Security Analysis of Android Factory Resets

Laurent Simon
University of Cambridge
lmrs2@cam.ac.uk

Ross Anderson
University of Cambridge
rja14@cam.ac.uk

Abstract

With hundreds of millions of devices expected to be traded by 2018[1], flaws in smartphone sanitisation functions could be a serious problem. Trade press reports[2] have already raised doubts about the effectiveness of Android "Factory Reset", but this paper presents the first comprehensive study of the issue. We study the implementation of Factory Reset on 21 Android smartphones from 5 vendors running Android versions v2.3.x to v4.3. We estimate that up to 500 million devices may not properly sanitise their data partition where credentials and other sensitive data are stored, and up to 630M may not properly sanitise the internal SD card where multimedia files are generally saved. We found we could recover Google credentials on all devices presenting a flawed Factory Reset. Full-disk encryption has the potential to mitigate the problem, but we found that a flawed Factory Reset leaves behind enough data for the encryption key to be recovered. We discuss practical improvements for Google and vendors to mitigate these risks in the future.

I. Introduction

The extraction of data from resold devices is a growing threat as more users buy second-hand devices[1]. A healthy second-hand market is valuable for vendors as people are more willing to buy expensive new devices if they know they can trade them in later[3]. So data sanitisation problems have the potential to disrupt market growth. If users fear for their data, they may stop trading their old devices, and buy fewer new ones; or they may continue to upgrade, but be reluctant to adopt sensitive services like banking or healthcare apps, thereby slowing down innovation. Last but not least, phone vendors may be held accountable under consumer protection or data protection laws. To sanitise their devices, users are advised to use the built-in "Factory Reset" function on device disposal. Previous reports [1] have raised occasional doubts about the effectiveness of the implementations of this in Android, with claims that data can sometimes be recovered. This paper provides the first comprehensive study of the problem. It (*i*) quantifies the amount of data left behind by flawed implementations, (*ii*) provides a detailed analysis of affected devices (versions, vendors), (*iii*) reveals the drivers behind the flaws, and (*iv*) discusses practical solutions to mitigate these problems (Section III and Section VI). Concretely, we find that a flawed Factory Reset lets an attacker access a user's Google account and its associated data backed up by Google services, such as contacts and WiFi credentials (Section IV). Our study unveils five critical failures: (*i*) the lack of Android support for proper deletion of the data partition in v2.3.x devices; (*ii*) the incompleteness of upgrades pushed to flawed devices by vendors; (*iii*) the lack of driver support for proper deletion shipped by vendors in newer devices (e.g. on v4.[1,2,3]); (*iv*) the lack of Android support for proper deletion of the internal and external SD card in all OS versions; (*v*) the fragility of full-disk encryption to mitigate those problems up to Android v4.4 (KitKat).

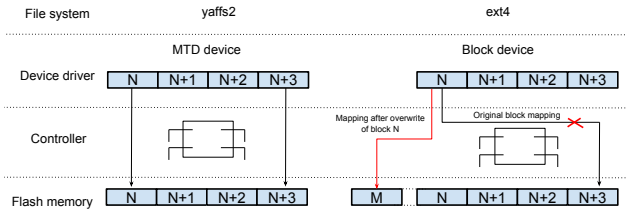


Fig. 1. Yaffs2 with raw flash access vs. Ext4 with logical block access. On an eMMC (right), the logical block N is mapped to the physical block $N+3$ by the eMMC, and remapped to block M after an overwrite. MTD stands for Memory Technology Device.

In summary, our contributions are as follows:

- We present the first comprehensive study of Android Factory Reset, by studying 21 Android smartphones from 5 vendors running Android versions from v2.3.x to v4.3.
- We highlight critical failures such as the lack of support by the Android OS and/or vendor-shipped drivers for secure deletion. These problems may affect devices even after upgrades are received.
- We discuss practical improvements for Google and vendors to mitigate these risks in the future.

