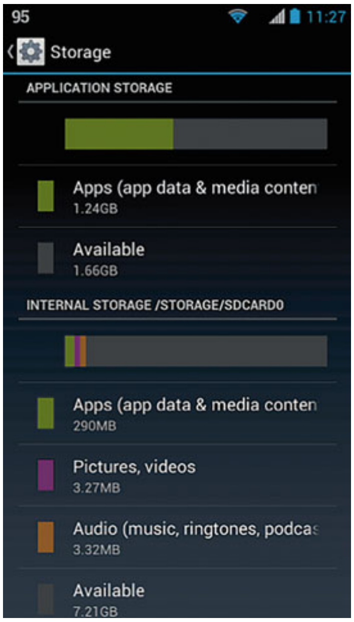
Chapter 15
Android Forensics

Learning Objectives
The objectives of this chapter are to:

- Understand fundamentals of mobile device forensics
- Understand different types of data acquisition methods for mobile devices
- Know how to extract data from an Android device
- Know how to analyze acquired data
- Become familiar with the tools necessary to examine Android devices

As mobile devices become more prevalent in our society, evidence relating to crimes will be more frequently found on mobile devices. Smartphones are the most common form of mobile devices. It is assumed that most people you come into contact with have access to a smartphone. With the increasing popularity of smartphones, they become more integrated into every aspect of our lives. These devices are constantly becoming more sophisticated. Because of the available computing power and hardware features (sensors, etc.), the range of applications available for download (and correspondingly the range of tasks that can be accomplished using one) is staggering. All of these applications have the potential to store information locally on the device. For example, instant messaging applications are widely used, and allow users to share a significant amount of personal information. It is likely that traces of this data can be found locally in smartphone storage. This has led to an increase in the need for smartphone digital forensics, especially for Android platform. This is due to the proliferation of Android devices, the many features they provide, the many applications available for them, and correspondingly the rich set of data that can be found in local storage. It can be evident that as of September 2015, there were approximately 1.4 billion active Android devices worldwide [1]. It has translated into strong demand for Android digital forensics.

X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_15



APPLICATION STORAGE

Apps (app data & media conten
1.24GB

Available
1.66GB

INTERNAL STORAGE /STORAGE/SDCARD0

Apps (app data & media conten
290MB

Pictures, videos
3.27MB

Audio (music, ringtones, podca
3.32MB

Available
7.21GB

Fig. 15.1 An example of storage memory in Samsung Galaxy S7

In this chapter, we shall examine smartphones, particularly Android devices. Smartphones can nevertheless be considered as a simplified version of computer. In reality, a smartphone today can be more powerful than a desktop or laptop computer many years ago. It is expected that very soon, a smartphone could be the only computer we need to fulfill all of our computing demands. However, from the perspective of digital forensics, smartphones are still different from regular computers in terms of various factors: First, unlike traditional file systems such as FAT, NTFS, as discussed in Chaps. 5, 6, 7, and 8, designed for hard disks used by regular computers, many modern smartphones make use of the NAND flash specific file systems, for example, YAFFS2 [1]. Second, no longer is a smartphone something that just makes telephone calls, but a convenient device that has an ever-growing number of installed applications to revolutionize our lives. Many of these applications process a significant amount of personal information. An excellent example of this is Instant Messaging (IM) applications. In addition to processing this information, there is a high likelihood that they store traces of it in local storage. As a result, a smartphone contains more information than a regular mobile phone (cell phone). For example, a lot of the information could be found on a smartphone, such as call history, contacts, calendars, SMS, MMS, Internet activities, photos or videos, GPS locations, data stored on SD cards, etc.

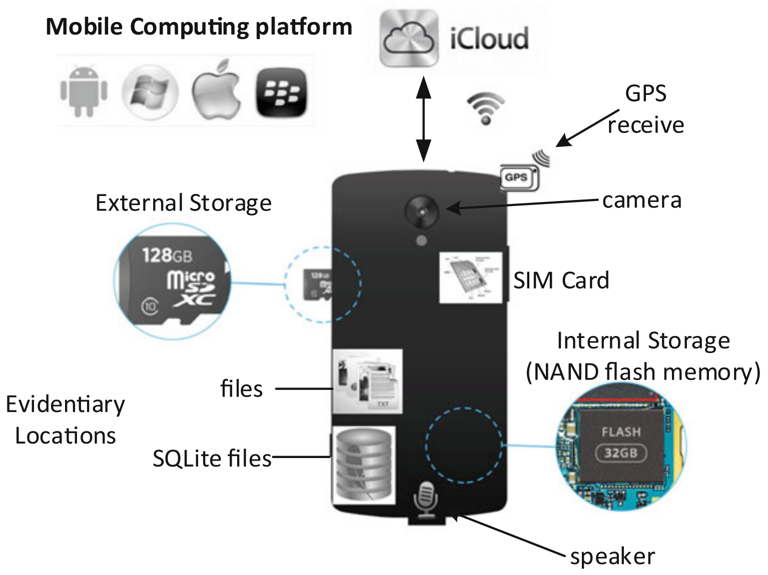


Fig. 15.2 Smartphone structure

15.1 Mobile Phone Fundamentals

Smartphones are becoming more powerful (e.g. increasing CPU processing power, a wide variety of sensors built into it, and user-friendly touchscreen interface); and thus, the number of applications available for such devices is surging, revolutionizing our lives. As shown in Fig. 15.2, a typical smartphone today has the following logical structure and consists of:

Processor It is considered as the brain of the smartphone. Examples of mobile processor are Apple's A8, Qualcomm's Snapdragon 810, and Samsung's Exynos range.

Storage Memory Today's modern smartphone provides various memory storage options. For example, Samsung Galaxy S7, an Android device, supports both built-in memory (internal memory or internal storage) and microSD cards (external storage) (Fig. 15.1). Also, internal memory of a smartphone is typically partitioned into two parts: System storage and Phone storage.

System storage, aka System Memory, is used to store the Android operation system as well as System Applications. Also, it stores all data and cache from apps you installed. This part of storage is inaccessible to users using the regular way. Phone storage is the space that can be directly accessed by the users. For example, users can install downloaded apps and store photos taken by them as well as download music, pictures, and videos files. When the phone is connected to a computer via USB, this part of storage area behaves much like an SD card. This is why it is also called internal SD but in reality is not a SD card. It is merely a non-removable storage media in the phone. So you can add or delete files just like what you do to data stored on the hard drive of a computer.

Sensors Unlike traditional computers, a growing number of smartphones are equipped with a variety of powerful sensors, such as accelerometer, digital compass, gravity, gyroscope, GPS, fingerprint sensor, and thermometer. These sensors make smartphones possible to capture diversity users' input as well as the nearby environment where they are in. As a result, a smartphone contains many increasingly sensitive data about its user.

SIM Card Most phones, particularly, GSM phones, need a SIM (Subscriber Identity Module) card to communicate with a cellular carrier. It is used to uniquely identify and authenticate a subscriber to its mobile service provider or carrier. In other words, it links a phone to the subscriber or user.

Network Connectivity Due to widely available Wi-Fi on smartphones today, the number of networked apps is surging. For example, social networking applications (e.g., Facebook and Twitter) are pre-installed on most of smartphones. In addition, cloud computing is becoming more and more prevalent on smartphones. In cloud computing, applications and data storage are provided as services to mobile users via the Internet.

Cameras and Speakers Today's smartphones come equipped with digital cameras and speakers.

However, currently there are countless different models of smartphones. Although they're basically very similar in terms of their internal architectures, phone manufacturers may store data in proprietary formats for their mobile phones, making it very challenging to forensically examine mobile phones. Nonetheless, there are less mobile operating systems. The most popular ones are Android and iOS.

15.2 Mobile Device Forensic Investigation

As discussed in Chap. 1, proper procedure must be followed to ensure that the evidence retrieved from a suspect's mobile phone is admissible in a court of law. As for mobile phones, it is more challenging since today's mobile phones are equipped with many wireless technologies, including Wi-Fi (short for Wireless-Fidelity), Bluetooth and cellular. If a mobile phone is still on and connected to the network, it is very important to isolate the phone from its surrounding wireless networks and devices to preserve the integrity of data stored on the phone. For example, place the mobile phone in a "Faraday Bag", which blocks Radio Frequency (RF) signals, including cell signals, satellite, Wifi, and Bluetooth frequencies (Fig. 15.3).

Also, mobile phones attached to a computer, for example via a USB cable, or cradle/docking station should be disconnected from the computer immediately.

Fig. 15.3 Faraday bag

There are three key aspects to a mobile device forensic investigation: Data storage location(s), data extraction, and data analysis [2]. Specifically, one must know where data is stored, how it is stored, and any associated file permissions before any type of extraction can be executed. Once this information is known, the data must be extracted. This is a critical aspect of a forensic investigation—to the extent that choosing an inappropriate method can ruin an investigation. There are many different extraction methods, each with its pros and cons. Lastly, one has access to the application data; in order to make sense of it, it must be analyzed, aggregated, and put into context.

Regarding these above key issues, this section focuses on presenting data storage locations, data acquisition methods and data analysis methods for Android digital forensics. Furthermore, case studies are conducted on two popular instant messaging applications to show how an acquisition method and data analysis methodologies are employed in practice to analyze the private storage of these social media applications.

15.2.1 Storage Location

In order to extract and analyze smartphone stored data, one must know where to find data of interest. We will now discuss standard storage locations on Android devices. Note that the file-system structure is not identical across all Android devices. However, there are specific locations that are fairly standard (such as app data being stored in the "/data/data/" directory). Figure 15.4 shows a hierarchical depiction of Android storage.

The top level (red) represents partitions (a subset of all partitions); the second and third levels (blue and green respectively) represent content found in these partitions. The "userdata" partition, mounted at "/data", contains all private application storage. This is protected content, only accessible by each application. This is an important observation when considering data acquisition methods—as a user must have root privileges to access this content directly. There are some interesting acquisition methods that address this issue, which will be discussed later.

The path to an application's storage is "/data/data/ <pack- ageName>". Specifically, Google Hangouts' application data is found at "/data/data/com.google.android.talk/", and Facebook Messenger's application data is found at "/data/data/

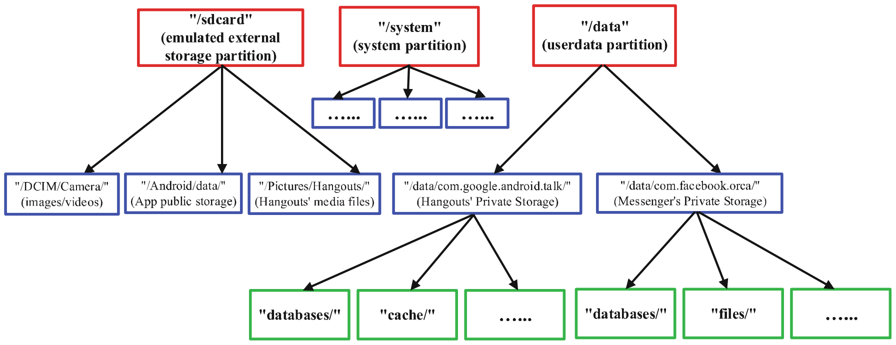

"/sdcard" (emulated external storage partition)
"/system" (system partition)
"/data" (userdata partition)
……
……
……
"/DCIM/Camera/" (images/videos)
"/Android/data/" (App public storage)
"/Pictures/Hangouts/" (Hangouts' media files)
"/data/com.google.android.talk/" (Hangouts' Private Storage)
"/data/com.facebook.orca/" (Messenger's Private Storage)
"databases/"
"cache/"
……
"databases/"
"files/"
……

Fig. 15.4 Android storage

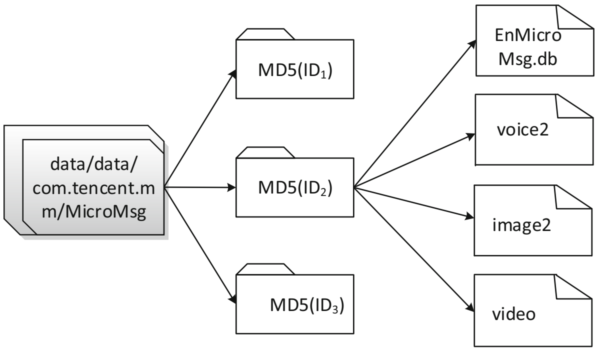


Fig. 15.5 The storage path of WeChat messages

com.facebook.orca/". Notably, WeChat's messages are saved in "data/data/com.tencent.mm/MicroMsg/MD5(ID)", where MD5(ID) is the MD5 hash (with total 32 characters) of WeChat account ID logged in the smartphone (Versions higher than 4.5 use encrypted storage). Text messages are stored directly in the database EnMicroMsg.db, while voice, image and video are only recorded by their path information of storage location, as shown in Fig. 15.5 [3]. For example, voice messages are stored in the subfolder called "voice2" with a special file extension "amr". Images are stored in the subfolder "image2" and videos are in the subfolder "video".

