IOWA STATE UNIVERSITY
Digital Repository

Creative Components

Iowa State University Capstones, Theses and Dissertations

Spring 2019

An Investigative Study on Android Verified Boot Process

Brett Weiss

Follow this and additional works at: https://lib.dr.iastate.edu/creativecomponents

 Part of the Computer and Systems Architecture Commons

Recommended Citation

Weiss, Brett, "An Investigative Study on Android Verified Boot Process" (2019). *Creative Components*. 278.
https://lib.dr.iastate.edu/creativecomponents/278

This Creative Component is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Creative Components by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

An Investigative Study on Android Verified Boot Process

by

Brett Weiss

A Creative Component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Yong Guan, Major Professor

Iowa State University

Ames, Iowa

2019

Copyright © Brett Weiss, 2019. All rights reserved.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

NOMENCLATURE

ADB	Android Debug Bridge
Odin	Flashing tool for Samsung devices
OEM	Original Equipment Manufacturer
OTA	Over-the-Air
OTG	On-the-go
SDK	Software Development Kit

ACKNOWLEDGMENTS

I would like to thank my major professor, Yong Guan, and PHD students, Cheng Chao-Chun and Chen Shi, for their guidance and support throughout the course of this research. Their assistance and expertise was extremely useful, since I did not have much experience with Android devices or mobile systems in general.

ABSTRACT

Android Verified Boot is a security feature of Android devices. It is designed to create a chain of trust and verify all executed code comes from a trusted source. Analyzing how Verified Boot works and its strengths and weaknesses as a security feature is useful to understanding the security of Android devices in general. Analysis was conducted on two Android devices, a Samsung Galaxy S5 and a OnePlus 5T.

Research on the OnePlus 5T device was conducted using Android Debug Bridge (ADB) and Fastboot, both part of the Android SDK. Research on the Samsung Galaxy S5 device was conducted using Odin. The devices were tested to see if and how modifications to the systems were handled. Attempts were made to change Android versions, change boot images, and modify the files in the system. Observing how the Android devices responded was useful in understanding how Verified Boot works and its limitations.

CHAPTER 1. INTRODUCTION TO ANDROID

Android is an operating system for mobile devices developed by Google. It is based on the Linux kernel, and is designed for touchscreen smartphones and tablets. The first commercial Android device was released in 2008.

Hardware

ARM is the main hardware platform for Android. ARM is the most widely used instruction set architecture. Android devices typically have a touchscreen. Some smartphones also contain GPS, accelerometers, gyroscopes, barometers, or barometers.

Kernel

A kernel is the core of a computing system, with control over the entire system. The Android kernel is based on the Linux kernel's long-term support branches. The requirements for Android kernels can be found on Androids website. https://source.android.com/devices/architecture/kernel/modular-kernels#core-kernel-requirements

Flash storage on Android devices is split into multiple partitions, including /system for the operating system and /data for user data and applications. Contrary to desktop linux, Android devices do not provide root access to the OS. Root access can be obtained by exploiting security flaws or installing custom images, in a process called rooting. This is similar to the iOS concept of jailbreaking (escalating access to an administrative-level).

Update Schedule

Major incremental upgrades to Android happen on a yearly basis [1].

Table 1.1 *Android Version History*

Code Name	Version Number	Linux Kernel Version	Initial Release Date	API level
(No codename)	1.0	?	September 23, 2008	1
Petit Four	1.1	2.6	February 9, 2009	2
Cupcake	1.5	2.6.27	April 27, 2009	3
Donut	1.6	2.6.29	September 15, 2009	4
Eclair	2.0 – 2.1	2.6.29	October 26, 2009	5 – 7
Froyo	2.2 – 2.2.3	2.6.32	May 20, 2010	8
Gingerbread	2.3 – 2.3.7	2.6.35	December 6, 2010	9 – 10
Honeycomb	3.0 – 3.2.6	2.6.36	February 22, 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	3.0.1	October 18, 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	3.0.31 to 3.4.39	July 9, 2012	16 – 18
KitKat	4.4 – 4.4.4	3.10	October 31, 2013	19 – 20
Lollipop	5.0 – 5.1.1	3.16	November 12, 2014	21 – 22
Marshmallow	6.0 – 6.0.1	3.18	October 5, 2015	23
Nougat	7.0 – 7.1.2	4.4	August 22, 2016	24 – 25
Oreo	8.0 – 8.1	4.10	August 21, 2017	26 – 27
Pie	**9.0**	4.4.107, 4.9.84, and 4.14.42	August 6, 2018	28
Android Q	10.0			29

Android updates can take a while to reach various devices. Due to variations in hardware, each upgrade must be configured to that device. Manufacturers may not even provide the upgrade to their older devices. Wireless carriers can also cause delays by customizing Android and testing before releasing the update to users.

CHAPTER 2. ANDROID BOOT PROCESS AND BOOTLOADER

The Android boot process contains quite a few steps. Understanding how Android starts is important to comprehend the security posture of an Android device.

Boot ROM

When the power button on an Android device is pushed, the code in the Boot ROM section is executed. This code is stored in a hardcoded location. This code is responsible for loading the bootloader into RAM and executing it. This can be compared to BIOS in a desktop computer.

Bootloader

The bootloader is separate from the Linux kernel. It sets up initial memories and loads the kernel to RAM. Different device manufacturers will often use proprietary bootloaders. These proprietary bootloaders can have custom security checks.