II. Technical Background

A. Flash & File Systems

Smartphones use flash for their non volatile memory storage because it is fast, cheap and small. Flash memory is usually arranged in pages and blocks. The CPU can read or write a page (of typically 512+16 to 4096+128 data+metadata bytes), but can only erase a block of from 32 to 128 pages. Each block contains both data, and "out-of-band" (OOB) data or metadata used for bad block

[1] www.forbes.com/sites/connieguglielmo/2013/08/07/used-smartphone-market-poised-to-explode-apple-iphone-holding-up-better-than-samsung-galaxy

[2] blog.avast.com/2014/07/08/tens-of-thousands-of-americans-sell-themselves-online-every-day

[3] blogs.which.co.uk/technology/phones-3/mobile-phone-price-tracking/

management, error correcting codes (ECC) and file system bookkeeping. Blocks must be erased prior to being written to; yet flash chips support only a finite number of program-erase cycles, so wear-levelling algorithms are used to spread the erase and write operations uniformly over all blocks. It is also worth mentioning that flash storage is usually over-provisioned, i.e. a chip has more internal space than it advertises to the OS, in anticipation of bad blocks and to further reduce wear.

Early Android devices (like Froyo, Android v2.2.x) used the flash-aware file system yaffs2 that handles wear-levelling and error correction. Since Gingerbread (v2.3.x), devices generally come with an embedded MultiMediaCard (eMMC) with proprietary wear-levelling algorithms implemented in hardware. An eMMC does not give the OS access to the raw flash, but exposes a block-like device, on top of which the OS lays a block file system like ext4. Blocks only give the OS a logical view of memory. Internally, each block is mapped to a corresponding physical block on the flash by the eMMC controller. When data in a logical block N is updated by the OS (Fig. 1, right), the corresponding physical block would typically be added to a "to-be-erased" list, then remapped to a "clean" physical block M, thereby leaving the original block content untouched. Therefore, to achieve secure deletion, eMMC standards define specific commands that must be used to remove data physically from memory.

B. Secure Deletion Levels

When removing a file, an OS typically only deletes its name from a table, rather than deleting its content. The situation is aggravated on flash memory because data update does not occur in place, i.e. data are copied to a new block to preserve performance, reduce the erasure block count and slow down the wear. Therefore, there exist various recommendations, guidelines and standards for sanitising data. The following levels of data sanitisation are relevant depending on the threat model considered [2].

The highest level of sanitisation is *analog sanitisation*, this degrades the analog signal that encodes information, so that its reconstruction is impossible even with the most advanced sensing equipment and expertise. For example, NIST's *Guidelines for Media Sanitization* (NIST 800-88) have a "purging" level that corresponds to analog sanitisation.

The second level is *digital sanitisation*. Data in digitally sanitised storage cannot be recovered via any digital means, including the bypass or compromise of the device's controller or firmware, or via undocumented drive commands. Unimpeded physical access to a flash chip and the manufacturer's data sheet may be required, but these are not available for typical smartphones.

The third level is *logical sanitisation*. Data in logically sanitised storage cannot be recovered via standard hardware interfaces like standard eMMC commands. For example, this corresponds to NIST 800-88's "clearing" level. For cellphones and PDAs, NIST 800-88 suggests "clearing" them by manually deleting data followed by a Factory Reset.

In this study, we consider cheap data recovery attacks that require neither expensive equipment to physically extract data from the chip nor specific per-chip knowledge. Only attacks that are oblivious to the underlying chip scale across devices and are likely to be profitable if exploited at scale. Therefore in the rest of this document, by "proper" or "secure" sanitisation we mean logical sanitisation.

C. Linux Kernel Deletion APIs

Privileged userspace programs can erase flash blocks through the *ioctl()* system call exposed by the Linux kernel. On raw flash, the *ioctl*'s *MEMERASE* option provides digital sanitisation. On an eMMC there are two possible options. The first is *BLKDISCARD* which provides no security guarantees. Internally, the kernel generally implements *BLKDISCARD* by passing the eMMC command "DISCARD" or "TRIM" to the chip. These do not request the eMMC to purge the blocks. They simply indicate that data is no longer required so that the eMMC can erase it if necessary during background erase events. They would typically be used when *unlink*ing a file. The second *ioctl* option is *BLK**SEC**DISCARD* and provides "secure" deletion. The kernel implements *BLKSECDISCARD* by passing to the chip one of the many "secure deletion" commands defined by eMMC standards. The actual eMMC command used depends on support by the chip. There is an "ERASE" command for logical sanitisation and commands such as "SECURE TRIM", "SECURE ERASE" and "SANITIZE" for digital sanitisation.

