International Journal of Security and Its Applications
Vol. 9, No. 4 (2015), pp. 317-326
http://dx.doi.org/10.14257/ijsia.2015.9.4.29

Android's External Device Attack: Demonstration and Security Suggestions

Zhang Wei, Yang Chao and Chen Yunfang*

Nanjing University of Posts and Telecommunications
chenyf@njupt.edu.cn

Abstract

A large number of users choose to manage their Android phones via assistant software based on the ADB tool. These users face threats of a variety of attacks, such as external device attacks, because the current Android system's security mechanism does not pay sufficient attention to the safety of the ADB tool. This paper provides a detailed introduction of the ADB tool and discusses threats its use can bring. These threats include silently installing malicious apps, pushing malicious privilege escalation files to the phone, copying private files from the phone, risking opening TCP debugging mode, and introducing vulnerabilities by the ADB tool. We demonstrate a peripheral device attack is presented to help understand how the designed and implemented flaws can be used to bypass security protection. The demonstration is realised by Raspberry Pi, an open-source single chip computer that can run an ARM-based Linux system. Finally, some suggestions are proposed for security improvements to address threats introduced by the ADB tool.

Keyword**:** *Android Security; ADB Tool; External Device Attack; Security Suggestions*

1.Introduction

With the increasing percentage of Android's market share, the security issues of Android are getting more and more attention from researchers, engineers and consumers. Android's traditional security mechanisms, such as Application permissions, Component encapsulation, and Signing applications, no longer protect users' security well and have given rise to some new attack methods that can easily bypass these mechanisms. For example, the Masterkey vulnerability [1] announced by Bluebox can be utilised to bypass the Signing applications mechanism.

To address these new attack methods, researchers continually propose new security solutions. For example, targeting root exploits, Smalley and Craig enable the use of SE Android [2] in the Android kernel, and we can see their work in Android 4. 3. Some other researchers have improved traditional security mechanisms to address the new threats, such as Nauma *et al.*, who proposed an Android Permission Extension framework (APEX) [3] to cope with the app's excessive permission applications. Google continues to introduce new versions of the Android system. One motivation doing so is to add new features; another is to repair the old versions' vulnerabilities. At this writing however, weaknesses that can be used by attackers still exist. The Android Debug Bridge (ADB) tool is one such weakness.

Android Debug Bridge is a command tool that can provide connections between a computer and an Android emulator or a real Android phone. The initial goal was to enable developers to debug their programs for an Android system. However, many corporations developed Android phone assistant software based on the ADB tool on a computer. Because the assistant is so convenient, large numbers of users choose to manage their Android phone with it. In 2013, the users of a popular software assistant named

ISSN: 1738-9976 IJSIA

WanDouJia increased by more than a half million per day in China [4]. However, the negative issue has arisen that the information of users of the ADB tool is no longer protected by the Android security mechanism.

The rest of the paper is structured as follows. Section 2 introduces related work. Section 3 describes the ADB tool. Section 4 analyses the security threats brought by using current unsafe ADB tool. Section 5 demonstrates the attack. Section 6 provides some suggestions to improve the ADB tool. Section 7 concludes the paper.

2. Related Work

Research into Android security has been increasing in recent years. Many researchers have put proposals forward, and most have focused on the safety of the Android system itself and concentrated on improving the Android permission mechanism [5] and component communication protection [6], designing a security framework for Android [7], and detecting malicious applications [8]. Because of security improvements in the Android system, more and more attackers have begun to attack Android phones from the devices that connect to the phones.

At the 2013 Black Hat Forum, a research team from the Georgia Institute of Technology demonstrated a malicious charger called “Mactans” [9] targeting IOS, alerting us that smart phones can be attacked by external devices. This problem also exists in the Android system. The ADB tool is a bridge connecting cellphones and computers or other devices. Many users have chosen assistant software based on the ADB tool to manage their cellphones, and that has caused them to be easily attacked in this way. Security researchers have not paid sufficient attention to the threats related to the ADB tool. Nazar et al. [10] indicate that cellphones that give open root permission can be attacked by devices in a local wireless net.

