Assessing data remnants in modern smartphones after factory reset

Blankesteijn, M.B.; Fukami, A.; Geradts, Z.J.M.H.

DOI
10.1016/j.fsidi.2023.301587

Publication date
2023

Document Version
Final published version

Published in
Forensic Science International: Digital Investigation

License
CC BY

Citation for published version (APA):
Assessing data remnants in modern smartphones after factory reset

Mattheüs B. Blankesteijn a, *, Aya Fukami a, b, Zeno.J.M.H. Geradts a, b

a University of Amsterdam, Science Park 904, Amsterdam, 1098, XH, the Netherlands
b Netherlands Forensic Institute, Laan van Ypenburg 6, The Hague, 2497, GB, the Netherlands

A R T I C L E   I N F O

Article info
Article history:
Received 17 August 2022
Received in revised form 2 June 2023
Accepted 4 June 2023
Available online 10 July 2023

Keywords:
Android
Mobile forensics
Factory reset
Google Pixel
Xiaomi Redmi
File based encryption
Metadata encryption
Flash drive image

A B S T R A C T

It is commonly believed by end-users that factory reset on an electronic device restores the state of the device back to when it was shipped from the factory. Nevertheless, user data has reportedly been recovered after a factory reset by applying forensic data recovery techniques. In order to protect end-users’ privacy, smartphone manufacturers started implementing security countermeasures such as encryption. Specifically for Android smartphones, the encryption scheme advanced as the Android version became higher. Meanwhile, the effectiveness of a factory reset on a modern Android device has not been publicly explored. In this paper, the effectiveness of factory reset on modern devices running Android 11 or 12 is investigated. This is done by looking at low level differences on data extractions before and after creating data and after resetting a device. Results show that some parts of the encrypted data are still accessible in their binary form as not all data is reset to factory state. Furthermore, different partitions do not wipe data that was created during device usage, from which information about the use of a device may be deduced.

© 2023 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Factory reset is a function in electronic devices that should restore a device to its initial factory state. However, previous research on factory reset in smartphones has shown that using forensic acquisition tools data is still accessible (Khramova and Martinez, 2018). Nowadays, modern smartphones encrypt user data to provide privacy and security. The impact of this on digital forensics has not yet been publicly explored. This research investigates the effectiveness of a factory reset in modern Android smartphones with respect to digital forensics.

As smartphones become more relevant in people’s lives, the extend to which secure erasure of data on these devices is possible also becomes more important. Nowadays, a large portion of the world population is accompanied by a smartphone (Zinkus et al., 2021) and although the motives for executing a factory reset can vary from person to person, the desired outcome of what is expected from a factory reset device is similar.

As modern Android smartphones encrypt their user data, from a forensic perspective, that kind of user data is not useable without the decryption keys. However, according to Farmer and Venema (2004), forensic relevant data can be found in all kinds of (un)usual places in computers. Unencrypted system data could still be readable and provide valuable information about what actions happened, or did not happen, on a phone at a certain point in time. Therefore, this research aims to investigate to what extent forensically relevant data on a modern Android device is still accessible after a factory reset.

To answer this question, data is generated on different smartphones running recent Android versions. Drive images are captured from these devices before and after creating data, and after factory resetting the devices. Comparison of these images then shows that data on an Android device is not always securely erased. Instead, it is left in encrypted or sometimes even unencrypted form.

In short, the contributions made by this research are as follows:

- Provide an overview of Android encryption, imaging and resetting, and its implications for forensics.
- Provide an analysis on what data remains after resetting a modern Android device.

The remainder of this paper is organised as follows. First of all, an overview of encryption, imaging and resetting in modern Android phones is given in Section 2. Secondly, other research...
closely related to this work is discussed in Section 3. After that, Section 4 details the approach followed to examine smartphone data remnants. In Section 5 the outcomes of this approach are listed. Section 6 then continues by interpreting and discussing these results, which is followed by a conclusion in Section 7.

2. Background

In this section, an overview is provided on the state-of-the-art in Android encryption, in extracting data from an Android device, and in resetting an Android device. The focus will be on consumer Android devices, leaving out details specific for company managed devices and working profiles.

Regarding the target audience for the coming (sub)sections; throughout this paper, we provide practical information on the Android data structure and the factory reset, and how the actual data change can be monitored before and after the factory reset. The authors believe the presented information can help both field officers, forensic practitioners, and researchers understand the actual low-level data change on factory-reset Android devices.

2.1. Android encryption

This section describes the general process and setup of encryption using Android. Nevertheless, individual device encryption differences may exist, as vendors can make their own design choices.

