International Journal of Security and Its Applications
Vol. 10, No. 9 (2016), pp.165-174
http://dx.doi.org/10.14257/ijsia.2016.10.9.17

Identifying and Analyzing Security Risks in Android Application Components

Ming Zhang

Department of Computer Science, Mianyang Polytechnic, Mianyang, China
372616951@qq.com

Abstract

Android operating system provides a rich inter-application message passing system. The mechanism encourages inter-application collaboration and reduces developer burden by facilitating component reuse. Unfortunately, message passing is also an application attack surface. The content of messages can be sniffed, modified, stolen, or replaced, which can compromise user privacy. In this paper, we examine Android application interaction and identify security risks in application components. We provide a method that detects application communication vulnerabilities. And the effectiveness of the method is verified by experiments.

Keywords*: Android security; Android components; Intents; message communication*

1. Introduction

Android and the applications (apps) that run on the platform have gained tremendous popularity recently [1]. Android phone manufacturers support third-party application developers by providing development platforms and software stores (*e.g.*, Android Market) where developers can distribute their applications.

Android's application communication model further promotes the development of rich applications. Android developers can leverage existing data and services provided by other applications while still giving the impression of a single, seamless application [2]. The communication model reduces developer burden and promotes functionality reuse. Through dividing applications into components and providing a message communication system so that components can communicate within and across application boundaries, Android applications can pass message with each other.

However, Android's message communication system would become an attack surface if used incorrectly. In this paper, the risks of Android message passing and identify insecure developer practices are discussed. If a message sender does not correctly specify the recipient, then an attacker could intercept the message and compromise its confidentiality or integrity. If a component does not restrict who may send it messages, then an attacker could inject malicious messages into it. So it is important to write secure applications that do not leak or alter user data.

The main goals and contributions of this paper are that we examine Android application interaction and identify security risks in application components. Android application communication vulnerabilities were detected by testing tools. We analyzed 20 Android applications and found 34 vulnerabilities in 12 of the applications. Most of these vulnerabilities stem from Intents, which can be used for both intra- and inter-application communication, so we provide recommendations for changing Android to help developers distinguish between internal and external messages.

This work is organized as follows. In Section 2 we introduce the basic components and their permissions of Android. In Section 3 security risks in Android based on Intent are given. In Section 4, we evaluate and analyze Android application security risks based on Android application components. In Section 5 we conclude this paper.

ISSN: 1738-9976 IJSIA

2. Android Overview

Android is a mobile operating system (OS) currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's kernel is based on one of the Linux kernel's long-term support (LTS) branches. On top of the Linux kernel, there are the middleware, libraries and APIs written in C, and application software running on an application framework which includes Java-compatible libraries. Development of the Linux kernel continues independently of other Android's source code bases. The architecture of Android is shown in Figure 1.

