Exploring USB Connection Vulnerabilities on Android Devices
Breaches using the Android Debug Bridge

João Amarante[1] and João Paulo Barros[1,2]
[1]*Polytechnic Institute of Beja, Beja, Portugal*
[2]*UNINOVA-CTS, Monte de Caparica, Portugal*

Keywords: Vulnerability, USB, Smartphone, Mobile Device, Computer Security, Physical Attack, Internet of Things, IoT, Mobile Cyber-physical Systems.

Abstract: The complexity of avoiding vulnerabilities in the modern mobile operating systems makes them vulnerable to many types of attacks. This paper presents preliminary work in the creation of scenarios to surreptitiously extract private data from smartphones running different versions of the Android Operating System. Three scenarios were already identified and a proof of concept script was developed, all based on the use of the Android Debug Bridge tool. When running in a computer, the script is able to extract private data from a USB connected smartphone. In two scenarios it was possible to extract the information in a totally surreptitious way, without the user knowledge. In the third scenario, using a newer version of the Android operating system, a user action is needed which makes the attack less likely to succeed, but still possible.

1 INTRODUCTION

In a world increasingly dominated by complex technologies, their negligent or ill-informed use provides the opportunity for attack scenarios, which from the point of view of programmers would seem improbable. This current generation of smartphones like the Apple iOS and Google Android based devices are powerful enough to accomplish most tasks that previously required a personal computer. In fact, this newly acquired computing power has given rise to many applications that try to leverage new hardware. These include Internet browsing, e-mail, GPS navigation, personalized messages and applications, among others. Additionally, mobile devices provide an ideal support for mobile cyber-physical systems. All these contribute to the presence of a large amount of personal data that is stored inside the device.

Among the many different devices that make the Internet of Things, the smartphones are still the most ubiquitous and also the ones with more users unaware of the related risks. This work focus on security and privacy in mobile systems, namely a specific type of attack to that kind of device: the exploit of USB connection vulnerabilities to target private data in Android devices (Google, 2017).

Android is being used by more than 2.1 billion people around the world (Statista, 2017) and the ubiquitous use and widespread adoption of the Universal Serial Bus (USB) led mobile device manufacturers to equip most third generation phones with USB ports. With such a large number of consumers, the space to produce a malicious attack is huge: whenever someone with an Android device connects it using a USB port the device can potentially become compromised. A small flaw, error, or misinterpretation of a security authority specification can potentially jeopardize the security of those systems. Proximity attacks, namely the ones based on the USB connection, are a good example of this and anecdotal evidence seems to indicate that the associated risks are easily neglected by users; in fact, a similar behaviour was already observed regarding USB drives (Tischer et al., 2016).

Currently, USB connections are primarily used as a means of charging the battery, communicating, and synchronizing the contents of the phone with computers and other phones. An important form of attack occurs when private data is extracted from mobile devices without user permission. The apparent unawareness of many users regarding the risks of connecting their device to a compromised computer led to the theme of this paper: the vulnerabilities in the Android USB connection. The work in this paper is part of a detailed study and evaluation of the USB based vulnerabilities present in devices running the Android Operating System: we present three concrete

Amarante, J. and Barros, J.
Exploring USB Connection Vulnerabilities on Android Devices - Breaches using the Android Debug Bridge.
DOI: 10.5220/0006475905720577
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT*, pages 572-577
ISBN: 978-989-758-259-2

and implemented scenarios resulting from the use of the Android Debug Bridge tool to extract private data from smartphones, following a connection between the Android mobile device and a compromised personal computer. Three scenarios were already identified and a proof of concept script was implemented to extract private data when a smartphone is connected to a computer. In two scenarios, it was possible to extract the information in a totally surreptitious way, without the user knowledge. In the third scenario, using a newer version of the Android operating system, a simple user action is needed which makes the attack less likely to succeed, but nevertheless clearly possible for less aware or knowledgeable users.

Section 2 present background information about the Android system and its security model. Section 3 presents the tested scenarios; Section 4 describes the implementation of the script used to extract data from smartphones without user awareness; Section 5 discusses related work and Section 6 concludes.

2 BACKGROUND

Android is an application execution environment for mobile devices that includes an operating system, application framework, and core applications. Here, we briefly present the problem of USB based vulnerabilities and the ADB tool.

