# 1

# ANDROID OS INTERNALS: UNDERSTANDING HOW YOUR DEVICE STARTS



IN THIS CHAPTER:

- The penguin down below: the Linux kernel
- Bootstrapping: How your device starts
- An introduction to custom bootloader and custom recovery processes

TO FULLY UNDERSTAND the process of rooting your device, gaining the control and power you need to truly customize it, you need to understand a little about how the Android operating system works—how the device goes from being powered off to a fully functioning state. It is in this process that developers usually exploit weaknesses to gain full access to the device. Usually some step in the boot process allows a developer to insert a bit of code or a script, and thus access functionality not intended by the Original Equipment Manufacturer (OEM).

## Linux Development and Open Source

Linux began in 1991 with Linus Torvalds working to make a completely free and open source operating system that could be used by hobbyists, academia and hackers. His operating system has grown to be one of the most powerful and flexible in the world today. From a handful of unknown geeks, the developer base has matured to include thousands of contributors every year. Some of the finest names in computer science and programming work on the development not only of Linux but also of Android.

Linux remains completely free and completely open source. This allows companies and individuals to have access to the power of computing devices without the complex legal and copyright concerns that come with closed source software.

# THE PENGUIN DOWN BELOW

Android is an operating system built on the Linux kernel. Thanks to Google and the Open Handset Alliance, Linux and its penguin mascot have found a home on Android devices. Android is essentially a highly customized distribution of Linux with various tweaks oriented towards mobile devices.

If you are familiar with the Linux operating system then you are going to feel quite at home with many aspects of the Android operating system. If you are comfortable with any other command-line operating system, such as DOS or the Windows command line, many of your skills there will be useful as well.

Android is, at its core, an implementation of the Linux operating system. Many of the commands you will be using in hacking an Android device are Linux commands. However, you do not need to be a programmer to become an Android hobbyist or enthusiast. Using the skills taught in this book, you can become adept at exploring and altering your Android device.

The differences between your Android device and a Linux desktop computer are many. The most striking difference is the way in which your device bootstraps (starts) when you power it on. It is in this start-up process that the hackers and elite developers find the vulnerabilities to exploit. Because Linux has a long history of being the go-to operating system of developers, hobbyists and hackers, there are many programmers and professional experts working

on tools that help you with the root process. Most of the "heavy lifting" is done long before the average Android hacker gets access to root on his or her device.

Although you do not need to be a Linux nerd to root and customize your Android device, being familiar with the Linux command line, and command lines in general, will help you feel more comfortable. For an excellent reference to the Linux command line, check out *Linux Command Line and Shell Scripting Bible,* 2nd Edition by Richard Blum (Wiley, 2011).

# HOW YOUR ANDROID DEVICE STARTS

The Android operating system has a complex and multistage start-up routine. Manufacturers lock the start-up process to protect revenue and maintain control of the device you purchase. The nature of the Android start-up process allows developers and hackers to replace parts of it to achieve full control of an Android device.

## BOOTSTRAPPING

Bootstrapping (or booting) is a term that describes what a computing device does when turned on. It "pulls itself up by its bootstraps." When you power on an Android device, a tiny piece of code on a memory chip initializes the memory and CPU. Usually the bootstrap code is referred to as the boot-loader. The bootloader is different from device to device, although all bootloaders do the same things: they check for hardware features and load the first part of the operating system into the device's memory.