D. Data Partitions

Android smartphones share three common partitions for data storage (Fig. 2). The first is the data partition, generally mounted on */data/*, that hosts apps' private directories. An app's private directory cannot be read or written to by other apps, so it is commonly used to store sensitive information such as login credentials. On older phones with a small data partition, one can also install apps on an external SD card; but this is usually not the default behaviour.

The second partition storing user data is the internal (primary) SD card. Despite its name, it is not an SD card per se, but a partition physically stored on the same chip. It is generally mounted on */sdcard/* or */mnt/sdcard/*, which is readable/writeable by applications. It is generally FAT-formatted or emulated with the Filesystem in Userspace (FUSE). In the latter case, files are physically stored on the data partition. The internal SD card is mainly used to store multimedia files made with the camera and microphone; it is generally exposed to a computer connected via USB – via Mass Storage, Media Transfer Protocol (MTP) or Picture Transfer Protocol (PTP).

The last partition containing user data is the external, removable SD card. It offers similar functionality to the internal SD card, but can be physically inserted and removed by the user. If there is no internal SD card on the device, the external one becomes the "primary SD card"; otherwise it is

called the "secondary SD card". The primary and secondary SD cards are sometimes referred to as "external storage".

Some devices also have hardware key storage. When supported, it is used principally by the default Account Manager app.

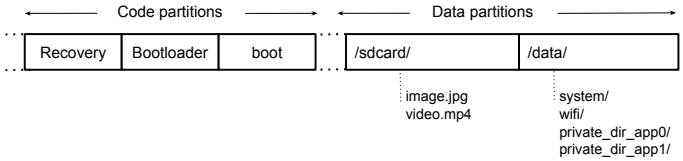
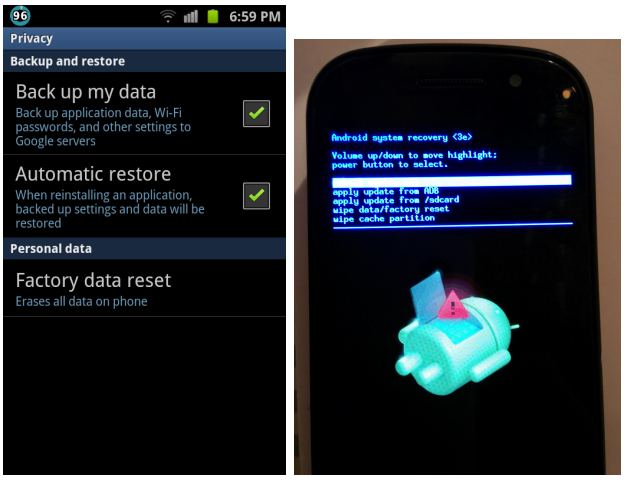


Fig. 2. Common Android partitions. Each rectangle represents a partition on the same flash storage.



(a) Factory Reset in Settings. (b) Factory Reset in Recovery.

Fig. 3. Factory Resets provided by most devices.

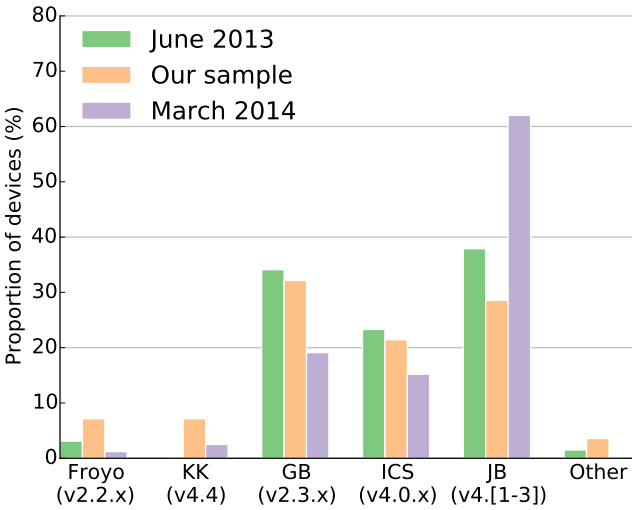


Fig. 4. Android OS distribution for Froyo, KitKat (KK), Gingerbread (GB), Ice Cream Sandwich (ICS) and Jelly Bean (JB).

