CHAPTER

1

# Looking at the Ecosystem

The word *Android* is used correctly in many contexts. Although the word still can refer to a humanoid robot, *Android* has come to mean much more than that in the last decade. In the mobile space, it refers to a company, an operating system, an open source project, and a development community. Some people even call mobile devices Androids. In short, an entire ecosystem surrounds the now wildly popular mobile operating system.

This chapter looks closely at the composition and health of the Android ecosystem. First you find out how Android became what it is today. Then the chapter breaks down the ecosystem stakeholders into groups in order to help you understand their roles and motivations. Finally, the chapter discusses the complex relationships within the ecosystem that give rise to several important issues that affect security.

## Understanding Android's Roots

Android did not become the world's most popular mobile operating system overnight. The last decade has been a long journey with many bumps in the road. This section recounts how Android became what it is today and begins looking at what makes the Android ecosystem tick.

## Company History

Android began as Android, Inc., a company founded by Andy Rubin, Chris White, Nick Sears, and Rich Miner in October 2003. They focused on creating mobile devices that were able to take into account location information and user preferences. After successfully navigating market demand and financial difficulties, Google acquired Android, Inc., in August 2005. During the period following, Google began building partnerships with hardware, software, and telecommunications companies with the intent of entering the mobile market.

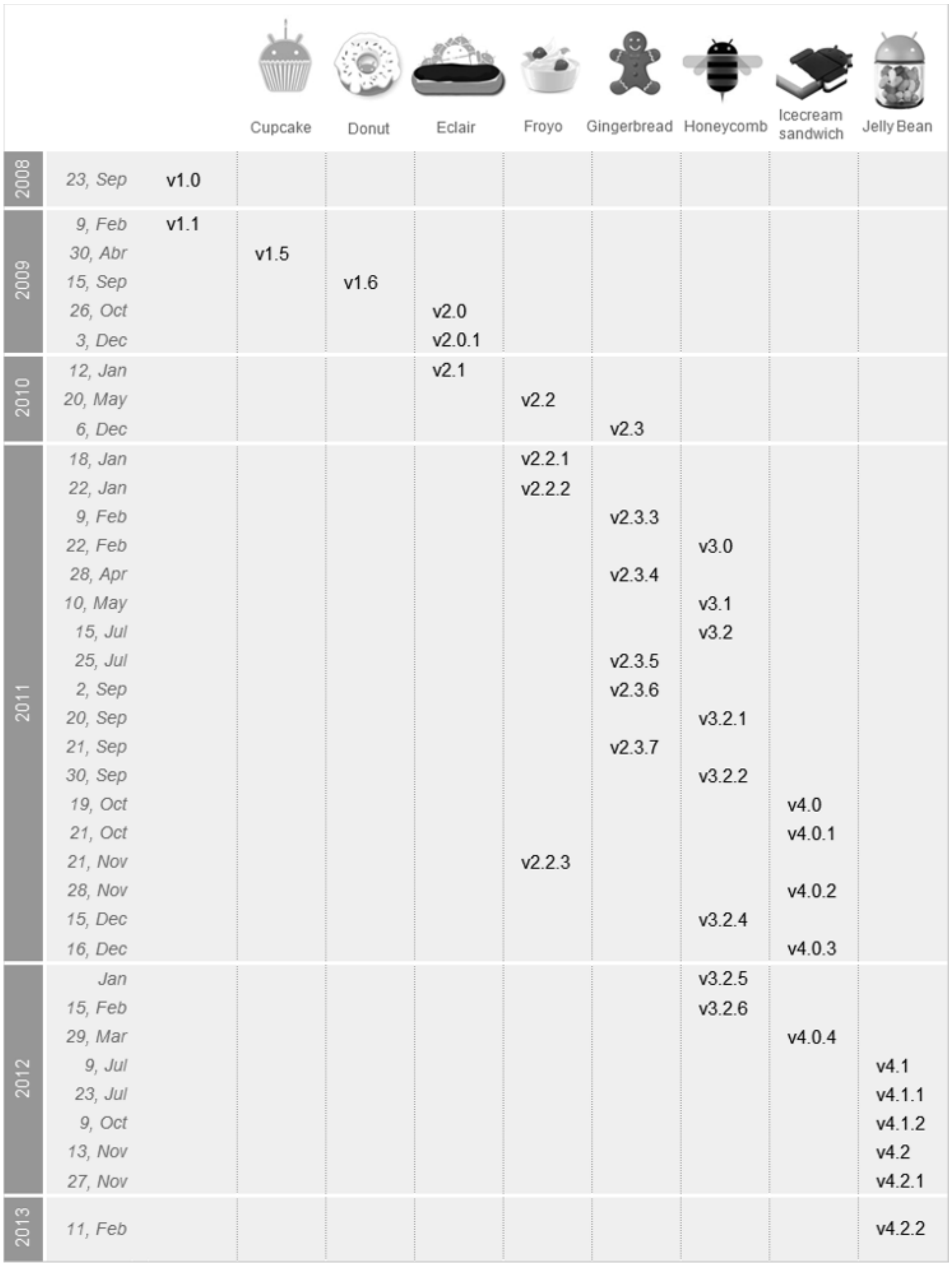
In November 2007, the Open Handset Alliance (OHA) was announced. This consortium of companies, which included 34 founding members led by Google, shares a commitment to openness. In addition, it aims to accelerate mobile platform innovation and offer consumers a richer, less expensive, and better mobile experience. The OHA has since grown to 84 members at the time this book was published. Members represent all parts of the mobile ecosystem, including mobile operators, handset manufacturers, semiconductor companies, software companies, and more. You can find the full list of members on the OHA website at `www.openhandsetalliance.com/oha_members.html`.

With the OHA in place, Google announced its first mobile product, Android. However, Google still did not bring any devices running Android to the market. Finally, after a total of five years, Android was made available to the general public in October 2008. The release of the first publicly available Android phone, the HTC G1, marked the beginning of an era.

## Version History

Before the first commercial version of Android, the operating system had Alpha and Beta releases. The Alpha releases where available only to Google and OHA members, and they were codenamed after popular robots *Astro Boy*, *Bender*, and *R2-D2*. Android Beta was released on November 5, 2007, which is the date that is popularly considered the Android birthday.

