*I.J. Computer Network and Information Security*, **2014, 12, 9-20**
Published Online November 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2014.12.02

# A Comprehensive Analysis of Android Security and Proposed Solutions

**Asim S. Yuksel**
Suleyman Demirel University/Computer Engineering Department, Isparta, 32260, Turkey
Email: asimyuksel@sdu.edu.tr

**Abdul H. Zaim**
Istanbul Commerce University/Computer Engineering Department, Istanbul, 34378, Turkey
Email: azaim@ticaret.edu.tr

**Muhammed A. Aydin**
Istanbul University/Computer Engineering Department, Istanbul, 34320, Turkey
Email: aydinali@istanbul.edu.tr

***Abstract***—The increasing popularity of smart devices have led users to complete all of their daily work with these devices. Users are now able to shop online, share information with the applications that they install on their smart devices. Installed applications gain access to various sensitive information, such as the user's contact list, phone number, location. However, there is no control mechanism in place that can check whether these applications are safe to install. Therefore, applications are installed according to the users' decisions, without any limitations or warnings. As a result, users become the target of malicious applications, and the personal security and privacy are compromised. In this study, we investigate the security solutions that aim to protect the privacy and security of Android users. We reveal the shortcomings of mobile security solutions and shed light on the research community. Additionally, we present the taxonomy of Android-based mobile security solutions.

***Index Terms***—Mobile Security, Mobile Privacy, Mobile Application Security, Android Operating System, Android Security Architecture.

## I. Introduction

There is a large number of operating systems that are used for mobile devices. Through use of these operating systems, Android continuously increases its popularity and market share. According to the information that Google provided in September 2012, 500 million Android devices have been activated [1]. In addition, the open-source nature of the Android platform, the ease of application development and the submission process with the application store have made this platform more attractive. However, security risks and threats have increased and continue to increase more so than for other mobile platforms, such as Apple's iOS. Anyone who wants to develop Android-based mobile applications is able to submit his/her application to Google's application store without any problems. The applications that are developed can compromise personal security, privacy and user experience by misusing sensitive information, such as photos, the contact list, e-mails, documents, SMS, calling services, the battery and the camera. This misuse of sensitive information is the most important and indispensable problem that affects these users and mobile devices. When we examine the literature, we see that there are many related studies that aim to provide solutions regarding user privacy and security that can make the Android operating system more secure. These studies expose privacy and security problems and provide solutions to these problems. A few of these studies depend on the Android permission model, and a few depend on rebuilding applications. Solutions that depend on rebuilding applications step down to the byte-code level and make changes to the behavior of applications according to specific/predefined rules. Although these provide certain levels of security and privacy, they are not user-friendly/focused because they are not submitted to provide a service to mobile device owners. The main goal of our study is to investigate the Android security mechanism and proposed security solutions for Android between the years 2008 and 2013 after Android was released as an open-source platform in November 2008. Additionally, we discuss the strengths and weaknesses of these proposed solutions. The selected papers were chosen from various security-related journals, workshops, technical reports and conferences that include Android-related research papers. We believe that this work will shed light for the research community who will research this subject and provide a basis for the future development of mobile security solutions. Our main contribution is the novel taxonomy of Android-based mobile security solutions that covers a tremendous number of research studies on this topic. The primary goal of creating this taxonomy is to organize mobile security solutions that can be used to help researchers understand the problems that affect the security of mobile devices and take better countermeasures. The remainder of this paper is organized as follows: In section 2, we

 

describe the Android operating system, its history, architecture and security models that emphasize application security and permissions. Section 3 presents our novel taxonomy of mobile security. In section 4, we summarize our findings and suggest future work. Finally, in section 5 we conclude our paper.

## II. Android Operating System

As the number of Android-based mobile devices increases, more data are used by these devices. Due to the enormous amount of personal data on these devices, they pose a threat and present an inviting environment within which cyber criminals can attack. To be able to defend against attacks and develop solutions, Android developers, companies, and researchers must fully understand the platform components, the platform's architecture and the operation principles of the Android platform.

### *A. Android History*

Android is an open-source, Linux-based operating system that was developed under the leadership of the Open Handset Alliance (OHA) and Google. The platform was previously developed by Android Inc., which was a Silicon Valley-based company. In 2005, Google acquired this company, and the Android operating system became a growing, developing platform. After 3 years of development, the first Android-based mobile device was available for sale in November 2008. Table 1 shows the milestones of the Android platform.

Table 1. Milestones of Android Platform

| Date | Event |
|---|---|
| 1 July 2005 | Google acquired Android Inc. |
| 12 November 2007 | Android was released. |
| 28 August 2008 | Android Market was announced. |
| 23 September 2008 | Android 1.0 platform was released. |
| 21 November 2008 | Android was released as open-source. |
| 13 February 2009 | Paid applications were accepted in the USA Android Market. |
| 2009-2013 | Android platform was updated to new versions and It continues to be updated. Latest version is Android 4.4 Kitkat. |

Two years after the announcement of the first Android-based mobile device, Android was the second biggest platform, with a 26% market share and 65 million users. Today, in 2014, it has become the largest platform, with a 52% market share, over 100 million users in the USA, and a 79% market share worldwide [2], [3]. In the following, Table 2 shows the market share of the Android platform in the USA in August 2014 and Table 3 shows the market share worldwide in the second quarter of 2013 and 2014. As it is seen from the table, Android has increased its market share by 5.1% while Apple's iOS has decreased its market share by 1.3%.

Table 2. Top 5 Mobile Platforms in the USA (May-Aug. 2014) [2]

| Smart Phone Owners in the USA (%) | | | |
|---|---|---|---|
| **Platforms** | **May 2014** | **Aug. 2014** | **Change** |
| Android | 52.1% | 52.0% | -0.1 |
| Apple iOS | 41.9% | 42% | 0.1 |
| Windows Phone | 3.4% | 3.5% | 0.1 |
| Blackberry OS | 2.3% | 2.3% | 0.0 |
| Symbian OS | 0.1% | 0.1% | 0.0 |

Table 3. Top 5 Mobile Platforms Worldwide Q2 2014 [3]

| Smart Phone Owners Worldwide (%) | | | |
|---|---|---|---|
| **Platforms** | **Q2 2013** | **Q2 2014** | **Change** |
| Android | 79.6% | 84.7% | 5.1 |
| Apple iOS | 13.0% | 11.7% | 1.3 |
| Windows Phone | 3.4% | 2.5% | -0.9 |
| Blackberry OS | 2.8% | 0.5% | -2.3 |
| Others | 1.2% | 0.6% | -0.6 |

### *B. Android Architecture*

To develop a security analysis, security software products or security services, it is necessary to have a good understanding of the Android architecture. In this section, we mention the details of the Android architecture and its layers. Fig.1 shows all of the layers of this architecture.

