The Solution of Denial of Service Attack on Ordered Broadcast Intent

Ji-Soo OH*, Min-Woo PARK*, Tai-Myoung CHUNG**

*Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea

**Department of Software, Sungkyunkwan University, Korea

jsoh@imtl.skku.ac.kr, mwpark@imtl.skku.ac.kr, tmchung@ece.skku.ac.kr

***Abstract*— The Android's message passing system provides late run-time binding between components in the same or different applications, and it promotes inter-application collaboration. However, the message passing mechanism has also numerous vulnerabilities, so that Android applications can be exposed to attacks from malicious applications. Denial of service (DoS) attack on ordered broadcasts is a typical attack that exploits vulnerabilities of message passing. A malicious application which launches the attack intercepts broadcast messages by setting itself high priority, and then aborts it to prevent other benign applications from receiving it. In this paper, we propose a security framework for detecting DoS attacks on ordered broadcasts. We insert our framework into Android platform, and then the framework inspects receivers of broadcast messages. If the framework detects any threats, it issues warning to user. Finally, we provides scenario about our framework, and discuss future directions.**

***Keywords*—Android, Mobile Phone Security, Intent, Ordered Broadcast, Denial of Service Attack**

I. INTRODUCTION

Modern smartphone has a lot of advantages compared with a feature phone. One of the greatest advantages of the smartphone is that users can personalize their own smartphone by using a various applications. Gartner says worldwide smartphone sales grew 46.5 percent in second quarter of 2013 from the same period last year, and Google Android kept its lead, garnering 79 percent of the smartphone operating system market [1]. However, the popularity of android smartphones also makes the platform attractive to attackers. New mobile threat families and variants rose by 49 percent in first quarter of 2013 from last quarter from 100 to 149 according to mobile threat report by F-Secure [2].

Although android applications run in isolated sandboxes for security and privacy, the Android platform provides Inter-Component Communication (ICC) channels like Intent for functionality of applications. Thus, developer can make more flexible and universal functions using collaboration of applications. For example, email application cooperates with web browser to show URL that is contained in received email. When user clicks URL, email application will call web browser to handle URL. However, ICC channels are vulnerable to Denial of service attacks on ordered broadcast, which is a classic attack to exploits vulnerabilities of message passing [3].

In this paper, we first fully explain the denial of service attack on ordered broadcast. We then propose a scheme to detect the attack that is added into the Android platform. Our security framework inspects receivers of the ordered broadcast, and if any threats are detected, the framework issue warning.

The rest of the paper is organized as follows. After we explain Android's message passing system overview in Section 2, we describe of our security framework and provide a scenario which describes the system in Section 3. Finally, we conclude the paper in Section 4.

II. MESSAGE PASSING

To prevent malicious behaviours of user applications, the Android platform runs each application on an isolated environment, which is called sandbox. At the same time, however, Android platform provides some channels for communication among applications. In this section, we describe a message passing system of the Android platform, which is called Intents.

A. Intents

The Intent is a message object containing data that describe the task to be accomplished. The Intent messaging provides late run-time binding between components in the same or different applications. In addition, the operating system uses Intent messaging as event notifications.

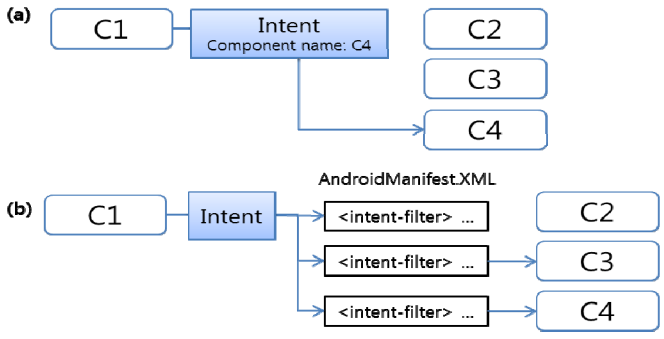


Figure 1. Intents: (a) Explicit Intent; (b) Implicit Intent

The Intents can be divided into two groups: an explicit Intent and an implicit Intent (Figure 1). An explicit Intent specifies the target component which needs to process the Intent by its name. An implicit Intent does not designate a target component's name. The Android platform will look at the other parameters in an Intent object to deliver the message.

Components can have zero or more Intent filters to receive implicit Intents. If the component does not have any Intent filters, it can receive explicit Intents only. An Intent filter is complex logic structures that can be defined inside the AndroidManifest using XML tags. Each filter describes the functional abilities of the component and limitations of the Intents that the component is pleased to handle. An Intent filter can match against actions, categories, and data in the Intent. An Intent filter also allows a priority attribute which is used to order matching filters. An implicit Intent is delivered to a component only if the Intent matches the component's Intent filter, while an explicit Intent is delivered to its target whatever it contains.

