Passe-Partout: a General Collection Methodology for Android Devices

Daniel Votipka, Timothy Vidas, and Nicolas Christin*, *Carnegie Mellon University*
{dvotipka,tvidas,nicolasc}@andrew.cmu.edu

***Abstract*—**
Like computers, mobile devices are both used to commit and are the subject of crimes. Digital forensics collection and analysis techniques need to adapt to cope with the unique characteristics of mobile devices. Fortunately, the rapid adoption of such devices has also resulted in a relatively homogeneous set of software platforms. In this paper, we describe a general methodology for performing digital forensic data collection on Android-based devices. By re-purposing a special Android boot mode, comprehensive extraction of evidence, with minimal potential for data corruption or omission, is possible.

EDICS Categories: FOR-EVID, HWS-FORE

I. Introduction

Modern mobile devices not only rival budget computers in terms of computing power, but are also capable of storing and producing substantial amounts of information. Among these modern mobile devices, the Android operating system has recently become the operating system of choice. The capabilities of these mobile devices encourage rapid adoption in enterprise and consumer settings; the open nature of Android facilitates scientific investigation and reproducibility.

Like computers, mobile devices are used to commit and are the target of crime. However, methods of performing criminal investigation on mobile devices remain immature. The increased presence of modern mobile devices necessitates the creation of tools and methods for incorporating mobile device data in investigations. Accordingly, the ability to collect and analyze mobile device data is a vital requirement to overcoming forthcoming investigative challenges [23].

To a large extent, forensic data collection is an "offensive" activity: to collect all the data that may be needed for further investigation, one needs to acquire elevated privileges on the device where data of interest is stored, often without the assent of the owner of the device. Worse, on modern mobile devices, very often the owner of the device herself does not even possess these elevated privileges. That is, unless she has "rooted" her phone, she may not, herself, have the level of access needed to perform sound forensic data collection. In other words, investigators must usually "break into" the device to collect the data they need [30]. Doing so without compromising the integrity of the data is a complex task.

While the issue of forensic data collection is not new, most practitioner-oriented forensics tools were developed out of necessity and thus focused exclusively on the dominant platform of the past two decades, that is, Microsoft Windows computers [23]. In contrast, the (historically) relatively small market share, and the hardware and software diversity of mobile phones has hindered the development of such tools for mobile devices.

The greater capabilities of smartphones – compared to traditional "feature phones" – bring added complexity. Today's mobile devices have much in common with computer systems. In fact, the two prevailing smartphone platforms, iOS and Android, are both based on contemporary, robust operating systems (OSX/FreeBSD and Linux, respectively). Nevertheless, the hardware and software used on these devices are undoubtedly different than the Windows computer for which current forensics tools and procedures are designed. For example, mobile devices do not share the modular hardware (e.g., readily removable RAM, hard disk drives) common to today's computer systems [23]. Mobile devices may include removable SD cards for memory expansion, which can be easily analyzed through techniques similar to those employed on traditional computer systems, however, they are only secondary storage units and many manufacturers are moving away from their use [1], [2]. Similarly, mobile devices often employ unfamiliar filesystems and use varying low-level protocols to access data storage areas that make better use of the embedded non-volatile memory. These inherent differences impair an examiner's ability to properly investigate crimes involving mobile devices through current means and require the creation new techniques that can effectively handle the new challenges presented by mobile devices.

The primary contribution of this paper is a general method for collecting forensically sound evidence from Android based devices. Our method takes advantage of the common middleware Android provides to offer image collection on a wide range of devices. Specifically, our work builds on the recovery image collection technique originally proposed by Vidas et al. for the Motorola Droid [32] and confirmed by James [28], and generalizes it to a comprehensive set of different Android devices, with different Android versions and markedly different underlying hardware. We test our method with 15 different devices. Using this technique, we are able to collect data from evidence-rich storage locations without any concerns of evidence tampering.

The remainder of this paper is organized as follows. We first provide some background on Android, forensic data collection, and related work in Section II. In Section III, we formalize the definition of forensic soundness through a set of objectives for data collection. In Section IV, we discuss creation of a custom recovery image to perform forensic data collection on Android. In Section V, we provide a comparative analysis of the application of our method to a set of fifteen different devices. We further elaborate, in Section VI, on how one can adapt our forensic techniques to other Android devices. We discuss implications of this work and conclude in Section VII.

II. Background and Related Work

Forensics is the application of a science to inform decisions in a legal process. Likewise, digital forensics is the application of computer sciences in this manner. Many process models have been suggested to formalize the concept of digital forensics. Two steps seemingly appear in every proposed model: collection and analysis. Collection of evidence is a necessary prerequisite for other, analytic steps [33]. Improper collection processes can make subsequent analysis impossible or even cause evidence to later be inadmissible in court.

Evidence can be collected in many ways, each often presenting trade-offs that must be weighed by the collector. The two representative methods are colloquially known as *logical* and *physical* collection. In either method, the collector must take steps to ensure the resultant duplication, or image, is collected in a sound manner. That is, the source evidence must only be altered to the minimal extent as required by the collection process. To this end, collectors often employ a hardware device engineered to prevent inadvertently altering evidence.

A physical collection results in the most complete acquisition of evidence. Physical collection is often referred to as a byte-for-byte, owing to a sequential copy process independent of higher data layers such as disk partitions or filesystems. A physical image is a prerequisite for some subsequent analysis techniques, such as the recovery of deleted files that are no longer known to a filesystem. On a typical computer, a physical image is typically created by first physically removing the hard disk drive from the computer. Then the device is duplicated using hardware or software (such as the UNIX disk duplicating utility, `dd`) to copy the smallest accessible storage units from the device.

Conversely, a logical image is created from a higher representation of data, such as files, or a filesystem. A logical image is commonly collected using software (such as the UNIX utility, `cp`) to copy only allocated data as maintained by a filesystem [25], [27]. As such, logical imaging necessarily results in a loss of evidence through omission, in this example *un*allocated data areas.

