Poster: Analyzing Inter-Application Communication in Android

Erika Chin*, Adrienne Felt*, Kate Greenwood†, David Wagner‡
*Graduate Student †Undergraduate Student ‡Professor
Department of Computer Science
University of California, Berkeley
Berkeley, California 94720
{emc, apf, kate_eli, daw}@cs.berkeley.edu

I. Introduction

Over the past decade, mobile phones have evolved from simple devices used for phone calls and SMS messages to sophisticated devices that can run third-party software. Phone owners are no longer limited to the simple address book and other basic capabilities provided by the operating system and phone manufacturer. They are free to customize their phones by installing third-party applications of their choosing. Mobile phone manufacturers support third-party application developers by providing development platforms and software stores (e.g., Android Market, Apple App Store [1], [2]) 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. For example, a restaurant review application can ask other applications to display the restaurant's website, provide a map with the restaurant's location, and call the restaurant. This communication model reduces developer burden and promotes functionality reuse. Android achieves this by dividing applications into components and providing a message passing system so that components can communicate within and across application boundaries.

Android's message passing system can become an attack surface if used incorrectly. In this poster, we show the risks of Android message passing and identify insecure developer practices. 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.

We have seen numerous malicious mobile phone applications in the wild. For example, SMS Message Spy Pro disguises itself as a tip calculator and forwards all sent and received SMS messages to a third party [3]; similarly, MobiStealth records SMS messages, call history, browser history, GPS location, and more [4], [5]. This is worrisome because users rely on their phones to perform private and sensitive tasks like sending e-mail, taking pictures, and performing banking transactions. It is therefore important to help developers write secure applications that do not leak or alter user data in the presence of an adversary.

We examine the Android communication model and the security risks it creates, including personal data loss and corruption, phishing, and other unexpected behavior. We present ComDroid, a tool that analyzes Android applications to detect potential instances of these vulnerabilities. We used ComDroid to analyze 20 applications and found 34 vulnerabilities in 12 of the applications. Most of these vulnerabilities stem from the fact that Intents 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.

II. Background

To understand the threats that the communication model introduces, we first give a brief overview of Android. Android applications are broken down into *components*, application building blocks. There are four types of components: Activities, Services, Broadcast Receivers, and Content Providers. Activities provide the user interfaces of the application. Services run in the background and do not interact with the user. Broadcast Receivers respond to messages sent by the system or other applications. Content Providers are databases that are addressable by their application-defined URIs.

Intents are messages passed between or within applications. Intents can be sent between 3 of the 4 components: Activities, Services, and Broadcast Receivers. Upon receipt of an Intent, each component performs a different functional task. Intents can start an Activity, start or bind to a Service (requesting a task to be performed), or be sent to be processed by a Receiver.

The Android system provides a few ways to limit a component's exposure to Intents. Components can be set to be accessible to only the application (private components) or global to the system (public components). They can also require that the invoking components/Intents have specific permissions (with varying degrees of difficulty to obtain).

Intent sending can also be restricted. Intents can be explicitly addressed to a particular component. This will prevent unauthorized components from receiving the Intent. Alternatively, they can be left unaddressed and the system will determine the appropriate recipient component.

III. Attacks and Analysis

We examine the security challenges of Android communication from the perspectives of Intent senders and Intent recipients. In our poster, we show how implicit, unprotected Intents to can be received by the wrong application, potentially leaking user information. In *Broadcast theft*, a broadcast Intent can be sniffed, tampered with or stolen by a malicious component. In *Activity hijacking*, an Activity Intent can be intercepted and a malicious Activity can be displayed in its place. This can result in unexpected actions being taken, phishing (password/data theft), and data injection through false response. In *Service hijacking*, a Service Intent can be intercepted by a malicious Service with similar results as Activity hijacking. It also is a more stealthy attack as no user interface is involved. All of these attacks also prevent legitimate actions from being performed by the rightful component recipient.