3.Introduction to the ADB Tool

3.1 ADB Tool Composition

The Android Debug Bridge (ADB) Tool is a client-server program supplied by the Android system to connect phones and external devices. As shown in Figure 1, the ADB tool consists of three parts: ADB Client, ADB Server, and ADB Daemon (ADBD). ADB Client and ADB Server normally run on external devices, while ADB Daemon runs on Android phones.

3.2 ADB Commands

External devices operate Android phones via commands that the ADB Client sends. According to references of Android development [11], there are various types of ADB commands used to realise several functions, most of which are intended to bring convenience to developers. These commands are as follows:

(1) adb install <path_to_APK> installs an app from external devices.

(2) adb push <local> <remote> pushes files to phones from external devices.

(3) adb pull <remote> <local> copies file from phones to external devices..

(4) adb tcp:<portnum> opens network debug.

(5) adb backup/restore <local> command backup will backup files on the phone to external devices.

3.3 ADB's Two Types of Work Modes

In the default situation, cellphones connect external devices via USB. Essentially, the ADB tool is a client-server program. The connection between devices is realised via sockets, and transmit sockets via the USB line in default mode if users do not choose another approach.

In an earlier phase, developers could debug an Android phone only via USB mode. However, after adb v1.0.25, TCP methods were added to the Android system. When ADBD starts, it first checks whether service.adb.tcp.port is set. If it is set, TCP mode will be adopted as the connect method,ADBD will connect external devices via TCP mode. This means that the Android phone will become a client that will respond to adb commands sent by external devices via a network (LAN or WLAN).

4.Security Threats

The original purpose of the ADB tool was to provide convenience for developers to debug their applications. The reason why the ADB tool is a security threat is that the tool that was originally provided for developers is not restrained by Android system security mechanisms. Figure 1 shows the classic Android system architecture.

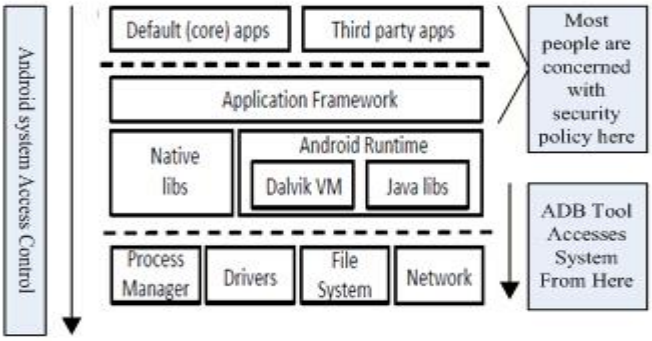


Figure 1. Android System Architecture and the ADB Tool

Current Android system security mechanisms work mainly at the framework or application levels, such as PackageInstallerActivity, which manages the Application permissions mechanism. Ruben et al. [12] proposed the Android system's access control, which is based on the application. The security framework at the top level can control applications well at the top level in accessing the file system. However, the ADB tool runs at the native level, unconstrained by security mechanisms and access control policies at the top level. When external devices operate the system via the ADB tool, serious threats occur.

4.1 Installing an App Silently

In default mode, apps not downloaded from play store [13] cannot be installed in the Android system, which is one of the Android security policies. This is because Google inspects apps for security and confirms their developer's identities before releasing them in the play store. Only after users select the "unknown source" option setting can users install apps that are not from the play store; the installation will continue to be protected by the Application permissions mechanism.

The Application permissions mechanism is one of the most important security mechanisms in Android [14]. When users click to install an app on a phone, there will be a security warning provided by the Application permissions mechanism reminding users

of the permissions that will be used when the app is running.