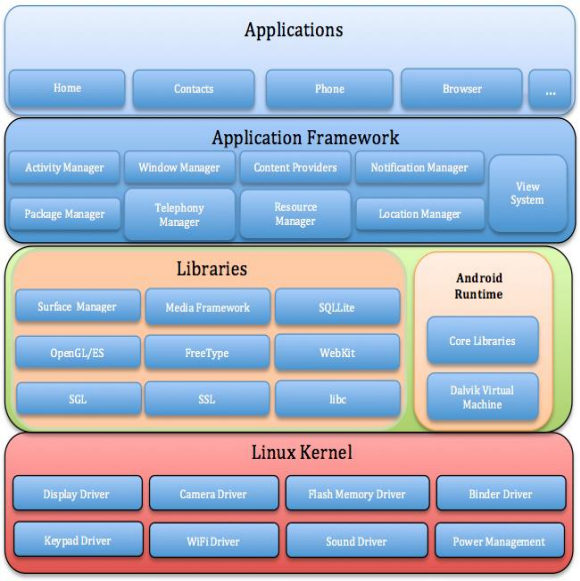


Fig.1. Android Architecture and Its Layers [4, 5]

Android architecture is comprised of 5 basic layers, and each layer has different program groups. The following list provides these layers [4, 5]:

- Application Layer
- Application Framework Layer
- Library Layer
- Runtime Layer
- Linux Layer

***1) Linux Layer:*** This layer resides at the bottom of the architecture. Although developers and users do not have a direct communication with this layer, it is the heart of the whole system. It provides following functions in Android system:

- Abstraction for hardware
- Memory Management
- Security
- Power management
- Hardware drivers
- Support for shared libraries
- Network connection
- A Binder framework for inter-process communication

***2) Library Layer:*** This layer resides on top of the Linux kernel layer and includes several libraries. These libraries provide functionalities that can handle various data. For instance, the Media Framework is responsible for the management of how different types of videos or audio will be played. The following list provides other open-source libraries that are included in this layer:

- **Surface Manager:** Responsible for the management of windows on the screen.
- **SGL:** Graphic library that provides 2D functionality.
- **OpenGL/ES:** Graphic library that provides 3D functionality.
- **Media Framework**: Responsible for audio, vide playback, recording, photo display, etc.
- **Freetype:** Library that manages fonts.
- **Webkit:** Browser engine.
- **Libc:** System C library.
- **SqlLite:** Serverless SQL database.
- **Open SSL:** Security library.

***3) Runtime Layer:*** This layer is located in the same level as the Library layer. It contains a Dalvik Virtual Machine (DVM) and Java libraries for users that are used in the development of applications. The virtual machine requires the applications to run on Android devices. It is register-based and optimized for low memory requirements. It runs on the application codes that are converted from Java class files to DVM compatible DEX files.

***4) Application Framework Layer:*** This layer is where the developed applications directly communicate. The applications manage the basic functionalities, such as phone resource management, sound management and call management. The management applications include the following:

- **Activity Manager:** Responsible for the activity life cycle of applications.
- **Content Provider:** Responsible for data exchange between applications.
- **Telephony Manager:** Responsible for all of the voice calls.
- **Location Manager:** Responsible for location management by using GPS coordinates and cell towers.
- **Resource Manager:** Responsible for the management of resources that are used by applications.

***5) Application Layer:*** This is the top level layer in Android architecture in which the standard applications reside and where users have the most interaction by making calls, receiving calls, surfing online, etc. The layers where the developers and programmers have the most interaction are the layers that are between the Linux Kernel layer and this layer.

### *C. Android Security*

The Android operating system has a security architecture that protects the security of users, data, applications, devices and networks. The architecture provides a multi-layered security model and maintains flexibility in its design due to its open-source nature [5]. The Android security architecture is developed with the aim of being the most functional, powerful, and secure mobile operating system by protecting the users' personal data and the system resources of mobile devices. To achieve this goal, it supplies the following security features:

- Powerful security mechanism on the Linux Kernel Level.
- Mandatory application isolation (sandboxing) for all of the applications.
- Secure inter-process communication
- Application signing
- User approved and application specific permissions.

The major factor that leads to the extensive use of Android devices is their mobile applications. Thus, in this study, we dwell on the security mechanism that is related to Application Security, with a special emphasis on the permission model.

### *D. Android Application Security and Permission Model*

Android applications are generally coded in the Java programming language, and they run on DVM. In addition, C/C++ language can be used. Applications are installed from a single file that has the ".apk" extension. The basic structure of an Android application includes the following:

- **Android Manifest File:** This file is labeled "AndroidManifest.xml", and it controls how high-level components (such as activities, services, content providers, and broadcast receivers) communicate with the system. Additionally, it defines what permissions are necessary to run the applications.

 

- **Activities:** An activity is a piece of code that focuses on one task. It generally contains a user interface and one of the activities is always the starting point of the application.
- **Services:** These are the code fragments that run on the background. They can run inside their own process or another application's process. Other components connect to these services and call methods by using the Remote Method Invocation technique.
- **Broadcast Receivers:** These objects take action when the processes that are called "Intent" are created by the operating system or by other applications. Applications register to these broadcast receivers and change their behavior according to the incoming data.