The bootloader sets up network, low-level memory management, and security options. Control is then passed to the Linux kernel.

Kernel and Init Process

The kernel performs functions similar to Linux kernels. It sets up the cache and protected memory. It also schedules load drivers. Next, the init process starts.

The init process mounts partitions like /sys and /dev. A file called init.rc is used to start system services, the file system, and any other parameters that need to be set up.

Zygote and System Service

Finally, a VM process called Zygote runs and initializes core library classes. Zygote forks a new process to start other Android services like Bluetooth, battery, location, sensor and other services.

CHAPTER 3. ANDROID APPLICATIONS

Android applications are written in Kotlin, Java, or C++. The Android SDK compiles application code into an APK (Android package) which is a type of archive file with a .apk file extension type. Android devices use the APK to install the application.

Security Principles

Android applications are separated into their own security sandboxes. Each application is assigned a different user in the Android system. The user id assigned to each app is only known to the system. The system sets permissions for all of the app's files so only the user id assigned to the app can access its files. By default, each app will run in its own Linux process. Each process has its own VM, so its code runs in isolation.

The Android system is designed with the security principle of least privilege in mind. This means each application by default will only have access to the components it needs to do its work. There are some ways for apps to share data with other apps and apps to access system services.

Application Permissions

Android apps must request permission to access sensitive user data, such as contacts or SMS. Certain system features, like camera, and internet, also require permissions. By design, no application has permission to perform any actions that impact other apps, the operating system, or the user [2].

There are three different levels of permissions for Android applications. Normal permissions cover areas where apps need to access information or resources out of the app's sandbox, but the risk of impacting user's privacy or other apps is very low. Signature permissions are granted when an app installs, but only when the app that uses the permission

is signed by the same certificate as the app that defines the permission. Dangerous permissions cover areas where user's private information or other apps could be affected. If an app needs dangerous permissions, the user must specifically grant permission through a prompt at app runtime.

Table 3.1 *Permission Requirements and Examples*

Permissions	Normal	Signature	Dangerous
Requirements	autogranted at install	certificate must match	prompt for permission at runtime
Examples of that Permission Type	• Bluetooth • Internet • Set Alarm • Vibrate	• Bind Text Service • Clear App Cache • Write Settings	• Read Call Log • Camera • Access Fine Location • Read Phone State • Send SMS • Read External Storage

CHAPTER 4. ANDROID SECURITY FEATURES

Application Sandbox

Android takes advantage of its Linux kernel to separate application processes. [3] Each application has a unique id. In Android 5.0, SELinux provided mandatory access control (MAC) separation between the system and apps. However, all third-party apps ran within the same SELinux context so inter-app isolation was primarily enforced by UID DAC.

In Android 6.0, the SELinux sandbox was extended to isolate apps across the per-physical-user boundary. Android also set safer defaults for application data.

In Android 8, apps were set to run with a seccomp-bpf filter. This filter limited the syscalls that apps could use, thus improving the security of the app/kernel boundary.

In Android 9 all non-privileged apps with targetSdkVersion >= 28 must run in individual SELinux sandboxes, providing MAC on a per-app basis. This protection prevents overriding safe defaults and prevents apps from making their data world accessible [4]. World accessible data can be accessed by any app, so it is dangerous to store app data in this form.

Security-Enhanced Linux

Security-Enhanced Linux (SELinux) enforces mandatory access control over all processes in an Android system. These restrictions even apply to devices running as root or with superuser privileges. SELinux operates on a deny by default basis.

In Android 5.0 and later SELinux is fully enforced. No processes other than init can run in the init domain. Android 6.0 reduced permissiveness to better isolate users and added IOCTL filtering. A majority of input and output control calls were not needed for

functionality [5]. IOCTL filtering reduces the number of unneeded input and output control calls.

Android 7.0 locked down the application sandbox and broke up the mediaserver stack into smaller processes. Android 8.0 updated SELinux to work with Treble. This separates lower level vendor code from the Android system framework. This update also allowed device manufacturers to update their parts of the policy and build their images.

Filesystem Encryption

Android 3.0 and later provides full filesystem encryption, so all user data can be encrypted in the kernel.

Android 5.0 and later supports full-disk encryption. Full-disk encryption uses a single key—protected with the device password—to protect the whole of a userdata partition. The user must provide their credentials upon boot before any part of the disk is accessible.

Android 7.0 and later supports file-based encryption. File-based encryption allows different files to be encrypted with different keys that can be unlocked independently. This allows a user to separate encryption for different parts of the device.

CHAPTER 5. VERIFIED BOOT

Android Verified Boot is a security feature of the newer Android versions. Its main purpose is to ensure that all executed code comes from a trusted source, rather than from an attacker or device corruption. It works using a chain of trust model, starting from the bootloader, and moving to the boot partition and other verified partitions like the system, vendor, and OEM partitions. When the device is started, each stage verifies the integrity and authenticity of the next stage before proceeding to it.

Device State

To understand Verified Boot, the concept of device state must be defined. A device's state indicates whether software can be flashed to the device. Locked devices follow steps to verify the boot. Unlocked devices avoid the checks. The bootloader can be locked or unlocked using fastboot.