User data on Android devices can be encrypted using symmetric key encryption (Android Developers, 2022b). From Android 4.4 until Android 7.0, the only built-in encryption option was full disk encryption (FDE). Using dm-crypt (The kernel development community, 2022a), user data on disk is encrypted and decrypted by writing to a device mapper (dm) partition. The blocks on the physical disk to which this partition maps are encrypted with the same disk encryption key but with a different initialisation vector for each block (Loftus and Baumann, 2017). This key resets when doing a factory reset and is wrapped using a key encryption key based on salted user authentication (Loftus and Baumann, 2017). In that way, the end-user can change her authentication, such as a pin code, without the device needing to be encrypted with a new key (Android Developers, 2022d; Loftus and Baumann, 2017). Since Android 7.0 the user data partition can be encrypted using file-based encryption (FBE). Devices that launch with an Android version higher than 10 are required to use this form of encryption (Android Developers, 2022b). With FBE, each file can use a separate key, and thus a separation can be made between files that can be unlocked by the device, and files that can only be unlocked after processing user credentials. The usage of only device encrypted files is called direct boot mode (Android Developers, 2022b). The device key used for this is protected by verification of the phone boot (Android Developers, 2022g). The area of storage that is protected by user credentials is unlocked on the first user unlock of a phone and only locked again when the phone is shut down (Android Developers, 2022g; Galindo et al., 2021).

FBE is split in encrypting the content of files and the names of files (Android Developers, 2022c). For both types a 256-bit variant of the advanced encryption standard (AES) encryption is used, optionally in combination with XChaCha12 (Android Developers, 2022c; The kernel development community, 2022b) (adiantum). For file contents, the XEX-based tweaked block cipher mode with ciphertext stealing (XTS) is used, while for file names, ciphertext stealing (CTS) or hash encrypt hash (HEH) mode can be used. The content of a file or folder is encrypted with an unique key that is derived by a key derivation function (KDF) from a master key and number-only-used-once (nonce) (Groß et al., 2019; The kernel development community, 2022b). The first version of this KDF, as implemented in the Linux kernel, generated a key by encrypting the master key with the nonce and truncating the result to the required length (The kernel development community, 2022b). From Android 11.0 on, a more flexible and secure variant of this KDF is required (The kernel development community, 2022b; Android Developers, 2022c).

As with FDE, the 512-bit master keys used for FBE are encrypted by a 256-bit key encryption key, which is stored in a trusted execution environment (TEE) (Android Developers, 2022c). Such a key in the TEE is protected by an authentication token, a stretched credential, and a secdiscardable hash (Android Developers, 2022c). The authentication token is generated on user login, if the user has set any credentials. The stretched credential is the salted and secdiscrypted variant of the user credential, if it exists. The secdiscardable hash is a 512-bit hash of random data discarded when the key is re-encrypted or deleted (Android Developers, 2022c).

When using FBE, file metadata such as permissions and modification timestamps are still readable. Therefore, from Android 11.0 metadata-encryption has to be used on top of FBE. This form of encryption is analogous to FDE and is protected by a verified boot (Galindo et al., 2021; Android Developers, 2022c).

2.2. Android imaging

There are different ways to extract data from an Android device. These ways can be ranked from almost non-invasive to very invasive. Among the least invasive is the built-in backup functionality to back up data from the device to a file. However, each application can decide individually whether or not it wants to be backed up by this functionality. Therefore, from a forensic perspective this functionality is less useful.

To get information from applications without their consent, one needs to increase the number of rights on a phone. In other words, root access to a device is needed. The way to get this privilege can differ from device to device (Binary Hick, 2021). For some devices, manufacturers do not stand in the way if the owner wants to root a phone, while for other devices, specific weaknesses need to be exploited to gain root privileges.

Among the most invasive ways to read data from an Android device is to read the raw bits directly from the memory chip. This can be done through the Android interface with root rights or by physically connecting to the chip through e.g. a joint test action group (JTAG) or in-system-programming (ISP) interface (Fukami et al., 2021). Alternatively, the chip can be physically removed from the device and read out separately. This process is also known as performing a chip-off.

When taking the device apart physically is not an option, one needs to use the tools available on the phone itself, or, in case of an unlocked bootloader, boot a program that is able to read the data and communicate with the examiner’s device. For this last option, recovery programs such as TWRP (Team Win L.L.C., 2022) can be used. A built-in way to communicate with an Android device is through the Android device bridge (adb) (Android Developers, 2022a). With this program, commands can be run on a device. On most Android phones, basic Linux tools such as dd (Rubin et al., 2022) and tar (Free Software Foundation, Inc., 2022) are already available. To copy the content of block devices, dd can be run as an adb shell command, and its output can be written to (external) storage on the phone, to a networking program such as netcat (Hobbit, 2022) or simply written to the standard output of the examiner’s device, from where it can be sent to a file. A logical extraction can be acquired by running the tar command on the directories of interest. From a forensic perspective, however, using tools available on the devices means one has to trust what these
tools do. Therefore, it is not uncommon for forensic research to use commercial off-the-shelf (COTS) tools for investigation. One example of this is the universal forensic extraction device (UFED) (Cellebrite, 2022). This tool is designed to make both physical and logical extractions from all kinds of devices while functioning mostly as a black box to researchers. For modern Android devices, this tool also requires to modify the subject of extraction by turning on developer mode with USB-debugging. Another commercial tool designed for devices using a MediaTek (MTK) system on a chip (SoC) is called Chinese Miracle-2 (CM2) (Infinity-Box, 2022). This tool works before booting the Android operating system (OS) and can dump all partitions on the device without the device being rooted or its bootloader unlocked. Nevertheless, how the tool achieves this is not publicly known, although it likely exploits some weakness in MTK chips. Furthermore, Mobile Relevator (Kerier, 2022) is a partly open source and non-commercial tool that can be used to extract data from Android devices. For example, when it comes to partition extraction, it just executes adb shell commands and pipes the output to netcat, as discussed before. Next to these tools, there are other commercial tools available, including, but not limited to, XRY (MSAB, 2021) Oxygen Forensic Detective (Oxygen Forensics, Inc, 2021) and GrayKey (Grayshift, LLC., 2022).

