Android Malware Attacks and Countermeasures: Current and Future Directions

Rahul Raveendranath, Venkiteswaran R, Anoop Joseph Babu
College of Engineering, Trivandrum, India
{rahul92, venkit07, manoopjoseph}@gmail.com

Soumya Kanti Datta
EURECOM
Biot, France
Soumya-Kanti.Datta@eurecom.fr

***Abstract*— Smartphones are rising in popularity as well as becoming more sophisticated over recent years. This popularity coupled with the fact that smartphones contain a lot of private user data is causing a proportional rise in different malwares for the platform. In this paper we analyze and classify state-of-the-art malware techniques and their countermeasures. The paper also reports a novel method for malware development and novel attack techniques such as mobile botnets, usage pattern based attacks and repackaging attacks. The possible countermeasures are also proposed. Then a detailed analysis of one of the proposed novel malware methods is explained. Finally the paper concludes by summarizing the paper.**

Index Terms—Android; Security threats; Countermeasures; Malware; Permissions.

I. INTRODUCTION

With the rapid development in mobile computing technology, smartphones and tablets have evolved to offer sophisticated functionalities at lower costs. The Android platform has been in the forefront of this mobile revolution and has gained enormous popularity over the last four years. But the popularity has also brought the attention of major malware developers towards the platform. The fact that Android offers an open market model unlike the closed app Store model of Apple, where each application (app) is manually inspected by security experts, makes it a more favourable target for malicious developers. The existence of many third-party app stores also contributes to the spreading of malicious apps for Android platform.

We have performed an extensive literature survey where we analysed current state-of-the-art on android malwares and countermeasures. The motivation for this research is to identify the current state of malware research in Android smart devices, classify existing malware techniques and their countermeasures and through that process come up with novel suggestions for tackling recent malwares. The main contributions of this paper are twofold: (i) survey and classification of existing techniques and (ii) proposal of novel development techniques and countermeasures.

There are various potential attack scenarios where an attacker can take advantage of the vulnerabilities of the Android platform to compromise a user. A possible scenario would be where a Trojan app performs some innocent task in the foreground, say download HD wallpapers, while it secretly leaks confidential private data such as contacts from users' mobile phone. In the case of the wallpapers app, it will have INTERNET permission for downloading the wallpapers. An unsuspecting user might give not check the permissions requested and might grant READ_CONTACTS permission as well accidentally. This data can be used for monetary benefits and/or propagating the malware by the attacker. In a different attack scenario, an attacker can attempt to kill the smartphone of a victim by draining its battery life by excessive use of resource consuming services like radio, GPS etc. These apps can be distributed as repackaged versions of popular apps such as ones which offer location-based social media services. In this way, the user will be kept in the dark about private data leakage.

The rest of the paper is structured as follows. In section II, we classify the existing malwares based on our study. Thereafter, a broad classification of various countermeasures to android malwares is presented in Section III. This is followed in Section IV by our proposals on novel malware applications that we can expect in the near future and their possible countermeasures. Thereafter, the paper concludes in Section V.

II. ANALYSIS OF MALWARES

Based on our study and analysis of current literature, we classify android malwares based on their behaviour as explained below.

A. Information Extraction

Applications can easily get user's contacts, browsing history, device IMEI etc. through API calls if they have the right permissions. Many malwares tend to exploit this functionality. Marketing companies will be willing to buy such user preferences for better product targeting. These details can also end up in the hands of cyber criminals. An example of this genre of malwares is *DroidDreamLight* [1]. Consumer IMEI numbers are valuable in black markets. Blacklisted IMEIs of stolen phones can be altered with such consumer IMEIs [2].

Zhou et al [3] observed in their study of 1260 malware samples that 10% of the samples collected SMS messages, 44% samples collected user contacts and 3% obtained and uploaded user account information. Many smartphone users tend to store their confidential information such as bank details in plain text which makes the situation disastrous. Apart from this there can be phishing attacks aimed at gathering user credentials. *FakeNetflix* [4] collects users' Netflix credentials by providing an identical UI.

