ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (338-347), Month: April - June 2016, Available at: www.researchpublish.com

An Introduction into Android Debug Bridge and Rooting

Nelson Devasagayam

Master of Computer Application, Mumbai University

Abstract: **The Android Debug Bridge (ADB) has attracted much attention from researchers, because it has a high privilege level and a low level of protection. Many attacks on Android systems have taken advantage of the security holes of ADB. Thus, in the updating patch of Android 4.2.2, a new security feature secure USB debugging was implemented so that only trusted hosts can use ADB. This research provides a detailed introduction of the ADB Tool and also studies the features of Android Rooting. Finally, some suggestions are proposed for security improvements to address threats introduced by the ADB tool.**

Keywords: **Android Debug Bridge (ADB), Android systems, security improvements, Android Rooting.**

1. INTRODUCTION

Debugging is the process which helps us detect and fix errors and bugs in a software which is under development, hence Debugging is an important procedure in software development lifecycle. Debugging process traces memory allocation, API usages and subroutine calling stacks. The software is totally transparent to the debugger when it is under debugging. Though so many details of the software can be fetched, the debugging process itself is generally not deemed as a dangerous process. That is because in general systems, debugging functions always come with a software development kit (SDK) for an integrated development environment (IDE), which is provided by trustworthy third parties or from the system's designers. Thus, only selected systems can run in the debugging mode to let applications be debugged on them. Usually those systems for debugging are restricted by extra security policies, and are not worthwhile to hack into for their lack of important data.

Android is an operating system (OS) designed for mobile/portable devices. In mobile devices, hardware capability is highly limited, and Android is designed for multitasking based on limited system resources. It is developed based on a Linux kernel with multiple virtual machine processes, called Dalvik, running to support its multitasking feature. Because of this lower performance (compared with non-portable computers), one important characteristic of Android is that, most android developers usually develop their APPs in an IDE running on regular computers, and compile them and then send the APPs to the Android environment for debugging.

To run and test the APP, the USB Debugging Mode must be turned on in Android devices. By default, it is disabled. Once an Android device is connected to a computer with a USB cable, the USB Debugging Mode can be enabled. This mode can authorize the device to establish a connection between an Android device and a computer using the Android Debug Bridge (ADB) utility. It also allows the computer-end software (primarily SDKs and IDEs) to read the debugging information of the tuned APP.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (338-347), Month: April - June 2016, Available at: www.researchpublish.com

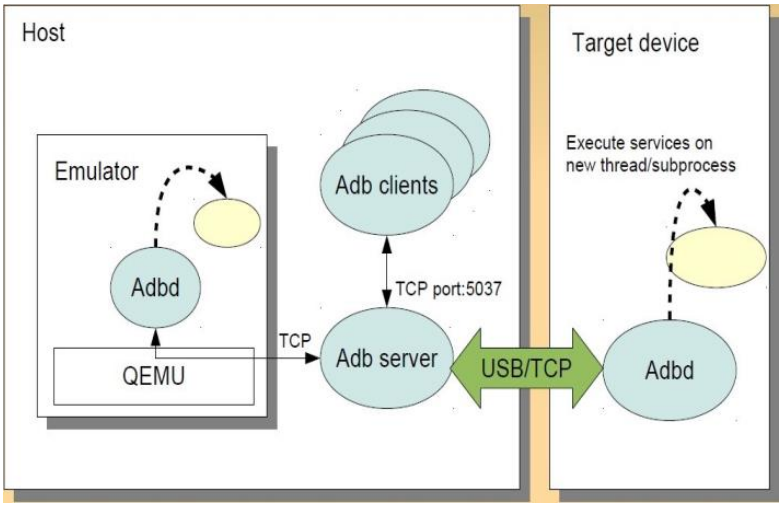


Figure: Structure of Android Debug Bridge.

Android designers provided ADB to help developers easily connect to the Android device from their development machines to debug APPs. As a default component of Android, the ADB has three parts that separately run on the host (desktop/laptop computer) and the device (Android testbed). The ADB on the device is a daemon process which receives commands from the host, executes them and returns the results. The host part of ADB is composed of the ADB server and a command line client.