When users choose to install apps through the ADB tool, unofficial apps can be installed on their phones even if they have not selected "unknown source". This means that even if users trust only official apps, attackers can also install unofficial apps via the ADB tool; the process of installation is silent, which means that the Application permissions mechanism is not applied. Users will not see any of the app's permissions.

4.2 Pushing Malicious Privilege Escalation Files and Copying Private Files

Files in external devices can be pushed to phones via the command adb push <local> <remote>. The security risk is that attackers can push malicious programs to users' phones to escalate privileges or implement other malicious actions. Many Android fans root their phones using tools released by hackers that are mostly based on the ADB tool, such as motochopper [15]. These root programs run on computers and push an executable file to Android phones via adb push, then get root privilege. We can find the adb commands in the motochopper run.bat files as follows:

```
adb push pwn /data/local/tmp/pwn          //push file pwn to /data/local/tmp at phone
adb shell chmod 755 /data/local/tmp/pwn   //change pwn's execute permission
adb shell /data/local/tmp/pwn             //execute pwn
```

Hence, root privileges can be obtained using the adb push command. In attacks, motochopper can root phones to obtain superuser privileges without the users' awareness because this process is transparent to users. As a result, the attackers can use this approach to push malicious files in external devices to users' phones to obtain root privileges and then implement other actions with root privileges.

Files on phones can be pulled to external devices via the command adb pull <remote> <local>. The security risk is that attackers can copy private phone files to external devices stealthily. In Android, users' private files are mostly found in directories /data and /sdcard. Files in /sdcard, including users' photos, videos and so on, can be pulled to external devices directly. Directory /data is a protected directory whose files cannot be read or be written by the ADB tool, so it is not possible to pull files from this directory to external devices directly.

However, with the help of the privilege escalation mentioned above, after getting root privilege, the ADB tool can be given the permission to read or write files in directory /data; files can then be pulled to external devices with the ADB tool, including users' contacts, contact records, messages and all data from apps.

4.3 Security Threats in TCP Debug Mode

As noted in section 3. 1, the ADB tool can be divided into three parts: ADB Client and ADB Server run on external devices and can be regarded as two parts of one program, while ADB Daemon runs in phones. ADB can be used on many platforms so that developers can debug their programs on multiple platforms. In releases later than Android 4.0, the ADB tool is included in the system, which means ADB Client and ADB Server can run in Android phones, so all three parts of the ADB tool are in the phones. That means an Android phone can debug itself; in other words, malicious apps can debug an Android phone by sending adb commands. However, if tcp mode is not opened, the result of running the ADB tool on phones is "cannot find the device" because USB mode is the default mode. When adb commands are issued, the adb server will try to find the USB connection. After opening TCP mode and connecting with wifi, the ADB Server in the phone can connect to the ADB Daemon in phone with the command adb connect

<IP_address> <port>; this means the apps can debug Android phone in which they reside. The security risk is much more serious than has been previously indicated [10] because attacks from apps within the phone are much more serious than attacks from a LAN. In this way, malicious apps in the phone can use the ADB tool for their own purposes and follow the procedure in sections 4.1 and 4.2, installing other app silently or transmitting private files to a server and so on.

5. Demonstration of Attacks

Raspberry Pi [17] is a single-chip computer based on Linux; it is credit card sized, which means that it can be put into a malicious portable power source. As noted in section 4.3, ADB Client and ADB Server can be compiled for many platforms, so we can easily obtain an ADB tool that can run on Raspberry Pi. The following demonstration of attacks is based on Raspberry Pi running a malicious script. The attacks begin as soon as an Android phone connects with Raspberry Pi. Our experimental phone is the SAMSUNG Galaxy Note 2 with Android version 4.1.2.