III. Analysis of Android Factory Resets

A. Methodology

Between January and May 2014, we bought second-hand Android phones from eBay and from phone recycling companies in the UK, randomly selecting devices based on availability. As the project might possibly uncover personal information, it was first submitted to our ethics process for approval. In the rest of the paper, we refer to a "device" as the unique pair (phone name, OS version).

We studied 26 different devices (list provided in Appendices) from 5 vendors[4], running Android versions ranging from v2.2 (Froyo) to v4.3 (Jelly Bean). These Android versions are resold more frequently and are being traded today. Fig. 4 shows the distribution of Android versions for our samples, compared to active devices in June 2013 and in March 2014, as reported by Google's Dashboard[5]. Our samples are not representative of the OS version distribution at the time of acquisition, but are similar to the world-wide distribution 6 months earlier, in June 2013 (as one might expect from the time taken for new phones to enter the secondhand market). In September 2013, Google announced that one billion devices had been activated[6]. This represents 340M Gingerbread (GB, v2.3.x) devices, 230M Ice Cream Sandwich (ICS, v4.0.x) devices and 380M Jelly Bean (JB, v4.[1-3]) devices. Our samples are representative of the second-hand market at the time of acquisition. We use the number of devices and their distribution across versions (in June 13) to approximate the number of devices affected by flawed Factory Resets throughout the following sections. Estimates are based on the assumption that each device accounts for the same percentage of the overall device population. This is not true in practice, and this is a limitation of our evaluation.

We tested each partition of interest (Section II-D) by overwriting it with unique identifying patterns. Then we sanitised the device with the Factory Reset function and attempted to recover the written patterns. We investigated the Factory Reset suggested by vendors[7], that is, the one in Android Settings (Fig. 3(a) – recommended by 90% of vendors) and the one in Recovery/Bootloader mode (Fig. 3(b) – recommended by 70% of vendors if the phone cannot be booted). The Recovery and Bootloader modes are special modes a phone can be booted into via a combination of hardware keys. The interested reader can refer to the Appendices for details on each step (pattern generation, flash reading and writing, pattern recovery). In the rest of the study, we refer to a "Factory Reset" or a "wipe" interchangeably.

B. Results and Discussion

1) Preliminary Results: We found that the sanitisation of external storage occurs only if a user selects the additional

[4]Samsung, HTC, LG, Motorola and Google
[5]developer.android.com/about/dashboards/index.html
[6]plus.google.com/+SundarPichai/posts/NeBW7AjT1QM
[7]Alcatel, BlackBerry, Apple, HTC, Samsung, Huawei, Nokia, Motorola, Lenovo, Sony, LG

TABLE I
CHRONOLOGY OF FACTORY RESET IMPLEMENTATIONS IN AOSP.

Code	Partition	OS Versions				
		Froyo (2.2.x)	GB (2.3.x)	ICS (4.0.x)	JB (4.1.[1-3])	KK (4.4)
Android	primary SD	*format()* ✗	*format()* ✗	*format()* ✗	*ioctl(BLKDISCARD)* ✗	*ioctl(BLKDISCARD)* ✗
	secondary SD	none ✗	none ✗	none ✗	none ✗	none ✗
Recovery	data	*ioctl(MEMERASE)* ✓	*ioctl(BLKDISCARD)* ✗	*ioctl(BLKSECDISCARD)* ✓	*ioctl(BLKSECDISCARD)* ✓	*ioctl(BLKSECDISCARD)* ✓

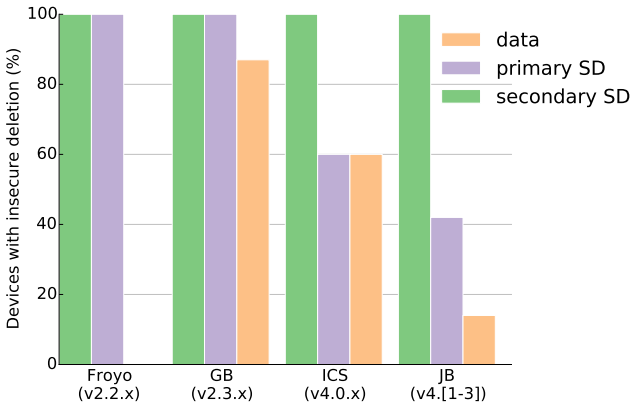


Fig. 5. Percentage of devices with flawed logical sanitisation. Results for primary and secondary SD cards (i.e. external storage) implicitly assume the use of the Factory Reset in Settings since the Recover/Bootloader Factory Reset only sanitises the data and cache partitions.

