

Android software development kit and android debug bridge

3

INFORMATION IN THIS CHAPTER

- Android platforms
- Software development kit (SDK)
- Android security model
- Forensics and the SDK

INTRODUCTION

The Android software development kit (SDK) provides not only the tools to create applications that run on the Android platform but it also provides documentation and utilities that can assist significantly in the forensic or security analysis of a device. While the Android hardware covered in Chapter 2 plays a major role in the capabilities of a device, the software harnesses these features to ultimately create the experience and functionality consumers seek. A thorough understanding of the Android SDK will provide many insights into the data and the device, as well as important utilities that we will leverage in investigations.

ANDROID PLATFORMS

Android was officially announced in November 2007 but has been under significant development since 2005. This, combined with the large and diverse hardware, which leverages Android, has created a diverse ecosystem adding significant complexity for the forensic analyst or security engineer.

An informative characteristic of Android is the version of the Android platform itself. The platform is a large factor in determining the features a device can support. The official Android platforms are each assigned an application programming interface (API) level, and all the newer versions receive a code name. The current release, as of January 2011, is Android 2.3 which has the code name Gingerbread. The next major release has a code name Honeycomb and appears to

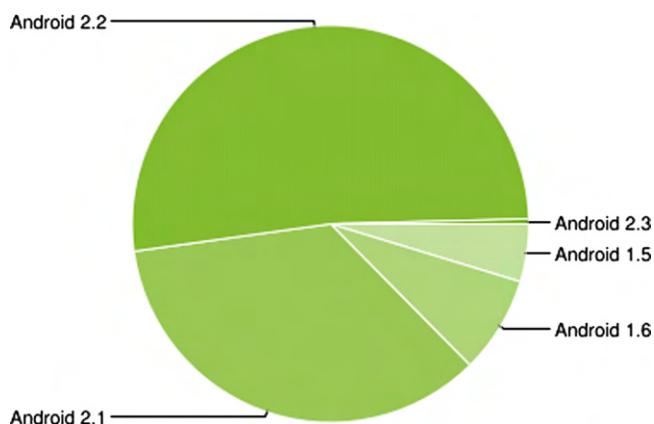
Table 3.1 Android Platforms

Platform	API Level	Code Name	Release Date
Android 2.3.3	10	Gingerbread	February 9, 2011
Android 2.3	9	Gingerbread	December 2010
Android 2.2	8	FroYo	May 20, 2010
Android 2.1	7	Eclair	January 11, 2010
Android 2.0.1	6	Eclair	December 11, 2009
Android 2.0	5	Eclair	October 5, 2009
Android 1.6	4	Donut	September 16, 2009
Android 1.5	3	Cupcake	April 27, 2009
Android 1.1	2	Petit Four	February 9, 2009
Android 1.0	1	N/A	September 23, 2008

target the anticipated growth of tablet devices. Table 3.1 gives the full list of Android platforms including API level, code name, and release date (Android timeline, n.d.).

While many Android versions exist, the distribution of each in current devices can have a large impact on forensic analysts and security engineers. Figure 3.1 shows Google's reports of distribution of Android versions based on a two-week survey of devices accessing the Android Market (Platform Versions, n.d.).

To put that in perspective, Table 3.2 shows the total number of devices in circulation in the United States by Android version. These data are based on an

**FIGURE 3.1**

Distribution of Android devices by platforms, January 2011.

Table 3.2 Approximate Number of Android Devices by Platform in the United States

Android Version	Total Devices
Android 2.3	63,960
Android 2.2	8,282,820
Android 2.1	5,628,480
Android 1.6	1,263,210
Android 1.5	751,530

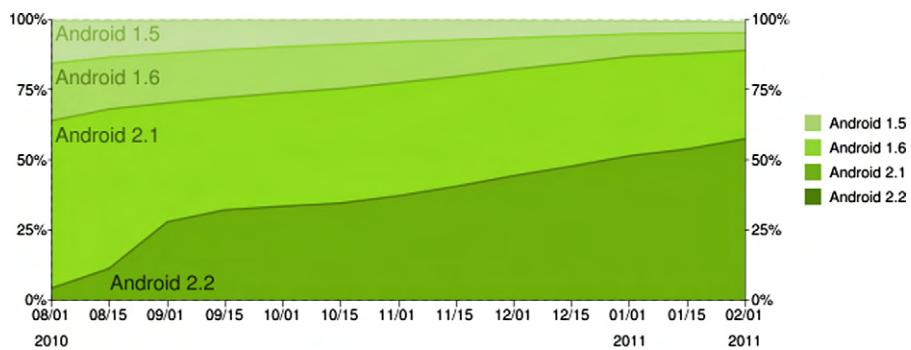


FIGURE 3.2

Historical distribution of Android version from August 2010 through February 2, 2011.

approximate US Android device population of 15.99 million as of November 2011 ([comScore Reports, n.d.](#)).

Google also released a graph displaying the historical distribution of Android versions for the seven-month period between August 2010 and February 2, 2011. The data are again based on devices accessing the Android Market but nicely displayed the progress of Android updates over time as shown in [Fig. 3.2 \(Platform Versions, n.d.\)](#).

While some devices will never support the latest version of Android, many do eventually receive the update. Future devices will probably be able to quickly support and upgrade to the latest version. However, from a forensics and security perspective, the older outliers cannot be ignored.

Android Platform Highlights Through 2.3.3 (Gingerbread)

Android is a sophisticated, heavily developed platform and any attempt to fully document all features would encompass a large portion of this book. However,

a brief overview of each major release can be helpful so that a forensic analyst is aware of the features a device may support. Generally speaking, the features build on each other so functionality available in Android 1.5 is likely available and improved in Android 2.3.3.

Android 1.5

Android 1.5, released April 2009, highlighted the features and updates listed in Table 3.3 (Android 1.5, n.d.).

Android 1.6

Android 1.6, released September 2009, highlighted the features and updates listed in Table 3.4 (Android 1.6, n.d.).

Table 3.3 Android 1.5 Features and Highlights

New User Features	New Developer Features, APIs, and Technologies	Built-in Applications
<ul style="list-style-type: none">• User Interface Refinements, including in-call experience, SMS/MMS and more• Performance Improvements to camera, GPS, browser, and Gmail• On-screen soft keyboard• Home screen widgets• Video recording and playback• Better Bluetooth support and functionality• Browser copy and paste, on-page searching, and more• Contact improvements including pictures, date/time stamps for call logs, and on-touch access to contact methods• View Google Talk friends' status in Contacts, SMS, MMS, Gmail, and e-mail applications• Upload videos to YouTube, pictures to Picasa	<ul style="list-style-type: none">• New Linux kernel (version 2.6.27)• SD card file system auto-checking and repair• Improved media framework• Speech recognition framework• Support 26 locales	<ul style="list-style-type: none">• Alarm clock• Browser• Calculator• Camcorder• Camera• Contacts• Custom locale (developer app)• Dev. tools (developer app)• Dialer• E-mail• Gallery• IME for Japanese text input• Messaging• Music• Settings• Spare parts (developer app)

Table 3.4 Android 1.6 Features and Highlights

New User Features	New Developer Features, APIs, and Technologies	Built-in Applications
<ul style="list-style-type: none"> Quick Search Box for Android Updated camera, camcorder, and gallery VPN, 802.1x support Battery usage indicator Android Market Updates including categorization, top apps, and screenshots 	<ul style="list-style-type: none"> 2.6.29 Linux kernel Expanded search framework Text-to-speech engine Support for gestures New accessibility framework Expanded support for screen densities and resolutions Telephony support for CDMA New version of OpenCore for better audio handling 	<ul style="list-style-type: none"> All apps in Android 1.5 Gestures Builder

Androids 2.0 and 2.1

Android 2.0 and 2.1, released October 2009 and January 2010, respectively, highlighted the features and updates listed in [Table 3.5 \(Android 2.1, n.d.\)](#).

Table 3.5 Android 2.0/2.1 Features and Highlights

New User Features	New Developer Features, APIs, and Technologies	Built-in Applications
<ul style="list-style-type: none"> Multiple accounts for e-mail and contact syncing, Quick contact feature Exchange support in e-mail SMS/MMS search functionality Many enhancements to camera such as built-in flash, digital zoom, and more Improvement in Android virtual keyboard Browser updates include bookmarks with web page thumbnails, double-tap zoom, and HTML5 support New calendar features such as inviting guests 	<ul style="list-style-type: none"> Revamped graphics architecture for improved performance that enables better hardware acceleration. Bluetooth 2.1 Live Wallpapers API 	<ul style="list-style-type: none"> Same apps as Android 1.6

Android 2.2

Android 2.2, released May 2010, highlighted the features and updates found in [Table 3.6](#).

Table 3.6 Android 2.2 Features and Highlights

New User Features	New Developer Features, APIs, and Technologies	Built-in Applications
<ul style="list-style-type: none"> • New Home screen tips widget • The Phone, applications Launcher, and Browser now have dedicated shortcuts on the Home screen • Exchange expanded with addition of numeric pin or alpha-numeric password options to unlock device; Remote wipe; Exchange Calendars are now supported; Auto-discovery; Global Address Lists look-up • Improved camera and gallery • Some devices can be a portable Wi-Fi hotspot that can be shared with up to eight devices. • Multiple keyboard languages • Improved performance in browser, Dalvik VM, graphics, and kernel memory management 	<ul style="list-style-type: none"> • 2.6.32 Linux kernel (support for RAM > 256 MB) • New media framework that supports local file playback and HTTP progressive streaming • Bluetooth improvements including voice dialing over Bluetooth, share contacts with other phones, and better compatibility with vehicles • Android Cloud to Device Messaging • Android Application Error Reports • Apps on external storage • Data backup APIs • Device policy manager 	<ul style="list-style-type: none"> • Same as Android 2.1

Android 2.3

Android 2.3, released December 2010, highlighted the features and updates listed in [Table 3.7](#).