Demonstration 1: Install Apps Silently

Figure 2 shows the permission requests when two compass apps downloaded from a third-party market are installing:

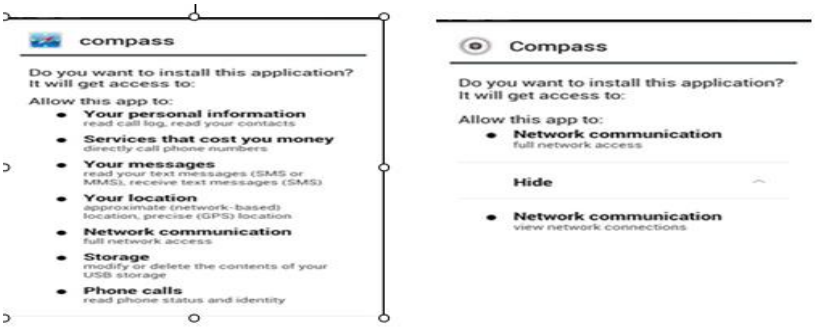


Figure 2. Two Compass Apps' Permission Requests

Because the second app is merely a compass app, almost every user will choose to install it because there are too many dangerous permissions required in the first. Our demonstration is a phone that installed the second compass app. After connecting to the malicious device, the compass app in this phone will be replaced with the former one stealthily. We can see the result in figure 3:

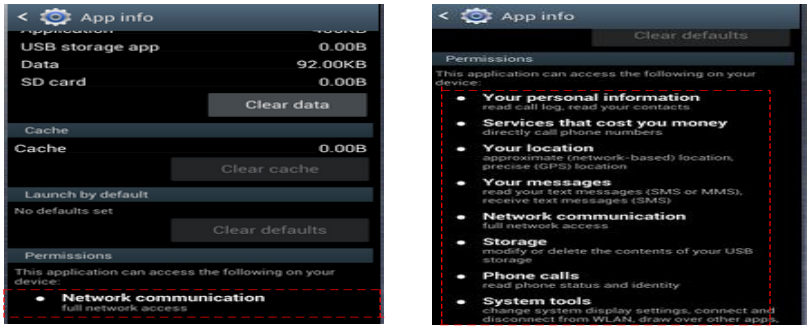


Figure 3. App Changed after Connecting with the Malicious Device

The two apps outputs here are different in appearance, but if attackers make a malicious app whose output is deliberately identical to that of a popular app like Angry Birds in users' phones, it will be hard for users to distinguish between them. The technology of repackaging apps can easily make a malicious app look the same as an app already in users' phones.

Demonstration 2: Push malicious file, Copy private files and Built in malicious app

After the phone connects to Raspberry Pi, a privilege escalation program will be pushed into the phone to obtain root privileges and copy private files (we choose contacts here) back to Raspberry Pi. The malicious compass application of demonstration 1 will be installed to the directory /system/app, from which users cannot uninstall it, and it will start itself every time the user turns the phone on.

Contacts2.db is in /data/data/com.android.providers.contacts/databases/contacts2.db in the phone's file system and cannot be copied directly via the adb pull command. To be copied, the file's properties must be changed after obtaining root privileges. Therefore we execute the command adb shell chmod 660 /data/data/com.android.providers.contacts/databases/contacts2.db to set directory /data's properties after obtaining root privileges.

Figure 6 shows the results after Raspberry Pi pushed the malicious compass app into the phone directory /system/app after obtaining root privileges. To do this, directory /system must be mounted as readable and writable by the mount command with root privileges. Once Android starts, it will load apps in /system/app; a normal user cannot uninstall any app in this directory. There is no uninstall option in the compass app, as we can see in figure 4; the app acts like the system process Android Sync Service.