Broadcast Intents are a kind of an implicit Intent that is delivered to all interested receivers. The broadcast Intents can be divided into normal broadcasts and ordered broadcasts. The normal broadcasts are not synced to anything else. Any receivers that subscribe to a normal broadcast are executes in any undecided order, commonly at the same time. On the other hand, the ordered broadcasts are delivered to a single receiver one by one. Receivers of the broadcast are run in turn and choose to propagate a result to the next receiver or to abort the broadcast. The order of receivers in delivery chain is determined using the priority attribute of their Intent filter by message passing system. The normal broadcasts and ordered broadcasts sent via a APIs those are sendBroadcast() and sendOrderedBroadcast() [5, 6].

B. Intent Routing

Activity Manager and Package Manager are included in application framework that is preinstalled on Android devices, and they are responsible for delivering Intent (Figure 2). Activity Manager controls an application's life cycle and mediates inter-component communication. Package Manager maintains meta-data about all applications loaded in the systems which includes Intent filters extracted from the manifest files. The Intents are delivered to the Activity Manager first, and the Activity Manager finds the target components using components' information that is managed by the Package Manager. Finally, the Activity Manager makes the appropriate decision for delivering the Intent, and then the Intent is delivered to selected components. Therefore, we focus on the Activity Manager to improve security of the Android's Intent delivery system [5].

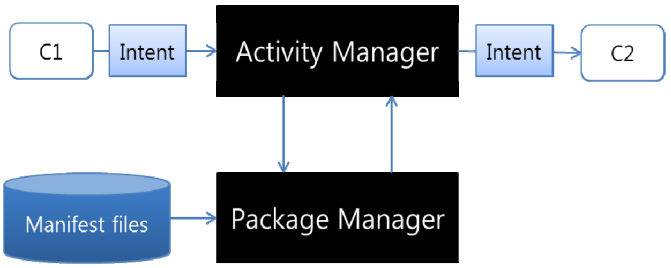


Figure 2. Intent routing

III. SECURITY FRAMEWORK

A. Denial of Service Attack

Android applications communicate with each other using the Intent, and this mechanism encourages cooperation of applications. Therefore, an application can reuse existing data and services provided by other applications. However, there are several vulnerabilities in message passing mechanisms, so that applications can be exposed to attacks which exploit these vulnerabilities. There is a typical attack that exploits vulnerabilities of ordered broadcast Intents, called denial of service (DoS) attack on ordered broadcasts.

Ordered broadcast Intents are serially delivered to receivers, and then receivers decide to propagate the Intent to the next receiver or to abort it. It is depended on receiver's priority to decide order of receivers. An active attacker could launch the DoS attack on ordered broadcast by setting itself high priority. The attacker receives the broadcast Intent first and impedes delivery to other receivers by aborting it (figure 3). To prevent all related benign components from receiving the Intent, the attacker sets abnormally high priority. After receiving the Intent, attacker calls abortBroadcast() method to abort the Intent. Therefore, if the first receiver of ordered broadcast Intent sets its priority aberrantly high and aborts the Intent, we can suspect that DoS attack on ordered broadcasts occur.

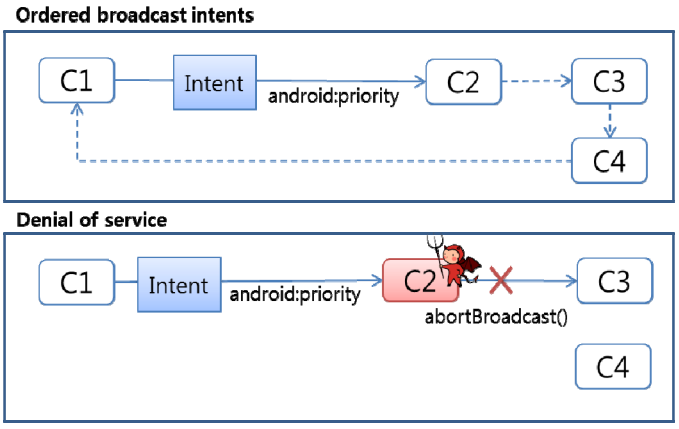


Figure 3. Denial of service on ordered broadcasts

B. Framework Overview

Our security framework is added to Android platform for detecting DoS attack on ordered broadcasts. As previously stated, Activity Manager is responsible for Intent delivery. Therefore, we insert the framework into Android platform by modifying Activity Manager, and then the framework thoroughly inspects a potential risk that a receiver tries to do DoS attack. If any threats are detected, our security framework issues warning about the threats to the Android user. After identifying the warning issued, the user can decide whether to keep using an application which has a possible risk of DoS attack or delete it. The Android distribution model allows developers to upload arbitrary applications on the Android application market, and the Android users tend to download applications incautiously. Therefore, the users are exposed to DoS attack by downloading malicious application carelessly, and the warning issued helps the user avoid threats from the attack.