The creation of a logical image is typically faster than the creation of a physical image, often requires less post processing, and in some cases a logical image is sufficient for the investigation at hand. However, where possible, most industry guidance and legal precedent dictate that evidence be comprehensively preserved. Further, some later analysis methods require a physical image, such as the recovery of deleted data from unallocated data areas. Both are compelling cases for physical imaging.

In the context of mobile devices, many of the incontrovertible collection procedures informed by decades of experience do not apply. In an extreme case, a logical collection may consist of methodically photographing the device screen, thereby missing any information not accessible via the standard user interface [33]. Similarly, a physical collection is inhibited by the device form factor. Unlike a computer hard disk drive, the physical form of mobile devices does not facilitate non-destructive removal of storage components [21]. Even if the storage media can be physically accessed [20], the component interfaces are often unfamiliar to collectors [21], [34]. For example, flash storage is commonly used in modern mobile devices, and the storage is permanently affixed to the device circuit board. With storage techniques that differ from magnetic media and proprietary physical allocation algorithms [33], flash storage blurs the accepted definition of "physical image" in the digital forensics community.

In addition to the new form factor, mobile devices differ from traditional computers in the manner they are used. Computers are used to access external services (e.g., social networking, electronic mail) but with large displays and keyboard/mouse input, they are also used for general purpose activities. Unlike computers, most of the value placed on a mobile device comes from external services. For instance, location-aware services, telephone calls, and text messaging are used frequently on modern smartphones. In light of legal investigation, this external connectivity can be advantageous as information can be obtained from the service providers. However, externally obtained information may not be sufficient. A service provider may only maintain data for a short period of time [33] or may be difficult to work with, especially when involving multiple providers. Not knowing a priori the set of service providers from which to solicit information, the physical device might be the best source for deducing a complete picture of prior events. Additionally, some information will exist solely on the device, such as pictures not yet transmitted to a remote service. In any case, external data can be used to corroborate evidence found locally on the device.

Unfortunately, no general collection technique currently exists for mobile devices [23]. The past ten years have seen sporadic collection and analysis research in mobile device forensics. Research has produced a handful of custom tools for collecting mobile device evidence. Each tool is often compatible with a disjoint set of devices. In order to collect evidence from a given device, a practitioner must be proficient in using a set of these tools [33]. In the absence of a general, sound technique for collection, practitioners have resorted to attacking devices with questionable software exploits in order to gain enough permission to perform collection [17], [24], [26]. When successful, this practice certainly changes the state of the device, a far from ideal situation, since these modifications can compromise the integrity of the data on the device.

To provide full physical collection without the negative effects seen from exploiting devices, Vidas et al. proposed *recovery image-based collection* [32]. In recovery based collection, the examiner flashes a custom recovery image that contains all their forensic tools onto the device's recovery partition. Since the recovery mode is not entered in normal use, the storage locations being collected are not modified in any way. The collected data is therefore unaltered from its original state, alleviating any concerns of evidence tampering. However, recovery image based collection has so far, only been successfully tested on Motorola Droid, using tools specific to that particular device.

Current collection and analysis methods are plagued by the diversity of previous generation mobile devices. The ubiquity of Android in mobile devices brings unprecedented commonality in device software and hardware. This commonality can be leveraged to overcome existing data collection and analysis problems. Most of the work to make tools available to large numbers of Android devices has been accomplished by the modding community. The purpose of the modding community is to utilize the open source code provided by Android to produce modified versions of the Android experience [10]. Our research differs from the work done in the modding community as their goal is to replicate the functionality of the entire Android framework, but we only attempt to replicate the functionality of the recovery image, which is a significantly smaller set of features.

III. Collection Objectives

We next present a methodology for collecting evidence from Android devices in a forensically sound manner. The collected data must facilitate subsequent analysis as performed by current and future tools and procedures. Moreover, any data omitted or corrupted during collection may cause the integrity of an investigation to be jeopardized or entirely obviate later analysis. Altering the device data may be considered akin to destruction of evidence, therefore collection procedures that modify data stores potentially containing evidence are undesirable. The methodology must adhere to the following general criteria, applicable to any forensic collection:

Data preservation: Ideally, all resident data can be collected from a device, even metadata and memory areas thought to be deleted or empty.

Atomic collection: During normal device operation, data is continuously being written to and deleted from memory. When attempting a comprehensive copy on an active device, the resulting image will be composed of data collected through many states of memory. For example, in the midst of a sequential copy operation data may be orthogonally moved from the end of memory to the beginning resulting in that data not appearing in the resultant image. Such an image may complicate analysis and in rare cases may not even represent a set of data that is even valid for the device.

Correctness: The methodology must account for the accurate collection of data from the device memory and accurate transfer to the ultimate destination.

Determinism: The collection process must be repeatable and produce expected output.

Evidence preservation: The collection process should not physically damage the device. Physical evidence should be preserved so that collection can be repeated if necessary to ensure the correctness and determinism of the process.

Usability: Since the methodology is intended to be used broadly by practitioners, the process must be easy to use and must be able to conclude in a reasonable span of time.

Vetting ability: All software and techniques used for collection should be verifiable or at the least taken from a trustworthy source.

Reproducibility: Given a device that the collection process has never been run on before, collection on the device should be possible with only minimal effort.

Our technique, based on Vidas et al.'s work [32], involves repurposing a special Android boot mode known as *recovery mode*. Standard Android devices have several partitions that house different types of data. For example, the *system* partition contains parts of the operating system, manufacturer and vendor added features, and applications that the user is not meant to modify. On the other hand, the *userdata* partition contains application specific data. The *recovery* partition typically contains seldom used software that can perform maintenance tasks. Common recovery mode tasks include returning the device to a stock configuration or applying a software update that cannot be applied while the system is executing in its standard operating mode. To accomplish such tasks, the recovery software is executed by circumventing the normal boot procedure A device enters this recovery mode by way of special key presses during the early stages of the boot process.

For digital forensics collection, we devise special recovery software to replace the incumbent software in the recovery partition. Many qualities of the standard recovery software are desirable for collection purposes. For instance, the code that enables a mobile phone's radio is never loaded so there is no risk of communications with the cellular tower altering data on the device. Even so, the standard boot image does not possess some components essential for data collection, primarily methods for extracting data from device memory and a methods for transferring collected data from the device.