option "External Storage" in the Factory Reset in Settings. In this case, the Android OS first performs the sanitisation of external storage and then reboots into Recovery/Bootloader mode where the data partition and the cache partition (which contains mainly optimized .odex java classes) are sanitised. If a user Factory Resets his device with Recovery/Bootloader instead of Settings, external storage is not sanitised – subtle difference between devices are provided in the following paragraphs. Fig. 5 shows the results of built-in sanitisation for all devices studied.

2) Sanitisation of the data partition: Recall from Section II-D that the data partition stores sensitive information from Google and third-party apps, such as credentials.

On devices running the oldest version of Android we studied (Froyo, v2.2.x), the data partition was logically sanitised. These devices use a raw flash with the yaffs2 file system (Section II-A), where it is infeasible to reformat a partition without first properly sanitising it. The Android Open Source Project (AOSP[8]) reveals the use of the *ioctl*'s *MEMERASE* command by the Recovery mode (Table I), this provides digital sanitisation and confirms the results of Fig. 5.

From Gingerbread onwards ($\geq$ v2.3.x), all devices we encountered use eMMCs (except one), where it is possible to reformat a partition without properly sanitising it first. Fig. 5 shows that about 90% (≈300M) of Gingerbread (v2.3.x) devices sanitise the data partition insecurely, in that at most a few hundred MB are deleted, representing between 60% and 99.9% of the data partition depending on its size. The Android Open Source Project (AOSP) reveals the use of the *ioctl*'s *BLKDISCARD* command by the Recovery mode (Table I), this does not provides logical sanitisation and therefore confirms the results of Fig. 5. The only device in our sample that properly deletes the data partition is the HTC Wildfire S (Gingerbread); and it is because it uses yaffs2 rather than an eMMC.

As shown in Table I, the following Android version (ICS, v4.0.x) marked the introduction of logical sanitisation support via *BLKSECDISCARD* in the AOSP code. This contradicts the results from Fig. 5 that show that 60% (≈140M) of ICS (v4.0.x) devices incorrectly sanitise the data partition. Many of these ICS devices in our sample were initially released with GB (v2.3.x) and received upgrades to ICS (v4.0.x). We verified that the phone binaries indeed contained the newest code from AOSP, i.e. with logical sanitisation support. We then turned our attention to lower-level code, and found that vendor upgrades likely omitted device drivers necessary to expose the logical sanitisation functionality from the underlying eMMC. In practice, this means that the secure command *BLKSECDISCARD* is not supported by *ioctl*, i.e. it returns errno 95 (*EOPNOTSUPP*). It could be the case that the eMMC itself does not support secure deletion, but we think this is unlikely since the 2007's 4.2 eMMC standard[9] already provided the compulsory "ERASE" command for logical sanitisation. We found evidence corroborating this claim on certain phones at least: when unlocking the Bootloader[10] of the HTC Sensation XE, the data partition was properly sanitised whereas it was not during a Factory Reset. Devices affected include the Samsung Galaxy S Plus, S (25M units sold[11]) and S2 (40M units sold[11]), and the HTC Sensation XE. Only the Google Nexus S in our sample properly sanitised its data partition after receiving upgrades to ICS. The problem is likely to persist after further upgrades to Jelly Bean (JB, v4.[1-3]), although we could not ascertain this as our samples did not contain such devices.