Because Android applications are hard-coded, our framework generates just one warning about each type of Intents after installing the application that has a potential threat. Our security framework is backward-compatible, so the modification of platform does not have an effect to existing system. The proposed security framework is based on the Android 4.2.1 sources.

C. Proposed security framework

(Figure 4) shows a sequence diagram for the ordered broadcasts. If a component has a request for sending an ordered broadcast, it calls a sendOrderedBroadcast() and it calls a broadcastIntent() that is in the ActivityManagerService class. A broadcastIntent() is core method that is used to make both normal and ordered broadcasts. Normal and ordered broadcasts are different to parameters. A major difference is a value of the serialized flag. A broadcastIntent() handles received parameters to make ordered broadcast, and then it performs a preparation stage for calling a broadcastIntentLocked(), which is in the same class. A broadcastIntentLocked() is a core method for Intent delivery. A destination of Intent delivery is decided in this method depending on the value of the serialized flag, which is delivered from a broadcastIntent(). If the serialized flag is set to the ordered flag, a broadcastIntentLocked() makes a list of receivers. The list is made by merging a list of receivers specified in Intent filters and another list of receivers registered dynamically by other components. The sequence of the list of receivers is related to the priority of receivers that is decided by developer. Thus, we insert our security framework into broadcastIntentLocked() to detect threatening receivers.

(Figure 5) shows overall process of our security framework. We can find the first receiver of an ordered broadcast using the merged list. To identify that the receiver aims to DoS attack on the ordered broadcast, our framework compares the receiver's priority attribute with pre-defined constant called SYSTEM_HIGH_PRIORITY, which has value 1000. The Android applications should never use filters with this or higher priorities. However, even though an application sets its priority higher than 1000, there is no restriction on the application. If the first receiver's priority is equal to or higher than SYSTEM_HIGH_PRIORITY, our framework considers that the receiver has a potential risk and proceeds to the next step of inspection process. After comparing priorities, we make sure whether the first receiver aborts the current broadcast or not. There is a flag defined as mAbortBroadcast, which indicates whether or not the receiver should abort the current broadcast. Hence, the framework can check the receiver using this flag. If the flag is set to true, our framework issues warning that the receiver may impede services. The warning contains the task requested or reported, sender of the Intent, and other receivers obstructed by the first receiver. The user identifies the warning, thereby he can make a decision whether to keep using the application or to delete the application. Pseudo code of our framework is showed in (figure 5).

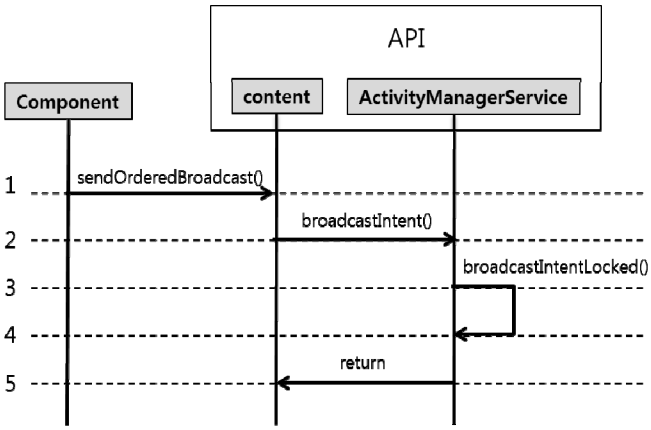
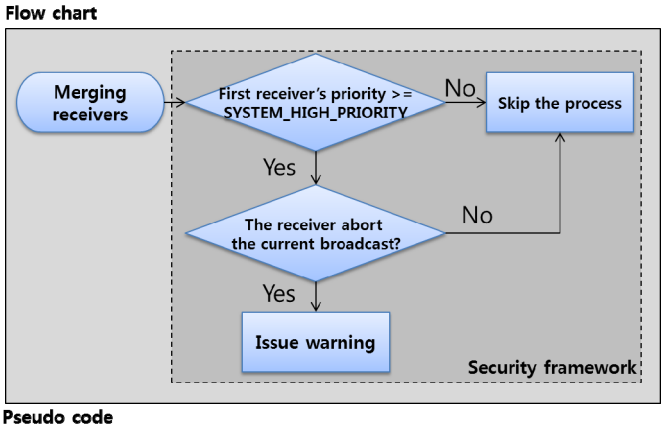