In Android, all of the applications run inside of a security isolation box that is referred to as the 'Sandbox". As a default, applications have access to limited system resources. The permission mechanism handles the management of Android applications' access to resources and checks whether they access resources properly and do not behave maliciously. Restrictions are developed by using different techniques. In certain cases, storage isolation is chosen for protection; in other cases, restrictions are created based on the "Permission List" mechanism that restricts access to sensitive APIs. A few of these protected APIs include the following:

- Camera
- Location (GPS)
- Bluetooth
- Phone
- SMS/MMS
- Network/Data (GSM and Wi-Fi)

These APIs can only be accessed through the operating system. SMS/MMS, Phone, Network/Data and NFC APIs are the most important APIs because the misuse of these APIs by malicious applications will cause financial harm to users. To be able to use the protected APIs, each application should specify the functionality of the API in its own manifest file. During the application installation process, the system presents the user with a dialog screen that contains a list of permissions and asks the user whether to continue with installation or not. This approach is based on an "accept all or reject all" principle and does not allow users to select specific permissions. The user either accepts the permissions that are listed on the screen, and the application is installed, or he/she rejects the permissions, and the application is not installed. As long as the application stays installed on the system, the permissions are valid, and the permission list window is never shown. When the application is uninstalled, the permissions are also removed. If a permission that is not specified in the manifest file is used, the system throws a security exception and stops the application from being launched. There are 134 permissions in the Android platform that should be specified in the manifest file before an application can be used [6]. Additionally, applications have the ability to define their own permissions. However, defining a new permission is not recommended since built-in permissions in the system cover many situations.

## III. Taxonomy of the Android Security

The taxonomy that we have created in this study is the process of collecting, organizing and representing the relevant solutions in the mobile security domain. It is based on the studies of existing security solutions proposed for Android. We threat Android security as the main domain and we analyze it to get a better understanding and overview. Before creating taxonomy, mobile security knowledge is prerequisite. When creating the taxonomy, the main problem is accessing and obtaining Android based mobile security solutions. The most efficient way is to survey researchers of security field. However, it is a huge area and it is impossible to survey all of them. Therefore, our strategy is to collect the security knowledge by examining the aims and scopes of security related journals, conferences that include mobile security, privacy, information security, and Android security as keywords. As a result of our efforts, we examine the proposed solutions for Android security under two main titles: "Software-Based Solutions" and "Hardware-Based Solutions". Fig.2 shows our taxonomy of Android security. In this study, we focus on software-based solutions.

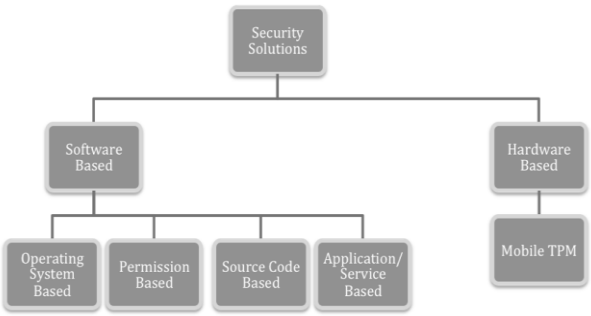


Fig.2. Taxonomy of Android Security

### A. Software Based Solutions

We classify the software-based solutions into four groups. These include operating system (OS) based, permission-based, source code based and application/service based solutions.

#### A.1. Operating System Based Solutions

These types of security solutions make changes on the operating system architecture. In [7], researchers developed a system (APEX) that allows selective permissions, the definition of constraints, the restriction of resource usage and user flexibility. With the help of an additional user interface, users have the power to allow or reject any of the services that are listed on the interface during the application installation phase. This system achieves its goal by making changes on the operating

system code. Although the conducted study adds powerful features to the Android permission mechanism and provides an option to select permissions, it does not contribute to the security or privacy of users who have no technical knowledge and do not know what types of risks they will face when allowing different permissions without knowing what these permissions entail. Additionally, for users to take advantage of this system, the developed system must be deployed to the users' devices. This is nearly impossible for normal users who do not have technical knowledge, and this process will void the devices' warranty. Finally, the developed software, Apex, was not submitted to the service of users, developers or researchers in mind.

Ref. [8] developed TrustDroid, a software framework that targets the business world and secures mobile devices in each layer of the Android architecture. At the kernel level, it has a Kernel MAC (Mandatory Access Control) manager that controls the inter-process communication and administrates the file system; at the middle layer, a Policy Manager and a Firewall Manager are used; and at the top layer, a customized Package Manager is used. The "Application Coloring" technique is used to provide security. In this technique, a color is assigned to each application to signify that the application will belong to a certain domain. Only the applications that belong to the same domain are allowed to communicate with each other and send and receive data. Therefore, applications that are marked with a color that indicates they are unsafe are never allowed to communicate with the applications that are marked as safe. The system in this study is developed by making changes to the Android operating system and by adding extensions to it. The users need to install this customized operating system to take advantage of it.

SELinux (Security-Enhanced Linux) is a core security module that is developed by adding extra features to Linux. In [9] authors aimed to create a more secure system by adapting the SELinux core module to the Android operating system. To achieve their goal, they recompiled the Android source code with the SELinux, which requires intervening the operating system. Their system is useful only if the mobile device owners remove the current operating system from their devices and install this modified version.

In [10], researchers developed SAINT, a software framework that protects the security of Android devices. The developed framework is built by modifying the middle layer (middleware) of the Android operating system. The architecture is comprised of 5 modules. The first module is Saint Installer Software. This module is a modified version of the Android application installer. It becomes active when installing an application; it inspects the permissions that the application uses and compares them to ones that are predefined in the second module, which is called the Application Policy Manager (AppPolicy). If there is no match or there is a conflict, the application is not installed. The third module is Saint Mediator. This module becomes active at runtime and audits the processes based on whether they operate according to predefined rules. The forth module is the Framework Policy Manager. It serves as an application and provides the option to change the predefined rules. Rules can only be changed via this application. The fifth module provides a way for application developers to add their own predefined rules to SAINT. Once again, the users can only take advantage of this solution by installing a modified version of operating system that includes SAINT software framework.

