# A Hybrid Approach for Enhancing Android Sandbox Analysis

Ngoc-Tu Chau*, Jaehyeon Yoon† and Souhwan Jung*
*School of Electronic Engineering
Soongsil University
Seoul, Korea 06978
Email: chaungoctu@soongsil.ac.kr, souhwanj@ssu.ac.kr
† Department of Software Convergence
Soongsil University
Seoul, Korea 06978
Email: yjh7593@naver.com

***Abstract*—Dynamic analysis solutions are applied to prevent malicious applications from bypassing Android sandbox using dynamic payload techniques. However, such dynamic analysis methods are vulnerable to malware that use Anti-Analysis and Anti-Emulator techniques. Malicious applications use Anti-Emulation techniques to archive sensitive information that can be used to distinguish between sandbox and real device. Upon identifying sandbox environment, malicious applications may implement several of evasion techniques to avoid from being analyzed. The main problem, however, is that it can be easy for even a novice user to get sensitive information provided by a sandbox with just a little effort. Although there are work-around solutions for solving the problem by directly updating the sensitive information before building the sandbox, they are still containing some limitations in practice. Firstly, it is inconvenient to change the sensitive information after the sandbox or instrumentation module are deployed. Secondly, the updated information can be inconsistent and illogical. To provide a flexible approach for these issues, this paper proposes a dynamic approach that updates the sensitive information based on Sensitive Information Provider server that is located outside the sandbox. The Sensitive Information Provider (SIP) could be a collector that retrieves and processes sensitive information from one or more seeder mobile devices or could be a set of mobile devices. Because of the device-based information, the proposed approach provides a consistence and logic output when it is compared with other solutions. Furthermore, since the proposed solution separates the source of sensitive information from the sandbox, it is possible to update the sensitive information even after the sandbox was deployed. However, the proposed approach sacrifices performance to flexibility and thus it is only suitable to specific environments. The implementation section also analyzes the use-cases which are suitable to apply the proposed solution.**

***Keywords–Android Analysis; Sensitive Information Provider; Anti-Analysis***

## I. INTRODUCTION

There are three main analysis methods that have been used by Android sandboxes to analyze an application: 1) Static analysis; 2) Dynamic analysis and 3) Hybrid analysis [1]. Static analysis (also known as Source code analysis) technique works by extracting and analyzing information based on the given android application package (APK) file. AndroidManifest.xml, resources and Dalvik bytecode are the most useful information for analysis since they contain the structure of the application, permissions for the application, and behavior of the application (through the byte-code). Representatives for static analysis approach are FlowDroid [2], Droid Intent Data Flow Analysis for Information Leakage (DIDFAIL), AndroSimilar [3]. Unlike static method that performs analysis through śtatić(or non-running) source code, dynamic approach performs application analysis by running the application inside a customized sandbox. Behaviors of an analyzed application are recorded and inspected to check for malicious activity. A hybrid solution is the combination of both static and dynamic methods. On the other hand, Android malware families also evolve themselves in order to avoid being scanned by the analyzer and to bypass the sandbox system. Dynamic payload techniques are usually used by an Android malware to deal with static analysis. With dynamic payload techniques, a malicious code can be encrypted or obfuscated within the Android package to go undetected by the analyzer. However, such dynamic payload techniques are futile against dynamic analysis sandbox since the analyzed application is installed and run directly on the sandbox environment. To handle with dynamic analysis sandbox, cybercriminals usually perform anti-analysis techniques to avoid detection [4]. There are workaround solutions for solving the problem of sensitive information. Sandbox provider can modify the sensitive information before building the sandbox or takes advantage of dynamic instrumentation tools like Xposed and Frida to provide a **fixed manipulation scenario**. However, it is troublesome to rebuild the whole source code in order to change the manipulation scenario. Moreover, the function does not always work since the return of sensitive value sometimes needs to be **logic** and **consistent**. In order to archive better flexibility of sensitive information, this paper introduces a dynamic approach to separate between the sensitive information provider and sandbox environment. The proposed model is aimed at increasing the flexibility of sensitive information inside analysis sandboxes. The rest of the paper is organized as follows. Section 2 provides more information and examples about the analysis methods, anti-emulation techniques. Section 3 introduces proposed approach. Section 4 shows the implementation result with analysis of the new proposed model. Last section summaries the contribution and future research for the research topic.