Secondly, the "sdcard" partition (mount point "/sdcard") can contain useful information. For example, this information can include: Pictures/videos taken by the smartphone's camera application, downloaded files, and public application storage. This location is unprotected, meaning that anyone has access to the content stored here. It is yet another location to look for forensically relevant content. Lastly, system, kernel, and application logs can be a valuable source of data. This data "provides an insight into the apps as well as the system running them" [4]. There are many utilities that can be used to recover logs.

15.2.2 Acquisition Methods

We have established that smartphone storage has the potential to contain vast amounts of information that may be relevant in a forensic investigation. Some of this data is due to application storing information locally on a smartphone, while other information is due to system, kernel and application logs. Now we must determine how to retrieve this data from a smartphone. First we must define the types of images of smartphone storage that can be obtained, as they will be referenced throughout the next few sections.

Logical Image A logical image can be thought of as a copy of files and folders from device storage. This means that when the data is copied, it makes sense; it is in a form that is recognizable. The files have their proper headers. The file-system is intact. However, this does mean that any deleted files, or seemingly "unused space" will not be copied. This is not a complete copy of a partition, rather it is a copy of the current logical contents of that partition (or set of folders). The benefit of a logical image is that it is easy to work with. All current files are listed, and can be analyzed immediately. The downside, mentioned already, is that some information will not be retrievable, such as deleted files.

Physical Image A physical image is a bit-by-bit copy or data dump of a storage device or partition. This means that all of the data (whether it is part of the current logical image, or deleted files, or "empty space") will be copied, and nothing will be lost. The benefit of this is obvious: The content that has been retrieved is greater than that of a logical image. Therefore, more potentially useful data may be recovered. The downside is the potential difficulty in reconstructing this data—for example using a method such as file carving.

Now let us discuss different data acquisition methods, highlighting their pros and cons, including a combination of hardware and software methods.

15.2.2.1 Chip-Off

This data extraction method is complex and requires a great deal of technical knowledge, dexterity, and confidence to actually disassemble a mobile device. The term "chip-off" is literal, meaning that the NAND flash chips are actually removed from the circuit boards, and interfaced directly with hardware tools through their pins. These chips are soldered to the circuit board, meaning tools such as a soldering iron are needed in order to physically extract the flash chips [4].

A downside to this method is the potential of damaging the flash chips while extracting them from the PCB. Secondly, carefully disassembling a smartphone can be a time consuming task. If there is a strict time limit on an investigation, this method may not be suitable.

A significant benefit of the chip-off method is that even if the smartphone in question is damaged, it may be possible to retrieve data from it (as long as the flash chips are not damaged). Other "nonessential" circuitry critical for the smartphone to run may be damaged, however the flash chips themselves may be unaffected. In contrast, software acquisition methods require that the device be bootable, and able to function normally.

15.2.2.2 JTAG (Joint Test Action Group)

JTAG is a communications protocol, often supported by processors "to provide access to their debug/emulation functions" [5]. "With JTAG, you connect directly to the device's CPU by soldering leads to certain JTAG pads on the printed circuit board" [5]. This connection (Data IN, Data OUT, Control, Clock; collectively called the Test Access Port [5]) allows JTAG software to interface directly with the CPU, and provide commands that are able to obtain "a complete binary memory dump of the NAND flash" [4]. The result of this is a complete bit-by-bit physical image of flash memory.

One benefit of JTAG over chip-off is that less physical modification of the device is required, meaning there is less probability of damaging the flash chips. However, the CPU must not be damaged for this method to be possible. The chip-off method may still work, even if the CPU is damaged and JTAG will not work. JTAG, however, may still work even if the device is somewhat damaged, and not able to boot up. The extent of damage to the device will play a role in deciding which extraction method to use.

Figure 15.6a–c are an illustration of how JTAG works on Huawei C8650 smartphone. We use the Z3X Easy JTAG Box [6] to extract the physical image from it. After the GND line is identified, the Easy JTAG Box is able to automatically identify the rest of the JTAG pins. Also, the complete memory contents of the Huawei C8650 smartphone are extracted. It is worth pointing out that it can be time-consuming to get the right pins manually.

Some downsides to the JTAG method are as follows: Not all devices support JTAG; Test Action Port connections may be difficult to find, or not even be included

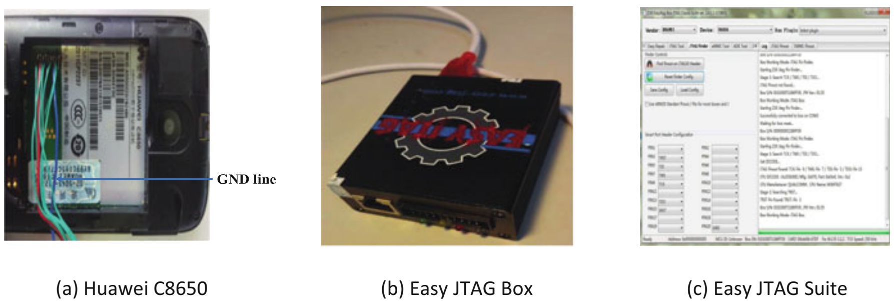


Fig. 15.6 JTAG mobile device imaging. (**a**) Huawei C8650. (**b**) Easy JTAG Box. (**c**) Easy JTAG Suite

on the PCB (in the latter case the TAP leads will need to be manually added—making this method more invasive); the extraction itself is relatively slow, especially when considering the internal storage size of modern smartphones.

Two shared benefits of both the Chip-Off and JTAG acquisition methods are: No knowledge of screen-lock credentials is required, as data is being extracted by directly interfacing with hardware. Secondly, a physical image is obtained, providing the investigator with the most possible information.

15.2.2.3 Forensic Software Suites

There are numerous available commercial forensic software suites designed for smartphone devices. Many of them utilize content providers [4]: An Android feature that enables applications to share data with one another. This sharing feature is necessary, due to the Android security model where all applications have their own private data and are not meant to interact with each other. This method enables forensic software to retrieve some logical data from an Android device, however, only that which the applications are enabled to share. There are many content providers, capable of sharing information such as SMS/MMS messages, contacts, call logs, etc..

Many commercial forensic software suites are able to retrieve much more contents than that provided through content providers. Some can acquire a complete physical image of device memory. This is due to built-in rooting support for many Android smartphones (and corresponding Android versions), which results in full administrative access to resources. Yang et al. [7] states that this rooting method is utilized by commercial forensic software suites Oxygen Forensics, AccessData MPE+, and MSAB XRY. They also mention that Cellebrite UFED 4PC "basically supports an ADB physical memory dump via rooting exploitation" [7]. Note that there is a great debate on whether rooting a device is a forensically sound method, due to the extent of device modification that is often required (violating data integrity).

They not only provide methods to extract data from devices, but also display it and store it in convenient ways. For example, software might show a statistical overview of how many messages you received from one individual, and when you received them. This type of data view can be used to observe patterns in user behavior, which would be relevant information in a forensic investigation. The ease of use and fountain of features are a definite plus.

There are, however, several downsides to commercial forensic software suites. The software can be quite expensive. Support for new devices may be somewhat slow (relative to a manual "do it yourself method"), due to the need to up-date the software accordingly. Lastly, as mentioned above, the extent of device modification can be a problem (for example through the rooting process).

15.2.2.4 ADB (Android Debug Bridge)

This is a command line tool that allows communication between your computer and an Android mobile device (or emulator). The client server program includes the following three components [8]: Client (runs on developer machine, and sends commands to the device), Daemon (runs as a background process on the Android device, and runs the commands sent to it from the client), Server (runs as a background process on the developer machine, and manages communication between the client and daemon).

This software includes commands such as: Pull (pull files from an Android device to the local machine), push (copy files from the local machine to the Android device), shell (execute a shell on the Android device. Useful for navigating the file system, executing programs, etc.), utilities for dumping log files, etc..

Obviously, from the aspect of Android forensics, the pull command can be especially useful for retrieving files from the device. If you do not have sufficient privileges on the target device (device not rooted), you may still be able to access some useful information through this command. Remember that some applications store content on the sdcard; but external storage that is not protected. Also, there can be useful system information in the "/proc" and "/sys" directories which are not protected [4].

On devices that are rooted, it is straightforward to pull basically any directories, such as the "/data" directory which contains all private application data (specifically in "/data/data/<app_package>") [4].

This acquisition method is severely limited by two factors:

- Whether the device is rooted or not (must be to recover "userdata" partition data);
- Whether the investigator has knowledge of screen-lock credentials.

In order to interface with a device using this method (with the exception of custom recovery modes), "USB debugging" must be enabled, and the screen must be unlocked. However, one must unlock the screen in the first place to enable "USB debugging" if it is not already enabled. Because of these limitations, it is easy to think of many situations where ADB cannot be used to recover content.

15.2.2.5 Backup Applications

If your device is not rooted, a backup application can make use of content providers to acquire a decent amount of data. "Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process" [12]. This mechanism will not copy all application data, as it only has access to the data applications which are programmed to share. Some useful information that can be accessed through content providers is (not exhaustive—there are many content providers, as all apps can share data this way): SMS/MMS, call logs, contacts, calendar, browser information, etc..

If your device is rooted, then these applications may be able to access privileged data directly (for example: make a complete copy of the "/data/data" directory). For example, Mutawa et al. [9] use the backup application "MyBackup (v2.7.7)" [9] to acquire a logical backup of a rooted Samsung Galaxy S device. This backup includes private application folders from the "/data/data/" directory, which are copied to an external micro SD card, which is subsequently removed from the smartphone and analyzed using their forensic workstation.

A benefit of this method is that it is straightforward. One simply needs to install the application (and maybe a microSD card), and let the application do its work.

There are also many downsides to this method. The first is related to device integrity. By installing an application you are actually modifying the "userdata" partition. This type of modification may be considered unacceptable. If rooting the device is part of the acquisition process, the device is modified even more. Lastly, in order to install such an application, you must have access to the device (i.e. know screen lock credentials). This means that this method will not be feasible if you don't have access past the lock screen. This method has many downsides, and in our opinion should be avoided if possible.

15.2.2.6 Firmware Update Protocols

The previously mentioned software based acquisition methods have all utilized either applications (backup apps/forensic apps), or software that interfaces with the device (adb, forensic software suites). The type of content that is recoverable depended on if the target device is rooted or not. If the device was rooted, then basically the whole file-system could be acquired. Otherwise, content providers were utilized to gather application shared data (that otherwise would be unreachable), and unprotected files/folders could be copied directly.