B. Premium Rate Calls and SMS

The cost of a premium-rate call or SMS is charged to the sender's phone bill. Many malware exploit this premium services to gather incentives to the hacker or create financial loss to the user. A malware *Fakeplayer* sends SMS message "798657" to multiple premium-rate numbers. Another well seen exploit is spamming the user contacts. Sending SMS spam is illegal in many countries. Thus sending from a compromised device reduces the risk of the spammer being caught. Also a good number of telecom operators use SMS as a medium to transfer credits without proper validation or security. This is easily exploited by the malware authors to transfer credits to their beneficiaries.

C. Root Exploits

Root exploits are carried out by both advanced users and malware authors. Users use these exploits to customize their devices whereas malwares use it to circumvent security mechanisms. The top three exploits are *Exploid, Rageagainstthecage (RATC)* and *Zimperlich*. These exploits are used to grant elevated privileges to malwares. *DroidDream, Zhash, DroidKungFu, and Basebridge* are reported to use these root exploits. *DroidKungFu* contains both *RATC* and *Exploid* root exploits in an encrypted form. When *DroidKungFu* runs, it first decrypts and launches the root exploits. If it is successful, the malware will gain root privilege and can access or modify any device on the phone including install of apps in the background without the user's knowledge [6].

D. Search Engine Optimization

A search engine's perception of relevance is influenced by the rate at which users click on the websites returned for a search term. A website will rise up in its page ranking if many people search for a specific search term and choose the link. The SEO malwares simulate this activity in the compromised device by artificially searching for the term and generating fraudulent clicks on the target website. A malware *HongTouTou* was built to boost the Baidu search result ranking of a Chinese website[1].

E. Dynamically Downloaded Code

Any android application may download executables containing native code and run those. Thus malware softwares which may originally seem legitimate can download malicious payload during runtime without being detected by anti-malware software. This loophole is considered one of the biggest issues remaining in Android security [8]. This category of malwares use sometimes use drive-by downloads in the form of plugins, extensions or updates to trick the user into downloading the payload.

F. Covert Channels

With the advent of TaintDroid, malwares that perform privacy leak have reduced considerably. To avoid detection by such tools, the use of overt and covert channels for malwares has been proposed in [9]. The authors of [10] propose improvements over the former by designing a covert channel with minimal permissions that is correlated to a user in order to be stealthy. A covert channel is a type of computer security attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy. Malware using covert channels are designed as multiple applications that communicate using non-traditional methods. They use covert channels like vibration settings, screen settings, volume settings etc. Muhammed et al. [11] proposed a new covert channel of using the file permissions to achieve communication between the collector and deliverer apps.

G. Botnets

An Android botnet is a network consisting of compromised Android smartphones controlled by a botmaster through a command and control (C&C) network [12]. The sophistication of Android botnets is increasing very rapidly. Traditional botnets communicate to the C&C server using IRC (Internet Relay Chat) protocol or using P2P (Peer-to-Peer) overlays. Popular botnet malwares like Geinimi, Pjapps, DroidDream and RootSmart exhibit traditional communication behaviour. The C&C server address is generally encrypted and stored so that it can surpass detection mechanism. A malware Geinimi applies DES encryption scheme to encrypt its communication to the server. Geng, Guining et al. [13] has proposed a new attack vector leveraging SMS structures for creating a heterogeneous mobile botnet model. Botnets can generally be used for various types of attacks such as spam delivery, DDoS attacks and for stealing personal data. A full blown botnet will have safety measures that cause bots to abort the mission and erase all traces of their existence if they are compromised. In this way, they can protect the botmaster and the C&C server.

III. Analysis of Counter measures

The countermeasures for identifying malicious applications in Android platform can be classified as follows

A. Static analysis

Many researchers suggest using static techniques for detecting possible malicious behaviour without actually executing the application. These can include extracting permissions requested from the Manifest file as well as analysing information passed through Intents, Inter-Component Communication and API calls. *DroidMat*, a tool proposed in [14] extracts these information from the byte-code and applies K-means & EM clustering algorithms to classify the app as malware or benign. Another recent paper [15] discusses an app called *Stowaway* which calculates the permissions that an app actually uses based on its API calls and compares it with the permissions requested by the app from the manifest file to detect malicious behaviour. The findings point to the extensive use of Java reflections by apps as a major reason for difficulty in tracing API calls and as a limitation of static code analysis. Also many malware authors have developed obfuscation based techniques that are especially effective against static analysis [6]. Factors that can be considered for static analysis include

- **Packages imported by the app:** This is considered by Zhou et al. [7] in their proposed app *DroidRanger*. It uses a heuristic based approach for detecting unknown malwares.