The first commercial version, version 1.0, was released on September 23, 2008, and the next release, version 1.1, was available on February 9, 2009. Those were the only two releases that did not have a naming convention for their codename. Starting with Android 1.5, which was released on April 30, 2009, the major versions' code names were ordered alphabetically with the names of tasty treats. Version 1.5 was code named *Cupcake*. Figure 1-1 shows all commercial Android versions, with their respective release dates and code names.



| | | | Cupcake | Donut | Eclair | Froyo | Gingerbread | Honeycomb | Icecream sandwich | Jelly Bean |
|---|---|---|---|---|---|---|---|---|---|---|
| 2008 | 23, Sep | v1.0 | | | | | | | | |
| 2009 | 9, Feb | v1.1 | | | | | | | | |
| | 30, Abr | | v1.5 | | | | | | | |
| | 15, Sep | | | v1.6 | | | | | | |
| | 26, Oct | | | | v2.0 | | | | | |
| | 3, Dec | | | | v2.0.1 | | | | | |
| 2010 | 12, Jan | | | | v2.1 | | | | | |
| | 20, May | | | | | v2.2 | | | | |
| | 6, Dec | | | | | | v2.3 | | | |
| 2011 | 18, Jan | | | | | v2.2.1 | | | | |
| | 22, Jan | | | | | v2.2.2 | | | | |
| | 9, Feb | | | | | | v2.3.3 | | | |
| | 22, Feb | | | | | | | v3.0 | | |
| | 28, Apr | | | | | | v2.3.4 | | | |
| | 10, May | | | | | | | v3.1 | | |
| | 15, Jul | | | | | | | v3.2 | | |
| | 25, Jul | | | | | | v2.3.5 | | | |
| | 2, Sep | | | | | | v2.3.6 | | | |
| | 20, Sep | | | | | | | v3.2.1 | | |
| | 21, Sep | | | | | | v2.3.7 | | | |
| | 30, Sep | | | | | | | v3.2.2 | | |
| | 19, Oct | | | | | | | | v4.0 | |
| | 21, Oct | | | | | | | | v4.0.1 | |
| | 21, Nov | | | | | v2.2.3 | | | | |
| | 28, Nov | | | | | | | | v4.0.2 | |
| | 15, Dec | | | | | | | v3.2.4 | | |
| | 16, Dec | | | | | | | | v4.0.3 | |
| 2012 | Jan | | | | | | | v3.2.5 | | |
| | 15, Feb | | | | | | | v3.2.6 | | |
| | 29, Mar | | | | | | | | v4.0.4 | |
| | 9, Jul | | | | | | | | | v4.1 |
| | 23, Jul | | | | | | | | | v4.1.1 |
| | 9, Oct | | | | | | | | | v4.1.2 |
| | 13, Nov | | | | | | | | | v4.2 |
| | 27, Nov | | | | | | | | | v4.2.1 |
| 2013 | 11, Feb | | | | | | | | | v4.2.2 |

**Figure 1-1:** Android releases

In the same way that Android releases are code-named, individual builds are identified with a short build code, as explained on the Code Names, Tags, and Build Numbers page at `http://source.android.com/source/build-numbers.html`. For example, take the build number JOP40D. The first letter represents the code name of the Android release (J is Jelly Bean). The second letter identifies the code branch from which the build was made, though its precise meaning varies from one build to the next. The third letter and subsequent two digits comprise a date code. The letter represents the quarter, starting from A, which means the first quarter of 2009. In the example, P represents the fourth quarter of 2012. The two digits signify days from the start of the quarter. In the example, P40 is November 10, 2012. The final letter differentiates individual versions for the same date, again starting with A. The first builds for a particular date, signified with A, don't usually use this letter.

## Examining the Device Pool

As Android has grown, so has the number of devices based on the operating system. In the past few years, Android has been slowly branching out from the typical smartphone and tablet market, finding its way into the most unlikely of places. Devices such as smart watches, television accessories, game consoles, ovens, satellites sent to space, and the new Google Glass (a wearable device with a head-mounted display) are powered by Android. The automotive industry is beginning to use Android as an infotainment platform in vehicles. The operating system is also beginning to make a strong foothold in the embedded Linux space as an appealing alternative for embedded developers. All of these facts make the Android device pool an extremely diverse place.

You can obtain Android devices from many retail outlets worldwide. Currently, most mobile subscribers get subsidized devices through their mobile carriers. Carriers provide these subsidies under the terms of a contract for voice and data services. Those who do not want to be tied to a carrier can also purchase Android devices in consumer electronics stores or online. In some countries, Google sells their Nexus line of Android devices in their online store, Google Play.

### *Google Nexus*

Nexus devices are Google's flagship line of devices, consisting mostly of smartphones and tablets. Each device is produced by a different original equipment manufacturer (OEM) in a close partnership with Google. They are sold SIM-unlocked, which makes switching carriers and traveling easy, through Google Play directly by Google. To date, Google has worked in cooperation with HTC, Samsung, LG, and ASUS to create Nexus smartphones and tablets. Figure 1-2 shows some of the Nexus devices released in recent years.

**Figure 1-2:** Google Nexus devices

Nexus devices are meant to be the reference platform for new Android versions. As such, Nexus devices are updated directly by Google soon after a new Android version is released. These devices serve as an open platform for developers. They have unlockable boot loaders that allow flashing custom Android builds and are supported by the *Android Open Source Project* (AOSP). Google also provides *factory images*, which are binary firmware images that can be flashed to return the device to the original, unmodified state.

Another benefit of Nexus devices is that they offer what is commonly referred to as a *pure Google experience*. This means that the user interface has not been modified. Instead, these devices offer the stock interface found in vanilla Android as compiled from AOSP. This also includes Google's proprietary apps such as Google Now, Gmail, Google Play, Google Drive, Hangouts, and more.

### Market Share

Smartphone market share statistics vary from one source to another. Some sources include ComScore, Kantar, IDC, and Strategy Analytics. An overall look at the data from these sources shows that Android's market share is on the rise in a large proportion of countries. According to a report released by Goldman Sachs, Android was the number one player in the entire global computing market at the end of 2012. StatCounter's GlobalStats, available at `http://gs.statcounter.com/`, show that Android is currently the number one player in the mobile operating system market, with 41.3 percent worldwide as of November 2013. Despite these small variations, all sources seem to agree that Android is the dominating mobile operating system.