Until recently, data could be extracted from a device in a forensically sound way by doing a chip-off and reading the bits on the lowest possible level. However, nowadays Android phones use encryption, which is why it is now more common to extract data from a running OS. For devices using FDE, one can copy the bits of the dm device corresponding to the data that one is interested in, usually the userdata partition. For FBE, files are decrypted on-the-fly which means all files need to be accessed individually or the master keys need to be extracted from the device, after which it should be possible to mount the device offline. The former can be achieved through the user system, for example by running the tar command, while the latter can be achieved by examining the random access memory (RAM) of the device (Groß et al., 2021).

2.3. Android reset

In this section, the process of an Android reset is described. When references are made to the Android source, this entails the code released for the latest Android 12.1 version. Nevertheless, most statements also apply to somewhat older versions.

Although there are multiple ways to reset an Android device, usually one uses the reset options provided in the settings menu. Among these options are usually functions to reset “Wi-Fi, mobile & bluetooth”, “app preferences”, “downloaded SIMs”, “external storage” and “all data (factory reset)”. This last option claims to erase “all data from your phone’s internal storage”. This is the functionality explored in this research.

When resetting an Android phone, there are first some safeguards to make sure it is the user that requests the reset and that it is done intentionally. This is implemented by requesting user credentials and having multiple views where the user needs to click a button to confirm the wipe (Android Developers, 2021a). The Android intent that is launched on confirming the wipe then sends a request to the RecoverySystem to wipe the device through the rebootWipeUserData function (Android Developers, 2021b). The comment explaining the function however already notes that factory reset is “something of a misnomer because the system partition is not restored to its factory state”. Nevertheless, the function calls a boot command with as arguments to wipe data, the default locale, a reason with an optional timestamp, and whether the system should shutdown afterwards (Android Developers, 2022a). When booting into recovery the arguments for this command are written to the bootloader control block (BCB), to restart the process in case of failure (Android Developers, 2020b). Then it is checked whether there is a reason for the wiping of data. If the reason is valid, WipeData is called (Android Developers, 2020a). Looking at the recovery function to wipe data, there are three volumes that get erased:/data, /cache and /metadata (Android Developers, 2021c). When the cache is wiped, the logs are first written to memory so they can be restored (Android Developers, 2021c). To wipe data, the function format_volume is used. In this function, any key location corresponding to the volume gets its first 4096 bytes overwritten with zeroes (Android Developers, 2020c). Furthermore, the volume with which the function is called is formatted by mke2fs or make_f2fs, depending on whether it is ext4 or F2FS formatted respectively. After formatting, the BCB is cleared again, so normal boot can occur (Android Developers, 2020a). In short, this means encryption keys are securely erased, and the/data, /cache and /metadata partitions are reformatted.

3. Related work

Schwamm and Rowe (2014) inspected multiple smartphones using Cellebrite software. On Android phones, quite often images, audio and text files were still available after factory reset. Data about contacts, IP connections and location could not often be recovered, however. Nevertheless, the investigated Android versions are between 1.5 and 4.2, and the effects of encryption are not investigated. Similarly, Simon and Anderson (2015) found user data after factory reset on Android versions between 2.2 and 4.3. They recommend the use of FDE to mitigate risks.

More recently, Khramova and Martinez (2018) investigated phones running Android versions between 2.3 and 7.0. They found it difficult to predict what might be found on a phone beforehand, as even supposedly identical phones may use different variants of memory chips responsible for carrying out erasure commands. Nevertheless, the quality of factory resets tends to improve with newer Android versions; for Android 6 and 7 the factory reset was reported to have “succeeded”. Also closely related is a recent blog post on detecting factory resets in Android (Binary Hick, 2021). Although this post also focuses on more recent Android devices, this research differs in focusing on data remnants instead of detecting resets.

With regard to the phone memory itself, Skorobogatov (2018) found that under some conditions non-volatile flash memory may not be erased properly. Moreover, Schneider et al. (2021) brought to attention that even new flash drives may already contain data, due to not being properly recycled in the factory.

In short, to the best knowledge of the authors, no literature has practically investigated the data after factory reset on modern Android devices. Therefore, the current literature may lead to speculation that data can still be recoverable after Android factory reset. In this research, we aim to reproduce former studies on Android factory reset on modern smartphones.

4. Methodology

This section describes the approach taken to generate the experimental results. The generic method flow is sketched in Fig. 1. In general, data is generated on clean modern Android phones. For this, two Google Pixel 3a phones and one Xiaomi Redmi 9 phone are used. The Redmi phone has a 32 GB embedded multimedia card (eMMC), runs a MTK SoC, and is updated to Android 11. The Pixel phones have an eMMC chip of 64 GB, a Qualcomm SoC and run factory images of Android 11 and 12 respectively. More specifically, the Pixel phones are flashed with sargo-sp2a.220505.006 (Android 12) and sargo-rq3a.211001.001 (Android 11) (Google Developers, 2022a). After flashing the factory images, the boot images for
these phones are rooted with Magisk 25.0 (Wu, 2022), which was installed through adb (Android Developers, 2022a). The rooted boot images were flashed with fastboot (Google Developers, 2022b). As user credentials a four number pin code is used on the Android 12 phone, while no explicit user credentials are set for the other phones to investigate any differences.