[1]https://blog.lookout.com/blog/2011/02/15/security-alert-hongtoutou-new-android-trojan-found-in-china/

This involves looking for dynamic loading of untrusted code (for eg, use of DexClassLoader) as well as looking for suspicious native code placed in non-standard locations.

- **Data flow policies via app manifest and content providers**: Fuchs et al. [18] proposed *SCanDroid* as a tool that performs data flow analysis for generating automated security certification for android applications. It detects intra-component flows by analysing uri-based addressing present in calls to Content Providers. It also detects inter-component flows by analysing intent-based addressing present in the manifest file. It then uses WALA[2], a collection of open source libraries for Java code analysis to perform data as well as flow analysis on the app.
- **Message passing through Intents**: Chin et al. [20] proposes *ComDroid*, a tool that analyses Android applications to detect communication based vulnerabilities. The tool statically analyses Dalvik executable files, performs flow sensitive intraprocedural analysis, and examines the permissions defined by the app, Intents sent by the app as well as components that receive Intents. Warnings are issued on detecting potential vulnerabilities.

B. Dynamic analysis

Dynamic or behaviour based analysis techniques involve running the app in a controlled environment, monitoring and analysing the actions performed by the app. Egele [21] provides a comprehensive overview of various automated dynamic analysis techniques. While considered more effective against various polymorphic and metamorphic malwares [22] which evade static analysis, dynamic analysis suffers from being highly resource intensive. *CrowDroid,* a tool proposed by Iker et al. [23] solves this issue by using a lightweight client application to collect system calls from different users, preprocess these and send them to a powerful remote server to perform the behavioural analysis. Key features considered for dynamic analysis include the following:

- **Data and control flow analysis:** *TaintDroid* proposed by Enck et al. [24] provides system-wide dynamic taint tracking for Android. TaintDroid marks data originating from sensitive sources like GPS, camera, microphone and other phone identifiers and monitors all network interfaces (taint sinks) for potentially sensitive data leaks.
- **Emulation based analysis:** *DroidScope* proposed by Yan et al. [25] uses virtual machine introspection to mirror the three levels of an Android device: hardware, OS and Dalvik Virtual Machine facilitating collection of detailed native and Dalvik instruction traces, profile API-level activity etc. *AASandbox* proposed by Blasing, Thomas et al. [26] executes the app in an isolated sandbox environment to analyse low level interactions with the system.
- **Logged behaviour sequence:** Zhao et al. [27] propose *AntiMalDroid* to detect Android malware that use logged behavior sequence as the feature, and construct the models for further detecting malware and its variants effectively in runtime.

[2]http://wala.sourceforge.net/

C. Application permission analysis

Permissions play an important role in governing the access rights of an app within the android system. The user must grant all the permissions requested by the app at the time of installation in order to install the app. Various approaches for analysing android malware centre around analysing the permission combinations requested by malwares. Shin, Wook et al. [28] provides a formal analysis of the permission based security model using a state machine based approach and verifies that the specified system operates satisfying the security property.

Rassameeroj et al. analysed 999 APKs in [29] and created a permission adjacency matrix on a weighted graph with permissions as nodes and the weight of each edge representing the frequency of the corresponding permissions' concurrence. They also created an APK adjacency matrix with APKs as nodes and weight of edges representing similarity of apps in terms of permissions requested. They observed that 7.9% of all APKs requests at least one combination of permissions that are considered potentially dangerous.