Now we will switch our focus to more technical acquisition methods that require a significant amount of knowledge regarding the low level details of how Android devices function. Also, device specific details (based on the smart phone manufacturer and the model of the device) are required. Note that there is not one set of guidelines that can be followed to acquire data from any Android powered device. In general, the order of the steps, and what each step accomplishes is the same. However there are subtle differences based on the specific device being worked on.

Yang et al. have proposed a new acquisition method based on firmware update protocols [7]. A device can be booted into firmware update or download mode, and when in this mode the firmware update process can run, which is used to flash new system firmware [7]. "Only the bootloader and USB function can operate in this mode" [7]. An important note is that "a firmware update protocol is the only way to access the flash memory directly through S/W" [7]. Therefore, there may be commands that can be used to access flash memory, and possibly acquire the data stored within. The bootloader and firmware update program could be analyzed through reverse engineering to determine the firmware update processes and commands. This analysis results in the discovery of specific commands that can be used to obtain a physical image of memory. “We analyzed the firmware update protocols used by LG, Pantech, and Samsung smart-phones. In the LG and Pantech models, we found that not only commands for direct access to flash memory and the write commands for flashing but also the read commands for flash memory dumping were preserved intact” [7]. Since the firmware update protocol can be used to directly access flash memory, and there is a command for reading this memory (note that this is not always possible for all devices), this can be used to acquire a physical image (partial or whole) of the flash memory of many Android devices.

In order to acquire data in this way, it implements the following steps:

- Determine the required firmware update protocol command/format: “Analyze the firmware update processes and commands by decompiling the bootloader and the update program” [7];
- Boot into firmware update mode (key combination varies by manufacturer and device)—Note: Integrity of data guaranteed, since only the boot-loader and USB are active [7];
- Connect device to workstation;
- Send command for reading flash storage.

These authors take their findings and implement this acquisition method in a software tool called Android Physical Dump (APD). They have reverse engineered the bootloader and update program of many devices, and include the associated commands required for reading memory in their software. There seems to be only a few downsides to this acquisition method. Firstly, it requires time and effort to reverse engineer bootloaders and firmware update software for different devices. However, this is only a one time investment. Once the commands are found they can be used immediately for that specific device any time they are required. Secondly, this method is not possible for all devices.

There are many benefits of this method. It provides a sophisticated way to recover information from many smart-phones, without affecting the device’s data integrity. Also, a physical image is recovered, which provides more data than a logical image. In contrast to the hardware acquisition methods, this method provides a way to recover a physical image without having to disassemble the smartphone. Therefore, there is no chance of physically damaging internal circuitry.

15.2.2.7 Custom Recovery Image

The last acquisition method that we will discuss is focused on the recovery partition and recovery mode of Android devices. Vidas et al. [10] outline a method for obtaining logical and physical images of device storage using this method. This method does require modifying the recovery partition. However, much of the content of interest is located on the “userdata” partition (as mentioned earlier is mounted at "/data"), and as such modifying the recovery partition won't affect this data.

Here is an outline of how this acquisition technique works [10]:

- Obtain or create a custom collection recovery image (one that includes specific utilities that enable data recovery, adb, and superuser);
- Flash this recovery image to device;
- Reboot into recovery mode;
- Utilize the "adb shell" command from your forensic workstation to execute data recovery binaries from the recovery image.

There are different data dumping utilities that can be used, depending on the flash storage technology being used. Smartphones may use different storage technologies based on how old they are. Older devices will use raw NAND flash storage, while newer devices will use eMMC. The difference lies in that eMMC includes a memory controller, while NAND flash does not. Android devices that utilize NAND flash implement a MTD (Memory Technology Device) software layer, which acts as the FTL (Flash Transition Layer). This MTD software acts as the memory controller, interfacing with the raw NAND flash chips. For MTD devices, NAND dump can be used to collect NAND data independent of the higher-level files system deployed on the memory. However, for devices that do not employ MTD, you must resort to other collection techniques. For example, the dd utility can be used to copy data [10]. Both of these utilities can be used to recover physical images. Note that ADB commands (such as pull) could be used for logically copying files.

Son et al. [11] also write a paper focusing on data acquisition through the use of custom recovery images. Their work continues that of Vidas et al., also focusing specifically on data integrity concerns. The acquisition technique can be described as follows:

The first step of the acquisition process is to prepare the custom recovery mode image (CRMI) which will include binaries for data acquisition. "It includes nanddump for imaging the file system that uses MTD like YAFFS2 and the busybox binary that includes two commands in order to send a file using Netcat, also known as TCP/IP Swiss Army Knife, which is a popular tool for reading from and writing to network connections using TCP or UDP" [11]. There are also methods which use ADB to pull specific files (logical acquisition). Note that the size of the CRMI must be considered, as the size of the boot partition is limited.

The device must then be booted into flash mode, so that the CRMI can be flashed to the recovery partition (or the boot partition). In some devices the bootloader is locked; And it must be unlocked before the CRMI can be flashed [11]. At this time they stress an important point related to the smartphone's data integrity. The smartphone must be manually entered into recovery mode, however, "if such booting in Recovery Mode fails, the device will proceed to boot normally. If that happens, the user data partition will be used, thus potentially damaging data integrity" [11]. Because of this concern they specify a second option, which is to flash the CRMI to the boot partition, and "then the device could enter into the recovery mode right away without the need for manual control" [11]. This is the method that they employ.

With regard to data acquisition, there are two options: The first is to recover a physical image through imaging a data partition (for example using nanddump or dd), the second is to recover a logical image through copying files (for example using adb pull). It is not necessary to mount a partition in order to recover the dump of a partition. However, special care must be taken when acquiring files. In this case the partition (for example "userdata") must be mounted. It is important that "the partition should be mounted in read only mode in order to guarantee data integrity" [11]. As mentioned, the data can be acquired from the mounted partition through the use of ADB pull.

Son et al. include a section describing how to return the target device to its former state. This stage includes obtaining the original boot image (which was overwritten with the CRMI), and subsequently overwriting the boot partition of the device with this original boot image (using ADB push, or flashing the partition through rebooting into flash mode) [11].

The final step is to take their findings (all of the steps outlined above), and create an automated GUI software suite called "Android Extractor" to execute this process.

Vidas et al. and Son et al. describe a viable method for data acquisition. Son et al. further the discussion, taking into consideration the data integrity of the device, which is an important aspect of digital forensic investigations.

Srivastava et al. [12] in their recent work do a data extraction (and subsequent analysis) of an Android smartphone based on the methods outlined by Vidas et al. and Son et al. They extend their experiment to include other data as well (and other extraction techniques), however this method still proves to be effective.

There are some downsides to this acquisition method. Firstly, custom recovery images must be produced that support different devices. This can be a time consuming task. However, this only has to be done once for each device. Secondly, device storage is modified somewhat. However, this modification is restricted to the recovery partition; separate from any stored application information. Lastly, a device's boot-loader could be locked, which would make the extraction process more difficult. Benefits of this method are as follows: It is relatively straight forward to execute. Once a custom recovery image has been produced, the steps required to extract information are simple. Compared to some other extraction methods (e.g. backup applications, forensic software suites), the extent of device modification is less. Both physical and logical images can be obtained. This flexibility is excellent. An investigator could start by analyzing a logical image, and subsequently analyze a physical image if required. Lastly, no knowledge of screen-lock credentials is required.

15.2.3 Data Analysis

After extracting data from the phone, it comes to the data analysis step for forensic purpose. Data analysis approaches vary with the Android applications such as instant message, phone call, web browser, and so on. In other words, each application requires its own unique forensic analysis method. As instant message and social network applications are the most popular applications in mobile phone, we focused on surveying data analysis methods in Android social networking applications, for example, Facebook [13], Whatsapp [14, 15], WeChat [3], etc. In most of these works, data are analyzed and the users' activities or habits are investigated from the following aspects (but not limited to):

- Analysis of contact information: The contact information allows an investigator to determine who the user was in contact with. By analyzing the list of contacts, the timestamp of a contact has been added to the database, or the blocked status of a given contact, the user's behavior or his/her contact information is disclosed to the investigators.
- Analysis of exchanged messages: The chronology of exchanged messages can be reconstructed by determining the timestamp of an exchanged message, the data it carried, the set of users involved in the conversation, and whether and when it has actually been received by its recipients. These information provide the investigators with the sender's and the receiver's relationship to some extent.
- Analysis of deletions: In some applications, the deleted records are kept in the device for a period. For example, in SQLite databases, the deletions can recovery from so-called unallocated cells, i.e. slack space stored in the file corresponding to the database [16]. These deletions give some guides for the investigation.

Many social networking applications are integrated into new smartphones, thus investigators may be able to find relevant evidence on a suspect's smartphone in cases involving social networks [13].

15.2.3.1 Facebook

In their paper, Mutawa et al. [13] discuss an experiment designed to recover social networking application data from Android, iPhone, and Blackberry smartphones. This experiment is focused on three applications: Facebook, Twitter, and MySpace. They recover a logical image of device storage (rooting the phone, then acquiring backup), and analyze the data stored by these applications. In the Android forensic examination for Facebook, The file "*com.facebook.katana_4130.zip*" contains three subdirectories: Databases, files, and lib. Each directory holds a number of files. The databases folder contains three SQLite files: "fb.db", "webview.db", and "webviewCache.db". The first file fb.db contains tables that hold records of activities performed by the Android Facebook application user, including created albums, chat messages, list of friends, friend data, mailbox messages, and uploaded photos. These records include significant information for the forensic investigator, such as the users' IDs, contents of exchanged messages, URL links of uploaded pictures, and timestamps of performed activities. Walnycky et al. [17] conduct a combined device storage and network forensic experiment to determine what type of content is recoverable from social-messaging applications. This experiment is unique, as it includes network traffic analysis. They analyze twenty applications, including Facebook Messenger.

Notably, albeit Facebook is one of the most popular social network applications in the world, there is limited work for in-depth forensic analysis of Facebook in Android smartphone.

15.2.3.2 WhatsApp

The works of [14, 15] both focus on the forensic analysis of WhatsApp Messenger on Android. The tests and analysis of [14] are performed with the aim of determining what kind of data and information can be found on the device's internal memory which are related to social messenger applications, e.g. chat log and history, sent and received image or video files, etc. These works only analyze chat database of the application. In order to cover more artifacts of WhatsApp Messenger, [18] provides a complete description of all the artifacts generated by WhatsApp Messenger. This work discusses the decoding and the interpretation of each one of them. Also, it shows how they can be correlated together to infer various types of information that cannot be obtained by considering each one of them in isolation. An investigator will be able to reconstruct the list of contacts and the chronology of the messages that have been exchanged by users by using the results. Furthermore, thanks to the correlation of multiple artifacts, he/she will be able to infer information like when a specific contact has been added, to recover deleted contacts and their time of deletion, to determine which messages have been deleted, when these messages have been exchanged, and the users that exchanged them. This is a very in-depth analysis of one application, in contrast to an overview of recoverable data artifacts from various applications of the same type.

Different from the aforementioned methods which deal with the identification and analysis of all the artifacts generated by WhatsApp Messenger, [19] focuses on the analysis of several IM applications (including WhatsApp Messenger) on various smartphone platforms, including Android, with the aim of identifying the encryption algorithms used by them. Also, [20] proposes a decryption method for the network traffic of WhatsApp, as well as extraction and analytical technology for the associated communication data.