Root of trust is the cryptographic key used to sign the version of Android on the device. Devices with multiple bootloaders may have multiple keys to verify. Users can set their own root of trust for some devices. This can allow custom versions of Android to be used without removing the security benefits of Verified Boot.

Verify Process

Verified Boot cryptographically verifies executable code before it is run. This includes the kernel, device tree, and the system and vendor partitions.

Small partitions, like boot (kernel) and dtbo (device tree) are checked using hashing. The entire partition is loaded into memory, then the hash is calculated and compared against an expected value. If the hash doesn't match, Android will not load.

Larger partitions that can't fit into memory (for example, file systems) need to use a hash tree where verification continuously happens as data is loaded into memory. The root hash is compared to an expected root hash value. If the values do not match at any point, Android enters an error state.

Expected hashes are stored at the beginning or end of each verified partition, or in a dedicated partition, depending on implementation. These hashes must be signed by the root of trust to ensure integrity.

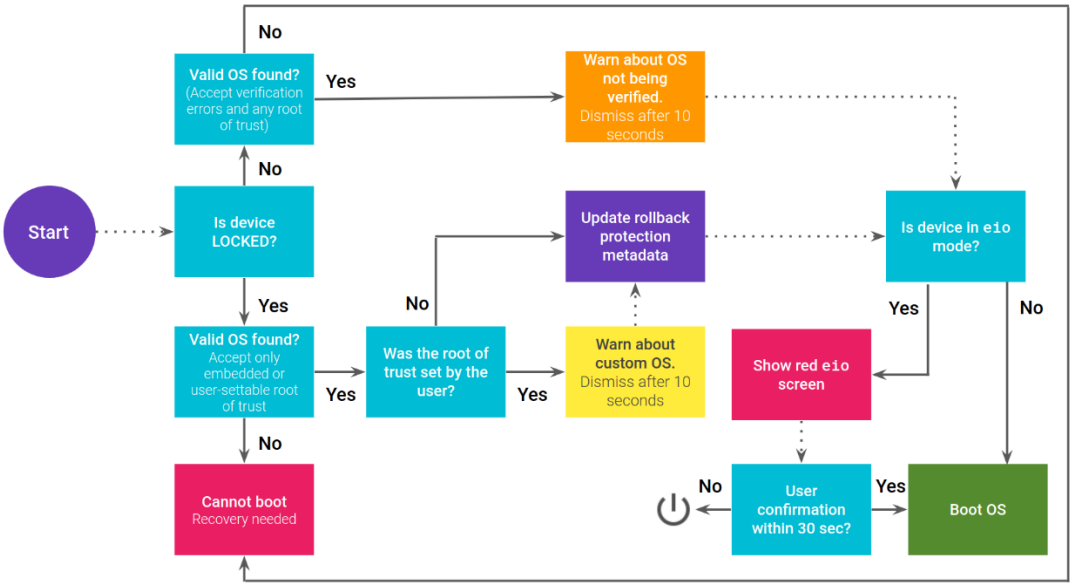

Figure 5.1 *Boot flow for Android devices* [6]

Rollback Protection

Verified Boot also checks for the correct version of android with rollback protection. Rollback protection works by using tamper-evident storage to record the current Android version. When the system is booted, the version is checked to make sure it is not lower than the recorded version. If it is, the device will refuse to boot. This prevents attackers from installing an older version of Android with vulnerabilities, then exploiting those vulnerabilities to compromise the system.

Support for Verified Boot was added in Android version 4.4 [7]. Android 7.0 began strictly enforcing Verified Boot so compromised devices would not boot.

CHAPTER 6. ROOTING

Rooting refers to the process of gaining root access to the Linux kernel. Root users have the ability to access all applications and application data. Many Android devices can be rooted by unlocking the bootloader and installing an alternate operating system. Because an attacker could use this procedure on a phone they have taken, unlocking the bootloader will erase user data as part of the unlock step. Root access that is gained by exploiting a kernel bug will avoid this problem.

Using Recovery Image

To root an Android device using a recovery image, the device must be put into Developer mode. USB debugging needs to be enabled and the bootloader needs to be unlocked. A custom recovery image needs to be flashed onto the device. One example of a custom recovery image is TeamWin Recovery Project (TWRP).

TWRP allows the following features:

- Backups of partitions in TAR or raw Image format
- Restore backups from internal storage, external SD storage or OTG devices
- Custom Firmware installation
- Partition wiping
- File deletion
- Terminal access
- ADB Root Shell
- Theme Support
- Possible decryption support depending on device

