International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 04 Issue: 3 | Mar -2017 www.irjet.net p-ISSN: 2395-0072

MALWARE BYTES – ADVANCED FAULT ANALYSIS

Shanmathi.T[1], Shravya.R[2], Sneha.M[3],Anitha Moses[4]

[1],[2],[3]Student of Computer Science and Engineering Department, Panimalar Engineering College

[4]Associate Professor of Computer Science and Engineering Department, Panimalar Engineering College

---

Abstract-*The quick development of smart phones has come as an inseparable unit with a comparative increment in the number and advancement of malignant programming focusing on these stages. Malware examination is a flourishing exploration zone with a significant measure of still unsolved issues. A critical source of security issues is the capacity to merge third-party applications from accessible online markets. On account of advanced smart phones, the noteworthy development both in malware and friendly applications is making progressively unreasonably expensive any human-driven analysis of possibly risky applications. Malware sample comprising of hidden and unclear modules containing vicious components that static examination tools disregard. ALTERDROID, is an open source tool for analyzing, through reverse engineering, unclear components conveyed as parts of an application bundle. Such segments are a portion of a vicious application and are covered up outside its fundamental code segments, as code parts might be accountable to static investigation by market operators. The Malware applications are displayed on the screen, and later the authorized user can uninstall the malicious application. When the same is attempted by the unauthorized user the capture module is triggered. The quality and viability of the application is enhanced by testing ALTERDROID over the relevant applications and malware samples.*

Key Words: ALTERDROID, Malware, Computing.

1 INTRODUCTION

A smartphone is a mobile phone with an advanced mobile operating system that combines features of a personal computer operating system with other features useful for mobile or handheld use. Smartphones, which are pocket-sized. Smartphones can access the Internet and can run a variety of third-party software components ("apps" from places like Google PlayStore). They typically have a color display with a graphical user interface that covers more than 76% of the front surface. Malware, short for malicious software, is any software used to disrupt computer or mobile operations, gather sensitive information, gain access to private computer systems, or display unwanted advertising. Before the term malware, malicious software was referred to as computer viruses. Malware is intended to steal information or spy on computer users for an extended period without their knowledge. Spyware or other malware is sometimes found embedded in programs supplied officially by companies, e.g., downloadable from websites, that appear useful or attractive, but may have hidden tracking functionality that gathers various user details. Software such as anti-virus and firewalls are used to protect against activity identified as malicious, and to recover from attacks.

1.1 LITERATURE SURVEY

The signatures that should detect the confirmed malicious threats are still mainly created manually, it is important to discriminate between samples that pose a new unknown threat, and those that are mere variants of known malware [2]. a class of smartphone malware that uses steganographic techniques to hide malicious executable components within their assets, such as documents, databases, or multimedia files[8]. The fundamental deficiency in the pattern-matching approach to malware detection is that it is purely syntactic and ignores the semantics of instructions. In this paper, malware detection algorithm addresses this deficiency by incorporating instruction semantics to detect malicious program traits[4]. a lightweight method for detection of Android malware that enables identifying malicious applications directly on the smartphone[1].A system that addresses this problem by relying on stochastic models of usage and context events derived from real user traces[13].

1.2 NEED OF THE PROJECT

Malware may be anything from a virus that crashes your system to an Adware program that flashes unwanted ads or pop-ups on your screen. Other times, malware may be a Spyware program that transmits information about your computing and Internet practices to a remote user. The first category of malware propagation concerns parasitic software fragments that attach themselves to some existing executable content. The fragment may be machine

code, scripts that infects some existing application, utility, or system program, or even the code used to boot a computer system. Malware may be stealthy, intended to steal information or spy on computer users for an extended period without their knowledge and use their sensitive information for illegal purposes. For example, the hackers design a login screen on the phone that are used to capture the users' private banking details. The result of these virus injections can sometimes cause the smartphone to hang. Hence our application is used to detect malware and eradicate them.

1.3 OVERVIEW

The Google Playstore may not contain any malicious application. The mobile users may copy some application from their friends, that may contain some malware. This project finds the obfuscated malware which is present in the application. A tool called ALTERDROID is used for detecting the malware. This application scans the list of installed application present in the phone and detects if any malware present in it. If it finds any malware, it asks for permission from the user to delete the application. If the application is used by any unauthorized user it captures their image and sends to the linked e-mail id. The application also detects the malware using the script method. In the script method "Search Component Algorithm" is used for finding the terms. As a result the malware is detected from the unsigned applications.

