A Survey of Android Security Threats and Defenses

Bahman Rashidi* and Carol Fung
Virginia Commonwealth University, Richmond, Virginia, USA
{rashidib, cfung}@vcu.edu

Abstract

With billions of people using smartphones and the exponential growth of smartphone apps, it is prohibitive for app marketplaces, such as Google App Store, to thoroughly verify if an app is legitimate or malicious. As a result, mobile users are left to decide for themselves whether an app is safe to use. Even worse, recent studies have shown that over 70% of apps in markets request to collect data irrelevant to the main functions of the apps, which could cause leaking of private information or inefficient use of mobile resources. It is worth mentioning that since resource management mechanism of mobile devices is different from PC machines, existing security solutions in PC malware area are not quite compatible with mobile devices. Therefore, academic researchers and commercial anti-malware companies have proposed many security mechanisms to address the security issues of the Android devices. Considering the mechanisms and techniques which are different in nature and used in proposed works, they can be classified into different categories. In this survey, we discuss the existing Android security threats and existing security enforcements solutions between $2010-2015$ and try to classify works and review their functionalities. We review a few works of each class. The survey also reviews the strength and weak points of the solutions.

Keywords: Android, Security, Privacy, Smartphone

1 Introduction

Since the first introduction in 2008, Android has gained a tremendous number of users over the last few years. Smartphones are the fastest growing technology market segment. According to Gartner [1], a technology research and advisory firm, 1.1 billion devices running on Google's Android OS were shipped in 2014 alone, marking its 80 percent mobile market share. Attributing to this fast-pace increament is the proliferation of Android apps, which provides an ever-growing application ecosystem. Officially reported by Android Google Play Store, the number of apps in the store has reached over 1.6 million in early 2015, surpassing its major competitor Apple Apps Store [2]. Mobile applications are essential to the smartphone experience. Mobile applications are getting increasingly sophisticated, robust, life-engaging, and privacy-intrusive. The market offers a wide variety of applications ranging from entertainment, productivity, health care, to online dating, home security, and business management [3]. Users depend more and more on mobile devices and applications.

As mobile applications are gaining increasing popularity among users, the privacy and security of smartphone users become a concern. Due to the large user base, smart devices are used to store sensitive personal information more frequently than laptops and desktops. As a consequence, a malicious third-party app can not only steal private information, such as the contact list, text messages, and location from its user, but can also cause financial loss of the users by making secretive premium-rate phone calls and text messages [4]. At the same time, the rapid growth of the number of applications on Android markets makes it hard for app market places, such as Google App Store for example, to thoroughly verify if

*Corresponding author: Department of Computer Science, Virginia Commonwealth University, Tel: +1-804-402-7575, Web: `http://people.vcu.edu/~rashidib/`

an app is legitimate or malicious. As a result, mobile users are left to decide for themselves whether an app is safe to use. In addition, unlike iOS, Android device owners do not have to root or ”jailbreak” their devices to install apps from ”unknown sources”. This gives Android users broad capability to install pirated, corrupted or banned apps from Google Play simply by changing a systems setting. This provides further incentive for the users to install third-party applications, but exposes their privacy to significant security risks [5].

The exponentially increasing number of Android applications, the unofficial apps developers, and the existing security vulnerabilities in Android OS encourage malware developers to take advantage of such vulnerable OS and apps and steal the private user information to inadvertently harms the apps markets and the developer reputation [6]. Moreover, since Android OS is an open source platform, it allows the installation of third-party market apps, stirring up dozens of regional and international app-stores such as PandaApp [7] and GetJar [8]. Android malware can gain control of device, steal private information from users, consume excessive battery, use telephony services to steal money from users’ bank accounts, and even turn the device into a botnet zombie.

There are a large variety of Android vulnerabilities and they can occur in any layers of Android OS stack, such as application layer or framework layer. Vulnerabilities also appear in benign apps through the accidental inclusion of coding mistakes or design flaws. As we described before, the flawed Android OS provides a fertile ground for attackers. There are a variety of security issues on Android phones, such as unauthorized access from one app to the others (information leakage), permission escalation, repackaging apps to inject malicious code, colluding, and Denial of Service (DoS) attacks.

Realizing these shortcomings in the current Android architecture, much efforts have been put towards addressing these problems [9]. In addition to Android OS’s various security measures such as sandboxing and Android permission model, many security and privacy solutions were proposed to cope with the existing security Android OS vulnerabilities, including many resource management systems such as [10, 11, 12] and security solutions using different approaches and techniques [13, 14, 15]. We will discuss the these techniques, approaches and tools in more details in section 3.

This paper aims at complementing the former reviews by expanding the coverage of Android security issues, and malware growth. In this survey, we will cover the major proposed works in Android OS security and privacy, and most of the existing deployed techniques and tools. The rest of this paper is organized as follows:

We first describe the Android OS and application and their architectures in Section 2 and then the Android security and its security issues in Section 3 and 4. After that, we explain all the existing security mechanism and proposed solutions in Section 5. We show the comparison results of the described solutions in section 6. Finally, conclusion in Section 7.

2 Android OS and Applications Architecture

In this section we describe the architecture of the Android OS and its applications. Android is being developed and maintained by Google and promoted by the Open Handset Alliance (OHA). Android OS is placed on top of the Linux kernel and it includes the middleware, libraries and APIs written in c language, and application software running on an application framework which includes Java-compatible libraries. Android’s source code is released by Google under open source licenses.

2.1 Framework Architecture

Android operating system is a stack of software components, which is roughly divided into five sections and four main layers as shown in the Figure 2. Android OS layers and components are explained as

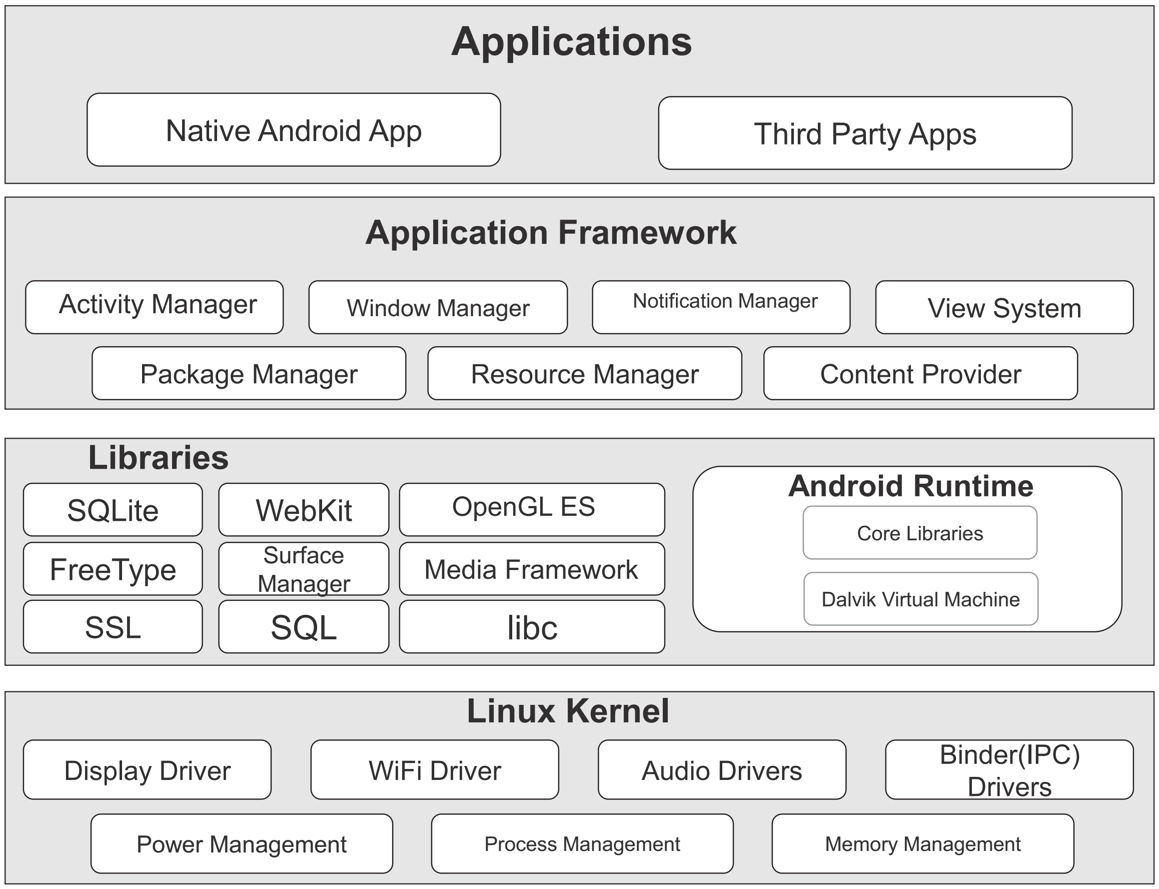

Figure 1: Android operating system architecture

follows[16][17]:

2.1.1 Applications

Application layer is located at the top of the Android software stack. These comprise both the pre-installed apps provided with a particular Android implementation and third-party apps developed by individuals (unofficial) app developers. Examples of such apps are Browser, Contacts Manager, and Email apps. More examples of such applications can be found from many official and unofficial app markets.

- **Application Framework :** The application framework is a set of services that collectively form the environment in which Android applications run and are managed. Services are provided to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications. Application framework includes the following major services [18]: *Activity Manager*, *Content Providers*, *Resource Manager*, *Notifications Manager* and *View System*.
- **Activity Manager :** Activity Manager Manages and controls all aspects of the application lifecycle and activity stack. This service interacts with the overall activities running in the system.
- **Content Providers :** Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the

standard interface that connects data in one process with code running in another process. In other words, this service allows applications to publish and share data with other applications.

- **Resource Manager :**This service provides access to non-code embedded resources such as strings, color settings and user interface layouts from apps. This service makes it possible to maintain apps' resources independently.
- **Notifications Manager :** Notifications Manager allows applications to display alerts and notifications to the user. With this service, apps can notify the user of events that happen in the background.
- **View System :** This service is an extensible set of views used to create application user interfaces.

2.1.2 Android Runtime:

This section describes a key component called *Dalvik Virtual Machine* (DVM), which is a Java Virtual Machine (JVM) specially designed and optimized for Android. Dalvik VM takes advantage of Linux core features such as multi-threading, multitasking execution environment and memory management, which is intrinsic in the Java language. Dalvik VM gives power to apps to run as a process directly on the Linux kernel and within its own VM (sandboxed). Since Dalvik is using JVM, it provides users with a set of libraries and APIs to develop Android apps predominantly using Java programming language. The standard Java development environment includes a vast array of classes that are contained in the core Java runtime libraries.

2.1.3 Libraries:

The Android's native libraries were developed on top of the Linux kernel. This layer enables the device to handle different types of data. It provides different libraries useful for the well-functioning of Android operating system. These libraries are written in C or C++ language and were developed for a particular hardware. Examples of some important native libraries include the open-source Web browser engine WebKit used to display HTML content, the well-known library libc, SQLite database engine used for data storage purposes, OpenGL used to render 2D or 3D graphics content to the screen, Media framework used to provide different media codecs, and SSL libraries for Internet security.

