Preventing Activity Hijacking Attacks in Android APP

Min-Hao Wu[1,3,a], Fu-Hau Hsu[2,4,b], Ting-Cheng Chang[1,3,c], Liang Hsuan Lin[2,4,d]

[1]College of Information & Mechanical and Electrical Engineering, Ningde Normal University

[2]Department of Computer Science and Information Engineering, National Central University

[3]No.1 College Road, Dong Qiao Economic and Technology Development Zone, Ning De City, Fu Jian Province, China.

[4]No. 300, Zhongda Rd, Zhongli District, Taoyuan City 32001, Taiwan.
[a]mhwu@ndnu.edu.cn; [b]hsufh@csie.ncu.edu.tw; [c]18250922163@163.com; [d]dlinliang258369@gmail.com

Abstract

In recent years, the hijacking vulnerabilities of Android components have been widely discussed, and hijacked Android components have been used to disclose personal information or private data to attackers. Such attacks redirect the Android component's original workflow to malicious code, or even execute malware. We propose an Activity Hijacking Attacks (AHA) scheme that examines the original activity workflow to keep track of every activity in the Android framework. This enables AHA to detect a malicious app that attempts to open an activity in the foreground or whose layout is similar to a login page, i.e., with a text field for account names and passwords. When such activities are detected, our scheme notifies users to be cautious of keying in their credentials for the new activity. Experimental results show that using AHA can prevent attacks designed to steal personal information. Furthermore, our proposed scheme can be easily patched into the existing Android system and has a negligible overhead.

Key words: Android Malware, Activity Hijacking, View System, Android forensics, Instant message.

Introduction

The use of smartphones has grown at an alarming rate, becoming the market's top mobile platform in a very short space of time. This success has been due to open source licenses, aggressive marketing strategies, and a unique combination of stylish interfaces. Smartphones have attracted a large number of users to handset manufacturers, mobile networks, silicon manufacturers, and application programs. Worldwide, there are numerous products, applications, and devices. In recent years, the Android operating system has achieved the highest market share of all smartphone devices. To improve the smart mobile device had existed of the most security issues or consistency of the product by comparison with the current closed for smart mobile devices (such as iOS or Windows Phone). Each application is subjected to strict examination and control to identify any security problems. Because Android upholds the principle of open source software, it has attracted a large number of users, mobile device manufacturers, application developers, and vendors. However, there are fewer security mechanisms in the market for open source applications, and so malicious behavior is rampant on Android systems. The reverse engineering of Android malware or detection technology has continued to develop. Variants of this malware are generated rapidly, such as value-added services for current software in which the malware subscribes to certain unwanted services. There is a considerable amount of adware in Android mobile devices. This adware may deliver advertising disguised as emergency notifications. In addition to adware, many types of malware attempt data theft, malicious downloads, crack software, and click-through fraud, and operate as spyware tools. These malicious tools present many risks to the software on Android systems, such as the theft of financial information. Anti-virus companies test and analyze a large number of suspicious samples, and the detection or feature extraction efficiency of static analysis scan engines detects malicious software under low. Finally, the Android market has serious security problems.

Nowadays, the number of smartphone users is growing at a steady pace. More and more people rely on smartphones to carry their business data or connect to social media. However, users are probably unaware that important personal information may be leaked from their smartphone. In 2015, the International Data Corporation [1] (IDC) reported that Android and iOS constitute almost all of the worldwide smartphone market. Fig. 1 shows that, in 2014, the Android system had an 81.5% market share of all smartphone shipments, with iOS second on 14.8 % and other systems totalling 3.7%. Consequently, Android security vulnerabilities or attacks have the potential to affect a large number of users.

