International Journal of Scientific and Technical Advancements
ISSN: 2454-1532

A Taxonomy of Prevention and Analysis Based Security Solutions for Android

Akshay Bhardwaj[1], A J Singh[2]
[1, 2]Department of Computer Science, Himachal Pradesh University, Himachal Pradesh, India
Email address: akshay117@gmail.com, aj.hpucs@gmail.com

***Abstract*—**Google's Android is a standout amongst the most well known portable working framework stages today, being sent on an extensive variety of cell phones from different makers. It is named as a benefit isolated working framework which actualizes some novel security components. Late research and security assaults on the stage, in any case, have demonstrated that the security model of Android is imperfect and is powerless against transitive use of benefits among applications. Benefit heightening assaults have been appeared to be noxious and with the across the board and developing utilization of the framework, the stage for these assaults is likewise becoming more extensive. This gives an inspiration to plan and execute better security structures and systems to alleviate these assaults. This paper talks about an arrangement and correlation of various systems and security expansions which are counteractive action and investigation based proposed in late research.

***Keywords*—** Taxonomy; solution; analysis; android; attacks.

I. INTRODUCTION

Android is a popular platform for smart phones and tablet devices. Since the first release in 2008, its popularity and sales of devices hosting the system have increased at a very fast rate.

A report by Strategy Analytics in January 2013 states that smartphone sales grew 38% in the last quarter of 2012 to reach 217 million units worldwide, and over 700 million units for the entire year. Of this number, 68.4% devices operate the Android platform. In October 2012, Google said that there were about 700,000 applications available for downloading onto Android devices matching the number of applications on Apple's App Store for iOS devices .There are a large number of end user devices and a large number of applications being used on them. Handsets today have become full-fledged computing platforms supporting complete operating systems and complex applications. However, this brings new security challenges. A recent mobile security report states that: "The sheer number of mobile applications at a time when the technology in mobile security is still in its infancy presents complex, multifaceted, and unprecedented security challenges to enterprises while putting individual privacy at high risk". The current model of the Android Application Market allows developers to upload arbitrary applications at a minimal fee. This creates a large attack surface for malicious applications to be published on the market and installed on end user devices. According to the Kaspersky Security Bulletin 2012, 99% of the newly discovered mobile malicious programs target the Android platform. Soundcomber is a trojan that uses innocuous permissions and context-aware tone- and speech-analysis to extract small amounts of targeted private data. The Trojan GGTracker (The Lookout Blog, 2012) sends SMS to a premium-rate number and can steal private information from the device. Apple screens applications posted to its AppMarket, which protects users from malicious applications. Nevertheless, applications could still have vulnerabilities that may be exploited. Google, on the other hand, does not screen applications being published to its market, making it possible for malicious applications to easily reach users. It occasionally takes down applications that are found to contain malware.

Android implements a permissions and sandboxing mechanism through its middleware layer to control access to resources and mediate inter-application communication. It is a privilege separated system, with each application having its own distinct system identity. This model is not able to prevent transitive usage of permissions that can be leveraged to launch privilege escalation attacks. It is possible for a malicious application to gain capabilities leaked from benign applications making its capabilities more than it is permitted to have. It is possible to prevent this by having strong checking of permissions; however, since developers are not security-minded, this is not used in practice. As a result, there is need for a more secure framework to be implemented to prevent these attacks.

II. TAXONOMY OF EXISTING SOLUTIONS

In this subsection we present our taxonomy of existing related security systems on Android OS. Since existing works are implemented in different ways and architectures using different techniques and mechanisms, we can categorize them in many ways. Our classification is objective-based. We group existing works in a category if they have same objective and characteristics. We categorize them into three main categories: (1) Prevention-based, (2) Analysis-based and (3) Runtime Monitoring. For this paper however we limit ourselves to discuss the first two only.