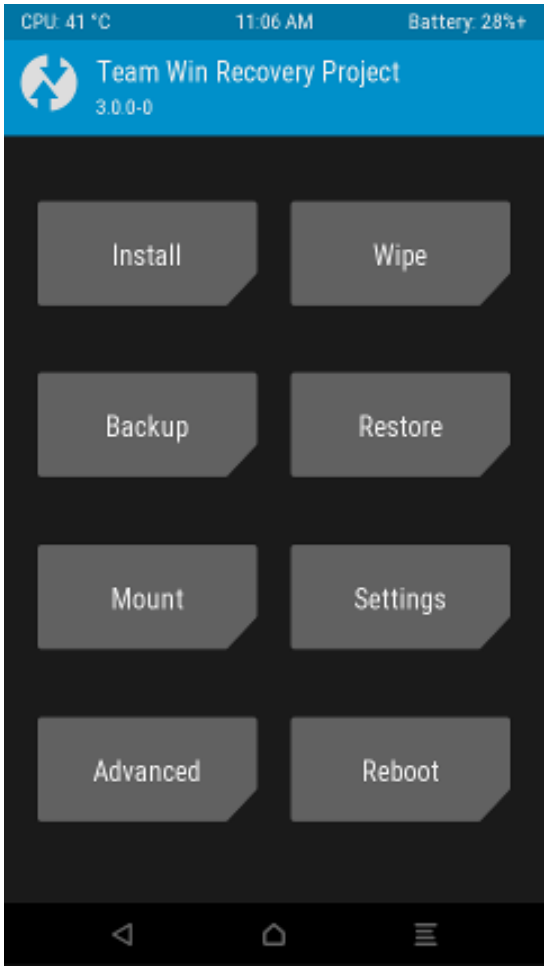

Figure 6.1 *TWRP options screen* [8]

Rooting Android App

Some applications have been developed to root Android devices. These applications are not available in the Google Play store and will require a manual download of the APK and changing settings to allow install from unknown sources. The apps typically exploit some kind of vulnerability in the Android kernel to gain root access. KingoRoot is an example of an app that roots a device.

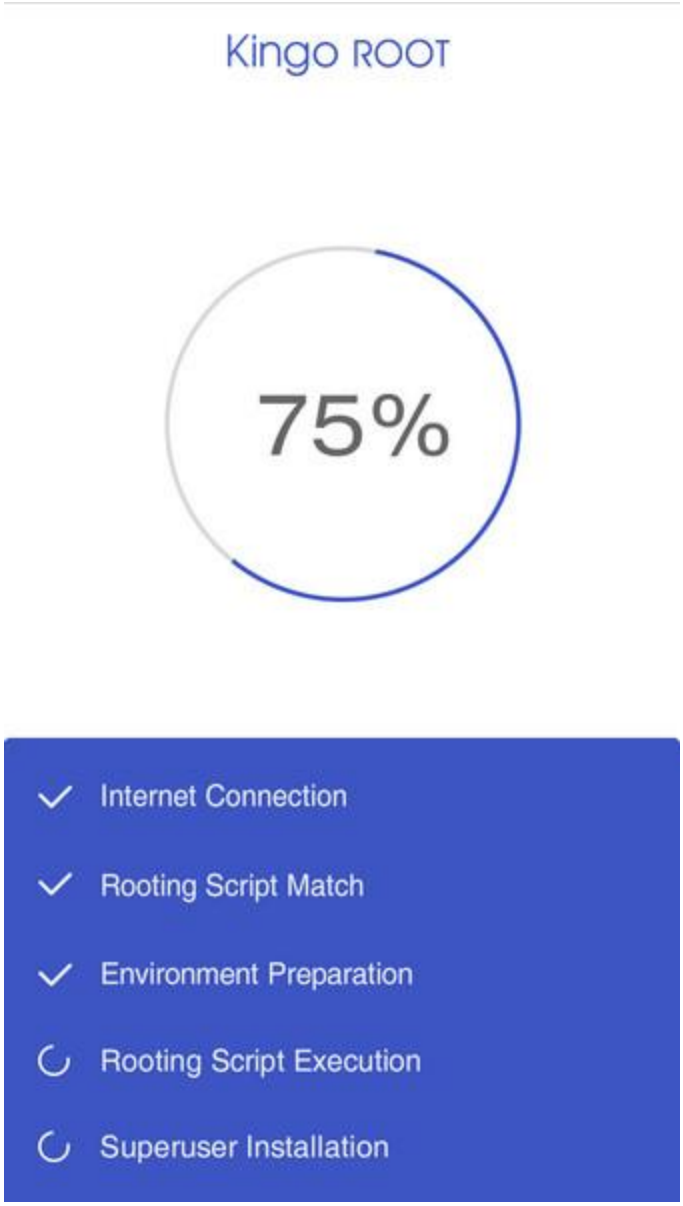

Figure 6.2 *KingoRoot app example screen* [9]

CHAPTER 7. PRIOR RESEARCH

Android Root Detection

Some research has already been conducted on Android Verified Boot. Research into the different rooting methods on Android had interesting results [10]. Pre-boot rooting of an Android device is commonly used for non-malicious purposes. A user may want to customize their Android build or install third party apps. Unlocking the bootloader is the main way to accomplish this task. The researchers suggested that adding a way for mobile apps to query the state of the bootloader would be useful for enhancing security.

Some Android apps are designed to be able to detect if a device has been rooted. These apps were separated into two categories. The first is applications that improve usability by informing users that the app will need root access to perform an action. The second reason is security. Some applications have sensitive or valuable user data that would be vulnerable on a rooted device. Apps needing detection for security reasons need a higher quality of usability detection. Most apps used a couple of techniques for detection, although 2/3 of the apps used 2 or 3 detection methods.

The research found that the only effective way to detect rooting would be using integrity-protected kernels or external trusted execution environments.

Using Firmware Update Protocols

Another paper tried a different approach to the rooting problem. Since most forensic tools rely on rooting or USB debugging, this team attempted to use the firmware update protocol in the bootloader to acquire the physical memory of an Android device [11]. By analyzing the firmware update protocols used by smartphone manufacturers, the commands

for direct access to flash memory were found. This allows access to the flash memory without needing to bypass a lock screen or engage USB debugging mode.