2.1.4 Kernel:

The Linux kernel is the fundamental layer of the entire system. This layer is customized specially for the embedded environment consisting of limited resources. The whole Android OS is built on top of the Linux kernel with some further architectural changes made by Google. This section also acts as an abstraction layer between the hardware and other software layers. Linux kernel provides the basic system functionality such as process management, memory management and device management. Linux kernel also provides an array of device drivers which make the task easier while interfacing the Android with peripheral devices.

2.2 Application Structure

Android applications which extend the functionality of devices are written primarily in the Java programming language. In this subsection, we explain Android app package structure and its main four components.

2.2.1 Android .apk package

An Android application contains several files and folders packed as a package with *.apk* extension used to distribute and install application software and middleware onto Google's Android operating system. Figure 2 depicts the structure of an Android application package. Some particular components in application files play an important role. For example, META-INF directory includes `MANIFEST.MF`, which contains a cryptographic signature and makes the entire contents of the distribution package validated. The *lib* directory contains the compiled code, which is specific to a software layer of a processor and *assets* is a directory containing applications assets, which can be retrieved by *AssetManager*. The `AndroidManifest.xml` is a key file within application structure, which is an additional Android manifest file, describing the name, version, access rights, and referenced library files for the application [19].

2.2.2 App Components

There are four different types of app components [20]. Each component, serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. Figure 3 shows the Android app components and related interactions.

Activities : Activity is an individual user interface screen in an Android Application. Android activity is where visual elements called Views (also known as widgets) can be placed and user can perform various actions by interacting with it.

Services : Service is the Android way of keeping an operation going on in the background. Services are used to perform the processing parts of your application in the background. Services are typically used for processes that take a significant period of time such as playing music, downloading data or uploading photos.

Content Provider : A content provider is a component for managing a data set. Content providers in Android provides a flexible way to make data available across applications. A simple example of content provider is the Contacts Manager app. You can get contacts in multiple applications such as your SMS application, Dialer application, etc.

Broadcasting : Broadcast receivers is one of Android application components that is used to receive messages that are broadcasted by the Android system or other Android applications. Examples of broadcasts initiated by the system are battery low, network state changed, phone starts and photo captured from camera.

3 Android Security Mechanisms

Android is a modern mobile platform that was designed to be open source and free. Android applications make use of advanced hardware and software, as well as local and served data, exposed through the platform to bring innovation and value to consumers. To protect that value, the platform must offer an application environment that ensures the security of users, data, applications, the device, and the network. Securing an open platform requires a robust security architecture and rigorous security programs. In this section we discuss mechanisms that Android uses to make the application environment secure [21].

3.1 Android Permission Framework

A basic Android application has no permissions associated with it by default to get access to the resources. Before installing an application, the current version of Android OS displays all required permissions by the application. The requested permissions is to enforce restrictions on the resources of the

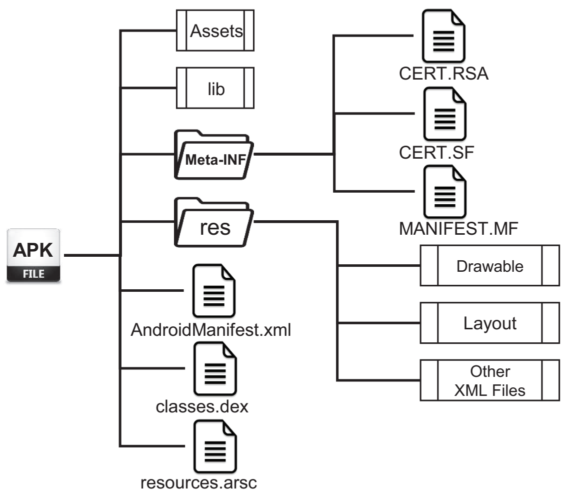

Figure 2: Android APK file structure

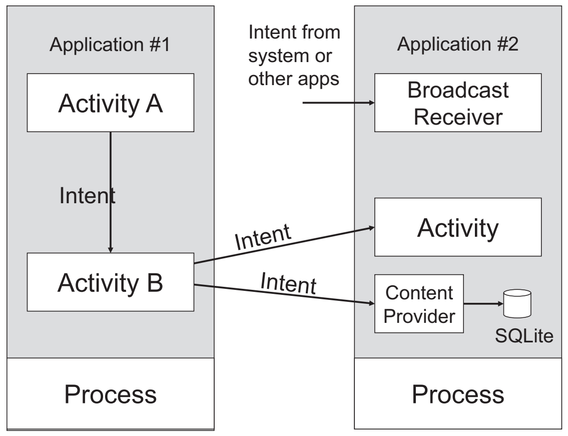

Figure 3: Android components and their interactions

devices such as Internet connections, SMS, Storage, and Camera, etc. After reviewing these permissions, the user can choose to accept or refuse them, installing the application only if they accept [21]. There are four classes of permissions: *Normal*, *Dangerous*, *Signature* and *SignatureOrSystem*. In this subsection, we explain all types of Android permissions [22].

Normal : A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user.

Dangerous : A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user.

Signature : A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission.

SignatureOrSystem : A permission that the system grants only to applications that are in the Android system image or that are signed with the same certificate as the application that declared the permission.

3.2 Application Sandboxing

Application sandboxing, also called application containerization, is an approach to software development and Mobile Application Management (MAM) that limits the environments in which certain code can execute. Android applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. In order to protect the application's data from unauthorized access, Android kernel implements the Linux Discretionary Access Control (DAC) to manage and protect the device's resources to be misused. Each app process is protected with an assigned unique ID (UID) within an isolated sandbox [23].

3.3 Inter-Component Communication (ICC)

Although each application executes within a dedicated sandbox, Android allows applications to communicate with each other through a well-defined Inter-Component Communication (ICC) mechanism or *Binder*. Android middleware mediates the ICC between application's components. The Binder or ICC takes care of migrating the execution of a request from the requester to the target process transparently to the applications. Applications can call the components or services of other applications as service [21].

4 Android Security Issues and Threats

Android security is built upon a permission-based mechanism which regulates the access of third-party Android applications to critical resources on an Android device. Such permission-based mechanism is widely criticized for its coarse-grained control of application permissions and the inefficient permission management, by developers, marketers, and end-users. For example, users can either accept all permission requests from an app to install it, or not to install the app. This type of permission management is proved to be undesirable for the devices security. In this section, we discuss the main security issues of the Android, which leads to user information leakage and puts the user's privacy in jeopardy [21].

4.1 Information leakage

In current Android architecture design, apps are restricted from accessing resources or other apps unless it is authorized by the users. Users have to grant all resource access requests before installing and using an app. Information leakage occurs when users grant resources without any restriction from OS. However, Android's permission control mechanism has been proven ineffective to protect user's privacy and resource from malicious apps. Studies showed that more than 70% of smart phone apps request to collect data irrelevant to the main function of the app [24][25]. With more than 1.4 million available apps in Google Play, and a great number of apps from miscellaneous third-party markets, a significant number of malicious apps have been exposed to Android users for installation. However, when installing a new app, only a small portion of users pay attention to the resource being requested, since they tend to rush through prompted permission request screens to get to use the application. Only a small portion (3%) of users are cautious and make correct answers to permission granting questions. In addition, the current Android permission warnings do not help most users make correct security decisions [26]. The ”blaming the users” approach has become a failure to protect Android users.

As pointed out in [26, 27], the reasons for the ineffectiveness of the current permission control system include: (1) inexperienced users do not realize resource requests are irrelevant and will compromise their privacy, (2) users have the urge to use the app and may be obliged to exchange their privacy for using the app.

4.2 Privilege escalation

Privilege escalation or permission escalation attacks were leveraged by exploiting publicly available Android kernel vulnerabilities to gain elevated access to resources that are normally protected from an application or user. This type of attack can result in unauthorized actions from applications with more privileges than intended, which causes many sensitive information leakage. Android exported components can be exploited to gain access to the critical permissions [28].

4.3 Repackaging Apps

Repackaging is one of the most important and common security issues of the Android OS. Repackaging is the process of disassembling/decompiling of .apk files using reverse-engineering techniques and adding (injecting) malicious code into the main source code. Repackaging techniques that can be used on the Android platform allow malicious code to be disguised as a normal app. It is difficult to distinguish between a repackaged malicious code and a normal app because the repackaged app usually appears to function in the same way as the legitimate one. The repackaging steps are as follows [29, 30]:

Unpacking : unpacking APK files using available tools such as *apktool*, which is a tool based on reverse-engineering.

Decompiling : decompiling the Java source code using JAD and extracting the source code of Java classes.

Code injection : injecting code and adding resources into the main source code using Java developing environments.

Repacking : rebuilding the files using *apktool* and signing the generated files using *jarsigner*.

Geimini and KungFu are examples of trojans which are based on APK repackaging. These trojans can be bundled into many valid Android apps.

4.4 Denial of Service (DoS) attack

The increasing number of smartphone users and prevalence of mobile devices (phones, tablets) which are connected to the Internet can be a platform for growth of DoS attacks. Since the majority of smartphones are not equipped with the same protections (i.e. anti-virus programs) as PCs, malicious apps find it as a proper platform for DoS attacks. Overusing limited CPU, memory, network bandwidth and battery power are the main goals of DoS attacks [31].

4.5 Colluding

Colluding threat is a client-side attack. In this attack, users install a set of apps developed by the same developer and same certificate and grant different types of permissions including sensitive and non-sensitive. After installing apps, these apps can take advantage of a shared UID and get access to all their permissions and resources [32].

5 Proposed Solutions

Considering the security issues that we described in section 4, so far there have been proposed many studies towards the principles and practices to manage resource usage [10][11][12][33][34][35] [36][37] and security solutions to address these vulnerabilities. The existing security and privacy solutions are classified into three categories. We explain the categories in more details in related sections. Since proposed works that we cover in this survey, use different tools and techniques to implement their solutions. Before going into further details, in this section we described the main categories and all existing applied techniques.

5.1 Existing techniques and mechanisms

5.1.1 Static Analysis

Those works that use static analysis approach [38][30] are based on the application's structure and code [39]. In this section we describe the main techniques of static analysis.

- **Application Signature** As we described before, any Android application has a unique signature. Signature-based solutions check the contents or patterns of an application against a dictionary of malware signatures. If they find a matching, they can take action. This method is somewhat limited by the fact that it can only identify a limited amount of emerging threats, e.g. generic, or extremely broad, signatures.

- **Permission Analysis** This mechanism works based on granted permissions to the applications. They assess the risk of the granted permissions and the sensitivity of the resources. Depending on the risk level, they analyze and detect the malicious apps.

- **Control Flow Analysis** In this type of static analysis techniques, it needs to extract apps' Control Flow Graph (CFG) and look for existing possible resource misusing and vulnerabilities within the application's code. Based on the discovered threats and vulnerabilities, they make a decision on maliciousness or vulnerability of the application.