2.1 Prevention-based

Since Dalvik bytecode is vulnerable to reverse engineering, hackers are increasingly aiming at binary code targets to launch attacks on high-value mobile applications (paid/free) across all platforms. They can directly access, compromise, and exploit the binary code (e.g., analyze or reverse engineer sensitive code, modify code to change

Akshay Bhardwaj and A J Singh, "A taxonomy of prevention and analysis based security solutions for android," *International Journal of Scientific and Technical Advancements*, Volume 2, Issue 4, pp. 85-90, 2016.

application behavior, or inject malicious code) .Based on a new research study done by ARXAN .97% of the top 100 paid Android apps and 87% of the top 100 paid Apple iOS apps have been hacked using repackaging.
In this subsection, we review remarkable existing works with focus on app repackaging attacks (code modification or code injection) and reverse engineering (code analysis).

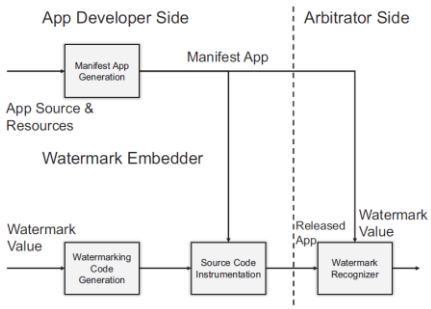

Fig. 1. Overall App ink architecture.

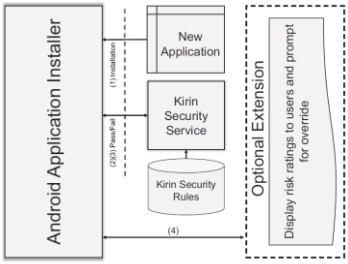

Fig. 2. Kirin based software installer.

2.1.1 Kirin

Mitigating malicious apps at install time using certification process on apps is the main goal of Kirin [1].
Kirin uses a set of predefined security rules on apps' requested permissions to find matched malicious permission requests and characteristics. Here, the rules are defined based on those permissions that are sensitive and leads to misusing of permissions and dangerous activities.
They use a static analysis tool called Pscout in order to extract all permission specifications for Android apps without modifying the apps. Using this system at install time can help users to make real-time decisions whether installing the apps or not. They tested the Kirin using 311 downloaded apps from top ranked applications from an official Android app Market. After experiments, Kirin detects 5 malicious apps with a high level of security risk. Figure 2 shows the Kirin based software installer flow and its components.

2.1.2 AppInk

In order to mitigate app repackaging, Zhou et al. propose and develop a graph-based dynamic watermarking mechanism for Android apps. They designed and developed a tool named AppInk, which takes the source code of an app and a watermark value as inputs, in order to automatically generate a new app with a transparently-embedded watermark and the associated manifest app.

They improve the system through embedding software watermarks dynamically into the running state of an app to represent the ownership of developers. After embedding the watermarks, the repackaged app can be verified by an authorized verifying party and embedded watermarks can be recognized through the manifest app without any user effort and interaction. It is worthy to note that the embedded code segments can be later recovered in order to extract the watermarks values. Figure 1 shows the overall AppInk architecture and its related components.

In order to demonstrate effectiveness and resistance of the proposed solution, they study two other works and the results indicate that AppInk is effective in defending against common automatic repackaging attacks.

2.2 Analysis-based solutions

Similar to PC malware, mobile malware has begun taking steps to evade detection by camouflaging as benign apps. In this category, the main goal is to use static and dynamic analysis to detect security sensitive and malicious behaviors of apps . Proposed works in this category focus types of attacks: (1) malicious behavior detection, (2) app similarity detection in order to detect repackaged apps, (3) misusing of granted permissions and (4) detecting apps' vulnerabilities. In this subsection we review works in any of the above subcategories.

2.2.1 RiskMon

RiskMon [4] tries to answer the question "are those behaviors necessarily inappropriate?". RiskMon is a machine-learning approach for coping with this challenge and present a continuous and automated risk assessment framework.