15.2.3.3 WeChat

WeChat is one of the most popular instant-messaging smartphone applications in the world, whose chat messages are all stored in local installation folder. Zhang et al. [3] investigates the data forensics of WeChat messages in local encrypted database as the text message is stored in encrypted SQLite database. This work analyzes its cryptographic algorithm, key derivation principles and presents the corresponding database decryption process in different practical forensic circumstances. Further, they exploit the data recovery of voice and deleted messages, which would also be helpful in data forensic for criminal investigation. By implementing an actual test, the proposed forensic techniques can successfully recover the encrypted and deleted messages, which provides a good solution for WeChat data forensics. In order to extract an encrypted and deleted chat history on WeChat, [21] makes an in-depth analysis on the structure of the volatile Android memory. The works of [22, 23] use ADB to extract the WeChat data. Wu et al. [23] recovers the scene of conversations, including who does the user communicate with and what is said, and what the user is sharing with the Moments. These information help the investigators to catch a better understanding in reconstructing activities related to usage of WeChat on Android smartphones.

15.2.3.4 Other Social Applications

Apart from the above social applications, many other social applications are also gaining increasing interests in Android forensics. Walnycky et al. [17] perform an experimental forensic study on twenty social-messaging applications for the Android mobile phone operating system. They are able to reconstruct or intercept data such as: Passwords, screenshots taken by applications, pictures, videos, audio sent, messages sent, sketches, profile pictures and more, from sixteen of the twenty applications tested. These information can be used for evidence collection purposes by digital forensic practitioners. This work shows which features of these instant messaging applications leave potential evidence allowing for suspect data to be reconstructed or partially reconstructed.

Satrya et al. [24] present details about digital forensics investigation on private chat in three social messenger applications: Telegram, Line and KakaoTalk. The focus of the investigation is Telegram's "Secret Chat", Line's "Hidden Chat", and KakaoTalk's "Secret Chat". They explain all the artifacts produced by social messengers. An investigator will be able to read, reconstruct, and present the chronology of the messages by the interpretations of the generated messages as well as how they relate to one another. Experiments are implemented in two smartphones with different brands and different Android OS versions, which conduct a digital investigation in a forensically sound manner. In another paper, Satrya et al. [25] provide a thorough description of all the artifacts that are generated by the messenger application Telegram on Android OS. They show how the remnant data relate to each other within the process from the acquisition to the analysis, such as when the user carries out the installation process, signup/in, add/delete/block contact, message sending process (text, figures, or voice), location/file sharing, sign out, and uninstall. Investigations are conducted from the aspects of application and user activity, contact information, messages exchanged, file and location sharing, and deleted communication.

ChatSecure is a secure IM application that provides strong encryption for transmitted and locally-stored data to ensure the privacy of its users. In [16], Anglano et al. provide a detailed forensic analysis of ChatSecure that is aimed at identifying all the relevant artifacts it generates, interpreting them, and using them to reconstruct the activities carried out by its users. Specifically, after identifying and extracting passphrase from the volatile memory of the device, decryption can be carried to discover the data in the databases for investigation. The authors discuss how to analyze and correlate the data stored in the databases used by ChatSecure to identify the IM accounts used by the user as well as his/her buddies to communicate. Forensic investigators are able to reconstruct the chronology and contents of the messages and files that have been exchanged by using ChatSecure on the smartphones.

15.2.4 Case Studies

Now that we have discussed where data is stored, and several different ways to extract it, we next will use case studies as examples of how to conduct an Android smartphone forensic investigation. This experiment will showcase an excellent data acquisition method; Showing how effective it is in practice. Secondly, we will show the rich data that is stored and is subsequently recoverable from the instant messaging applications Facebook Messenger and Google Hangouts.

15.2.4.1 Experiment Setup

Smartphone:

- Asus Zenfone 2 Laser
- Model: ASUS Z00TD (ZE551KL)
- Android Version: 6.0.1
- MicroSD card: 8GB

Applications:

- Facebook Messenger—version: 86.0.0.17.70
- Google Hangouts—version: 11.0.130004787

Software:

- DB Browser for SQLite (version 3.9.0)
- Notepad ++ (version 5.9)
- Dcode (version 4.02a) [16]
- HxD—Hexeditor (version 1.7.7.0)
- Cygwin Terminal
- Android Debug Bridge (version 1.0.31)

- Fastboot Binaries
- TWRP recovery image [26]

15.2.4.2 Application Use

Before data could be extracted and analyzed, it had to be generated. We use the two applications for an extended period of time (several weeks), utilizing all of their available features.

This included: Sending/receiving textual messages, images, videos, voice-clips, location information, live voice/video calls, etc. in both group and personal conversations. By using all application features, we give the applications a chance to locally store data relating to each of these features.

15.2.4.3 Extraction

The applications have now been thoroughly used, meaning that they have both had a chance to store information locally on the smartphone. Therefore, the next stage of this experiment is to extract this application data from smart-phone storage.

We will be using the custom recovery image data extraction method (as described in Sect. 15.2.2.7). There are several reasons for this choice. Note that we are not advocating that this is always the best method, but that it is an excellent method for this work. Our reasons for choosing this method are as follows:

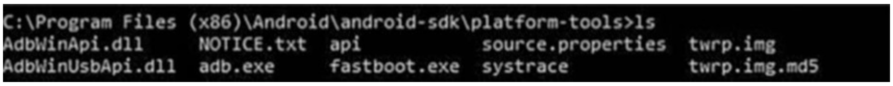
- ***Smartphone support***: There is a custom recovery image already available for the Asus smartphone being used in this work. It would be possible for us to create our own custom recovery image that includes the appropriate utilities for data extraction, etc. However, this would be a time consuming process. Instead, we will be using an open source recovery image [26] that is compatible with the Asus smartphone. The availability of this recovery image is one reason for choosing this extraction method.
- ***Data integrity***: We would argue that the custom recovery image extraction method is less invasive than other readily available methods. For example, many popular "one click root tools" install binaries and applications on the device to enable access to protected files. Commercial forensic software suites (workstation and application based) often root a target device to facilitate data extraction. Obviously forensic applications must be installed on a device to function properly. All of this modification affects the "userdata" partition, as well as the "system" partition. This modification directly affects partitions that include data of interest to forensic investigations. This violation of integrity is un-acceptable. It is true that flashing a custom recovery image to the recovery partition can modify device storage (Note however that in this work the flashing procedure is non-persistent.). However, none of the target data that we will be recovering is on this partition. Therefore, the modification is acceptable.

- ***Simplicity***: As mentioned, we will be using an already available recovery image. The steps require that the data extraction is quite straightforward. The smartphone does not need to be disassembled (JTAG, Chip-Off). No soft-ware needs to be reverse engineered (firmware update protocols). The recovery image simply needs to be flashed to the device. This simplicity is a bonus, as it allows us to use an excellent acquisition method without spending a significant amount of time on this stage of the experiment. Secondly, with more complicated methods, there is often a greater probability of damaging the device (or its data) in some way.
- ***Do not require knowledge of screen-lock credentials***: This is a major benefit of the custom recovery image method over some of the others. In a forensic investigation there is no guarantee that you will know the screen-lock credentials of a given device. This significantly reduces the efficacy of some acquisition methods. For example, to interface with a device using ADB, "USB debugging" must be enabled, and the screen must be unlocked to interface with the device; the same is true for many "one click root tools". Some commercial forensic software suites (depending on the device being analyzed) also require that you have access to the device. You do not need such access with the recovery partition modification extraction method. We are able to boot the device and interface with it in flash mode, reboot into the custom recovery mode, backup device storage onto an external MicroSD card, and remove this card and access its data all without needing any credentials.

In summary, the recovery partition modification method is an excellent data extraction method for this work (as well as other similar scenarios). The reason for this is a combination of the device being studied (Asus Zenfone 2 Laser), and the inherent attributes of the acquisition method. There is a freely available CRMI that supports this specific Asus smartphone (and many others); simplifying this stage of the experiment. Flashing the CRMI to the smartphone actually results in no storage modification, as it can be done in a non-persistent way; therefore not violating data integrity. All logical data can be recovered even if the smartphone is locked with "USB debugging" disabled, with no knowledge of the screen-lock credentials. These observations are very favorable when considering the recovery partition modification data acquisition method.

The TWRP custom recovery image can be downloaded from [26]. It is also important to download the md5 checksum value and check the integrity of the recovery image before proceeding. We have named the TWRP image "twrp.img", and the checksum "twrp.img.md5". Both files have been saved to the same directory (platform-tools) that contains the adb.exe and fastboot binaries. See Fig. 15.7.

The sequence of steps to flash the recovery partition is as follows:



```
C:\Program Files (x86)\Android\android-sdk\platform-tools>ls
AdbWinApi.dll     NOTICE.txt  api           source.properties  twrp.img
AdbWinUsbApi.dll  adb.exe     fastboot.exe  systrace           twrp.img.md5
```

Fig. 15.7 Platform tools

- Put smartphone in flash mode using device specific key combination;
- In command prompt, navigate to folder containing adb and fastboot binaries, as well as the TWRP image;
- Connect smartphone to computer via USB;
- Use command “fastboot devices” to make sure the device is recognized;
- Use command “fastboot flash recovery twrp.img” to flash recovery image to device;
- Reboot device using command “fastboot reboot”. As soon as device powers off, hold appropriate keys to boot device into recovery mode.

When the TWRP recovery image is booted into, the user will be asked if they want to allow TWRP to write to the system partition, or if it should be mounted read only. For the sake of data integrity, we have chosen the read only option. The result will be a non-persistent boot.

To extract the data, one simply has to select the “Back-up” menu from the TWRP homepage, select “MicroSD” as the location to store the backup, select the “Data” partition (same as “userdata”) to be backed up, and then start the backup process. Once it is completed, simply power off the device and remove the Micro SD card.

In addition, we used ADB to interface with the smart-phone while in recovery mode to extract photos and videos from the “sdcard” partition. This publicly available content can supplement the other recovered data well.

15.2.4.4 Data Analysis

Extracting the data backup is an extremely important step in Android forensic. Obviously, in order to analyze data it must be first extracted. However, there is still much work to be done. The data analysis phase of this work may prove to be quite time consuming and difficult, depending on the amount of files (their formats, etc.) that need to be analyzed. All files must be parsed for relevant artifacts.

Nevertheless, the locations of the application data vary among mobile phones and different applications. We will use instant messaging applications as an example to illustrate our analysis methodology and techniques. This is because instant message applications are widely used (as can be seen by their install numbers alone), and can contribute to this rich set of data. Applications such as Facebook Messenger and Google Hangouts enable users to share a vast amount of information, including images, videos, voice clips, location information (e.g. GPS coordinates), message text, as well as live voice and video conversations. If these applications store such information in local storage, it could be very useful in a forensic investigation. Therefore, we will focus on analyzing the private storage of these two applications.

As mentioned previously, most of the data of interest will be found in the location: “/data/data/<packageName>”, where the <packageName> field depends on the application. For the two applications being studied in our book, the root paths to data that will be analyzed are as follows:

- Messenger: /data/data/com.facebook.orca/,

Table 15.1 Messenger artifacts

Path	Data
cache/audio/...	Audio clips (sent/received)
cache/image/...	Shared images (sent/received)
files/video-cache/...	Videos (sent/received)
databases/	Message content: sender, text, time-stamp, attachment info, location, video/voice call info

- Hangouts: /data/data/com.google.android.talk/

Messenger Analysis We will analyze all subfolders and files found in the "/data/data/com.facebook.orca/" directory. There are many (31 to be specific) subfolders found in this directory. Some are empty, some contain many more subfolders and files, and some contain content that seems to be irrelevant (or redundant). In order to provide an organized and understandable overview of Messenger's storage, we will summarize some key recovered content and its location in Table 15.1.

The analysis of Facebook Messenger's storage is time consuming, mainly due to the number of folders and files that had to be analyzed. As mentioned above, there are 31 subfolders that require parsing. Many of these subfolders have layers of subfolders themselves. Simply navigating through the file-system for files is a time consuming task.