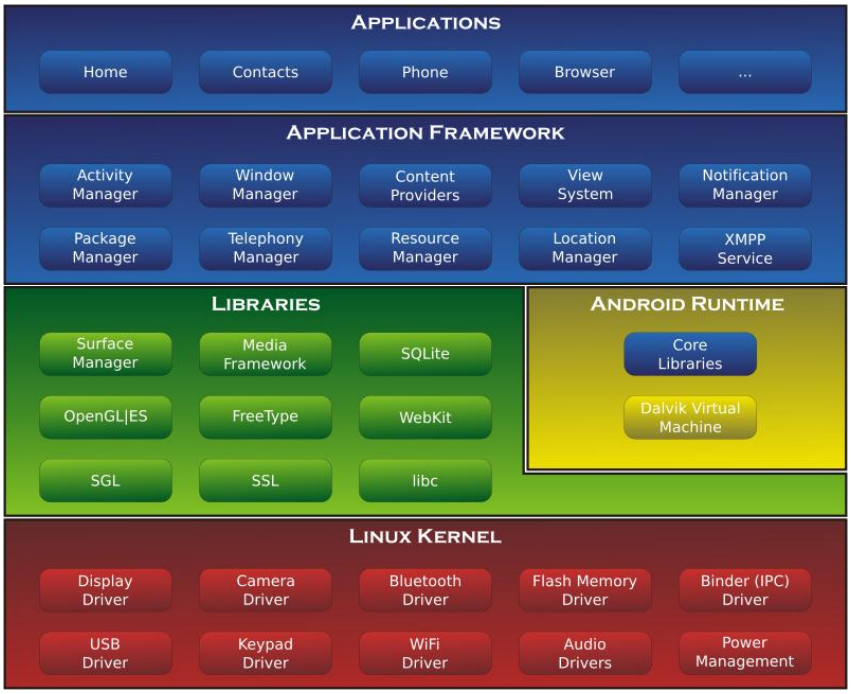

Figure 1. Android's Architecture Diagram

2.1. Components

An Intent is a message that declares a recipient and optionally includes data; an Intent can be thought of as a self-contained object that specifies a remote procedure to invoke and includes the associated arguments [3]. Applications use Intents for both inter-application communication and intra-application communication. Additionally, the operating system sends Intents to applications as event notifications. Some of these event notifications are system-wide events that can only be sent by the operating system. We call these messages system broadcast Intents [4].

Intents can be used for explicit or implicit communication. An explicit Intent specifies that it should be delivered to a particular application specified by the Intent, whereas an implicit Intent requests delivery to any application that supports a desired operation. In other words, an explicit Intent identifies the intended recipient by name, whereas an implicit Intent leaves it up to the Android platform to determine which application(s) should receive the Intent. For example, consider an application that stores contact information. When the user clicks on a contact's street address, the contacts application needs to ask another application to display a map of that location. To achieve this, the contacts application could send an explicit Intent directly to Google Maps, or it could send an implicit Intent that would be delivered to any application that says it provides mapping functionality. Using an explicit Intent guarantees that the Intent is delivered to

the intended recipient, whereas implicit Intents allow for late runtime binding between different applications [5].

Android defines four types of components:

Activities provide user interfaces. Activities are started with Intents, and they can return data to their invoking components upon completion. All visible portions of applications are Activities.

Services run in the background and do not interact with the user. Downloading a file or decompressing an archive is examples of operations that may take place in a Service. Other components can bind to a Service, which lets the binder invoke methods that are declared in the target Service's interface. Intents are used to start and bind to Services.

Broadcast Receivers receive Intents sent to multiple applications. Receivers are triggered by the receipt of an appropriate Intent and then run in the background to handle the event. Receivers are typically short-lived; they often relay messages to Activities or Services. There are three types of broadcast Intents: normal, sticky, and ordered. Normal broadcasts are sent to all registered Receivers at once, and then they disappear. Ordered broadcasts are delivered to one Receiver at a time; also, any Receiver in the delivery chain of an ordered broadcast can stop its propagation. Broadcast Receivers have the ability to set their priority level for receiving ordered broadcasts. Sticky broadcasts remain accessible after they have been delivered and are re-broadcast to future Receivers.

Content Providers are databases addressable by their application-defined URIs. They are used for both persistent internal data storage and as a mechanism for sharing information between applications.

Intents can be sent between three of the four components: Activities, Services, and Broadcast Receivers. Intents can be used to start Activities; start, stop, and bind Services; and broadcast information to Broadcast Receivers. All of these forms of communication can be used with either explicit or implicit Intents. By default, a component receives only internal application Intents.

2.2. Android Permissions

For a Service, Activity or Broadcast Receivers to receive Intents, it must be declared in an XML-formatted file named AndroidManifest.xml. A component is considered exported, or public, if its declaration sets the EXPORTED flag or includes at least one Intent filter. Exported components can receive Intents from other applications, and Intent filters specify what type of Intents should be delivered to an exported component.