5.1.2 Dynamic Analysis

Existing works that use dynamic analysis [40][41][42][43][44][45][46] mainly work based on application behavior analysis during the runtime process. There are three main parameters that can be considered as application's behavior and activities: system calls, battery consumption, and network usage [47].

5.1.3 Crowdsourcing

Crowdsourcing is the process of obtaining needed services, ideas, or content by soliciting contributions from a large group of people [48]. In Android OS security scope, it is defined as a process in which we collect data from users or devices toward improving the security of devices and privacy preserving. For example, it can be collecting the system call log of a device or users' reviews on an app.

5.1.4 Policy-based

Using this technique, solutions require users to define several policies on the prepared services in order to customize the service. For example, in smartphone security area, the policies can be the level of permission granting restriction to applications.

5.1.5 Recommendation-based

In this approach, they help inexperienced users through providing recommendations on challenging app security and privacy decisions such as granting permissions to apps and restricting apps' resource accesses.

5.2 Taxonomy of existing solution

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 category: (1) Prevention-based, (2) Analysis-based and (3) Runtime Monitoring.

5.3 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 application behavior, or inject malicious code) [49]. Based on a new research study done by ARXAN [50], 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).

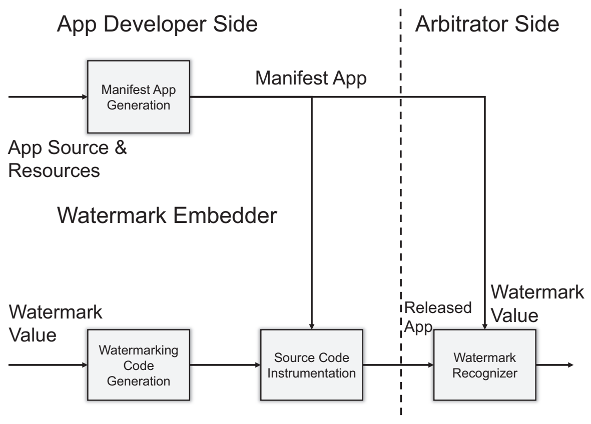

Figure 4: Overall AppInk architecture

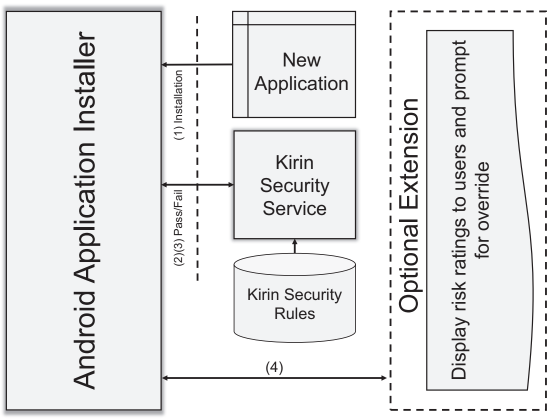

Figure 5: Kirin based software installer flow and its components

5.3.1 Kirin

Mitigating malicious apps at install time using certification process on apps is the main goal of Kirin [46]. 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 5 shows the Kirin based software installer flow and its components.

5.3.2 AppInk

In order to mitigate app repackaging, Zhou et al. [51] 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 4 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 [52, 53] and the results indicate that AppInk is effective in defending against common automatic repackaging attacks.

5.4 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 [54]. 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.

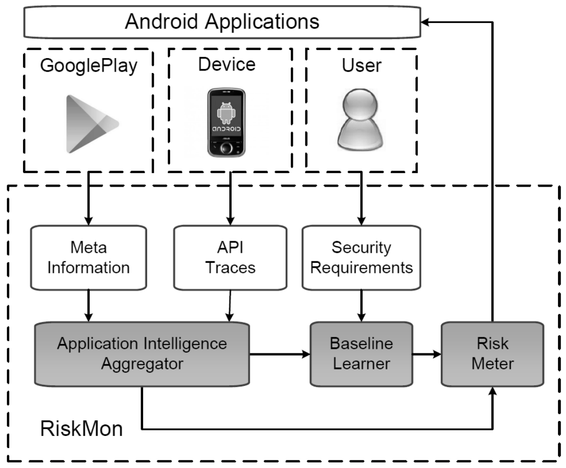

Figure 6: RiskMon architecture

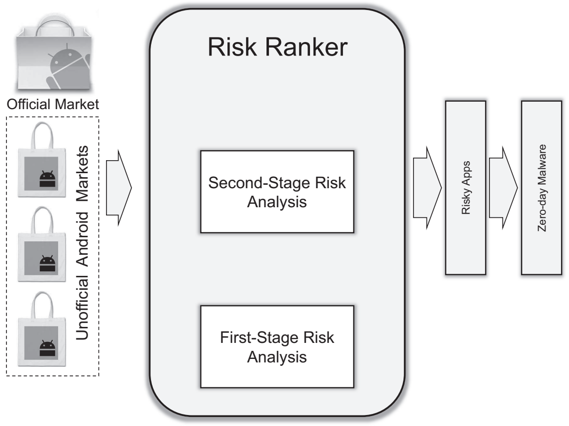
Official Market
Unofficial Android Markets
Risk Ranker
Second-Stage Risk Analysis
First-Stage Risk Analysis
Risky Apps
Zero-day Malware

Figure 7: RiskRanker architecture

5.4.1 RiskMon

RiskMon [44] 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 6 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.

5.4.2 RiskRanker

RiskRanker [55], is a proactive scheme to spot zero-day Android malware [56]. 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 nonobfuscated executions of apps that invoke (i) launching root exploits, (ii) illegal cost creation, and (iii) privacy violation attacks. In the second stage 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 7 shows the RiskRanker's 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.

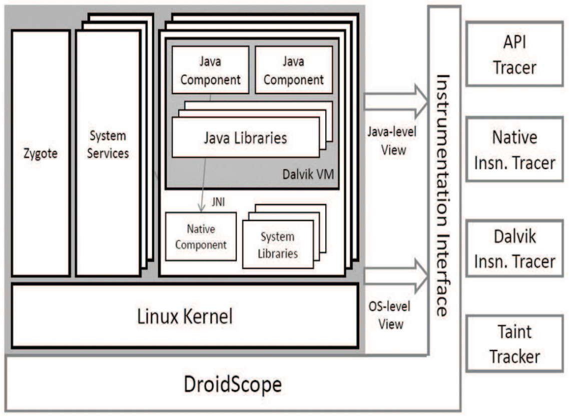

Figure 8: DroidScope's architecture and its instrumentation interface

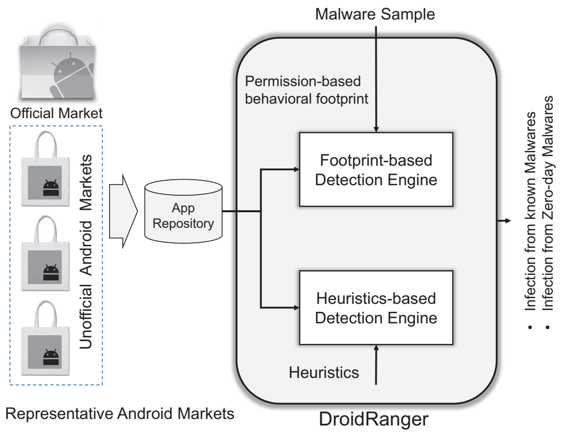
Malware Sample
Official Market
Unofficial Android Markets
App Repository
Permission-based behavioral footprint
Footprint-based Detection Engine
Heuristics-based Detection Engine
Heuristics
Infection from known Malwares
Infection from Zero-day Malwares
Representative Android Markets
DroidRanger

Figure 9: DroidRanger architecture

5.4.3 DroidScope

Lok et al. present DroidScope [57], 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 [58] 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 8 shows the DroidScope's architecture and its instrumentation interface.

5.4.4 DroidRanger

In this work [30], 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 9.

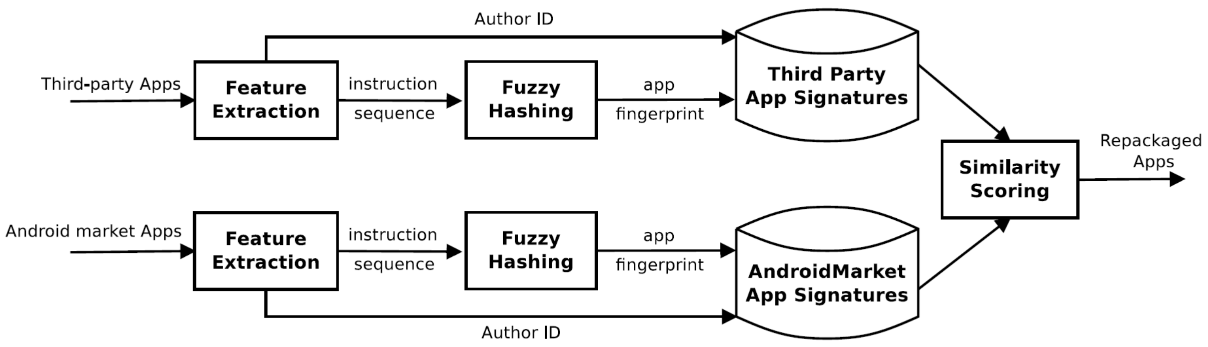

Figure 10: DroidMOSS overview

5.4.5 DroidMOSS

In this work, an application similarity measurement system called DroidMOSS [30] is proposed that applies a fuzzy hashing technique [59][60] 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 5.4.5.

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.

5.4.6 WHYPER

Pandita et al. propose WHYPER [61] 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 11 depicts an overview of WHYPER including its related components.

5.4.7 PScout

PScout [62] 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 [63], PScout is able to extract more permission specification. In the reported experimentation, they use PScout to analyze 4 versions of Android spanning

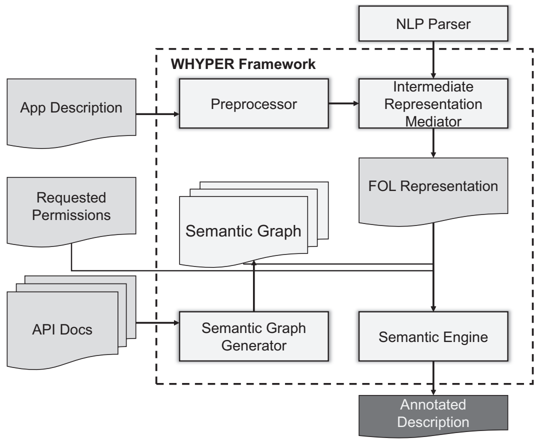

Figure 11: Overview of WHYPER

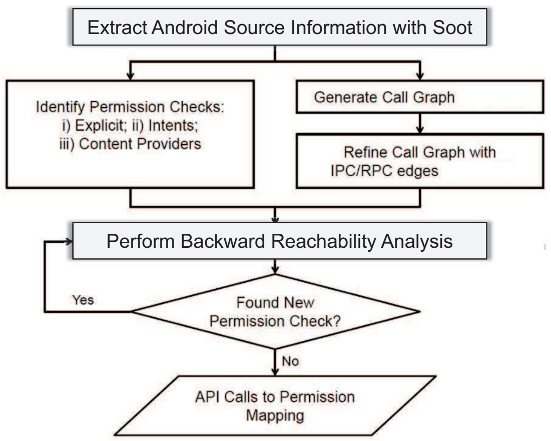
Extract Android Source Information with Soot
Identify Permission Checks: i) Explicit; ii) Intents; iii) Content Providers
Generate Call Graph
Refine Call Graph with IPC/RPC edges
Perform Backward Reachability Analysis
Yes
Found New Permission Check?
No
API Calls to Permission Mapping

Figure 12: PScout architecture

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 12 shows PScout architecture.