Tang, Wei et al. [30] proposes the concept of security distance based on the permissions used by the application. They assign each combination of permission a threat point (TP). They classify the permission combinations into four groups - Safe SD (TP - 0), Normal SD (TP - 1), Dangerous SD (TP - 5) and Severely dangerous SD (TP - 25). Before a new application is installed, from the permissions it requests the threat point for the application is found. Most of the applications have a threat point of 1-20. These can be considered to be legitimate. A very few application had a TP of 21-50. Users are warned to be careful about possible malicious activity. They found that the Geinimi malware had a TP of 500, a clear variation from legitimate applications.

Kirin, a lightweight application certification proposed by Enck et al. compares permission requests & related security configurations of apps against its security policy rules at install time [31]. If an app fails to pass all the security policies, the installation can be aborted or the user can be alerted.

D. Anomaly detection

Anomaly detection usually uses machine learning algorithms for learning known malware behaviour and predicting unknown or novel malware. *AndroMaly*, a behavioural malware detection framework for android devices proposed by Shabtai, Asaf et al. [32] continuously monitors various features and events obtained from the mobile device and applies machine learning techniques to classify apps as malicious or benign. The machine learning algorithms used by AndroMaly includes Logistic Regression, Bayesian Networks etc.

Researchers have proposed VirusMeter [33], a novel and general malware detection method which detects malicious activity by monitoring the power consumption of the mobile device. MADAM [34], a multi-level anomaly detector for android malware concurrently monitors Android at the kernel - level and user-level for issues system calls as well as smartphone parameter like user activity/idleness. It uses machine learning techniques to distinguish between standard

behaviours and malicious ones and thus detect real malicious infections.

E. Cloud-based malware protection

Considering the fact that mobile devices have computational and energy wise limitations, many malware protection solutions outsource the heavy processing tasks to remote servers. CrowDroid solution discussed earlier, crowdsources system call monitoring to lightweight client side apps and perform machine learning and clustering of apps on a cloud server. *Paranoid Android* proposed by Portokalidis, Georgios et al. [19] uses remote servers capable of running hundreds of replicas of phones in virtual environment to apply multiple detection techniques simultaneously. Researchers [16] foresee novel third party device administration solutions which provide users with 'install and forget' plans as commonly available for PCs becoming a reality in the near future. These solutions can be developed such that the data is passed to server after filtering identifiable personal user information. This helps preserve the privacy of the user.

F. Reputation based application recognition

This method consists of a central server which gathers information about all the applications in the app market. Each app gathers what is called reputation. As time passes the app gathers either good or bad reputation, which is obtained as a feedback from the users. When a new user tries to install such an application, the central server is contacted to get the reputation of the application. Only if the reputation is above a predefined threshold, the application will be allowed to be installed. This method could be unfair to newly released applications, since their reputation would be NIL. This can be overcome by assigning a default reputation to new apps and then allow the application to lose its reputation based on user rating and reporting information.

IV. Proposed Novel Malware Methods and Counter Measures

The constant improvement in anti-malware methods has helped to detect and keep many malwares in check. However, malware authors are looking for new methods to attack users. There is a lot of scope of improvements for writing new malware. Here we propose some novel methods in which future malwares can be developed and their possible counterattacks. During the course of our research, we would like to investigate how to develop some of these malwares and countermeasures that will identify and remove these types of malwares.

A. Distributed malware

One author can have multiple apps registered under their name. One exploit that could be done is to distribute the whole malware algorithm to more than one application. The apps could perform seemingly legitimate functions, like collecting users' contacts independently and communicate with a C&C server. Then these data could be merged in a remote server or a "master app" which can then carry out malware attacks.

B. Usage pattern based attacks

Many applications collect usage information from users. This is done mostly to provide services customised to the taste and interest of the user, such as better targeted advertisements. Legitimate applications do this preserving the anonymity of the user. This method could be used by malwares. The malicious application could initially collect usage statistics of the user and send it to a C&C (command & control) server. The server could analyse the collected data and using various machine learning algorithms, it could find out the best possible way to unleash the actual malicious code in order to avoid detection.