Table 3.7 Android 2.3 Features and Highlights

New User Features	New Developer Features, APIs, and Technologies	Built-in Applications
<ul style="list-style-type: none"> • UI refinements for simplicity and speed • Faster, more intuitive text input • One-touch word selection and copy/paste • Improved power management • Support for Internet/SIP calling (VoIP) • NFC Reader application lets the user read and interact with near-field communication (NFC) tags. • Downloads management • Camera improvements, support for front- and rear-facing camera 	<ul style="list-style-type: none"> • Linux kernel 2.6.35 • Enhancements for gaming including performance improvements, new sensors, graphics, audio and power management routines • Rich multimedia support such as mixable audio effects • Significant upgrades and enhancements in the Dalvik runtime and supporting libraries • Support for 57 languages/locales 	<ul style="list-style-type: none"> • Same apps as Android 2.2 • Downloads • Search • Speech Recorder

Table 3.8 Android 2.3.3 Features and Highlights

New User Features	New Developer Features, APIs, and Technologies	Built-in Applications
<ul style="list-style-type: none"> Same as Android 2.3 	<ul style="list-style-type: none"> Improved and extended support for near-field communications (NFCs) Tweaks to Bluetooth, graphics, media framework, and speech recognition Support for 57 languages/locales 	<ul style="list-style-type: none"> Same apps as Android 2.3

Android 2.3.3

Android 2.3.3, released February 2011, highlighted the features and updates found in [Table 3.8](#).

SOFTWARE DEVELOPMENT KIT (SDK)

The Android software development kit (SDK) is the development resource needed to develop Android applications. It includes software libraries and APIs, reference materials, an emulator, and other tools. The SDK is supported in many environments including Linux, Windows, and OS X and can be downloaded free from <http://developer.android.com>.

The SDK is also a powerful forensic tool used by analysts in many situations to aid in the investigation of an Android device.

SDK Release History

While the Android platforms mark the officially supported releases of Android, the SDK is updated more frequently. [Table 3.9](#) provides the complete SDK release history that can aid in these situations ([SDK Archives, n.d.](#)).

Table 3.9 Archived Android Platforms Releases

Platform	API Level	Release Date
Android 1.6 r1	4	September 2009
Android 1.5 r3	3	July 2009
Android 1.1 r1	2	February 2009
Android 1.0 r2	1	November 2008

SDK Install

Since the SDK is critical in the investigation of an Android device, examiners should have a working installation. The following sections provide step-by-step directions for installing the SDK on the supported platforms.

Linux SDK Install

These steps are based on the Ubuntu VM used to download and compile the Android Open Source Project (AOSP) from Chapter 1 which already includes most of the prerequisites including the Java development kit. From a terminal window, install the needed 32-bit libraries:

NOTE

32-Bit libraries

Since the Ubuntu VM built in Chapter 1 used the 64-bit version of Ubuntu, we must install the 32-bit libraries to install the SDK. If, however, you are using a 32-bit Linux workstation, you need not complete this step. While the 32-bit workstation can run the SDK, it cannot build the AOSP after version 2.2.

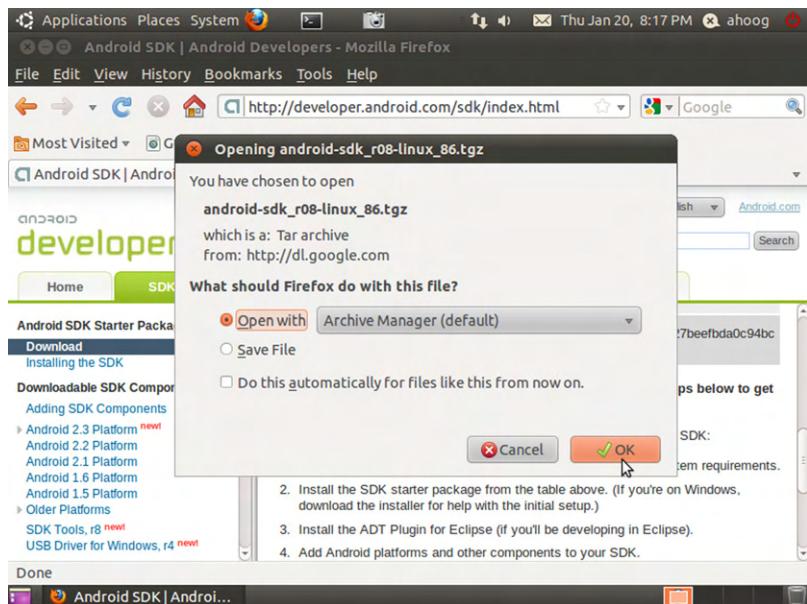


FIGURE 3.3

Download Android SDK for Linux.

```
#install 32-bit libraries  
sudo apt-get install ia32-libs
```

Next, start Firefox and navigate to <http://developer.android.com/sdk> and download the Linux i386 platform (android-sdk_r08-linux_86.tgz, as of January 2011). The default action will open the archive in the archive manager as shown in Fig. 3.3.

Then right click and extract the archive to your home directory as shown in Fig. 3.4.

Next, from the terminal window:

```
#navigate to the tools/ directory in the Android SDK  
cd ~/android-sdk-linux_x86/tools  
  
#run android  
../android
```

This will run the Android SDK and Android Virtual Device (AVD) manager, which will allow you to download and manage the additional necessary components as shown in Fig. 3.5.

To fully leverage the Android SDK, additional components are required. Minimally, we want to install the platform's specific SDK tools and at least one SDK

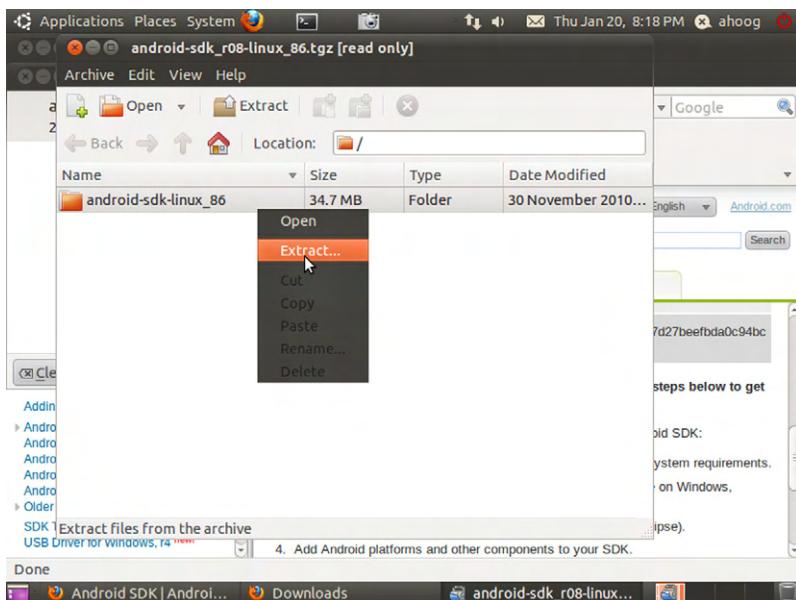
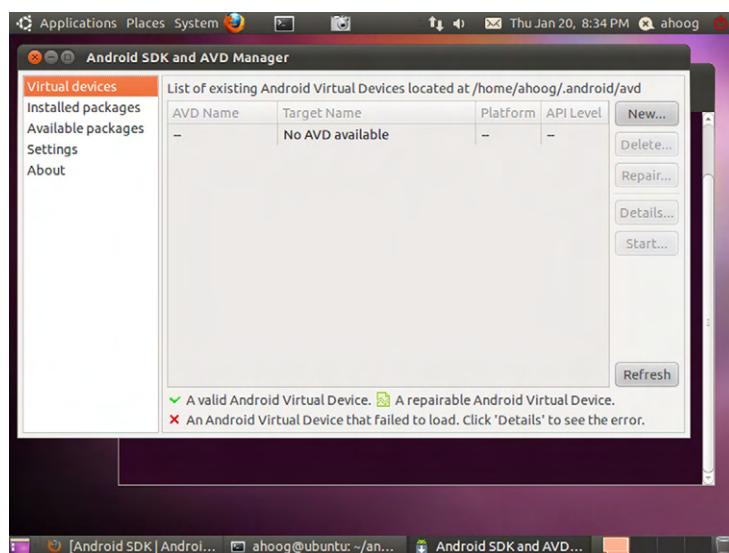


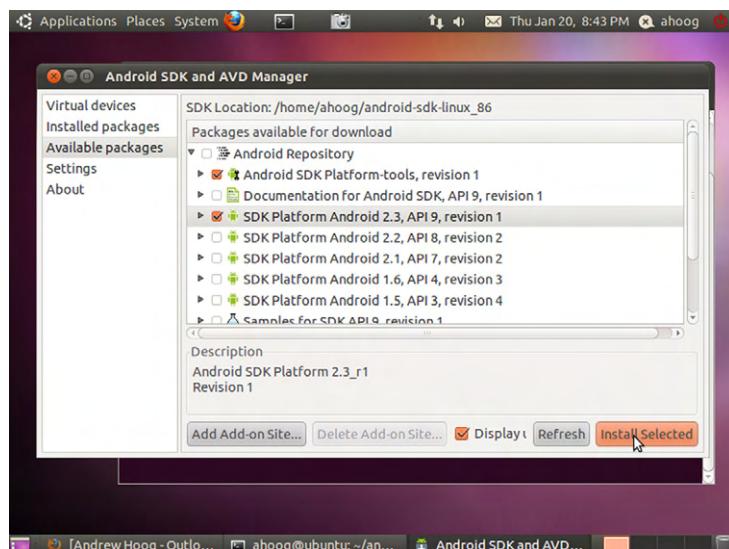
FIGURE 3.4

Extract Android SDK for Linux.

**FIGURE 3.5**

Android SDK and AVD manager in Linux.

platform (in this case, Android 2.3) so that we can run the emulator. To complete the installation, select the Available packages from the left navigation pane and then the two additional packages as shown in Fig. 3.6.

**FIGURE 3.6**

Select additional Android SDK packages.

And then choose Install Selected. You will be prompted to approve the license for all packages as shown in Fig. 3.7.