## II. RELATED WORKS

In this section, in order for the reader to easily reach to the subject mentioned in the proposed models, we introduce the basic knowledge related to the analysis. In addition, solutions related to anti-analysis are also mentioned, along with related articles for providing knowledge about traditional methods of checking sandbox environment.

 ISBN: 978-1-61208-652-1 

### A. Analysis Methods

Analysis methods aim at assessing the application vulnerability as well as analysis an application for malicious code. The analysis methods consists of: 1) Static analysis; 2) Dynamic analysis and 3) Hybrid analysis. The following subsections describe detail about each approach.

*1) Static Approach:* Static method analyzes an application without actually executing the APK file. Most of static analysis methods are usually depend on the decompilation technique to repackage the APK file. There are various types of static analysis methods including: Resources-based and Bytecode-based analysis. Basically, resources-based analysis extracts the configuration and resources files for detecting abnormal information. An application are considered abnormal either when it contains patterns that match with a specific malware signature or when it requests a combination of sensitive permissions [5]–[8]. Bytecode-based analysis extracts classes, methods or instructions and applies control-flow analysis to check for malicious actions. Taint analysis that uses to keep track the data that propagate from a source of sensitive information to sink [9] [10]. Data propagation tracking method usually applied in bytecode-based analysis to detect privacy leakage. Since there is no requirement to deploy and execute application, the performance of static analysis is quite fast. However, static analysis can be easily bypassed by using dynamic payload technique like code encryption, dynamic loading, or reflection technique. There are many applicants for static analysis including Androguard, AndroSimilar, APKInspector, Drozer (also known as Mercury).

*2) Dynamic Approach:* In dynamic analysis, an application is installed and run within a customized sandbox. In order to collect behaviors of an application, the sandbox or android framework running inside sandbox has to be modified. In order to interact with the applications, tools that generate events have been used. One of the example for generating random events are monkey tool that is provided together with the Android SDK. There are various methods that applied to the sandbox to check for malicious application, but they are useless if the application refuses to execute all of its code. Because of that reason, the most challenge point in deploying a dynamic analysis sandbox is to prevent an application from performing Anti-Emulation techniques. Because an application needs to be run and inspected inside the sandbox, dynamic analysis is a trade-off between performance and efficiency. The representatives for dynamic analysis are Andromaly, Bouncer, TaintDroid, Droidbox and various of research topics about dynamic sandbox [5], [11]–[15].

*3) Hybrid Approach:* Hybrid approaches take advantage of both static and dynamic analysis. [1] has proposed a hybrid approach for Android malware analysis where both static analysis and dynamic analysis are performed and outputs are analyzed to check for suspicion behaviors. Although there are not many representatives for the hybrid approach, this idea could be a new direction for anti-malware researchers.

### B. Anti-Emulation Methods

Anti-Emulation methods are used by cybercriminals to check for the execution environment and to avoid being scanned by the sandbox. This paper divides Anti-Emulation techniques into two type of methods: 1) Sandbox Evasion and 2) Sandbox Detection. The following subsections describe detail about each approach.

Sandbox evasion technique is the method of hiding a part of source code until one or more conditions are met [16] [17]. An evasion technique that requires human interactions can be solved by tools that generate random events like Monkey tool [18]. It is note that not all applications that use evasion technique are malicious, some applications that related to financial or banking environment usually use evasion technique to avoid being run on the rooted device. Some applications that contain Easter egg, which is a hidden message or feature, also use evasion technique for hidden features. On the other side, malicious applications also depend on the evasion technique to hide their malicious code. Since both benign and malicious applications sometimes use the same evasion techniques, it is difficult for the sandbox to determine between the benign and malicious application. There are various type of evasion technique. One of the simplest methods is to configure the execute time. In the time configuration method, a malicious code will be executed whenever a certain time condition is met. The time configuration technique is effective for sandboxes that only spend fix amount of time to do analysis. The other evasion technique is human behaviour configuration. In human behaviour configuration, a specific code will be execute only if a specific human action is detected, for example: touch or scroll onto the screen. Sandbox evasion technique can be solved by simply satisfying the condition given by application. For example, a time configuration method can be circumvented by updating the sleep duration using repackaging technique or manipulating the clock of sandbox. Repackaging technique allows sandbox to decode and make modification before rebuild the source code. An evasion technique that requires human interactions can be solved by tools that generate random events like Monkey tool [18]