Figure 3 shows the basic architecture of the RiskMon. RiskMon combines users' expectations and runtime behaviors of trusted applications to generate a risk assessment baseline that captures appropriate behaviors of applications. Users' perceptions on applications is the key part of the framework. First, it collects the user's expectations on the installed apps on the device and the ranking of permission groups in terms of their relevancy to the corresponding application. Then, based on the collected information from the user, it builds the risk assessment baseline for her applications. Finally, using the generated baseline, RiskMon ranks installed applications based on risk of the app's interactions, which is measured by how much it deviates from the risk assessment baseline.

Regarding the implementation of RiskMon, it does not address the interactions between third-party applications and interactions that do not utilize Binder. This, indeed illustrates potential attack vectors that can bypass RiskMon.

2.2.2 RiskRanker

RiskRanker [5], is a proactive scheme to spot zero-day Android malware [6]. It tries to assess potential security risks caused by untrusted apps. The authors develop an automated system in order to analyze the dangerous behavior of apps dynamically.

RiskRanker's assessment system performs a two-stage risk analysis. First, it identifies apps with high and medium risk. In order to identify these apps it traces non obfuscated executions of apps that invoke (i) launching root exploits, (ii) illegal cost creation, and (iii) privacy violation attacks. In the second stage

Akshay Bhardwaj and A J Singh, "A taxonomy of prevention and analysis based security solutions for android," *International Journal of Scientific and Technical Advancements*, Volume 2, Issue 4, pp. 85-90, 2016.

of analysis, in order to discover those apps that encrypt exploit code to evade the first stage analysis it performs a further investigation through analyzing suspicious app behavior. To address this challenge, they develop a set of heuristics to map apps to related risk categories (High, Medium, and Low risk). Figure 4 shows the RiskRanker's architecture.

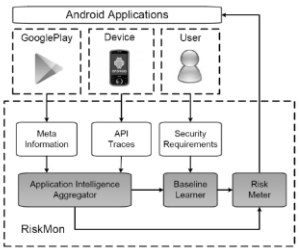

Figure 6: RiskMon architecture

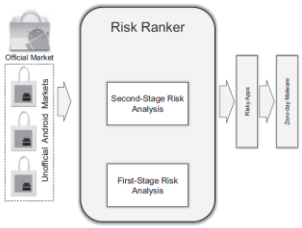

Figure 7: RiskRanker architecture

In order to evaluate the proposed solution, they implement a prototype to evaluate using 118,318 apps (104,874 distinct apps) collected from different official and unofficial app markets. After the evaluation process, the first-stage risk analysis has discovered 2,461 suspicious apps and the second-stage analysis identified 840 apps. Among these discovered 3, 281 unique apps, they successfully uncover 322 (or 9.81%) zero-day malware belonging to 11 distinct families. It should be noted that the main challenge of the RiskRanker is that they use a same set of simple heuristics against encryption and code loading, which is not effective.

2.2.2 *DroidScope*

Lok et al. present DroidScope [7], an Android analysis platform, which is based on Virtualization Malware Analysis (VMA). DroidScope reconstructs both the OS level and Java-level semantics views. In fact, DroidScope is a Virtual Machine Introspection (VMI) dynamic analysis and it is built on QEMU [8]

Emulator with a set of defined APIs as custom analysis plugins. In order to collect apps' activities and trace executions, DroidScope exports three types of APIs related to three layers of Android device: hardware, framework and Dalvik Virtual Machine.

DroidScope is tested using two Android malware families, DroidKungFu and DroidDream, and the results show that DroidScope detects them successfully. Figure 5 shows the DroidScope's architecture and its instrumentation interface.