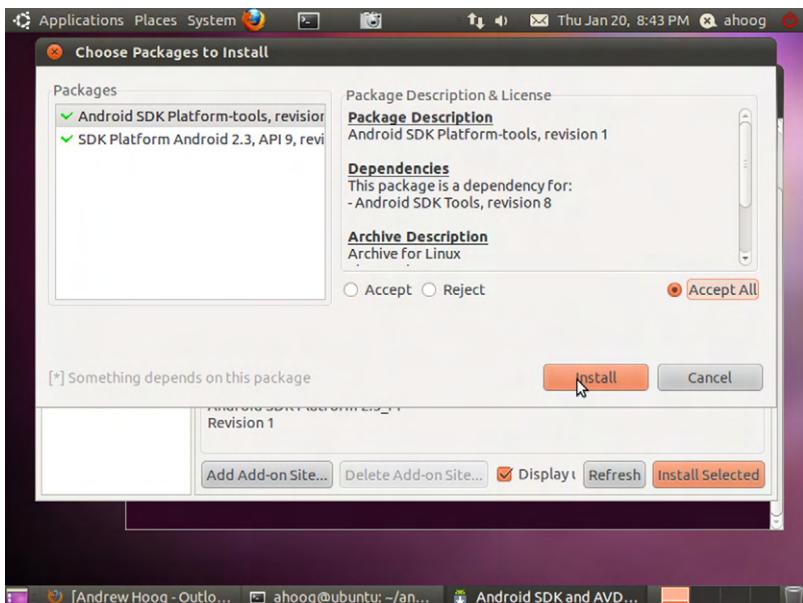


FIGURE 3.7

Accept and install Android SDK packages.

Select Accept All (provided you agree) and then install. The Android SDK and AVD manager will then download and install the components.

Optionally, you may want to add the binary directories to your operating system (OS) execution path so you do not have to specify the full path to the programs each time. In Linux, do the following:

```
# open your .bashrc in an editor
nano -w ~/.bashrc

#add the following line, substituting your login name
export PATH=$PATH:/home/ahoog/android-sdk-linux_86/tools/
export PATH=$PATH:/home/ahoog/android-sdk-linux_86/platform-tools/
```

Save, exit, and then re-open (Ctrl-O) a new shell.

One final step you must take in Ubuntu is to create USB profiles for each Android device manufacturer in the system's configuration, specifically the udev rules. From a terminal session as root, edit/create the udev rule:

```
sudo nano -w /etc/udev/rules.d/51-android.rules
```

Copy the following contents (vendor IDs are supplied on <http://developer.android.com/guide/developing/device.html#VendorIds>):

```
#Acer
SUBSYSTEM=="usb", SYSFS{idVendor}=="502", MODE="0666"
#Dell
SUBSYSTEM=="usb", SYSFS{idVendor}=="413c", MODE="0666"
#Foxconn
SUBSYSTEM=="usb", SYSFS{idVendor}=="489", MODE="0666"
#Garmin-Asus
SUBSYSTEM=="usb", SYSFS{idVendor}=="091e", MODE="0666"
#HTC
SUBSYSTEM=="usb", SYSFS{idVendor}=="0bb4", MODE="0666"
#Huawei
SUBSYSTEM=="usb", SYSFS{idVendor}=="12d1", MODE="0666"
#Kyocera
SUBSYSTEM=="usb", SYSFS{idVendor}=="482", MODE="0666"
#LG
SUBSYSTEM=="usb", SYSFS{idVendor}=="1004", MODE="0666"
#Motorola
SUBSYSTEM=="usb", SYSFS{idVendor}=="22b8", MODE="0666"
#Nvidia
SUBSYSTEM=="usb", SYSFS{idVendor}=="955", MODE="0666"
#Pantech
SUBSYSTEM=="usb", SYSFS{idVendor}=="10a9", MODE="0666"
#Samsung
SUBSYSTEM=="usb", SYSFS{idVendor}=="40000000", MODE="0666"
#Sharp
SUBSYSTEM=="usb", SYSFS{idVendor}=="04dd", MODE="0666"
#Sony Ericsson
SUBSYSTEM=="usb", SYSFS{idVendor}=="0fce", MODE="0666"
#ZTE
SUBSYSTEM=="usb", SYSFS{idVendor}=="19d2", MODE="0666"
```

And then save the file. Finally, make the file readable by all users:

```
sudo chmod a+r /etc/udev/rules.d/51-android.rules
```

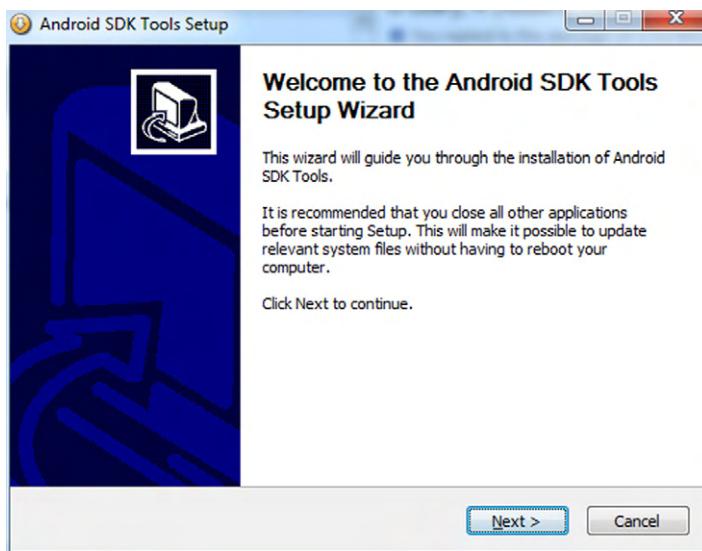
You can either restart the udev daemon or simply reboot.

Windows SDK Install

The latest version of the Android SDK for Windows, shown in Fig. 3.8, is now packaged as an executable installer, which will determine if you have the necessary Java dependencies properly installed and, if not, will download and install them for you. However, the installer will only detect the 32-bit install of the JDK and will not automatically install the JDK on a Windows 7 64-bit install. If you are running a 32-bit version of Windows (such as Windows XP), then the installer may be a good option and you can simply download the package from <http://developer.android.com/sdk/index.html> and run the installer.

However, many analysts and engineers have moved to 64-bit OSs. To install the Android SDK on Windows, first install the Java SE SDK by downloading it at <http://java.sun.com/javase/downloads/>. Make sure you install the full SDK.

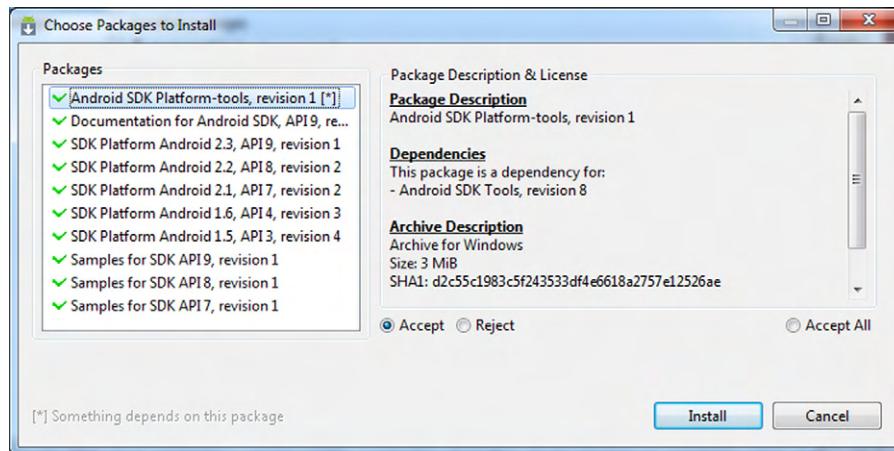
After the SDK is installed, download the zipped version of the Window's Android SDK at <http://developer.android.com/sdk/index.html> and extract it to your

**FIGURE 3.8**

Android SDK installer for Windows.

hard drive. For our example, we will extract directly to C:\ that will then create the folder C:\android-sdk-windows.

Open that directory and double click SDK Manager.exe to begin the update process. Be sure that you select at least the Android SDK Platform-tools, as in Fig. 3.9, and one release platform (2.3 in this example).

**FIGURE 3.9**

Android SDK manager for Windows.

When working with Android devices in Windows, you need to specify USB drivers. The Android SDK recently updated how the USB drivers are installed. First, make sure you are running the SDK Manager and select Available packages. Expand Third party Add-ons → Google Inc. add-ons and finally choose Google Usb Driver package as shown in Fig. 3.10.

Then, accept the license and install as shown in Fig. 3.11.

After the USB drivers are installed, you should have all the necessary components. However, to simplify running tools from the Android SDK, you should update

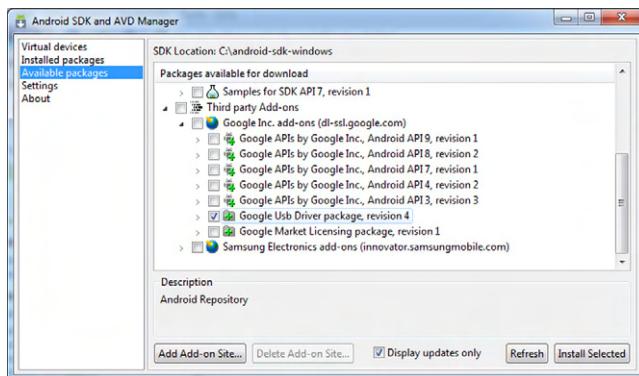


FIGURE 3.10

Google USB driver package for Windows.