Android determines which Intents should be delivered to an exported component by matching each Intent's fields to the component's declaration. An Intent can include a component name, an action, data, a category, extra data, or any subset thereof. A developer sends an explicit Intent by specifying a recipient component name; the Intent is then delivered to the component with that name. Implicit Intents lack component names, so Android uses the other fields to identify an appropriate recipient.

Multiple applications can register components that handle the same type of Intent. This means that the operating system needs to decide which component should receive the Intent. Broadcast Receivers can specify a priority level (as an attribute of its Intent filter) to indicate to the operating system how well-suited the component is to handle an Intent. When ordered broadcasts are sent, the Intent filter with the highest priority level will receive the Intent first. Ties among Activities are resolved by asking the user to select the preferred application. Competition between Services is decided by randomly choosing a Service.

It is important to note that Intent filters are not a security mechanism. A sender can assign any action, type, or category that it wants to an Intent, or it can bypass the filter system entirely with an explicit Intent [6]. Conversely, a component can claim to handle

any action, type, or category, regardless of whether it is actually well-suited for the desired operation.

At the core of the Android security model is a permission-based system that by default denies access to features or functionality that could negatively impact the user experience, the system, or other applications installed on the device [7]. Examples of these features are sending messages or making phone calls, which may incur monetary cost to the user; keeping the device screen on or accessing the vibrator, which could result in battery drain; and reading the user's address book which could result in privacy violations.

To make use of the restricted functionality, Android requires application developers to declare which of the restricted features are intended to be used by their application. Failure to declare a particular permission will result in the related system call or inter-process communication being denied by Android. There are currently 110 items of functionality which are identified as requiring an explicit permission in order for Android to grant access [8]. These permissions control access to network and GPS functions, personal information, system hardware and settings, and many other device features. However, Android is designed such that any third party application can define new functionality and make that specific functionality available to other applications based on developer-defined permissions. In the case of developer-defined permissions, Android enforces that both the caller and callee applications have matching permissions to allow the IPC to take place.

3. Security Risks in Android based on Intent

Android platform provides a security model [9], all applications are treated as potentially malicious under this model. Each application runs in its own process with a low-privilege ID, and applications can only access their own files by default. These isolation mechanisms aim to protect applications with sensitive information from malware. Despite their default isolation, applications can optionally communicate via message passing. Communication would become an attack vector [10]. The security challenges of Android communication from the perspectives of Intent senders and Intent recipients were examined in this section.

When an application sends an implicit Intent, there is no guarantee that the Intent will be received by the intended recipient. So a malicious application can easily intercept an implicit Intent by declaring an Intent filter with all of the actions, data, and categories listed in the Intent. The malicious application then gains access to all of the data in any matching Intent, unless the Intent is protected by a permission that the malicious application lacks. Interception can also lead to control-flow attacks like denial of service.

There are different kinds of security risks which are caused by mounting on Intents intended for Broadcast Receivers, Activities, and Services [11].

3.1. Activity Hijacking

In an Activity hijacking attack, a malicious Activity is launched in place of the intended Activity. The malicious Activity registers to receive another application's implicit Intents, and it is then started in place of the expected Activity, the entire process is shown in Figure 2.

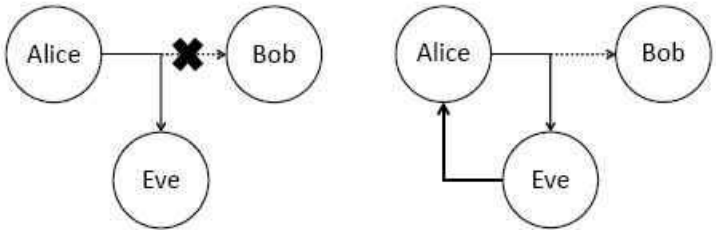

Figure 2. Activity Hijacking

In Figure 2, Alice accidentally starts Eve's component instead of Bob's, and Eve returns a malicious result to Alice, but Alice thinks the result comes from Bob.