C. Repackaging attacks

Due to large number of applications being added to the Android app store every day, it is very difficult for malicious applications to gather enough popularity. In order to save the effort of garnering popularity, malware authors could download popular legitimate applications, such as Facebook and NetFlix, reverse-engineer them to obtain the source code, insert the malicious code and upload the new application to the app-markets. There is much higher possibility of users' downloading such repackaged apps compared to a newly written application [5].

D. Mobile botnets

In the future, we can expect botnets to make use of many new attack vectors. Various social networking and Internet based messaging clients have become popular within the Android platform. Botnets can use these channels in an encrypted manner to communicate with C&C servers in stealth. Social engineering based means of propagation will be a key aspect of such bots. Other attack vectors include open Wi-Fi Access Points (AP). Even though lot of research is being done on detecting botnets, we observed that remedies being suggested to bring down an established botnet network are less. Further studies need to be done to analyse possible attack vectors and how to restrict these.

E. Hardware attacks

Another possible malware action could be to continuously perform spurious computations in the background consuming a lot of the resources such as CPU cycles and battery life. Future malware applications could increasingly perform such hardware attacks.

F. Malwares that exploit NFC protocol

The use of NFC for financial transactions is expected to increase in the coming years. Malware developers could target this area to perform financial fraud as well as spread malware from device to device. Our future work would aim to uncover such possibilities.

In the coming years, Android platform will face more malware threats owing to the increase in smartphone penetration as well as increase in popularity of m-commerce platforms. These malware will be more sophisticated in nature. The basic level premium-SMS Trojans is expected to grow in number. More seriously, new Trojans that will possibly use advanced polymorphism and metamorphism based techniques making it impossible to detect them solely through static

analysis. Android malware with kernel-level rootkit has been demonstrated as a proof-of-concept already[3]. Such malwares when released will be harder to combat since they will be able to modify OS level code of the system. Researchers [17] predict worms capable of self-replicating without human intervention as the next step in the evolutionary development of malware.

G. Novel Countermeasures

Novel countermeasures will also need to be implemented to combat future malwares. The most crucial aspect for strengthening the platform will be to incorporate many advanced security measures into the Android system architecture itself. Fedler, Rafael et al. [8] suggest following enhancements to harden Android at the system level:

- Stricter Controls for Native Code Execution.
- Improving Antivirus Capabilities through a System Interface.
- Native Code Hash and Signature Validation.

YAASE [7], Yet Another Android Security Extension which provides super fine-grained access control over apps using a policy based model is also a novel countermeasure. It gives control over what kind of contacts an app can access, which sites an app can connect over the internet etc. and thus enhance the Android permission model.

In order to combat the complexity of future malwares, antimalware solutions will have to evolve and adapt multiple countermeasures in a hybrid approach rather than using one of the conventional methods.

V. USE CASE: DISTRIBUTED MALWARES

In this section, we present a detailed use case of the above proposed distributed malwares. As mentioned in the previous section, distributed malware refers to a set of applications that collectively achieve the spurious activity. We can further explain distribute malwares with the help of the following example. Suppose the following apps by the same author are present in the app market: 'Duplicate Contacts Remover' that takes the READ_CONTACTS permission; 'Missed Call Responder' that sends an auto response to missed calls and takes the SEND_SMS permission; 'Easy Music' that helps to search and download songs and takes the INTERNET permission. With such a seemingly legitimate system, malicious activity can be achieved as follows.

- The 'Duplicate Contacts Remover' application could read the user's contacts and send it to the 'Missed Call Responder' app.
- In the meanwhile, the 'Easy Music' application could download links to malicious websites from a remote server when the user is using the application and communicate it over to the 'Missed Call Responder' application.
- Now, the 'Missed Call Responder' app could send the downloaded links from 'Easy Music' app to all the users' contacts via SMS.

[3]http://www.reuters.com/article/2010/07/30/us-hackers-android-idUSTRE66T52O20100730

Thus, we can see how extensive malicious behaviour can be achieved with a very simple system. Figure3 describes the above mentioned attach scenario as well as an alternate attack scenario. Our research would primarily focus on developing and implementing such a system.