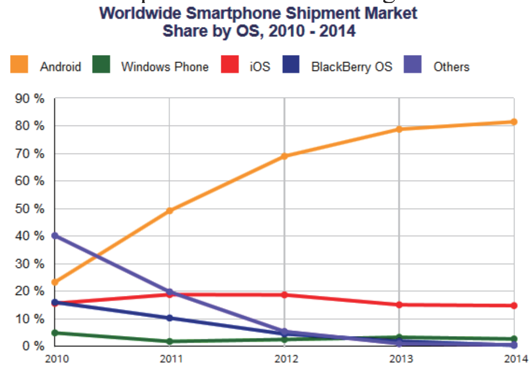


Fig. 1. Smartphone market share by OS

The Android system is no longer just a telephone—the system now affects everyday life, operating as a computer, television, and global positioning system. The Internet business where the ability to gather a large number of Apps in a short time, as long as a sufficient amount of Apps in the Google Play. Many Android apps are free, using in-app advertising to earn fees. However, some can also steal private data to make money, or even directly steal airtime. These malicious Android apps have been known to make a lot of money for hackers in a short space of time. As the Android system has become more popular, the number of developers has grown significantly. Regardless of whether the app developer is professional or amateur, some security vulnerabilities may be unknowingly introduced during the development phase. When such apps are published to Google Play or third-party websites, users may download the apps complete with vulnerabilities. The user's personal information or business data could then be leaked or stolen by an attacker. Attacks using keystroke inference have been used to break the integrity of the graphical user interface (GUI) and steal the users credentials [2─4].

In 2011, a vulnerability was discovered on Android devices that allowed a user's private information to be stolen from an application [5]. The malicious developer had to install an app on the Android device that allowed an application to create an overlay screen that peeked at another application's login screen. When the user went to login to the desired application, their private information would be sent to the correct application as well as to the remote server of the malicious developer, who then had access to the victim's private information. There are many attack methods for hijacking Android components to access private information or change the system behavior. If the content provider is hijacked, the attacker may be able to alter the user's call logs, text messages, and sensitive data. After an attacker has successfully hijacked a component, they may acquire sensitive data or device information in the background, and even modify the system workflow.

In Section 2, we introduce the Android architecture and the fundamental structure (e.g., ActivityManagerService) of our solution. To describe the concept of Activity Hijacking Attacks, we introduce the related Component Hijacking Vulnerabilities in Section 3. We then present our Activity Hijacking Attacks scheme in Section 4. contains the results of an evaluation in which we examine Activity Hijacking Attacks using well-known benchmarks, proof-of-concept test cases, and measure the performance overhead of the system. Finally, conclusions to this study.

Related works

This section gives a brief introduction to the Android system, its architecture, and fundamental building blocks. We then discuss some studies related to the Android system in detail.

In recent years, Android has become one of the most popular mobile operating systems. However, the poor design of certain applications leads to security weaknesses that seriously undermine the security and privacy of users. In 2012, Lu et al. [6] proposed an automatic static analysis method called Android Component Hijacking Attacks that identifies the vulnerability of an app's dataflow. This method analyzes Android apps and detects possible hijack-enabled flows by conducting low-overhead reachability tests on customized system dependence graphs. The component hijacking vulnerabilities use static analysis with an automatic audit of Android application elements to exploit hijacking weaknesses. This method detects possible hijacks with low-overhead reachability tests that to flow with the analytical point of view the information from these weaknesses in the Android application. This method uses a static programming analysis approach with an anti-analysis technology that runs a dataflow analyzer on the Android programming.

Fig. 3 shows a flowchart of the CHEX method. In the first step, the entry point discovery was even listeners that defined in the manifest and methods overloaded relationships. For each iteration of the entry point discovery process, a context insensitive call graph is generated. This call graph is used to call a method or instantiate a class. In the second step, a split dataflow summary is constructed. This is the most computationally heavy step in the entire analysis. The third step constructs intra-split dataflows to analyze the intermediate representation. Finally, permutations are generated to determine the connectivity between the inter-split flows and permutation dataflow summary.

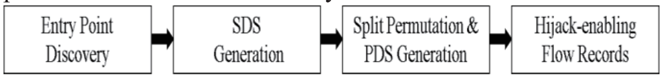

Entry Point Discovery
SDS Generation
Split Permutation & PDS Generation
Hijack-enabling Flow Records

Fig. 3. The CHEX method of the flowchart

The Android system is based on the Linux kernel, which provides services such as power management, memory management, network stack, driver model, and security settings for mobile devices. Because the Linux kernel is well developed, Android can simply patch some kernel codes to use special functionalities, such as Android Binder IPC (Inter-Process Communication) and other features for mobile users.

In the Android system, each application runs within its own well-isolated Dalvik Virtual Machine (Dalvik VM). The Dalvik VM is a device-agnostic environment. Application developers do not need to hard-code any communication or handling of the target device. Instead, they can focus on how to design the functionalities of apps, rather than dealing with the resource management of graphics, media, SQLite, and even memory.

From the application developer's point of view, the application framework is the access point for acquiring the system resources. The application framework provides a number of APIs, e.g., View System serves as an interface for the developer to construct the UI frames of the apps, and if an app wants device location information or uses Google Maps, it can call LocationManager.