In the simplest form of this attack, the malicious Activity could read the data in the Intent and then immediately relay it to a legitimate Activity. In a more sophisticated active attack, the hijacker could spoof the expected Activity's user interface to steal user-supplied data.

If an Activity hijacking attack is successful, the victim component may be open to a secondary false response attack. Some Activities are expected to return results upon completion. In these cases, an Activity hijacker can return a malicious response value to its invoker. If the victim application trusts the response, then false information is injected into the victim application.

3.2. Broadcast Theft

Broadcasts can be vulnerable to eavesdropping or active denial of service attacks, the process is shown in Figure 3.

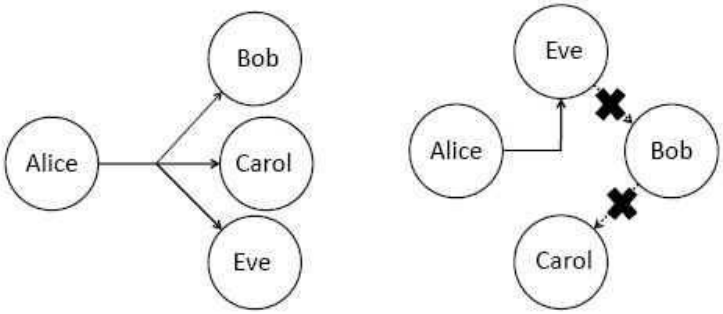

Figure 3. Broadcast Eavesdropping

Normally, expected recipients Bob and Carol receive the Intent, but so does Eve. Then, Eve steals the Intent and prevents Bob and Carol from receiving it.

Eavesdropping is a risk whenever an application sends a public broadcast. A malicious Broadcast Receiver could eavesdrop on all public broadcasts from all applications by creating an Intent filter that lists all possible actions, data, and categories. There is no indication to the sender or user that the broadcast has been read.

Furthermore, an active attacker could launch denial of service or data injection attacks on ordered broadcasts. Ordered broadcasts are serially delivered to Receivers in order of priority, and each Receiver can stop it from propagating further. If a malicious Receiver were to make itself a preferred Receiver by registering itself as a high priority, it would receive the Intent first and could cancel the broadcast. Non-ordered broadcasts are not vulnerable to denial of service attacks because they are delivered simultaneously to all Receivers. Ordered broadcasts can also be subject to malicious data injection. As each Receiver processes the Intent, it can pass on a result to the next Receiver; after all Receivers process the Intent, the result is returned to the sending component. A malicious

Receiver can change the result, potentially affecting the sender and all other receiving components.

3.3. Broadcast Theft

Service hijacking occurs when a malicious Service intercepts an Intent meant for a legitimate Service. The result is that the initiating application establishes a connection with a malicious Service instead of the one it wanted. The malicious Service can steal data and lie about completing requested actions. Unlike Activity hijacking, Service hijacking is not apparent to the user because no user interface is involved. When multiple Services can handle an Intent, Android selects one at random; the user is not prompted to select a Service.

As with Activity hijacking, Service hijacking can enable the attacker to spoof responses. Once the malicious Service is bound to the calling application, then the attacker can return arbitrary malicious data or simply return a successful result without taking the requested action. If the calling application provides the Service with callbacks, then the Service might be able to mount additional attacks using the callbacks.

4. Security Risks Evaluation

In our experiment, the top 50 popular paid applications and top 50 popular free applications on the Android Market were selected. We gave warning rates of applications and discuss common application weaknesses. Those warnings about potential security were issues, and manual review is needed to determine the functionality of the Intent or component and decide whether the exposure can lead to a severe vulnerability. We manually examined 20 applications to check warnings, evaluate our method, and detect vulnerabilities.

4.1. Automated Security Risks Analysis

We detected a total of 1414 exposed surfaces across 100 applications. There were 401 warnings for exposed components and 1013 warnings for exposed Intents. The fraction of sent implicit Intents is shown in Figure 4. From Figure 4 we can see that on average, about 40% are implicit.