### Release Adoption

Not all Android devices run the same Android version. Google regularly publishes a dashboard showing the relative percentage of devices running a given version of Android. This information is based on statistics gathered from visits to Google Play, which is present on all approved devices. The most up-to-date version of this dashboard is available at `http://developer.android.com/about/dashboards/`. Additionally, Wikipedia contains a chart showing dashboard data aggregated over time. Figure 1-3 depicts the chart as of this writing, which includes data from December 2009 to February 2013.

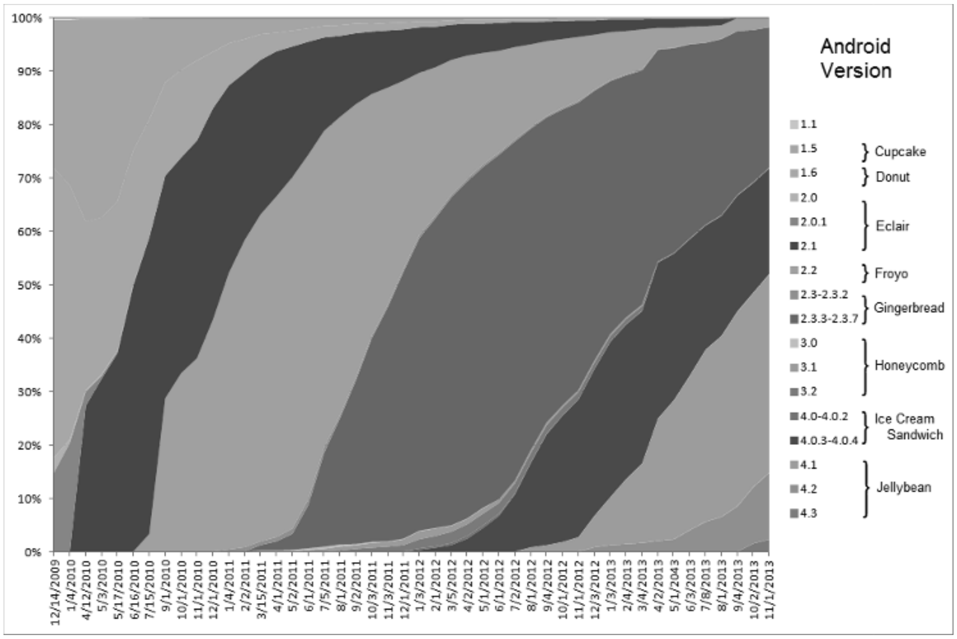


**Figure 1-3:** Android historical version distribution

Source: fjmustak (Creative Commons Attribution-Share Alike 3.0 Unported license) `http://en.wikipedia.org/wiki/File:Android_historical_version_distribution.png`

As shown, new versions of Android have a relatively slow adoption rate. It takes in excess of one year to get a new version running on 90 percent of devices. You can read more about this issue and other challenges facing Android in the "Grasping Ecosystem Complexities" section later in this chapter.

## Open Source, Mostly

AOSP is the manifestation of Google and the OHA members' commitment to openness. At its foundation, the Android operating system is built upon many different open source components. This includes numerous libraries, the Linux kernel, a complete user interface, applications, and more. All of these software components have an Open Source Initiative (OSI)–approved license. Most of the Android source is released under version 2.0 of the Apache Software License that you can find at `apache.org/licenses/LICENSE-2.0`. Some outliers do exist, mainly consisting on *upstream* projects, which are external open source projects on which Android depends. Two examples are the Linux kernel code that is licensed under GPLv2 and the WebKit project that uses a BSD-style license. The AOSP source repository brings all of these projects together in one place.

Although the vast majority of the Android stack is open source, the resulting consumer devices contain several closed source software components. Even devices from Google's flagship Nexus line contain code that ships as proprietary binary blobs. Examples include boot loaders, peripheral firmware, radio components, digital rights management (DRM) software, and applications. Many of these remain closed source in an effort to protect intellectual property. However, keeping them closed source hinders interoperability, making community porting efforts more challenging.

Further, many open source enthusiasts trying to work with the code find that Android isn't fully developed in the open. Evidence shows that Google develops Android largely in secret. Code changes are not made available to the public immediately after they are made. Instead, open source releases accompany new version releases. Unfortunately, several times the open source code was not made available at release time. In fact, the source code for Android Honeycomb (3.0) was not made available until the source code for Ice Cream Sandwich (4.0) was released. In turn, the Ice Cream Sandwich source code wasn't released until almost a month after the official release date. Events like these detract from the spirit of open source software, which goes against two of Android's stated goals: innovation and openness.

## Understanding Android Stakeholders

Understanding exactly who has a stake in the Android ecosystem is important. Not only does it provide perspective, but it also allows one to understand who is responsible for developing the code that supports various components. This section walks through the main groups of stakeholders involved, including Google, hardware vendors, carriers, developers, users, and security researchers.

This section explores each stakeholder's purpose and motivations, and it examines how the stakeholders relate to each other.

Each group is from a different field of industry and serves a particular purpose in the ecosystem. Google, having given birth to Android, develops the core operating system and manages the Android brand. Hardware fabricators make the underlying hardware components and peripherals. OEMs make the end-user devices and manage the integration of the various components that make a device work. Carriers provide voice and data access for mobile devices. A vast pool of developers, including those who are employed by members of other groups, work on a multitude of projects that come together to form Android.

Figure 1-4 shows the relationships between the main groups of ecosystem stakeholders.

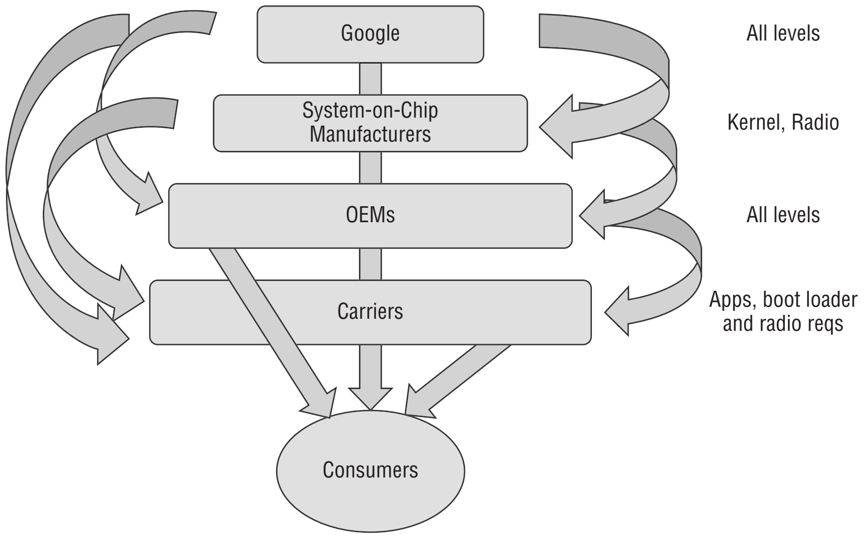

Google
System-on-Chip Manufacturers
OEMs
Carriers
Consumers
All levels
Kernel, Radio
All levels
Apps, boot loader and radio reqs