4.1. Extraction

In this section, it will be described in more detail how data is acquired from Android devices. Firstly, data acquisition from Google Pixel phones is described, followed by the method used for extraction on a Xiaomi Redmi phone.

4.1.1. Google Pixel

As soon as root access is achieved on the phones a baseline image is extracted from the phone. This is done by executing dd in the Android device shell and sending the output to the examiners device. For dd version 0.8.3 and 0.8.4 from the toybox package is used for Android 11 and 12 respectively, which is installed by default on the devices. As we use the production builds of the Android software, it is impossible to run adb in root mode. However, it is possible to send commands to the shell on the device and acquire root rights there. Output can then be fed to the standard output of the examiner’s device. Fig. 2 shows an example how data can be extracted in this way.

4.1.2. Xiaomi Redmi

For the Xiaomi Redmi phone extraction is performed without rooting or unlocking the bootloader through the low level MTK tool CM2 2.34 (Infinity-Box, 2022). To make this program work on recent Android phones, a specific algorithm needs to be followed when connecting the phone to the examiner’s device. The phone should be powered off when connecting the device through USB. Normally it will start charging as it receives power from the examiner’s workstation. CM2 can then be started and asked to identify the connected phone, without the need to select the correct phone first. When it asks to reconnect the USB, one should disconnect the USB, wait until the phone left its charging mode, and then hold both volume and down buttons when reconnecting the phone. The program was not found to work without holding these buttons for phones running a newer version than Android 10. For the extraction of the partitions, the memory mode of CM2 is used.

4.2. Data generation

On all phones, data is generated in similar ways. We loosely follow the Mobile Device Data Population Setup Guide (National Institute of Standards and Technology (NIST), 2016), but chose to follow more on day-to-day use. Thus, more focus is placed on data generation by different applications than, for example, the sending of multimedia messaging service (MMS) messages. In detail, first of all, for every phone a new Google account is created, to prevent any contamination or inconsistencies from historical data. With this account, the applications WhatsApp, Signal, Telegram, Element, NS (Dutch railway planner), TikTok, Instagram, Mijn Bijbel (Bible reader), Flitsmeister, and Schiphol Amsterdam Airport are installed from the Play Store. All applications present on the phone are updated to their latest versions (June 2022).

For the same reasons, every phone uses a new SIM card. With the corresponding phone number, accounts are created for the applications WhatsApp, Signal and Telegram. The Google account is used to log in to TikTok and Element. For Instagram, a new account has to be created, which follows the Instagram and National Geographic channels. With each of the messaging applications, some messages with Dutch compound words that should be unique and recognizable in plain text data are sent. For TikTok and Instagram, a couple of images and videos are viewed. On the pre-installed YouTube application, the video “The Mandalorian OST - Main Theme” is watched. With NS a travel from Den Haag Ypenburg to Bad Bentheim is planned. Using Mijn Bijbel chapter 7 of Ecclesiastes is viewed. Google Maps and Flitsmeister are used to navigate various places in the Netherlands. Different intents in the Schiphol app are viewed.

In addition, some built-in actions are taken on the phone. Among these are scrolling through menus, news and settings, taking some pictures, use the browser, receive e-mails, reconnecting a charger, modifying brightness, setting a wallpaper, sending and receiving SMS messages, making and receiving a call, connecting to WiFi networks and toggling airplane, flashlight and rotate modes. For the Android 11 phones the NFC Tools application is also used to read a MIFARE DESFire card, Bluetooth is used to connect and play an alarm through a headphone and some screenshots are made.

4.3. Experiments

To examine what data is left on the Android devices after factory reset, the following experiment is set up. After being able to get an

```bash
adb shell "su -c 'dd if=/dev/block/<device> bs=<size>" | tee <filename> > sha256sum > <filename>.sha256"
```

Fig. 2. General extraction command. As devices mmmch8k0 and all dm-N devices are extracted. For `size`, 1024 seems to work well. `filename` can be any valid filename. The file can be piped directly to a hash calculator, such as sha256sum. Optionally, the output of `dd` can be piped to gzip first for compression. For properly encrypted data, this should not decrease the file size much, however. To create an archive of decrypted files, the `tar` command can be used instead of `dd`.

Fig. 1. General flow for the methodology. First an extraction is made from a device, as described in Section 4.1. Thereafter, data is generated on the device as described in Section 4.2. Another extraction is followed by a factory reset as described in Section 4.3. This is followed by a final after-reset extraction. The extractions that are made can than be compared to see which created data was remnant.
5. Results

This section contains the results of the described experiments. In general, the resets take little more than a minute for the phone to execute. For the Pixel phone running Android 11, there were 62 seconds between the start of the erasure process and the booting of the phone into the OS. This makes it implausible that the whole storage is reset to factory state.