The encrypted bootloader is the beginning of all things Android, effectively locking out the user from customizing the firmware and software. Locking the bootloader is the rough equivalent to a computer manufacturer forcing you to use a particular version of Windows, along with a theme of their choosing. The bootloader is the primary point of contention between owners of mobile devices and the original equipment manufacturer (OEM). Many, if not most, OEMs specifically do not want you to have access to that boot-loader code. The reasons that OEMs do not want users to have access to this code are varied but fall into the following categories:

- **The cost of honoring warranties:** Altering the bootloader code can permanently disable the device. This is problematic for device manufacturers because broken devices are returned to them under warranty. It is difficult to determine if a device is broken because the user did something silly to it or if it is, in fact, defective. This means that the manufacturer may have to replace a device that became defective through no fault of the manufacturer. Replacing defective devices costs money and those costs may be passed on to the consumer.
- **The need to protect carrier agreements:** Carriers are paid to pre-install applications from third parties on devices. Many organizations, from car rental companies to streaming video startups, have a mobile application. To get exposure for their products, they pay carriers to include those applications on your device; to ensure that exposure, the carrier blocks the user's ability to remove the application. After all, it simply wouldn't do to have Blockbuster pay hundreds of thousands of dollars to have their application on your device only to have you remove it to make room for Angry Birds three minutes after you walk out of the store. Locking the bootloader allows carriers and OEMs to declare some applications as "system" applications. This removes them from typical management tasks, such as deletion or moving them to an SD card.
- **Planned obsolescence:** Devices with a very long life are bad for OEMs. The development and release cycle of new mobile devices has become incredibly fast, outpacing even old standards in technology. When a device is released, the device that will obsolete it is often already in production. Android operating system updates have new features and stability that users desire. Because OEMs depend on selling new features and the latest Android operating system, they need consumers to want the newest devices. Allowing consumers to update the operating system and software themselves effectively reduces the need to purchase the latest device from the OEM or carrier.

  In essence, planned obsolescence from the carriers and OEMs is designed to make the consumer spend more money to get the latest Android updates. If you can hack those updates into the perfectly good device you purchased six months earlier, the OEMs lose money.

When you power on an Android device, the bootloader is the first program code that runs. Bootloading is typically a two-part process, utilizing a primary and a secondary bootloader.

On most Android devices, the primary bootloader cannot be replaced. This is because the primary bootloader is hardcoded into an application-specific integrated circuit (ASIC) in the device. These hardcoded instructions load the secondary bootloader into memory and tell it where the memory, CPU and operating system are located and how they can be accessed.

## Taking Responsibility for Your Hacks

It is important to note that if you choose to hack your device, you take responsibility for replacing it. It is unfair and unethical to do something silly to your device that disables it and then expect the carrier or OEM to replace it. Good hackers go into their hacks knowing the possible outcomes and willing to take responsibility for their own failures. When it comes to OEM and carrier ill-will towards hackers, ensure you are part of the solution not part of the problem. Never try to return a bricked or disabled device for replacement. Learn how to fix it or take responsibility and replace it.

## ADDING A CUSTOM BOOTLOADER

A custom bootloader is a secondary bootloader that allows you to gain access to the file system with more control than you can with an OEM bootloader. Custom bootloaders open up the possibilities of replacing the original operating system files with customizations as varied as a new user interface or a supercharged kernel. Despite the manufacturer's objections, the hacker's goal is to interrupt the standard bootloading process and use a custom bootloader that enables hacking of the device.

## UNDERSTANDING THE BOOTLOADER PROCESS

Your Android device follows certain steps when booting up. The following steps and Figure 1-1 are simplified and made generic to apply to most Android devices.

1. Special code in the boot read-only memory (ROM) locates the first-stage bootloader and loads it into memory. The boot ROM is an ASIC that has its code permanently programmed.
2. The first-stage bootloader loads the second-stage bootloader after initializing some memory and getting the hardware ready.

   The bootloader checks to see if the security flag is on (`S-ON`). If it is on, then the bootloader will load only signed (official) kernels. If the security flag is off (`S-OFF`), then the bootloader no longer checks for signatures. Setting `S-OFF` also releases other security lock downs, making the entire file system writable and enabling other goodies, such as allowing you to install a custom recovery process on the device.

   This is the step in which you want your custom bootloader to be loaded. The holy grail of hacking a manufacturer's handset is to load a custom bootloader so that a custom kernel can be loaded.