Since the recovery software is Android-based, the obvious tools for data duplication would be standard Linux tools such as `cp` or `dd`. As discussed above, using `cp` would necessarily perform a logical collection omitting unallocated spaces, and in the case of modern mobile devices `dd` would also result in an incomplete collected image. While transparent to the typical user, the use of flash memory chips for data storage can create critical problems for forensics collection. Data copied using `dd` will omit memory areas known as *spare* or *out-of-band*, which is used to store metadata about associated memory segments. By employing a tool specifically designed to interpret flash memory, an image can be collected that contains more information than could be collected via standard Linux tools.

In order to transmit collected data from the device without enabling wireless radios, we employ an Android debugging tool. The Android Debug Bridge (ADB) is often used across USB to debug physical devices. While not available in the standard recovery software, the device-specific components can be added to our special recovery software. In order to interact with the device, an investigator may download or build[1] the complementary computer-specific ADB components. Then, utilizing ADB in recovery mode, an investigator can interact with the recovery software using a computer, and perform data collection using TCP over USB.

As shown in Figure 1, once crafted, the collector must transfer the collection-oriented software to the device. This transfer requires device-specific software tools that overwrite the existing recovery partition with the new software. While this process does unavoidably overwrite data only in the recovery partition, this partition is extremely unlikely to contain potential evidence. The collector then reboots the device and directs it into recovery mode. Once executing in the context of the new recovery software, data is collected using ADB and the bundled flash duplication tools.

When booted into recovery mode, partitions other than the recovery partition are not modified as would typically occur during normal operation. By using recovery mode for forensic collection, data in other partitions can be duplicated with no risk of data in those partitions being altered during collection. Similarly, since data in other partitions is never altered, repeated collection results in identical collected images. By utilizing special duplication software, this collection method collects more data than standard data duplication tools, particularly capturing flash out-of-band areas.

From the perspective of a practitioner, there is little need to fully understand the inner workings of the recovery software, or even how to construct such software. Instead, the practitioner may be concerned only with obtaining vetted recovery software and understanding how to perform the collection process. Thus, the collection process does not require specific expertise and, in practice, tested devices sustain approximately 4.3 MB/s allowing for full collection to occur in a reasonable amount of time.

IV. Android Recovery Image

To generalize the recovery method for Android, we focus on two key components: the recovery image and the flashing mechanism. In this section we will further discuss the components of the recovery method and give the steps taken to create a custom image

A. Recovery Image Structure

The recovery image consists of three main components: the header, kernel, and the ramdisk.[2] The Android *bootimg* structure can be seen in Figure 2.

Header. The header is placed at the start of the image and provides essential metadata to the bootloader for loading the other components when booting into recovery mode. The structure of the header is given in *bootimg.h* [3] in the Android source.

[1]Source code and pre-built executables are available at http://developer.android.com.

[2]A secondary ramdisk is optional, however it is unnecessary to our application and for simplicity has been omitted from our discussion

[3]*bootimg.h* can be found at *<path to source tree>/system/core/mkbootimg/bootimg.h*

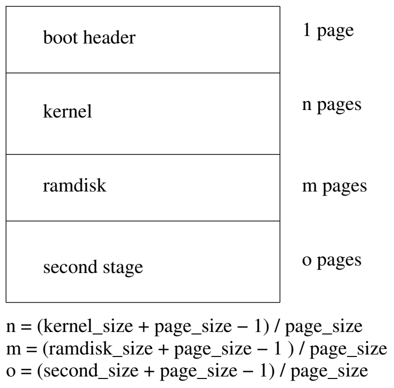
boot header
1 page
kernel
n pages
ramdisk
m pages
second stage
o pages
n = (kernel_size + page_size – 1) / page_size
m = (ramdisk_size + page_size – 1) / page_size
o = (second_size + page_size – 1) / page_size

Fig. 2. **Bootimg Structure**: structure of a common Android *bootimg*

Name	Description
default.prop	Default Build Properties
init.rc	System-wide initialization values
init	Initialization executable
system/	System files and tools
sbin/	Additional tools
sbin/adb	Android Debug Bridge executable
sbin/recovery	Recovery executable
res/	Images for recovery mode

TABLE I
Important Ramdisk Files

The image header includes a magic signature, the size and memory location in which to load the kernel, ramdisk, and secondary image, the physical address of any kernel tags, the image's page size, command line options, and a checksum.

Kernel. The second segment of the recovery image is a gzipped kernel. The kernel stored in the recovery image is a customized variant of the Linux 2.6 kernel and acts as a mediary between the Android framework and the hardware, memory management, process management, the network stack, and the driver model [5], so the kernel for each device is tuned to work with a specific set of hardware.

Ramdisk. The last segment of the recovery image is the ramdisk. The ramdisk contains the core files needed for system initialization. Table I gives the files and folders most commonly found in the recovery image and their usage. The most important files are the *init* binary, *init.rc*, which controls most system-wide properties, and the *recovery* binary, which is executed when the phone is booted into recovery mode [13].

Even though the image structure given above is defined by the Android framework, not every device uses the same image structure. Specifically, Samsung devices use a different method where the ramdisk is built into the kernel. However, the components for each method remain the same, the only difference is in the way they are packaged.

B. Custom Image

To build a custom recovery image, we split the process between the three components of the image: the header, kernel, and ramdisk.

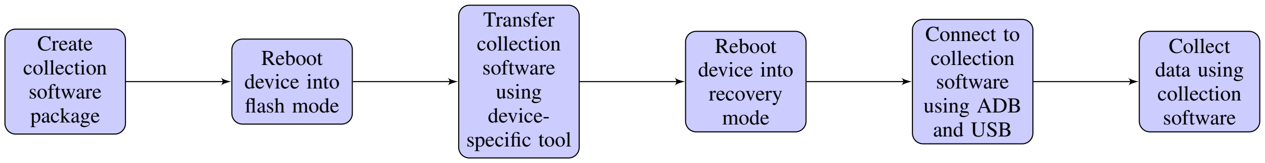
Create collection software package
Reboot device into flash mode
Transfer collection software using device-specific tool
Reboot device into recovery mode
Connect to collection software using ADB and USB
Collect data using collection software

Fig. 1. **Collection Process** First the collector creates or obtains collection software for the device. The collector then reboots the device into a special flash mode and uses a manufacturer-specific tool to transfer the collection software to the device. Next, the device is rebooted into recovery mode, which executes the collection software. The collector then connects a computer to the device using a USB cable and the Android Debug Bridge (ADB) software. Finally, data is duplicated to the computer using the collection software.