CHAPTER 8. METHODS AND TECHNIQUES

OnePlus 5T

The OnePlus 5T manufactured by OnePlus was the first Android device I analyzed. The OnePlus 5T was released on November 21, 2017. The 5T can be interacted with using the standard Fastboot and ADB tools provided by Android.

ADB

ADB stands for Android Debug Bridge, and it is a command line tool that can be used to interact with smartphones, tablets, and other Android systems [12]. ADB provides functionality to an Android device that would not be needed for everyday use. Using ADB, apps can be installed from outside of the Play Store, as well as debugged. ADB also allows access to hidden features and allows a user to bring up a Unix shell. Due to this extra level of access, Develper Options need to be unlocked and USB Debugging needs to be enabled on a device prior to using ADB. Some common ADB commands are listed below.

Table 8.1 *Common ADB Commands*

ADB Command	Description
adb devices	lists the current connected devices
adb install *APKLocation*	installs an application from *APKLocation* to the device
adb reboot	reboots the device in normal mode
adb reboot recovery	reboots the device in recovery mode
adb reboot bootloader	rebaits the device into bootloader mode
adb push *Source Destination*	sends a file from *Source* to the given *Destination*

Fastboot

Fastboot is a tool that writes data directly to a devices flash memory. It is used to flash images like recoveries, bootloaders, and kernels to an Android device. It can also be used to flash system updates. Common Fastboot commands are listed below

Table 8.2 *Common Fastboot Commands*

Fastboot Command	Description
fastboot devices	lists the current connected devices
fastboot oem unlock	unlocks the bootloader
fastboot oem lock	locks the bootloader
fastboot erase boot	wipes a device's boot partition
fastboot flash boot *kernel-file.img*	flashes the kernel

Samsung Galaxy S5

Samsung has a tool for flashing called Samsung Odin. Odin was originally developed for internal use at Samsung. It can be used to flash stock or custom firmware images.

Odin

Odin works using firmware in the form of a tar md5 file. These files can be difficult to find for older models of Samsung phones, especially when looking for an older version of Android as well. Samsung doesn't maintain a hosting of firmware files for their older phones, although the USB drivers can be downloaded from their site.

CHAPTER 9. TESTING

Samsung Galaxy S5

Setup

Odin was installed onto a Windows computer, and the image files for flashing various versions of Android. Samsung has firmware packages for different versions of Android. These files are no longer hosted by Samsung, but they are archived on various web sites for access. These files are tar archives verified with MD5 checksums. To use Odin, the Samsung phone must be set to download mode. On the Samsung Galaxy S5, this is accomplished by pressing and holding the volume down, power, and home buttons simultaneously.

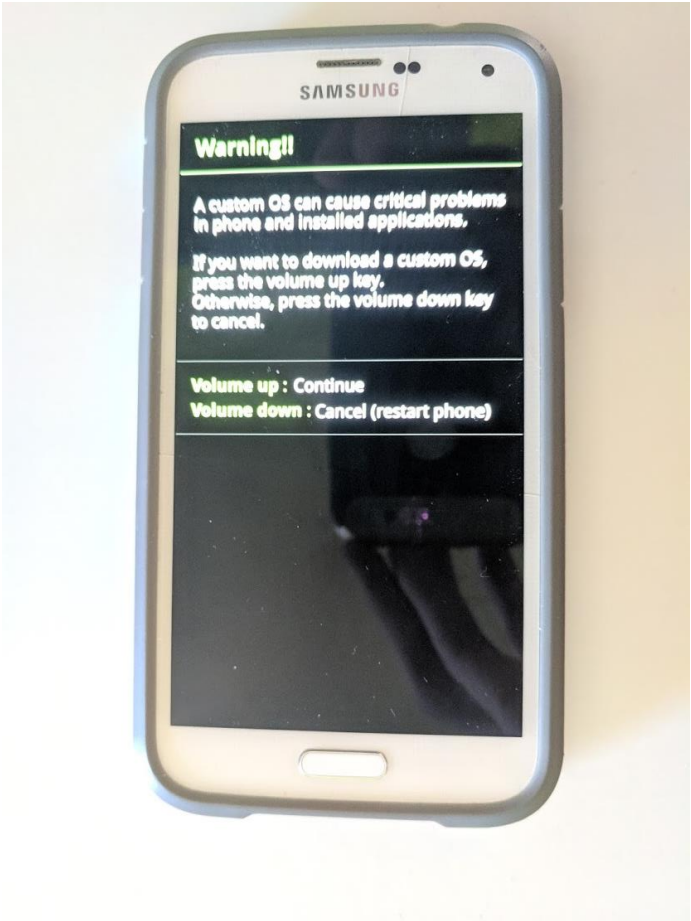

Figure 9.1 *A Samsung Galaxy S5 in download mode*Once the device is in download mode, it can be connected to a computer running Odin. The device will appear in Odin as a connected device. The tar archive is set as an input, then the flash can be attempted.

Reverting to an older version

Attempting to revert to an older version of Android from Android 5.0 on Verizon devices will cause problems. The model SM-G900V devices use special bootloaders made by Verizon. The OE1 and OG5 bootloaders prevents downgrading [13], as seen in Figure 9.2. Verizon updates their bootloaders using OTA (over-the-air) updates, so a device will automatically receive the updated bootloader.