One difficulty faced here was related to the files themselves. Most of the files have no extensions (e.g. .txt, .pdf, .sqlite, .jpg, .mp4, ...), or not the extension that would be expected (e.g. all cached images having extension ".cnt" instead of .jpg, .png, .gif, ...). In some circumstances this made it difficult to know what program to use to examine the file. In some cases the location of the file itself helped determine the program to use. For example, files found in the "databases" folder have a good chance of being SQLite database files. However this is not always the case.

Choosing a program is often a case of trial and error. Using Notepad++ and/or HxD is helpful for inspecting files for headers that could be used to identify their format. Another method that we find useful to identify a file type (very useful for media files especially) is to right click on the file, and select "MediaInfo" (v0.7.42BETA). This utility displays encoded media information in the file. It is important to note that one of the fields shown is the file format. We find this utility especially useful when trying to determine the format of various cached image files.

There are some text files scattered in different folders that contain somewhat useful information (e.g. name, email, uid, etc.), however this information is mostly redundant, as it is also found in SQLite database files. Most media files (shared images, map images, and audio clips) are found in the "cache" folder. The exception is that videos are found in the "files" folder. With exception to these, the vast majority of interesting information is found in SQLite databases, specifically "threads db2". This database contains information about all threads, their participants, and the content of all shared messages including the attachments, time-stamp, sender, shared location, message text, etc.

This is not true of the secret conversation thread. We are unable to recover plaintext messages or attachments corresponding to this conversation. It appears that Messenger is doing something right with regards to protecting the privacy of this conversation.

Another difficulty faced here is drawing a direct connection between cached images ("cache/image/..."), cached videos ("files/video-cache/..."), cached received voice clips ("cache/audio/...") and messages with media attachments. The difficulty arose because the cached images, videos, and received voice clips do not have time-stamps corresponding to when a message was sent (there are a few exceptions). With respect to the images, this observation would only be relevant if the image is taken within the messenger application (meaning the image is taken, and immediately sent). If a picture is taken with the camera application, and sometime later attached to a message, the time stamps of when the picture is taken and when a message is sent would not match anyways.

We are, however, able to draw a direct connection between cached sent voice clips and messages. Sent voice clips including an "Encoded date" tag, which corresponds directly to the time-stamp of the message it is sent in. This observation is important, as it allows the investigator to determine exactly when this audio message is created and sent.

The attachment information describing what is attached to a message (if it is an image, video, or voice clip) provides a URL (not for audio clips) pointing to the media file, and does not appear to reference a valid file name found in the backup. The URL, however, is not valid for shared images ("URL signature expired" error message). Therefore, one cannot directly infer what cached image or received audio clip corresponds to each message with attached media. There are several URLs corresponding to each video attachment. The video is not accessible, however a preview is. This preview shows the first frame of the video. Therefore, by comparing this image to stored videos, it is possible to determine which video it corresponds to, and therefore which video is sent/received in that message.

Instead of directly connecting media files to messages, one can try to infer which image/video/audio-clip is sent/received based on the context of the messages in the same time frame.

A second resource related to media files is the recovered images/videos taken by the camera application (recovered using ADB). These might be considered a tangent to the Messenger analysis, as they are not stored directly by the messenger application. However, for completeness we will briefly mention them. It is important to note that photos taken by the Camera application include GPS location tags (that specific setting is enabled in the camera settings), while photos taken within the Messenger application do not (nor do any of the cached images, whether they are taken by the messenger app or not). Therefore, this embedded data within the image files can be very valuable in determining where the user has been. Also, the embedded time-stamps in these recovered images and videos are accurate. Therefore, not only can one determine the users previous location, but also the time at which they are at that location.

These findings are substantial. The information recovered from the files described in Table 15.1 would allow an investigator to reconstruct group and private messages (but no secret conversations), which includes: All message text, images sent/received, audio clips sent/received, location information sent/received, and the time messages are sent/received based on timestamps.

If all of the above information is available, then an investigator would be able to reconstruct where a subject has been (GPS location information and photo), when they are there (timestamps), who they are talking to (thread information), and what is said (text and audio clips). Even a subset of this information could prove to be valuable in a forensic investigation.

The analysis above shows that Facebook Messenger stores a significant amount of personal information locally in smartphone storage. If users make use of the many features of this application (excluding secret conversations), data traces will be stored, which can subsequently be recovered by a forensic investigator.

Hangouts Analysis We will now analyze all subfolders and files found in the“/data/data/com.google.android.talk/” directory, which is where all of Hangouts’ private application data is stored. There are fewer subfolders (eight in total) found in this directory compared to the Messenger analysis, however, there does not appear to be less available information. The recoverable data found in these folders and files is organized in a way that makes parsing it fairly straightforward. In our opinion, the analysis is easier than that of Facebook Messenger. See Table 15.2 for a summary of important recovered data artifacts.

We encounter some similar difficulties with the Hangouts analysis as we do with the Messenger analysis. Many files are stored without extensions, making it difficult to know what program should be used to analyze them. Again, we use a combination of trial and error, along with Notepad++ and/or HxD to analyze file headers, and the “MediaInfo” utility to look for encoded media meta-data. This shows that data analysis is not as simple as just looking through files (at least not all the time). First the files must be opened with the appropriate application.

A second difficulty is the time and attentiveness required to analyze text documents. There are many files that have a combination of interesting data and extraneous information. The volume of the extraneous information can be somewhat

Table 15.2 Hangouts’ artifacts

Path	Data
cache/image manager disk cache/	Shared images (sent/received)
cache/scratch/	Shared images (sent)
databases/	Message content: author, text, time-stamp, remote URL (shared image), stream URL (shared video), latitude and longitude, address

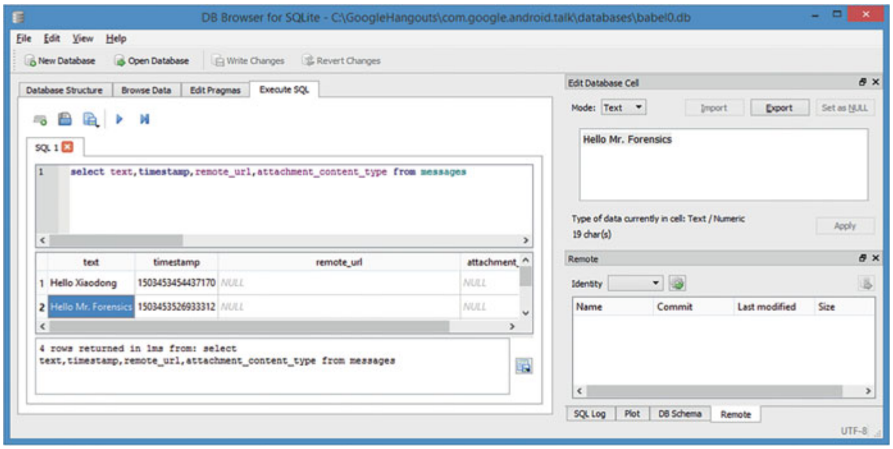
overwhelming. These files often lack “nice” formatting, which makes parsing them for relevant information a tedious task.

To help with this issue, it is helpful to consider the context of the file, and to search for specific strings within the file. For example, in the folder “cache/compressed call logs/ . . ./”, there are log files. There is sufficient information in the file path alone to allow us to create some search string ideas. We know that these logs are related to calls made with the Hangouts application. This will be either voice or video calls. Some information that an investigator might like to know would be: When was the call place? Who was the recipient? etc.

Therefore, some search string parameters could include date key words (month, year, etc.), names (other files give contact information that could be used here), etc. Once interesting information is found, the data immediately surrounding it can also be analyzed for data fragments. This method allows for a quick review of files for interesting information. A more thorough analysis can be done after (if necessary).

Remember that it is somewhat difficult in the Messenger analysis to directly link cached shared images and videos to the message they are attached to. This is not the case with Hangouts. In fact, the cached files don’t even need to be considered to do this (although they are still an important resource). Each message that has an image attachment also has a URL pointing to that image. Similarly, each message containing a video attachment has two URLs: “remote url” pointing to an image of the first frame of the video; The actual video (“stream url”). You simply have to copy and paste the URL into a browser to view/watch the image/video corresponding with the message. The same is true for shared location maps.

The databases/subfolder mainly contains Babel#.db file (SQLite database), for example, babel0.db, babel1.db, etc. The layout and contents of the “babel#.db” database (“babel0.db” on our test device) make the data analysis process quite straightforward. All shared images, videos, location, and message details are available in this one database. For example, as shown in Fig. 15.8, a table named



```
select text,timestamp,remote_url,attachment_content_type from messages
```

	text	timestamp	remote_url	attachment_
1	Hello Xiaodong	1503453454437170	NULL	NULL
2	Hello Mr. Forensics	1503453526933312	NULL	NULL

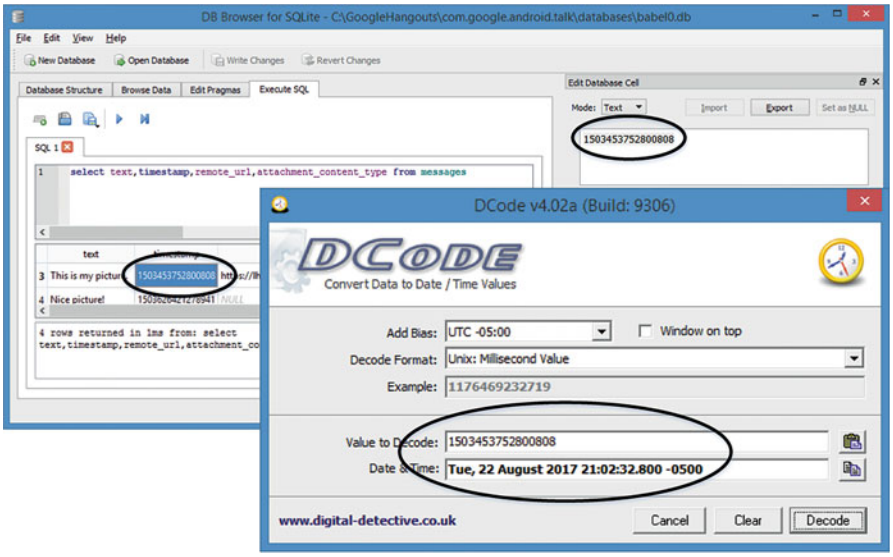
```
4 rows returned in 1ms from: select
text,timestamp,remote_url,attachment_content_type from messages
```

Hello Mr. Forensics

Type of data currently in cell: Text / Numeric
19 char(s)

Name	Commit	Last modified	Size

Fig. 15.8 Messages → text


DB Browser for SQLite - C:\GoogleHangouts\com.google.android.talk\databases\babel0.db

```
select text,timestamp,remote_url,attachment_content_type from messages
```

	text	timestamp	
3	This is my pictur	1503453752800808	https://lh
4	Nice picture!	1503626421278941	NULL

```
4 rows returned in 1ms from: select
text,timestamp,remote_url,attachment_co
```