Header. To generate a header, we need to select values for the size and loading location of the kernel, ramdisk, and secondary ramdisk, the address of the kernel tags, image page size, command line options, and checksum. We omit the magic signature in our discussion because it is constant across all images.

The sizes for the kernel, ramdisk, and secondary ramdisk can be calculated once we create the other custom components[4]. The loading location for each component is generated as a standard offset from a base address[5] (given in *mkbootimg.c*[6]). Therefore, we only need to select a single base offset. Additionally, the checksum is calculated over all our custom components.

To select the correct values for the base address, page size, and command-line options, we can either extract them from another working image's header or from a kernel built for our device.

Kernel. For our custom recovery image to work on a new device, we need a kernel that understands how to communicate with the specific hardware found on that device. Obtaining a working kernel can be done in one of two ways, building the kernel from source or extracting it from a working image.

Some manufacturers have made the kernel source for all their devices publicly available [14], [18]. For these devices, we can simply download their source and an ARM cross-compiler [19] and build to get a working kernel. Note, this method is preferable to extracting a kernel binary because the source can be read and analyzed to ensure that no malicious or unwanted actions are executed when running a kernel, whereas identifying unwanted effects becomes much more difficult when looking at a kernel binary [22].

For future analysis and comparison purposes, when building each device kernel for our test set we minimize the configuration so that only necessary components are included. Because our forensic application only requires a small set of features, we removed any unnecessary devices, such as audio, OpenGL, and networking drivers, and only included code necessary to run the recovery executable, access memory, and establish an *adb* connection over USB.

For devices where no source is available, we extracted our kernels from recovery images that are known to work for our target device. The first method obtains a "stock" image, the image that is provided by the manufacturers for that device. With the secondary device that is unrelated to the investigation and is the same make and model, a recovery image can be extracted using a rooting technique. The second method for obtaining a working recovery image is through the modding community. "Modded" images provide a modified Android experience for users that want to push the performance of their devices or add features not supported by Android. "Modded" images can be obtained from many sources, such as forums [4], [6], [11] and developer sites [3], [10]. Because of the strong Android development community, images for all devices can typically be found.

A stock image is preferable to an image from the modding community because Android and the manufacturer are assumed to test the images more thoroughly and are motivated by sales to ensure their code works with no negative effects, whereas "modders" may have other motivations which could lead to unwanted side effects.

Ramdisk. To generate a custom ramdisk we utilize the openness of the Android platform. Using the Android source, we simply modify a few configuration files to match the configuration of our device, and add our forensics tools into the build tree for inclusion into the image.

To build a ramdisk for a new device we have to add a device specific directory to the *device/* directory, which contains all the information necessary to build the Android source for a given device. Configuration information includes all the custom header and kernel information, and other make information. Note, to prepare for device build comparison and analysis we reduce the device configuration to its minimal set of necessary options, to avoid including any spurious information in our analysis.

To add our forensic tools to the ramdisk, we have to make two modifications: add custom busybox source to the build tree and update init.rc.

Busybox is an executable which contains size optimized versions of common Linux tools designed for resource limited systems [8]. By adding busybox to the build we provide the *adb* remote shell with commonly used tools such as *ls*, *cat*, *file*, etc with little effort. Also, busybox is open source and extendable [9], allowing us to add our forensic tools, such as *nanddump* and *transfer*, into the ramdisk easily.

The init.rc file is used to control system-wide properties, such as partition mounting and starting services [13]. For our forensic application, we modify the init.rc file to mount the partitions we want to image as globally readable, give our

[4]The size of the secondary ramdisk will always be zero because it is never used in our application.

[5]Kernel address = base + 0x0000800, ramdisk address = base + 0x01000000, secondary ramdsik address = 0x00F00000, kernel tags address = base + 0x00000100.

[6]*mkbootimg.c* can be found at *<path to source tree>/system/core/mkbootimg/mkbootimg.c*.

forensic tools root level permissions, and ensure that the adb service is running at startup.

C. Flashing Tools

Once we have created an image for the device, we then need to find a way to write the custom image to the recovery partition. For maintenance purposes, manufacturers tend to provide tools for flashing custom images.

Acquiring the Tool. If the tool is provided publicly (such as *fastboot* [7]), then it can simply be downloaded from the developer's website and used to write the recovery image. However, these tools are typically not open source and commonly are only built for the manufacture's operating system of choice (i.e. Windows).

Many of the proprietary tools have been reverse engineered [12], [16] by the modding community and distributed publicly for multiple host operating systems. Modded tools are easier to obtain and avoid the challenges presented by proprietary tools, however, they should be used with caution, as they are provided by a less reputable source.

When obtaining a flashing tool it is always recommended to obtain the proprietary tool directly from the manufacturer. The manufacturers understands the protocol best and reverse engineered tools should only be relied on if no proprietary tool is available. Proprietary tools may be available for download from other sources such as modding forums, however they should not be given the same trust as they may have been modified before posting.

Locked Bootloader. After we have the flashing tool it is important to understand what level of privilege we have to write to the device. In some cases our access may be completely unrestricted. However, this is not always the case. Some phones may have an unlock-able or signed bootloader which restricts write access to the device. In the case of an unlock-able bootloader, the manufacturer allows the user to overwrite the device images, but only after they receive a key from the manufacturer. With a signed bootloader, only images which are signed by the manufacturer's cryptographic secret key can be written to the device.

For our analysis we bypass bootloader restrictions for the sake of building recovery images for a greater range of devices. We assume that in a real forensic investigation, the authority prosecuting the case could obtain the manufacturer key through the legal process.