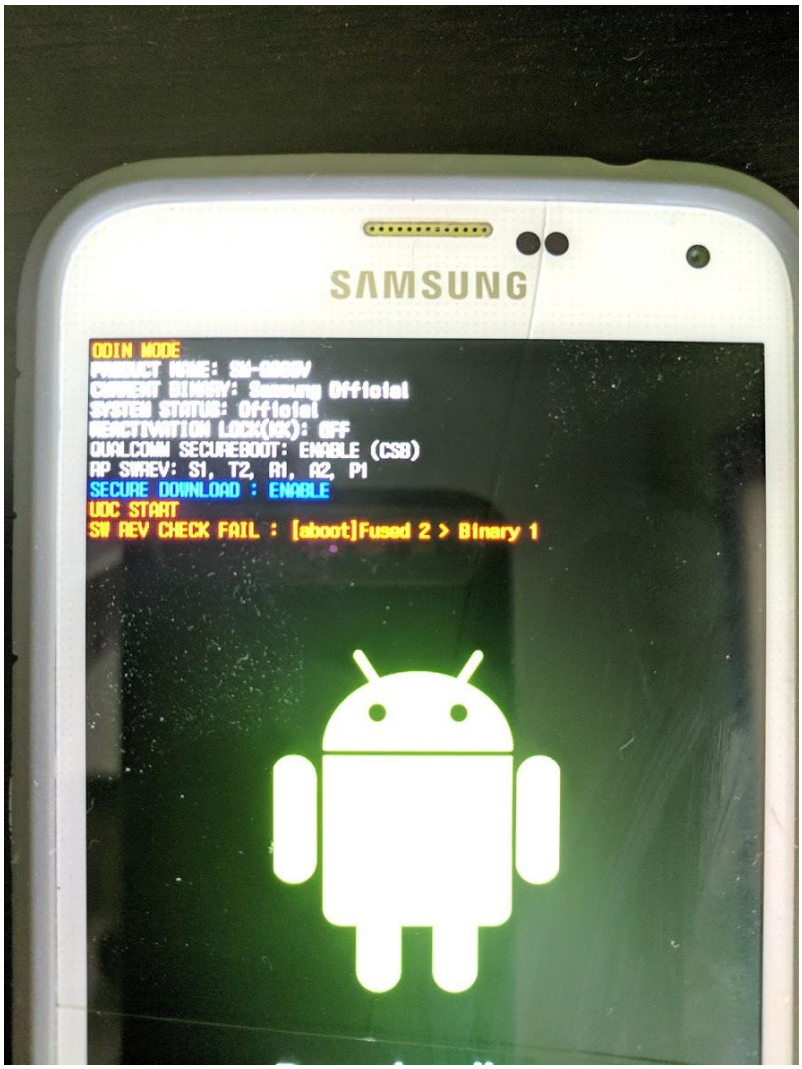

Figure 9.2 *Error message displayed when trying to downgrade firmware.*Odin will also display an error message when the process fails. In this case, the tar file used is for Android 4.4 Kitkat.