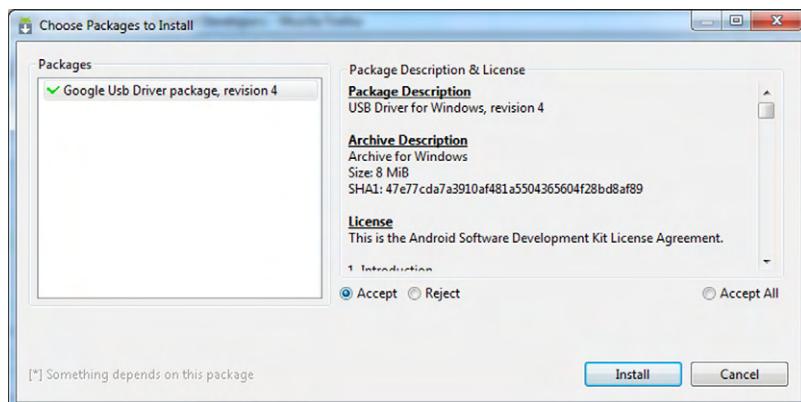
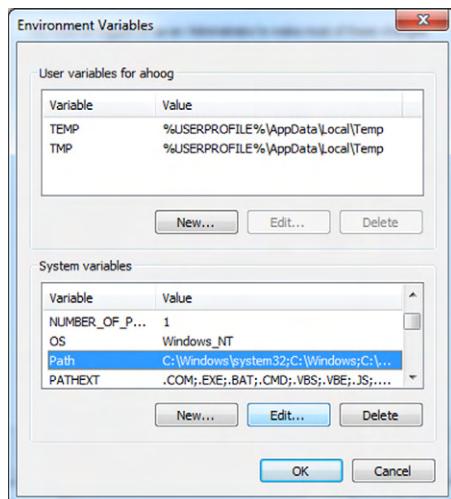


FIGURE 3.11

Accept and install license.

**FIGURE 3.12**

Update PATH environment variable (Windows 7 64 bit).

your workstation's environment variables, specifically the PATH to executable files. To do this, go to your Control Panel and open the System application. You should then select the tab where you can update the Environment variable, whose location will vary depending on your exact Windows version, shown in Fig. 3.12. Finally, locate the Path system variable, select Edit, and append the full path to your Android SDK platform-tools directory, which in our example would be ;C:\android-sdk-windows\platform-tools.

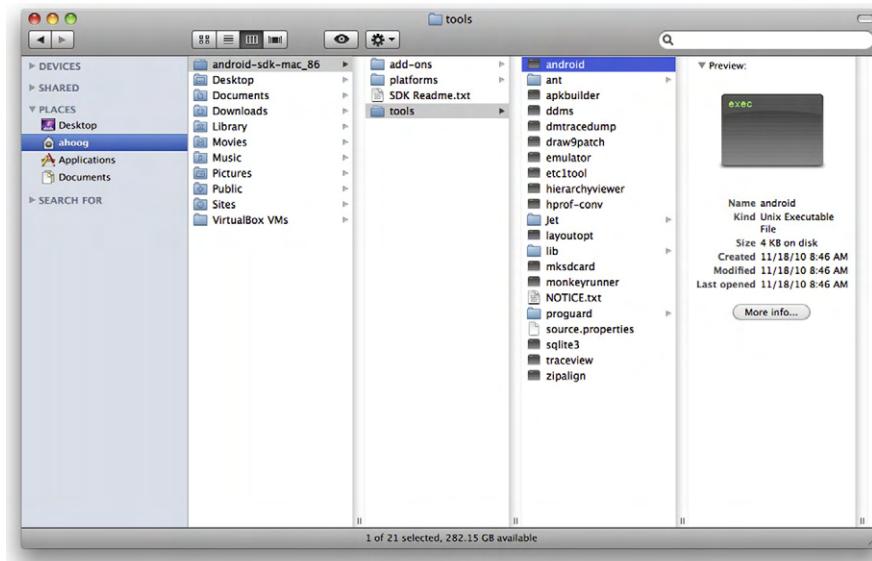
The “;” is important, as it is the delimiter between path locations. Once you complete this update, make sure you exit and wait for command prompts indicating that the new setting has taken effect.

OS X SDK

To install the Android SDK on OS X, first download the archive from <http://developer.android.com/sdk/index.html>, from which OS X will then automatically extract.

Navigate to the tools subdirectory as shown in Fig. 3.13, and then double click Android to run the Android SDK and AVD manager as shown in Fig. 3.14.

When the Manager runs, select Available packages, expand Android Repository and then select the Android SDK platform-tools and at least one Android platform as shown in Fig. 3.15.

**FIGURE 3.13**

Extracted Android SDK for OS X.

Then accept the licenses and complete the install. Finally, to simplify running tools from the Android SDK, you should update your executable PATH. On OS X 10.6, run Terminal (Applications → Utilities) and do the following:

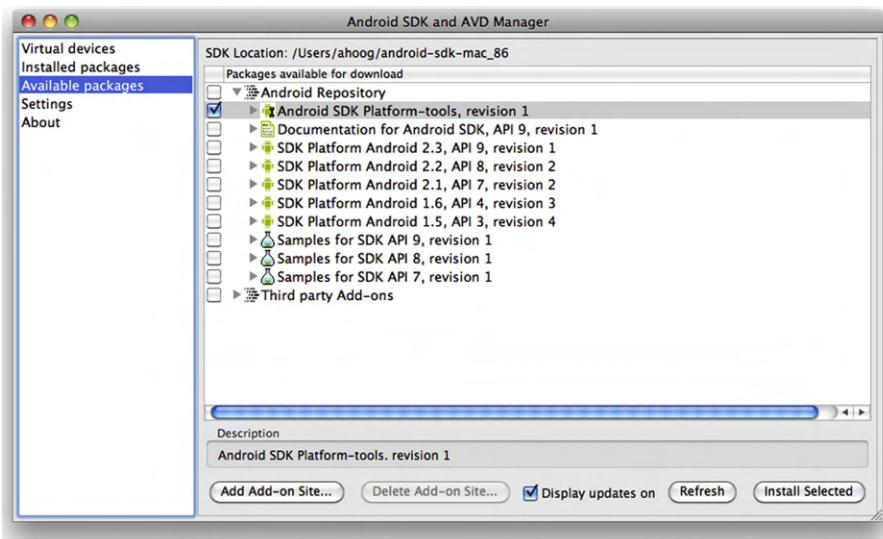
```
#edit your bash_profile
nano -w ~/.bash_profile

#add the following line substituting your full path to the platform-tools
#directory PATH=$PATH:/Users/ahoog/android-sdk-mac_86/platform-tools

#save with Ctrl-O and then Ctrl-X to exit. Exit Terminal
exit
```

**FIGURE 3.14**

Open Android on OS X.

**FIGURE 3.15**

Install Android SDK components on OS X.

Make sure you fully exit the Terminal app and then restart. From the terminal, type:

```
echo $PATH
```

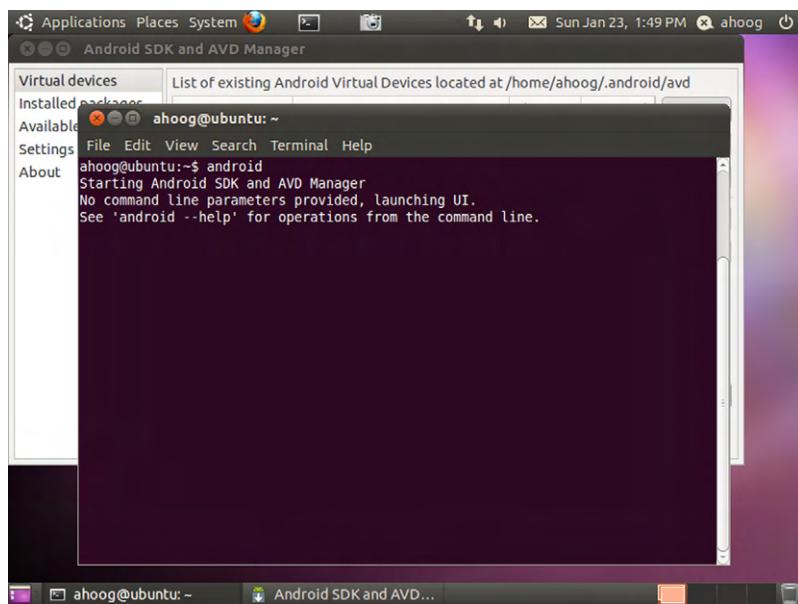
This should return your executable path with the platform-tools appended.

Android Virtual Devices (Emulator)

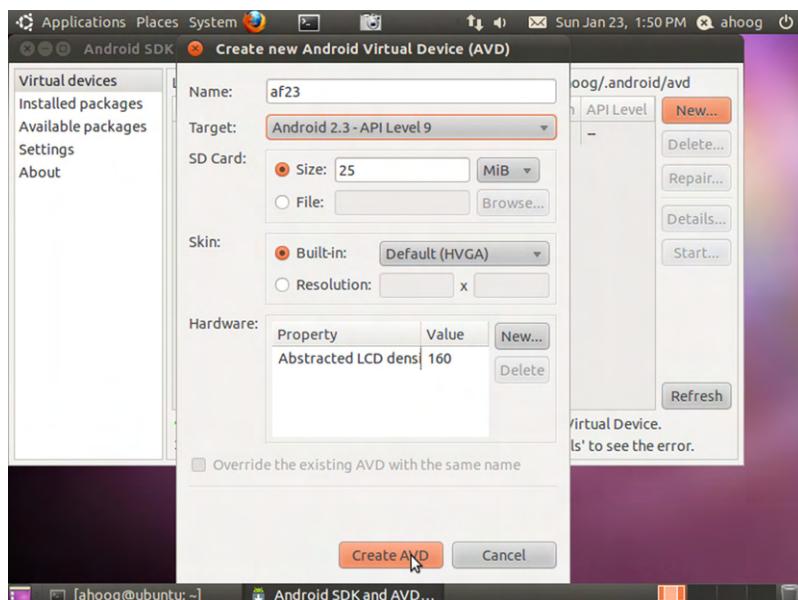
Once you have the Android SDK installed on your workstation and have at least one release platform downloaded, you are ready to create an AVD, a virtual mobile device, or emulator, which runs on your computer. The emulator is especially helpful for developers for creating custom applications. However, there is great value for the forensic analyst and security engineer because you can profile how applications execute on a device. This could be important to validate your findings in an investigation, or to test how a forensic tool affects an Android device.