### *A.2. Permission-Based Solutions*

Each API call in the Android operating system corresponds to a permission in the manifest file (AndroidManifest.xml) that contains the list of permissions. When a user installs an application, the list of permissions is presented to the user. When the permission is granted, API calls become active. However, users can only allow or reject all of the permissions and do not have the power to select certain permissions. Allowing many unnecessary permissions causes security and privacy problems. Once the permissions are granted at the installation time, there is no way of changing these permissions. Furthermore, the model does not support dynamic permission assignment. After an application is granted permissions, the users have no idea about how the application will use the data on their devices and what effects it will have on privacy and security. Permission-based security solutions provide experimental analysis and practical solutions. Experimental analysis studies generally investigate how the permission mechanism is used or misused and whether the applications use excessive or incomplete permissions, and then the studies produce related reports. However, practical solutions aim to provide permission-based filtering and remove permissions that are considered to be harmful.

Ref. [11] experimentally analyzes the permission model in the Android operating system. The main purpose of the study is to investigate the weak and strong parts of the system and to reveal how the permission mechanism is used in practice, which refers to whether the design considerations meet the expectations of real world features. Authors analyzed 1100 Android-based applications and tried to reveal how application developers use the permission model. 2D visualization of the data analysis was performed by employing Self Organizing Maps. According to their analysis results, a very small portion of android permissions were actually used out of the 100 permissions that were given. They also found that the INTERNET permission was used frequently and suggested that a mechanism that controls the usage of this permission needs to be developed. According to their results, 60% of the applications only use the INTERNET permission. Even if an application does not require the Internet permission, developers are forced to add this permission because the advertisements are published through the Internet. In addition to the Internet permission, many applications also enable SMS reading and SMS writing permissions. These permissions can be used without the knowledge of users and can cause higher bills. As a result, excessive permission

 

usage affects the users negatively. The study suggests that the Android permission mechanism should be improved; note that well-defined permissions will provide users with more control. Furthermore, their results show that users unconditionally allow permissions that are presented on their screen without knowing why the application requires them. However, the study does not provide a risk analysis or the security measures to be taken; it only reveals how the permission model is used, whether the applications in the same category use the same permissions and how many permissions are used by applications.

In [12], researchers downloaded mobile applications by crawling the Android Market; they used a statistical analysis on the byte-code level and matched the method calls in the Android API to the permissions in the manifest file. Although the study does not provide a user-focused security solution, it reveals which applications use too many or incomplete permissions.

Similar to the other studies, [13] researched the permission list. With the help of a developed software tool that is called Stowaway, authors inspected the permission file and source code of applications and exposed the applications that made API calls on the code level and did not define the necessary permissions in the manifest file. Furthermore, they identified the maximum number of permissions that an application may require and compared this to the permissions that already existed in the manifest file. According to their results, 35% of the applications used unnecessary permissions. They analyzed the reasons behind why the applications behave this way and investigated suspicious behaviors, method calls, and unnecessary permission usage. Finally, the authors created a permission map by matching the API calls to the permissions, with the success rate of 85%. The study makes an important contribution to mobile security research in terms of permission mapping.

In [14], researchers downloaded 10,000 applications from the Android Market, analyzed them by using data mining techniques, investigated which permissions were popular and were used more often, and researched how many of them were actually used and how they affected the users. To do this, the authors made use of information in the “mention” section of the Stackoverflow, an online information-sharing site for software developers. According to their analysis results, 40% of applications used unnecessary permissions. They found that there was an association between the popularity (which was found according to the number of mentions) of permissions and the misuse of them. More popular permissions were misused more. Although researchers do a detailed analysis of Android application, they do not provide a solution for the problem they mentioned.

In [15], authors investigated 204,040 applications that were downloaded from the top 5 popular applications in the Android Market. They developed a system that was called DroidRanger and identified malicious applications by employing their novel technique that creates permission based on behavioral footprints and filters the permissions that are unnecessary or cause harm. The developed system identified 211 applications as being malicious and contagious. These malicious applications infected 260,000 users within the 48 hours before Google removed them from the market. Researchers inspected the known malicious applications and the permissions that they used for the behavioral foot-printing process. After creating the footprints of malicious applications, they matched these to the scanned applications. The heuristic filtering technique was chosen for the applications that did not have a footprint. Heuristic filtering is the inferring process that identifies the unknown application's behaviors by employing the known suspicious behaviors of malicious applications. According to their results, the developed system delivered successful results that even popular anti-virus software could not match. Although their system produces successful results, their heuristic approach only works for certain behaviors that they defined and does not identify new malicious behaviors. Furthermore, the study does not incorporate mobile device users into the study.

Ref. [16] developed an application that runs on mobile devices and aimed to prevent security and privacy problems in the Android platform. The application is comprised of two main modules. The first module is called Mr. Hide. This module modifies the permissions in the manifest file. For instance, the INTERNET permission that is considered to be very dangerous is modified to a permission that limits it to a specific domain (Internet-URL(domain)). This provides more control over the permission mechanism. The second module is called Dr. Android. This module takes the permissions from the first module into account, modifies the byte-code of the application and rebuilds it with new permissions. Their solution does not make any changes at the operating system level. However, the first module causes an extra 10-50% overhead on the system, and the time to rebuild the application takes an average of one minute. The study was conducted on 19 popular free applications in different categories and 7 permissions that are considered to be dangerous by researchers. According to their results, the system successfully rebuilt the modified applications, and the applications ran without any problems. Because the developed system runs in the background as a service and rebuilds the code on the device, it causes overload, and modified applications run slower than their original ones. Additionally, the proposed system was not released to the public, which makes it unusable. Finally, the application dataset that was used in the study is not large enough and is limited to 19 popular free applications, which means it is difficult to tell whether it is successful or not.

Ref. [17] developed a system that analyzes the Android applications by using static and dynamic code analysis techniques. In the static analysis module, the permission file and the API function calls that correspond to permissions were inspected, and a report that shows the inconsistencies was created. The dynamic analysis module works on the application byte-code and gathers information about the behavior of the application. This study only reports the analysis and results and is not user-

 

focused. Furthermore, the cost of using dynamic analysis on applications decreases performance and limits the system's usability.

In [18], researchers developed Kirin, an application installer software that extends the Android permission system. The software replaces the default Android application installer, processes the permissions from the manifest file during the installation process and checks whether they match the predefined security rules. If the permissions match these rules, the application is installed. The security rules were created by utilizing security requirements engineering, and a security language that was named the Kirin Security Language (KSL) was developed. The fundamentals of this language were explained in the paper in detail and are out of this paper's scope.