Edit Database Cell — Mode: Text — 1503453752800808

DCode v4.02a (Build: 9306) — Convert Data to Date / Time Values

Add Bias: UTC -05:00
Decode Format: Unix: Millisecond Value
Example: 1176469232719
Value to Decode: 1503453752800808
Date & Time: Tue, 22 August 2017 21:02:32.800 -0500

www.digital-detective.co.uk

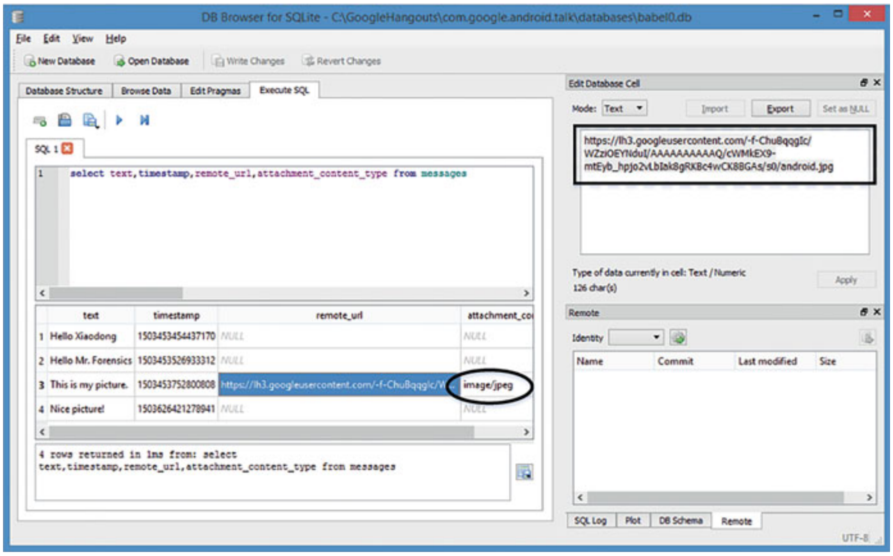
Fig. 15.9 Messages → timestamp

"messages" is used to store the history of many detailed conversations. Some important columns in the messages table include text, timestamp, remote_url, and author_chat_id. The text column is used to contain the content of the message (Fig. 15.8), while the timestamp column is the date/time formatted in a Linux epoch timestamp. This means it must be translated, using a tool like Dcode [16], into a human readable format (Fig. 15.9). The remote url column is a publicly accessible url that can retrieve images shared in the message (Figs. 15.10 and 15.11). Finally, we can also determine the author of a message by correlating the author_chat_id column with the chat_id column in the *participants* table in the same database. For example, the following query returns all the message texts and their authors as well as when these conversations took place

```
SELECT DISTINCT messages.text, participants.full_name, participants.
    fallback_name, messages.timestamp FROM messages, participants WHERE
    messages.author_chat_id = participants.chat_id
```

This database is an excellent place to start an investigation. It provides a detailed and organized overview of the IM application's state. Then, if an investigator needs to dig into something specific and needs more detail than the database provides, further analysis of Hangouts' files can be conducted.

Note that most of the information recovered is from private application storage. However, photo and video files recovered from the "sdcard" storage location supplement this information well. All images sent/received are available through URLs found in the "babel0.db" SQLite database, however, none of these photos include GPS tag information (even the ones that are taken by the smartphone camera



```
select text,timestamp,remote_url,attachment_content_type from messages
```

	text	timestamp	remote_url	attachment_co
1	Hello Xiaodong	1503453454437170	NULL	NULL
2	Hello Mr. Forensics	1503453526933312	NULL	NULL
3	This is my picture.	1503453752800808	https://lh3.googleusercontent.com/-f-ChuBqqglc/W…	image/jpeg
4	Nice picture!	1503626421278941	NULL	NULL

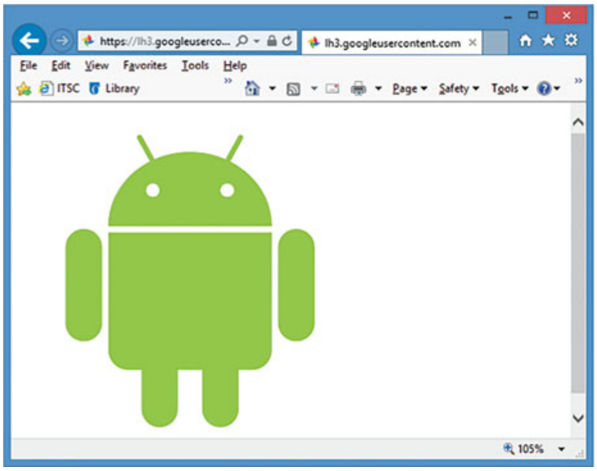
```
4 rows returned in 1ms from: select
text,timestamp,remote_url,attachment_content_type from messages
```

https://lh3.googleusercontent.com/-f-ChuBqqgIc/WZziOEYNduI/AAAAAAAAAAQ/cWMkEX9-mtEyb_hpjo2vLbIak8gRKBc4wCK8BGAs/s0/android.jpg

Type of data currently in cell: Text / Numeric
126 char(s)

Fig. 15.10 Messages → image URL

Fig. 15.11 Messages → image URL rendered


https://lh3.googleuserco...
lh3.googleusercontent.com
File Edit View Favorites Tools Help
ITSC
Library
Page
Safety
Tools
105%

app). Also, reference is made about sent/received videos, however no video files are found in private application storage. Nevertheless, they are available through URLs in “babel0.db”. It appears that any picture taken through the Hangouts application does not include GPS tag information, while pictures taken via the camera application do (with GPS location tags enabled in the camera app settings). However, these GPS tags are not included in the images accessible via the URLs. The only other place that pictures with GPS tags are found is the “cache/scratch/” directory, but not all of the sent photos can be found here. Therefore, to obtain all possible information from images (i.e. GPS tags) they should be recovered from external storage (emulated sdcard) and compared with those in private application storage.

A vast amount of information is stored by Google Hangouts. This application allows users to share location information, images, videos, send text, and do live voice/video calls among other things. As shown in Table 15.2, it stores information locally on the smartphone regarding each of these functions. Also, the application stores URLs where images, videos, and shared location maps can be accessed through a browser. Recovering this data can tell a great deal to a forensic investigator about the user of this application; Specifically about their previous activities. An investigator could determine: Who they have been communicating with, what they have said, where they have been, when they are there, when all types of communication are conducted, and what they may do in the future.

The ability to reconstruct a suspect's previous activities (and possibly predict future activities) is of great relevance to a forensic investigation. This analysis shows that a great deal of information is stored locally on a smartphone by the Hangouts application.

Review Questions

1. In Android terms, what is rooting?
2. What is the Android Debug Bridge (ADB)?
3. List at least five data acquisition methods for mobile phones.
4. What is JTAG?
5. In which default folder is Google Hangouts' data stored on an Android device?
6. Briefly explain how to identify the author of a message in Google Hangouts.
7. Describe in your own words, how Custom Recovery Image works? Why this method is chosen to extract data in this chapter?
8. In ADB, which command is used to copy files from an Android device to your computer?

15.3 Practice Exercise

The objective of this exercise is to practice mobile phone forensics. Particularly, you will learn how to acquire data of an Android application and analyze the data.

15.3.1 Setting Up Practical Exercise Environment

Step 1: Download and install Android Studio

In the absence of a physical Android device, we will create an Android Virtual Device (AVD) to emulate one, for example, Nexus 5X. AVDs are essentially emulators that emulate real Android devices so developed Android applications can be tested without the necessity to install the applications on a physical Android based device. Nevertheless, we conduct a forensic analysis on an AVD in following lab exercises. As the Android device is emulated, we simulate a user's Google Hangouts over a short period of time. We then perform a forensic analysis to learn the user's chat history.

1. Download the "Android Studio" installer from the following website:
   https://developer.android.com/studio/index.html

Note: Be sure to install the JDK (Java Development Kit) first before you install and use the Android Studio for Android application development. You can download OracleJDK by going to

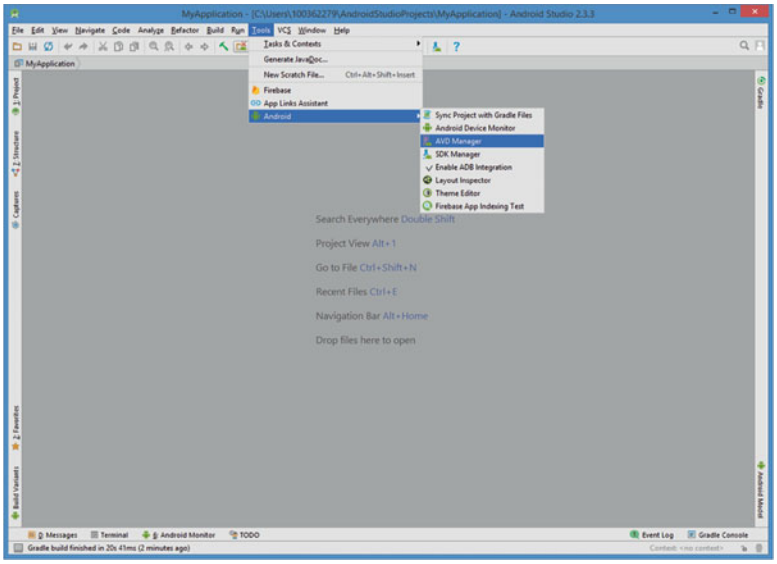
http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

Nevertheless, for your convenience, a distribution of Android Studio which includes all the tools you need to build apps for Android has been available for easy installation. You are recommended to download the one that Includes Android SDK.

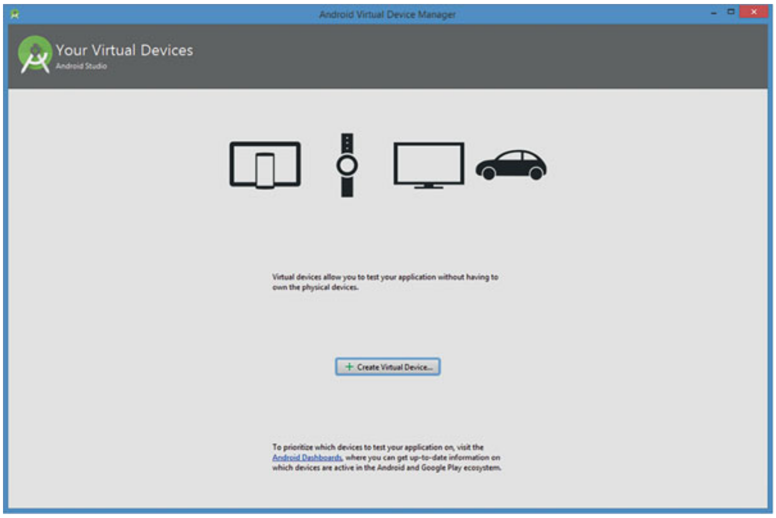
2. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings.

Step 2: Creating an Android Virtual Device (AVD)

1. Launch the Android Studio.
2. Once opening an existing project or completing the project creation, open the AVD Manager by clicking Tools > Android > AVD Manager.


MyApplication - [C:\Users\100362279\AndroidStudioProjects\MyApplication] - Android Studio 2.3.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication
Tasks & Contexts
Generate JavaDoc...
New Scratch File... Ctrl+Alt+Shift+Insert
Firebase
App Links Assistant
Android
Sync Project with Gradle Files
Android Device Monitor
AVD Manager
SDK Manager
Enable ADB Integration
Layout Inspector
Theme Editor
Firebase App Indexing Test
Search Everywhere Double Shift
Project View Alt+1
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open
0: Messages
Terminal
6: Android Monitor
TODO
Event Log
Gradle Console
Gradle build finished in 20s 41ms (2 minutes ago)

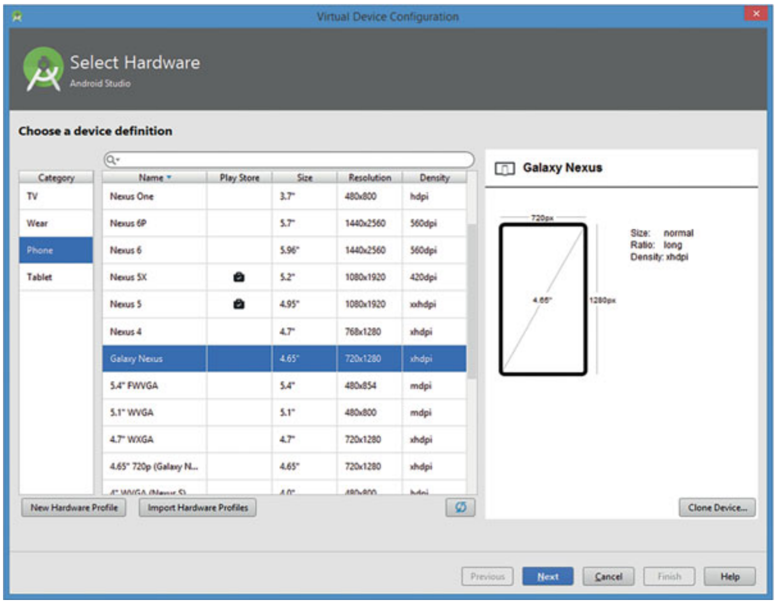
3. Click on Create Virtual Device.