2 EXISTING SYSTEM

One of Android's main defense mechanisms against malicious application is a risk communication mechanism which warns the user about the permissions an application requires before the application is being installed by the user, trusting that the user will make the right decision. Majority of android applications in use today require multiple permissions. The users usually neglect these permission requests by trusting the third party app-store. When a user sees what appears to be the same warning message for almost every app, warnings quickly lose any effectiveness as the users are conditioned to ignore such warnings. The user is unaware about the malicious activities involved while accepting these permissions. Every time you install an application onto your phone, you're asked to allow that app certain permissions. For example, to use your camera, track your location , view your contacts and more. While some of these permission are necessary for the application to function, some applications take advantage of that process to gather (and exploit) information they may not actually need, say consuming mobile data etc. The existing system uses a static malware analysis for detecting malware in smartphones. Static or code malware analysis fail to identify the malicious components that are embedded inside the code. This specific approach used in android has been shown to be ineffective at informing users about potential risks. This is because it usually alerts the users about the risks involved in the application only after it is being executed which does not happen in the case of static method. The major drawbacks of the existing system is that allows malicious applications and reports the errors in standalone manner. The existing system can only scan the existing files and not the applications that are installed in the device. The proposed system overcomes these limitations.

3 PROPOSED SYSTEM

ALTERDROID combines static analysis with formal methods(i.e., mathematically based techniques for the specification, development and verification of software and hardware systems). The proposed system uses a dynamic analysis approach. At the heart of our approach is a modular static analysis technique for Android applications, designed to enable incremental and automated checking of applications as they are installed, removed, or updated on an Android device. Through static analysis of each application, our approach extracts essential information and captures them in an analyzable formal specification language. Differential analysis between a candidate fault-injected application and the original application is carried out. These formal specifications are intentionally at the architectural level to ensure the technique remains scalable, yet represent the true behavior of the implemented software, as they are automatically extracted from the installation artifacts. The set of models extracted in this way are then checked as a whole for vulnerabilities that occur due to the interaction of applications comprising a system. Dynamic Malware Analysis is typically performed after static malware analysis has reached a dead end. Dynamic Malware Analysis is also a great way to identify the type of malware quickly. This approach is used to provide the holistic understanding of the behavior of an application. This feature challenges the identification of grayware and the attribution of malicious behavior to components of the application. ALTERDROID uses Alloy as a specification language which is a collection of constraints that describes a set of structures and also uses the Alloy Analyzer as the analysis engine(i.e., solver that takes the constraints of a model and finds structures that satisfy them). Alloy is a formal specification language based on first order logic, optimized for automated analysis. It includes the following: ALTERDROID 's ability in effective

compositional analysis of Android inter-app permission leakage vulnerabilities in the order of minutes and uninstall it on field. It scans the whole mobile and checks for any malware files that are stored while installing applications. It also scans the separate files to detect for any malware scripts. Secures the mobile if unauthorized person accesses your mobile to uninstall the application.

3.1 ARCHITECTURE

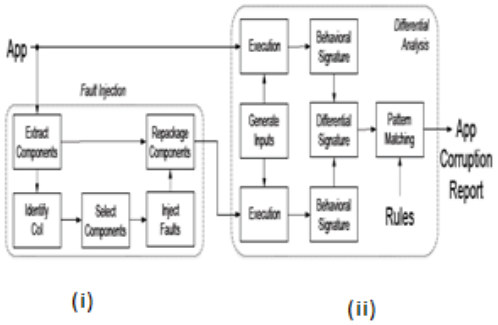

Fig.1. ALTERDROID Architecture (i)Fault Injection (ii)Differential Analysis

The architecture uses two major blocks (i) Fault Injection and (ii)Differential Analysis.The first block takes the original application as input wherein firstly the components are extracted and the components of interest (CoI) are identified. Then these components are chosen and the faults are injected. Finally these components are repackaged with the original components. The second block executes the original application and the fault injected application separately with the inputs generated by the user and the context. The behavior of these executions are transformed into two separate behavioral signatures. These signatures are then compared using the Levenshtein distance where one signature is transformed to another, called as differential signatures. A set of predefined rules undergo a pattern matching process along with these signatures to generate a report.