Ref. [19] developed a statistical model (Probabilistic Generative model) that calculates the risk scores of the Android applications. Applications that have a high score were considered to be high-risk applications; applications that have a low score were considered to be low-risk applications. Risks were determined according to the permissions that were included in the manifest file. The developed model serves as a feedback mechanism for developers. Although they explain the theoretical details of risk scoring, a working system that makes of use of this statistical model does not exist. Furthermore, there are no details about how the cumulative score was calculated or what permissions caused the risks.

Ref. [20] developed a system that generates risk signals according to the permissions that exist within the application. When defining the risk signals, 121 malicious applications and 150,000 harmless applications were analyzed. The study focused on 26 critical permissions out of 122 Android permissions. Harmless and malicious applications were separated by employing the weighted Support Vector Machine (SVM). In addition, category information of applications was used to create risk signals. The reason behind utilizing category information was to understand whether the application was doing what it was supposed to do. This study is different from other studies in terms of risk assessment. However, it only calculates the risk scores according to certain data that were in hand.

### *A.3. Source Code Based Solutions*

Studies under this category provide solutions by processing the byte-code of applications. Some of these solutions make decisions based on static and dynamic analysis techniques and some modifies the byte-code and rebuilds the applications.

Ref. [21] developed a novel, state of the art open-source decompiler that was called DED and converted executable codes into source codes that could be read and understood by people. The study was conducted on 1100 popular free applications that were downloaded from the Android Market. Researchers applied the static analysis technique to examine the 21 million rows of source code that were decompiled from ".apk" image files by using their DED compiler. The results of their analysis revealed that applications posed a threat in terms of personal privacy because they used sensitive information, such as location and information that identifies a user. The developed DED compiler is the first and most successful source code generation utility, with a success rate of 94%, and is used widely by the research community whose work involves examining permissions by looking deep into the source code. This study's main contributions are its detailed security analysis on source code, its state of the art DED compiler and its success.

Ref. [22] developed a software framework that analyzes the applications on the code level by employing a static analysis technique, which automatically detects the leakage of sensitive information, such as one's phone number, contacts list, Wi-Fi data, and recorded sounds. The study was conducted on 24,350 applications and revealed that 7414 applications caused 57,299 potential information leaks. Their results showed that advertisement libraries caused most of these leaks. Although the software framework has the ability to generate detailed reports about leaks, an expert needs to look through the report to determine and approve of the existence of an information leak. Therefore, this developed tool targets security experts and facilitates the inspection process. However, users who do not have any knowledge of security cannot benefit from this tool. In addition, there is no information, download links or resources that explain how the software will be used.

In [23], researchers developed a security application that is called AppGuard and works on Android devices. This application has 3 main features. The first feature is its ability to generate security policies. It provides methods that generate policies to allow or limit the usage of GPS, camera, socket creation, and access to personal information. In regard to the second feature, it can rewrite the byte-code. The last feature is that it provides users with a user interface to create and customize application specific policies. The application utilizes the Java Reflection API to trace critical methods. This API assists developers in the intervention and examination of any method or variable without having to know the name of the method or variable. When a critical method is called, the parameters of policy methods are examined by using reflection. Whenever a security-related method is called, the monitoring interface calls the corresponding protection method. The developed application cannot work on already installed applications. In this situation, the application needs to be uninstalled and re-installed. The study was conducted on 13 applications that were downloaded from the Android Market. According to the test results, the system was overloaded because the applications were rebuilt on the device. For instance, the time that is needed to rebuild the popular game of Angry Birds is 45 seconds, while Instagram requires 66 seconds and WhatsApp Messenger requires 58 seconds. Furthermore, the overload on function calls that is caused by applying the security policies is 5%. In the event of removing the application from the device, applications revert to their previous states. This is the main disadvantage of the developed system. In May 2013,

 

Google removed this application from the Android Market, and the application has been hosted under the company's website and is only available for download from their website.

Ref. [24] developed a software framework that protects application security by rewriting the byte-code of the application according to the user's requirements. The system applies the static analysis technique and finds the methods that need to be changed according to the data that are supplied by the user. The resulting methods are replaced with the ones that were modified, and these modified methods are called. The study was conducted on 30 random applications that were chosen from 100 popular applications in the Android Market. The system intercepts the basic methods, such as Math.sqrt, url.OpenStream, and StringBuilder.append, and replaces them with customized methods. In addition, the authors added logging capability to the system and instantly tested the methods to see whether they were really called or not. According to their results, there were no issues during the application rewriting process, and the applications ran without any problems. However, the user must present the full list of methods that will be modified, including their return types, their parameters and the packages to which they belong, before the rewriting process. Otherwise, the system fails. Furthermore, the user should specify the behavior that he/she wants from the application and needs to be created by writing Java code. Considering these features, the system is unusable for normal users who do not have any technical knowledge about which methods cause security risks and need to be modified. The expected behavior is added by writing code, which means that this is not a good approach because it expects normal users to write code.

In [25], researchers developed a tool that is called Aurasium and aims to protect the privacy and security of Android devices. The developed system provides user-centric solutions. The tool has two components. The first component is the mechanism that adds management code and repacks the application. The second component communicates with the operating system, traces the applications and intercepts them. The tracing mechanism warns the user when a method is called that will compromise the user's security and privacy. It also asks the user whether it should prevent these method calls and saves the user's answer for future operations.

Ref. [26] developed an analysis tool that is called ScanDroid and is based on WALA, an open-source Java code analysis tool. The developed tool traces the data flow in applications and takes security measures. In addition, the tool reveals whether it is safe to run the applications with the permissions that they include. However, the tool only works on the provided byte-code and does not have the ability to unpack the ".apk" files. Furthermore, it has not been tested for Android applications in any official and unofficial Android Market.