5.1. Commercial off-the-shelf (COTS)

Auto-detection with UFED can recognise the Google Pixel phones. However, when manually looking into the list of supported devices the Google Pixel 3a is not there. As a result, on both the Android 11 and Android 12 phones almost all extractions fail. For example, when trying to perform a physical extraction UFED throws an error when trying to reboot. Manual reboot does not help to solve this problem.

However, some extractions could be performed on the Pixel 3a by not selecting it as a Pixel 3a, but as Pixel 3a XL. Physical extraction then still fails on reboot, though an ‘advanced logical’ extraction becomes possible. Nevertheless, UFED 7.54.0.488 can still show nondeterministic behaviour by raising different kinds of warnings/errors when repeating the same process on the same phone. In addition, a temporary application is run on the phone showing a warning that it was developed for an older Android version. Furthermore, through this method the analyser of UFED is not able to read chats, such as those included in the crypt14 database of WhatsApp. This may be because an ‘advanced logical’ extraction does not seem to give access to android application sandboxes, such as the ones containing necessary keys for database decryption.

A separate method provided by UFED that worked through the pixel 3a XL interface was to capture chats. This works by actually opening the application on screen and scrolling through chats. For Signal the software is capable of clicking through the interface to make a manual backup. As any user-installed applications are gone after a factory reset these are also not relevant options for this research.

While UFED requires manual unlock of a phone and enabling of USB debugging, CM2 is able to extract data from the phone without any adjustments to the phone, other than holding the volume up and down buttons when connecting to the forensic workstation. Through this method a sparse image of the userdata partition could be acquired. However, when extracting this image and mounting it there is not much data visible. Binary analysis of the image does show that the file consists of mostly zeroes. Nevertheless, it is also possible to extract raw partitions, as discussed in Section 5.2. As expected the files on this partition are encrypted after extraction.

5.2. Partitions

Table 1 shows an overview of which partitions are modified at which point in the experiment for the Pixel phones. As visible, the major part of partitions does not get modified at any point in the experiment. The partitions that do get modified are discussed below.

Firstly, on modem_a the bytes 0xd554 are changed to 0x8652, which is repeated multiple times throughout the file. As this partition is related to the global system for mobile communication (GSM) (mirfatif, 2017) it might indicate some kind of identification or version number that is rolled back on reset of the phone. On fsc about half of the bytes is changed on every capture. As the entropy of this partition is around five bits per byte the data may be encoded, while encryption is less likely. The data itself is said to contain filesystem cookies for the modem (mirfatif, 2017). Similarly for modemst1 and modemst2 almost all bytes are changed on every capture. For these partitions the entropy is almost eight bits per byte, indicating random or encrypted data. The remnant bytes for this partition consist of almost only individual bytes that can be attributed to chance.
Table 1
Differences between partitions in different states of the experiment on the Google Pixel 3a phones in bytes. The numbering shows the order of partitions on disk as indicated by mmls (Carrier and Contributors, 2019). All sizes are in bytes. Sizes and partitions are the same for Android 11 and 12. The last column shows the bytes that are the same for the after data and after reset captures, while being different for the empty and after data captures. Thus these are bytes that are created on data creation and end up to be the same on the same position on disk after reset.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Size</th>
<th>Empty-Reset</th>
<th>Data-Reset</th>
<th>Empty-Data</th>
<th>Remnant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Android 11</td>
<td>Android 12</td>
<td>Android 11</td>
<td>Android 12</td>
</tr>
</tbody>
</table>
For persist however, there is some data that is more truly remnant after reset. Interestingly this shows to be a partition data generally persisting instead of being modified on use. The modified data that persists after a reset includes Qualcomm data on battery status, such as qcom_charge_full and qcom_cycle_count_bins, gauge capacity and some binary files. The forensic value of this data is limited however, as it is closely tied to the physical device itself rather than the effect of actions on the device. One thing that may be deduced is how intensively the phone has been based on the reported capacity that is left. Table 2 shows some details on remnant data on the persist partition for the Android12 phone.

For frp (factory reset protection) there is a difference between Android 11 and 12. On Android 12 there are 110 bytes that are modified during data creation. From these 110 bytes, the first 32 bytes contain some data that is restored to the initial state. The other bytes are changed back to zeroes on reset. With exception to another 18 bytes, the rest of the partition consists of only zeroes on all measurements. The purpose of this is to store some settings with regard to developer options (mirfatif, 2017). As settings are restored on reset (i.e. developer mode is disabled again) it is logical that this partition gets back to default settings. For Android 11 however, most of the data that is written on reset is different from the initial state. Some bytes even persist from the data that was created. Alternatively, these bytes are created when connecting to the Forensic workstation as the developer option of USB debugging is enabled for that.

For metadata we see that the key files have changed after the reset, specifically the encrypted_key, keymaster_key_blob and secdiscorable files. There is some remnant bytes however, although manual inspection shows that this all consists of bytes that were changed to zeroes when creating data. Thus the only remnant is here is that the phone has been in use, if those bytes always contain data on a freshly installed phone.

For the userdata partition the contents are encrypted and the keys thrown away on factory reset. However, as the whole partition does not get overwritten with zeroes the original data can still remain, which is visible by the remnant bytes for this partition. There are relatively more bytes remnant on the Android 11 phone when compared to the Android 12 phone. Most of the created bytes are not the same on both phones anymore though.