2.1 Vulnerabilities in USB Connection

The USB protocol is fully software controlled; therefore, it is similar on all Android devices. There are two sub-protocols that are supported by Android: (1) the mass-storage class (for example a USB removable disk) and (2) the Android Debugger Bridge (ADB). Each peripheral function has an associated device class document that specifies the default protocol for that function. This allows hosts compatible with the class and peripheral functions to interact without detailed knowledge of each other's operations. Class compliance is potentially dangerous if the host and peripheral are provided by different entities.

The USB connection does not support the network class, audio class, or other classes, such as mass storage class (MSC) or Media Transfer Protocol (MTP). By default, ADB is disabled, and the computer refers to Android as a mass storage device, with none of the additional functions.

Initially, only the device's SD card is exposed via USB, rather than its system and data partitions. When "USB debugging" (ADB) is enabled, the device can be controlled with the same "adb" tool that is provided in the Android SDK. This tool makes it possible to insert and retrieve files from and to the device, install APK files, TCP and UDP redirects, etc.

Due to the vulnerabilities that this process presented, since version 4.4.2 (KitKat), Google has created a security process for this connection: ADB pairing, the pairing between a USB connection and an Android Debug Bridge (ADB). Making sure each USB connection has a pair of RSA keys accepted by the host computer that the Android device will connect to, so that each time the Android smartphone tries to connect to a new host, it must have been previously accepted by it.

Next, we present the ADB tool, which was used to implement the attack scenarios described in this paper.

2.2 The Android Debugger Bridge

The Android Debug Bridge (ADB) (Android Open Source project, 2017) is a command line tool that allows communication with a connected Android device or an emulator allowing several actions, namely through a Unix shell. It facilitates several device actions including app installation and debugging applications. It is a client-server program with three components: (1) **A client**, which sends commands and runs on the development computer — it is possible to issue an adb command to invoke the client from a command line terminal; (2) **A daemon**, which executes commands on a device and runs as a background process on each emulator or device instance; (3) **A server**, which manages communication between the client and the daemon; the server runs as a background process on the development computer.

When starting, an adb client checks to see if there is an adb server process running. If not, the adb client starts the server. When the server is started, it binds to local TCP port 5037 and listens for commands sent from adb clients. All adb clients use that same port to communicate with the adb server.

The server then configures connections to all emulator/device instances that are running. It locates the emulator/device instances with an odd-port scan in the range of 5555 to 5585, which is used by emulators/devices. Where the server encounters an adb daemon, it configures a connection to the port in question. Each emulator/device instance acquires a pair of sequential ports: an even-numbered port for console connections and an odd-numbered port for adb connections.

When the server configures connections for all emulator instances, it is then possible to use adb com-

mands to access those instances. Because the server manages the connections to the emulator/device instances and manages all commands from multiple adb clients, it is possible to control any emulator/device instance of any client (or script). More information about the ADB can be found at the official site from where this information was also extracted (Android Open Source project, 2017).

3 SCENARIOS

Three scenarios for the retrieval of private information were identified. All are associated with the USB connections. After, a script was implemented allowing the following actions:

1. Obtain device information;
2. List of all installed packages;
3. Full copy of the entire SD card contents;
4. Copy a file to the device;
5. Install an application on the device;
6. Run an application on the device;
7. Obtain the contact list;
8. Obtain the messages;
9. Unlock the device screen;
10. Overcoming the ADB Pairing;
11. Rooting the device.

For mobile devices, three devices, each with a different version of Android, were tested. Specific conditions were also tested for each version. USB debugging is essential to exploit the vulnerabilities considered in this work, since the only contact of the phone with the machine containing the script is through a USB cable.

It was decided to write a script for the Windows 7 Home Premium Operating System using one of its available tools: the Windows PowerShell ISE 5.0 [1]. Hence, the file has the .ps1 extension, which is the format associated with Windows PowerShell from Microsoft Corporation. The three scenarios correspond to vulnerabilities that the script is able to exploit. The versions used represent a small percentage of the devices that have recently visited the Google Play Store[2], but the real number of these devices in use is probably quite higher, as older and rooted devices are much less likely to access the Play Store and the number of rooted devices can be higher than usually expected (e.g. (BusinessofApps, 2015; arsTechnica, 2016)). Additionally, it is easy to find sites teaching how to enable USB debugging, apparently with no warnings about the possible associated risks (e.g. (TechAdvisor, 2016; Hacks, 2015)). Also the third scenario is also significant and applicable to newer versions of the Android system. Yet, to date, only version 5.0 Lollipop was tested.