The system service provides a long-running operation in the Android system. It helps application framework APIs to access the underlying hardware, and even provides functionalities for user-level applications. For instance, the ActivityManagerService, which is described in detail later, is used to manage the lifecycle of each application's activity in the entire system, and WindowManagerService is the interface that an application uses to draw or change the foreground window.

The Binder IPC provides a comparable mechanism between applications, and also helps the application framework to access the system services to acquire device information. For example, an application that wishes to obtain the device location using LocationManager. The upper-left shows an example app that is using the LocationManager provided by the framework. The LocationManager ask the service manager for permission to use the location service, and the service manager needs the binder's help to pass the message to the location manager. Finally, the real GPS information of the device is passed to the app.

Owing to the API exposed by the Application Framework, developers can build their own applications with thousands of interesting functionalities. The application components are the fundamental building blocks of an app. Each component provides its own special purpose for users to build their application. The following table 1 shows the four major components of the Android system.

Table 1. Android Components

Component	Description
Activity	Any application must contain at least one activity. Provide a single user interface of the application.
Service	A service usually offers a long-running operation in the system background and does not provide a user interface.
Content Provider	A content provider allows apps sharing data to each other, to store app specific data in the SQLite database, or query the system data.
Broadcast Receiver	A broadcast receiver is the use of manager the broadcast messages from the Android system or from the other applications.

Our Scheme

The AMS interacts with WindowManager and the View System. To prevent our event from being driven by Activity Hijacking Attacks, we first provide an overview of Activity Hijacking Attacks, including the design principles and core components. The system implementation is then described in detail.

The threat model and attack motivation were described in Section 1. Such attacks first collect information about the device, and, being a service in the system, wait for the user to attempt a login. At this point, the attack tool launches a forged login activity to trick users into keying in their personal information.

Therefore, the Activity Hijacking Attacks design principles follow the original activity in terms of what is shown on the screen, and the new activity appears in the system under a different package name. The new activity's layout has two EditText fields with one password input type. Under the above principles, we can identify the new activity as a possible forged login activity. To achieve these principles, our proposed Activity Hijacking Attacks scheme is implemented in the Android framework.

Activity Hijacking Attacks can be separated into three components. The first component is ActivityDetector, which detects and collects the popup activity information. The second component is the TextViewFinder, which detects the activity layout and determines whether the popup activity has a password input type field. The third component is the Activity Hijacking Attacks Service of the main controller components, which controls the alert system. Fig. 10 shows the Activity Hijacking Attacks system architecture. If the ActivityDetector or TextViewFinder acknowledge that an event appears, a message is sent to the Activity Hijacking Attacks Service to determine whether the popup activity is malicious.

The ActivityDetector is designed to detect popup activity from the system perspective. Under the Activity Hijacking Attacks framework, the Android system maintains a history stack that is patched with ActivityStack.java to catch activities. Every time an activity starts, it is recorded in the startActivityMayWait () function. The top of the history stack still shows the original activity, which the user is still interacting with. As the new activity is still in preparation, it has not yet appeared at the top of the history stack.

ActivityStack.java is patched in Activity Hijacking Attacks codes to obtain the original foreground activity package name, status, and ActivityInfo. Similarly, ActivityStack.java acquires the package name, status, and ActivityInfo of the new activity. If the above two package names are not equal, the two activities are not from the same application, and are sending different ActivityInfo. These data are sent to the Activity Hijacking Attacks Service.

The Activity Hijacking Attacks Service is the core of our proposed scheme. The AMS will complete all tasks belonging to the activity, and then send the WindowManager to draw the layout of the activity. Hence, the ActivityDetector and TextViewFinder are not going to catch the event at the same time. To coordinate the two components, the Activity Hijacking Attacks Service is used as a system service in Android. System services can run persistently, allowing ActivityDetector and TextViewFinder to pass information and useful data to the Activity Hijacking Attacks Service.

In the Activity Hijacking Attacks Service, the activity's package name and the ActivityInfo passed from the ActivityDetector are stored as member variables until the TextViewFinder sends the password input type information. After receiving information from these components, the Activity Hijacking Attacks Service examines the original foreground activity package name and the new activity package name. If they are not the same, the two activities are from different applications. Otherwise, these two activities are from the same application, in which case we clear all objects passed from the ActivityDetector and return the system to the original workflow. When the Activity Hijacking Attacks Service receives the TextViewFinder's information, it examines whether the new activity has the password input type. If so, the Activity Hijacking Attacks Service sends a message to the user alerting them to the situation. Fig. 3 illustrates the flowchart of the Activity Hijacking Attacks Service.