4 Modules of ALTERDROID

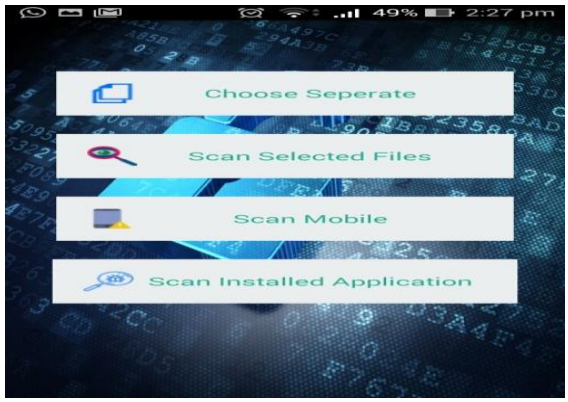

Fig .2. Modules of MalwareDetector

4.1 Choose Separate Module

Android has a growing selection of third party applications, which can be acquired by users either through an app store such as Google Play or the Amazon Appstore, or by downloading and installing the application's APK file from a third-party site. The Play Store application allows users to browse, download and update apps published by Google and third-party developers. The app filters the list of available applications to those that are compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons. But most of the users download the APK files from third party servers and installed into mobiles, Most of the apps from trusted sources are not malware, but the third party server providing malwares in modified APK. So user has the power to list all the apps installed in their mobile, then user can identify the Application is Risk or not. The choose separate module is available to the authorized and the guest users. In this module the users are able to move inside the internal storage of the mobile device and select a particular file.

4.2 Scan Selected File Module

Basically Android apps are differed by two types, Signed Application and Unsigned Applications. The user installs the third party apps from an unknown source known as unsigned apps only because signed apps are only allowed in Google play store. In the scan selected file module, particularly the selected file is alone been scanned by the device. Then the script is made given. The device checks if any script is present inside the file. If the script is made found it highlights the scripts and displays. The script may be obfuscated malware.

4.3 Scan Mobile Module

When the users share and receive the files with friends using Bluetooth, WiFi or usb, they sometimes receive unwanted files without user's knowledge and hence are stored in device. These can sometimes be the malware files such as exe, vbscript, bin that affect the android systems. This module scans the whole mobile and detects all the malware files present in the mobile device. This module does not need any input,it directly scans the device. It gives alert dialogue box and a voice note that the malware is made found. In this module the detected malware files cannot be made deleted as the files contain different paths.

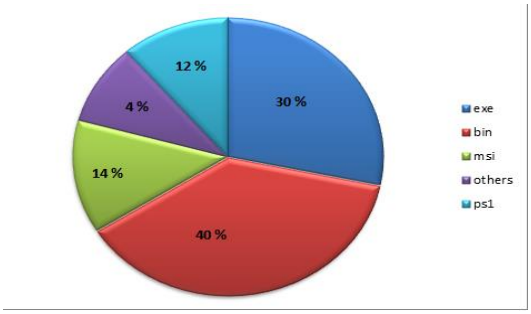

Chart 1- Percentage of various malware extensions in scan module.

4.4 Scan Installed Application Module

When the user downloads the files from internet they do not know whether the file is safe or if it contains malware. This module helps us scan the complete application. This scan installed application lists all the application present in the device. A particular application can be selected and it can be scanned .If the application is a signed application it displays all the packages present inside the application. If it is a unsigned application it detects the malware present in it and warns the user to uninstall it. To unistall the application the user is made authenticated with a mobile number and password. If it unmatched for three attempts it captures the user's image and sends it to the authorized user mail id.

4.5 Camera Module

This module is of prime importance and is also called as the hidden module. This is executed at the background without the knowledge of the user. This camera module is used for the security purpose. When an unauthorized user access the device, this module is made activated. The captured image is stored in the gallery also.

5 ALGORITHMS

5.1 Behavioral Signatures

An app interacts with the platform where it is executed by requesting services through a number of system calls. An application's behavior is described through the activities it executes.

A = {a1, a2,....} is a set of all relevant and observable activities an app can execute. The execution flow of an app P may follow different paths depending on its inputs. A sequence u of user-provided inputs. For example, those acquired through the touch screen. A sequence t of contexts, defining the state of the environment when the execution takes place like current location, time, energy level, temperature. The observable behavior resulting from the execution of P(u|t) is summarized in a behavioral signature [P(u|t)].

5.2 Differential Signature

Analyzing the differences between two observed behaviors given by their respective behavioral signatures. The differential signature $\Delta(\sigma_1, \sigma_2)$ provides a sequence of insertions, deletions, and substitutions that transforms σ_1 into σ_2.This is computed using Levenshtein distance.