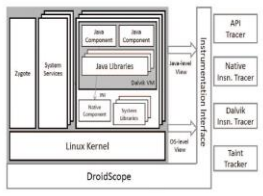

Figure 8: DroidScope's architecture and its instrumentation interface

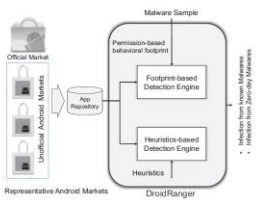

Figure 9: DroidRanger architecture

2.2.4 *DroidRanger*

In this work [2], authors present a study to evaluate the safety of apps on Google Play and some other existing unofficial Android app markets. They propose a two-stage analysis to detect current known malware and zero-day malware. In order to detect known malware, they use a permission-based behavioral footprinting scheme. In the second stage, they apply a heuristics-based filtering scheme to identify certain inherent behaviors of unknown malicious families (zero-day malware).

They tested the DroidRanger using 204, 040 apps collected from five different Android Markets. The results show that DroidRanger detected 211 malicious apps: 32 from the official Android Market (0.02% infection rate) and 179 from alternative marketplaces (infection rates ranging from 0.20% to 0.47%).The overall architecture of DroidRanger is shown in Figure 6.

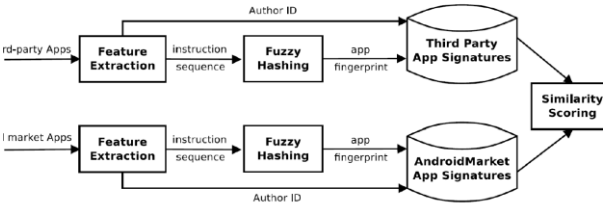

Figure 10: DroidMOSS overview

2.2.5 *DroidMoss*

In this work, an application similarity measurement system called DroidMOSS [2] is proposed that applies a fuzzy hashing technique [9][10] to localize and detect the changes from app-repackaging behavior. In fact, DroidMOSS is proposed to detect repackaged applications on third-party Android marketplaces. Given an app from a third-party Android marketplace, they measure its similarity with those apps from the official Android markets.

In order to detect a repackaged app, DroidMOSS extracts the DEC opcode sequence of an app and generates a signature fuzzy hashing signature from the opcode. Lastly, they calculate the edit distance to see how similar each app pair is. When the similarity exceeds certain threshold, they consider one app in the pair is repackaged. The above scenario is showed in Figure 7

DroidMOSS has several disadvantages. First, it only calculates the similarity for DEX bytecode and ignores the native code. Second, the opcode sequence does not consist of high level semantic information and this causes false negatives.

2.2.6 *WHYPER*

Pandita et al. propose WHYPER [11] as a Natural Language Processing (NLP) solution to measure the compatibility of requested permissions from apps. The related apps' descriptions provided by developers and the answers of why the app needs the requested permissions are used to access the compatibility of the permission requests. WHYPER takes an application's description from the market (provided by developers) and a semantic model of a permission as input,

and determines which sentence (if any) in the description indicates the use of the permission.

They have tested the WHYPER using 581 applications collected from current Android app markets.
The results show 82.8% accuracy, and an average recall of 81.5% for three special permissions (address book, calendar, and record audio) that protect frequently used security and privacy sensitive resources.

The main challenge of WHYPER is those apps that are not described by app developers and this causes false-positive detection. Figure 8 depicts an overview of WHYPER including its related components.

2.2.7 PScout

PScout [12] is proposed as a tool in order to extracts the permission specification (permission map) from the Android OS source code using static analysis. PScout works based on a call graph, constructed from API calls. The way that PScout extracts permission specifications is through performing repeated reachability analyses between API calls and permission checks on a call graph that is constructed from the Android framework's code base.Compared to the closest related work, Stowaway [13], PScout is able to extract more permission specification. In the reported experimentation, they use PScout to analyze 4 versions of Android spanning version 2.2 up to the recently released Android 4.0. On Android 2.2, PScout extracts 17, 218 mappings, whereas Stowaway derives only 1,259. Figure 9 shows PScout architecture