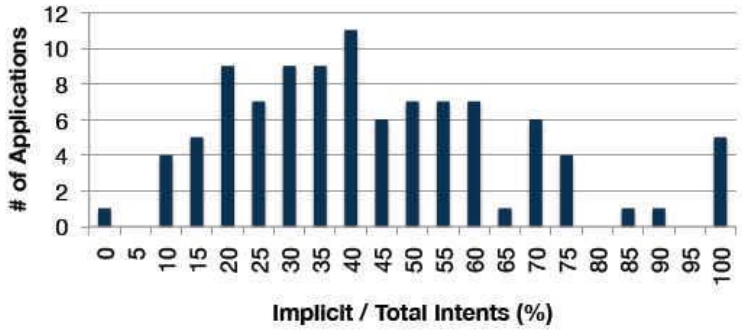

Figure 4. The Percentage of Implicit Intents Out of Total Intents for Each Application

In Figure 5, we show the frequency of exposed components out of the total number of components for each application, separated by component type. 50% of Broadcast Receivers are exposed, and most applications expose less than 40% of Activities to external applications.

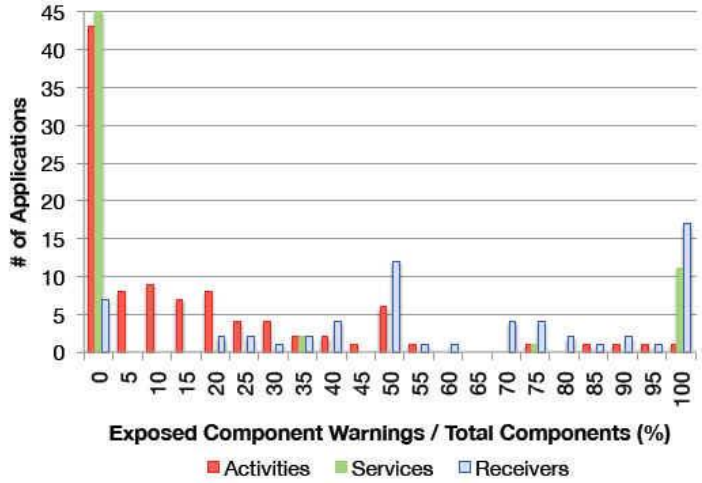

Figure 5. Exposed Component Warnings/Total Components (%)

A warning for an application's primary launcher Activity does not be generated by our method, consequently, many applications that show zero exposed Activities warnings may have one public launcher Activity. There is no clear distribution for the exposure of Services because few applications have multiple Service components.

The breakdown of warnings by type is also shown in Figure 6. Intuitively, there is more Intent communication between Activities so there are more exposure warnings for Activity-related Intents than Broadcast- and Service-related Intents combined.

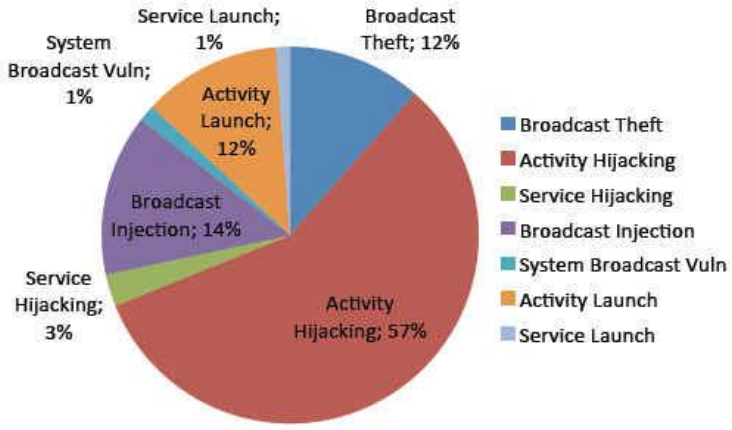

Figure 6. Breakdown of Warnings by Type

Over 56% of applications have a Broadcast Receiver that may be vulnerable to a Broadcast injection attack. Broken down by number of exposed Receivers per application, which is shown in Figure 7, we can see that most applications expose one or two Receivers, if any.