**Figure 1-4:** Ecosystem relationships

These relationships indicate who talks to who when creating or updating an Android device. As the figure clearly shows, the Android ecosystem is very complex. Such business relationships are difficult to manage and lead to a variety of complexities that are covered later in this chapter. Before getting into those issues, it's time to discuss each group in more detail.

## Google

As the company that brought Android to market, Google has several key roles in the ecosystem. Its responsibilities include legal administration, brand management, infrastructure management, in-house development, and enabling outside development. Also, Google builds its line of Nexus devices in close cooperation with its partners. In doing so, it strikes the business deals necessary to make sure that great devices running Android actually make it to market. Google's ability to execute on all of these tasks well is what makes Android appealing to consumers.

First and foremost, Google owns and manages the Android brand. OEMs cannot legally brand their devices as Android devices or provide access to Google Play unless the devices meet Google's compatibility requirements. (The details of these requirements are covered in more depth in the "Compatibility" section later in this chapter.) Because Android is open source, compatibility enforcement is one of the few ways that Google can influence what other stakeholders can do with Android. Without it, Google would be largely powerless to prevent the Android brand from being tarnished by a haphazard or malicious partner.

The next role of Google relates to the software and hardware infrastructure needed to support Android devices. Services that support apps such as Gmail, Calendar, Contacts, and more are all run by Google. Also, Google runs Google Play, which includes rich media content delivery in the form of books, magazines, movies, and music. Delivering such content requires licensing agreements with distribution companies all over the world. Additionally, Google runs the physical servers behind these services in their own data centers, and the company provides several crucial services to the AOSP, such as hosting the AOSP sources, factory image downloads, binary driver downloads, an issue tracker, and the *Gerrit* code review tool.

Google oversees the development of the core Android platform. Internally, it treats the Android project as a full-scale product development operation. The software developed inside Google includes the operating system core, a suite of core apps, and several optional non-core apps. As mentioned previously, Google develops innovations and enhancements for future Android versions in secret. Google engineers use an internal development tree that is not visible to device manufacturers, carriers, or third-party developers. When Google decides its software is ready for release, it publishes factory images, source code, and application programming interface (API) documentation simultaneously. It also pushes updates out via over-the-air (OTA) distribution channels. After a release is in AOSP, everyone can clone it and start their work building their version of the latest release. Separating development in this fashion enables developers and device manufacturers to focus on a single version without having to track the unfinished work of Google's internal teams. As true as this may be, closed development detracts from the credence of AOSP as an open source project.

Yet another role for Google lies in fostering an open development community that uses Android as a platform. Google provides third-party developers with development kits, API documentation, source code, style guidance, and more. All of these efforts help create a cohesive and consistent experience across multiple third-party applications.

By fulfilling these roles, Google ensures the vitality of the Android as a brand, a platform, and an open source project.

## Hardware Vendors

The purpose of an operating system is to provide services to applications and manage hardware connected to the device. After all, without hardware the Android operating system software wouldn't serve much purpose. The hardware of today's smartphones is very complex. With such a small form factor and lots of peripherals, supporting the necessary hardware is quite an undertaking. In order to take a closer look at the stakeholders in this group, the following sections break down hardware vendors into three subgroups that manufacture central processing units (CPUs), System-on-Chip (SoC), and devices, respectively.

### *CPU Manufacturers*

Although Android applications are processor agnostic, native binaries are not. Instead, native binaries are compiled for the specific processor used by a particular device. Android is based on the Linux kernel, which is portable and supports a multitude of processor architectures. Similarly, Android's *Native Development Kit* (NDK) includes tools for developing user-space native code for all application processor architectures supported by Android. This includes ARM, Intel x86, and MIPS.

Due to its low power consumption, the ARM architecture has become the most widely used architecture in mobile devices. Unlike other microprocessor corporations that manufacture their own CPUs, ARM Holdings only licenses its technology as intellectual property. ARM offers several microprocessor core designs, including the ARM11, Cortex-A8, Cortex-A9, and Cortex-A15. The designs usually found on Android devices today feature the ARMv7 instruction set.

In 2011, Intel and Google announced a partnership to provide support for Intel processors in Android. The Medfield platform, which features an Atom processor, was the first Intel-based platform supported by Android. Also, Intel launched the Android on Intel Architecture (Android-IA) project. This project is based on AOSP and provides code for enabling Android on Intel processors. The Android-IA website at `https://01.org/android-ia/` is targeted at system and platform developers whereas the Intel Android Developer website at `http://software.intel.com/en-us/android/` is targeted at application developers. Some Intel-based smartphones currently on the market include an Intel proprietary binary translator named `libhoudini`. This translator allows running applications built for ARM processors on Intel-based devices.

MIPS Technologies offers licenses to its MIPS architecture and microprocessor core designs. In 2009, MIPS Technologies ported Google's Android operating system to the MIPS processor architecture. Since then, several device manufacturers have launched Android devices running on MIPS processors. This is especially true for set-top boxes, media players, and tablets. MIPS Technologies offers source code for its Android port, as well as other development resources, at `http://www.imgtec.com/mips/developers/mips-android.asp`.

### System-on-Chip Manufacturers

*System-on-Chip* (SoC) is the name given to a single piece of silicon that includes the CPU core, along with a graphics processing unit (GPU), random access memory (RAM), input/output (I/O) logic, and sometimes more. For example, many SoCs used in smartphones include a baseband processor. Currently, most SoCs used in the mobile industry include more than one CPU core. Combining the components on a single chip reduces manufacturing costs and decreases power consumption, ultimately leading to smaller and more efficient devices.