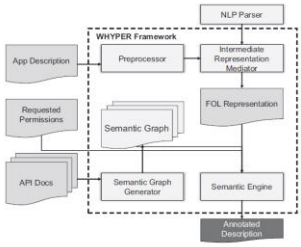

Figure 11: Overview of WHYPER

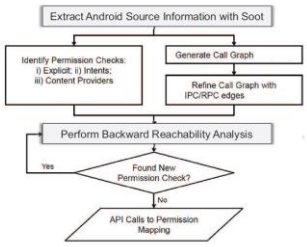

Figure 12: PScout architecture

2.2.8 AndroSimilar

In [3] authors propose AndroSimilar, an approach which generates signature by extracting statistically improbable features, to detect malicious Android apps. They claim that it is effective against code obfuscation and repackaging. AndroSimilar uses techniques such as string encryption, method renaming, junk method injection, and control flow modification to detect Android malware. AndroSimilar is a syntactic footprinting mechanism [14] that finds regions of statistical similarity with known malware to detect those unknown, zero day samples. In AndroSimilar, they use a statistical attribute extraction approach that explores improbable byte features for capturing code homogeneity among variants of known apps. After capturing the common similarities among known apps, they identify code similarity of an unknown sample and explore its similarity with known malicious family.
In fact, they generate signatures of known malware applications for different families of malware as a database of knowledge. Later, they compare the unknown applications with the captured features. If the similarity score of the comparison passes the pre-defined threshold, they label the app as a malware or repackaged app.

2.2.9 ComDroid

ComDroid [15] was proposed to detect application communication vulnerabilities. Since most of these vulnerabilities stem from the fact that Intents can be used for both intra and inter-application communication, ComDroid examines Android application interactions and identifies security risks in application components. Vulnerabilities include personal data loss and corruption, phishing, and other unexpected behaviors.

ComDroid is a two-stage solution. First, it disassemble application DEX files using the publicly available Dedexer tool [16]. After disassembling apps, it parses the disassembled output from Dedexer and logs potential component and Intent vulnerabilities. The results of the reported experimentation on 20 apps shows that ComDroid found 34 exploitable vulnerabilities; 12 of the 20 applications have at least one vulnerability.

In addition to described works in this section, there are many other related works: FlowDroid [17],Amandroid [18], AppsPlayGround [19], ScanDroid [20], VetDroid [21], Pegasus [22], AppIntent [23], Mobile-Sandbox [24], PiggyApp [25], AnDarwin [26], Juxtapp [27], Stowaway [13], DNADroid [28],Androguard [29], APKInspector [30], JEB [31], Andrubis [32], AndroTotal [33], RobotDroid [34],CHEX [35], Androwarn [36], MAdFraud [37], DECAF [38], DroidChecker [39], MARVIN [40], Shinichi et al. [41], and ProtectMyPrivacy [42]. These works all use static and dynamic analysis tools to detect apps' vulnerabilities and detect malicious apps.

III. CONCLUSION

Alongside the expanding predominance of Android cell phones, the quantity of Android applications including malware is expanding day by day. Regardless of sent Android security systems, malware exploit the Android security gaps to abuse the allowed assets. Subsequently, numerous endeavors have been proposed to limit the effort of vulnerabilities in Android gadgets. In this study we examined the current proposed works in two static and element bunches. The proposed works are fundamentally conduct based and their primary commitment is following the applications' framework calls and breaking down the exercises to confine them from high hazard exercises. Subsequent to looking into these works we concocted two inquiries that proposed works are not fit for noting fittingly. To begin with, are those practices fundamentally unseemly? Second, would we be able to name the applications as malware or amiable in view of the conduct?

REFERENCES

[1] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proc. of the 16th ACM Conference on Computer and Communications Security (CCS'09), Chicago, Illinois, USA. ACM, November 2009, pp. 235–245. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653691

[2] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in Proc. of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY'12), San Antonio, Texas, USA.ACM, March 2012, pp. 317–326. [Online]. Available: http://doi.acm.org/10.1145/2133601.2133640

[3] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: Robust statistical feature signature for android malware detection," in Proc. of the 6th International Conference on Security of Information and Networks (SIN'13), Aksaray, Turkey. ACM, November 2013, pp. 152–159. [Online]. Available: http://doi.acm.org/10.1145/2523514.2523539