A more aggressive use of evasion techniques is actively detection of analysis sandbox. Sandbox detection techniques are based on the fact that sandbox is not a real mobile device [4], [19]. This paper calls the information that used to distinguish between a sandbox and real device as sensitive information. A sandbox is made by various system layers including: 1) Application layer and 2) Virtualization layer. Because of that reason, there are various type of sensitive information that can be achieved through those sandbox layers. In application layer, users (both malicious and benign) can get sensitive information through API call provided by Android framework. For example: getDeviceId call from TelephonyManager object return a device registration number of a mobile device, but it is return null in sandbox like Android Virtual Device. In Virtualization layer, sensitive information are information that related to different of network information, process difference, or caching.

## III. Proposed Model

This section describes the hybrid approach as a work-around solution for sandbox detection technique. The word *hybrid* means a combination between sandbox and mobile device. The main motivation is to separate between sandbox and sensitive information source. Figure 1 illustrates the design for our approach.

The proposed approach creates an interceptor module called Sensitive Information Interceptor (SI Interceptor) that

 ISBN: 978-1-61208-652-1

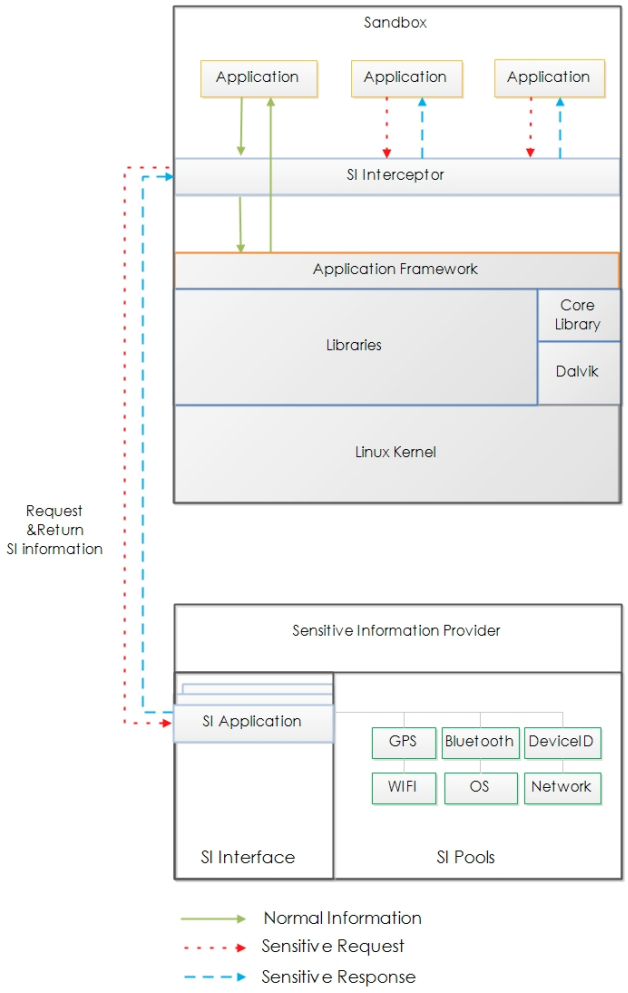


Figure 1. Hybrid Sandbox Architecture

stays between application and application framework. The SI Interceptor will check all request for information of sandbox and intercept sensitive requests. Sensitive request is a request that is expected to return a sensitive data. The sensitive request is forwarded to Sensitive Information Provider (SIP). Upon receiving the request to communicate, the SIP create an SI Application to handle the request. After get the request information, an SI Application processes and queries to the Sensitive Information Pools (SI Pools) for the related sensitive information. After getting the sensitive information, the SI Application send sensitive response to SI Interceptor that will return the data back to the requested application.