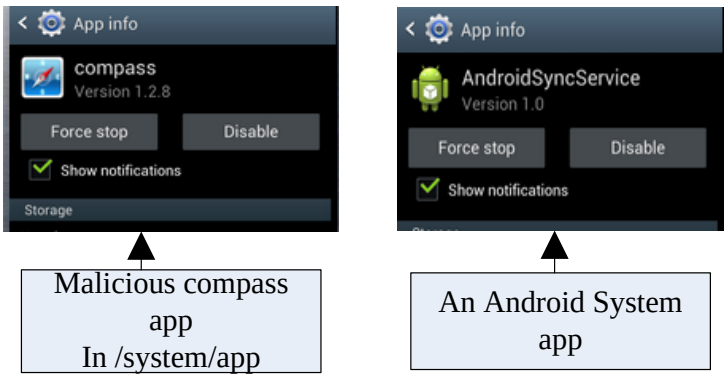


Figure 4. The Compass App that Users Cannot Uninstall

Demonstration 3: Security Risks of TCP Debug Mode

As introduced in section 4.3, using TCP Debug mode brings serious security risks. Figure 5 shows how, after opening TCP debug mode, a terminal emulator app in the phone can operate the phone via adb commands.

Figure 5. After Opening a TCP Mode Terminal Emulator App, Install Malicious App in the Phone

Figure 5 shows the results of a normal permission terminal emulator app using adb commands to install a malicious app after TCP mode is opened. First, it connects to the ADB Daemon via command adb in which the target IP address is the Android phone's IP address, which can be found by command netcfg. Second, it checks the ADBD that connected the terminal emulator to ensure that the terminal emulator app has connected ADBD. Finally, it installs a malicious app (the malicious compass app mentioned above). This process is executes completely on the Android phone; a normal app truly operates the Android phone via the ADB tool, and there is no security warning.

6. Security Suggestions

When improved versions of Android have been released, ADB tool functions have been enriched as well, but there have been no significant improvements in security. The only security improvement has been that in versions later than adb v1.0.31, which is built into Android 4.2.2, when an external device launches an adb connection to the Android phone, there is a confirmation in the phone, as shown in Figure 6.

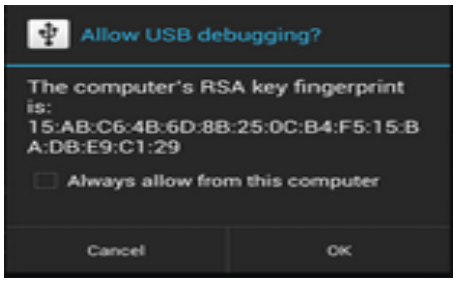


Figure 6. ADB Connection Confirmation

This is an excellent improvement. However, this improvement is insufficient as long as some users do not know what to do when it appears. Attackers can deceive users with a "click ok" or other social engineering methods. We can see that Google is concerned about potential abuse of the ADB tool that can threaten users' security.

6.1 An Android Distinction between Developers and Normal users

In Android system parameters, a parameter called ro.kernel.qemu can be set in system file /data/local.prop. In users' Android phones, the parameter's value is 0; while the Android phones are in the factory, its value is 1. This parameter's value determines the ADB shell's privileges. In the factory, the ADB shell's privilege is as root, which gives the ADB tool more power to debug Android phones. Out of the factory, the ADB shell's privilege is a normal shell, so that the ADB tool cannot be used to modify system

attributes (though root privileges can still be achieved via the root exploit above). Therefore, the parameter can be used to distinguish the role of the ADB tool. Developers are usually knowledgeable about how to modify the system, so they can change the value of this parameter to put the ADB tool into developer mode. The developers will then not be limited by the security mechanism that protects normal users. Normal users do not know how to change the system parameter, which protects them.

6.2 Security Mechanisms to be Added to the ADB Tool

(1). A password must be implemented for ADB connections. Essentially, the ADB tool is a client-server program that can be used via connections. A security password can be set up for connections so that malicious external devices cannot connect to users' Android phones without their consent.

(2). Giving a permission warning when app is installed. When installing apps to Android phones via the ADB tool, Package Installer Activity also must be called to give a permission warning so that users can be told of the potential risk of the apps.

(3).Transmissions of files must be confirmed by users. In the attack demonstration, pushing files to the phone from an external device and pulling files from the phone to external devices both bring security threats. In the process of using the ADB tool, a file transmission should obtain users' confirmation whether it is an adb push or adb pull.