5.3 Analyzing Differential Signature

$$\mathcal{P}' = \Psi(\mathcal{P}) = \Psi_r^{c_r} \circ \Psi_{r-1}^{c_{r-1}} \circ \cdots \circ \Psi_1^{c_1}(\mathcal{P})$$

be the app resulting after the sequential application of FIOs $\Psi_1, \ldots, \Psi_r$ to components $c_1, \ldots, c_r$ of app $\mathcal{P}$. Let $\sigma[\mathcal{P}]$ and $\sigma[\Psi(\mathcal{P})]$ be the behavioral signatures obtained after executing $\mathcal{P}$ and $\Psi(\mathcal{P})$ under the same conditions[2], and let $\Delta(\sigma[\mathcal{P}], \sigma[\Psi(\mathcal{P})])$ be their differential signature.

5.3.1 FIO Classes

A FIO ci is said to be indistinguishable if it does not affect the execution flow of any app. A distinguishable FIOs always manifest as nonempty differential signatures.

$$\text{ind}(\Psi^{c_i}) = \begin{cases} \texttt{true} & \text{if } \Psi^{c_i} \text{ is indistinguishable} \\ \texttt{false} & \text{otherwise} \end{cases}$$

5.4 Identifying Components of interest

The first step in the analysis of an app is identifying components of interest (CoIs), i.e., parts of an app suspicious of containing hidden functionality. This does not require the source code of the app to be available.

We say that a component c of type $\tau(c)$ in an app $\mathcal{P}$ is of interest if it does not fit a model $\mathcal{M}_{\tau(c)}$ defined for all components of type $\tau(c)$. In our current version of ALTERDROID, models measure statistical features only, such as for example the expected entropy, the byte distribution, or the average size. Such features are computed from a dataset of components of the same type, such as text files, pictures, code, etc. For each model $\mathcal{M}$, we assume a Boolean function test$(c, \mathcal{M})$ that returns `true` if c complies with $\mathcal{M}$, and `false` otherwise.

Input
App: $\mathcal{P} = \{c_1, c_2, \ldots, c_k\}$
Set of type normality models: $\{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n\}$
Set of FIOs: $\{\Psi_1, \Psi_2, \cdots, \Psi_m\}$
Mode: `normal / exhaustive`
Procedure:
1. ColS $\leftarrow \emptyset$
2. **For each** $c \in \mathcal{P}$ **do**
3. **if** [test$(c, \mathcal{M}_{\tau(c)}) =$ `false`] or [(mode = `exhaustive`) and $(\exists\, \Psi_i : \tau(\Psi_i) = \tau(c))$] **then**
4. ColS $\leftarrow$ ColS $\cup \{c\}$
5. **return** ColS

Fig .3. Algorithm for obtaining CoIS ffrom an app

simple models suffice to spot the most common and rather simple obfuscation methods observed in smartphone malware where the code is camouflaged. ALTERDRIOD also supports an exhaustive analysis mode in which some additional components maybe considered CoIs even if they comply with their type model. In this mode a component is considered CoI if there exists an indistinguishable operator for it. Formally,

$$c \in \text{ColS}(\mathcal{P}) \iff \big(\text{test}(c, \mathcal{M}_{\tau(c)}) = \text{false}\big) \text{ or } \big(\exists\, \Psi^c : \text{ind}(\Psi^c)\big).$$

The rationale for including this mode is also to check components for which is known in advance that alterations do not translate into noticeable differences.

5.5 Generating fault –injected application

Components of interests identified in the previous stage are injected with faults and reassembled, together with the remaining app components. In ALTERDROID, fault-injected application are generated one at a time and sent for differential analysis. If no evidence of malicious behavior is found in the differential analysis, the fault injection process is invoked again to generate a different faulty application.

The resulting fault injection process is as follows:

1) For each FIO Ψ_j, generate $\mathcal{P}'_j$ by applying it to all $c_i \in$ CoIS

$$\mathcal{P}'_j = \Psi_j(\mathcal{P}) = \Psi_{i_j}^{c_1} \circ \Psi_{i_j}^{c_2} \circ \cdots \circ \Psi_{i_j}^{c_n}(\mathcal{P}),$$

where $\Psi_j^{c_i}$ is the identity operator if Ψ_j is not applicable to c_i. The resulting P'_j is sent for differential analysis with respect to the original $\mathcal{P}$.
2) If there is one $\mathcal{P}'_j$ such that the differential analysis spots malicious behavior, the component responsible for it can be identified by searching over all $c_i \in$ CoIS with just the corresponding FIO Ψ_j.