Table 2 contains an overview of which partitions exist and get modified on the Xiaomi Redmi phone. The total amount of partitions is smaller when compared to the Pixel phones. This is mostly due to the A/B system for simple recovery in case of over-the-air update failure that is used on the Pixel devices.

For the frp partition we see the same behaviour as on Android 12. On protect1 and protect2 there is a file NA77_005 that is modified on writing data and on reset, but also contains some remnants from data creation. This is a MTK and MIUI related file containing some binary flags associated with file system operations (Elenedeath, 2020). On expdb data was created on reset but no data remains from earlier modifications. In fact, the partition was not modified by creating data.

Interestingly, metadata is not modified. Manual inspection shows that this partitions is full of zeroes and does not seem to be UEFI update has been performed. It also notes here that a fastboot device has been connected and 67108864 bytes are downloaded. This information is not wiped on factory reset. Table 3 shows some data on from the klog partition where some parts with high and low entropy are shown.

For persist however, there is some data that is more truly remnant after reset. Interestingly this shows to be a partition data generally persisting instead of being modified on use. The modified data that persists after a reset includes Qualcomm data on battery status, such as qcom_charge_full and qcom_cycle_count_bins, gauge capacity and some binary files. The forensic value of this data is limited however, as it is closely tied to the physical device itself rather than the effect of actions on the device. One thing that may be deduced is how intensively the phone has been based on the reported capacity that is left. Table 2 shows some details on remnant data on the persist partition for the Android12 phone.

For frp (factory reset protection) there is a difference between Android 11 and 12. On Android 12 there are 110 bytes that are modified during data creation. From these 110 bytes, the first 32 bytes contain some data that is restored to the initial state. The other bytes are changed back to zeroes on reset. With exception to another 18 bytes, the rest of the partition consists of only zeroes on all measurements. The purpose of this is to store some settings with regard to developer options (mirfatif, 2017). As settings are restored on reset (i.e. developer mode is disabled again) it is logical that this partition gets back to default settings. For Android 11 however, most of the data that is written on reset is different from the initial state. Some bytes even persist from the data that was created. Alternatively, these bytes are created when connecting to the Forensic workstation as the developer option of USB debugging is enabled for that.

For metadata we see that the key files have changed after the reset, specifically the encrypted_key, keymaster_key_blob and secdiscorable files. There is some remnant bytes however, although manual inspection shows that this all consists of bytes that were changed to zeroes when creating data. Thus the only remnant is here is that the phone has been in use, if those bytes always contain data on a freshly installed phone.

For the userdata partition the contents are encrypted and the keys thrown away on factory reset. However, as the whole partition does not get overwritten with zeroes the original data can still remain, which is visible by the remnant bytes for this partition. There are relatively more bytes remnant on the Android 11 phone when compared to the Android 12 phone. Most of the created bytes are not the same on both phones anymore though.

Table 2 contains an overview of which partitions exist and get modified on the Xiaomi Redmi phone. The total amount of partitions is smaller when compared to the Pixel phones. This is mostly due to the A/B system for simple recovery in case of over-the-air update failure that is used on the Pixel devices.

For the frp partition we see the same behaviour as on Android 12. On protect1 and protect2 there is a file NA77_005 that is modified on writing data and on reset, but also contains some remnants from data creation. This is a MTK and MIUI related file containing some binary flags associated with file system operations (Elenedeath, 2020). On expdb data was created on reset but no data remains from earlier modifications. In fact, the partition was not modified by creating data.

Interestingly, metadata is not modified. Manual inspection shows that this partitions is full of zeroes and does not seem to be
used by Xiaomi, while it is the place to store encrypted keys on Google Pixel phones. Instead, there is a `vold` folder available on `md_udc` but this folder is also empty. On factory reset the folder `gsi/dsu/avb` containing some public keys is removed on this partition. On data creation some apex sessions are created but these are removed again on reset. Inspection of the `userdata` partition shows that there is a folder named `unencrypted` that contains the partially encrypted information needed to derive the master key.

The partition `nvcfg` contains some values in `sensor/mag_bias.json` that are created on data creation. These values persist after reset. In addition, a few bytes from the file `fg/old_fg_data` are equal for data and reset images while different on empty images. However, most of the data in this file is changed on every image so this may also be equal by chance. Similarly on `nvdata` some binary files get modified on data creation and again on reset. Manual inspection shows that files that were already there before data creation are mostly refilled with zeroes. Meanwhile, files that were not there before data creation are kept after reset and marginally modified. One of these remnant files contains LDI data which may be related to a liquid damage indicator.

On the persist partition, the modified files are `fdsd/st` and `LOG.ini`. For the former, this includes the incremental of the `extra_reference_suggestion` parameter. The latter has an entropy of more than seven bits per byte which may indicate that this file contains an encrypted log.

For the cache partition, some new data is created on reset, but no real remnants are left. This is expected, as new system logs are written as soon as the phone is in use. For `userdata` about one in nineteen of the created bytes stay on disk after reset. This means that this data can potentially be recovered if one knows the encryption key used.

### 5.3. Android 12 experiments

In this section, the results for Bluetooth, reflashing and pure factory reset on the Android 12 Google Pixel 3a are described.