As mentioned previously, ARM-based devices dominate the Android device pool. Within ARM devices, there are four main SoC families in use: OMAP from Texas Instruments, Tegra from nVidia, Exynos from Samsung, and Snapdragon from Qualcomm. These SoC manufacturers license the CPU core design from ARM Holdings. You can find a full list of licensees on ARM's website at `www.arm.com/products/processors/licensees.php`. With the exception of Qualcomm, SoC manufacturers use ARM's designs without modification. Qualcomm invests additional effort to optimize for lower power consumption, higher performance, and better heat dissipation.

Each SoC has different components integrated into it and therefore requires different support in the Linux kernel. As a result, development for each SoC is tracked separately in a Git repository specific to that SoC. Each tree includes SoC-specific code including drivers and configurations. On several occasions, this separation has led to vulnerabilities being introduced into only a subset of the SoC-specific kernel source repositories. This situation contributes to one of the key complexities in the Android ecosystem, which is discussed further in the "Grasping Ecosystem Complexities" section later in this chapter.

### Device Manufacturers

Device manufacturers, including original design manufacturers (ODMs) and OEMs, design and build the products used by consumers. They decide which combination of hardware and software will make it into the final unit and take care of all of the necessary integration. They choose the hardware components that will be combined together, the device form factor, screen size, materials, battery, camera lens, sensors, radios, and so on. Usually device manufacturers partner up with a SoC manufacturer for a whole line of products. Most choices made when creating a new device relate directly to market differentiation, targeting a particular customer segment, or building brand loyalty.

While developing new products, device manufacturers have to adapt the Android platform to work well on its new hardware. This task includes adding new kernel device drivers, proprietary bits, and user-space libraries. Further, OEMs often make custom modifications to Android, especially in the Android Framework. To comply with the GPLv2 license of the Android kernel, OEMs are forced to release kernel sources. However, the Android Framework is licensed under the Apache 2.0 License, which allows modifications to be redistributed in binary form without having to release the source code. This is where most vendors try to put their innovations to differentiate their devices from others. For example, the *Sense* and *Touchwiz* user interface modifications made by HTC and Samsung are implemented primarily in the Android Framework. Such modifications are a point of contention because they contribute to several complex, security-related problems in the ecosystem. For example, customizations may introduce new security issues. You can read more about these complexities in the "Grasping Ecosystem Complexities" section, later in this chapter.

## Carriers

Aside from providing mobile voice and data services, carriers close deals with device manufacturers to subsidize phones to their clients. The phones obtained through a carrier usually have a carrier-customized software build. These builds tend to have the carrier logo in the boot screen, preconfigured Access Point Name (APN) network settings, changes in the default browser home page and browser bookmarks, and a lot of pre-loaded applications. Most of the time these changes are embedded into the system partition so that they cannot be removed easily.

In addition to adding customization to the device's firmware, carriers also have their own quality assurance (QA) testing procedures in place. These QA processes are reported to be lengthy and contribute to the slow uptake of software updates. It is very common to see an OEM patch a security hole in the operating system for its unbranded device while the carrier-branded device remains vulnerable for much longer. It's not until the update is ready to be distributed to the carrier devices that subsidized users are updated. After they have been available for some time, usually around 12 to 18 months, devices are discontinued. Some devices are discontinued much more quickly—in a few cases even immediately after release. After that point, any users still using such a device will no longer receive updates, regardless of whether they are security related or not.

## Developers

As an open source operating system, Android is an ideal platform for developers to play with. Google engineers are not the only people contributing code to the Android platform. There are a lot of individual developers and entities who contribute to AOSP on their own behalf. Every contribution to AOSP (coming either from Google or from a third party) has to use the same code style and be processed through Google's source code review system, *Gerrit*. During the code review process, someone from Google decides whether to include or exclude the changes.

Not all developers in the Android ecosystem build components for the operating system itself. A huge portion of developers in the ecosystem are application developers. They use the provided software development kits (SDKs), frameworks, and APIs to build apps that enable end users to achieve their goals. Whether these goals are productivity, entertainment, or otherwise, app developers aim to meet the needs of their user base.

In the end, developers are driven by popularity, reputation, and proceeds. *App markets* in the Android ecosystem offer developers incentives in the form of revenue sharing. For example, advertisement networks pay developers for placing ads in their applications. In order to maximize their profits, app developers try to become extremely popular while maintaining an upstanding reputation. Having a good reputation, in turn, drives increased popularity.

### *Custom ROMs*

The same way manufacturers introduce their own modifications to the Android platform, there are other custom firmware projects (typically called *ROMs*) developed by communities of enthusiasts around the world. One of the most popular Android custom firmware projects is *CyanogenMod*. With 9.5 million active installs in December 2013, it is developed based on the official releases of Android with additional original and third-party code. These community-modified versions of Android usually include performance tweaks, interface enhancements, features, and options that are typically not found in the official firmware distributed with the device. Unfortunately, they often undergo less extensive testing and quality assurance. Further, similar to the situation with OEMs, modifications made in custom ROMs may introduce additional security issues.

Historically, device manufacturers and mobile carriers have been unsupportive of third-party firmware development. To prevent users from using custom ROMs, they place technical obstacles such as locked boot loaders or

NAND locks. However, custom ROMs have grown more popular because they provide continued support for older devices that no longer receive official updates. Because of this, manufacturers and carriers have softened their positions regarding unofficial firmware. Over time, some have started shipping devices with unlocked or unlockable boot loaders, similar to Nexus devices.

## Users

Android would not be the thriving community that it is today without its massive user base. Although each individual user has unique needs and desires, they can be classified into one of three categories. The three types of end users include general consumers, power users, and security researchers.

### *Consumers*

Since Android is the top-selling smartphone platform, end users enjoy a wide range of devices to choose from. Consumers want a single, multifunction device with personal digital assistant (PDA) functions, camera, global position system (GPS) navigation, Internet access, music player, e-book reader, and a complete gaming platform. Consumers usually look for a productivity boost, to stay organized, or stay in touch with people in their lives, to play games on the go and to access information from various sources on the Internet. On top of all this, they expect a reasonable level of security and privacy.

The openness and flexibility of Android is also apparent to consumers. The sheer number of available applications, including those installable from sources outside official means, is directly attributable to the open development community. Further, consumers can extensively customize their devices by installing third-party launchers, home screen widgets, new input methods, or even full custom ROMs. Such flexibility and openness is often the deciding factor for those who choose Android over competing smartphone operating systems.