5.4.8 AndroSimilar

In [38] 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 [64] 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 pp.

5.4.9 ComDroid

ComDroid [65] 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 [66]. 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 [67], Amandroid [68], AppsPlayGround [69], ScanDroid [70], VetDroid [71], Pegasus [72], AppIntent [73], Mobile-Sandbox [74], PiggyApp [75], AnDarwin [76], Juxtapp [77], Stowaway [63], DNADroid [78], Androguard [79], APKInspector [80], JEB [81], Andrubis [82], AndroTotal [83], RobotDroid [84], CHEX [85], Androwarn [86], MAdFraud [87], DECAF [88], DroidChecker [89], MARVIN [90], Shinichi et al. [91], and ProtectMyPrivacy [92]. These works all use static and dynamic analysis tools to detect apps' vulnerabilities and detect malicious apps.

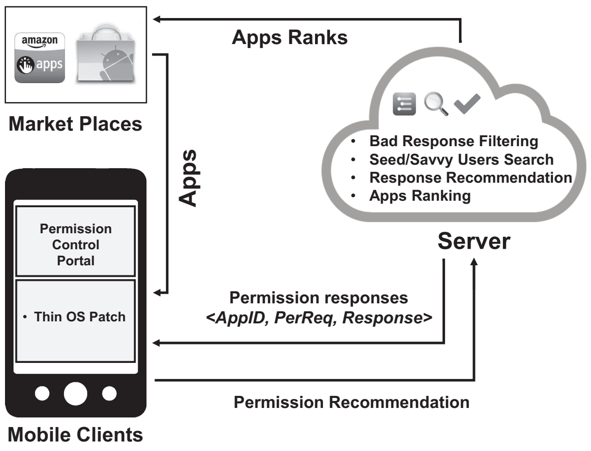

Figure 13: RecDroid Components and Architecture

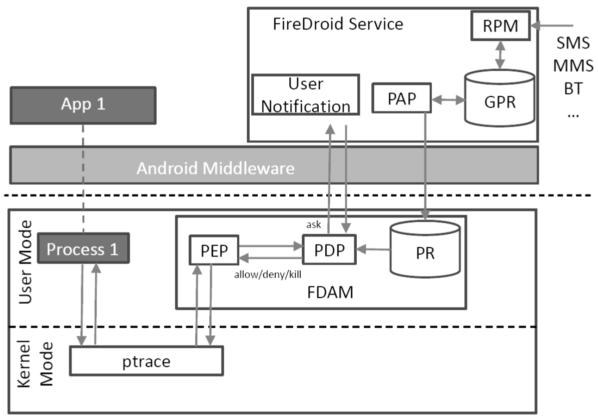

Figure 14: FireDroid Components and Architecture

5.5 Runtime Monitoring

As we described before, at the middleware level, each application is sandboxed, i.e., it is running in its own instance of Dalvik VM, and interaction and sharing between apps are allowed only through an inter-process communication (IPC) mechanism. Android middleware provides a list of resources and services such as sending SMS, access to contacts, or internet access. Android enforces access control to these services via its permission mechanism. In Android permission mechanism, each service/resource is associated with a certain unique permission tag, and each app must request permissions to the services it needs at installation time. Everytime an app requests access to a specific service/resource, Android runtime security monitor checks whether the app has the required permission tags for that particular service/resource. In addition to privilege escalation protection, information leakage can be monitored too.

In this section, we describe those works which are based on monitoring apps' activities and permissions accesses. Proposed works in this category continuously run on a device to either prevent, detect malicious activity, or enforce fine-grained policy. It is worthy to note that policies can be defined either by users or systems.

5.5.1 RecDroid

RecDroid [40][93][94] is a resource access permission control framework through crowdsourcing. RecDroid tries to assist users to make a right decision as for whether a permission request should be accepted. RecDroid is a crowdsourcing recommendation framework that collects apps' permission requests and users' permission responses, from which a ranking algorithm is used to evaluate the expertise level of users and a voting algorithm is used to compute an appropriate response to the permission request (accept or reject). To bootstrap the recommendation system, RecDroid relies on a small set of seed expert users that could make reliable recommendations for a small set of applications. RecDroid also uses a game-theoretic Bayesian model to detect the malicious users and ignore their responses [95][96].

Figure 13 illustrates RecDroid's architecture and its components. First, the framework allows users to use apps without giving all permissions and receive help from expert users when permission requests appear. RecDroid allows users to install untrusted apps under a *"probation"* mode, while the trusted ones are installed in normal *"trusted"* mode. In probation mode, users make realtime resource granting decisions when apps are running. The framework also facilitate user-help-user environment, where expert users' decisions are recommended to inexperienced users. The framework provides the following functionalities:

- Two app installation modes for apps that are about to be installed: trusted mode and probation mode. In probation mode, at run time, an app has to request permission from users to access sensitive resources (e.g. GPS traces, contact information, friend list) when the resource is needed. In trusted mode, the app is fully trusted and all permissions are all granted.
- An architecture to intercept and collect apps' permission requests and responses, from which recommendations are made as for what permission from which apps should and should not be granted.
- A recommendation system to guide users with permission granting decisions, by serving users with recommendations from expert users on the same apps.
- A user-based ranking algorithm to rank security risks of mobile apps.

5.5.2 FireDroid

FireDroid [43], is a policy-based framework for enforcing security policies by interleaving process system calls. In FireDroid, an application monitor is created to track all processes spawned in Android and allow/deny them based on human managed policies. FireDroid identifies at runtime if an application is executing illicit or potentially dangerous actions by intercepting the system calls the application executes. No matter if the malware is new or a repackaged version of an existing one: when the malware executes dangerous system calls, FireDroid can detect and enforce the appropriate security policies.

In FireDroid, they use the *ptrace()* function to monitor the applications' behavior at runtime without modifying the application or Android framework. Figure 14 show the architecture of the FireDroid and its existing components.

The main advantage of FireDroid is that it is completely transparent to the applications as well as to the Android OS. This fact means that users are not involved with a heavy interaction. Users only need to define the policies for the applications and load them into the system through the FireDroid provided policy portal.

5.5.3 MockDroid

The aim of the MockDroid [41] is avoiding of giving out sensitive data by granting fake permission and allowing the user to provide fake or 'mock' data to applications interactively as the application is being used. MockDroid allows users to revoke access to particular resources at execution time and encourage them to consider the trade-off between functionality and the disclosure of personal information whilst they use an application. For example, providing "no location fix available" for GPS location access could be a way to protect from sensitive data disclosure.

While such approaches reduce the risk of leaking private information and critical resources, MockDroid requires users to make decisions on every resource permission request, which is difficult for inexperienced users and time consuming for others.

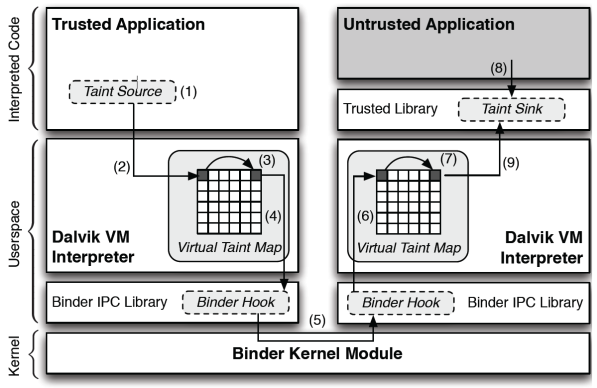

Figure 15: TaintDroid Components and Architecture

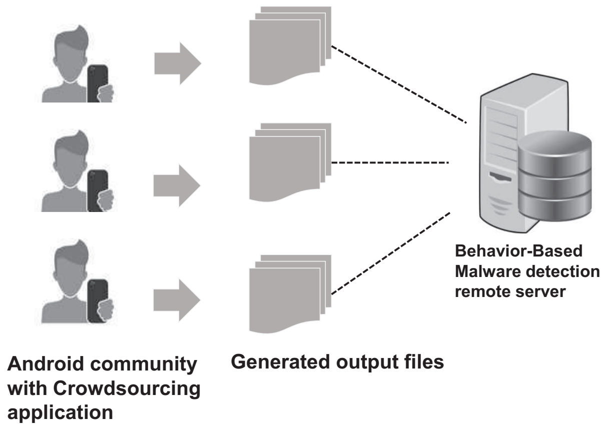
Android community with Crowdsourcing application
Generated output files
Behavior-Based Malware detection remote server

Figure 16: Crowdroid Architecture

5.5.4 Crowdroid

Crowdroid [45] is a behavior based malware detection system. In Crowdroid, they detect anomalously behaving applications through a crowdsourcing framework. The authors propose a framework to analyze the behavior of Android applications to distinguish between applications that have the same names and versions, but behave differently.

As Figure 16 shows, it has two components, a lightweight client application which needs to be installed on users' devices and a remote malware detection server. The application records the behavior of the installed applications such as system calls and send them as a log file to the centralized remote server. The application records the system calls through a system utility called *Strace*. The log file consists of the device information, list of installed applications, and behavioral data. On the server side, the remote server will be in charge of parsing data and creating a system call vector per interaction for users within their applications. Finally, the collected data will be clustered by 2-means partition clustering to detect whether the applications are benign or malicious.

Since the client applications must always be available, draining the available device resources is the main limitation of Crowdroid. The main drawback of this work is that they transfer the data on FTP protocol, which is not safe.

5.5.5 TaintDroid

TaintDroid [97] is a data flow tracking system, which allows users to track and analyze flows of sensitive data and potentially identify suspicious apps. TainDroid provides a tool for expert users to discover misbehaving from potential apps. The sensitive data is automatically tainted in order to keep track whether the labeled data leaves the device. When the labeled data leaves the device, TaintDroid records the label of the data and the app which sent the data along with its destination address.

TaintDroid uses fine-grained labels Variable-level, Method-level, File-level, and Message-level. Using Variable-level semantics provided by the interpreter provides valuable context for avoiding the taint explosion observed in the x86 instruction set. TaintDroid uses Message-level tracking between applications' messages in order to minimize IPC overhead while extending the analysis systemwide. Method-level tracking is used for Android native libraries that are not directly accessible to apps but through modified firmware. Finally, TaintDroid uses File-level tracking to ensure persistent information conservatively retains its taint markings. The above scenario is displayed in Figure 15.