Input:
App: $\mathcal{P}$
CoIS $= \{c_1, c_2, \ldots, c_n\}$
Set of FIOs: $\mathcal{F} = \{\Psi_1, \Psi_2, \cdots, \Psi_m\}$
Procedure:
1. maliciousComp $\leftarrow$ `null`
2. **For each** FIO Ψ_j **do**
3. $\mathcal{P}'_j \leftarrow \mathcal{P}$
4. **For each** $c_i \in$ CoIS **do**
5. **if** Ψ_j is applicable to c_i **then**
6. $\mathcal{P}'_j = \Psi_j^{c_i}(\mathcal{P})$
7. **if** DiffAnalysis$(\mathcal{P}, \mathcal{P}', \Psi_j) \neq \emptyset$ **then**
8. maliciousComp $\leftarrow$ SearchComponent$(\Psi_j, \mathcal{P}, \text{CoIS}, 1, n)$
9. **return** maliciousComp

Function SearchComponent$(\Psi_j, \mathcal{P}, \text{CoIS}, min, max)$
1. $\mathcal{P}'_j \leftarrow \mathcal{P}$
2. **For** $i = min$ **to** max **do**
3. **if** Ψ_j is applicable to c_i **then**
4. $\mathcal{P}'_j = \Psi_j^{c_i}(\mathcal{P})$
5. **if** DiffAnalysis$(\mathcal{P}, \mathcal{P}', \Psi_j) \neq \emptyset$ **then**
6. **if** $min = max$ **then**
7. **return** c_{min}
8. **else**
9. SearchComponent$(\Psi_j, \mathcal{P}, \text{CoIS}, min, (max - min)/2)$
10. SearchComponent$(\Psi_j, \mathcal{P}, \text{CoIS}, (max - min)/2, max)$

Fig .4. Algorithm for injecting faults and searching for malicious components after differential analysis.

5.6 Applying Differential Analysis

Differential analysis between a candidate fault-injected app and the original app is carried out.

Both the original and the fault-injected app are tested under the same conditions and using the same inputs. Note that this assumes that the execution of an app is completely deterministic.
Generate the differential signature $\Delta(\sigma[\mathcal{P}(u|t)], \sigma[\mathcal{P}'(u|t)])$ from the behavioral signatures obtained above.
Apply sequentially all rules R_i over $\Delta(\sigma[\mathcal{P}(u|t)], \sigma[\mathcal{P}'(u|t)])$ and return those for which a match is obtained.

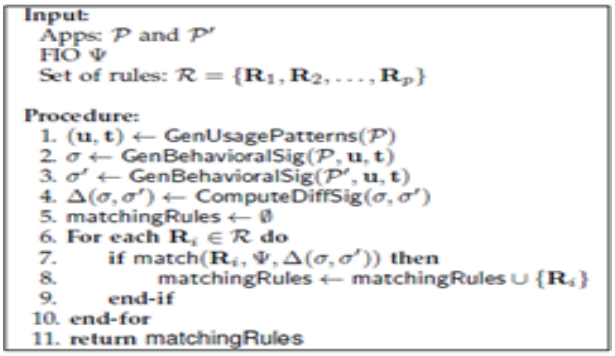

Fig .5. Algorithm for generating differential signatures and identifying matching rules.

6 RESULTS

ALTERDROID was used to test a dataset composed of 10 applications retrieved from play store. The results are tabulated in terms of signed and unsigned applications. Signed applications are those applications that displays a list of permissions which signifies that it is a trusted application. Unsigned applications are those applications that does not display its list of permissions, meaning it is an untrusted application.

Name of the app	Signed	Unsigned	Malware Ext.
Bluetooth AppSender		☑	
Limeroad	☑		
Amazon Kindle		☑	.bin
Hotstar	☑		
Whatsappp		☑	.bin, .ps1, .msi
PhotoGrid	☑		
Malfunction		☑	
Results AnnaUniv	☑		
InternalStorage		☑	.exe
Wynk Music	☑		

TABLE 1

Analysis of signed and unsigned apps with their respective malware extentions

7 CONCLUSIONS

This research focuses on a dynamic analysis approach for detecting malware in smartphones. This dynamic malware detection methodology saves a lot of time and effort as it can detect the application affected by malware without executing them. Apart from just detecting malware it also allows the users to uninstall the malicious applications. This application is brought into implementation with the belief that enormous number of people will be benefitted.