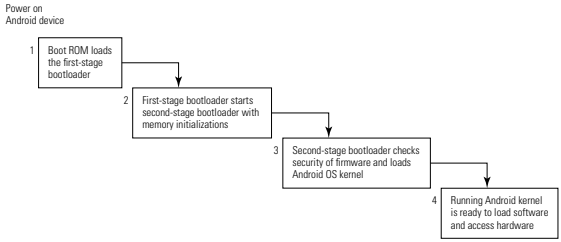


Figure 1-1: The Android boot process

Fastboot (see Chapter 3) is a protocol that allows low-level commands to be sent to a device to do such things as write files (such as custom bootloaders, recoveries and ROMs) to the operating system. Most manufacturers, therefore, disable the Fastboot protocol at the factory. Because the second-stage bootloader is the step in the boot process where the Fastboot protocol is enabled or disabled, this part of the code is frequently encrypted or otherwise locked down by OEMs. Some devices, such as Nexus devices and the Xoom, can be unlocked, allowing the Fastboot protocol to be enabled.

3. The bootloader loads a Linux kernel and customizations into memory.

   At this point, the bootloader hands off control of the hardware to the Linux kernel. The Linux kernel and any software or firmware customizations are usually all packaged together. On some devices, they are called a ROM. The name ROM is a slight misnomer because NAND storage is not truly read-only. Other devices require custom images (in IMG format) to be written to memory; still others have the kernel package written from an RUU file. However the kernel package is placed on the device, the bootloader must know where it is located and how to hand over the reins to it.

4. The last step is the initialization (INIT) process. The INIT process is the mother of all other processes that run on your device. It initializes all of the processes necessary for basic hardware access and device functionality. It also starts up the Dalvik virtual machine processes where most applications are executed.

Through this whole start-up process, the important thing for you to understand is that most of the hoops you have to jump through when rooting your Android are to achieve one or both of two goals:

- to set `S-OFF`, thereby allowing you to load your own custom kernel package
- to install a custom second-stage bootloader to allow you to ignore the `S-ON` or `S-OFF` state and load your own custom kernel package.

On some devices, neither goal is achievable and you must use workarounds to carry out device customizations. Devices with completely encrypted bootloaders, such as the Milestone and DroidX, can still be customized to some extent. The amount of customization you are able to achieve on these devices is limited and the process is usually a little more complex.

## CUSTOM RECOVERIES: THE HOLY GRAIL

A recovery is a separate, standalone piece of code on a partition that can be booted in order to update Android and maintain the device. Almost all Android devices have a recovery mode into which they can be booted. One of your goals as an Android hacker is to get a custom recovery onto your device. Custom recoveries allow you to include many extra features, including easy customization and backup.

A recovery allows you to do useful things such as resetting a device to factory settings, clearing the data cache, and installing an official signed update to the Android operating system. Figure 1-2 shows the Amon Ra recovery screen. Unfortunately, the catch is that the default recovery process for most devices only installs updates to Android that have been signed with the OEM's digital signature.

If you can achieve full root and full custom recovery, you can easily change the ROM or firmware package installed on your Android device and create full file system backups, including backing up application data. Developers of custom recovery processes include many options not included in the standard Android boot process. Figure 1-3 shows the screen for the popular ClockworkMod recovery. This recovery gives you the capability of flashing a custom firmware package to your Android device very easily, as well as backing up the firmware, data, and cache and storing them on your SD card.