- **Attack Scenario I:**
  Configuration:
  - Hardware: Samsung Galaxy Mini GT-S5570
  - Android version: 2.2 (Froyo)
  - USB debugging on
  - Device rooted

  Achieved results:

  1. Obtain device information;
  2. List of all installed packages;
  3. Full copy of the entire SD card contents;
  4. Copy a file to the device;
  5. Install an application on the device;
  6. Run an application on the device;
  7. Obtain the contact list;
  8. Obtain the messages;

  This is the simpler attack scenario, as the debugging is on and the device is rooted. This device has a specific feature that, in practice, can increase the vulnerability of USB Debugging: the user can ”enable” the connection, but keep it ”inactive”, possibly giving a false sense of protection, but simultaneously leaving the door open to an attack. In this way, and with rooting access to the device, it was possible to obtain all type of desired information, compromising the device security.

- **Attack Scenario II:**
  Configuration:
  - Hardware: Sony Xperia Miro ST23i
  - Android version: 4.0.4 (Ice Cream Sandwich)
  - USB debugging on
  - Device ”unrooted”

  Achieved results:

  1. Obtain device information;
  2. List of all installed packages;
  3. Full copy of the entire SD card contents;
  4. Copy a file to the device;
  5. Install an application on the device;

[1] e.g. https://msdn.microsoft.com/en-us/powershell/scripting/core-powershell/ise/introducing-the-windows-powershell-ise

[2] https://developer.android.com/about/dashboards/index.html

6. Run an application on the device;
   If application roots the device than the following three are also possible:

(a) Obtain the contact list;
(b) Obtain the messages;
(c) Unlock the device screen;

In this scenario there is no rooting access to the device, which would limit the range of possible attacks. However, by enabling USB debugging, the device is still exposed to SD card attacks, namely the extraction of private data.

- **Attack Scenario III:**
  Configuration:
  - Hardware: Aquaris E5 HD
  - Android version: 5.0 (Lollipop)
  - USB debugging off
  - Device ”unrooted”

  Achieved Results:
  It was not possible to extract data from the device. Yet, it was possible to confirm that, if USB debugging is changed to ”on” and after ADB pairing, the script can still extract data .

4 IMPLEMENTATION

The developed script is based on the auto-detection of a USB connection: when a device is connected to a computer, it will be automatically detected and the script will be started. Once the device is identified, various attempts to obtain information will be made through Windows PowerShell ISE 5.0 and/or Android Debug Bridge (ADB) commands. All these processes are invisible to the victim and through them one can access the entire contents of the SD card and also various information of the device. At http://y2u.be/TDgUgxgOt_o, a screencast shows the script being run when a smartphone is connected to the compromised host computer. Next, we present what is executed by the script and the respective results for each of the actions listed in Section 3:

1. **Obtain Device Information:** It was possible to obtain the identification of the device, as well as information about it, such as the letter assigned by the Operating System, the Android version, and the model. These data includes the paths to execute the intended attacks.
2. **List of All Installed Packages:** Obtaining a list of all packages installed on the device was then used to get the exact name that the system assigns to each package. This was useful to retrieve data about the application that was installed in order to execute it. In the item **Run an application on the device** the usefulness of this information will be explained.
3. **Full Copy of the Entire SD Card Contents:** Due to the little storage space that the devices have, most users save a large amount of personal content on the SD card. This action retrieves all the files and folders of the card such as photographs and movies.
4. **Copy a File to the Device:** Once the script gets the contents of the SD card, it can copy a file (in this case an application that allows you to do Android Rooting) to the device and verify that it was on the card. This process allows putting in the SD card any type of file. If it is a malicious one, it becomes possible to compromise the whole device.
5. **Install an Application on the Device:** As the file copied to the device was an application it became possible to perform its installation. This process is extremely dangerous for the device if the application being installed is some type of malware, capable of corrupting it in any way.
6. **Run an Application on the Device:** After installing the application on the device is was possible to execute it, which in the case of being a malicious application raises serious security problems This process happens due to the fact that it is possible to obtain a list of all installed applications and their information. From this list it is possible to get the exact path to execute the full command to run the application.
7. **Obtain the Contact List:** One of the main results was the retrieval of private information from the device. In this case the contents of the contact list have been copied. This is only possible if the device is rooted, but since it was possible to copy, install and run applications, with USB debugging on, it was possible to do this invisibly to the victim and obtain this type of information. After granting root access, it is then possible to change access permissions to the protected content, so that it is possible to give the command to copy the information, which allowed the retrieval of the desired list in SQL and plain text format.
8. **Obtain the Messages:** The content of all messages in the device was another type of retrieved data. Using the previously described process it was also possible to obtain the information of the respective list in SQL and plain text format.
9. **Unlock the Device Screen:** In the identified and implemented scenarios it is possible to by-