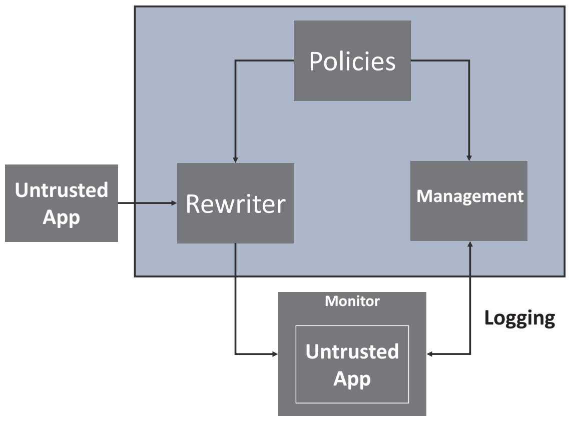

Figure 17: AppGuard architecture

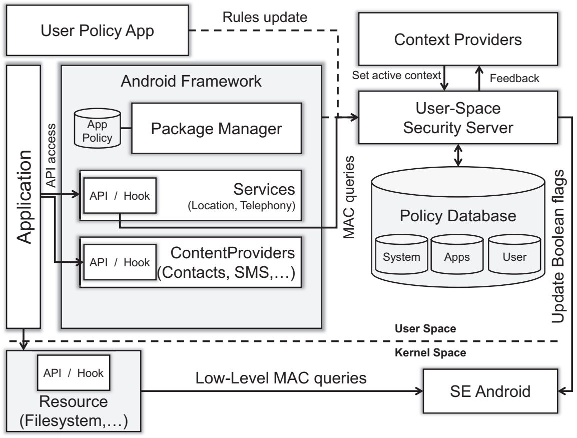

Figure 18: Overview of FlaskDroid

5.5.6 AppGuard

AppGuard [42], is a flexible system for the enforcement of user-customizable security policies on untrusted Android apps. Indeed, AppGuard works based on Inline Reference Monitoring (IRM) [98], and helps users to enforce pre-defined user policy on third'party apps. AppGuard, not only restricts the outreach of the third-party apps but also the operating system.

Figure 17 shows the overall view of the AppGuard. First, this solution modifies the APK packages of third-party apps in a way that it invokes a security monitor before each security-relevant program operation at runtime. In fact, it calls the security call before each function call to the Android system libraries. Second, the security monitor checks whether currently enforced security policies allows the system call from the application, and then depends on the pre-defined policy by user grant the permission to execute or deny the requested permission and call a function to return a dummy data as result.

In order to define the policies by user, AppGuard provides user with a standalone application, which is installed on the user device. Since AppGuard only modifies the application package and not the operating system, it allows for enforcing policies without rooting phones or modifying the operating system, which is the main advantage of this solution.

5.5.7 FlaskDroid

FlaskDroid [99] is proposed as a generic security architecture, in order to provide Mandatory Access Control (MAC) simultaneously on both Android's middleware and kernel layers. FlaskDroid support as a flexible and effective ecosystem to instantiate different security solutions and multiple fine-grained security policies. In order to extract operational semantics at the Android middleware, they design a policy language inspired by SELinux [100] tailored to the specifics of Android's middleware semantics.

They evaluated the flexibility of the FlaskDroid by policy driven instantiations of selected security models such as the existing work Saint [101] as well as a new privacy protecting, user-defined and fine-grained per-app access control model. They also evaluated the efficiency and effectiveness of the work on SE Android 4.0.4 [102]. Figure 18 illustrates FlaskDroid architecture and its components.

5.5.8 Porscha

In order to protect DRM-based contents (e.g., MP3-based MMS, SMS, or email), Ongtang et al. propose Porscha [103] as a system, developed in content proxies and reference monitors within the Android middleware to enforce DRM policies embedded in received content. The main goal of Porscha is improving the DRM policy enforcement mechanism to ensure: (1) authorized access to protected content (2) content

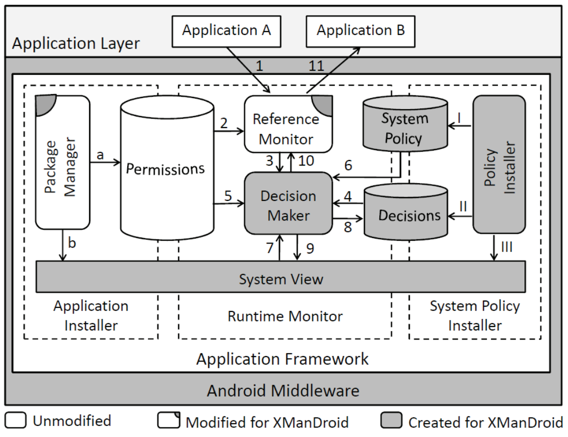

Figure 19: XManDroid architecture

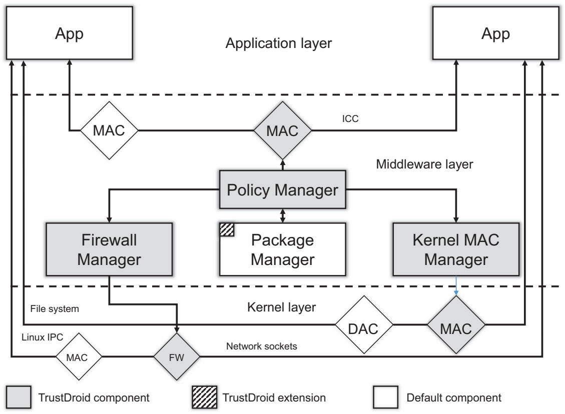

Figure 20: TrustDroid architecture

accessibility by provider-endorsed applications, and (3) ability to access contents under policy-defined contextual constraints, e.g., time limitation, a maximum number of viewings, etc.

In Porscha, policy enforcing is a two-stage process: protection of transmitted content to the device, and the regulation of content use on the phone. For the first stage, Porscha binds policy and ensures content confidentiality ”on the wire” using constructions and infrastructure built on Identity-Based Encryption [104]. For the second stage, Porscha enforces policies by proxying content channels (e.g., POP3, IMAP, Active Sync) and placing reference monitor hooks within Android’s *Binder* IPC framework. They evaluated the Porscha using the three most popular content types: SMS messages, MMS messages, and email. In the reported experimentation, Porscha has low content delivery latency less than 1 second.

5.5.9 QUIRE

In order to deal with ICC-based attacks and regulating interapp ICC, Dietz et al. proposed QUIRE [105], a security mechanism, which is based on a call-chain tracking technique that provides important context in the form of provenance and OS managed data security to local and remote apps communicating by IPC and RPC respectively.

QUIRE is developed based on two main techniques. The first technique, which is based on tracking the call chain of IPCs and annotating IPCs occurring within the phone such that the recipient of an IPC request can observe the full call chain associated with the request. Second, QUIRE uses simple cryptographic mechanisms to protect data moving over IPC and RPC channels. This way, QUIRE enables applications to propagate call chain context to downstream callees and to authenticate the origin of data that they receive indirectly.

They evaluated QUIRE through performing a set of experiments were on the standard Android developer phone using two self-developed apps. In the reported experimentation the overhead of QUIRE for network RPC is practically insignificant.

5.5.10 XManDroid

In order to privilege escalation attacks Bugiel et al. presented XManDroid [106]. XManDroid is a security framework that extends the monitoring mechanism of Android OS to detect and prevent application-level privilege escalation attacks at runtime based on a system-centric system policy.

XManDroid monitors all interactions between apps and dynamically analyzes applications’ transitive permission usage. Monitored communication links should pass a verification process against a set of policy rules. Finally, depending on predefined policies, system representation allows for an effective de-

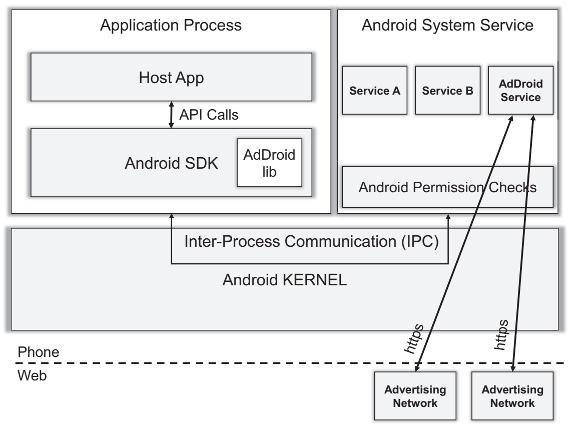

Figure 21: AdDroid design

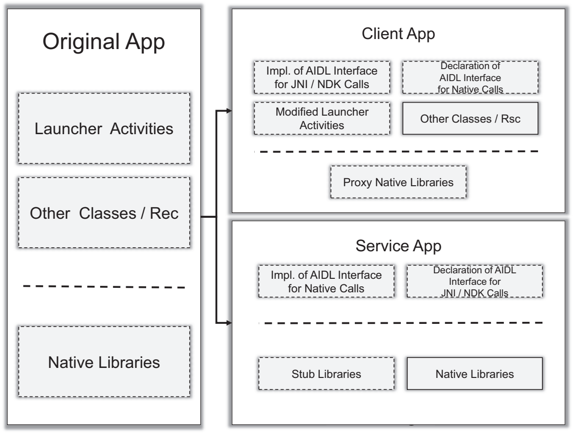
Original App
Launcher Activities
Other Classes / Rec
Native Libraries
Client App
Impl. of AIDL Interface for JNI / NDK Calls
Declaration of AIDL Interface for Native Calls
Modified Launcher Activities
Other Classes / Rsc
Proxy Native Libraries
Service App
Impl. of AIDL Interface for Native Calls
Declaration of AIDL Interface for JNI / NDK Calls
Stub Libraries
Native Libraries

Figure 22: NativeGuard architecture

tection of (covert) channels established through the Android system services and content providers while simultaneously optimizing the rate of false positives. XManDroid architecture is shown in figure 19.

They evaluated the effectiveness of XManDroid on our test suite that simulates known application-level privilege escalation attacks (including Soundcomber), and demonstrate successful detection of attacks that use Android's inter-component communication (ICC) framework (standard for most attacks).

5.5.11 TrustDroid

Bugiel et al. proposed TrustDroid [107], a security architecture, that enables lightweight domain isolation on each layer of the Android software stack. In the other words, TrustDroid tries to isolate data and applications of multiple trust levels in a lightweight way. Here, each domain stands for a security level of Android OS including kernel layer, middleware layer and network layer.

In order to address the three domain isolations mentioned above, they modify three layers of the OS: extending the Android middleware and the underlying Linux kernel to mediate IPC apps associated to different domains, modifying the standard Android kernel firewall to filter network traffic using netfilter and managing the file system through extending the current Android Linux kernel with TOMOYO Linux based Mandatory Access Control [108] and corresponding TOMOYO policies. They evaluated TrustDroid on an Android devices (Nexus One) and in the reported results, TrustDroid performance has a low overhead, and a low butter consumption. Figure 20 depicts TrustDroid architecture.