A. Permissions

In this system, the malicious code is written in such a way that the activity performed by each app only uses the permissions that the legitimate or user facing parts of the application requires. More specifically, the 'Duplicate Contacts Remover' app is expected to use the READ_CONTACTS permission in order to remove duplicate contacts. Using this permission, it can also perform the malicious behavior of reading the users contacts and storing them in a database or a shared file. Similarly, the 'Easy Music' app would definitely have internet permissions in order to download songs. With this permission, it can also download malicious links from a remote server. Also, the 'Missed Call Responder' app would require the SEND_SMS permission to send the auto-response. Using this permission, the application could send the malicious links downloaded by the 'Easy Music' app to the users contacts read by the 'Duplicate Contacts Remover' application, via SMS. Since the applications, use only their expected permissions and do not use additional permissions for performing the malicious activity, the usual static and dynamic methods cannot detect such a malware system.

B. Activation:

The 'Duplicate Contacts Remover' app could be triggered at the following points – (i) whenever the application is run and (ii) when new contact is added. Each time a new contact is added, the application is expected to run. The app can simultaneously update the file or database as well.

- The 'Easy Music' could be triggered each time the user downloads a song.
- The 'Missed Call Responder' app could be triggered whenever the user misses a call. Along with sending an auto response to the number from which the call was received, another message could be sent to a contact with the malicious link. As soon as the message is sent, the app could also hide it so that the user cannot detect it. At the same time, since the message is sent only when an auto response is sent, users in most cases do not detect the additional charge loss from their balance.

C. Limitations

In spite of the efficient stealthiness that can be achieved using the above system from both static and dynamic analysis, a limitation of this method is that all the apps that together form the malware must be present in the target phone.

VI. CONCLUSION

In this paper, we have presented an analysis and classification of the present landscape of android malwares and their countermeasures. We have provided classifications for these in terms of their behaviour. We have analysed latest research to identify novel malware techniques that can be

expected to come into action in the foreseeable future. We have also identified major system level enhancements for the Android platform as well as novel countermeasures that can be used for countering these advanced attacks. We hope to build a malware detecting and privacy leak preventing application in the near future leveraging the novel countermeasures and strategies discussed in this paper.

ACKNOWLEDGMENT

The work is a part of the Android Security project launched by Mentorship Program of Lab-X Foundation, Inc., a non-profit organization in Boston, USA.

REFERENCES

[1] Balanza, Mark et al. "Droiddreamlight lurks behind legitimate android apps." Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on 18 Oct. 2011: 73-78.

[2] Felt, Adrienne Porter et al. "A survey of mobile malware in the wild." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices 17 Oct. 2011: 3-14.

[3] Zhou, Yajin, and Xuxian Jiang. "Dissecting android malware: Characterization and evolution." Security and Privacy (SP), 2012 IEEE Symposium on 20 May. 2012: 95-109.

[4] Jiang, Xuxian, and Yajin Zhou. "A Survey of Android Malware." Android Malware (2013): 3-20.

[5] Datta, S.K.; Bonnet, C.; Nikaein, N., "Android power management: Current and future trends," Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012 First IEEE Workshop on, pp.48-53.

[6] Zhou, Yajin et al. "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 Feb. 2012.

[7] Russello, Giovanni et al. "Yaase: Yet another android security extension." Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom) 9 Oct. 2011: 1033-1040.

[8] Fedler, Rafael, Julian Schütte, and Marcel Kulicke. "On the Effectiveness of Malware Protection on Android." (2013).

[9] Marforio, Claudio et al. "Analysis of the communication between colluding applications on modern smartphones." Proceedings of the 28th Annual Computer Security Applications Conference 3 Dec. 2012: 51-60.

[10] Lalande, Jean-Francois, and Steffen Wendzel. "Hiding Privacy Leaks in Android Applications Using Low-Attention Raising Covert Channels." Availability, Reliability and Security (ARES), 2013 Eighth International Conference on 2 Sep. 2013: 701-710.

[11] Ali, Mohammad, Humayun Ali, and Zahid Anwar. "Enhancing Stealthiness & Efficiency of Android Trojans and Defense Possibilities (EnSEAD)-Android's Malware Attack, Stealthiness and Defense: An Improvement." Frontiers of Information Technology (FIT), 2011 19 Dec. 2011: 148-153.