### *Power Users*

The second type of user is a special type of consumer called *power users* in this text. Power users want to have the ability to use features that are beyond what is enabled in stock devices. For example, users who want to enable Wi-Fi tethering on their devices are considered members of this group. These users are intimately familiar with advanced settings and know the limitations of their devices. They are much less averse to the risk of making unofficial changes to the Android operating system, including running publicly available exploits to gain elevated access to their devices.

### *Security Researchers*

You can consider security researchers a subset of power users, but they have additional requirements and differing goals. These users can be motivated by fame, fortune, knowledge, openness, protecting systems, or some combination of these ideals. Regardless of their motivations, security researchers aim to discover previously unknown vulnerabilities in Android. Conducting this type of research is far easier when full access to a device is available. When elevated access is not available, researchers usually seek to obtain elevated access first. Even with full access, this type of work is challenging.

Achieving the goals of a security researcher requires deep technical knowledge. Being a successful security researcher requires a solid understanding of programming languages, operating system internals, and security concepts. Most researchers are competent in developing, reading, and writing several different programming languages. In some ways, this makes security researchers members of the developers group, too. It's common for security researchers to study security concepts and operating system internals at great length, including staying on top of cutting edge information.

The security researcher ecosystem group is the primary target audience of this book, which has a goal of both providing base knowledge for budding researchers and furthering the knowledge of established researchers.

## Grasping Ecosystem Complexities

The OHA includes pretty much all major Android vendors, but some parties are working with different goals. Some of these goals are competing. This leads to various partnerships between manufacturers and gives rise to some massive cross-organizational bureaucracy. For example, Samsung memory division is one of the world's largest manufacturers of NAND flash. With around 40 percent market share, Samsung produces dynamic random access memory (DRAM) and NAND memory even for devices made by competitors of its mobile phones division. Another controversy is that although Google does not directly earn anything from the sale of each Android device, Microsoft and Apple have successfully sued Android handset manufacturers to extract patent royalty payments from them. Still, this is not the full extent of the complexities that plague the Android ecosystem.

Apart from legal battles and difficult partnerships, the Android ecosystem is challenged by several other serious problems. Fragmentation in both hardware and software causes complications, only some of which are addressed by Google's compatibility standards. Updating the Android operating system itself remains a significant challenge for all of the ecosystem stakeholders. Strong roots in open source further complicate software update issues, giving rise to increased exposure to known vulnerabilities. Members of the security research community are troubled with the dilemma of deciding between security and openness. This dilemma extends to other stakeholders as well, leading to a terrible disclosure track record. The following sections discuss each of these problem areas in further detail.

## Fragmentation

The Android ecosystem is rampant with *fragmentation*, due to the differences between the multitudes of various Android devices. The open nature of Android makes it ideal for mobile device manufacturers to build their own devices based off the platform. As a result, the device pool is made up of many different devices from many different manufacturers. Each device is composed of a variety of software and hardware, including OEM or carrier-specific modifications. Even on the same device, the version of Android itself might vary from one carrier or user to another. Because of all of these differences, consumers, developers, and security researchers wrestle with fragmentation regularly.

Although fragmentation has relatively little effect on consumers, it is slightly damaging to the Android brand. Consumers accustomed to using Samsung devices who switch to a device from HTC are often met with a jarring experience. Because Samsung and HTC both highly customize the user experience of their devices, users have to spend some time reacquainting themselves with how to use their new devices. The same is also true for longtime Nexus device users who switch to OEM-branded devices. Over time, consumers may grow tired of this issue and decide to switch to a more homogeneous platform. Still, this facet of fragmentation is relatively minor.

Application developers are significantly more affected by fragmentation than consumers. Issues primarily arise when developers attempt to support the variety of devices in the device pool (including the software that runs on them). Testing against all devices is very expensive and time intensive. Although using the emulator can help, it's not a true representation of what users on actual devices will encounter. The issues developers must deal with include differing hardware configurations, API levels, screen sizes, and peripheral availability. Samsung has more than 15 different screen sizes for its Android devices, ranging from 2.6 inches to 10.1 inches. Further, High-Definition Multimedia Interface (HDMI) dongles and Google TV devices that don't have a touchscreen require specialized input handling and user interface (UI) design. Dealing with all of this fragmentation is no easy task, but thankfully Google provides developers with some facilities for doing so.

Developers create applications that perform well across different devices, in part, by doing their best to hide fragmentation issues. To deal with differing screen sizes, the Android UI framework allows applications to query the device screen size. When an app is designed properly, Android automatically adjusts application assets and UI layouts appropriately for the device. Google Play also allows app developers to deal with differing hardware configurations by declaring requirements within the application itself. A good example is an application that requires a touchscreen. On a device without a touchscreen, viewing such an app on Google Play shows that the app does not support the device and cannot be installed. The Android application Support Library transparently deals with some API-level differences. However, despite all of the resources available, some compatibility issues remain. Developers are left to do their best in these corner cases, often leading to frustration. Again, this weakens the Android ecosystem in the form of developer disdain.

For security, fragmentation is both positive and negative, depending mostly on whether you take the perspective of an attacker or a defender. Although attackers might easily find exploitable issues on a particular device, those issues are unlikely to apply to devices from a different manufacturer. This makes finding flaws that affect a large portion of the ecosystem difficult. Even when equipped with such a flaw, variances across devices complicate exploit development. In many cases, developing a universal exploit (one that works across all Android versions and all devices) is not possible. For security researchers, a comprehensive audit would require reviewing not only every device ever made, but also every revision of software available for those devices. Quite simply put, this is an insurmountable task. Focusing on a single device, although more approachable, does not paint an adequate picture of the entire ecosystem. An attack surface present on one device might not be present on another. Also, some components are more difficult to audit, such as closed source software that is specific to each device. Due to these challenges, fragmentation simultaneously makes the job of an auditor more difficult and helps prevent large-scale security incidents.

## Compatibility

One complexity faced by device manufacturers is compatibility. Google, as the originator of Android, is charged with protecting the Android brand. This includes preventing fragmentation and ensuring that consumer devices are compatible with Google's vision. To ensure device manufacturers comply with the hardware and software compatibility requirements set by Google, the company publishes a compatibility document and a test suite. All manufacturers who want to distribute devices under the Android brand have to follow these guidelines.