pass/disable the pattern unlock on Android via ADB Commands, but only if two special conditions are met: (1) the device is rooted and (2) the pin code known. This is because it is necessary to remove or update the system file containing the screen lock key, which is only possible by changing the access permissions to it, as described previously, and restarting the device, which requires the pin code.

10. **Overcoming the ADB Pairing:** If the device is rooted, it is possible to eliminate this security enhancement by removing the system file containing the ADB pairing key. However, to eliminate this issue permanently, a first access to the device must be given.
11. **Rooting the Device:** It is possible to root a device, simply by copying, installing, and running an application.

5 RELATED WORK

USB connection is just one of several ways to exfiltrate data from mobile devices. (Do et al., 2015) presents a catalogue of those methods and provides a good starting point to this subject. Here we focus on USB connections. Due to its relative importance and relation to USB exfiltration, we also discuss an available USB device that is able to disguise as a keyboard thus providing a medium to extract data from other devices.

As already stated, USB connections have been traditionally trusted mostly due to the physical proximity it implies but also due to the presumption that both devices belonged to the same owner. The work by (Wang and Stavrou, 2010) was possibility the first to demonstrate that this trust could be abused. In particular, it discusses attacks where a smartphone acts as Human Interface Device and sends keystrokes to control the victim host showing how to boot a smartphone to take over another phone using a specially crafted cable. The same article, also proposes defence mechanisms to counter those USB attacks. In (Wang et al., 2012) additional proposals are presented and discussed.

The work by Xu (Xu, 2014; Xu et al., 2015) found that the feature available in Android 4.2.2 cannot provide sufficient protection when the host machine connected to the Android device has been compromised. It presents an implementation demonstrating this vulnerability. Hence, it complements the work here presented.

(Pereira et al., 2014) presents a vulnerability to exploit the USB connection in a vendor customization that allows extending the reach of AT (ATtention) commands, where the system understands and allows these commands to be sent over USB. Those commands allows flashing a compromised boot partition without the user's consent thus gaining root access, enable ADB, and install a surveillance application that is impossible to uninstall without re-flashing the Android boot partition. In the case of Samsung, a large list of its family of smartphones has this vulnerability, where it is possible to communicate with the modem through the USB channel, without any previous configuration in the device, something that does not happen with ADB. Samsung expands the standard AT command set that comes with 3GPP and GSM standards, enhancing the interaction capabilities the computer software (Kies) has on the device. Kies for Windows uses the standard set and expanded AT owner set of commands to get the contacts, the contents of the SD card, and update the firmware of the device. Through this process (Pereira et al., 2014), it is possible to force USB debugging without prior authorization thus allowing the use of ADB to compromise the device.

Attacks based on USB devices constitute another area of research where the objective is related, but somehow symmetric to the work here presented: the objective is to attack a (typically larger) machine using a USB device, yet they can also be used to attack a mobile device. The USB Rubber Ducky is a device that resembles a normal USB flash drive (PEN). When connected to a computer, it is recognized as a keyboard, but it quickly introduces its malicious code. It is a commonly used and very useful tool by pentesters. An online video shows how to use the device to hack an Android phone (Hak5, 2012) .

Another type of attack consists in eavesdropping the communications between an USB device and a host. Neugschwandtner et. al describe and implement one of those attacks — a USB sniffing attack — where a USB device passively eavesdrops on all communications from the host to other devices, without being situated on the physical path between the host and the victim device. They also present UScramBle, a lightweight encryption solution to prevent that kind of attack (Neugschwandtner et al., 2016).