V. Evaluation

Acquiring all Android devices is impossible due to the sheer number of distinct models and carrier-hardware combination. Instead, we have to build a subset which provides sufficient coverage of the smart device market. To produce a such a subset, we focused our resources to acquire devices that varied across the following characteristics:

Popularity: By acquiring the most popular devices currently available, we can show that our method works for the majority of devices a practitioner may encounter.

Manufacturers: From our initial research we discovered that the majority of differences between devices come from the different manufacturers. To cover the largest set of devices with our test bed, we obtained at least one device from each of the major manufacturers (Motorola, HTC, Samsung, LG, Huawei, and Acer).

Software Version: According to Google, approximately 30% of devices that are currently being used run an older version of the OS than that currently available. Thus, even though the software on the device is old, the practitioner is still likely to come in contact with older devices. Also, by including the OS version as a component of our procurement strategy, we are also able to test the effectiveness of our method to changes over time.

Price: Because our methodology may be used in criminal cases, we believe it is essential to ensure its effectiveness with possible "burn" phones, which may be purchased without need for a contract only for limited use then discarded to hide any connections to evidence found on the device.

Using these characteristics we selected and collected the set of fifteen devices shown in Table II. These devices present a combination of software diversity (Android versions ranging from 2.1 to 4.0), price heterogeneity, manufacturer diversity and cover a reasonably wide range of different hardware.

A. Results

After porting the recovery image method to the fifteen Android devices of Table II, we identified the similarities and difference between each attempt. We compare the devices across the four components of the recovery method identified in Section IV. To compare the recovery method across our device set we compared each component of the recovery image method individually and ensured each component was reduced to a minimal set of configuration options to guarantee no extraneous information was included in our analysis. The components analyzed include the image header, kernel, and ramdisk, and the flashing tools.

Header: From our discussion in Section IV we know that our values of interest in the header are the partition location base address and the kernel page size, and the command line options.

In our reduction process we found that the command line options could be left empty without any effect. However, the base address and kernel page size were found to be essential, as any changes to the base address or page size caused the recovery mode to fail. On further analysis of the kernel, we discovered that these values are defined in the kernel, making changes necessary for the header a subset of the kernel.

Kernel: When analyzing the kernel differences, we first identified for which devices kernel source code was available. In our test set, source was available for thirteen of the fifteen devices as shown in Table III.

To compare the kernel source, we reduced the build for each device to a minimal set and removed any code unnecessary to the recovery mode, such as audio, video, and networking features. After reducing the kernel source, we performed manual comparison of the code. We identified configuration differences between USB drivers, chip sets, etc. though many of the differences seemed semantically equivalent. However,

Device	Manufacturer	Carrier	Release Date	OS Version	Cost
Liquid E	Acer	Rogers	Feb 2010	2.2	\$200
Droid	Motorola	Verizon	Oct 2009	2.2	\$100
Droid 2	Motorola	Verizon	Aug 2010	2.2	\$170
XOOM	Motorola	Verizon	Jan 2011	3.2, 4.0	\$400
Nexus S	Google	T-Mobile	Dec 2010	2.3	\$350
Xperia Mini Pro	Sony Ericsson	AT&T	Aug 2011	2.1	\$150
Optimus S	LG	Sprint	Dec 2010	2.2	\$140
MyTouch 4G	LG	T-Mobile	Nov 2010	2.3	\$290
Desire	HTC	U.S. Cellular	Feb 2010	2.2	\$280
Evo 4G	HTC	Sprint	Jun 2010	2.3	\$250
Thunderbolt	HTC	Verizon	Mar 2011	2.2	\$200
Sensation	HTC	T-Mobile	Jul 2011	2.3	\$300
Galaxy SII	Samsung	AT&T	May 2011	2.3	\$550
Pulse	Huawei	T-Mobile	Aug 2009	2.1	\$170
Streak	Dell	AT&T	Dec 2010	2.2	\$218

TABLE II
DEVICE TEST SET

our analysis was very basic and more time, resources, and knowledge of the Linux kernel is required to better understand kernel build differences. When source was not available and we were required to use an extracted kernel binary, we did not perform any analysis on the compiled binaries due to the difficulty of extracting any useful information.

Ramdisk: Across our implementations we discovered that all the files found in the ramdisk directory, excluding the *init* binary and *init.rc* file, were identical across all devices except the HTC Sensation.

Because we used the Android source to build our ramdisks, we were able to trace differences in the *init* binary to different base address and page size values given in the device configuration.

The difference between *init.rc* files in our test set comes from the different memory layouts used. Because the partition map for each device is different, the partitions listed in *init.rc* must also reflect this difference.

The only exception to the ramdisk generality was the HTC Sensation. We were unable to create a custom ramdisk for the Sensation using the Android source because it requires an extra set of executables to be built into the recovery binary. Because the Sensation was the only device we were unable to generalize to and we tested several other similar devices by the same manufacturer, both older and newer, without the same result, we believe that it is an anomaly and does not detract from the generality of the result.

Flashing: In our analysis we made three observations: there are three major flashing tools, the flashing tool is manufacturer dependent, and there are four levels of bootloader availability.

The first, and most commonly used flashing tool was *fastboot* [7]. *Fastboot* is provided by Google and is included in the Android source code[7]

The other two flashing tools observed are proprietary tools for Motorola and Samsung devices. Both Motorola and Samsung use a custom packaging of the recovery image components which require a different flashing mechanism.

Motorola packages their images into a proprietary format called SBF[8] The SBF file provides a structure for organizing multiple firmware components and images into one file [32]. To flash an SBF file to a device RSD Lite (Windows Only) is used by Motorola. However, RSD Lite is not distributed publicly. As an alternate to RSD Lite, *sbf_flash* [16] developed by Optical Delusion can be used.

Similarly, Samsung replaces the standard recovery image structure with the initramfs structure [15], which embeds the ramdisk in the kernel. The process for creating an initramfs is slightly more complicated than the standard bootimg and requires the ramdisk to be inserted into the kernel source[9] Once the initramfs is created, it can be flashed to the device using Samsung's proprietary tool, ODIN. ODIN is also only available for Windows, closed source, and not publicly available. Alternatively, Heimdall [12], an open source implementation, can be used to flash a recovery initramfs.