### Compatibility Definition Document

The Android *Compatibility Definition Document* (CDD) available at `http://source.android.com/compatibility/` enumerates the software and hardware requirements of a "compatible" Android device. Some hardware must be present on all Android devices. For example, the CDD for Android 4.2 specifies that all device implementations must include at least one form of audio output, and one or more forms of data networking capable of transmitting data at 200K bit/s or greater. However, the inclusion of various peripherals is left up to the device manufacturer. If certain peripherals are included, the CDD specifies some additional requirements. For example, if the device manufacturer decides to include a rear-facing camera, then the camera must have a resolution of at least 2 megapixels. Devices must follow CDD requirements to bear the Android moniker and, further, to ship with Google's applications and services.

### Compatibility Test Suite

The Android *Compatibility Test Suite* (CTS) is an automated testing harness that executes unit tests from a desktop computer to the attached mobile devices. CTS tests are designed to be integrated into continuous build systems of the engineers building a Google-certified Android device. Its intent is to reveal incompatibilities early on, and ensure that the software remains compatible throughout the development process.

As previously mentioned, OEMs tend to heavily modify parts of the Android Framework. The CTS makes sure that APIs for a given version of the platform are unmodified, even after vendor modifications. This ensures that application developers have a consistent development experience regardless of who produced the device.

The tests performed in the CTS are open source and continually evolving. Since May 2011, the CTS has included a test category called *security* that centralizes tests for security bugs. You can review the current security tests in the master branch of AOSP at `https://android.googlesource.com/platform/cts/+/master/tests/tests/security`.

## Update Issues

Unequivocally, the most important complexity in the Android ecosystem relates to the handling of software updates, especially security fixes. This issue is fueled by several other complexities in the ecosystem, including third-party software, OEM customizations, carrier involvement, disparate code ownership, and more. Problems keeping up with upstream open source projects, technical issues with deploying operating system updates, lack of back-porting, and a defunct alliance are at the heart of the matter. Overall, this is the single largest factor contributing to the large number of insecure devices in use in the Android ecosystem.

### *Update Mechanisms*

The root cause of this issue stems from the divergent processes involved in updating software in Android. Updates for apps are handled differently than operating system updates. An app developer can deploy a patch for a security flaw in his app via Google Play. This is true whether the app is written by Google, OEMs, carriers, or independent developers. In contrast, a security flaw in the operating system itself requires deploying a firmware upgrade or OTA update. The process for creating and deploying these types of updates is far more arduous.

For example, consider a patch for a flaw in the core Android operating system. A patch for such an issue begins with Google fixing the issue first. This is where things get tricky and become device dependent. For Nexus devices, the updated firmware can be released directly to end users at this point. However, updating an OEM-branded device still requires OEMs to produce a build including Google's security fix. In another twist, OEMs can deliver the updated firmware directly to end users of unlocked OEM devices at this point. For carrier-subsidized devices, the carrier must prepare its customized build including the fix and deliver it to the customer base. Even in this simple example, the update path for operating system vulnerabilities is far more complicated than application updates. Additional problems coordinating with third-party developers or low-level hardware manufacturers could also arise.

### *Update Frequency*

As previously mentioned, new versions of Android are adopted quite slowly. In fact, this particular issue has spurred public outcry on several occasions. In April 2013, the American Civil Liberties Union (ACLU) filed a complaint with the Federal Trade Commission (FTC). They stated that the four major mobile carriers in the U.S. did not provide timely security updates for the Android smartphones they sell. They further state that this is true even if Google has published updates to fix exploitable security vulnerabilities. Without receiving timely security updates, Android cannot be considered a mature, safe, or secure operating system. It's no surprise that people are looking for government action on the matter.

The time delta between bug reporting, fix development, and patch deployment varies widely. The time between bug reporting and fix development is often short, on the order of days or weeks. However, the time between fix development and that fix getting deployed on an end user's device can range from weeks to months, or possibly never. Depending on the particular issue, the overall patch cycle could involve multiple ecosystem stakeholders. Unfortunately, end users pay the price because their devices are left vulnerable.

Not all security updates in the Android ecosystem are affected by these complexities to the same degree. For example, apps are directly updated by their authors. App authors' ability to push updates in a timely fashion has led to several quick patch turnarounds in the past. Additionally, Google has proven their ability to deploy firmware updates for Nexus devices in a reasonable time frame. Finally, power users sometimes patch their own devices at their own risk.

Google usually patches vulnerabilities in the AOSP tree within days or weeks of the discovery. At this point, OEMs can cherry-pick the patch to fix the vulnerability and merge it into their internal tree. However, OEMs tend to be slow in applying patches. Unbranded devices usually get updates faster than carrier devices because they don't have to go through carrier customizations and carrier approval processes. Carrier devices usually take months to get the security updates, if they ever get them.

### *Back-porting*

The term *back-porting* refers to the act of applying the fix for a current version of software to an older version. In the Android ecosystem, back-ports for security fixes are mostly nonexistent. Consider a hypothetical scenario: The latest version of Android is 4.2. If a vulnerability is discovered that affects Android 4.0.4 and later, Google fixes the vulnerability only in 4.2.x and later versions. Users of prior versions such as 4.0.4 and 4.1.x are left vulnerable indefinitely. It is believed that security fixes may be back-ported in the event of a widespread attack. However, no such attack is publicly known at the time of this writing.

### *Android Update Alliance*

In May 2011, during Google I/O, Android Product Manager Hugo Barra announced the Android Update Alliance. The stated goal of this initiative was to encourage partners to make a commitment to update their Android devices for at least 18 months after initial release. The update alliance was formed by HTC, LG, Motorola, Samsung, Sony Ericsson, AT&T, T-Mobile, Sprint, Verizon, and Vodafone. Unfortunately, the Android Update Alliance has never been mentioned again after the initial announcement. Time has shown that the costs of developing new firmware versions, issues with legacy devices, problems in newly released hardware, testing problems on new versions, or development issues could stand in the way of timely updates happening. This is especially problematic on poorly selling devices where carriers and manufacturers have no incentive to invest in updates.