The emulator takes considerable resources, so an ideal workstation would have a newer sufficient CPU and RAM. A bit of patience from the examiner may also be required. To create an AVD, first run the Android SDK and AVD manager application as seen in Fig. 3.16. If you updated your OS's path to include the tools directory in the SDK, you should be able to run Android from a shell, terminal, or command prompt.

In the left pane, select Virtual devices and then select New, as in Fig. 3.17.

**FIGURE 3.16**

Start Android SDK and AVD manager.

**FIGURE 3.17**

Creating a new AVD.

Make sure you populate the following fields:

- Name: Provide a name for the virtual device, for example, af23 (Android Forensics 2.3).
- Target: Select the target platform, in this case Android 2.3—API level 9.
- [optional] SD card: Optionally create an SD card for the virtual device.

You can set additional properties. However, for now we will create the most basic AVD. Also, if you encounter an Android device running on an older platform, you can create virtual devices running the older version by simply downloading the Android platform using the Android SDK and AVD manager. When you click Create AVD, the device will be created and you will receive a confirmation screen similar to that shown in Fig. 3.18.

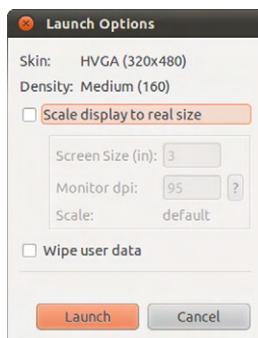
Ensure that the new AVD is highlighted and then click Start, at which point you will be prompted for launch options as shown in Fig. 3.19.

Select any options you wish and click Launch. At this point, the AVD will begin the boot process, which could take a few minutes or longer. During that time, you will see Android starting up. This is illustrated in Fig. 3.20.

Finally, you will be presented with the fully functioning AVD as shown in Fig. 3.21.

**FIGURE 3.18**

AVD-created confirmation.

**FIGURE 3.19**

AVD launch options.

**FIGURE 3.20**

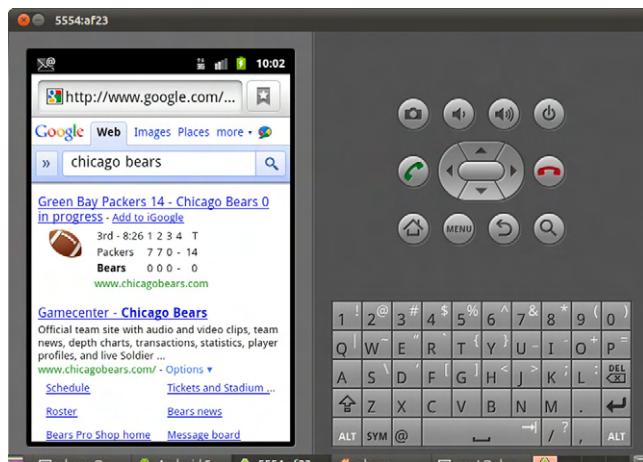
AVD launching.

**FIGURE 3.21**

Running AVD.

The AVD is very powerful and fully functional. For example, you can easily jump online, as demonstrated in Fig. 3.22, and surf the web site. You can configure e-mail accounts, send test SMS messages to other AVD and of course, if you are a developer, deploy and test your application.

When an AVD is created and then launched, the data created are valuable for forensic and security research. The files are created in your home directory, which

**FIGURE 3.22**

AVD running browser.

Table 3.10 AVD Storage Directory

Workstation Operating System	AVD Storage Directory	Example
Ubuntu Linux	/home/<username>/.android	/home/ahoog/.android
Mac OS X	/Users/<username>/.android	/Users/ahoog/.android
Windows 7	C:\Users\<username>\.android	C:\Users\ahoog\.android

varies by platform, in a folder called .android (note the dot prefix in the filename). Table 3.10 provides specific OS paths.

Inside AVD's .android directory you will find configuration and data files needed to run the AVD.

```
ahoog@ubuntu:~/android$ tree
.
├── androidtool.cfg
└── avd
    ├── af23.avd
    │   ├── cache.img
    │   ├── config.ini
    │   ├── emulator-user.ini
    │   ├── sdcard.img
    │   ├── userdata.img
    │   └── userdata-qemu.img
    └── af23.ini
├── default.keyset
└── modem-nv-ram-5554
    └── repositories.cfg

2 directories, 11 files
```

Files of particular forensic and security interest include the following:

- cache.img: disk image of /cache partition
- sdcard.img: disk image of SD card (if created during AVD setup)
- userdata-qemu.img: disk image of /data partition

The cache.img and userdata-qemu.img are YAFFS2 file systems that are not supported by current forensic software and will be covered in Chapter 4. However, standard forensic tools will work quite well on sdcard.img, which is a FAT32 file system.

```
ahoog@ubuntu:~/android/avd/af23.avd$ file sdcard.img
sdcard.img: x86 boot sector, code offset 0x5a, OEM-ID "MSWIN4.1", Media
descriptor 0xf8,
sectors 51200 (volumes > 32 MB), FAT (32 bit), sectors/FAT 397, reserved3
0x800000,
serial number 0x1d0e0817, label: "      SDCARD"
```

Forensic analysts and security engineers can learn a great deal about Android and how it operates by leveraging the emulator and examining the network, file system, and data artifacts.

Android OS Architecture

It is important to understand the high-level architecture of Android, especially for security procedures and moving beyond logical forensic analysis.

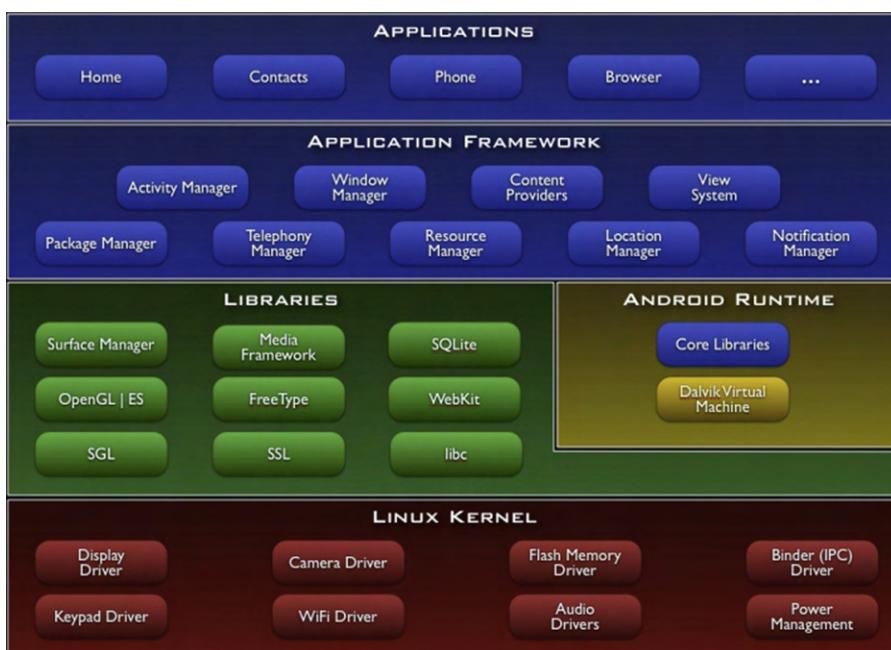
Android is based on the Linux 2.6 kernel that provides the fundamental software needed to boot and manage both the hardware and Android applications. While the functionality that the kernel provides is quite extensive, we will focus on core areas highlighted in Fig. 3.23.

As illustrated in Fig. 3.23, low-level functions include power management, Wi-Fi.com, display, audio drivers, and more. Perhaps most important from a forensics perspective is the flash memory driver, which will be explored in detail in Chapter 4.

After the kernel, a set of libraries are available, which provide core functionality needed by developers and device owners alike. These include the WebKit library for rendering HTML in both the bundled browser and third-party apps. Other libraries handle fonts, displays, various media, and secure communications using Secure Socket Layers (SSLs). Finally, the SQLite library provides a method for structured data storage on Android and is an area forensic analysts and security engineers will focus on.

The core libraries are then bundled with a custom Java virtual machine (VM) to provide the Android runtime environment, which is where applications run.

Finally, the SDK provides access to these resources via APIs and an application framework. The framework is the primary layer that third-party developers interact with and it provides them abstract access to key resources needed for their application. As we explore logical forensic techniques, an important aspect of the application framework—content providers—will be explained in more detail because they provide the primary mechanism by which we can extract data from an Android device.

**FIGURE 3.23**

Android architecture.

Dalvik VM

The Dalvik Virtual Machine (Dalvik VM) was developed by Google to create an efficient and secure mobile application environment.

To achieve the desired security, each application is run on its own Dalvik VM. As such, the Dalvik VM was written so that many VMs could run at once on an Android device. The Dalvik VM relies heavily on the Linux OS to provide low-level functions such as access to core libraries and hardware, threat and security management, memory management, and more.

To achieve efficiency, applications that run in a Dalvik VM have a special format called a Dalvik Executable (.dex) file. Developers write and compile their programs with Sun's Java Development Kit and the resulting byte code is then transformed into a .dex file which provides efficient storage and is optimized for execution in the Dalvik VM. An interesting project developed by JesusFreke, an accomplished and well-known Android hacker, is called smali/baksmali. This project allows a user to decompile a .dex file to determine what an application does ([smali, n.d.](#)).

Dalvik is a unique aspect of Android and a critical component in the forensic and security analysis of a device.

Native Code Development

While most Android applications are written in Java using the SDK, Google provides a lower level development platform with their native development kit (NDK). The NDK was first released in June 2009 and has gone through five revisions, with the latest release in November 2010.

The NDK allows developers to write code in C/C++ and compile it directly for the CPU. While this adds complexity to the development process, some developers can benefit from this approach by reusing an existing code base in C/C++ or by implementing certain functions that can be optimized outside the Dalvik VM. The NDK does not allow developers to create full applications that run outside of the Dalvik VM; instead the C/C++ components are packaged inside the application's .apk file and are called by the application within the VM.