The major functionality of the ADB server is to monitor the connection between the host and the device. The command line client is an interface for getting user input commands and sends them to the device via TCP/IP or USB connection. This paper will discuss the security of the ADB, with the primary focus on the new security feature of ADB, secure USB debugging, which was introduced in Android version 4.2.2, and allows only authorized hosts to use ADB. This feature will undoubtedly enhance the security of ADB and the overall Android system. However, a detailed analysis of the feature is needed in order to understand its functionality, effects and drawbacks. In this paper, we will perform a thorough analysis. Through the analysis, it was identified as a security issue that cannot be fully addressed by this feature. Therefore, it is possible that potential attacks can be launched by exploiting it.

2. ANDROID DEBUG BRIDGE (ADB)

Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device. It is a client-server program that includes three components:

A client, which runs on your development machine. You can invoke a client from a shell by issuing an adb command. Other Android tools such as DDMS also create adb clients.

A server, which runs as a background process on your development machine. The server manages communication between the client and the adb daemon running on an emulator or device.

The ADB Daemon runs on an Android device. Whether or not it runs is controlled by the “USB Debugging” setting inside an Android device’s settings menu. As the name of the setting suggests, it enables communicating with the device over USB, but also supports using a TCP port for communications.

You can find the adb tool in <sdk>/platform-tools/.

2.1 Privileges of USB Debugging Mode:

In USB Debugging Mode, users are granted special permissions to facilitate the actions required during the debugging process. Due to the reason that source code must be modified frequently, the APP is sent from computer’s IDE and installed on the device. The debugging mode let users install/uninstall APPs freely, without the restrictions which are applied to the normal Android installation process. Users are allowed to trace the APP’s actions such as APIs called, memory allocated, and system settings influenced. The activities of an APP are able to be tuned manually in the USB Debugging Mode. Also, there are some special commands which can only be executed in this mode.

Basically, the realization of the above privileges depends on ADB utilities and the communications through the USB cable connection. All the debugging functionalities can only be possible through the connection established between the ADB server and the client. By using ADB, users can directly install APPs to the connected device, and the Android system will automatically give APPs all the permissions they need, without any further security check. The debugging information is collected by the Android system and is sent via the ADB connection. The special utilities will be functional when they receive commands issued from ADB. In addition, the ADB can act as a terminal client to the Linux kernel on Android devices.

2.2 Sensitive Commands of ADB:

ADB provides a wide range of functions for the interaction between the host and the device. These functions are usually executed by typing commands in a command-line interface on the host. Some of the commands are security sensitive. For usability reasons, a very high privilege level is given to ADB. Once an Android device is connected with a host through ADB, all commands can be directly executed without any further authorization.

The ADB command "install" can enable users to install a new package to the system. That means if an attacker subverted a device through ADB, he/she can silently install malicious APPs to it, and those dangerous APPs will then be granted all the permissions they need from ADB. In addition, the "push" and "pull" commands can transport files between the host and the device. They can send files to and get files from any directories on the device, either in the system storage or the memory card.

In addition, it was found some ADB utilities are not included in the Google's

Android documentation website. One critical function of them is the "tcpip" command. This utility is used to restart the device's ADB daemon to enable it to listen for the TCP/IP requests on a specified port. By using this function, a host can connect to a device's ADB utility via a TCP/IP network. Compared with the USB connection, it is more flexible with less restriction.

Some ADB Commands

(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.

2.3 ADB Security Issues:

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. By taking advantage of ADB connections, users could install APPs with any permission they want. Those permissions are the basic access control components in Android. APPs with all permissions are granted full access to the whole system. It can read/write any file in the storage, control voice calls and SMS messages, change system settings, and read all the account information on the device. If the ADB feature is used by a person who is not the owner of the device, it may result in severe personal information leakage. Through the ADB connections, users are also able to input commands to the

Linux kernel. Several methods are available for obtaining the root user privilege. Users will take full control of the system if the device is "rooted", and some of the rooting methods are completed by interacting with kernel through ADB.