5.5.12 AdDroid

AdDroid [109] is proposed to manage and to separate privileged advertising functionality from host applications in a way that applications show advertisements without requesting privacy sensitive permissions. AdDroid modifies the Android OS and tries to separate host application and the core advertising code run in separate protection domains. It also introduces a new advertising API and corresponding advertising permissions for the Android platform. It is worthy to note that, AdDroid introduces two new Android permissions, namely `ADVERTISING` and `LOCATION_ADVERTISING`. The AdDroid design and its key components are presented in Figure 21.

AdDroid contains a new Android privilege separation service in order to manage advertising privileges. In this way developers can incorporate advertisements into their application using the predefined AdDroid advertising APIs. Therefore, apps are able to display advertisements without granting privacy-sensitive permissions. In the reported experimentation, using AdDroid, 27% of advertising-supported applications do not need Internet access, 25% do not need location information, and 8% do not re-

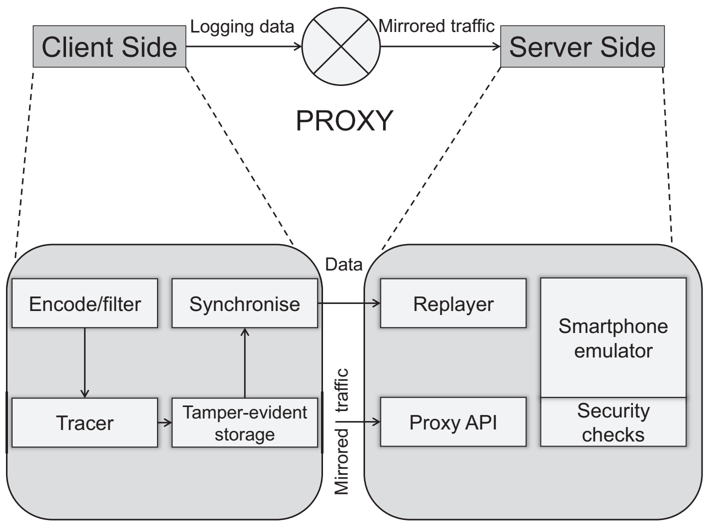

Figure 23: Paranoid Android overview

quest for phone state information. They report that AdDroid would reduce over-privileging in 46% of advertising-supported applications.

5.5.13 NativeGuard

In NativeGuard [110], Sun et al. present a framework that utilizes Android's process isolation to sandbox native libraries of an application. NativeGuard leverage the process-based protection in Android through isolating native libraries from other components in Android applications. Architecture of NativeGuard is shown in Figure 22.

NativeGuard isolation process has two main aspects. First, it separates the native libraries within an Android application and a standalone application, where native code is not granted to have full access to the entire application address space and the interactions between the native libraries and Java code is fulfilled via IPC mechanism. Second, after the isolation process the native libraries are not granted the permissions which are granted to the app at the installation process. In this way, NativeGuard decreases the chance of permission misusing and over-privileging attacks.

5.5.14 Paranoid Android

Paranoid Android [111] is a security check system which is applied on a remote security server (cloud-based detection framework) that host exact replicas of the phones in virtual environments. Moving the checking process from the user device to a remote server is the main novelty of this work. It is worthy to note that the main reason for doing the security checks on a remote server is because of lack of enough computational resources and battery consumption on smartphones. Figure 5.5.13 shows the Paranoid Android architecture overview.

The security check mechanism that PA follows is a two-stage process. The first stage, which should be done on-device, is monitoring the apps' activities and collecting and transferring logs to the server. In order to reduce the overhead of log transfer, it sends the log only if the device is awake and connected to the Internet. The second stage is analyzing the collected logs from devices. PA uses a ClamAV based antivirus [112] to scan files. In addition, PA performs a taint analysis to detect memory corruption attacks. They evaluate the system and it is reported that it is both practical and scalable: generating no more than 2KiBps and 64Bps of trace log for high-loads and idle operation respectively, and supporting

Table 1: Comparison results of the proposed solutions for addressing the permission management on Android OS

Solution	Objective				Mechanisms							Deployment			Tech. Properties		
	Prevention	Assessment	Analysis	Detection	Static	Dynamic	Behavioral	System Call	Recommendation	Crowdsourcing	Policy-based	On-Device	Off-Device	Distributed	OS Modification	Availability	Tools
RecDroid [40][93][94]		✓				✓	✓	✓	✓	✓	✓			✓	✓	✓	✓
FireDroid [43]				✓		✓		✓			✓	✓			✓		
MockDroid [41]				✓		✓		✓			✓	✓			✓	✓	
Crowdroid [45]		✓		✓		✓	✓	✓		✓				✓	✓		strace
RiskMon [44]		✓	✓			✓	✓	✓		✓			✓		✓		
TaintDroid [97]				✓		✓	✓	✓					✓		✓	✓	
AppGuard [42]			✓			✓		✓			✓	✓			✓	✓	
AndroSimilar [38]				✓	✓								✓			✓	
DroidMOSS [30]				✓	✓								✓				keytool
AppInk [51]	✓					✓							✓				
RiskRanker [55]			✓		✓			✓					✓				
DroidScope [57]			✓			✓	✓	✓					✓		✓	✓	QEMU
DroidRanger [30]			✓	✓	✓	✓		✓					✓				
Kirin [46]	✓	✓			✓							✓			✓	✓	
WHYPER [61]		✓			✓								✓			✓	NLP
Pscout [62]			✓		✓			✓					✓				soot
FlaskDroid [99]				✓		✓		✓			✓			✓	✓	✓	
ComDroid [65]			✓		✓			✓					✓				dedexer
Porscha [103]			✓			✓		✓			✓	✓			✓		
QUIRE [105]			✓		✓			✓					✓		✓		
XManDroid [106]	✓			✓		✓	✓	✓				✓			✓	✓	
TrustDroid [107]	✓					✓		✓				✓			✓		
Paranoid Android [111]				✓		✓	✓						✓		✓		ClamAV
AdDroid [109]				✓	✓			✓				✓			✓		
NativeGuard [110]	✓					✓		✓				✓			✓		apktool

more than a hundred replicas running on a single remote server.

In addition to described works in this section, there are many other related works: BayesDroid [113], TISSA [114], AppFence [115], LP-Gaurdian [116], Andromaly [117], AirBag [118], AdSplit [119], AFrame [120], LayerCake [121], CRePE [122], Apex [123], ASM [124], Aurasium [125], Dr. Android and Mr. Hide [126], Saint[101], IPC Inspection [127], MalloDroid [128], L4Android [129], AppSealer [130], Pyandrazzi [131], Morbs [132], PatchDroid [133], CryptoLint [134], IccTA [135], DroidBox [136], Drozer [137], CooperDroid [138], DroidPAD [139], DroidTracker [140], ProtectMyPrivacy [92], Ismail et al. [141]. These works all capture apps' activities and system calls, and using a set of predefined policies to restrict apps' activities.

6 Comparison and Discussion

In this section we have a comparison on the covered solutions in this survey. As we described before, depending on the objective of them, they can be categorized into *Assessment*, *Analysis*, or *Detection* categories. In addition, based on the used techniques and methodologies, we can classify them into *crowdsourcing-based*, *policy-based*, *dynamic*, *static*, and *recommendation-based* classes. Considering the way that they implement the proposed solution, they can modify the framework or not. Table I shows the comparison results for all the covered works in this survey.

In this section we describe following four important aspects that can be concluded from the comparison table and covered works:

First, most of the dynamic-based solutions use crowdsourcing and policy-based techniques. Since dynamic-based solutions try to assess, analyze, or detect malicious apps based on the behavior of the apps, it is not possible to make an exact decision on maliciousness of the apps. Although, some the proposed solutions make decisions on resource access requests with high level of accuracy, most of them involve users to make their decisions more accurate. Making users responsible for their devices and granted permissions is another advantage of the involving users in defining policies.

Second, as we described before, the way of implementation for proposed works are different. They can be implemented on the application level or framework and Linux kernel level. As you can see in the Table I, most of the dynamic solutions modify the framework and Linux kernel. The need to modify the low level of Android OS comes from this fact that dynamic solutions are based on the apps' activities. Therefore, the only way to monitor the applications' activities such as system calls is modifying the kernel or the framework.

Third, rooting the device and implementing the solution in application level is another way of implementation. In this case, they root the device and manage resource access requests through an application service. The main problem of this type of implementation is that in fact they make the device unsafe.

Fourth, the only solution, which is based on crowdsourcing, and recommendation is RecDroid [40]. As we mentioned before, the missing key in the policy-based solutions is that none of them helps users to make a decision on the resource requests. RecDroid is the only work that assists users through making safe recommendations on granting the permissions to the applications.

7 Conclusion

Along with the increasing prevalence of Android smartphones, the number of Android apps including malware is increasing daily. In spite of deployed Android security mechanisms, malware take advantage of the Android security holes to misuse the granted resources. Thereby, many efforts have been proposed to restrict the outreach of vulnerabilities in Android devices. In this survey we investigated the current proposed works in two static and dynamic groups. The proposed works are primarily behavior-based and their main contribution is tracing the apps' system calls and analyzing the activities to restrict them from high risk activities. After reviewing these works we came up with two questions that proposed works are not capable of answering appropriately. First, *are those behaviors necessarily inappropriate?*. Second, *can we label the apps as malware or benign based on the behavior?*.

References

[1] Gartner, "Gartner: 1.1 billion android smartphones, tablets expected to ship in 2014," Online; accessed at June 5, 2015, http://tinyurl.com/n8t3h9y.

[2] Victor, “Android’s google play beats app store with over 1 million apps, now officially largest,” Online; accessed at May 12, 2013, http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680.

[3] “Number of available applications in the google play store,” 2015, http://www.statista.com/.

[4] W. Rothman, “Smart phone malware: The six worst offenders,” Online; accessed at April 17, 2015, http://www.nbcnews.com/tech/mobile/smart-phone-malware-six-worst-offenders-f125248.

[5] “Bit9 report: Pausing google play: More than 100,000 android apps may pose security risks,” 2012, https://www.bit9.com/files/1/Pausing-Google-Play-October2012.pdf.

[6] “Number of android applications,” Online; accessed at August 7, 2015, http://www.appbrain.com/stats/number-of-android-apps.

[7] “Pandaapp,” Online; accessed at August 7, 2015, http://www.pandaapp.com.

[8] “Pandaapp,” Online; accessed at August 7, 2015, http://www.getjar.mobi.