At this time, the NDK supports the ARMv5TE and ARMv7-A CPUs, and in the future will support Intel's x86 CPU architecture. When a developer writes code in one platform (e.g., Mac OS X) but compiles it for another CPU, the technique is referred to as cross-compiling an application. The NDK greatly simplifies this process and provides a set of libraries the developer can use.

From a forensics and security viewpoint, cross-compiling is an important component for research and development of new techniques and exploits. While most forensic analysts and security engineers do not need to compile code, understanding how the process works, and what role it plays in the process, is important. For example, the initial Android 1.5 root exploit targeted a Linux kernel bug (CVE-2009-2692) to gain privileges. The initial code was distributed as source code and required cross-compiling. One significant advantage to this approach is that an examiner can describe in exact detail how the device was exploited and, if necessary, provide the source code.

As Android matures, expect to see additional developments in the NDK and natively compiled code.

ANDROID SECURITY MODEL

The Android platform implements security through a number of controls designed to protect the user.

When an application is first installed, Android checks the .apk file to ensure it has a valid digital signature to identify the developer. Unlike SSL, the digital certification does not need to be signed by a Certificate Authority. However, the developer must keep the key safe; otherwise someone could sign a malicious application and distribute it as that developer. For example, if a financial institution's digital signature was compromised, a malicious developer could publish an update to the banking application, which steals critical data.

After the .apk file is validated, Android checks the special file created by the developer that specifies, among other items, what access an application needs to the system. For example, an application may request access to the user's contacts, SMS messages, and the network/Internet. If this application adds functionality to the

SMS system, these permissions seem reasonable. If, however, the application simply changes your background images, then a user should question the permission and can choose not to install the application. In practice, users quickly allow all permissions and application requests, and thus may allow a malicious application to install.

After an application has been verified and the user granted the requested permissions, the application can now install on the system. A key part of the Android security model is that each application is assigned a unique Linux user and group ID and runs in its own process and Dalvik VM. During the installation, the system creates a specific directory on the device to store the application's data and only allows that application to access the data leveraging the Linux user ID and group ID permissions. In addition, the application's Dalvik VM is run in its own process as the specific user ID. These key mechanisms enforce data security at the OS level as applications do not share memory, permissions, or disk storage. Applications can only access the memory and data within their Dalvik VM.

Of course, there are a few exceptions to this process. First, a developer can sign more than one application with the same digital certification and specify that it can share the same user ID, process, memory, and data storage as one of their other applications. This situation is exceptional and is most commonly used when a developer has both a free and a paid version. If a user upgrades to the paid version, they can leverage the data accumulated while using the free version and thus no data are lost.

Also, most Android users have the option to allow apps to be installed from non-Market locations and to skip the digital signature check. This option can be accessed from the Applications menu in the device's Settings and, when selected, displays a warning to the user as shown in Fig. 3.24.



FIGURE 3.24

Android setting to allow apps installs from unknown sources.

The most common situation is that users could now install apps from web sites by directly downloading an .apk file. The install process also skips the digital signature check. A recent AT&T phone (Motorola Backflip) removed this option from Android upsetting many users ([Android On Lockdown, n.d.](#)). However, a work-around using the Android SDK does exist and will be discussed in Chapter 6.

As a result of the security architecture built into Android, forensic examiners do not have a simple way to extract core user data from a device. Barring exploits, the security architecture is effective in isolating and protecting data between applications.

FORENSICS AND THE SDK

So how is the SDK important in forensics? The SDK not only provides a set of tools and drivers enabling the analysis of Android devices but is also useful for application profiling and other forensic research.

Connecting an Android Device to a Workstation

It is important to note how an Android device actually connects to a VM. Android devices, to date, have a physical USB interface that allows them to connect, share data and resources, and typically to recharge from a computer or workstation. If you are only running a single OS, the USB device should be detected and accessible. However, additional configuration or drivers may be required. If you are running a VM though, you simply want the host OS to pass the connection through to the VM. For example, if your host OS is OS X and you are running VMWare fusion, you select the menus Virtual Machine → USB and then Connect the device (High Android Phone in this case), as shown in [Fig. 3.25](#).

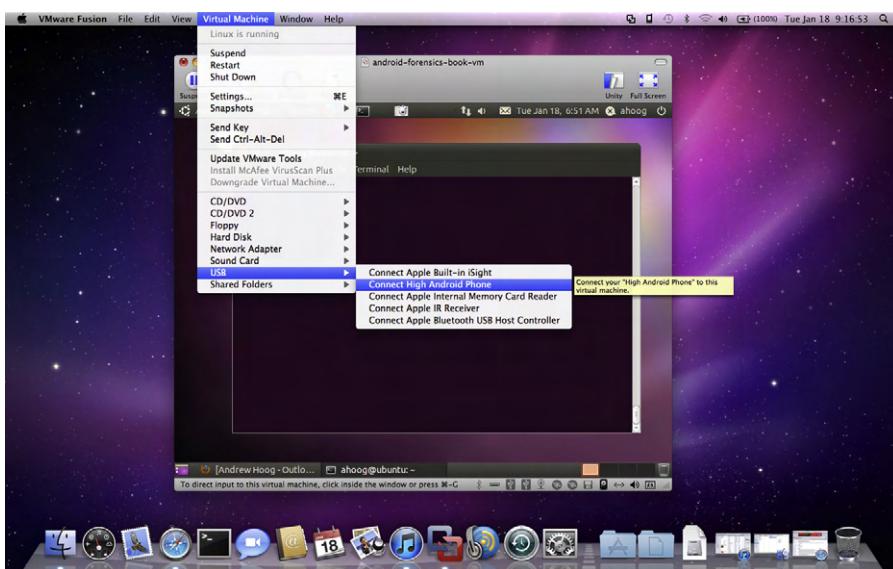
Similarly, when your host OS is Linux, and you are running the VM using Oracle's VirtualBox, you must first ensure that you are a member of the `usbusers` group. So, from a terminal session, execute the following:

```
#create usbusers group  
sudo addgroup usbusers  
  
#Add your username to the userusers group:  
sudo usermod -a -G usbusers ahoog
```

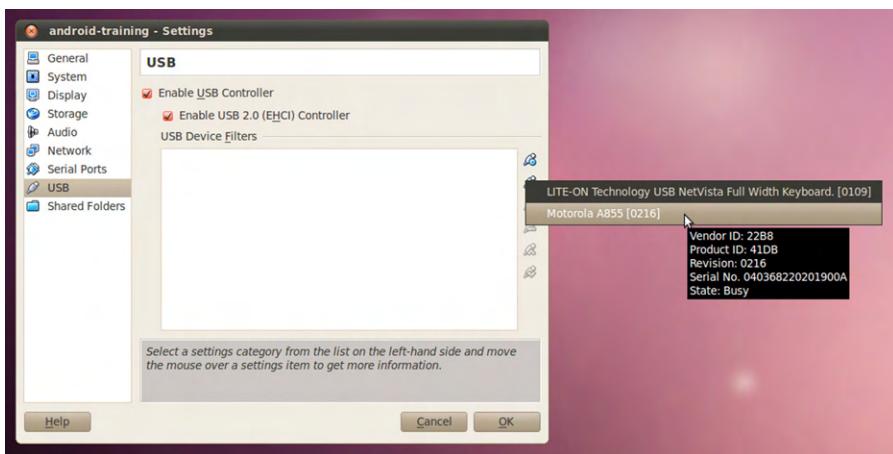
Next, you go into the VM's Settings and add a USB Filter for the device, as shown in [Fig. 3.26](#).

Finally, you can connect the USB device as shown in [Fig. 3.27](#).

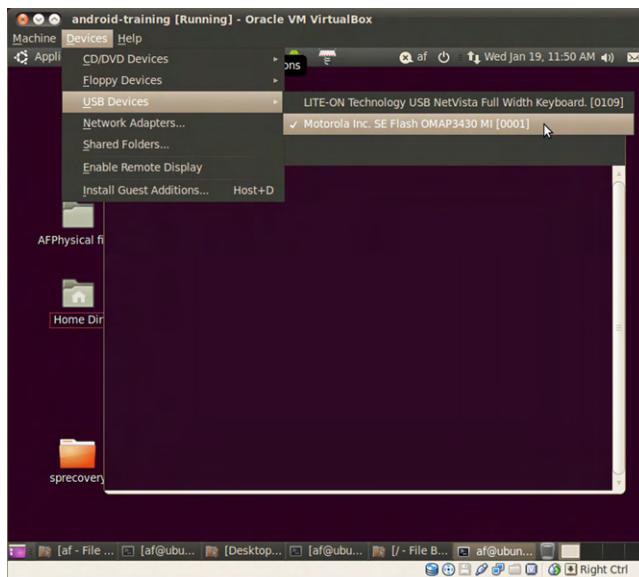
Finally, here are the steps if you are running the VM headless (VirtualBox 3.2.10 as outlined in Chapter 1). First, you need to install VBox Additions, which will

**FIGURE 3.25**

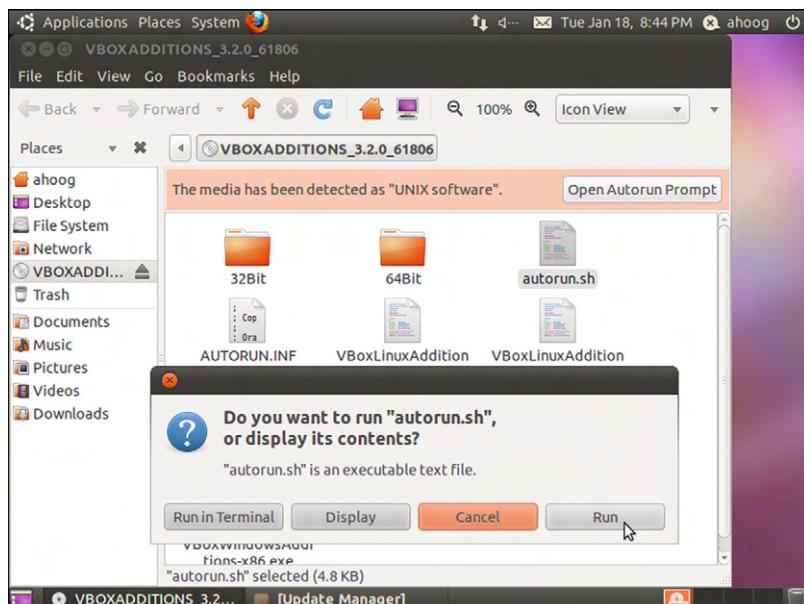
Connect USB device to Ubuntu VM in VMWare Fusion.