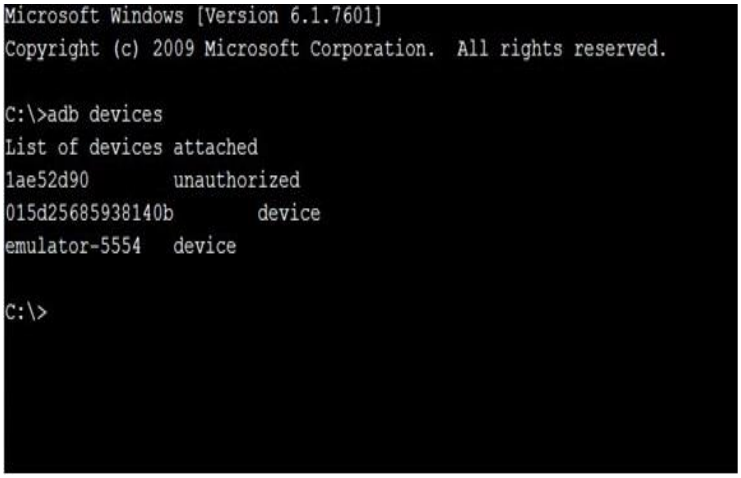
With ADB, high privileged operations can be performed to control the Android device. Therefore, it has become an attractive attack vector. There have been a number of attacks against Android systems by exploring ADB's security hole. In recent years, many security holes and potential problems caused by ADB have been found.

2.4 ADB Services:

This section introduces the services provided by ADB. All services are come from two components of ADB utility. The first one is ADB daemon, and the second one is the host server. ADB daemon is a daemon process of the Linux kernel of

Android system [29]. This daemon receives service request from remote client. Based on the different type of services being requested, daemon will execute certain instruction locally and return the result. So, the services provided by ADB daemon are called "local services". The host server is a background process runs on the host machine that connects to Android devices. This server process manages all ADB connections between machine and each device. Thus, it also provides services for client for the purposes of maintaining and managing those ADB connections. This type of services is called "host services".

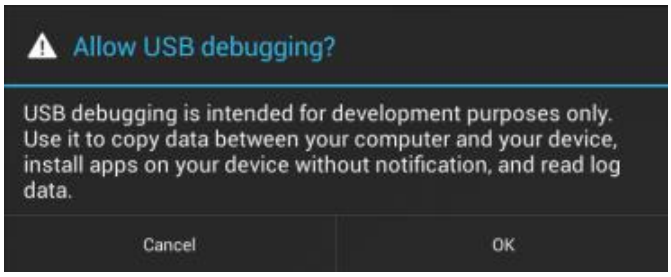
The host services provide functionalities for maintaining ADB connections with Android devices [31]. For example, the "listing device" service can return client a list of devices currently connecting with the host machine. The corresponding command for this service is adb devices. We have introduced several security sensitive commands. Here we talk about some ADB services which should be concerned for security reason. The total number of services is huge, and we cannot discuss all of them in the paper.



[Listing Devices using ADB]:

3. SECURE USB DEBUGGING

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 Updating patch 4.2.2. It hides the USB mode enabling option from the system setting menu. In order to make the option checkbox visible, a user needs to touch seven times on the "Build Number" section under the "About Phone/Tablet" menu in system settings. The purpose of this design is to prevent users from accidentally turning on the USB debugging mode. However, it is hard to deem it as a security improvement.



[Activating USB Debugging]:

Thus, in the 4.2.2 update, designers introduced the new security policy on ADB called "secure USB debugging". In this solution, only authorized hosts are allowed to use the USB connection with the device. If a device is connected to an unauthorized host, the host will not see files on the device and cannot establish an ADB connection to the device. The authorization process is completed by the device administrator when a new host is connected for the first time. After activating the device, a prompt dialog will ask the administrative user to confirm the authorization. Once the confirmation is made, the device will communicate normally with this host. If the administrator chooses to always allow this host, it will be added to the white list of the device and no authorization will be required when subsequent connections occur [3].

3.1 The Need for Secure ADB?

If you've done any development, you know that "debugging" is usually the exact opposite of "secure." Debugging typically involves inspecting (and sometimes even changing) internal program state, dumping encrypted communication data to log files, universal root access, and other scary but necessary activities. Debugging is hard enough without having to bother with security, so why further complicate things by adding additional security layers? Android debugging, as provided by the ADB, is quite versatile and gives you almost complete control over a device when enabled. This feature is, of course, very welcome when developing or testing an application (or the OS itself), but it can also be used for other purposes.