### *A.4. Application/Service-Based Solutions*

We categorize the application/service-based solutions into two groups. The first group of solutions is installed on smart devices and examines the system as a background process, or this group is installed on desktop computers, and the analysis is performed either manually or through the use of an application. Mobile anti-virus software and data analysis tools are included in this group. The second group of solutions performs a security check and analysis in the cloud and provides Security as a Service (SaaS). Data are collected from devices, sent to the cloud, and analyzed on remote servers, and security reports are produced.

Ref. [27] developed a software framework to protect Android-based mobile devices from malicious applications. The developed software is installed on the devices, continuously monitors mobile devices, and classifies the applications as being malicious or harmless by employing Machine Learning techniques. Furthermore, the software gathers real-time data that are related to CPU usage, the sent Wi-Fi data amount, processes that are running in the background, and battery status. The gathered data are analyzed; threat assessment is completed; and the software presents a warning dialog to the user. In addition to the warning, options such as removing the application, stopping the application from running in the background, and locking the device are presented. In this study, the developed software must be installed on user devices. It also needs to run as a background process and gather data to perform an analysis. The data analysis process affects the user experience negatively. It exhausts the system and excessively consumes system resources due to the training period that is necessary for its learning process.

Ref. [28] designed a system called AppInspector that will examine applications that are submitted to popular application stores, will identify applications that have malicious behavior and will produce easy to understand reports that present users with potential privacy risks. This study differs from other studies in terms of producing security risk reports. However, the authors only drew a high level picture of their design, and the system was not developed.

In [28], researchers gathered 1260 malicious applications from Android Markets for a one-year period and analyzed them in a systematic way. The analysis was performed manually, and no software was developed. They classified the malicious applications into 3 main groups. Malicious fake applications that modify the popular applications, embed malicious code fragments and repack them form the first group. A total of 86% of malicious applications apply these processes. The second group of malicious applications works on the kernel level and embeds malicious code by exploiting the Android Platform's security holes. A total of 36% of malicious applications exploit these security holes. The third group of applications turns the mobile devices into bots and has command and control centers on remote servers. A total of 90% of malicious applications have this feature. Researchers published their results and the success rates of popular anti-virus applications, such as AVG, Trend Micro, and Norton anti-virus software. As a conclusion, the authors showed that these anti-virus applications had

 

a success rate of 79% in the best-case scenarios and 20% in the worst-case scenario. This study is a comprehensive study that is based on data gathering and analysis. The data that is gathered in this study are a result of one year of hard work and are included in the largest dataset (so far) that is dedicated to researchers. However, it is only an informative study that does not provide any security solutions.

Anti-virus software applications scan, slow, exhaust and overload the operating systems. Moreover, different anti-virus software applications use different techniques to identify malicious applications that produce different scanning results. [30] proposed the idea of "Anti-virus as a Service" to reduce the overload on the operating systems and to eliminate the differences between anti-virus applications. According to their concept of security service, files or applications that are needed to be scanned are sent to the cloud, and different anti-virus applications on different virtual machines inspect these files/applications in parallel and produce reports for users. The results of anti-virus applications are averaged, and a success rate is produced. Their results showed that the success rate was higher with this system than it was when a single anti-virus application was run, and overload on the operating system was reduced. However, running more than one anti-virus software causes conflicts, licensing problems and financial problems. In addition to these issues, companies generally settle with only one anti-virus company because company policy limits the usage of the proposed system. Although this study was conducted on desktop-based computers, it provides an idea of how to apply this type of solution to mobile platforms.

Ref. [31] developed a software service that is called Paranoid Droid and moves the security control mechanism to the cloud. The system saves the process fingerprints of mobile devices and encrypts and sends them to virtual machines on a remote computer. Then, they are subjected to parallel multi security checks. The developed prototype system has 2 types of security control mechanisms. In the first type, the system performs a dynamic code analysis that checks for code injection and buffer overflow. The second mechanism includes open-source anti-virus software that scans the files and performs a security check on them. Although the developed system provides a certain level of security, overload that is caused by the process of sending the fingerprints to remote machines is high and reduces the battery by 30%. If 3G connection is chosen when sending the data to the remote server, this will increase the cost, and the user will need to wait for a Wi-Fi connection to be active.

In [32], the authors developed a reputation-based security mechanism. A reputation score that was calculated in the cloud is presented before the user installs the application. If the reputation score is lower, the user is warned; otherwise, the application is installed. Reputation scores are based on the feedback of users who previously installed the application on their devices. The system that runs on the cloud that calculates the reputation score that is gathered from this feedback.

Ref. [33] developed a service that employs statistical methods and regular expressions to examine Android applications and produces online reports. The report contains information in three categories, including privacy leaks, access to personal information and suspicious method calls. Furthermore, libraries and API calls that the application uses are shown. Although the system produces detailed reports, it is not user-focused, and only security experts can understand the technical details of the report.

Ref. [34] is similar to the study in [30] and moves the virus scanning and finding procedure to the cloud. The system consists of 2 modules. The first module is comprised of anti-virus applications that run on virtual machines on a remote server. The second module runs on mobile devices and sends the files that need to be scanned to the cloud. The developed system separates the processes that require high memory and CPU from mobile devices and proposes a virus scanning and finding mechanism that has better performance.

### *B. Hardware Based Solutions*

Hardware-based security solutions are under development, and they intend to use the Trusted Platform Module (TPM). TPM is a technology that has been used in laptops and desktops for many years, and its aim is to maintain the integrity of computers and to check whether malicious software have made unwanted changes in the system. However, this module currently does not exist for mobile platforms. The Trusted Computing Group (TCG) has been working to define the TPM standards for mobile platforms. Although a hardware chip does not exist, there are software-based emulators that perform the job of TPM chips.

In [35], researchers developed a software-based emulator and a software framework that measures and attests to the integrity of the whole Android system in an efficient way. The developed system has the ability to measure the integrity of the Android platform and the installed applications. When measuring the integrity of applications, the integrity of all of the classes that reside in the application is measured separately. After the measurement process, data that are gathered from the mobile device are sent to a remote system for remote attestation. This is the first promising study that shows how TPM chips can be integrated into mobile platforms and how the security of mobile devices can be protected with this type of solution.