**FIGURE 3.26**

Adding USB filter on Linux host running Oracle's VirtualBox.

**FIGURE 3.27**

Connecting USB device on Linux host running Oracle's VirtualBox.

**FIGURE 3.28**

Install VBox additions over on Ubuntu VM remote desktop protocol.

enable shared folder, better video, USB support (if you downloaded/bought the PUEL edition), and other features. From the host workstation:

```
wget
http://download.virtualbox.org/virtualbox/3.2.0/VBoxGuestAdditions_3.2.0.iso

VBoxManage registerimage dvd ~/VBoxGuestAdditions_3.2.0.iso

VBoxManage storageattach af-book-vm --storagectl "IDE Controller" --port 1
--device 0 \
--type dvddrive --medium ~/VBoxGuestAdditions_3.2.0.iso
```

The DVD should now be available on the Ubuntu VM. Remote desktop into the VM again (see Chapter 1 for necessary steps) and double click VBOX-ADDITIONS_3.2.0_61806 DVD on your desktop to open the DVD. Then double click autorun.sh and select the Run option. You will be prompted for your password after which the install will proceed. [Figure 3.28](#) illustrates this step.

Now that you have VBox Additions installed, you can connect USB devices to your guest OS. But first, you must shutdown the VM. Then, follow these steps:

```
#create usbusers group
sudo addgroup usbusers

#Add your username to the userusers group:
sudo usermod -a -G usbusers ahoog

#Determine attached USB device info
VBoxManage list usbhost

Oracle VM VirtualBox Command Line Management Interface Version 3.2.8
(C) 2005-2010 Oracle Corporation
All rights reserved.

Host USB Devices:

UUID:          b1c23004-db71-49ec-b5cb-348e2038b409
VendorId:      0x0781 (0781)
ProductId:     0x554f (554F)
Revision:      2.0 (0200)
Manufacturer:  Best Buy
Product:       Geek Squad
SerialNumber:  153563119AC07CAD
Address:        sysfs:/sys/devices/pci0000:00/0000:00:1d.0/usb2/2-1/
2-1.5//device:/dev/bus/usb/002/004
Current State: Busy

#Create the USB filter to connect the device
VBoxManage usbfilter add 0 --target af-book-vm --vendorid 0781
--productid 554F \
--name "Geek Squad" --active yes

#Ensure USB is enabled
VBoxManage modifyvm Win2003SvrR2 --usb on

#Power on the guest (again recommended from inside screen)
VBoxHeadless -startvm af-book-vm -p 3392 &
```

Using this example, the USB device should now be passed through to the VM.

USB Interfaces

While you connect an Android device to your workstation or VM through a single USB port, the hardware and Android itself generally expose more than one virtual USB interface. For example, when you connect the HTC Incredible over USB, you are presented with a menu of four options:

1. Charge only—Charge phone over USB
2. HTC Sync—Sync contacts and calendar
3. Disk drive—Mount as disk drive
4. Mobile Broadband Connect—Smart phone's mobile networks with PC

The default selection, shown in Fig. 3.29, is the Charge only option. Both HTC Sync and Mobile Broadband Connect options are custom options and programs HTC and, at times, the wireless carrier support for the device.

CD-ROM Interface

The disk drive option is more universally used. This option connects the Android device to the workstation as a disk drive. This is one key area where the device exposes multiple USB devices to the workstation. When you first plug HTC

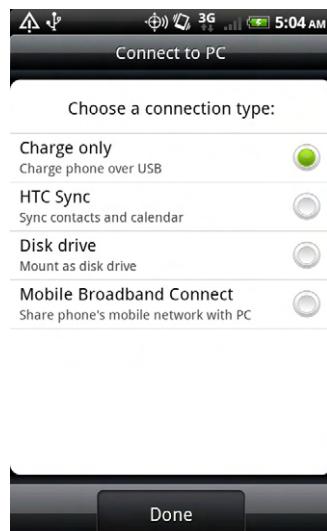


FIGURE 3.29

HTC Incredible connect to PC options.

Incredible into the computer, it actually registers three separate types of drives: one CD-ROM and two USB mass storage devices. The following listing is taken from the Linux workstation's kernel messages with the dmesg command:

```
[ 210.336135] usb 1-1: new high speed USB device using ehci_hcd and address 3
[ 210.646221] scsi4 : usb-storage 1-1:1.0
[ 211.649296] scsi 4:0:0:0: Direct-Access      HTC      Android Phone 0100
PQ: 0 ANSI: 2
[ 211.652056] scsi 4:0:0:1: Direct-Access      HTC      Android Phone 0100
PQ: 0 ANSI: 2
[ 211.654291] scsi 4:0:0:2: CD-ROM            HTC      Android Phone 0100
PQ: 0 ANSI: 2
[ 211.657317] sd 4:0:0:0: Attached scsi generic sg2 type 0
[ 211.658364] sd 4:0:0:1: Attached scsi generic sg3 type 0
[ 211.661956] sr1: scsi3-mmc drive: 0x/0x caddy
[ 211.662569] sr 4:0:0:2: Attached scsi CD-ROM sr1
[ 211.662755] sr 4:0:0:2: Attached scsi generic sg4 type 5
[ 211.678409] sd 4:0:0:0: [sdः] Attached SCSI removable disk
[ 211.686339] sd 4:0:0:1: [sdः] Attached SCSI removable disk
```

As you can see, two Direct-Access drives are found at 4:0:0:0 and 4:0:0:1, and a CD-ROM is found at 4:0:0:2. The CD-ROM contains custom programs and drivers that HTC bundles with the device to enable the syncing and broadband connect features. Obviously, there is no physical CD-ROM. However, a portion of the device's storage is dedicated to the CD-ROM and is formatted as an ISO9660. The host OS can then mount the drive as a CD-ROM and, in Windows, would potentially even support the auto-run feature. Leveraging TSK's fsstat program, we can see more details about the partition:

```
ahoog@ubuntu:~$ sudo fsstat /dev/sr2

==== PRIMARY VOLUME DESCRIPTOR 1 ====
FILE SYSTEM INFORMATION
-----
File System Type: ISO9660
Volume Name: Verizon Mobile
Volume Set Size: 1
Volume Set Sequence: 1
Publisher: Publisher
Data Preparer: Publisher
Recording Application: Application
Copyright:

METADATA INFORMATION
-----
Path Table Location: 23-23
Inode Range: 0 - 9
Root Directory Block: 26

CONTENT INFORMATION
-----
Sector Size: 2048
Block Size: 2048
Total Sector Range: 0 - 2383
Total Block Range: 0 - 2383
```

```
==== SUPPLEMENTARY VOLUME DESCRIPTOR 1 ====
FILE SYSTEM INFORMATION
-----
File System Type: ISO9660
Volume Name:
Volume Set Size: 1
Volume Set Sequence: 1
Publisher:
Data Preparer: Publisher
Recording Application:
Copyright:

METADATA INFORMATION
-----
Path Table Location: 25-25
Root Directory Block: 29
Joliet Name Encoding: UCS-2 Level 1

CONTENT INFORMATION
-----
Sector Size: 2048
Block Size: 2048
Total Sector Range: 0 - 2383
Total Block Range: 0 - 2383
```

As you can tell from the Volume Name, the CD-ROM contains software provided by Verizon to use the additional features of the device.

SD Cards (Removable and Virtual)

Far more important from a forensic standpoint are the SD card(s) available through the device. Placing user's files, especially larger files such as multimedia, is a key strategy in Android. Most Android devices have a removable media slot, which accepts a micro-SD card. The core application data remain on the device (under /data/data), but the files that are likely important in an investigation may also exist on the SD card.

In the previous section, when an Android device was connected via USB, the Linux workstation's kernel messages displayed the various USB devices available. The two SCSI removable disks that were listed, sdb and sdc, represent the SD cards on an HTC Incredible. If you choose the "Mount as disk drive" option under Connect to PC, the following additional messages show up on the kernel messages:

```
[ 325.669335] sd 4:0:0:1: [sdc] 3911680 512-byte logical blocks: (2.00 GB/
1.86 GiB)
[ 325.672039] sd 4:0:0:1: [sdc] Assuming drive cache: write through
[ 325.678282] sd 4:0:0:1: [sdc] Assuming drive cache: write through
[ 325.678294]   sdc: sdc1
[ 327.671951] sd 4:0:0:0: [sdb] 13844464 512-byte logical blocks: (7.08 GB/
6.60 GiB)
[ 327.674074] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 327.679387] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 327.679395]   sdb:
```

You will now see additional information about the SD card. The drive sdc has one partition, sdc1. And its size is 2 GB. We can see additional partition information by running TSK's mmcls:

```
ahoog@ubuntu:~$ sudo mmcls /dev/sdc
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

  Slot   Start       End       Length      Description
00: Meta  0000000000  0000000000  0000000001  Primary Table (#0)
01: ----  0000000000  0000000128  0000000129  Unallocated
02: 00:00  0000000129  0003911679  0003911551  DOS FAT16 (0x06)
```

As you will see, the SD card is formatted with a FAT16 file system, but often you will find FAT32 or you might encounter multiple file systems like FAT32 and native Linux file system ext3 and ext4.