Figure 4. Sequence diagram for ordered broadcasts



Pseudo code

```
Set firstReceiver has the highest priority in receivers list
IF firstReceiver.getPriority() >= SYSTEM_HIGH_PRIORITY THEN
        IF firstReceiver.getabortBroadcast() == TRUE THEN
                issueWarning(the task requested by intent,
                        sender of intent, other receivers)
```

Figure 5. Security framework

D. Scenario

(Figure 6) describes a scenario that our security framework warns the Android user about an application which aims to DoS attack on ordered broadcasts. There is an application which contains a malicious receiver named malApp. The receiver intercepts SMS messages and impedes other services associated with the messages. If the Android device receives a SMS message, malApp takes an Intent about the SMS message first and abort it to hide the message from other applications. malApp's Intent filter specified its action to SMS_RECEIVED and priority to 9999 for intercepting the SMS message. In principle the Android applications' priorities should never be set equal to or higher than SYSTEM_HIGH_PRIORITY, but it is not actually limited by the Android system.

When a SMS message is sent to the Android device, an ordered broadcast about the SMS message is delivered to the Activity Manager of the device. The Activity Manager finds receivers that specified their action as SMS_RECEIVED in each Intent filter. It can get information about applications through the Package Manager. Then, the Activity Manager decides order of receivers depending on their priority. Because malApp's priority is set abnormally high, it is selected as a first receiver of the broadcast by the Activity Manager. Our security framework inspects the first receiver of the ordered broadcast, so that malApp is analysed by the framework. Its priority is higher than SYSTEM_HIGH_PRIORITY and its mAborBroadcast flag is set to true. Therefore, our framework issues warning which contains related information. After confirming the warning issued, the user is informed that malApp impedes delivering Intent about SMS message to other receivers. Actually the user did not even know the message received until the warning is issued. Once noticing the warning, the user finds that SMS message is already arrived, and malApp hide the message. Finally, the user can aware that malApp is malicious receiver via the warning, and then the user deletes the application containing malApp.

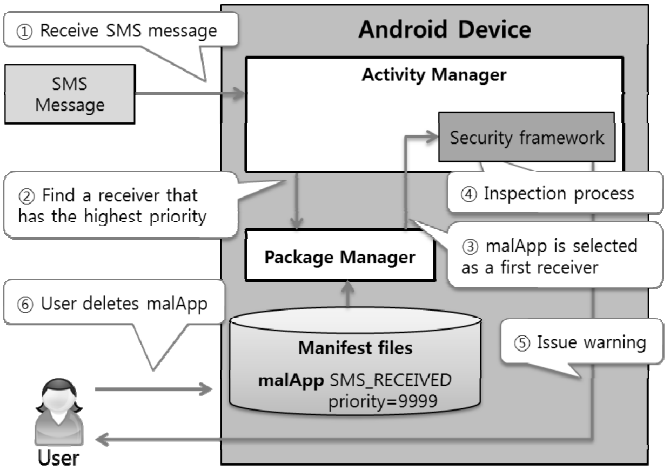


Figure 6. A scenario for operation of security framework

IV. Conclusion

In this paper, we describe the Android message passing system and address the problem of Denial of Service (DoS) attack on ordered broadcasts. We propose a security framework for detection DoS attack on ordered broadcasts. Our framework inspects the first receiver of an ordered broadcast Intent, and if there is any threat, the framework issues warning to the Android user. The user can decide whether to keep using the application or delete it. Finally, we show a scenario that our framework issues warning about a result of inspection process and then the user deletes a malicious application which intercepts SMS message. For our future work, we plan to extend our security framework to detect and prevent various attacks. There are diverse threats in the Android massage passing system, so that we aim to implement the extended security framework

Acknowledgment

This work was supported by Priority Research Centers Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012-0005681)

References

[1] A.Gupta, "Market Share Analysis: Mobile Phones, Worldwide, 2Q13", Gartner, Aug. 2013.
[2] F-Secure, "Mobile Threat Report January-March 2013", Apr. 2013
[3] E.Chin, A. P. Felt, K. Greenwood, and D. Wangner, "Analyzing Inter-Application Communication in Android", In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), June. 2011
[4] D.Kantola, E. Chin, W. He, and D. Wagner, "Reducing Attack Surfaces for Intra-Application Communication in Android", In Proceeding of the 2nd ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Oct. 2012
[5] Google. The Android developer's guide, http://developer.android.com/
[6] W.Jackson, "Learn Android App Development", 2013
[7] W.Enck, M.Ongtang, and P.McDaniel, "Understanding Android Security", IEEE Security & Privacy Magazine, Jan. 2009