#### 5.3.1. Bluetooth

There are two Bluetooth partitions on the device, which are 1MiB (1048576 bytes) in size. Initially both these partitions are

| Table 3 | xxh hexadecimal output parts of the klog partition. A selection of output lines with remnant data for both high entropy and low entropy sections is shown. Blue color indicates remnant data. | 8 |

<table>
<thead>
<tr>
<th>Entry</th>
<th>Data</th>
<th>Read</th>
<th>p</th>
</tr>
</thead>
</table>

*On the persist partition, the modified files are fdsd/st and LOG.ini. For the former, this includes the incremental of the extra_reference_suggestion parameter. The latter has an entropy of more than seven bits per byte which may indicate that this file contains an encrypted log.*

*For the cache partition, some new data is created on reset, but no real remnants are left. This is expected, as new system logs are written as soon as the phone is in use. For userdata about one in nineteen of the created bytes stay on disk after reset. This means that this data can potentially be recovered if one knows the encryption key used.*

5.3. Android 12 experiments

In this section, the results for Bluetooth, reflashing and pure factory reset on the Android 12 Google Pixel 3a are described.

5.3.1. Bluetooth

There are two Bluetooth partitions on the device, which are 1MiB (1048576 bytes) in size. Initially both these partitions are
filled with zeros. However, no modifications to these partitions were seen during undertaking actions through Bluetooth, neither after taking these actions, nor after doing a reboot.

5.3.2. Reflashing

Although the reflashing of data seemed to have succeeded, trying to boot the phone again results in the screen as visible in Fig. 4.

Looking through the TWRP recovery at the logs of the phone, it appears that phone cannot recognise a F2FS filesystem. As this is followed by some cryptology warnings, this likely means the metadata encrypted partition could not be decrypted and thus not recognised as containing a F2FS formatted filesystem.

5.3.3. Factory reset files

Looking at factory reset related files some things can be noted. Some specific files with 'factory reset' in the name, such as time_since_factory_reset were empty both before and after the reset.

---

Table 4
Differences in bytes between partitions found by CM2, in different states of the experiment on Xiaomi Redmi 9, sorted by size. Bold partitions are also on the Pixel phones.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Size</th>
<th>Empty-Reset</th>
<th>Data-Reset</th>
<th>Empty-Data</th>
<th>Remnant</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sgpt</td>
<td>16896</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>pgpt</td>
<td>32768</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>efuse</td>
<td>524288</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>misc</td>
<td>524288</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>para</td>
<td>524288</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>frp</td>
<td>1048576</td>
<td>0</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>spmfw</td>
<td>1048576</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>spm_1</td>
<td>1048576</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>spm_2</td>
<td>1048576</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>lk</td>
<td>2097152</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>lk2</td>
<td>2097152</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>sec1</td>
<td>2097152</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>proinfo</td>
<td>3145728</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>preloader</td>
<td>4194304</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>tee1</td>
<td>5242880</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>tee2</td>
<td>5242880</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>scp1</td>
<td>6291456</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>scp2</td>
<td>6291456</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>devinfo</td>
<td>8388608</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>logo</td>
<td>8388608</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>protect1</td>
<td>8388608</td>
<td>25881</td>
<td>5978</td>
<td>27084</td>
</tr>
<tr>
<td>21</td>
<td>seccfg</td>
<td>8388608</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>vbmeta</td>
<td>8388608</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>vbmeta_system</td>
<td>8388608</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>vbmeta_vendor</td>
<td>8388608</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>protect2</td>
<td>11919360</td>
<td>37286</td>
<td>5304</td>
<td>37256</td>
</tr>
<tr>
<td>26</td>
<td>ffu</td>
<td>12058624</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>dtbo</td>
<td>16777216</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>flashinfo</td>
<td>16777216</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>gsort</td>
<td>16777216</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>gz1</td>
<td>16777216</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>gz2</td>
<td>16777216</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>expdb</td>
<td>20971520</td>
<td>578941</td>
<td>578941</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>md,uuc</td>
<td>23699456</td>
<td>41217</td>
<td>55528</td>
<td>66251</td>
</tr>
<tr>
<td>34</td>
<td>boot_param</td>
<td>27262976</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>metadata</td>
<td>33554432</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>nvmsg</td>
<td>33554432</td>
<td>136511</td>
<td>17737</td>
<td>144290</td>
</tr>
<tr>
<td>37</td>
<td>gsp</td>
<td>45088768</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>persist</td>
<td>50331648</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td>persistbak</td>
<td>50331648</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>boot</td>
<td>67108864</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>mdwata</td>
<td>67108864</td>
<td>2084305</td>
<td>1875755</td>
<td>846678</td>
</tr>
<tr>
<td>42</td>
<td>nvram</td>
<td>67108864</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>recovery</td>
<td>67108864</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>mdimg</td>
<td>134217728</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td>minidump</td>
<td>134217728</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td>exaid</td>
<td>402653184</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>47</td>
<td>cache</td>
<td>452988432</td>
<td>10000815</td>
<td>10000814</td>
<td>179</td>
</tr>
<tr>
<td>48</td>
<td>cust</td>
<td>1073741824</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>super</td>
<td>7516192768</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
| 50          | userdata| 20796375040| 8282720331| 17277213977| 12530880890| 656458204

Fig. 4. Resulting screen when reflashing a flash image after factory reset.
However, their metadata is different, which can imply that the time of reset may be read from there. Furthermore, the main path for Magisk changes to a different location after reset. Further inspection shows that this path actually changes on every reboot. In addition, there are a couple of files that seem to fulfill the same function, but get a new name after a factory reset. Specifically, filenames related to Google mobile services (GMS) tend to be generated using unique numbers on reset.