There are some points that must be considered for this architecture: 1) Feasibility, 2) Performance, and 3) Security. Because new interceptor module is added into the sandbox, it is obvious that the performance will be reduced. There are some factors that are related to performance problems including the cost of interception, network, and SI request. The SI request cost is depending on the difference between SIP side and sandbox. However, since the SIP server can provide information for many applications as well as sandboxes at the same time, SIP server can apply caching technique to some or all sensitive information. Network cost can be reduced by setup the sandbox and SIP server in the same gateway. The interception cost is depending on the intercept method that is applied to the system.

The security also needs to be considered. Since the sensitive request will be sent out to the network, it should be protected in a way that it can not be manipulated by Android applications. However, in case if the request for network information is considered as sensitive, the Android application will receive information from SIP server, which will not exposed the sandbox network information. The second security consideration is the security of SIP server. If SIP server is not a server that collect mobile information but a mobile device, it could be harm by the malicious sensitive request. In this case, the SI Interceptor should wisely decide which request could be considered as sensitive information.

TABLE I. LIST OF HYBRID API

| Type | API name | Return Type |
|---|---|---|
| TelephonyManager | getDeviceId | String |
| TelephonyManager | getLine1Number | String |
| TelephonyManager | getSubscriberId | String |
| TelephonyManager | getSimCountryIso | String |
| TelephonyManager | getNetworkCountryIso | String |
| TelephonyManager | getSimSerialNumber | String |
| TelephonyManager | getSimState | Integer |
| TelephonyManager | getNetworkType | Integer |
| LocationManager | getLastKnownLocation | Location |
| ConnectivityManager | getNetworkInfo | NetworkInfo |

## IV. IMPLEMENTATION

This paper analyses the effects of hybrid architecture on 28 malware samples that are known to include anti emulator techniques that check for sensitive information. The samples are provided by VirusShare. We decided to use only a small samples set since the main purpose of this implementation is to demonstrate the possibility of our design. Furthermore, since the approach is only at the prototype stage, the API covered by this implementation is also limited. The SIP server is a mobile device with Universal Subscriber Identity Module (USIM) setup. The authors have chosen 3 android API packages that are usually used by malware to check the sensitive information. These packages are 1) android.telephony.TelephonyManager, 2) android.location.LocationManager, and 3) android.net.ConnectivityManager. List of hybrid API is shown in the Table I.

Table II shows the result between Log API and Hybrid API when executing all apps in android VM. Log API means that the authors only log the API call and do nothing with the result. Some apps need to run with UI tool to simulate user behaviours. The result shows hybrid solution could reveal more information called by the malware in some cases.

### A. Performance Problem

In the second implementation, the author chose one method to be intercepted is ”getDeviceId” that class by the object of TelephonyManager class. This function will return the device ID of a mobile device. In case of the sandbox environment, the ”getDeviceId” method will return a null value or 00000000000000 since the value is fixed before sandbox