Here's a selective list of things ADB lets you do:

• Copy files to and from the device

• Debug apps running on the device (using JWDP or gdbserver)

• Execute shell commands on the device

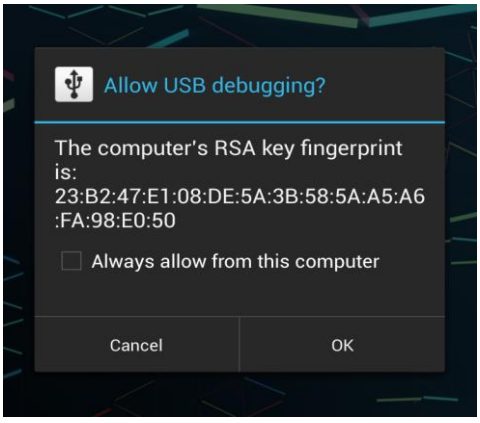
• Get the system and apps logs

• Install and remove apps

If debugging is enabled on a device, you can do all of the above and more (for example, inject touch events or input text in the UI) simply by connecting the device to a computer with a USB cable. Because ADB does not depend on the device's screen lock, you don't have to unlock the device in order to execute ADB commands, and on most devices that provide root access, connecting via ADB allows you to access and change every file, including system files and password databases. Worse, you don't actually need a computer with development tools in order to access an Android device via ADB; another Android device and a USB On-The-Go (OTG) cable are sufficient. Android tools that can extract as much data as possible from another device in a very short time are readily available. If the device is rooted, such tools can extract all of your credentials, disable or brute force the screen lock, and even log into your Google account. But even without root, anything on external storage, most notably photos, is accessible, as are your contacts and text messages.

3.2 The Implementation of Secure USB Debugging:

In the design of secure USB debugging [2], a 2048-bit RSA encryption method is used to secure the authentication process. The host's private key and public key are generated from its desktop ADB server utility. The host's public key is treated as the host identity. When the device is connected to a host, the device sends a 20-byte random message to the host. Then the host encrypts the message's SHA1withRSA signature using its private key and sends this encrypted signature back. The device will decrypt this signature and compare it with the signature calculated from its original message. If the verification fails, the device will stay in the offline status, which means the host cannot perform any action on the device. Failure of the verification may be a result of no corresponding public key on the device or the two signatures do not match. If the device does not contain this host's public key in its storage, the host will send its public key to the device before it sends back the verification message. After receiving the public key, a confirmation dialog shows on the device with the key's MD5 hash waiting for user actions. If the user selects "OK", then the device will use this key to decrypt the verification message. If the "Always allow from this computer" checkbox is checked, the device will save this key in its storage drive.



[Secure USB Debugging confirmation dialog for first time connection.]:

This method provides a white list design for ADB connection authorizations for the sake of better security and usability. It does not require lots of user actions to authorize a host; instead, just a simple click would suffice. This security design helps to protect Android systems from malicious attacks originated from ADB connections. All the unknown hosts are not allowed to connect to the protected device. Some attacks begin with establishing the ADB connection with the Android system and then try to obtain administrator privileges by sending and installing "rootkit" files to the system via the ADB connection. Now these attacks cannot work anymore because they are not allowed to connect to the target Android devices.

Naturally, this secure USB debugging is only effective if you have a reasonably secure lockscreen password in place.

4. ANDROID ROOT ACCESS

In the preceding chapters, we introduced Android Debug Bridge and discussed how implementing Secure USB Debugging into Android has reinforced its security. In this chapter, we take a bit of a right turn and introduce methods that can be used to circumvent Android's security model.

In order to perform a full OS update or to restore the device to its factory state, it's necessary to escape the security sandbox and gain full access to a device, because even the most privileged Android components are not given complete access to all system partitions and storage devices.

Additionally, while having full administrative (root) access at runtime is clearly against Android's security design, executing with root privileges can be useful in order to implement functionality not offered by Android, such as the addition of custom firewall rules or full (including system partitions) device backup. Indeed, the wide availability of custom Android builds (often called ROMs) and apps that allow users to extend or replace OS functionality using root access (commonly known as root apps) has been one of the reasons for Android's success.