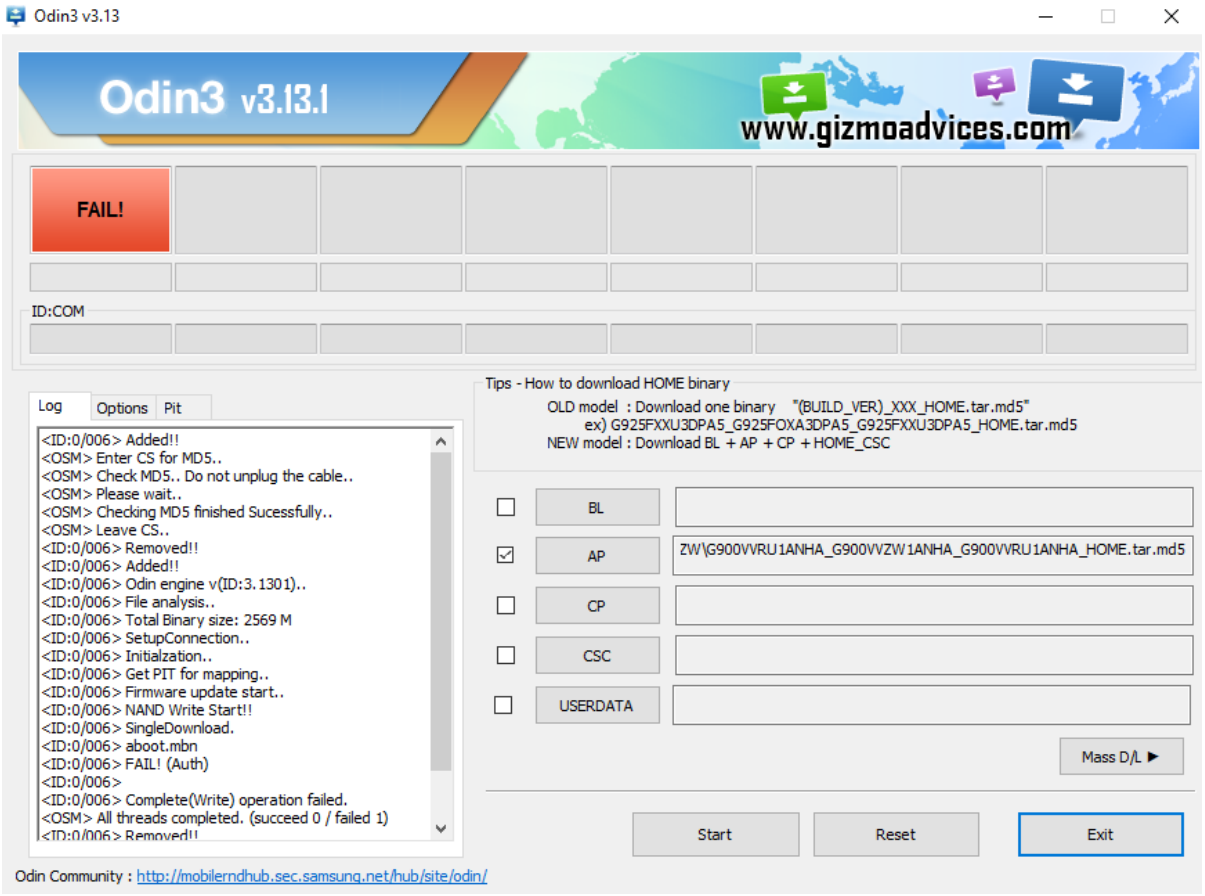

Figure 9.3 *Odin Error message when attempting to downgrade firmware.*

OnePlus 5T

The OnePlus 5T phone was released in November 2017. ADB and Fastboot can be used to modify the device. The device does need USB Debugging mode enabled and the OEM unlocked in order to use Fastboot. After the device was set up, it was started in Bootloader mode.

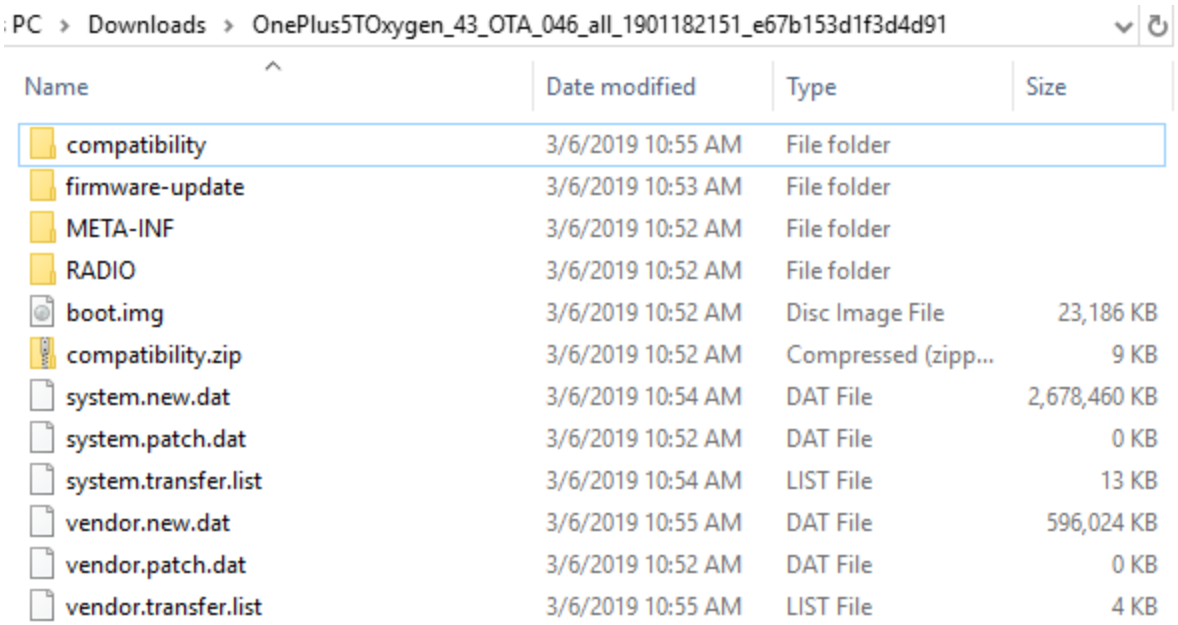

Figure 9.4 *OnePlus 5T firmware package.*

A new boot.img file was flashed into the system. However, this resulted in the bootloader displaying the “Your device is corrupt. It can't be trusted and may not work” error message.