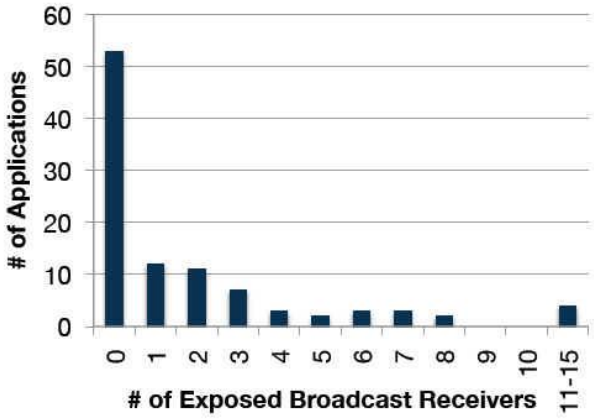

Figure 7. The Number of Broadcast Receivers with Warnings per Application

13% of applications have a public Receiver that accepts a system Broadcast action but does not check that the Intent actually contains that action. 57% of applications have at least one Activity that may be vulnerable to a malicious Activity launch. The other 43% only expose the main launching Activity. We display the break down of these malicious Activity launch warnings by number of malicious launch warnings per application, which is shown in Figure 8.

On average, applications have one exposed Activity in addition to the launch Activity. We can also see a handful of applications expose 11 to 20 Activities and can benefit from further investigation. Finally, 14% of applications have at least one Service that may be vulnerable to malicious Service launches.

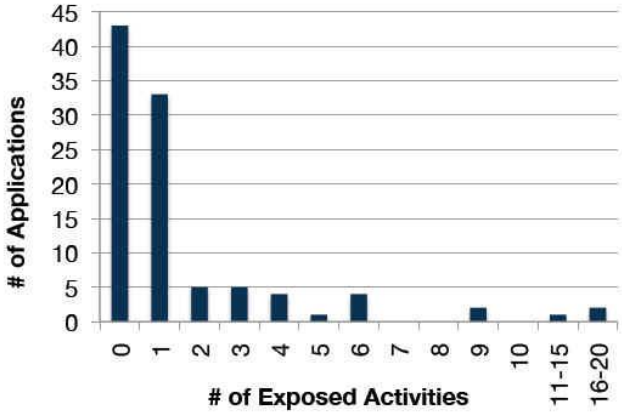

Figure 8. The Number of Activities with Warnings per Application

Our above analysis results indicate that Broadcast- and Activity- related Intents play a large role in application exposure.

4.2. Manual Security Risks Analysis

In this section, 20 applications from the 100 mentioned earlier were selected randomly, and then we manually inspected warning for these applications to evaluate how many warnings correspond to vulnerabilities. 181 warnings for the 20 applications were generated. We manually reviewed all of them and classified each warning as a vulnerability, not a vulnerability, or undetermined. A vulnerability is defined as a component or Intent that exposes data or functionality that can be detrimental to the user.

We further divide vulnerabilities into two types: dangerous vulnerabilities that do not rely on user interaction and spoofing vulnerabilities that might occur if the user is tricked.

In order to detect vulnerabilities, we reviewed the disassembled code of the application components with warnings. When necessary, we built "attack" code to confirm or disprove vulnerabilities. Of the 181 warnings, we discovered 20 definite vulnerabilities, 14 spoofing vulnerabilities, and 16 common, unintentional bugs. Of the 20 applications examined, 9 applications contain at least 1 definite vulnerability and 12 applications have either definite or spoofing vulnerabilities. It demonstrates the prevalence of insecure Intent communication. The number of vulnerabilities and warnings for each category is shown in Table 1.

