Comparative overview of basic cybervulnerabilities of mobile applications for android operating system

S. Semenov, T. Shypova, O. Movchan
Department «Computer Engineering and programming»,
National Technical University «Kharkiv Politechnical Institute»
Kharkiv, Ukraine
s_semenov@ukr.net

Abstract: With the market for mobile applications for Android platform constantly growing and more security-dependent tasks moving to mobile platforms, security of Android applications is a major concern for developers and users. In this paper, an overview of Android operating system security model is given. Components of Android application are studied, with special attention given to mechanisms of Inter-process communication via Intents. An overview of basic vulnerabilities of Android applications and vulnerabilities of IPC in Android applications is performed. Recommendations for avoiding described vulnerabilities are given.

Keywords: Android, security, inter-process communication, vulnerability, attack vectors.

I. INTRODUCTION

At the current moment of time, mobile devices are extremely widely used and the numbers of mobile device users is growing more and more. In its core, a modern mobile device is a portable computer with telephony capabilities. And, as is the case with regular computers, functionality of mobile device is limited to software installed on the device. Today mobile applications are used for various tasks like social media, communication and entertainment. However, more and more security-dependent tasks, for example banking and enterprise management, are going mobile as well. And it is important to provide the necessary security level to protect the system from attacks.

The purpose of the paper is to present the overview of basic vulnerabilities in applications for Android platform with regard to the architecture of Android applications and the programming language used in development, with special attention towards vulnerabilities in Inter-process communication mechanisms as a primary source of application vulnerabilities. It should be noted that this paper focuses on individual vulnerabilities of the application under testing rather than overall testing methodology.

II. ANDROID SECURITY MODEL OVERVIEW

In order to analyze vulnerabilities in Android applications, it is required to have the knowledge of the security system provided by the OS. The security system that is enforced by Android can be described as a two-tier system.

Android, at its core, relies on one of the security features provided by Linux kernel – running each application as a separate process with its own set of data structures and preventing other processes from interfering with its execution [1, 2]. Parts of the system are also separated into distinct identities. Linux thereby isolates applications from each other and from the system. This mechanism is called sandbox and it is displayed in figure 1.

More detailed security mechanism of “permissions” allows finer control of access of application to device and OS features [3]. A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely affect the user experience or any data on the device. To make use of protected features of the device, you must include one or more <uses-permission> tags in your app manifest.

If your app lists *normal* permissions in its manifest (that is, permissions that don't pose much risk to the user's privacy or the device's operation), the system automatically grants those permissions.

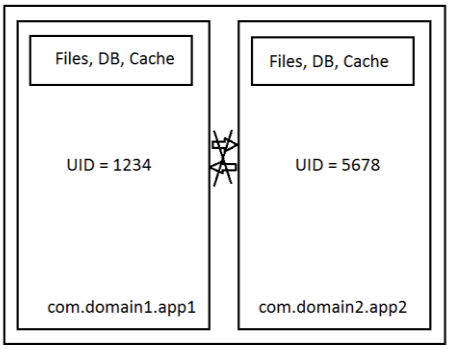

Figure 1 – Android application sandbox

If your app lists *dangerous* permissions in its manifest (that is, permissions that could potentially affect the user's privacy or the device's normal operation), the system asks the user to explicitly grant those permissions.

III. ANDROID APPLICATION STRUCTURE OVERVIEW

Android applications are developed using Java programming and Android SDK in a majority of cases, with the exception of games and other CPU-intensive

apps, where Android NDK and native languages like C and C++ are used. Considering the fact that most Android applications are built using Java and Android SDK, only Android SDK elements will be overviewed.

There are four main components of the Android application: activities, BroadcastReceivers, ContentProviders and services. They communicate between each other using messages called Intents [4].

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface.