Similarly, we consider vulnerabilities related to receiving malicious external Intents. If an application's components are declared incorrectly or are not strongly protected via permissions, external applications can invoke its components in surprising ways or inject malicious data into the application. We call this class of attacks *Intent spoofing*. In *malicious broadcast injection*, a malicious Intent is sent to a trusting receiver. This can result in an unauthorized action being taken and/or data injection. In *malicious Activity/Service launch*, a malicious Intent is sent to a trusting Activity/Service. This can result in an unauthorized action being taken, application state change, data modification/injection, and/or data leakage and corruption.

We built a tool, ComDroid, to detect these vulnerabilities. The tool statically analyzes Dalvik executable files. As no application source code is needed, ComDroid can be used by end-users, the Android Market, and third-party reviewers in addition to developers. ComDroid performs flow-sensitive, intraprocedural analysis, and examines the permissions defined by the application and the Android system, Intents sent by the application, and components that receive Intents. It issues warnings when it finds potential vulnerabilities.

IV. Results

We ran ComDroid on the top 50 popular paid applications and on 50 of the top 100 popular free applications on the Android Market [1]. We manually examined 20 of these applications to check ComDroid's warnings, evaluate our tool, and detect vulnerabilities.

ComDroid generated 181 warnings for the 20 applications and our manual review confirmed 20 definite vulnerabilities (which do not rely on user interaction), 14 spoofing vulnerabilities (which may occur if the user is tricked), and 16 common, unintentional bugs. Of the 20 applications examined, 9 applications contain at least 1 definite vulnerability and 12 applications contain either definite or spoofing vulnerabilities.

In 25 cases, we were unable to determine whether warnings were vulnerabilities. We cannot always determine whether a surface is intentionally exposed without knowing the developer's intentions. We were uncertain of 25 of the 181 warnings. The remaining 106 warnings were false positives, i.e., not dangerous or spoofing vulnerabilities or common bugs. Of these, 6 of the warnings should not have been generated and can be attributed to shortcomings in our implementation of ComDroid. The remaining 100 false positives are still exploitable attacks. However, the impact of these attacks is minor: They would be merely a nuisance to the user. For example, an Activity that turns on a "flashlight" when launched or takes some other trivial action would fall into this category. Because they represent only a nuisance, we conservatively decided not to classify them as vulnerabilities.

In our poster, we discuss the results of a few of the applications we manually examined to illustrate how applications can be attacked and we reveal common developer misunderstanding of the current Intent system. We also recommend changes to the Intent system that will eliminate or decrease the number of vulnerabilities in these applications.

V. Conclusion

While the Android message passing system promotes the creation of rich, collaborative applications, it also introduces the potential for attack if developers do not take precautions. We examine inter-application communication in Android and present several classes of potential attacks on applications. Outgoing communication can put an application at risk of Broadcast theft (including eavesdropping and denial of service), data theft, result modification, and Activity and Service hijacking. Incoming communication can put an application at risk of malicious Activity and Service launches and Broadcast injection.

We provide a tool, ComDroid, that developers can use to find these kinds of vulnerabilities. Our tool relies on DEX code, so third parties or reviewers for the Android Market can use it to evaluate applications whose source code is unavailable. We analyzed 100 applications and verified our findings manually with 20 of those applications. Of the 20 applications, we identified 12 applications with at least one vulnerability. This shows that applications can be vulnerable to attack and that developers should take precautions to protect themselves from these attacks.

References

[1] "Android Market," http://www.android.com/market/.
[2] "iPhone App Store," http://www.apple.com/iphone/apps-for-iphone/.
[3] M. A. Troy Vennon, "Android malware: Spyware in the Android Market," SMobile Systems, Tech. Rep., March 2010.
[4] T. Vennon, "Android malware: A study of known and potential malware threats," SMobile Systems, Tech. Rep., February 2010.
[5] "MobiStealth," http://www.mobistealth.com/.