4.1 Introduction:

Rooting an Android device is interesting for two different groups of people. On one site it is interesting for the user itself, because he could get more control over the phone and do tasks with the phone which are not allowed or possible with an unrooted phone. Restrictions on an unrooted phone may occur because the manufacturer does not like the user to do some things or because there are security restrictions. Beside the user also an attacker may be interested in gaining root access. If an attacker could achieve root access he could do whatever he wants on the victims phone without asking permission to do sensible actions or access secret data.

4.2 Advantages of rooted Smartphones:

For the user a rooted smartphone could have lots of advantages and gives him full control over the phone. It is for example possible to customize the look and feel of the user interface, uninstall unwanted applications of the manufacturer or provider which were shipped with the phone or tune the phone for better speed and battery life. There are also applications available, which are just running on a rooted phone. For example automated backup applications, ad blocking applications or applications for using wireless tethering even if not allowed by the telecommunication provider.

4.3 Risks of a rooted Smartphone:

A rooted device has lots of advantages, however it also brings some risks for the user. One risk is that the user gives applications permissions to do everything on the device. The default permission system has no effect for applications which are granted root access. And because many applications which are providing additional features are closed source, there are just limited ways to control what they are really doing under the hood of the phone. There are applications which at least give the user the control over which applications are allowed to gain root access, but if these applications have security vulnerabilities it is maybe possible to bypass the restrictions and every installed application is able to gain root access easily.

4.4 How Rooting works:

The process of rooting itself is very straightforward. It is just necessary to copy a su binary to one location which is set in the systems PATH variable and make the su binary executable. The difficulty is to gain the permission to do so. There are two different ways to achieve this goal and root a phone: It is rather possible to Soft Flash the running system or to flash a modified ROM to the device.

4.4.1 Custom ROM Flashing:

The first method is to flash a ROM with a modified operating system, which already contains the su binary, to the device. In the most cases a ROM with an aftermarket Android distribution like CyanogenMod is used. The most alternative distributions are heavily modified and provide additional features for the smartphone out of the box.

The biggest challenge of this method is to circumvent the boot loader of the device. The most manufactures ship their phones with a locked boot loader, which means that just ROMs which are signed by the manufacturer are runnable on the device. This is a security function but also an attempt of the manufacturer to gain control over the software and apply restrictions. In earlier days the most boot loaders were locked and it was thus necessary to find a security vulnerability in the firmware to unlock or replace the boot loader. Today many manufacturers accepted that the users like to customize their phone and thus officially provide a method to unlock the boot loader.

1. Advantages and Disadvantages

The biggest advantage of flashing over exploiting is that the root access is permanent. The disadvantage is that updates must be shipped by the ROM provider which is often a private person or group which is not interested in long term support of a free ROM.

2. Risks

One risk of rooting a phone by flashing is that it is necessary to trust the ROM provider. The ROM provider may have not just added the su binary, but also added a backdoor which allows him to fully control the device over the network or use it in his botnet.

A further risk is to loose the warranty of the phone. Using a custom ROM is not allowed by the most manufacturers and if a user violates this rule, he looses the warranty. This is also true if the manufacturer provides an official method to unlock the boot loader. It is also not possible to unroot the phone before using the warranty because many devices have a counter build in which increases every time a new ROM is flashed to the device and a violations to the warranty terms is thus easily detectable.

3. Unrooting

It is possible to unroot a rooted device by flashing a recovery ROM provided by the manufacturer of the device. The process is the same as for rooting the device originally, just with another ROM.

4.4.2 Soft Flashing:

Soft flashing is for users who like to keep the factory ROM provided by the manufacturer of the device nearly untouched. With soft flashing just the needed su binary is added to the stock operating system and a root control application is installed. Soft flashing shares the risks of the custom ROM flashing method.

For soft flashing it is necessary to flash a custom recovery image to the smartphone. When the phone is booted into recovery mode, a script is executed, which copies the su binary and the root control application to the devices main operating system. When both components are copied, the original stock recovery image is installed again and no traces (except an incremented flashing counter) are left.

1. Unrooting

To unroot the device it is just necessary to delete the su binary. This could be done with some root control applications, like SuperSU.