[12] G. Geng, G. Xu, M. Zheng, Y. Gou, G. Yeng, and C. Wei, "The design of sms based heterogeneous mobile botnet," Journal of Computers, vol. 7, (1) pp. 235-243, 2012.

[13] Geng, Guining et al. "An improved sms based heterogeneous mobile botnet model." Information and Automation (ICIA), 2011 IEEE International Conference on 6 Jun. 2011: 198-202.

[14] Wu, Dong-Jie et al. "DroidMat: Android Malware Detection through Manifest and API Calls Tracing." Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on 9 Aug. 2012: 62-69.

[15] Felt, Adrienne Porter et al. "Android permissions demystified." Proceedings of the 18th ACM conference on Computer and communications security 17 Oct. 2011: 627-638.

[16] Jeter, Lukas, and Shivakant Mishra. "Identifying and quantifying the android device users' security risk exposure." Computing, Networking and Communications (ICNC), 2013 International Conference on 28 Jan. 2013: 11-17.

[17] Castillo, Carlos A. "Android malware past, present, and future." White Paper of McAfee Mobile Security Working Group (2011).

[18] Fuchs, Adam P, Avik Chaudhuri, and Jeffrey S Foster. "SCanDroid: Automated security certification of Android applications." Manuscript, Univ. of Maryland, http://www. cs. umd. edu/~ avik/projects/scandroidascaa (2009).

[19] Portokalidis, Georgios et al. "Paranoid Android: versatile protection for smartphones." Proceedings of the 26th Annual Computer Security Applications Conference 6 Dec. 2010: 347-356.

[20] Chin, Erika et al. "Poster: Analyzing Inter-Application Communication in Android."

[21] Egele, Manuel et al. "A survey on automated dynamic malware-analysis techniques and tools." ACM Computing Surveys (CSUR) 44.2 (2012): 6.

[22] You, Ilsun, and Kangbin Yim. "Malware obfuscation techniques: A brief survey." Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on 4 Nov. 2010: 297-300.

[23] Burguera, Iker, Urko Zurutuza, and Simin Nadjm-Tehrani. "Crowdroid: behavior-based malware detection system for android." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices 17 Oct. 2011: 15-26.

[24] Enck, William et al. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones." OSDI 4 Oct. 2010: 255-270.

[25] Yan, Lok Kwong, and Heng Yin. "Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis." Proceedings of the 21st USENIX Security Symposium 8 Aug. 2012.

[26] Blasing, Thomas et al. "An android application sandbox system for suspicious software detection." Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on 19 Oct. 2010: 55-62.

[27] Zhao, Min et al. "AntiMalDroid: An Efficient SVM-Based Malware Detection Framework for Android." Information Computing and Applications (2011): 158-166.

[28] Shin, Wook et al. "Towards formal analysis of the permission-based security model for android." Wireless and Mobile Communications, 2009. ICWMC'09. Fifth International Conference on 23 Aug. 2009: 87-92.

[29] Rassameeroj, Ittipon, and Yuzuru Tanahashi. "Various approaches in analyzing Android applications with its permission-based security models." Electro/Information Technology (EIT), 2011 IEEE International Conference on 15 May. 2011: 1-6.

[30] Tang, Wei et al. "Extending Android security enforcement with a security distance model." Internet Technology and Applications (iTAP), 2011 International Conference on 16 Aug. 2011: 1-4.

[31] Enck, William, Machigar Ongtang, and Patrick McDaniel. "On lightweight mobile phone application certification." Proceedings of the 16th ACM conference on Computer and communications security 9 Nov. 2009: 235-245.

[32] Shabtai, Asaf et al. ""Andromaly": a behavioral malware detection framework for android devices." Journal of Intelligent Information Systems 38.1 (2012): 161-190.

[33] Liu, Lei et al. "Virusmeter: Preventing your cellphone from spies." Recent Advances in Intrusion Detection 1 Jan. 2009: 244-264.

[34] Dini, Gianluca et al. "MADAM: a multi-level anomaly detector for android malware." Computer Network Security (2012): 240-253.