Virtual devices allow you to test your application without having to own the physical devices.

To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.

4. Select a hardware profile, for example, Galaxy Nexus, and then click on Next.



Choose a device definition

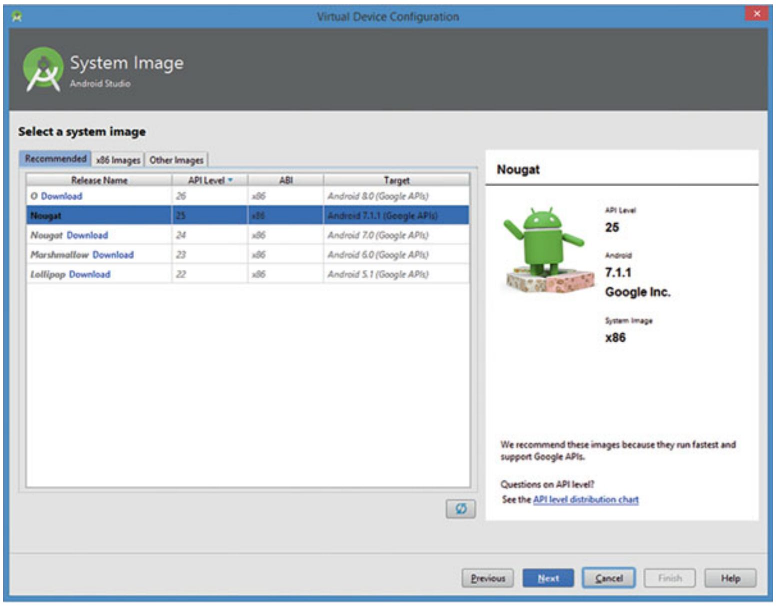
Category
TV
Wear
Phone
Tablet

Name	Play Store	Size	Resolution	Density
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2560	560dpi
Nexus 6		5.96"	1440x2560	560dpi
Nexus 5X	●	5.2"	1080x1920	420dpi
Nexus 5	●	4.95"	1080x1920	xxhdpi
Nexus 4		4.7"	768x1280	xhdpi
Galaxy Nexus		4.65"	720x1280	xhdpi
5.4" FWVGA		5.4"	480x854	mdpi
5.1" WVGA		5.1"	480x800	mdpi
4.7" WXGA		4.7"	720x1280	xhdpi
4.65" 720p (Galaxy N...		4.65"	720x1280	xhdpi
4" WVGA (Nexus S)		4.0"	480x800	hdpi

Galaxy Nexus

Size: normal
Ratio: long
Density: xhdpi

5. In the Select a System Image dialog box, the system image for a particular API level, and then click on Next.



Select a system image

Recommended | x86 Images | Other Images

Release Name	API Level	ABI	Target
O Download	26	x86	Android 8.0 (Google APIs)
Nougat	25	x86	Android 7.1.1 (Google APIs)
Nougat Download	24	x86	Android 7.0 (Google APIs)
Marshmallow Download	23	x86	Android 6.0 (Google APIs)
Lollipop Download	22	x86	Android 5.1 (Google APIs)

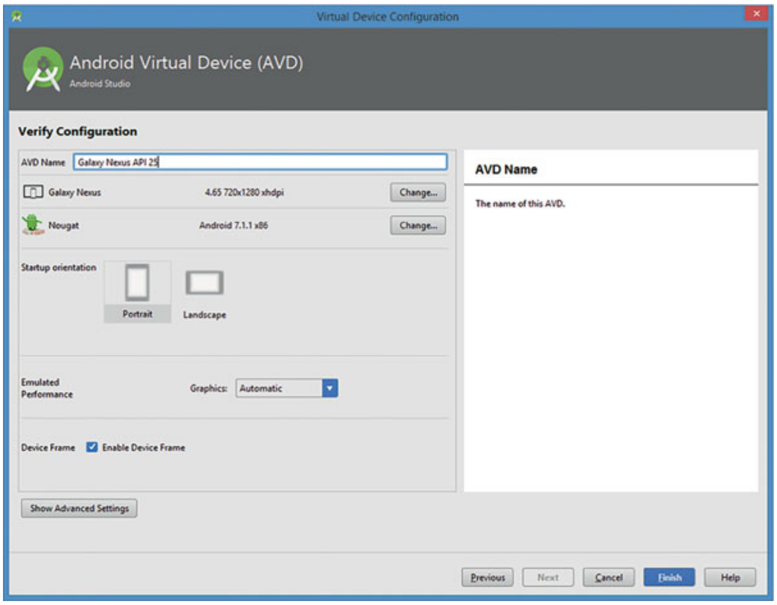
Nougat

API Level
25
Android
7.1.1
Google Inc.
System Image
x86

We recommend these images because they run fastest and support Google APIs.

Questions on API level?
See the API level distribution chart

6. Continue through the prompts and accept License Agreement.
7. On the Finish page specify the AVD Name and validate the AVD Setting to ensure it's correct then click Finish.



Verify Configuration

AVD Name: Galaxy Nexus API 25

Galaxy Nexus 4.65 720x1280 xhdpi

Nougat Android 7.1.1 x86

Startup orientation: Portrait

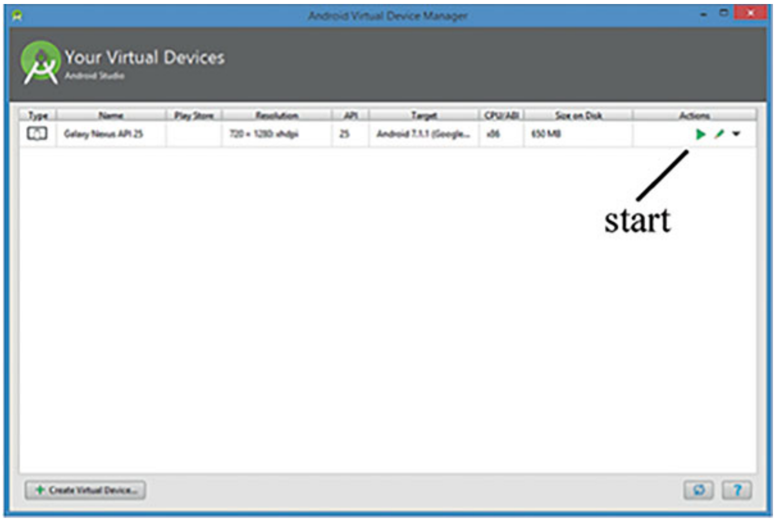
Emulated Performance Graphics: Automatic

Device Frame ✓ Enable Device Frame

AVD Name

The name of this AVD.

8. Once you have completed the creation of your AVD, the AVD is ready for use.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Galaxy Nexus API 25		720 × 1280: xhdpi	25	Android 7.1.1 (Google...	x86	650 MB	

Step 3: Configuring Android Virtual Device (AVD)

1. Launch the AVD in the emulator (Fig. 15.12).
2. Install Google Hangouts on the AVD.

   Note: If Google Hangouts is not preinstalled on the AVD, you must install it.

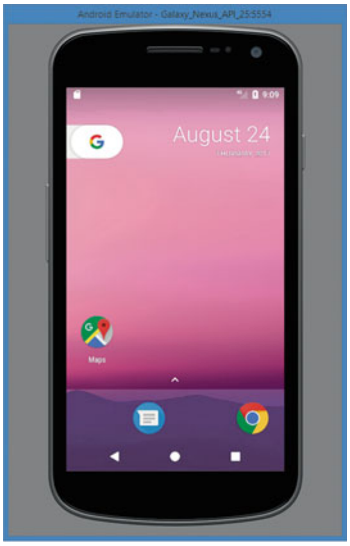
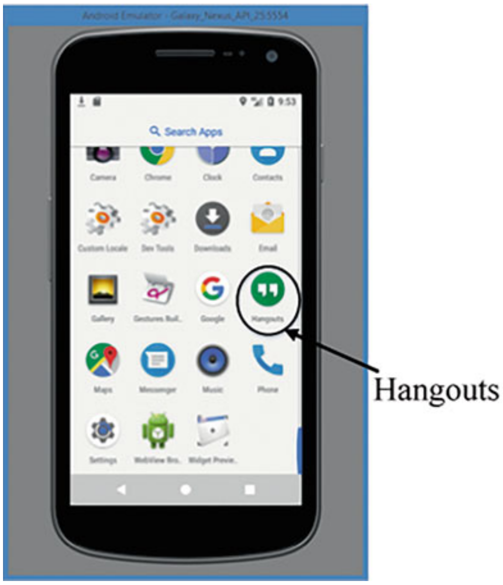


Fig. 15.12 Running emulated Android virtual device

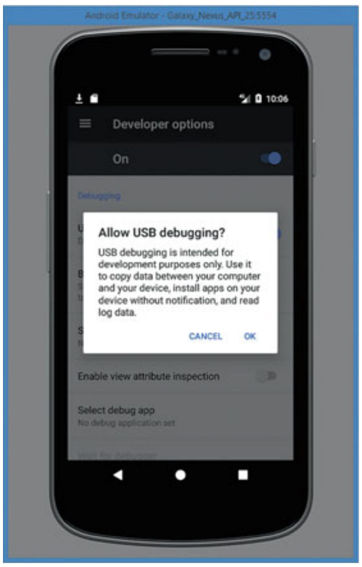


3. Enable USB Debugging on the AVD and Root the AVD.

Note: In order to acquire the data of Google Hangouts using ADB, we need to enable USB Debugging on the AVD. There are many online resources for learning how to enable USB debugging and root an android device (or AVD). For example, you can refer to the following link for the instructions on how to access Developer Options and enable the USB Debugging option in Android 7 Nougat:

http://www.neuraldump.com/2017/05/how-to-enable-developer-options-in-android-7-nougat/

Also, we need to root android emulator so we can gain access to the stored chat history in Google Hangouts.



Developer options

On

Debugging

Allow USB debugging?

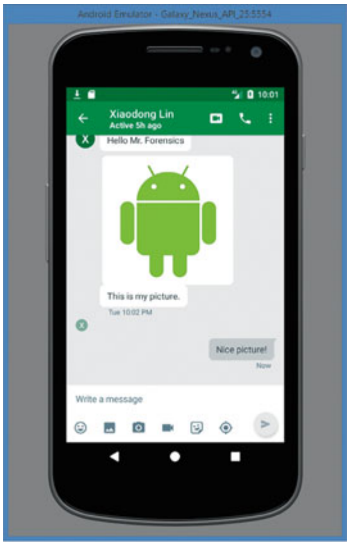
USB debugging is intended for development purposes only. Use it to copy data between your computer and your device, install apps on your device without notification, and read log data.