## IV. Discussions and Future Work

In this section, we report our findings on currently proposed Android based mobile security solutions based on our taxonomy. Followings are the criteria to make our

assessment:

- **Overhead:** Does the solution cause overhead on the users' mobile devices?
- **Usability:** Is the proposed solution user-friendly?
- **Independency:** Is proposed solution device dependent?
- **Availability:** Is proposed solution available to end-users?

Operating system-based solutions does not cause overhead since they are part of the operating system. However, users can only take advantage of this solution only if they install it on their devices. Therefore, the solution is device dependent. Additionally, these types of solutions are not publicly available for end-users and they are not official. Permission based solutions make changes in the permission list file of Android applications and they are device dependent. Modifying the permissions cause overhead. Although the solutions are usable, they are not publicly available to end-users. Source code based solutions runs on mobile devices and make changes in the source code of applications. These changes cause overhead on the mobile device. Although these are available to researchers, they are not publicly available to end-users. Application based solutions are installed on users' mobile devices that makes them device dependent. They also cause overhead since they continuously run in the background. These solutions are usable and publicly available to mobile device owners. Service based solutions are more effective than other solutions. The security is moved to cloud that makes them device independent. The end users can take advantage of these solutions since they are publicly available. Table 4 summarizes our findings.

Table 4. Assessment of Mobile Security Solutions

| | **Overhead** | **Usable** | **Independent** | **Available** |
|---|---|---|---|---|
| Operating System Based | No | Yes | No | Yes |
| Permission Based | Yes | Yes | No | No |
| Source Code Based | Yes | No | No | No |
| Application Based | Yes | Yes | No | Yes |
| Service Based | No | Yes | Yes | Yes |

As a future work, we propose a combination of service and permission based approach that can be designed and developed to better protect users' security. This kind of service will be device independent, available to end-users and researchers. Additionally, the proposed solution will not cause overhead on the mobile devices. The users will be able to make decisions about whether or not to install applications by searching the application in our system or by uploading the application file to our system. The design and implementation of this solution is left for future work.

## V. Conclusion

In the Android operating system, there is no security mechanism in place that checks whether applications are safe to install based on their malicious code or access to sensitive personal information. Applications are installed according to the users' free will, without any limitations or warnings. Considering the fact that users always blindly accept the requested permissions without knowing whether the application is safe or not, users' mobile devices become attractive targets of malicious applications, and the personal security and privacy of users are compromised.

In this study, we investigated the security architecture of Android, the most popular operating system that has the largest market share worldwide. Furthermore, we analyzed the security solutions that were developed to protect the privacy and security of Android users in detail and presented the weaknesses and strengths of these solutions. Our study revealed the shortcomings of mobile security solutions. We believe that it will shed light on the topic for the research community who will work on this subject and that it will provide a basis for the development of mobile security solutions. Additionally, we presented the taxonomy of mobile security to help researchers better understand current state and challenges.

In conclusion, the security mechanism of the Android operating system does not protect users effectively. Studies that aim to fill the gaps in terms of security fail to provide user-centric solutions. To provide solutions to existing security and privacy problems in the Android platform, there is an urgent need for user-centric, service-oriented solutions that examine whether an application is safe to install, accesses sensitive personal information and compromises the user's security and privacy. The solution must be device independent and must run on remote machines. Furthermore, it must help users become aware of the security implications of applications before installing them by providing easy to understand warnings and reports. Finally, the users must be able to customize the service rules according to their understanding of privacy.

## References

[1] Hugo Barra, Official Android Engineering team, https://plus.google.com/u/0/+HugoBarra/posts/R5YdRRyeTHM, 09-12, 2012. Last accessed: 13-10-2014.

[2] Us smartphone subscriber market share. http://www.comscore.com/Insights/Market-Rankings/comScore-Reports-August-2014-US-Smartphone-Subscriber-Market-Share. Last accessed: 13-10-2014.

[3] Smartphone operating system market share worldwide. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Last accessed: 13-10-2014.

[4] Official system architecture diagram of Android OS. http://developer.android.com/images/system-architecture.jpg. Last accessed: 13-10-2014.