Intents are messages through which other application components (activities, services, and Broadcast Receivers) are activated. They can be thought of as messages stating which operations/actions need to be performed. Intents can be explicit and implicit. Explicit intents specify the component to start by name (the fully-qualified class name). Explicit intent are usually used to start a component in the same app, because the class name of the activity or service that is intended to start, is known. Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

A service is an application component that can perform long-running operations in the background for an application. It does not have a UI component to it, but it executes tasks in the background. Other applications can be running in the front while services will be active behind the curtain even after the user switches to a different application component or application.

Content providers provide applications with a means to share persistent data. A content provider can be thought of as a repository of data, and different applications can define content providers to access it. Providers and provider clients enable a standard interface to share data in a secure and efficient manner. When an application wants to access data in a content provider, it does so through ContentResolver.

Component can be declared exported (public) in order to be accessible to other applications. This can be done by setting the EXPORTED flag in the manifest or by including at least one IntentFilter. After being declared exported, component can be launched via an implicit Intent that confines to an IntentFilter, or via an explicit Intent, which bypasses IntentFilters entirely. This mechanism of launching exported components enables many attack surfaces for basing attack on.

IV. Android application basic attack surfaces

Considering the platform and the language used in development of the application for Android platform, vulnerabilities of Android applications can be divided into following:

[1] general vulnerabilities of mobile and web applications;

[2] vulnerabilities specific to the Android platform.

General vulnerabilities are vulnerabilities that do not feature Android specific application elements as an attack vector. Vulnerabilities in this category are quite common in mobile and web applications and are based on application architecture flaws or development bad habits. The list of these vulnerabilities consists of, but not limited to:

[3] using raw user input as query parameters;

[4] weak or no cryptography on sensitive user data;

[5] insecure data storage;

[6] poor authentication and authorization controls;

[7] security decisions vie untrusted inputs;

[8] logging sensitive user information.

One of the more common mobile vulnerabilities, insecure data storage vulnerability is a result of storing sensitive user information in an insecure storage like a database on the device. Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data-stores on the device, which is usually never the case. Filesystems are easily accessible for malicious users. It is possible to extract the data from the filesystem using special tools. Insecure data storage can result in data loss for one or more users. Common valuable pieces of data seen stored include usernames, authentication tokens, passwords, cookies, personal information like date of birth, address, credit card data and application data like logs and configuration files.

According to OWASP, in order to prevent insecure data storage vulnerabilities, it is recommended to avoid storing data on the device unless necessary [5]. When it is impossible to avoid storing sensitive data on the device, the following actions are advised for Android platform:

[9] force encryption on local file storages with setStorageEncryption;

[10] use manual encryption for data on SD card;

[11] ensure any shared preferences properties are not MODE_WORLD_READABLE unless explicitly required for information sharing between app;

[12] avoid hardcoding encryption or decryption keys when storing sensitive information.

Another common class of vulnerabilities, security vulnerabilities via untrusted inputs exist when application has no validation of inputs in secure method realizations. Developers can assume that only high-level user can call specific secure method and, because of it, do not validate status of the caller. This allows attacker to gain access to secure functionality or even gain higher-level permissions.

In order to avoid these vulnerabilities, it is advised to follow the rules:

[13] if IPC is required, only white-listed applications should have access to the API and mechanisms;

[14] all input parameters, that are received from IPC entry points, like Intents and broadcasts, should undergo thorough validation, especially their origin;

[15] if possible, passing of sensitive data using IPC should be avoided.

Android specific vulnerabilities are vulnerabilities that feature Android specific elements and OS features as attack vector. The majority of these vulnerabilities are located in IPC mechanisms of the system [6]. Attacks that target vulnerabilities in IPC using mechanism of Intents are:

[16] Intent interception;

[17] Intent spoofing.

Intent interception involves a malicious app receiving an intent that was not intended for it. This can cause a leak of sensitive information, but more importantly, it can result in the malicious component being activated instead of the legitimate component. The attacks are:

[18] Broadcast Theft;

[19] Activity hijacking;

[20] Service hijacking.