The final case observed across our device set was the lack of a bootloader mode, which was seen in our LG devices. In cases where no bootloader mode was available, we were forced to root the standard boot mode, mount the recovery partition as writable, and overwrite the recovery image through the standard boot mode. Using a rooting method to overwrite the recovery image is not recommended for an investigation, however, it was used in our analysis to better understand recovery image differences over more devices.

Along with the flashing tools we identified, we also discovered that the flashing tool used is manufacturer dependent. Through our analysis we observed that all devices made by the same manufacturer use the same flashing tool. The discovery that the flashing tool is manufacturer dependent is important as it means there is a limited number of possible tools and the set of flashing tools should be relatively static.

Our final observation with regard to device flashing is that there are four levels of bootloader availability: unlocked,

[7]*Fastboot* source can be found at <location of Android source>directory/system/core/fastboot.

[8]The exact structure of an SBF is not extremely important here, software that can create a well formed .sbf file when provided a bootimg can be found at http://www.ece.cmu.edu/~tvidas/.

[9]Instructions for building the initramfs can be found at http://forum.xda-developers.com/showthread.php?t=1294436.

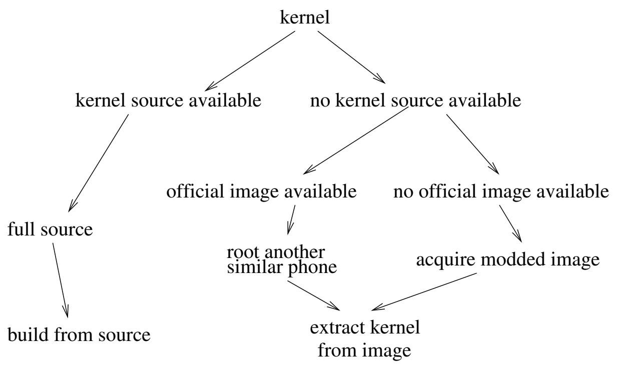
kernel
kernel source available
no kernel source available
full source
build from source
official image available
no official image available
root another similar phone
acquire modded image
extract kernel from image

Fig. 3. **Kernel Decision Tree**: approach used for obtaining a kernel for a new device

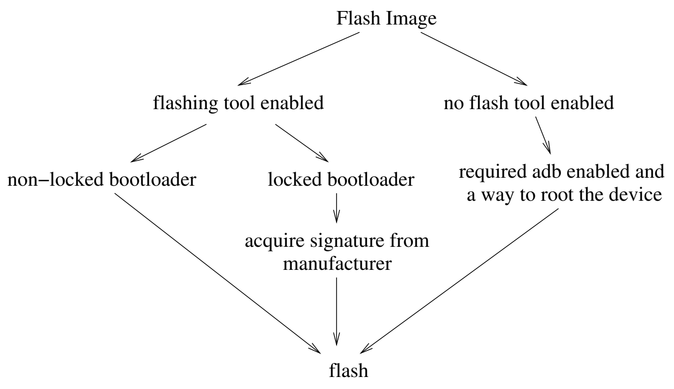

Fig. 4. **Flasher Decision Tree**: approach used for obtaining the flashing tool for a new device

locked, signed, or non-existent. Of the four bootloader availability levels, we discovered that locked bootloaders are the most common. The distribution of bootloader access levels across our device test set can be seen in Table III.

VI. Generalizing our Collection Methodology

From our analysis we identified four key characteristics which are crucial to building a forensic recovery image for an Android device: image header, kernel, ramdisk, and flashing tool. On further analysis we concluded that the information necessary to build the ramdisk and the header is a subset of the kernel information. Therefore, only a new kernel and flashing method are required to collect forensic data from a new device.

We also discovered that the kernel can be obtained by following the decision tree given by Figure 3. Building a kernel from source is the optimal solution for obtaining the kernel because the analyst can read the code and verify it has no unwanted properties. However, source may not be available, in these cases the kernel must be extracted from another working image for the device in question. When choosing which image to extract the kernel from, it is recommended that an official image is acquired from another device of the same model and using one of the rooting methods to extract and image. If an official image is impossible to acquire, then a modified image created by the modding community can be used.

Similarly, the level of flashing privilege can be determined using the decision tree given in Figure 4. When determining the level of write privilege on a new device, it is important to first identify whether a flashing mode is enabled. As shown in Section V we found that the flashing mode was only disabled on two phones. If a flashing mode is enabled, then the next question that should be answered is whether the bootloader is locked. If locked, the investigator must acquire a signature from the manufacturer to cryptographically sign the image before the device will execute the recovery image. If the device is not locked or once the signature is obtained, then the practitioner simply needs to acquire the flashing tool for the device and use this to overwrite the recovery image with the custom forensics recovery image. We also discovered that the flashing tool is manufacturer dependent, and in our experience only three different tools were necessary. Therefore, a set of flashing tools should be maintained by the forensic practitioner. Finally, if the device does not have a flashing mode enabled, then the device needs to be “rooted" and the practitioner can then use the *adb shell* to install the recovery image.

To conclude we provide a table of important information for each of the devices in our test set. Table III should be used as a reference for practitioners when building a forensic recovery image.

VII. Discussion and Conclusion

We have demonstrated a general methodology for collecting a physical memory dump from smart devices running the Android operating system. Evaluating our methodology against the criteria given in Section III we see that by utilizing the method given by Vidas et al. in [32] we achieve user data preservation, atomic collection, correctness, determinism, evidence preservation, and usability and reproducibility and vetting ability are provided through the techniques of image creation given by our methodology.