Our method has an overall vulnerability and bug detection rate of 27.6%. Broken down by Unauthorized Intent Receipt vulnerabilities/bugs and Intent Spoofing vulnerabilities/bugs, it has a rate of 22.6% and 38.6%, respectively. As shown in Table 1, Activity hijacking has the highest number of false positives, with a lower detection rate of 15.2%. Examining only the broadcast-related vulnerabilities (theft, injection, and system broadcasts without action check), our method has a detection rate of 61.2%.

Table 1. The Number of Vulnerabilities and Bugs Found from the Warnings

Type of Exposure	Definite Vulnerabilities	Spoofing Vulnerabilities	Unintentional Bugs	Total Warnings	Vuln. and Bug Percentage
Broadcast Theft (without data)	1	0	6	10	70．0%
Broadcast Theft (with data)	2	0	2	4	100%
Activity Hijacking (without data)	2	11	0	91	14.3%
Activity Hijacking (with data)	0	3	0	14	21.2%
Service Hijacking (without data)	0	0	1	5	20.0%
Service Hijacking (with data)	0	0	0	0	--
Broadcast Injection (without data)	10	0	2	24	50.0%
Broadcast Injection (with data)	3	0	0	7	42.9%
Activity Launch (without data)	0	0	0	9	0.0%
Activity Launch (with data)	0	0	2	10	20.0%
Service Launch (without data)	0	0	0	2	0.0%
Service Launch (with data)	1	0	0	1	100．0%

5. Conclusion

While the Android message passing mechanism promotes the creation of rich applications, it also introduces the potential security risks if developers do not take precautions. In this paper, inter-application communication in Android was examined and several classes of potential attacks on applications were presented. Inter-application communication would put an application at risk of Broadcast theft, data theft, result modification, Broadcast injection, and Activity and Service hijacking. We provide a method that developers can use to find these kinds of vulnerabilities. Through the method, We analyzed 100 applications and verified our findings manually with 20 of those applications. 12 applications of the 20 applications were identified to at least one

vulnerability. The analysis result shows that applications can be vulnerable to attack and that developers should take precautions to protect themselves from these attacks.

References

[1] C. Hu and I. Neamtiu, "Automating GUI testing for Android applications", Proceedings of the 6th International Workshop on Automation of Software Test, **(2011)**.
[2] A. Israeli and G. Feitelson, "The Linux kernel as a case study in software evolution", Journal of Systems & Software, vol. 3, no. 83, **(2010)**.
[3] S. Raemaekers, A. Deursen and J. Visser, "Measuring software library stability through historical version analysis", Proceedings of IEEE International Conference on Software Maintenance, **(2012)**.
[4] A. Dresnos, "Android: Static Analysis using Similarity Distance", Proceedings of the 45th Hawaii International Conference on System Sciences, **(2012)**.
[5] D. Hou and X. Yao, "Exploring the Intent Behind API Evolution: A Case Study", Proceedings of the 18th Working Conference on Reverse Engineering, **(2011)**.
[6] L. Martie, V. Palepu, H. Sajnani and C. Lopes, "Trendy Bugs: Topic Trends in the Android Bug Reports", Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, **(2012)**.
[7] E. Shihab, Y. Kamei and P. Bhattacharya, "Mining Challenge 2012: The Android Platform", Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, **(2012)**.
[8] W. Enck, M. Ongtang and P. D. McDaniel, "Understanding Android Security", IEEE Security and Privacy, vol. 1, no. 7, **(2009)**.
[9] W. Enck, D. Octeau, P. McDaniel and S. Chaudhuri, "A Study of Android Application Security", Proceedings of the 20th USENIX Security Symposium, **(2011)**.
[10] M. Zibran, F. Eishita and C. Roy, "Useful, but Usable? Factors Affecting the Usability of APIs", Proceedings of the 18th Working Conference on Reverse Engineering, **(2011)**.
[11] M. Zibran, "What Makes APIs Dificult to Use?", International Journal of Computer Science and Network Security, vol. 4, no. 8, **(2008)**.