 ISBN: 978-1-61208-652-1 

TABLE II. LOG API AND HYBRID API

| Number | Process name | Log Only | Hybrid |
|---|---|---|---|
| 1 | nang.dv (with UI tool) | 8 | 8 |
| 2 | com.googleapi.cover | 1 | 1 |
| 3 | com.software.application | 1 | 1 |
| 4 | com.hou.jokescreen | 7 | 9 |
| 5 | com.mobi.screensaver.fzllove1 | 1 | 1 |
| 6 | com.liuwei.XiaopinClub | 17 | 33 |
| 7 | net.xfok.info.liujialing | 2 | 4 |
| 8 | com.readnovel.book_32415 | 6 | 11 |
| 9 | com.xiaoyangrenworkroom.facerecognize | 15 | 25 |
| 10 | ru.android.apps | 3 | 3 |
| 11 | Jk7H.PwcD | 5 | 5 |
| 12 | com.bratolubzet.stlstart | 2 | 2 |
| 13 | ngjvnpslnp.iplhmk | 8 | 8 |
| 14 | com.zhuaz.bugaishipdq | 0 | 0 |
| 15 | com.soft.install | 1 | 1 |
| 16 | install.app (with UI tool) | 0 | 7 |
| 17 | com.android.mmreader739 | 5 | 5 |
| 18 | com.googleapi | 2 | 2 |
| 19 | com.hdc.bookmark1566 | 1 | 2 |
| 20 | com.outfit7.talkinggingerfree | 9 | 10 |
| 21 | com.sinosoft.duanxinwzw | 39 | 62 |
| 22 | com.android.system | 0 | 6 |
| 23 | com.tencent.token | 12 | 12 |
| 24 | com.baoyi.meijiaba | 19 | 43 |
| 25 | com.unitepower.mcd33305 | 6 | 16 |
| 26 | azbc88881.jingdian10 | 6 | 25 |
| 27 | com.android.kmax.tie | 13 | 22 |
| 28 | com.androidbox.lz3net2 | 1 | 1 |

TABLE III. 1ST TIME REQUEST BETWEEN HYBRID AND NO HYBRID

| Number of 1st Times Request | Hybrid | No Hybrid |
|---|---|---|
| 1 | 210 | 0 |
| 2 | 205 | 0 |
| 3 | 403 | 1 |
| 4 | 31 | 0 |
| 5 | 149 | 0 |
| 6 | 103 | 0 |
| 7 | 79 | 0 |
| 8 | 96 | 1 |
| 9 | 130 | 0 |
| 10 | 104 | 0 |

is built. On the other hand, if the method is called by an application in a mobile phone, the return value is the device ID of that mobile. This demo involves the mobile phone as SIP server and a Virtual Machine (VM) run Android OS. Both VM and mobile phone connect to the same gateway. Also, a simple application is installed inside Android OS.

After the implementation, with 20 requests sent to the SIP server, the average time is only 7 milliseconds with caching from SIP server. The application shows a very slow response from the first request. Table III shows the cost (in milliseconds) for each 1st time request (by clearing the cache of SIP server for each try).

It is easy to notice that only the first request cost much performance, about 151 milliseconds for each request. Because of that reason, system with distributed SIP applications may reduce the average time more closely to the performance of non-intercepted sandbox.

Based on the implementation result, the effectiveness for one application with one method in proposed model could be calculated as follows:

$$\delta = \begin{cases} T_I + T_N + T_R & \text{if } 1^{st} \text{ request} \\ T_I + T_N + T_C & \text{if not } 1^{st} \text{ request} \end{cases}$$

Where $\delta$ is the average performance per request. $T_I$, $T_N$, $T_R$, and $T_C$ are the performance cost for interception, networking, request of SI information (in the SIP server), and cost for getting information from cached.

And the effectiveness for one application with n methods will be calculated as follows:

$$\delta = \frac{T(1)_I + T(1)_N + T(1)_R + \sum_{i=2}^{n} (T(i)_I + T(i)_N + T(i)_C)}{n}$$

At last, the effective of m applications with n methods is:

$$\delta = \frac{T(1)_I + T(1)_N + T(1)_R + m * \sum_{i=2}^{n} (T(i)_I + T(i)_N + T(i)_C)}{n * m}$$

As m goes larger, the SI request time will get smaller. In this case, the performance of proposed approach will depend on the time cost for intercept a method and cost for request transmit to the network. In a LAN network (SIP and sandbox have same gateway), the cost of request transmit could be very small. In an idea condition, the different between performance of proposed approach and non-intercepted approach is only depending on the interception algorithm.