[4] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Riskmon: Continuous and automated risk assessment of mobile applications," in Proc. of the 4th ACM Conference on Data and Application Security and Privacy (CODASPY'14), San Antonio, Texas, USA. ACM, March 2014, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2557547.2557549

[5] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: Scalable and accurate zero-day android malware detection," in Proc. of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys'12), Low Wood Bay, Lake District, UK. ACM, June 2012, pp. 281–294. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307663

[6] WikiPedia, "Zero-day attacks," Online; accessed at July 8, 2015, https://en.wikipedia.org/wiki/Zero-day.

[7] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in Proc. of the 21st USENIX Conference on Security Symposium (Security'12), Bellevue, WA, USA. USENIX Association, August 2012, pp. 29–29. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362822

[8] F. Bellard, in Proc. of the 2005 USENIX Annual Technical Conference, FREENIX Track, Anaheim, CA, USA. USENIX Association, April 2005, pp. 41–46.

[9] S. Server, "Fuzzy clarity: Using fuzzy hashing techniques to idēntify malicious code," Online; accessed at July 8, 2015, http://www.shadowserver.org/wiki/uploads/Information/FuzzyHashing.pdf.

[10] D. French, "Fuzzy hashing techniques in applied malware analysis," Online; accessed at July 8, 2015, http://blog.sei.cmu.edu/post.cfm/fuzzy-hashing-techniques-in-applied-malware-analysis.

[11] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in Proc. of the 22nd USENIX Conference on Security (SEC'13), Washington, D.C., USA. USENIX Association, August 2013, pp. 527–542. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534766.2534812

[12] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA. ACM, October 2012, pp. 217–228. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382222

[13] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proc. Of the 18th ACM Conference on Computer and Communications Security (CCS'11), Chicago, Illinois, USA.ACM, October 2011, pp. 627–638. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046779

[14] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in Security and Privacy (SP), 2012 IEEE Symposium on (SP'12), San Francisco Bay Area, CA, USA, May 2012, pp. 143–157.

[15] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in Proc. of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11), Bethesda, Maryland, USA. ACM, June 2011, pp. 239–252. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000018

[16] G. Paller, "Dedexer," Online; accessed at June 25, 2015, http://dedexer.sourceforge.net/.

[17] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," ACM SIGPLAN Notices, vol. 49, no. 6, pp. 259–269, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2666356.2594299

[18] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," in Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14), Scottsdale, Arizona, USA.ACM, November 2014, pp. 1329–1341. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660357

[19] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," in Proc. of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY'13), San Antonio, Texas, USA. ACM, February 2013, pp. 209–220. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435379

[20] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," ACM SIGPLAN Notices, vol. 48, no. 10, pp. 641–660, Oct. 2013. [Online]. Available:http://doi.acm.org/10.1145/2544173.2509549

[21] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS'13), Berlin, Germany.ACM, November 2013, pp. 611–622. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516689

[22] K. Z. Chen, N. Johnson, S. Dai, K. Macnamara, T. Magrino, E. Wu, M. Rinard, and D. Song, "Contextual policy enforcement in android applications with permission event graphs," 2013.

[23] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: analyzing sensitive data transmission in android for privacy leakage detection," in Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS'13), Berlin, Germany. ACM, November 2013, pp. 1043–1054. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516676

[24] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a deeper look into android applications," in Proc. of the 28th Annual ACM Symposium on Applied Computing (SAC'13), Coimbra, Portugal. ACM, March 2013, pp. 1808–1815. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480701

[25] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of "piggybacked" mobile applications," in Proc. of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY'13), San Antonio, Texas, USA. ACM, February 2013, pp. 185–196. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435377

[26] J. Crussell, C. Gibler, and H. Chen, "Andarwin: Scalable detection of semantically similar android applications," in Computer Security (ESORICS'13),, J. Crampton, S. Jajodia, and K. Mayes, Eds.Springer Berlin Heidelberg, 2013, vol. 8134, pp. ¯182–199. [Online]. Available:http: //dx.doi.org/10.1007/978-3-642-40203-6 11