CANCEL OK

Enable view attribute inspection

Select debug app
No debug application set

4. To gather data for analysis, we simulate a user's normal use of Google Hangouts over a short period of time. For example, you create a Test Google account. Then, you sign into your Test account to chat with friends, such as, your own Google account. In other words, you message your own Google account. This includes sending/receiving textual messages and images.



Hello Mr. Forensics

This is my picture.
Tue 10:02 PM

Nice picture!
Now

Step 4: Install the DB Browser for SQLite

1. Download the "DB Browser for SQLite" installer from the following website:
   http://sqlitebrowser.org/
2. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings.

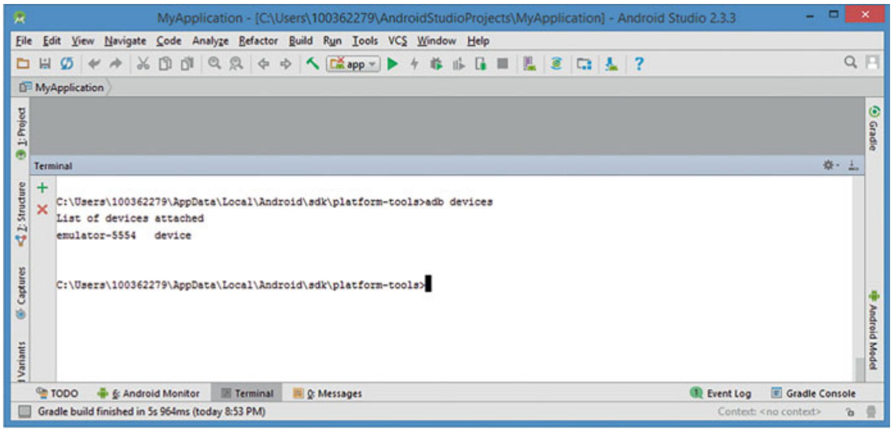
15.3.2 Exercises

Part A: Extracting Google Hangouts Chat History Using "adb pull"

As discussed in Sect. 15.2.4.4, Google Hangouts Chat History can be found in the following location:

/data/data/com.google.android.talk/

1. Verify that your AVD has been successfully started by running “adb devices”.

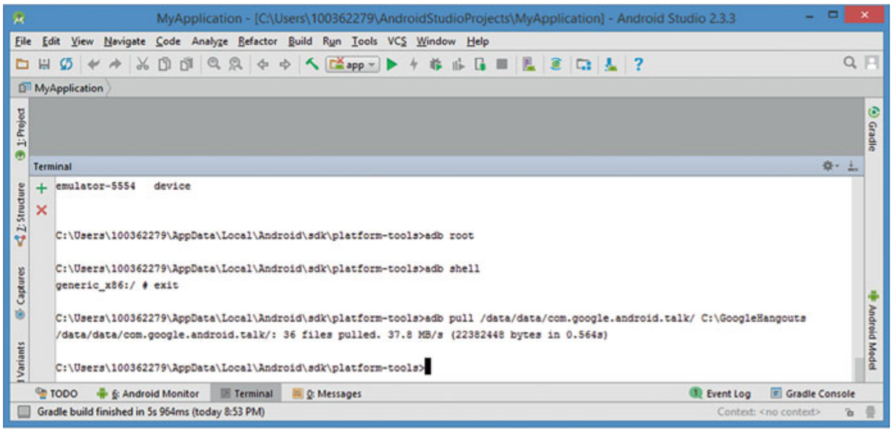

MyApplication - [C:\Users\100362279\AndroidStudioProjects\MyApplication] - Android Studio 2.3.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
MyApplication
Terminal
1: Project
2: Structure
Captures
Variants
Gradle
Android Model
TODO
6: Android Monitor
Terminal
0: Messages
Event Log
Gradle Console
Gradle build finished in 5s 964ms (today 8:53 PM)
Context: <no context>

```
C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb devices
List of devices attached
emulator-5554   device

C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>
```

2. Pull all the files under the package “/data/data/com.google.android.talk”.

   adb pull /data/data/com.google.android.talk/ C:\GoogleHangouts

   where /data/data/com.google.android.talk is the source location where Google Hangouts stores its chat history and C:\GoogleHangouts is the destination location where extracted Hangouts chat history is stored.


MyApplication - [C:\Users\100362279\AndroidStudioProjects\MyApplication] - Android Studio 2.3.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
MyApplication
Terminal
1: Project
2: Structure
Captures
Variants
Gradle
Android Model
TODO
6: Android Monitor
Terminal
0: Messages
Event Log
Gradle Console
Gradle build finished in 5s 964ms (today 8:53 PM)
Context: <no context>

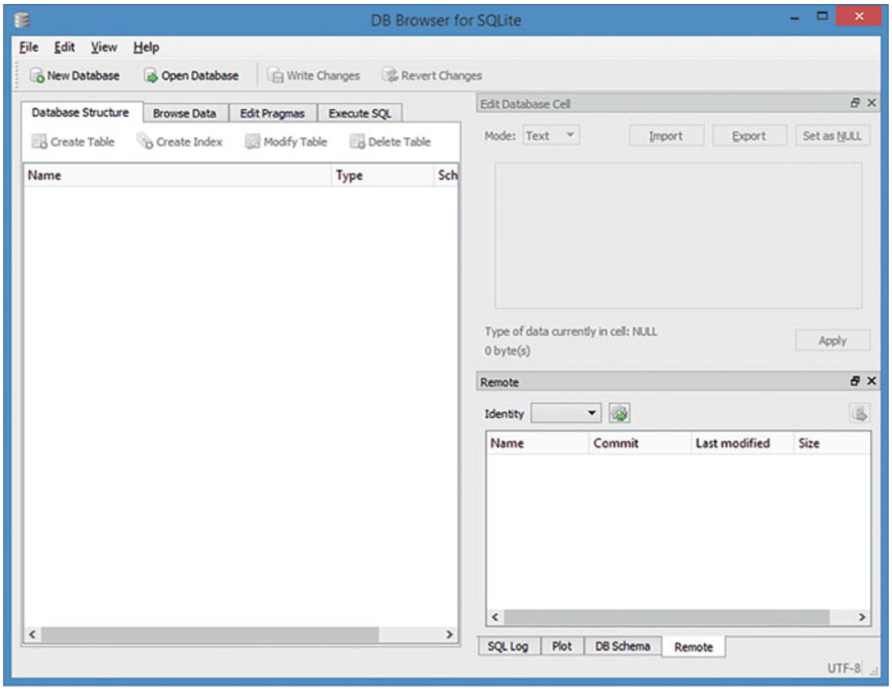
```
emulator-5554   device

C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb root

C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb shell
generic_x86:/ # exit

C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb pull /data/data/com.google.android.talk/ C:\GoogleHangouts
/data/data/com.google.android.talk/: 36 files pulled. 37.8 MB/s (22382448 bytes in 0.564s)

C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>
```

Part B: Analyzing Google Hangouts Chat History
Answer Google Hangouts chat analysis questions in the end. Note that the extracted data from the above steps may contain “databases” directory, which has the SQLite database files that contain your Hangouts chat history.

1. Navigating into C:\GoogleHangouts and Locating SQLite database files.

   Note that SQLite database files in Google Hangouts are named like "Babel#.db", for example, "Babel0.db", "Babel1.db", etc.

2. Extract the chat messages using DB Browser for SQLite.



Google Hangouts Chat Analysis Questions

Q1. How many messages or texts you have sent or received in Google Hangouts?

Q2. Find the first message or text you sent. What have you said (message text)? Who were you communicating with (contact/conversation participant information)? When did this conversation take place?

Q3. Find the last message or text you received. What has your contact said (message text)? Who were you communicating with (contact/conversation participant information)? When did this conversation take place?

References

1. androidcentral: http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide (Accessed Dec. 1st, 2016).
2. N. Scrivens, X. Lin: Android digital forensics: data, extraction and analysis. ACM TUR-C 2017: 26:1-26:10.

3. L. Zhang, F. Yu, Q. Ji: The forensic analysis of WeChat message. Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control, (2016): 500–503.
4. A. Hoog: Android forensics: investigation, analysis and mobile security for google android. Elsevier, 2011.
5. XJTAG: https://www.xjtag.com/about-jtag/what-is-jtag/ (Accessed Dec. 1st, 2016).
6. Z3X Easy JTAG: http://easy-jtag.com/ (Accessed Dec. 1st, 2016).
7. S.J. Yang, J.H. Choi, K.B. Kim, T. Chang: New acquisition method based on firmware update protocols for android smartphones, Digital Investigation. 14 (2015): S68–S76.
8. Android Developers: https://developer.android.com/guide/topics/providers/content-providers.html (Accessed Dec. 1st, 2016).
9. N. Al Mutawa, I. Baggili, A. Marrington: Forensic analysis of social networking applications on mobile devices, Digital Investigation. 9 (2012): S24–S33.
10. T. Vidas, C. Zhang, N. Christin: Toward a general collection methodology for android devices, Digital Investigation. 8 (2011). S14–S24.
11. N. Son, Y. Lee, D. Kim, J.I. James, S. Lee, K. Lee: A study of user data integrity during acquisition of android devices, Digital Investigation. 10 (2013): S3–S11.
12. H. Srivastava, S. Tapaswi: Logical acquisition and analysis of data from android mobile devices, Information & Computer Security. 23.5 (2015): 450–475.
13. N. A. Matuwa, I. Baggili, A. Marrington: Forensic analysis of social networking applications on mobile devices. Digital Investigation, 9(2012): S24–S3.
14. A. Mahajan, M.S. Dahiya, S.P. Sanghvi: Forensics analysis of instant messenger applications on Android devices. International Journal of Computer Applications, 68(2013): 38–44.
15. N. S. Thakur: Forensic analysis of WhatsApp on Android smartphones. M.Sc. thesis, University of New Oeleans, New Orleans, LA, Aug. 2013.
16. C. Anglano, M. Canonico, M. Guazzone: Forensic analysis of the ChatSecure instant messaging application on android smartphones, Digital Investigation 19 (2016): 44–59.
17. D. Walnycky, I. Baggili, A. Marrington, J. Moore, F. Breitinger: Network and device forensic analysis of android social-messaging applications, Digital Investigation. 14 (2015): S77–S84.
18. C. Anglano: Forensic analysis of whatsapp messenger on android smartphones, Digital Investigation. 11 (2014): 201–213.
19. N. A. Barghuthi, H. Said: Social networks IM forensics: encryption analysis. Journal Communications, 2013;8(11).
20. F. Karpisek, I. Baggili, F. Breitinger: Whatsapp network forensics: decrypting and understanding the Whatsapp call signaling messages. Digital Investigation, 15(2015): 110–118. https://doi.org/10.1016/j.diin.2015.09.002 special Issue: Big Data and Intelligent Data Analysis. http://www.sciencedirect.com/science/article/pii/S174228761500098.
21. F. Zhou, Y. Yang, Z. Ding, G. Sun: Dump and analysis of android volatile memory on wechat. IEEE International Conference on Communications (ICC), 2015: 7151–7156. https://doi.org/10.1109/ICC.2015.7249467.
22. C. Silla, Wechat forensic artifacts: Android phone extraction and analysis. Master's thesis. Purdue University, 2015.
23. S. Wu, Y. Zhang, X. Wang, X. Xiong, L. Du: Forensic analysis of WeChat on Android smartphones. Digital Investigation, 21 (2017): 3–10.
24. G. B. Satrya, P. T. Daely, and S. Y. Shin. Android forensics analysis: Private chat on social messenger. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 430–435, July 2016.
25. G. B. Satrya, P. T. Daely, M. A. Nugroho: Digital forensic analysis of telegram messenger, International Conference on Information, Communication Technology and System (ICTS), 2016.
26. TeamWin — TWRP: https://dl.twrp.me/Z00T/twrp-3.0.2-4-Z00T.img.html