba, and S. Goto, "Detecting Android Malware by Analyzing Manifest Files," pp. 23–31, 2013.

[30] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java." [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed: 16-Dec-2015].

[31] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance evaluation on permission-based detection for android malware," Adv. Intell. Syst. Appl. - Vol. 2, vol. 21, pp. 111–120, 2013.

[32] S. Ben-david, Understanding Machine Learning : From Theory to Algorithms. 2014.

[33] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "PUMA: Permission usage to detect malware in android," Adv. Intell. Syst. Comput., vol. 189 AISC, pp. 289–298, 2013.

[34] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, "Towards formal analysis of the permission-based security model for Android," 5th Int. Conf. Wirel. Mob. Commun. ICWMC 2009, pp. 87–92, 2009.

[35] W. Tang, G. Jin, J. He, and X. Jiang, "Extending android security enforcement with a security distance model," 2011 Int. Conf. Internet Technol. Appl. iTAP 2011 - Proc., 2011.

[36] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," Proc. 16th ACM Conf. Comput. Commun. Secur. - CCS '09, pp. 235–245, 2009.

[37] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android Malware Detection through Manifest and API Calls Tracing," 2012 Seventh Asia Jt. Conf. Inf. Secur., pp. 62–69, 2012.

[38] L. Kozma, "k Nearest Neighbors algorithm ( kNN )," 2008.

[39] "androguard - Reverse engineering, Malware and goodware analysis of Android applications ... and more (ninja !) - Google Project Hosting." [Online]. Available: https://code.google.com/p/androguard/. [Accessed: 01-Dec-2015].

[40] J. Kim, Y. Yoon, and K. Yi, "S CAN D AL : Static Analyzer for

Detecting Privacy Leaks in Android Applications."

[41] E. R. Wognsen, H. S. Karlsen, M. C. Olesen, and R. R. Hansen, "Formalisation and analysis of Dalvik bytecode," Sci. Comput. Program., vol. 92, no. December 2012, pp. 25–55, 2014.

[42] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," Proc. Second ACM Conf. Data Appl. Secur. Priv. - CODASKY '12, pp. 317–326, 2012.

[43] "[Utility][Tool][Windows] Baksmali / Smali Ma… | Android Development and Hacking." [Online]. Available: http://forum.xda-developers.com/showthread.php?t=2311766. [Accessed: 22-Dec-2015].

[44] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," Secur. Priv. Commun. Networks, vol. 127, pp. 86–103, 2013.

[45] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid : Automated Security Certification of Android Applications."

[46] W. Enck, D. Octeau, and P. Mcdaniel, "A Study of Android Application Security," no. August, 2011.

[47] D. Octeau, S. Jha, and P. McDaniel, "Retargeting Android applications to Java bytecode," in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12, 2012, p. 1.

[48] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon, "Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot," 2012.

[49] "A framework for analyzing and transforming Java and Android Applications." [Online]. Available: http://sable.github.io/soot/. [Accessed: 07-Nov-2015].

[50] "Main Page - WalaWiki." [Online]. Available: http://wala.sourceforge.net/wiki/index.php/Main_Page. [Accessed: 07-Nov-2015].

[51] E. Chin, A. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," Proc. 9th …, pp. 239–252, 2011.

[52] "Dedexer user's manual." [Online]. Available: http://dedexer.sourceforge.net/. [Accessed: 08-Nov-2015].

[53] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," ACM Comput. Surv., vol. 44, no. 2, pp. 1–42, 2012.

[54] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," Proc. - 2010 Int. Conf. Broadband, Wirel. Comput. Commun. Appl. BWCCA 2010, pp. 297–300, 2010.

[55] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," Proc. 1st ACM Work. Secur. Priv. smartphones Mob. devices - SPSM '11, p. 15, 2011.

[56] "strace download | SourceForge.net." [Online]. Available: http://sourceforge.net/projects/strace/. [Accessed: 22-Dec-2015].

[57] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': a behavioral malware detection framework for android devices," J. Intell. Inf. Syst., vol. 38, no. 1, pp. 161–190, 2012.

[58] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: An efficient SVM-based malware detection framework for android," Commun. Comput. Inf. Sci., vol. 243 CCIS, pp. 158–166, 2011.

[59] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," Osdi '10, vol. 49, pp. 1–6, 2010.

[60] L. Yan and H. Yin, "Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," Proc. 21st USENIX Secur. Symp., p. 29, 2012.

[61] F. Wu, H. Narang, and D. Clarke, "An Overview of Mobile Malware and Solutions," J. Comput. Commun., vol. 2, no. 2, pp. 8–17, 2014.

[62] T. Bläsing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," Proc. 5th IEEE Int. Conf. Malicious Unwanted Software, Malware 2010, pp. 55–62, 2010.