6 CONCLUSIONS

In this paper, three attack scenarios for USB connection in Android Systems were presented, together with a script as a proof of concept. The scenarios use three different devices and Android versions. It

has been demonstrated — for the three specific scenarios — that whenever USB debugging is available, the device is potentially compromised. In the tested versions of the OS, the only way to prevent attacks is to never make USB Debugging available. Once given this access door, the entire device can be compromised in seconds and in a form that is completely invisible to the victim.

In the specific model Samsung Galaxy Mini GT-S5570 with the version 2.2 Froyo there is still a peculiarity: the USB Debugging can be available, but in "not active" state, which allows a false illusion to the user to be protected when in fact it is not.

It has also been found that even the presence of protection software, such as the anti-virus, does not prevent the installation process of a potentially malicious application, at it only warns that a certain application may be harmful.

As future work new attack scenarios will be investigated, namely for newer versions of Android, overcoming the need for ADB pairing. It would also be interesting to have a social experience counting how many devices are be attacked by hour, day or even week, in places where it was offered the possibility to charge the devices' batteries for free. Different places in different environments would allow the identification of cases where more people are most likely to be victims.

ACKNOWLEDGEMENTS

This work was partially financed by Portuguese Agency FCT Fundação para a Ciência e Tecnologia, in the framework of project UID/EEA/00066/2013.

REFERENCES

Android Open Source project (2017). Android debug bridge. https://developer.android.com/studio/command-line/adb.html [Online; accessed 07-June-2017].

arsTechnica (2016). 10 million android phones infected by all-powerful auto-rooting apps. https://arstechnica.com/security/2016/07/virulent-auto-rooting-malware-takes-control-of-10-million-android-devices/ [Online; accessed 07-June-2017].

BusinessofApps (2015). 80% of android phone owners in china have rooted their device. http://www.businessofapps.com/80-android-phone-owners-china-rooted-device/ [Online; accessed 07-June-2017].

Do, Q., Martini, B., and Choo, K.-K. R. (2015). Exfiltrating data from android devices. *Comput. Secur.*, 48(C):74–91.

Google (2017). android. http://www.android.com [Online; accessed 07-June-2017].

Hacks, G. (2015). How to enable developer options & usb debugging. https://android.gadgethacks.com/how-to/android-basics-enable-developer-options-usb-debugging-0161948 [Online; accessed 07-June-2017].

Hak5 (2012). Android hacking with the USB rubber ducky. https://www.hak5.org/episodes/hak5-1216 [Online; accessed 07-June-2017].

Neugschwandtner, M., Beitler, A., and Kurmus, A. (2016). A transparent defense against USB eavesdropping attacks. In *Proceedings of the 9th European Workshop on System Security*, EuroSec '16, pages 6:1–6:6, New York, NY, USA. ACM.

Pereira, A., Correia, M., and Brandão, P. (2014). USB connection vulnerabilities on android smartphones: Default and vendors' customizations. In De Decker, B. and Zúquete, A., editors, *Communications and Multimedia Security: 15th IFIP TC 6/TC 11 International Conference, CMS 2014, Aveiro, Portugal, September 25-26, 2014. Proceedings*, pages 19–32. Springer Berlin Heidelberg, Berlin, Heidelberg.

Statista (2017). Number of smartphone users worldwide from 2014 to 2020 (in billions). https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [Online; accessed 07-June-2017].

TechAdvisor (2016). How to get developer options on android. http://www.pcadvisor.co.uk/how-to/google-android/34-useful-things-you-can-do-in-android-developer-options-new-3590299 [Online; accessed 07-June-2017].

Tischer, M., Durumeric, Z., Foster, S., Duan, S., Mori, A., Bursztein, E., and Bailey, M. (2016). Users really do plug in usb drives they find. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 306–319.

Wang, Z., Johnson, R., Murmuria, R., and Stavrou, A. (2012). Exposing security risks for commercial mobile devices. In *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security: Computer Network Security*, MMM-ACNS'12, pages 3–21, Berlin, Heidelberg. Springer-Verlag.

Wang, Z. and Stavrou, A. (2010). Exploiting smart-phone usb connectivity for fun and profit. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 357–366, New York, NY, USA. ACM.

Xu, M. (2014). Security enhancement of secure USB debugging in Android system. Master's thesis, University of Toledo, USA. in http://utdr.utoledo.edu/theses-dissertations.

Xu, M., Sun, W., and Alam, M. (2015). Security enhancement of secure USB debugging in android system. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 134–139.