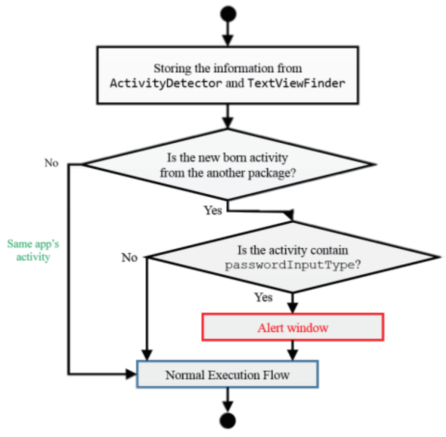

Storing the information from ActivityDetector and TextViewFinder
Is the new born activity from the another package?
No
Yes
Same app's activity
Is the activity contain passwordInputType?
No
Yes
Alert window
Normal Execution Flow

Fig. 3. The flowchart of the Activity Hijacking Attacks Service

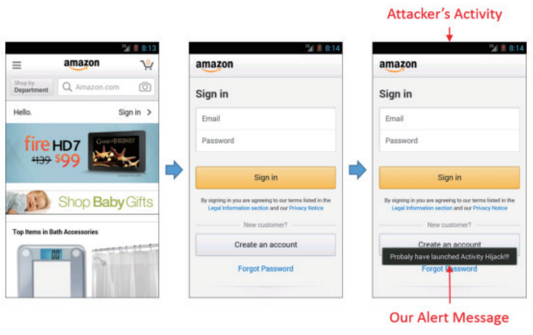
Experimental Results

Using the device emulator of the AOSP, we conducted the following evaluation. The test environment is summarized in Table 2.

Table 2. Testing environment

OS	Ubuntu 14.04.2 LTS
CPU	Intel (R) Core 2 Quad Q9650 3.00GHz
Memory	4GB

Activity Hijacking Attacks is implemented in the application framework of AOSP. When catching an Activity Hijacking Attack event, the app notifies the user that the device is under attack. Fig. 4(a) shows a malicious application that pops out an activity on the screen. Activity Hijacking Attacks detects this malicious behavior, and displays an alert message to the user. As shown in Fig. 4(c), the user can then decide whether to trust this new activity. Our proposed scheme does not disturb or bother others to force the activity to close.



(a) (b) (b)

Fig. 4. Implementation snapshot of Activity Hijacking Attacks

Conclusions

In this paper, we have described an Activity Hijacking Attacks scheme that prevents users from being tricked into typing their account name and password into TextView fields in forged login activities. The app examines the ActivityManager and WindowManager workflows, and uses the Android View System to identify the layout of the forged activity. Finally, the proposed event-driven solution in Activity Hijacking Attacks alerts users to the possibility that some popup activity is malicious. Activity Hijacking Attacks operates in the Android framework layer to catch the execution flow and layout of suspicious activities. The proposed scheme can successfully alert users to and prevent such attacks. Activity Hijacking Attacks protects end users from the malicious hijacking of foreground activities. It not only defends against attacks, but also has a negligible performance overhead.

ACKNOWLEDGEMENTS

This research was supported by grants from Recruiting High Level Talent program of Ningde Normal University (2018Y22).

References

[1] I.D. Corporation. Worldwide Quarterly Mobile Phone Tracker [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS25450615

[2] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, 2012, p. 9.

[3] C.C. Lin, H. Li, X.y. Zhou, and X. Wang, "Screenmilker: How to Milk Your Android Screen for Secrets," in *21st Annual Network and Distributed System Security Symposium* (NDSS), San Diego, California, USA, 2014.

[4] L. Cai and H. Chen, "TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion," in *Proceedings of the 6th USENIX conference on Hot topics in security, HotSec'11*, USENIX Association, Berkeley, CA, USA, 2011, pp. 9-9.

[5] S. Schulte. TWSL2011-008: Focus Stealing Vulnerability in Android [Online]. Available: https://www.trustwave.com/Resources/SpiderLabs-Blog/TWSL2011-008--Focus-Stealing-Vulnerability-in-Android/

[6] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 229-240.

[7] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it: Ui state inference and novel android attacks," in *23rd USENIX Security Symposium* (USENIX Security 14), 2014, pp. 1037-1052.

[8] AnTuTu Benchmark [Online]. Available: http://www.antutu.com/index.shtml

[9] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011, pp. 239-252.

[10] M. Zhang and H. Yin, "AppSealer: Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks in Android Applications," in *Proceedings of the 21th Annual Network and Distributed System Security Symposium* (NDSS 2014), 2014.