7. Conclusion

In this paper, we have analysed the ADB tool that is basis of the popular software assistants and noted the security threats of using it. To demonstrate that the threats truly exist, we have implemented demonstrations in which malicious external devices attack an Android phone via the ADB tool. From the demonstration, we can conclude that Android devices are vulnerable in the face of malicious attacks from external devices. This is because Android's security mechanisms for the ADB tool are not sufficient. Although Google has realised this and improved the mechanisms, it is not yet perfect. We have demonstrated analyses of how to attack Android phones with examples of chargers and portable power. Finally, we have provided some suggestions for the ADB tool to be realised in future work.

References

[1] BlueBox Inc. , "Commentary-on-the-android-master-key-vulnerability-family", https://bluebox.com/technial/commentary-on-the-android-master-key-vulnerability-family/, **(2013)**.

[2] S. Smalley and R. Craig, "Security enhanced (se) android: Bringing flexible mac to android", 20th Annual Network and Distributed System Security Symposium (NDSS), **(2013)**.

[3] M. Nauman, S. Khan and X. W. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints", Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ACM, **(2010)**.

[4] PingWest Inc., "wandoujia-impressive-number", http://www. pingwest.com/wandoujia-impressive-number/, **(2013)**.

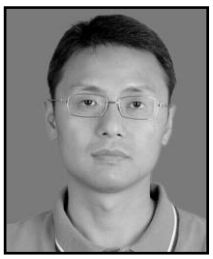
[5] H. Banuri, M. Alam, S. Khan, J. Manzoor, B. Ali, Y. Khan and X. Zhang, "An Android runtime security policy enforcement framework", Personal and Ubiquitous Computing, vol. 16, no. 6, **(2012)**, pp. 631-641.

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

[7] C. Lee, J. Kim, S. J. Cho, J. Choi and Y. Park, "Unified security enhancement framework for the Android operating system", The Journal of Supercomputing, vol. 67, no. 3, **(2014)**, pp. 738-756.

[8] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss, "'Andromaly': a behavioral malwa re detection framework for android devices", Journal of Intelligent Information Systems, vol. 38, no. 1, **(2012)**, pp. 161-190.
[9] Black Hat, "Malicious Chargers", https://media. blackhat. com/us-13/US-13-Lau-Mactans-Injecting-Malware-into-iOS-Devices-via-Malicious-Chargers-WP. pdf, **(2013)**.
[10] N. Assem, M. M. Seeger and H. Baier, "Rooting Android–Extending the ADB by an Auto-con necting WiFi-Accessible Service", Information Security Technology for Applications, Springer Be rlin Heidelberg, **(2012)**, pp. 189-204.
[11] Google Inc., "Android Debug Bridge", http://developer. android. com/tools/help/adb. html.
[12] R. J. G. Vargas, E. A. Anaya, R. G. Huerta and A. F. M. Hernandez, "Security controls for An droid", CASoN Google Inc. , "Google play", https://play. google.com., **(2012)**.
[14] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev and C. Glezer, "Google android: A co mprehensive security assessment", IEEE Security and Privacy, vol. 8, no. 2, **(2010)**, pp. 35-44.
[15] Motochopper., http://forum.xda-developers.com/showthread. php?t=2233852.
[16] ADB Restore, http://seclists.org/fulldisclosure/2013/Jun/115.
[17] Raspberry Pi., http://www.raspberrypi.org/.

Authors



Zhang Wei, Male, he was born in Taizhou, China in 1973. Now, he is a professor in Nanjing University of Posts and Telecommunications, and his research directions are social computing, information security and so on.

YangChao, Male, he was born in Changzhou, China in 1990. Now, he is a graduate student in Nanjing University of Posts and Telecommunications, and his area of interest is Android security..

Chen Yunfang, Male, he was born in Zhenjiang, China in 1976. Now, he is an associate professor in Nanjing University of Posts and Telecommunications, and his research directions are social computing, artificial immune and so on.