5. SECURITY THREATS

The tremendous strides made by Android as an operating platform for mobile phones and tablets has not gone unnoticed by malware developers and hackers. More than a billion devices have been sold and that offers tremendous opportunities to such developers.

They are also helped by the fact that most users still do not perceive their device as something that can be hacked into and are lax about security. They do not understand or think that the Android gadget is just like their desktop or laptop and is susceptible to virus attacks.

5.1 So What Kind Of Security Risks Do Android Users Face?

Rooted Android device users can access core system files, reconfigure system settings and install apps they otherwise couldn't. But rooting opens a device up to security risks that can compromise sensitive data and jeopardize enterprise resources.

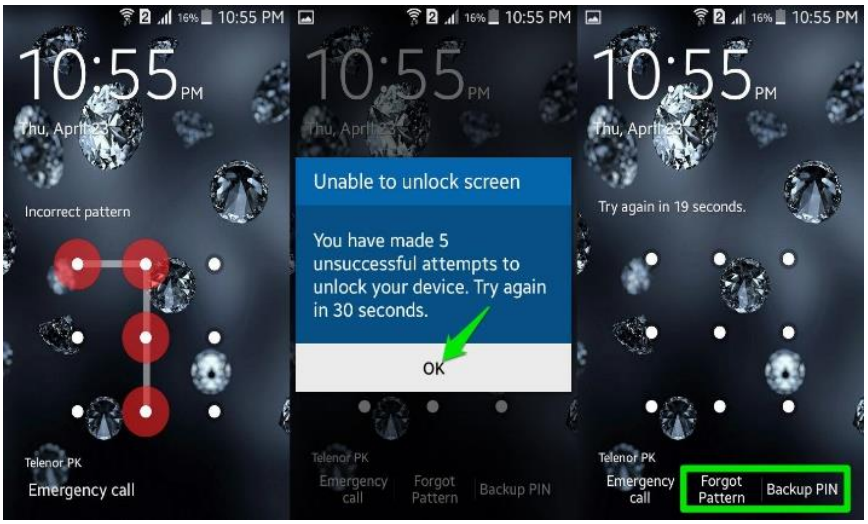
Since Android is a Linux derivative, which implies a certain level of system access and configurability, but Android devices typically ship with limited privileges. Users can install apps from Google Playstore and change some of the system settings, but they cannot access the core operating system or take too many steps that could alter its built-in protections -- at least not without some effort. Each app also runs in its own container with its own user ID, which keeps its operations and data isolated from other apps. This restricted state helps protect against malicious code and other threats.

Android also makes it possible for users to root their devices; that is, override the usual safeguards to install apps that can modify the operating system, access all other apps (and their data) and perform other operations that would normally be restricted. Rooted Android device users can also download any apps from anywhere they want, not just Google Play.

There are many good reasons why users would root their Android devices. For example, they can install advanced backup and security apps that require full system access, and they can uninstall the bloat ware that ships on most devices. Rooting also lets users install updated OS versions -- a handy feature when the device manufacturer fails to provide those updates in a timely manner, as is often the case with Android.

5.2 Security Threats:

Android comes with built-in protection feature to prevent unauthorized use of your phone. The main reason to set up some sort of lock screen security on your smartphone is to keep strangers (or friends) from checking out your messages or private pictures. Beyond that, you don't want anyone who dares to steal your phone to get full access to your mails, pictures or other sensitive data. But what if you're the one who cannot access your phone? You could forget your PIN or pattern, right? Or someone pranks you by setting up a lock screen pattern and just leaves you struggling with it. You can use the medium (Pattern) or high (Password) security, according to your requirement to prevent people from accessing your phone data.



[Android Lockscreen]:

5.2.1 Removing Android Lockscreen using ADB:

Method 1:

Connect the phone to the computer via USB (phone should be turned on).

Open a terminal window (Linux) or cmd (windows).

Type the below commands one by one, pressing enter.

adb devices
adb shell
cd data/system
su
rm *.key
That's all. Reboot the device.

Method 2:

Open terminal (Linux)/cmd (Windows) and type the following commands, each at a time, followed by enter.
adb shell
cd /data/data/com.android.providers.settings/databases
sqlite3 settings.db
update system set value=0 where name='lock_pattern_autolock';
update system set value=0 where name='lockscreen.lockedoutpermanently';
.quit
Done. Reboot the device.