Besides upgrade issues, devices shipped with newer Android versions such as ICS (v4.0.x) and JB (v4.[1-3]) are not free of problems either (Fig. 5). They too are not always shipped with proper device drivers for secure deletion. For example, the LG Optimus L5 shipped with ICS did return errno *EOPNOTSUPP* when we attempted a secure deletion. More intriguing, the Motorola Razr I, shipped with JB (v4.[1-3]), did not return any errors, but the secure deletion

[8]android.googlesource.com

[9]www.jedec.org/standards-documents/docs/jesd-84-b42

[10]This procedure lets users install custom software, but wipes data to prevent a thief from recovering user's data via forensic software

[11]www.tomshardware.com/news/Samsung-Galaxy-S-S2-S3,20438.html

resulted in no block being deleted at all. Due to these driver issues, Fig. 5 shows 15% (≈55M) of JB (v4.[1-3]) devices improperly sanitise the data partition.

3) Sanitisation of the Primary SD Card: Recall from Section II-D that the primary SD card corresponds either to the internal one or to a physically removable one in the absence of the former. It mainly hosts multimedia files made with the camera and possibly third-party apps.

We found that no Froyo (v2.2.x) and Gingerbread (GB, v2.3.x) devices we examined logically sanitised their primary SD card. This represents more than 340M devices. The AOSP code reveals that in these versions, Android only formats the primary SD card with a call to *Fat::format()* (Table I), this confirms the results from Fig. 5. In practice, a few dozen MB at most are logically sanitised.

As depicted in Table I, the AOSP reveals no code changes in the sanitisation of the primary SD card in the following Android version (ICS, v4.0.x). Yet, Fig. 5 shows about 40% (≈90M) of ICS devices properly sanitise their primary SD card (i.e. ≈140M of devices do not). One may conclude that vendors have customised the AOSP code and added secure deletion support for the primary SD card, but this is incorrect. This logical sanitisation is due to the following reasons: (*i*) the primary SD card on these phones corresponds to the internal one, (*ii*) these devices use an emulated SD card physically stored on the data partition (Section II-D) and (*iii*) proper sanitisation of the data partition is implemented as per AOSP, and so gets "inherited" by the primary SD card. Only when these three fortunate conditions are met can we be confident that the primary SD card is logical sanitised.

The following Android version (JB, v4.[1-3]) marked the addition of insecure deletion via *ioctl*'s *BLKDISCARD* command. This confirms why 40% (≈150M) of devices still fail to logically sanitise their primary (internal or external) SD card. At the time of writing, we are not aware of any changes to the handling of the primary SD card, therefore we expect these results to hold on all other Android versions.

4) Sanitisation of the Secondary SD Card: In our sample, no devices properly sanitised the secondary (external) SD card. We found that Android generally does not attempt to sanitise it at all (Table I), which explains the results from Fig. 5.

5) Vendor Customisation Inconsistencies: Besides the various differences of sanitisation between versions and models already highlighted, we discovered other vendor issues. For example, we mentioned that only the Factory Reset in Settings provides an option to sanitise the primary SD card (Section III-B1). Therefore one might advise users to use Settings rather than Recovery to sanitise devices. Unfortunately, vendor customisations sometimes make Recovery more reliable. For example, the two HTC One-series phones in our sample properly sanitised their primary (internal) SD card in Recovery (contrary to AOSP), but not in Settings (as we would expect from the AOSP source code). It is likely that this result holds for many of the other HTC One-series devices. This also violates HTC guidelines: on its website[12], it suggests its users to first try Settings, and resort to Recovery only "if you can't turn HTC One [X] on or access settings". HTC has put up a note to discharge itself of any responsibility: "A factory reset may not permanently erase all data from your phone, including personal information".

6) eMMC implementation of logical sanitisation: In general, we found that devices in our sample logically sanitised all bytes requested through the *ioctl* command, except for one phone: the Google Nexus 4. This has an 6189744128B data partition, fully used by the file system. The last 16KB were not sanitised and fully recoverable about 20% of the time after a Factory Reset. Our hypothesis is that this might be a bug in the eMMC itself (or its corresponding drivers), since we have not seen similar problems in other devices.