[9] W. Enck, “Defending users against smartphone apps: Techniques and future directions,” in *Proc. of the 7th International Conference on Information Systems Security (ICISS’11), Kolkata, India, LNCS*, S. Jajodia and C. Mazumdar, Eds., vol. 7093. Springer Berlin Heidelberg, December 2011, pp. 49–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25560-1_3

[10] R. Mittal, A. Kansal, and R. Chandra, “Empowering developers to estimate app energy consumption,” in *Proc. of the 18th annual international conference on Mobile Computing and Networking (CMCN’18), Istanbul, Turkey*. ACM, August 2012, pp. 317–328. [Online]. Available: http://doi.acm.org/10.1145/2348543.2348583

[11] O. R. E. Pereira and J. J. P. C. Rodrigues, “Survey and analysis of current mobile learning applications and technologies,” *ACM Computing Surveys*, vol. 46, no. 2, pp. 27:1–27:35, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2543581.2543594

[12] 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

[13] E. Fernandes, B. Crispo, and M. Conti, “Fm 99.9, radio virus: Exploiting fm radio broadcasts for malware deployment,” *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 6, pp. 1027–1037, June 2013.

[14] R. Fedler, J. Schütte, and M. Kulicke, “On the effectiveness of malware protection on android,” June 2013.

[15] C. Jarabek, D. Barrera, and J. Aycock, “Thinav: Truly lightweight mobile cloud-based anti-malware,” in *Proc. of the 28th Annual Computer Security Applications Conference (ACSAC’12), Orlando, Florida, USA*. ACM, December 2012, pp. 209–218. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420983

[16] S. Brahler, “Analysis of the android architecture,” 2010. [Online]. Available: https://os.itec.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf

[17] Android, “Google android documents, android framework architecture,” Online; accessed at May 28, 2015, https://source.android.com/devices/. [Online]. Available: https://source.android.com/devices/

[18] N. Elenkov, *Android Security Internals: An In-Depth Guide to Android’s Security Architecture*, 1st ed. San Francisco, CA, USA: No Starch Press, 2014.

[19] Android, “Google android documents, android application architecture,” Online; accessed at May 28, 2015, http://developer.android.com/guide/components/fundamentals.html. [Online]. Available: http://developer.android.com/guide/components/fundamentals.html

[20] J. Annuzzi, L. Darcey, and S. Conder, *Introduction to Android Application Development: Android Essentials*, ser. Developer’s Library. Addison Wesley, 2014. [Online]. Available: https://books.google.com/books?id=c1kXAgAAQBAJ

[21] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. Gaur, M. Conti, and R. Muttukrishnan, “Android security: A survey of issues, malware penetration and defenses,” 2015. [Online]. Available: http://dx.doi.org/10.1109/COMST.2014.2386139

[22] K. Makan and S. Alexander-Bown, *Android Security Cookbook*. Packt Publishing, 2013.

[23] Android, "Google android documents, android application sandboxing mechanism," Online; accessed at May 25, 2015, http://developer.android.com/training/articles/security-tips.html. [Online]. Available: http://developer.android.com/training/articles/security-tips.html

[24] C. Efstratiou and I. Leontiadis, "What is the price of free," Online; accessed at April 17, 2012, http://www.cam.ac.uk/research/news/what-is-the-price-of-free.

[25] S. Gunasekera, *Android Apps Security*, 1st ed. Berkely, CA, USA: Apress, 2012.

[26] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. of the 8th Symposium on Usable Privacy and Security (SOUPS'12), Pittsburgh, PA, USA*. ACM, July 2012, pp. 3:1–3:14. [Online]. Available: http://doi.acm.org/10.1145/2335356.2335360

[27] 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, IL, USA*. ACM, October 2011, pp. 627–638.

[28] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *Proc. of the 13th Information Security Conferenec (ISC'11), Boca Raton, Florida, USA, LNCS*, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, Eds., vol. 6531. Springer Berlin Heidelberg, October 2011, pp. 346–360. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18178-8_30

[29] H. Huang, S. Zhu, P. Liu, and D. Wu, "A framework for evaluating mobile app repackaging detection algorithms," in *Proc. of the 6th International Conference on Trust and Trustworthy Computing (TRUST'13), London, UK, LNCS*, M. Huth, N. Asokan, S. Čapkun, I. Flechais, and L. Coles-Kemp, Eds., vol. 7904. Springer Berlin Heidelberg, June 2013, pp. 169–186. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38908-5_13

[30] 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

[31] E. Kovacs, "Wi-fi direct flaw exposes android devices to dos attacks," Online; accessed at July 8, 2015, http://www.securityweek.com/wi-fi-direct-flaw-exposes-android-devices-dos-attacks.

[32] C. Marforio, A. Francillon, and S. Capkun, *Application Collusion Attack on the Permission-Based Security Model and Its Implications for Modern Smartphone Systems*. Department of Computer Science, ETH Zurich, 2010. [Online]. Available: https://books.google.com/books?id=nvszMwEACAAJ

[33] 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

[34] A. Gember, C. Dragga, and A. Akella, "ECOS: Leveraging software-defined networks to support mobile application offloading," in *Proc. of the 8th ACM/IEEE symposium on Architectures for networking and communications systems (ANCS'12), University of Texas in Austin, TX, USA*, October 2012, pp. 199–210. [Online]. Available: http://doi.acm.org/10.1145/2396556.2396598

[35] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing," in *Proc. of the 12th ACM Conference on Ubiquitous Computing (UbiComp'12), Pittsburgh, PA, USA*. ACM, September 2012, pp. 501–510. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370290

[36] D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot, "Understanding and improving app installation security mechanisms through empirical analysis of android," in *Proc. of the 2nd ACM workshop on Security and privacy in smartphones and mobile devices (SPSM'12), Raleigh, NC, USA*. ACM, October 2012, pp. 81–92. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381949

[37] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps," 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. 267–280. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307661

[38] 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

[39] A. Gosain and G. Sharma, “A survey of dynamic program analysis techniques and tools,” in *Proc. of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA’14), Odisha, India*, S. C. Satapathy, B. N. Biswal, S. K. Udgata, and J. Mandal, Eds., vol. 327. Springer International Publishing, November 2015, pp. 113–122. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11933-5_13

[40] B. Rashidi, C. Fung, and T. Vu, “Recdroid: A resource access permission control portal and recommendation service for smartphone users,” in *Proc. of the 2014 ACM MobiCom Workshop on Security and Privacy in Mobile Environments (SPME’14), Maui, Hawaii, USA*. ACM, September 2014, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2646584.2646586

[41] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, “Mockdroid: Trading privacy for application functionality on smartphones,” in *Proc. of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile’11), Phoenix, Arizona, USA*. ACM, March 2011, pp. 49–54. [Online]. Available: http://doi.acm.org/10.1145/2184489.2184500

[42] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, “Appguard: Enforcing user requirements on android apps,” in *Proc. of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’13), Rome, Italy*. Springer-Verlag, March 2013, pp. 543–548. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36742-7_39

[43] G. Russello, A. B. Jimenez, H. Naderi, and W. van der Mark, “Firedroid: Hardening security in almost-stock android,” in *Proc. of the 29th Annual Computer Security Applications Conference (ACSAC’13), New Orleans, Louisiana, USA*. ACM, December 2013, pp. 319–328. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523678

[44] 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

[45] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, “Crowdroid: Behavior-based malware detection system for android,” in *Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM’11), Chicago, Illinois, USA*. ACM, October 2011, pp. 15–26. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046619

[46] 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

[47] T. Ball, “The concept of dynamic analysis,” *ACM SIGSOFT Software Engineering Notes (SEN)*, vol. 24, no. 6, pp. 216–234, Oct. 1999. [Online]. Available: http://doi.acm.org/10.1145/318774.318944

[48] WikiPedia, “Wikipedia, crwodsourcing definition,” Online; accessed at May 29, 2015, http://en.wikipedia.org/wiki/Crowdsourcing. [Online]. Available: http://en.wikipedia.org/wiki/Crowdsourcing

[49] R. Lord, “How to hack a mobile app: It’s easier than you think!” Online; accessed at July 16, 2015, https://www.arxan.com/hack-mobile-app-easier-think/.

[50] ARXAN, “Arxan technologies, securing mobile apps in the wild with app hardening and run-time protection,” Online; accessed at July 16, 2015, https://www.arxan.com/resources/mobile-application-protection-handbook/.

[51] W. Zhou, X. Zhang, and X. Jiang, “Appink: Watermarking android apps for repackaging deterrence,” in *Proc. of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS’13), Hangzhou, China*. ACM, May 2013, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/2484313.2484315

[52] E. Lafortune, “Proguard: Java shrinker, optimizer, obfuscator, and preverifier,” Online; accessed at June 25,

2015, http://proguard.sourceforge.net.

[53] M. Zheng, P. P. C. Lee, and J. C. S. Lui, "Adam: An automatic and extensible platform to stress test android anti-virus systems," 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. 82–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37300-8_5

[54] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: Analyzing contextual use of permissions in android applications," in *Proc. of the 37th International Conference on Software Engineering (ICSE'15), Florence, Italy*. IEEE, May 2015.

[55] 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

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

[57] 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

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

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

[60] 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.

[61] 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

[62] 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

[63] 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

[64] 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.

[65] 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

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

[67] 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

[68] 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

[69] 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

[70] 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

[71] 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 &#38; Communications Security (CCS'13), Berlin, Germany*. ACM, November 2013, pp. 611–622. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516689

[72] 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.

[73] 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 &#38; Communications Security (CCS'13), Berlin, Germany*. ACM, November 2013, pp. 1043–1054. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516676

[74] 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

[75] 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

[76] 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

[77] 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

[78] 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

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

[80] 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

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

[82] 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.

[83] 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

[84] 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

[85] 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

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

[87] 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

[88] 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

[89] 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

[90] 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.

[91] 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

[92] 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

[93] B. Rashidi, C. Fung, and T. Vu, "Dude, ask the experts!: Android resource access permission recommendation with recdroid," in *Proc. of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM'15), Ottawa, Canada*, May 2015, pp. 296–304.

[94] B. Rashidi, C. Fung, G. Bond, S. Jackson, M. Pare, and T. Vu, "Demo: Recdroid: An android resource access permission recommendation system," in *Proc. of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'15), Hangzhou, China*. ACM, June 2015, pp. 403–404. [Online]. Available: http://doi.acm.org/10.1145/2746285.2764930

[95] B. Rashidi and C. Fung, "A game-theoretic model for defending against malicious users in recdroid," in *the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM'15), Ottawa, Canada*, May 2015, pp. 1339–1344.

[96] ——, "Disincentivizing malicious users in recdroid using bayesian game model," *Journal of Internet Services and Information Security (JISIS)*, vol. 5, no. 2, pp. 33–46, May 2015.

[97] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10), Vancouver, BC, Canada*. USENIX Association, October 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[98] WikiPedia, "Reference monitor," Online; accessed at July 8, 2015, https://en.wikipedia.org/wiki/Reference monitor.

[99] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and fine-grained mandatory access control on android for diverse security and privacy policies," in *Proc. of the 22nd USENIX Security Symposium (USENIX Security'13), Washington, D.C., USA*. USENIX Association, October 2013, pp. 131–146. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bugiel

[100] P. Loscocco and S. Smalley, in *Proc. of the 2001 USENIX Annual Technical Conference, FREENIX Track, Boston, Massachusetts, USA*. USENIX Association, June 2001.