Method 3:

1. SMS Bypass [Root Required] - Download & Install It On Your Device.
2. This App Allows You To Remotely Bypass Your Phone's Screen Lock By Sending A SMS.
3. It Removes Your Gesture Pattern Or Password After Receiving A Preset Keyword Along With A Secret Code Via SMS.
4. SMS Bypass App Requires Root.

Procedure:

1. First, make sure you give permanent root access to the app.

2.Change the secret code to your preferred choice. The default password is: 1234

3.To reset your screen lock, send the following message from another phone:

secret_code reset

Example:

1234 reset

Note 1: There is a space between your secret code and reset. Also the secret code is case sensitive.

Note 2: There is an option available to change the preset keyword. Default is: reset - Your phone will restart and your lock screen will be reset.

5.2.2 Fingerprint Hijacking:

It's bad enough when attackers obtain legitimate passwords -- so what about when they get their hands on fingerprint images?

The researchers have found severe issues with the Android's current fingerprint scanning framework.

5.3 Security Suggestions:

To combat the security loop holes in the Android system, Google has been taking steps like enhancing the security in latest Operating Systems. They have much better security features such as encryption of data by default when the device is switched on for the first time.

Having mentioned the above, Google needs to be wary of “rooting tools” used cleverly by hackers to manipulate system settings of user’s devices. It needs to make its systems more rugged and capable enough to resist malware apps effectively.

Security Mechanism to be added to the Android OS.

(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.

6. CONCLUSION

6.1 Conclusion:

In this research, after a comprehensive introduction about Android Debug Bridge, We analyzed the protection effectiveness of secure USB debugging. The results show that the new feature has increased ADB's security but still lack the capability in defending against the intrusions from the subverted trusted host. We can conclude that Android devices and Android's security are most vulnerable when they are rooted. This is because Android's security mechanisms are not sufficient. Although Google has realized this and improved the mechanisms, it is not yet perfect. Finally we have provided some security enhancement for the ADB Tool.

ACKNOWLEDGEMENT

This paper is a part of academic research on Understanding Application security in android made in partial fulfilment of the course of Masters in Computer Application.

REFERENCES

[1] (2013). Android Debug Bridge | Android Developers [Online]. Available:http://developer.android.com/tools/help/adb.html

[2] Mingzhe Xu(2014). Security enhancement of secure USB debugging in Android system [Online]: https://etd.ohiolink.edu/!etd.send_file?accession=toledo1417536423&disposition=inline

[3] N. Elenkov. (2013). Secure USB debugging in Android 4.2.2 [Online] Available:http://nelenkov.blogspot.com/2013/02/secure-usb-debugging-in-android-422.html

[4] Android, the world's most popular mobile platform [Online]. Available:http://developer.android.com/about/index.html

[5] Android Debug Bridge | Android Studio [Online]. Available:https://developer.android.com/studio/command-line/adb.html

[6] Two Security Issues Found in the Android SDK Tools [Online]. Available:http://www.droidsec.org/advisories/2014/02/04/two-security-issues-found-in-the-android-sdk-tools.html

[7] Android Rooting and Real World Security Threat [Online]. Available:https://www.marcobeierer.com/papers/android-rooting-and-real-world-security-threat

[8] What is USB Debugging? [Online]. Available:http://www.makeuseof.com/tag/what-is-usb-debugging-mode-on-android-makeuseof-explains/

[9] Android Security Internals Android Security Internals an In-Depth Guide to Android's Security Architecture [Book].

[10] Bypass Android LockScreen |http://www.hackcave.net/2015/10/tutorial-hackingbypassing-android.html

[11] Emerging Android Threat |http://www.darkreading.com/vulnerabilities---threats/6-emerging-android-threats/d/d-id/1321117?image_number=5

[12] Rooted Android Device Risks |http://searchmobilecomputing.techtarget.com/tip/Rooted-Android-device-risks-include-network-access-data-theft

[13] Android Rooting Attack |https://www.oneclickroot.com/root-android/almost-every-android-device-found-to-be-vulnerable-to-rooting-attack/