### B. Pre-initialize Solution for SIP server

The main problem of proposed approach is that it takes the first sensitive request a long time to response. Since an application may usually call a method for one time only during its life-cycle, the performance will be very slow if the sensitive result have not been cached. In order to solve the problem, the SIP server could run pre-initiate function that caches common and high frequency sensitive methods before establishing communication channel with any SI interceptor. By doing the pre-initiate method, the effective of common sensitive methods could be improved into:

$$\delta = \frac{\sum_{i=1}^{n} (T(i)_I + T(i)_N + T(i)_C)}{n}$$

### C. Discussions Of The Approach

The solution can be applied as an additional module for supporting dynamic analysis. The experiments focus on feasibility and performance overhead of the method before further development. Since it is only at the prototype stage, a small number of dataset were applied. According to the performance test result, there is a delay for the first request of sensitive information in which a malicious app can use as a fingerprint for sandbox detection. However, for the second time or if there is another app already request the same information, the delay is the same as provided by existing instrumentation method.

 ISBN: 978-1-61208-652-1 

## V. Summary and Future Work

The existing solutions for manipulating sensitive information are through instrumentations and modification of Android source code. However, those existing approaches often provide fixed manipulation scenario with illogical information. Because of the above problem, this paper proposed an instrumentation-based approach for a sandbox to improve quality of sensitive information. Basically, the proposed model provides an intercept-based module for handling the request for sensitive information and forward to a remote Sensitive Information Provider (SIP) server. The SIP server has the responsibility to process and returns the value that is similar to a sensitive information of the mobile device. The performance result shows a close to non-intercepted from the second request of the same method. In the future, more research will be done in order to provide depth analysis of security problems and to optimize the architecture.

## Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2018-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion) and Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning)

## References

[1] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and et.al., "Android security: a survey of issues, malware penetration, and defenses," IEEE communications surveys & tutorials, vol. 17, no. 2, 2015, pp. 998–1022

[2] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, and et.al., "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," Acm Sigplan Notices, vol. 49, no. 6, 2014, pp. 259–269.

[3] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: robust statistical feature signature for android malware detection," in Proceedings of the 6th International Conference on Security of Information and Networks. ACM, 2013, pp. 152–159.

[4] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware," in Proceedings of the Seventh European Workshop on System Security. ACM, 2014, p. 5.

[5] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "andromaly: a behavioral malware detection framework for android devices," Journal of Intelligent Information Systems, vol. 38, no. 1, 2012, pp. 161–190.

[6] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014, pp. 576–587.

[7] G. K. Chin Erika, Felt Adrienne Porter and W. David, "Analyzing inter-application communication in android," in Proceedings of the 9th international conference on Mobile systems, applications, and services. ACM, 2011, pp. 239–252.

[8] B. Sanz, I. Santos, C. Laorden, Ugarte-Pedrero, and et.al., "Puma: Permission usage to detect malware in android," in International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions. Springer, 2013, pp. 289–298.

[9] 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." in NDSS, vol. 25, no. 4, 2012, pp. 50–52.

[10] J. Kim, Y. Yoon, K. Yi, J. Shin, and S. Center, "Scandal: Static analyzer for detecting privacy leaks in android applications," MoST, vol. 12, 2012.

[11] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," EuroSec, April, 2013.

[12] D. Damopoulos, G. Kambourakis, and G. Portokalidis, "The best of both worlds: a framework for the synergistic operation of host and cloud anomaly-based ids for smartphones," in Proceedings of the Seventh European Workshop on System Security. ACM, 2014, p. 6.

[13] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, and et.al., "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, 2014, p. 5.

[14] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011, pp. 15–26.

[15] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction," in Proceedings of the 36th International Conference on Software Engineering. ACM, 2014, pp. 1036–1046.

[16] J. Oberheide and C. Miller, "Dissecting the android bouncer," SummerCon2012, New York, 2012.

[17] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in Proceedings of the 9th ACM symposium on Information, computer and communications security. ACM, 2014, pp. 447–458.

[18] Google, "Ui/application exerciser monkey. online: http://developer.android.com/tools/help/uiautomator/index.html."

[19] D. Maier, T. Müller, and M. Protsenko, "Divide-and-conquer: Why android malware cannot be stopped," in Availability, Reliability and Security (ARES), 2014 Ninth International Conference on. IEEE, 2014, pp. 30–39.

 ISBN: 978-1-61208-652-1 