[101] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," in *Proc. of the 2009 Annual Computer Security Applications Conference (ACSAC'09), Austin, TX, USA*. IEEE, December 2009, pp. 340–349.

[102] S. Smalley and R. Craig, "Security enhanced (SE) android: Bringing flexible MAC to android," in *Proc. of the 2014 Network and Distributed System Security Symposium (NDSS'14), San Diego, CA, USA*. The Internet Society, February 2013. [Online]. Available: http://internetsociety.org/doc/security-enhanced-se-android-bringing-flexible-mac-android

[103] M. Ongtang, K. Butler, and P. McDaniel, "Porscha: Policy oriented secure content handling in android," in *Proc. of the 26th Annual Computer Security Applications Conference (ACSAC'10), Austin, Texas, USA*. ACM, December 2010, pp. 221–230. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920295

[104] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Proc. of the 21st Annual International Cryptology Conference (CRYPTO'01), Santa Barbara, California, USA, LNCS*, J. Kilian, Ed., vol. 2139. Springer Berlin Heidelberg, August 2001, pp. 213–229. [Online]. Available: http://dx.doi.org/10.1007/3-540-44647-8_13

[105] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach, "Quire: Lightweight provenance for smart phone operating systems." in *Proc. of the 20th USENIX Security Symposium, San Francisco, CA*. USENIX Association, August 2011.

[106] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," Technische Universität Darmstadt, Technical Report TR-2011-04, Apr. 2011.

[107] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on android," in *Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'11), Chicago, Illinois, USA*. ACM, October 2011, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046624

[108] N. D. Corporation, "Tomoyo linux, a mandatory access control (mac) implementation for linux," Online; accessed at July 5, 2015, http://tomoyo.osdn.jp/about.html.en.

[109] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proc. of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS'12), Seoul, Korea*. ACM, May 2012, pp. 71–72. [Online]. Available: http://doi.acm.org/10.1145/2414456.2414498

[110] M. Sun and G. Tan, "Nativeguard: Protecting android applications from third-party native libraries," in *Proc. of the 2014 ACM Conference on Security and Privacy in Wireless &#38; Mobile Networks (WiSec'14), Oxford, UK*. ACM, July 2014, pp. 165–176. [Online]. Available: http://doi.acm.org/10.1145/2627393.2627396

[111] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: Versatile protection for smartphones," in *Proc. of the 26th Annual Computer Security Applications Conference (ACSAC'10), Austin, Texas, USA*. ACM, December 2010, pp. 347–356. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920313

[112] L. Gibelli, "Clamav, an open source (gpl) anti-virus engine," Online; accessed at July 5, 2015, http://www.clamav.net/index.html.

[113] O. Tripp and J. Rubin, "A bayesian approach to privacy enforcement in smartphones," in *Proc. of the 23rd USENIX Conference on Security Symposium (SEC'14), San Diego, CA, USA*. USENIX Association, August 2014, pp. 175–190. [Online]. Available: http://dl.acm.org/citation.cfm?id=2671225.2671237

[114] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Proc. of the 4th International Conference on Trust and Trustworthy Computing (TRUST'11),*

Pittsburgh, PA, USA, LNCS, vol. 6740. Springer Berlin Heidelberg, June 2011, pp. 93–107.

[115] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in *Proc. of the 18th ACM Conference on Computer and Communications Security (CCS'11), Chicago, Illinois, USA*. ACM, October 2011, pp. 639–652. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046780

[116] K. Fawaz and K. G. Shin, "Location privacy protection for smartphone users," in *Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14), Scottsdale, Arizona, USA*. ACM, November 2014, pp. 239–250. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660270

[117] 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, pp. 161–190, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10844-010-0148-x

[118] C. Wu, Y. Zhou, K. Patel, Z. Liang, and X. Jiang, "Airbag: Boosting smartphone resistance to malware infection," in *Proc. of the 2014 Network and Distributed System Security Symposium (NDSS'14), San Diego, CA, USA*. The Internet Society, February 2014. [Online]. Available: http://www.internetsociety.org/doc/airbag-boosting-smartphone-resistance-malware-infection

[119] S. Shekhar, M. Dietz, and D. S. Wallach, "Adsplit: Separating smartphone advertising from applications," in *Proc. of the 21st USENIX Security Symposium (USENIX Security'12), Bellevue, WA, USA*. USENIX Association, August 2012, pp. 553–567. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/shekhar

[120] X. Zhang, A. Ahlawat, and W. Du, "Aframe: Isolating advertisements from mobile applications in android," in *Proc. of the 29th Annual Computer Security Applications Conference (ACSAC'13), New Orleans, Louisiana, USA*. ACM, December 2013, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523652

[121] F. Roesner and T. Kohno, "Securing embedded user interfaces: Android and beyond," in *Proc. of the 22nd USENIX Security Symposium (USENIX Security'13), Washington, D.C., USA*. USENIX Association, August 2013, pp. 97–112. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/roesner

[122] M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Proc. of the 13th International Conference (ISC'11), Boca Raton, Florida, USA, LNCS*, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilić, Eds., vol. 6531. Springer Berlin Heidelberg, October 2011, pp. 331–345. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18178-8_29

[123] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending android permission model and enforcement with user-defined runtime constraints," in *Proc. of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS'10), Beijing, China*. ACM, April 2010, pp. 328–332. [Online]. Available: http://doi.acm.org/10.1145/1755688.1755732

[124] S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi, "Asm: A programmable interface for extending android security," in *Proc. of the 23rd USENIX Security Symposium (USENIX Security'14), San Diego, CA, USA*. USENIX Association, August 2014, pp. 1005–1019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/heuser

[125] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Proc. of the 21st USENIX Conference on Security Symposium (Security'12), Bellevue, WA, USA*. USENIX Association, August 2012, pp. 27–27. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362820

[126] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. android and mr. hide: Fine-grained permissions in android applications," in *Proc. of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'12), Raleigh, North Carolina, USA*. ACM, October 2012, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381938

[127] A. P. Felt, S. Hanna, E. Chin, H. J. Wang, and E. Moshchuk, "Permission re-delegation: Attacks and defenses," in *Proc. of the 20th Usenix Security Symposium (Usenix Security'11), San Francisco, CA, USA*. USENIX Association, August 2011.

[128] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love

android: An analysis of android ssl (in)security," in *Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA*. ACM, October 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[129] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4android: A generic operating system framework for secure smartphones," in *Proc. of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM'11), Chicago, IL, USA*. ACM, October 2011, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046623

[130] M. Zhang and H. Yin, "AppSealer: Automatic generation of vulnerability-specific patches for preventing component hijacking attacks in Android applications," in *Proc. of the 2014 Network and Distributed System Security Symposium (NDSS'14), San Diego, CA, USA*. The Internet Society, February 2014.

[131] K. Kennedy, E. Gustafson, and H. Chen, "Quantifying the effects of removing permissions from android applications."

[132] R. Wang, L. Xing, X. Wang, and S. Chen, "Unauthorized origin crossing on mobile platforms: Threats and mitigation," in *Proc. of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security (CCS'13), Berlin, Germany*. ACM, November 2013, pp. 635–646. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516727

[133] C. Mulliner, J. Oberheide, W. Robertson, and E. Kirda, "Patchdroid: Scalable third-party security patches for android devices," in *Proc. of the 29th Annual Computer Security Applications Conference (ACSAC'13), New Orleans, Louisiana, USA*. ACM, December 2013, pp. 259–268. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523679

[134] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proc. of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security (CCS'13), Berlin, Germany*. ACM, November 2013, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516693

[135] L. Li, A. Bartel, T. BissyandÃ©, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps," in *Proc. of the 37th ACM/IEEE International Conference on Software Engineering (ICSE'15), Firenze, Italy*, vol. 1. IEEE, May 2015, pp. 280–291.

[136] A. Desnos and P. Lantz, "Droidbox: An android application sandbox for dynamic analysis," Online; accessed at July 2, 2015, https://code.google.com/p/droidbox/. [Online]. Available: https://code.google.com/p/droidbox/

[137] Drozer, "Drozer - a comprehensive security and attack framework for android," Online; accessed at June 20, 2015, https://www.mwrinfosecurity.com/products/drozer/. [Online]. Available: https://www.mwrinfosecurity.com/products/drozer/(Online;LastAccessed11thFebruary2013)

[138] K. Tam, K. Salahuddin, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors," in *Proc. of the 2015 Network and Distributed System Security Symposium (NDSS'15), San Diego, CA, USA*. The Internet Society, February 2015.

[139] W. Luo, S. Xu, and X. Jiang, "Real-time detection and prevention of android sms permission abuses," in *Proc. of the 1st International Workshop on Security in Embedded Systems and Smartphones (SESP'13), Hangzhou, China*. ACM, September 2013, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/2484417.2484422

[140] S. Sakamoto, K. Okuda, R. Nakatsuka, and T. Yamauchi, "Droidtrack: Tracking and visualizing information diffusion for preventing information leakage on android," *Journal of Internet Services and Information Security (JISIS)*, vol. 4, no. 2, pp. 55–69, May 2014.

[141] Q. Ismail, T. Ahmed, A. Kapadia, and M. K. Reiter, "Crowdsourced exploration of security configurations," in *Proc. of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'15), Seoul, Republic of Korea*. ACM, April 2015, pp. 467–476. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702370

Author Biography

Bahman Rashidi is a PhD student in Computer Science at the Virginia Commonwealth University, USA. He received his BSc and MSc in computer engineering from University of Isfahan and Iran University of Science and Technology, Tehran, Iran, in 2011 and 2014 respectively. He is mainly interested in the Distributed Systems, Cloud Computing, Mobile Computing, Mobile Devices, and Privacy. Currently, he is doing research on a Malware detection framework for smartphones. He is the recipient of Distinguished Masters Student of the year in research award for two consecutive years in 2012 and 2013 from Iran University of Science and Technology and Outstanding Early-career Student Researcher, Virginia Commonwealth University, April 2015.

Carol Fung received her Bachelor degree and Master degree in computer science from the university of Manitoba (Canada), and her PhD degree in computer science from the university of Waterloo (Canada).Her research interests include collaborative intrusion detection networks, social networks, security issues in mobile networks and medical systems, Security issues in next generation networking, and machine learning in intrusion detection. She is the recipient of the young professional award in IEEE/IFIP IM 2015, Alumni Gold Medal of university of Waterloo in 2013, best dissertation awards in IM2013, the best student paper award in CNSM2011 and the best paper award in IM2009. She received numerous prestige awards and scholarships including Google Anita Borg scholarship, NSERC Postdoc fellowship, David Cheriton Scholarship, NSERC Postgraduate Scholarship, and President's graduate scholarship. She has been a visiting scholar at POSTECH (South Korea), a software engineer intern at Google, and a research intern at BlackBerry.