Data preservation: The Android OS stores each partition in its own, distinct memory address range, ensuring that any writes to the recovery partition will not overwrite blocks on other partitions. The base address of the partition is given in the imageÕs header, as discussed in Section III, along with the partition size. If the base address/size pair given to the flashing device overlaps another partition it is possible that blocks from another partition could be overwritten. On some devices memory maps are stored in known locations and can be read from a running device before an image is written. However, memory maps are not always available, which is why we pulled base address and size values from images distributed by the manufacturer or built from source. In practice, if an invalid address/size pair is given to the flashing tool it will be rejected and not written to the device. In our test cases no, allocated or unallocated blocks of user data were overwritten. Due to the use of a distinct memory range, In all our test cases and over 15 devices we observed that the integrity of user data, in both allocated and unallocated blocks, was never violated. The data preservation of the recovery method is also confirmed through the analysis of James et al. in [28] where the authors compare data collected from the recovery method

Device	Bootloader	Base Addr	Page Size	Open Source Kernel	Flasher	Ramdisk From Source
Liquid E	unlocked	0x20000000	4096	X	Fastboot	X
Droid	unlocked	0x10000000	2048	-	RSD Lite	X
Droid 2	signed	0x10000000	2048	X	BSD Lite	X
XOOM	locked	0x10000000	2048	X	Fastboot	X
Nexus S	locked	0x30000000	4096	X	Fastboot	X
Xperia Mini Pro	signed	0x00200000	2048	X	Fastboot	X
Optimus S	non-existent	0x12200000	2048	X	N/A	X
MyTouch 4G	non-existent	0x00200000	2048	X	N/A	X
Desire	locked	0x20000000	2048	X	Fastboot	X
Evo 4G	locked	0x20000000	2048	X	Fastboot	X
Thunderbolt	locked	0x05200000	2048	X	Fastboot	X
Sensation	locked	0x40400000	2048	X	Fastboot	-
Galaxy SII	unlocked	0x40000000	4096	X	ODIN	X
Pulse	unlocked	0x00200000	4096	-	Fastboot	X
Streak	unlocked	0x20000000	2048	X	Fastboot	X

TABLE III
DEVICE INFORMATION

to the results acquired through the use of the JTAG hardware debugging port over multiple attempts.

Atomic Collection: Atomic collection is obtained by our method through the reduction of services used by the recovery image. In building our image from source, we removed any code unnecessary to collection, restricting execution to only the collection process. In the cases when source was not available for a device and kernel was pulled from another image, atomic collection is not guaranteed, but is likely due to the reduced nature of operations in recovery mode.

Correctness: To ensure the data is correctly collected, we performed a hash of the phone image before and after transmission from the device and used well-trusted bit-by-bit copying tools for flash memory. In all our test cases we observed correct collection. Similar to the case of data preservation, our results were confirmed by James et al. [28] who reports correct collection in all tests of the recovery method through their comparison to the results produced by acquisition via JTAG debugging port.

Determinism: For each device in our test set we performed three collections. Across all the devices we observed the same result for each collection. Also, determinism is confirmed from the results of James et al. [28] who produced consistent results across five collections per device on nine different devices.

Evidence preservation: Our technique makes no changes to the physical device, preventing any physical damage to the evidence. At the software level, because the recovery image is the only partition altered, there were no cases of corrupting or "bricking" the device even when a bad recovery image is used. Because no physical damage is done to the device and the device can continue to function, the collection process can be repeated and the evidence is preserved.

Usability: Once a recovery image is crafted for a specific device, collection only requires the practitioner to follow a simple set of steps to flash the image, connect and collect data for each partition. In practice the entire flashing and connection process takes at most two minutes and the collection process varies on the amount of user data the investigator wishes to collect, but we observed average transfer speeds of 4.3 MB/s allowing collection to complete in a reasonable amount of time.

Vetting ability: For each component of the collection image we either build from source or acquire pre-built components from trusted sources whenever possible. The only case that non-vetted components are necessary comes when source for a device kernel is unavailable from the manufacturer and second device stock kernel cannot be obtained. However, source was available for all but two devices in our test set, therefore creating a fully vetted image is possible for almost all devices. Also, each non-vetted case was an older device and the manufacturer for both has since begun to release source for newer devices.

Reproducibility: Given a new Android device, the forensic examiner simply needs to collect the kernel and memory layout to generate a recovery image. Our method for forensic recovery image creation was shown to be successful for all but one of the fifteen devices tested, therefore providing sufficient reproducibility.

Finally, looking at our research from a more general perspective we can see further applications beyond forensics. We can modify the set of tools provided in the image ramdisk to provide many different features. Some possible applications include Antivirus scanning, which currently can not be properly accomplished due to the privilege restriction of application level antivirus [31], or for malicious intents such as an Evil Maid attack [29]. Due to the possible malicious use of the recovery image technique, manufacturers must take into account the ability to flash the recovery image as a possible security threat.

Future work: Although our method is quite general over a wide range of devices, we determined that we still need to obtain a kernel specific to the device examined. Because only a minimal set of features were shown to be needed by the kernel, we believe it would be possible to create a kernel specialized for recovery.

To provide a more general result for forensic image collection, further research must be conducted on other mobile operating systems. The exact details of our implementation are specific to Android, however, the concept of custom recovery mode creation for forensic collection can be exploited on other platforms due to the general need for the recovery functionality. Different operating systems have different names and data structures for the recovery image and different techniques for image flashing, but the general methodology of using the recovery mode for collection will work for all modern mobile devices. As an example, iPhones utilize a similar recovery mode which can be overwritten using the same technique we propose for data acquisition [35].

We believe that our contributions can provide a useful solution for forensically sound image collection on a large set of devices and a step toward general collection for mobile devices. To this end, we have provided working recovery images and data collected from all devices tested at http://dogo.ece.cmu.edu/~tvidas/pp/.

Acknowledgments

This work was supported in part by CyLab at Carnegie Mellon under Army Research Office grant DAAD19-02-1-0389, National Science Foundation ITR award CCF-0424422 and IGERT award DGE-0903659, as well as a hardware donation by Google Inc.

References

[1] Five iphone 5 features tim cook will not announce today. http://www.forbes.com/sites/adriankingsleyhughes/2012/09/12/five-iphone-5-features-tim-cook-will-not-announce-today/.

[2] Here is another reason google continues to shun sd cards - multiuser support. http://www.androidpolice.com/2012/10/30/here-is-another-reason-google-continues-to-shun-sd-cards-multiuser-support/.