6. Discussion

In this section, key insights from the results are given, specifically on encrypted remnants and unencrypted system data. This is followed by recommendations for future work and some ethical considerations.

6.1. Key insights

One of the main insights that can be gathered from the results is that some user data is still available, although in encrypted form. Nevertheless, it should be noted that the master keys to decrypt the data are not recoverable in a straightforward way due to the precautions as described in Section 2.1. In addition, the majority of the bytes has been changed after reset. The most unmodified bytes in the same position are found on the Xiaomi Redmi 9 while the least amount of bytes are found on the Google Pixel 3a running Android 12. As only the encryption keys are securely erased from the phones this means that there is some trade-off between time and security. Nevertheless, when compared to previous work on this topic it has become more difficult to find forensically relevant data remnants. Even so, for some users it may be important that data is completely wiped. In that case it may be helpful if Android also provided an option to overwrite all user data with zeroes.

Next to the encrypted user data, there also stays some partially unencrypted system data on other partitions. In case of the klog partition some information may be deduced such as whether somebody has connected with the device. Furthermore, one could deduce how a device has been used by exploiting information such as the battery capability and calibration of certain sensors. There is also binary data left untouched on factory reset that may contain further information on how the device has been used.

Furthermore, although the keys have been securely erased, simple rollback of the keys such as by restoring a full flash drive image is not enough to get access to unencrypted data again. This is likely due to the relationship between the key that is in the hardware of the device and the metadata containing encrypted key information that is destroyed on factory reset.

Most partitions available on an Android device are not used to write data to by using a device for a few days. In addition, for some partitions it can be unclear what the actual function is. An example of this is the separately tested Bluetooth partition on Android 12. It may be that this is a leftover of older android versions that once used a specific partition scheme. Alternatively, it may be that very specific actions need to be taken for this partition to store data. If the latter is the case it is likely that this data also stays after a factory reset.

6.2. Future work

There are multiple directions that can be taken to elaborate on this research. First of all, the current approach leaves out inspection of the RAM. However, as inspection of the source code has shown there are some specific logs written to this memory. It can be interesting to see what kind of data may be leftover or reloaded into RAM after a factory reset.

Secondly, as there are multiple binary files that are modified during data creation but stay untouched on reset, more research could be done on the function of these files. Challenges for this task include missing or unclear documentation and the wide variety of implementations by different vendors.

The same holds for the userdata partition for which it is clear that some data gets not removed. Nevertheless, more work can be done on figuring out which data it is that stays and whether that is similar across devices and resets.

Another shortcoming of this research is that data was extracted while the device was powered on. It can be interesting to compare the outcomes that require the use of adb with physically untangling the device and trying to read bytes on a lower level.

In addition, with the current approach on the last Android 12 experiment, it is not always possible to make a clear distinction between data that changes on reboot and data that changes on reset. Future research should also focus on untangling this difference.

Lastly, more results may be obtained by doing a long-term observation on data creation on the devices, to check whether other partitions also change. In addition, the research can be extended to other phones as some things are vendor-specific.

6.3. Ethical considerations

In this section some ethical aspects of this research are discussed. Firstly, to avoid data contamination, new accounts for synthetic users have been created for every inspected device. However, to minimise the amount of accounts, whenever possible logins to new applications were done through the use of an already created account, such as the use of the created Google account to login to TikTok.

Furthermore, some proprietary forensics software, such as UFED and CM2, was used in this research. The workings of these tools are not transparent to the end user. As it can nevertheless have large consequences what these tools can extract from a device we think them to be relevant to include in the research. Moreover, public research on these kind of tools can contribute to a more open forensic environment.

Lastly, the research itself can have implications for end users, who can be confused about what information does and what information does not get erased from a device, and to which extend. However, we believe that by publishing information about this, people can at least make a more informed decision on what actions they want to take with a device.

7. Conclusion

Since smartphones can store large amounts of sensitive information, it is important that this information can be wiped if the owner of the data desires so. Factory reset is marketed as a way to achieve this. However, as shown in this research by resetting some modern Android 11 and 12 devices, user data may still be recovered if the encryption keys are somehow available, as not all bytes are wiped. In addition, there is some system data that does not get erased on other partitions, which may be used to infer information about how a device was used. Nevertheless, when compared to older devices it has become harder to extract forensically relevant data for modern smartphones.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
Data availability

Data will be made available on request.

Acknowledgments

This work is originally published as part of a research project for the Master Security and Network Engineering of the University of Amsterdam, coordinated by Francesco Regazzoni (psp03.nl). The research is conducted in cooperation with the Netherlands Forensic Institute. We are thankful to both institutions for providing knowledge and resources. This research did not receive any other specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References


Wu, J., Github Contributors, 2022. Magisk - the magic mask for android. URL: https://gopigo.wu.github.io/Magisk/.