REFERENCES

[1] Arp.D, Spreitzenbarth.M, Ubner.M.H, Gascon.H, and Rieck.K, "Drebin: Effective and explainable detection of android malware in your pocket," in Proc. NDSS, February 2014.

[2] Egele.M, Scholte.T, Kirda.E, and Kruegel.C, "A survey on automated dynamic malware-analysis techniques and tools," ACM Comput. Surv., vol. 44, no. 2, pp. 6:1–6:42, Mar. 2012.

[3] Cai.L and Chen.H, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in Proc. USENIX, ser. HotSec'11, Berkeley, CA, USA, 2011, pp. 9–9.

[4] Christodorescu.M, Jha.S, Seshia.S, Song.D, and Bryant.R, "Semantics-aware malware detection," in Security and Privacy, 2005 IEEE Symposium on, May 2005, pp. 32–46.

[5] C-Skill, "Rage against the cage," https://github.com/bibanon/ android-development-codex/wiki/rageagainstthecage, 2011.

[6] Grace.M, Zhou.Y, Zhang.Q, Zou.S, and Jiang.X, "Riskranker: scalable and accurate zero-day Android malware detection," in Proc., ser. MobiSys '12. ACM, 2012, pp. 281–294.

[7] Levenshtein.V.I., "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," S. Physics Doklady, vol. 10, p. 707, 1966.

[8] Suarez-Tangil.G, Tapiador.J.E., and Peris-Lopez.P, "Stegomalware: Playing hide and seek with malicious components in smartphone apps," in INSCRYPT 2014, December 2014.

[9] Linn.C and Debray.S, "Obfuscation of executable code to improve resistance to static disassembly," in Proc. 10th ACM CCS. ACM, 2003, pp. 290–299.

[10] Rangwala.M, Zhang.P, Zou.X, and Li.F, "A taxonomy of privilege escalation attacks in android applications," Int. J. Secur. Netw., vol. 9, no. 1, pp. 40–55, Feb. 2014.

[11] Schrittwieser.S, Katzenbeisser.S, Kieseberg.P, Huber.M, Leithner.M, Mulazzani.M, and Weippl.E, "Covert computation: hiding code in code for obfuscation purposes," in Proc. 8th ACM SIGSAC, ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, pp. 529–534.

[12] Skill.C, "Gingerbreak," http://c-skills.blogspot.hk/2011/04/ yummy-yummy-gingerbreak.html, 2011.

[13] Suarez-Tangil.G, Conti.M, Tapiador.J.E and Peris-Lopez.P, "Detecting targeted smartphone malware with behavior-triggering stochastic models," in ESORICS 2014, ser. LNCS, vol. 8712. Springer International Publishing, 2014, pp. 183–201.

[14] Suarez-Tangil.G, Lombardi.F, Tapiador.J.E, and Di Pietro.R, "Thwarting obfuscated malware via differential fault analysis," IEEE Computer, vol. 47, no. 6, pp. 24–31, June 2014.[8] Rastogi.V, Chen.Y, and Enck.W, "Appsplayground: automatic security analysis of smartphone applications," in Proc. ACM, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 209–220.

[15] Suarez-Tangil.G, Tapiador.J.E, Peris.P, and Ribagorda.A, "Evolution, detection and analysis of malware for smart devices," IEEE Comms. Surveys & Tut., vol. 16, no. 2, pp. 961987, May 2014.

[16] Suarez-Tangil.G, Tapiador.J.E, Peris-Lopez.P, and Blasco.J, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," Expert Systems with Applications, vol. 41, no. 1, pp. 1104–1117, 2014.

[17] Wang.Y, Streff.K, and Raman.S, "Smartphone security challenges," IEEE Computer, vol. 45, no. 12, pp. 52–58, 2012.

[18] Zheng.C, Zhu.S, Dai.S, Gu.G, Gong.G, Han.X, and Zou.W, "Smartdroid: an automatic system for revealing UI-based trigger conditions in Android applications," in Proc. ACM, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 93–104.

[19] Zheng.M, Sun.M, and Lui.J.C, "Droidray: A security evaluation system for customized android firmwares," in Proc. ACM, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 471–482.