[3] Amon ra recovery tool - android wiki. http://android-dls.com/wiki/index.php?title=Amon_Ra_recovery_tool, Apr 2012.

[4] Android & windows phone: Tablets, apps, & roms @ xda-developers. http://www.xda-developers.com/, Mar 2012.

[5] Android developers. http://developer.android.com/, Apr. 2012.

[6] Android mods | android forum, android hacks, android tips, android applications, android downloads. http://androidmodz.com/, Mar 2012.

[7] Building for devices | android open source. http://source.android.com/source/building-devices.html, Mar 2012.

[8] Busybox. http://busybox.net/about.html, Mar 2012.

[9] busybox - busybox: The swiss army knife of embedded linux. http://git.busybox.net/busybox/tree/docs/new-applet-HOWTO.txt, Mar 2012.

[10] Cyanogenmod wiki. http://wiki.cyanogenmod.com/index.php?title=What_is_CyanogenMod, Mar 2012.

[11] Droid mod. http://www.droidforums.net/forum/droid-mod/, Mar 2012.

[12] Heimdall - glass echidna. http://www.glassechidna.com.au/products/heimdall/, Feb 2012.

[13] Howto: Unpack, edit, and re-pack boot images. http://android-dls.com/wiki/index.php?title=HOWTO:_Unpack\%2C_Edit\%2C_and_Re-Pack_Boot_Images, Mar 2012.

[14] Htcdev - htc kernel source code and binaries. http://htcdev.com/devcenter/downloads, Mar 2012.

[15] Initramfs - gentoo linux wiki. http://en.gentoo-wiki.com/wiki/Initramfs, Jan 2012.

[16] Opticaldelusion: sbf_flash. http://blog.opticaldelusion.org/2010/05/sbfflash.html, Mar 2012.

[17] Rom manager. https://market.android.com/details?id=com.koushikdutta.rommanager&hl=en, Mar 2012.

[18] Samsungopen source release center. https://opensource.samsung.com/index.jsp, Mar 2012.

[19] Sourcery g++ lite 2008q1-126 for arm gnu/linux. https://sourcery.mentor.com/sgpp/lite/arm/portal/release324, Mar 2012.

[20] M. Al-Zarouni. Introduction to mobile phone flasher devices and considerations for their use in mobile phone forensics. *Proc. 5th Australian digital forensics conference*, Dec. 2007.

[21] M. Breeuwsma, M. D. Jongh, C. Klaver, R. V. D. Knijff, and M. Roeloffs. Forensic data recovery from flash memory. *Small*, 1(1):1-17, 2007.

[22] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of android application security. *Proc. USENIX Security Symposium*, August 2011.

[23] S. Garfinkel. Digital forensics research: The next 10 years. *Digital Investigation*, 7:S64–S73, 2010.

[24] A. Hoog. Android forensics. *Mobile Forensics World*, 29, 2009.

[25] A. Hoog. Open source android digital forensics application. http://computer-forensics.sans.org/blog/2010/03/01/open-source-android-digital-forensics-application/, Mar 2010.

[26] J. Lessard and G. Kessler. Android forensics: Simplifying cell phone examinations. In *Small Scale Dig. Dev. Forensics J.*, vol. 4, Sept 2010.

[27] G. Me and M. Rossi. Internal forensic acquisition for mobile equipments. *Proc. IEEE IPDPS 2008*, April 2008.

[28] S. Namheun, Y. Lee, D. Kim, J. James, S. Lee, and K. Lee. A study of user data integrity during acquisition of android devices. *Proc. DFRWS 2013*, Aug. 2013.

[29] J. Rutkowska. The invisible things lab's blog: Why do i miss microsoft bitlocker? http://theinvisiblethings.blogspot.com/2009/01/why-do-i-miss-microsoft-bitlocker.html, Jan 2009.

[30] J. Sylve, A. Case, L. Marziale, and G. G. Richard. Acquisition and analysis of volatile memory from android devices. *Digital Investigation*, 8(3-4):175 – 184, 2012.

[31] T. Vidas, D. Votipka, and N. Christin. All your droid are belong to us: A survey of current android attacks. *Proc. WOOT 2011*, Aug. 2011.

[32] T. Vidas, C. Zhang, and N. Christin. Toward a general collection methodology for android devices. *Proc. DFWRS 2011*.

[33] R. Walls, E. Learned-Miller, and B. Levine. Forensic triage for mobile phones with dec0de. In *Proc. USENIX Security Symposium*, Aug. 2011.

[34] S. Y. Willassen. Forensic analysis of mobile phone internal memory. *Memory*, page 191-204, 2005.

[35] J. Zdziarski. *iPhone forensics - recovering evidence, personal data and corporate assets*. O'Reilly, 2008.

Daniel Votipka received the B.S. degree in Computer Science from the Illinois Institute of Technology, Chicago, IL in 2010 and the M.S. degree in Information Security, Technology, and Management from Carnegie Mellon University, Pittsburgh, PA in 2012. He is currently serving in the United States Air Force as a Cyberwarfare Officer. His current research interests include forensics, code analysis, and mobile security.

Timothy Vidas is an Electrical and Computer Engineering Ph.D. candidate at Carnegie Mellon University. Tim's recent research interests revolve around mobile platform security and privacy, volatile data collection and analysis, reverse engineering, digital forensics, and malware analysis. Tim previously held research positions at CERT, the Naval Postgraduate School and the University of Nebraska. He holds a B.S. and M.S in computer science and two DEFCON CTF black badges. Tim is a member of the Shmoo group and is a DC3 Forensics Challenge grand champion.

Nicolas Christin is an Assistant Research Professor in Electrical and Computer Engineering and CyLab at Carnegie Mellon University. He holds a Diplôme d'Ingénieur from École Centrale Lille, and M.S. and Ph.D. degrees in Computer Science from the University of Virginia. His research interests are in computer and information system security; most of his work is at the boundary of systems and policy research. He has most recently focused on online crime, security economics, mobile security, and psychological aspects of computer security.