Broadcast Theft is an attack that targets vulnerability that is present when an application uses implicit Intent to send data. Any component is able to intercept an implicit Intent so, if a malicious component is able to intercept the intent, then it can access the data. An attacker could perform a denial-of-service attack on the Ordered Broadcasts, since an Intent can only be spread on them if the first component receiving the Intent to uses it for output. Additionally, it could be used to perform Man-in-the-Middle attacks with its subsequent data injection on the spread Intents.

By taking advantage of Activity hijacking vulnerability, a malicious Activity is launched instead of the expected one, so the user will be in a wrong application without being aware. This happens when the change of an Activity depends on an implicit Intent. The attacker registers a more accurate Intent Filter and controls it. The presence of this vulnerability allows executing phishing attacks, as well as leaks of the information handled by the user in the involved Activity. Additionally, this vulnerability allows the attacker modifying the data, putting at risk its integrity.

Service hijacking is a vulnerability similar to Activity hijacking with only difference being that it targets services instead of activities. This vulnerability is more persistent, however, due to the fact that it is transparent to the user because the services do not include graphic interface for it.

For intent spoofing, a typical scenario is that the vulnerable application has a component which only expects to receive intents from other components of the same application. However, if the component is exported, and it becomes exported when declaring intent filter, then any application can send intents to it. Moreover, they do not have to be implicit intents, and if they are explicit then they do not even have to match the intent filter.

These vulnerabilities share the cause – mechanism of implicit Intents and their inherent lack of security. It is advised to avoid using implicit Intents for IPC and instead use explicit Intents when possible, because explicit Intents always target specific component and cannot be intercepted by malicious component. When use of implicit Intents is required, parameters of the intent, especially its origin, should be validated.

Vulnerabilities, described above, can be avoided if developers of the application are aware of both the vulnerabilities, and rules and guidelines to develop secure applications. Security specialists offer guidelines to secure coding for various platforms and programming languages. For example, CERT (Computer Emergency Response Team) offers “The CERT Oracle Secure Coding Standard for Java” that covers the rules for developing secure Java applications. Most of these rules apply to Android platform as well. CERT also offers a set of rules for Android specifically. Another set of guidelines is provided by developers of Android and is featured in the official developers guide to Android [7]

Conclusion

As a result of the Android security model and IPC mechanisms overview, basic IPC vulnerabilities of Android applications are described. It is shown, that mechanism of implicit Intents is the source of the most of IPC vulnerabilities, which is connected to the inherent lack of security of the mechanism. Considering this, it is advised to minimize usage of implicit Intents for IPC. When it is impossible to avoid using implicit Intents, source of them should be validated.

REFERENCES

[1] System and kernel security | Android open source project: [Electronic resource]. – Mode of access: https://source.android.com/security/overview/kernel-security.html.

[2] Dubey Abhishek Android Security – Attacks and Defenses / Abhishek Dubey, Anmol Misra // Taylor & Francis Group 2013 P. 272.

[3] System permissions | Android developers: [Electronic resource]. – Mode of access: http://developer.android.com/guide/topics/security/permissions.html.

[4] Application fundamentals | Android developers: [Electronic resource]. – Mode of access: http://developer.android.com/guide/components/fundamentals.html.

[5] Exploring the OWASP Mobile Top 10: M1 Insecure data storage: [Electronic resource]. – Mode of access: http://community.hpe.com/t5/Protect-Your-Assets/Exploring-The-OWASP-Mobile-Top-10-M1-Insecure-Data-Storage/ba-p/5904609#.Vtd0A9CjXeg

[6] Chin E., Porter Felt A., Grenwood k., Wagner D. Analyzing Inter-Application Communication in Android [Electronic resource] / Erika Chin, Adrienne Porter Felt, Kate Greenwood, David Wagner. Mode of access: https://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf

[7.] Security Tips | Android developers: [Electronic resource]. – Mode of access: http://developer.android.com/training/articles/security-tips.html