### *Updating Dependencies*

Keeping up with upstream open source projects is a cumbersome task. This is especially true in the Android ecosystem because the patch lifecycle is so extended. For example, the Android Framework includes a web browser engine called WebKit. Several other projects also use this engine, including Google's own Chrome web browser. Chrome happens to have an admirably short patch lifecycle, on the order of weeks. Unlike Android, it also has a successful bug bounty program in which Google pays for and discloses discovered vulnerabilities with each patch release. Unfortunately, many of these bugs are present in the code used by Android. Such a bug is often referred to as a *half-day* vulnerability. The term is born from the term *half-life*, which measures the rate at which radioactive material decays. Similarly, a half-day bug is one that is decaying. Sadly, while it decays, Android users are left exposed to attacks that may leverage these types of bugs.

## Security versus Openness

One of the most profound complexities in the Android ecosystem is between power users and security-conscious vendors. Power users want and need to have unfettered access to their devices. Chapter 3 discusses the rationale behind these users' motivations further. In contrast, a completely secure device is in the best interests of vendors and everyday end users. The needs of power users and vendors give rise to interesting challenges for researchers.

As a subset of all power users, security researchers face even more challenging decisions. When researchers discover security issues, they must decide what they do with this information. Should they report the issue to the vendor? Should they disclose the issue openly? If the researcher reports the issue, and the vendor fixes it, it might hinder power users from gaining the access they desire. Ultimately, each researcher's decision is driven by individual motivations. For example, researchers routinely withhold disclosure when a publicly viable method to obtain access exists. Doing so ensures that requisite access is available in the event that vendors fix the existing, publicly disclosed methods. It also means that the security issues remain unpatched, potentially allowing malicious actors to take advantage of them. In some cases, researchers choose to release heavily obfuscated exploits. By making it difficult for the vendors to discover the leveraged vulnerability, power users are able to make use of the exploit longer. Many times, the vulnerabilities used in these exploits can only be used with physical access to the device. This helps strike a balance between the conflicting wants of these two stakeholder groups.

Vendors also struggle to find a balance between security and openness. All vendors want satisfied customers. As mentioned previously, vendors modify

Android in order to please users and differentiate themselves. Bugs can be introduced in the process, which detracts from overall security. Vendors must decide whether to make such modifications. Also, vendors support devices after they are purchased. Power user modifications can destabilize the system and lead to unnecessary support calls. Keeping support costs low and protecting against fraudulent warranty replacements are in the vendors' best interests. To deal with this particular issue, vendors employ boot loader locking mechanisms. Unfortunately, these mechanisms also make it more difficult for competent power users to modify their devices. To compromise, many vendors provide ways for end users to unlock devices. You can read more about these methods in Chapter 3.

## Public Disclosures

Last but not least, the final complexity relates to public disclosures, or public announcement, of vulnerabilities. In information security, these announcements serve as notice for system administrators and savvy consumers to update the software to remediate discovered vulnerabilities. Several metrics, including full participation in the disclosure process, can be used to gauge a vendor's security maturity. Unfortunately, such disclosures are extremely rare in the Android ecosystem. Here we document known public disclosures and explore several possible reasons why this is the case.

In 2008, Google started the `android-security-announce` mailing list on Google groups. Unfortunately, the list contains only a single post introducing the list. You can find that single message at `https://groups.google.com/d/msg/android-security-announce/aEba2l7U23A/vOyOllbBxw8J`. After the initial post, not a single official security announcement was ever made. As such, the only way to track Android security issues is by reading change logs in AOSP, tracking Gerrit changes, or separating the wheat from chaff in the Android issue tracker at `https://code.google.com/p/android/issues/list`. These methods are time consuming, error prone, and unlikely to be integrated into vulnerability assessment practices.

Although it is not clear why Google has not followed through with their intentions to deliver security announcements, there are several possible reasons. One possibility involves the extended exposure to vulnerabilities ramping in the Android ecosystem. Because of this issue, it's possible that Google views publicly disclosing fixed issues as irresponsible. Many security professionals, including the authors of this text, believe that the danger imposed by such a disclosure is far less than that of the extended exposure itself. Yet another possibility involves the complex partnerships between Google, device manufacturers, and carriers. It is easy to see how disclosing a vulnerability that remains present in a business partner's product could be seen as bad business. If this is the case, it means Google is prioritizing a business relationship before the good of the public.

Google aside, very few other Android stakeholders on the vendor side have conducted public disclosures. Many OEMs have avoided public disclosure entirely, even shying away from press inquiries about hot-button vulnerabilities. For example, while HTC has a disclosure policy posted at `www.htc.com/www/terms/product-security/`, the company has never made a public disclosure to date. On a few occasions, carriers have mentioned that their updates include "important security fixes." On even fewer occasions, carriers have even referenced public CVE numbers assigned to specific issues.

The *Common Vulnerabilities and Exposures* (CVE) project aims to create a central, standardized tracking number for vulnerabilities. Security professionals, particularly vulnerability experts, use these numbers to track issues in software or hardware. Using CVE numbers greatly improves the ability to identify and discuss an issue across organizational boundaries. Companies that embrace the CVE project are typically seen as the most mature since they recognize the need to document and catalog past issues in their products.

Of all of the stakeholders on the vendor side, one has stood out as taking public disclosure seriously. That vendor is Qualcomm, with its Code Aurora forum. This group is a consortium of companies with projects serving the mobile wireless industry and is operated by Qualcomm. The Code Aurora website has a security advisories page available at `https://www.codeaurora.org/projects/security-advisories`, with extensive details about security issues and CVE numbers. This level of maturity is one that other stakeholders should seek to follow so that the security of the Android ecosystem as a whole can improve.

In general, security researchers are the biggest proponents of public disclosures in the Android ecosystem. Although not every security researcher is completely forthcoming, they are responsible for bringing issues to the attention of all of the other stakeholders. Often issues are publicly disclosed by independent researchers or security companies on mailing lists, at security conferences, or on other public forums. Increasingly, researchers are coordinating such disclosures with stakeholders on the vendor side to safely and quietly improve Android security.

## Summary

In this chapter you have seen how the Android operating system has grown over the years to conquer the mobile operating system (OS) market from the bottom up. The chapter walked you through the main players involved in the Android ecosystem, explaining their roles and motivations. You took a close look at the various problems that plague the Android ecosystem, including how they affect security. Armed with a deep understanding of Android's complex ecosystem, one can easily pinpoint key problem areas and apply oneself more effectively to the problem of Android security.

The next chapter provides an overview of the security design and architecture of Android. It dives under the hood to show how Android works, including how security mechanisms are enforced.