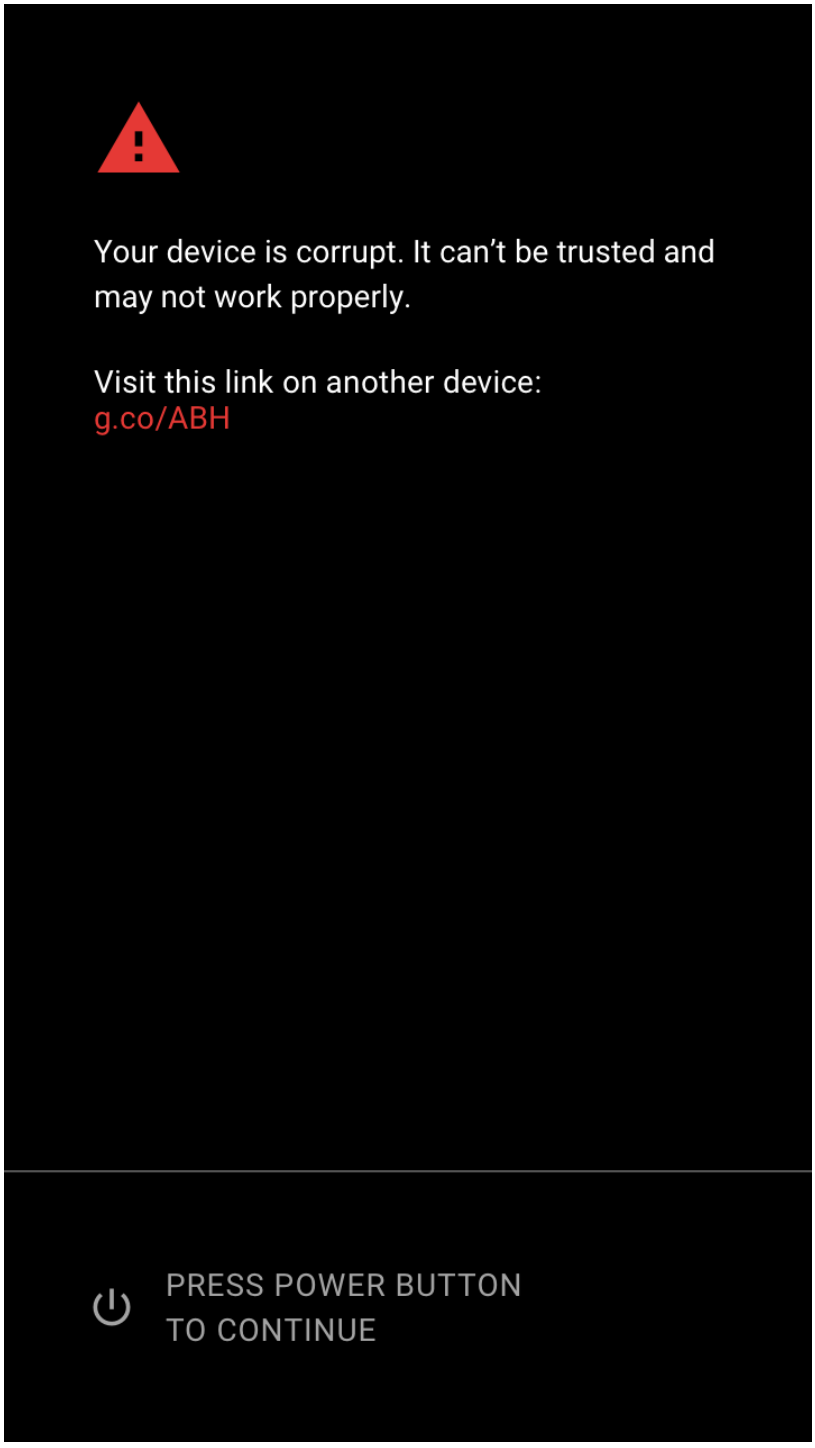

Figure 9.5 *Error message when attempting to boot*

This message indicates the boot was blocked by Android Verified Boot [6]. Since the OnePlus 5T phone was using a version of Android that enforces Verified Boot, the phone would not start.

CHAPTER 10. SUMMARY AND FUTURE WORK

There are many different factors that contribute to the security of an Android device. The Android version (Kitkat, Lollipop, Marshmallow, etc) can influence the security. The manufacturer of the phone can also make a large difference. Different manufacturers can choose different levels of accessibility and ease-of-use for their devices. They can also provide tools to make changes to the firmware of a user's phone. Some manufacturers will even provide different bootloaders, for example Verizon's bootloaders for the Samsung Galaxy S5 that prevent reverting to previous Android versions.

More research is needed into Android security features. Each new Android version and even each new Android device presents a new security profile, so there is a wide variety of profiles to study. Android security profiles continue to diffuse with each passing years as the number of profiles increases.

In the future, more work could be done testing Android devices from other manufacturers. The Google Pixel line of phones would be interesting to test. Google provides full OTA images for Nexus and Pixel devices on their web site.

Another interesting concept is comparing the behavior of different devices with the same Android version. By controlling the Android version and keeping it a constant, the differences between devices and manufacturers could be more easily shown.

References

[1] Wikipedia, "Android Version History".

[2] Android Developers, "Permissions Overview".

[3] Android Open Source Project, "System and Kernel Security," Google.

[4] Android Open Source Project, "Application Sandbox".

[5] T. Spring, "How Google Shrank the Android Attack Surface," Threatpost, 2017.

[6] Android Open Source Project, "Boot Flow".

[7] Android Open Source Project, "Verified Boot," Google.

[8] TeamWin, "What is TWRP?".

[9] Kingo Root, "How to Root Android without Computer".

[10] S.-T. Sun, A. Cuadros and K. Beznosov., "Android rooting: Methods, detection, and evasion," *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. ,* pp. 3-14, 2015.

[11] S. J. Yang, J. H. Choi, K. B. Kim and T. Chang, "New acquisition method based on firmware update protocols for Android smartphones," *Digital Investigation,* vol. 14, pp. S68-S76, 2015.

[12] XDA Developers, "What is ADB? How to Install ADB, Common Uses, Advanced Tutorials".

[13] muniz_ri, "Update to G900V_PF4 - 6.0.1," XDA Developers, 2015.