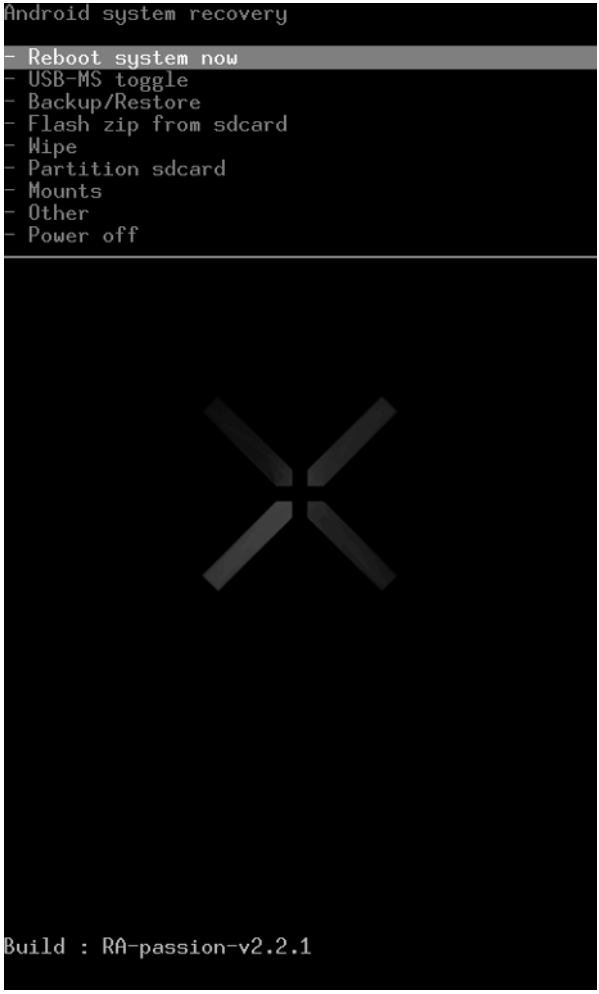


Figure 1-2: Amon Ra recovery screen

Which custom recovery you use depends on personal taste and the compatibility of your device. The Amon Ra and ClockworkMod recoveries each work on some devices. The XDA forums are a good resource to see if your device is supported by either of those custom recoveries. Typically, the process of rooting a device includes installing one of these recoveries. If your device is supported by a custom recovery, you should install it immediately after rooting. You can check the developer websites for device support.

Chapter 4 includes a complete walkthrough for the ClockworkMod recovery.

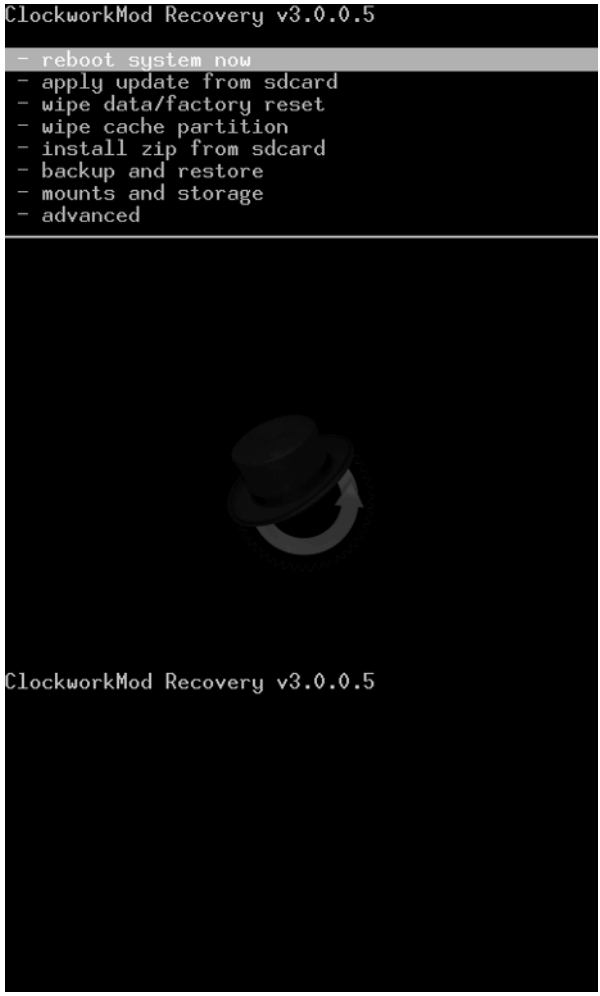


Figure 1-3: The ClockworkMod recovery screen