[27] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A scalable system for detecting code reuse among android applications," in Proc. of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'12), Heraklion, Crete, Greece. Springer-Verlag, July 2013, pp. 62–81. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37300-8 4

[28] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on android markets," in Proc. of the 17th European Symposium on Research in Computer Security (ESORICS'12), Pisa, Italy, LNCS, S. Foresti, M. Yung, and F. Martinelli, Eds., vol. 7459. Springer Berlin Heidelberg, September 2012, pp. 37–54. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33167-1 3

[29] Androguard, "Blackhat : Reverse engineering with androguard," Online; accessed at May 23, 2015, https://code.google.com/androguard.

[30] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: Automated security validation of mobile apps at app markets," in Proc. of the 2nd International Workshop on Mobile Cloud Computing and Services (MCS'11), Bethesda, Maryland, USA. ACM, 2011, pp. 21–26. [Online]. Available: http://doi.acm.org/10.1145/1999732.1999740

[31] JEB, "Jeb decompiler," 2013. [Online]. Available:http://www.android-ecompiler.com/(Online; LastAccessed11thFebruary2013).

[32] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in Proc. of the 3rd International Workshop on Building Analysis Datasets and Gathering

Experience Returns for Security (BADGERS'14), Wroclaw, Poland, September 2014.

[33] F. Maggi, A. Valdi, and S. Zanero, "Andrototal: A flexible, scalable toolbox and service for testing mobile malware detectors," in Proc. of the 3rd ACM Workshop on Security and Privacy in Smartphones; Mobile Devices (SPSM'13), Berlin, Germany.ACM, November 2013, pp. 49–54.
[Online]. Available: http://doi.acm.org/10.1145/2516760.2516768

[34] M. Zhao, T. Zhang, F. Ge,and Z. Yuan, "Robotdroid: A lightweight malware detection framework on smartphones," Journal of Networks, vol. 7, no. 4, 2012.
[Online].Available:http://ojs.academypublisher.com/index.php/jnw/article/view/jnw0704715722

[35] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: Statically vetting android apps for component hijacking vulnerabilities," in Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA.ACM, October 2012, pp. 229–240. [Online]. Available:
http://doi.acm.org/10.1145/2382196.2382223

[36] T. Debiaze, "Detecting malicious behavior for android applications by static analysis," Online; accessed at May 23, 2015, https://github.com/maaaaz/androwarn.

[37] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," in Proc. of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'14), Bretton Woods, NH, USA. ACM, June 2014, pp. 123–134. [Online]. Available: http://doi.acm.org/10.1145/2594368.2594391

[38] B. Liu, S. Nath, R. Govindan, and J. Liu, "DECAF: Detecting and characterizing ad fraud in mobile apps," in Proc. of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14), Seattle, WA, USA. USENIX Association, April 2014, pp. 57–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616455

[39] P. P. Chan, L. C. Hui, and S. M. Yiu, "Droidchecker: Analyzing android applications for capability leak," in Proc. of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC'12), Tucson, Arizona, USA. ACM, April 2012, pp. 125–136. [Online]. Available: http://doi.acm.org/10.1145/2185448.2185466

[40] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis," in Proc. of the 39th Annual International Computers, Software & Applications Conference (COMPSAC), 2015.

[41] S. Matsumoto and K. Sakurai, "A proposal for the privacy leakage verification tool for android application developers," in Proc. of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC'13), Kota Kinabalu, Malaysia. ACM, January 2013, pp. 54:1–54:8.
[Online]. Available: http://doi.acm.org/10.1145/2448556.2448610

[42] Y. Agarwal and M. Hall, "Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing," in Proc. of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'13), Taipei, Taiwan. ACM, June 2013, pp. 97–110. [Online]. Available:
http://doi.acm.org/10.1145/2462456.2464460