More recently, devices also have an emulated or virtual SD card feature that uses the device's NAND flash to create a nonremovable SD card. This more closely models the iPhone where the user data partition is located directly on the NAND flash and cannot be removed. In the previous example, the sdb device provides access to the emulated SD card. Unlike the physical SD card, sdc does not have a partition table and the file system simply starts immediately. To see important information, run TSK's fsstat:

```
ahoog@ubuntu:~$ sudo fsstat /dev/sdb
FILE SYSTEM INFORMATION
-----
File System Type: FAT32

OEM Name: BSD 4.4
Volume ID: 0xc7f80810
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory):
File System Type Label: FAT32
Next Free Sector (FS Info): 562580
Free Sector Count (FS Info): 13376448

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 13844463
* Reserved: 0 - 31
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 2
* FAT 0: 32 - 1721
* FAT 1: 1722 - 3411
* Data Area: 3412 - 13844463
** Cluster Area: 3412 - 13844435
*** Root Directory: 3412 - 3475
** Non-clustered: 13844436 - 13844463
```

```
METADATA INFORMATION
-----
Range: 2 - 221456838
Root Directory: 2

CONTENT INFORMATION
-----
Sector Size: 512
Cluster Size: 32768
Total Cluster Range: 2 - 216267

FAT CONTENTS (in sectors)
-----
3412-3475 (64) -> EOF
3476-3539 (64) -> EOF
3540-5267 (1728) -> EOF
5268-7379 (2112) -> EOF
<snip>
```

In this particular case, the file system is in fact FAT32 and you will notice that while the volume has no Label, the OEM Name is set BSD 4.4.

WARNING

Auto-mounting USB devices

In the Ubuntu VM configuration section of Chapter 1, the auto-mount feature is disabled to prevent the OS from automatically detecting and mounting USB mass storage devices. Forensic analysts should take extreme precautions to prevent this from happening on a device being investigated. Beyond disabling auto-mount, devices should generally be connected through a USB write blocker.

In Ubuntu, if you do not have auto-mounting of USB devices disabled (which you should in nearly all situations), the SD cards are automatically mounted for you. If the device is attached to a hardware write blocker, mounted read-only, or in a situation where write blocking is not needed (e.g., research and development), you can run the df command in Linux to see where they were mounted:

```
ahoog@ubuntu:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdal       19G  3.4G   15G  19% /
none            369M 228K  369M  1% /dev
none            375M 252K  375M  1% /dev/shm
none            375M 100K  375M  1% /var/run
none            375M     0  375M  0% /var/lock
/host:/        931G 663G 269G 72% /mnt/hgfs
/dev/sdc1      1.9G 200M  1.7G 11% /media/E0FD-1813 (physical
                           2GB SD Card)
/dev/sdb        6.6G 227M  6.4G  4% /media/C7F8-0810 (emulated SD Card)
```

The physical SD card was mounted on /media/E0FD-1813 and the emulated SD card on /media/C7F8-0810.

On the Android device itself, the two SD cards are mounted as follows:

```
/dev/block/vold/179:9 /mnt/sdcard vfat  
rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0702,  
allow_utime=0020,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8,  
errors=remount-ro 0 0  
/dev/block/vold/179:3 /mnt/emmc vfat  
rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0702,  
allow_utime=0020,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8,  
errors=remount-ro 0 0
```

USB Debugging

One final, and very important, USB interface exposes the Android Debug Bridge (ADB) that allows a developer, forensic analyst, or security engineer to communicate and control an Android device over USB. By default, an AVD (running in the emulator) will have USB debugging enabled. However, non-emulator devices must explicitly enable USB debugging. To enable, select Applications → Development from the devices Setting's, as shown in Fig. 3.30. Finally, check USB debugging.

Once set, the device will run the adb daemon (adbd) in the background and wait for a USB connection. The daemon will run under the non-privileged shell user account to limit the access it has to data. AVDs and physical devices that have root access enabled will run adbd as root providing complete access to the system. Additional details on this topic will be covered in Chapter 6.

In newer versions of Android, anytime a device with USB debugging enabled is connected over USB, it will display a security warning as seen in Fig. 3.31.

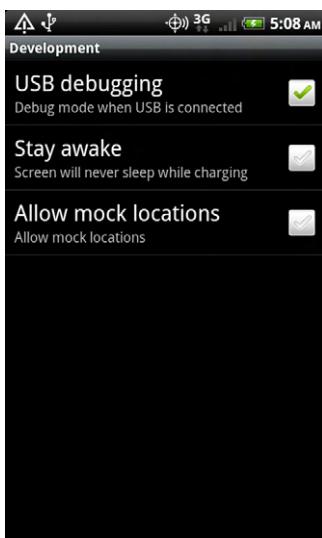


FIGURE 3.30

Enable USB debugging.

**FIGURE 3.31**

USB debugging warning.

For every current logical Android forensic tool, USB debugging must be enabled. While this is trivial to achieve if the device is unlocked, it is far more difficult if the device has a pass code. There are some techniques that can circumvent the pass code, discussed in Chapter 6. However, they do not work on every platform.

Introduction to Android Debug Bridge

Throughout the rest of this book, we will leverage adb extensively, so covering the basics now is important. There are three primary components involved when utilizing adb:

1. The adbd running on the Android device
2. The adbd running on your workstation
3. The adb client program running on your workstation

As previously covered, when you enable USB debugging on an Android device, the daemon will run and listen for a connection. Communication between the device's adbd and your workstation's adbd takes place over the virtual network running on top of the USB connection. The daemons communicate over their local host on ports 5555 through 5585. When the workstation's adbd detects a new emulator or device, it creates two sequential port connections. The even port communicates with the device's console while the odd port is for adb connections. The local adb client program uses port 5037 to communicate with the local adbd.

The most basic adb command you can issue is the adb devices command, which provides a list of connected devices.

```
ahoog@ubuntu:~$ adb devices
List of devices attached
HT08XHJ00657    device
```

Another important command provides the ability to kill your local adb service. To achieve this, type the following:

```
ahoog@ubuntu:~$ adb kill-server
ahoog@ubuntu:~$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
HT08XHJ00657    device
```

As you can see, if the adbd on the workstation is not running, it will be automatically started. On Ubuntu, if you ever receive the following response:

```
ahoog@ubuntu:~$ adb devices
List of devices attached
????????????    no permissions
```

it is likely that the connected Android device has a new vendor ID which must be identified (sudo lsusb -v) and added to the udev rule as discussed in the “SDK install” section. In Microsoft Windows, if the Android device is not recognized you will be alerted and you must install the proper USB drivers from Google or the manufacturer.

One powerful adb command all analysts and engineers should know is “adb shell,” which allows you to open a shell on the Android device and interact with the system. This is an important feature for anyone exploring Android. For example, start an AVD and follow these steps to view the application data directories on the device:

```
ahoog@ubuntu:~$ adb shell
# cd /data/data
# ls
com.android.sdksetup
com.android.calculator2
com.android.packageinstaller
com.android.providers.userdictionary
com.android.development
com.android.soundrecorder
com.android.providers.drm
com.android.spare_parts
com.android.providers.downloads.ui
com.android.protips
com.android.fallback
com.android.browser
com.android.providers.applications
com.android.netspeed
com.android.wallpaper.livepicker
android.tts
com.android.htmlviewer
```

```
com.android.music
com.android.certinstaller
com.android.inputmethod.pinyin
com.android.providers.subscribedfeeds
com.android.inputmethod.latin
com.android.gallery
com.android.systemui
com.android.contacts
com.android.camera
com.android.term
com.android.speechrecorder
com.android.server.vpn
com.android.quicksearchbox
com.android.defcontainer
com.svox.pico
com.android.customlocale
com.android.providers.settings
com.android.settings
com.android.providers.contacts
jp.co.omronsoft.openwnn
com.android.phone
com.android.launcher
com.android.providers.telephony
com.android.mms
com.android.providers.media
com.android.providers.downloads
com.android.deskclock
com.android.email
```

The functionality of adb has increased with each new SDK and is a very powerful tool. Some of the features will be explored in detail in Chapter 6, including:

1. Running shell commands on the device
2. Installing applications using command line
3. Forwarding ports between your workstation and the device
4. Copying files and folders recursively to and from the device
5. Viewing device log files

Full documentation for the adb command can be found on the Android Developer web site <http://developer.android.com/guide/developing/tools/adb.html#commandsummary>.

Testing various commands using an Android emulator is an excellent way to understand the tool prior to leveraging it in an investigation.

SUMMARY

The Android SDK not only provides deep insight into the Android platform but also provides powerful tools to investigate a device, from both a forensic and security viewpoint. Once the SDK is installed on a forensic workstation, the examiner has the ability to interact with an Android device connected via USB, provided the USB debugging feature is enabled. Not only is it possible to query information from the device but apps can also be installed, run, and ultimately

data extracted from the device. The Android SDK is an important tool used for forensic and security analysis.

References

- Android timeline.* (n.d.). Android tutorials, news, views and forums, Android Academy. Retrieved March 12, 2011, from <http://www.androidacademy.com/1-android-timeline>.
- Platform Versions,* (n.d.). Android developers. Retrieved March 12, 2011, from <http://developer.android.com/resources/dashboard/platform-versions.html>.
- comScore Reports November 2010 U.S. Mobile Subscriber Market Share—comScore, Inc.* (n.d.). comScore, Inc.—Measuring the digital world. Retrieved March 12, 2011, from http://www.comscore.com/Press_Events/Press_Releases/2011/1/comScore_Reports_November_2010_.
- Android 1.5 Platform.* (n.d.). Android developers. Retrieved March 12, 2011, from <http://developer.android.com/sdk/android-1.5.html>.
- Android 1.6 Platform.* (n.d.). Android developers. Retrieved March 12, 2011, from <http://developer.android.com/sdk/android-1.6.html>.
- Android 2.1 Platform.* (n.d.). Android developers. Retrieved March 12, 2011, from <http://developer.android.com/sdk/android-2.1.html>.
- SDK Archives.* (n.d.). Android developers. Retrieved March 13, 2011, from http://developer.android.com/sdk/older_releases.html.
- smali-Project Hosting on Google Code.* (n.d.). Google code. Retrieved March 13, 2011, from <http://code.google.com/p/smali/>.
- Android On Lockdown: AT&T Removes Best Parts of Android from Backflip* (n.d.). AndroidGuys. The trusted source for Android news and opinion, Est. 2007. Retrieved March 13, 2011, from <http://www.androidguys.com/2010/03/08/android-lockdown-at-removes-parts-android-backflip/>.