[5] Official documentation of Android security overview. https://source.android.com/devices/tech/security/index.html. Last accessed: 13-10-2014.
[6] Official documentation of Android permissions. https://developer.android.com/reference/android/Manifest.permission.html. Last accessed:13-10-2014.
[7] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM. doi:10.1145/1755688.1755732.
[8] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Stephan Heuser, Ahmad-Reza Sadeghi, and Bhargava Shastry. Practical and lightweight domain isolation on android. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '11, pages 51–62, New York, NY, USA, 2011. ACM. doi:10.1145/2046614.2046624.
[9] A Shabtai, Y. Fledel, and Y. Elovici. Securing android-powered mobile devices using selinux. Security Privacy, IEEE, 8(3):36–44, May 2010. doi:10.1109/MSP.2009.144.
[10] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. In Computer Security Applications Conference, 2009. ACSAC '09. Annual, pages 340–349, Dec 2009. doi:10.1109/ACSAC.2009.39.
[11] David Barrera, H. Gunes ¸Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM. doi:10.1145/1866307.1866317.
[12] R. Johnson, Zhaohui Wang, C. Gagnon, and A Stavrou. Analysis of android applications' permissions. In Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on, pages 45–46, June 2012. doi:10.1109/SERE-C.2012.44.
[13] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM. doi:10.1145/2046707.2046779.
[14] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and Hao Chen. Asking for (and about) permissions used by android apps. In Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, pages 31–40, May 2013. doi:10.1109/MSR.2013.6624000.
[15] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In NDSS, 2012.
[16] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. 2012. Dr. Android and Mr. Hide: fine-grained permissions in android applications. In Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices (SPSM '12). ACM, New York, NY, USA, 3-14. doi:10.1145/2381934.2381938.
[17] Zhaohui Wang, Ryan Johnson, Rahul Murmuria, and Angelos Stavrou. Exposing security risks for commercial mobile devices. In Igor Kotenko and Victor Skormin, editors, Computer Network Security, volume 7531 of Lecture Notes in Computer Science, pages 3–21. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-33704-8_2.
[18] William Enck, Machigar Ongtang, and Patrick Mcdaniel. Mitigating android software misuse before it happens. Technical report, 2008. doi:i10.1.1.170.3793.
[19] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, pages 241–252, New York, NY, USA, 2012. ACM. doi:10.1145/2382196.2382224.
[20] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: A perspective combining risks and benefits. In Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, SACMAT '12, pages 13–22, New York, NY, USA, 2012. ACM. doi:10.1145/2295136.2295141.
[21] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In Proceedings of the 20th USENIX Conference on Security, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.
[22] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In Proceedings of the 5th International Conference on Trust and Trustworthy Computing, TRUST'12, pages 291–307, Berlin, Heidelberg, 2012. Springer-Verlag. doi:0.1007/978-3-642-30921-2_17.
[23] Michael Backes, Sebastian Gerling, Christian Hammer, Matteo Maffei, and Philipp von Styp-Rekowsky. 2013. AppGuard: Enforcing user requirements on android apps. In Proceedings of the 19th international conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13), Nir Piterman and Scott A. Smolka (Eds.). Springer-Verlag, Berlin, Heidelberg, 543-548. DOI=10.1007/978-3-642-36742-7_39.
[24] Benjamin Davis, Ben S, Armen Khodaverdian, and Hao Chen. I-arm-droid: A rewriting framework for in-app reference monitors for android applications. In Proceedings of the Mobile Security Technologies 2012, MOST 12. IEEE, 2012. doi:10.1.1.298.7191.
[25] Rubin Xu, Hassen Sa¨ıdi, and Ross Anderson. Aurasium: Practical policy enforcement for android applications. In Proceedings of the 21st USENIX Conference on Security Symposium, Security'12, pages 27–27, Berkeley, CA, USA, 2012. USENIX Association.
[26] Adam P. Fuchs, Avik Chaudhuri, and Jeffrey S. Foster. Scandroid: Automated security certification of android applications, 2009. doi:10.1.1.148.2511.
[27] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. Andromaly: a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, 38(1):161–190, 2012. doi:10.1007/s10844-010-0148-x.
[28] Peter Gilbert, Byung-Gon Chun, Landon P. Cox, and Jaeyeon Jung. Vision: Automated security validation of mobile apps at app markets. In Proceedings of the Second International Workshop on Mobile Cloud Computing and Services, MCS '11, pages 21–26, New York, NY, USA, 2011. ACM. doi:10.1145/1999732.1999740.
[29] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In Proceedings of the 2012 IEEE Symposium on Security and Privacy,

SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society. doi:10.1109/SP.2012.16.

[30] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Cloudav: N-version antivirus in the network cloud. In Proceedings of the 17th Conference on Security Symposium, SS'08, pages 91–106, Berkeley, CA, USA, 2008. USENIX Association.

[31] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: Versatile protection for smartphones. In Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC '10, pages 347–356, New York, NY, USA, 2010. ACM. doi:10.1145/1920261.1920313.

[32] Welderufael Berhane Tesfay, Todd Booth, and Karl Andersson. Reputation based security model for android applications. In Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM '12, pages 896–901, Washington, DC, USA, 2012. IEEE Computer Society. doi:10.1109/TrustCom.2012.236.

[33] L. Batyuk, M. Herpich, S.A Camtepe, K. Raddatz, A-D. Schmidt, and S. Albayrak. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications. In Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on, pages 66–72, Oct 2011. doi:10.1.1.388.4511.

[34] Jon Oberheide, Kaushik Veeraraghavan, Evan Cooke, Jason Flinn, and Farnam Jahanian. Virtualized in-cloud security services for mobile devices. In Proceedings of the First Workshop on Virtualization in Mobile Computing, MobiVirt '08, pages 31–35, New York, NY, USA, 2008. ACM. doi:10.1145/1622103.1629656.

[35] Mohammad Nauman, Sohail Khan, Xinwen Zhang, and Jean-Pierre Seifert. Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform. In Alessandro Acquisti, SeanW. Smith, and Ahmad-Reza Sadeghi, editors, Trust and Trustworthy Computing, volume 6101 of Lecture Notes in Computer Science, pages 1–15. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13869-0_1.

## Authors' Profiles

**Asim S. Yuksel** obtained his B.S. degree in computer engineering from Ege University, Izmir, Turkey in 2006. He completed his Masters degree in computer science from Indiana University School of Informatics and Computing, Indiana, USA in 2010. He is currently a Ph.D. candidate in Istanbul University Department of Computer Engineering. He has been working as a research assistant in Suleyman Demirel University Department of Computer Engineering since 2012. His research interests include mobile security and privacy, social network security and privacy, object oriented programming and object oriented analysis and design.

**Abdul H. Zaim** received his B.S. degree (with honor) in computer science and engineering from Yildiz Technical University, Istanbul, Turkey, in 1993, his MSc degree in computer engineering from Bogazici University, Istanbul, Turkey, in 1996, and his PhD degree in electrical and computer engineering from North Carolina State University, Raleigh, in 2001. As a teaching assistant and lecturer, he taught several courses at Istanbul and Yeditepe Universities between 1993 and 1997. In 1998 to 1999, he was with Alcatel, Raleigh, North Carolina. Between 2001 and 2002 he worked at MCNC, Raleigh, NC. He worked as a full professor for Istanbul University until 2010. He is currently a professor in Istanbul Commerce University Computer Engineering Department and director of Information Technology Application and Research Center. His research interests include computer performance evaluation, satellite and high-speed networks, network protocols, optical networks, and computer network design.

**Muhammed A. Aydin** obtained his B.S. degree in computer engineering from Istanbul University in Istanbul, Turkey in 2001. He completed his Masters degree in computer engineering from Istanbul Technical University, Istanbul, Turkey in 2005. He received his Ph.D. degree in computer engineering from Istanbul University, Istanbul, Turkey in 2009. He has been working as an assistant professor in Istanbul University Department of Computer Engineering since 2009. His research interests include optical networks, network security, information security and cryptography.

**How to cite this paper:** Asim S. Yuksel, Abdul H. Zaim, Muhammed A. Aydin,"A Comprehensive Analysis of Android Security and Proposed Solutions", IJCNIS, vol.6, no.12, pp.9-20, 2014. DOI: 10.5815/ijcnis.2014.12.02

 