7) Number of logical blocks to sanitise: If issued with the non-secure sanitisation command "DISCARD", an eMMC applies a "don't care" policy to the block. According to the standard, "the original data may be remained partially or fully accessible to the host dependent on device". This further means that data originally exposed at a logical block located at logical offset LO_{org}, could be re-mapped at a different logical offset, say $LO_{remapped}$ as shown in Fig. 6. If the file system of size M does not fill the entire partition, there is a risk that the deleted block's data "crosses" the filesystem boundary. Therefore if the sanitisation only attempts to securely sanitise the blocks used by the file system ($[0, M]$ in our example), it is plausible that some remapped blocks (within $]M, S]$) would not be purged. We stress that we have not found evidence of this happening in our sample devices. However in order to reduce the possibility of this happening, we suggest vendors erase the entire partition, rather than just the part used by the file system. In our device sample, the HTC One-series phones left a few MB of space at the end of the data partition. It would be more cautious to sanitise the entire partition. Similarly, the AOSP code currently truncates the partition size to a multiple of 4096B when creating the file system and when computing the size to wipe during Factory Reset. Sanitising the entire partition would be more prudent.

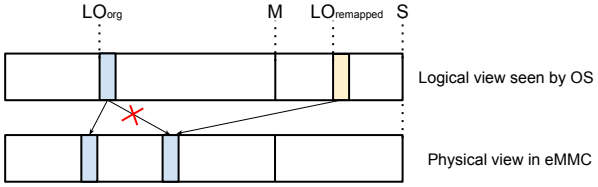


Fig. 6. Cross file system boundary example for the data partition.

IV. Data Recovery in Practice

Our objective here is not to implement new tools and algorithms, but to evaluate the feasibility and scalability of

[12]www.htc.com/us/support/htc-one-s-t-mobile/howto/315367.html

TABLE II
PRACTICAL RECOVERY OF DATA

	Storage	Extraction	Percentage Devices	Attack usage	Comments
Phone Owner	data partition	automated pattern matching on third-party apps like Facebook or Google accounts app	100%	contact user to blackmail (assuming compromising data is recovered)	extraction through default Phonebook app's data generally requires some human intervention
Installed apps	data partition	automated pattern matching on */data/system/packages.xml*	100%	identify high-value targets / adjust forensic strategy based on easy-to-retrieve app-formatted data	
Contacts	data partition	automated pattern matching on third-party apps like Facebook or WhatsApp (Fig. 7(a))	100%	sell in underground markets	associating names to contact details in Phonebook app data generally requires some human intervention
Browsing	data partition	automated pattern matching	100%	blackmail user	
Credentials	data partition	automated pattern matching for browser cookies, WiFi (Fig. 7(b)), Google (Fig. 7(c)) and other apps.	100%	sell in underground markets	Google master token recovered 80% of the time
Multimedia	data partition, primary SD	automated file carving (*Photorec* [3]) for camera-made images and video and web thumbnails	100%	blackmail user	
Conversations	data partition	automated pattern matching for third party messaging apps and emails	100%	blackmail or sell in underground markets	identifying SMS required some human intervention, emails recovered in 80% of devices but only a few

attacks. Data on Android smartphones is generally stored in SQLite databases and text-like files.

To extract SQLite files, we initially investigated the use of file carving with *Scalpel* [4]. File carving is the practice of searching for files by leveraging the knowledge of their content and structure, rather than relying on file system metadata[13]. In practice, we found that the database file header and its content were not always contiguously allocated, probably because of repetitive updates. Furthermore, each applications database has a different layout, and records are not always located contiguously on the partition. Therefore parsing the data from such fragments is not fully reliable with simple techniques. So we used SQLite file carving only as a preliminary step.

We quickly realised that most of the data (including database files), exhibits specific and distinct formats. For instance, the list of installed apps is stored in the file */data/system/packages.xml* with a well-defined structure of the form *<package name="com.mycompany.myapp" codePath="/system/app/myapp.apk" ...>*. Therefore we primarily used pattern matching to recover data on all second-hand phones from Section III-A, and file carving solely to extract multimedia files.

A. General Results

We hunted for the information shown in Table II in devices with a flawed Factory Reset. For example, we recovered some "Conversations" (SMSes, emails, and/or chats from messaging apps) in all devices (Column "Percentage Devices") using pattern matching (column "Extraction"). Compromising conversations could be used to blackmail victims (column "Attack usage"). Gmail app emails were stored compressed. By searching for relevant headers, we were able to locate candidates and then decompress them. We found emails in 80% of our sample devices, but generally only a few per device (column "Comments").

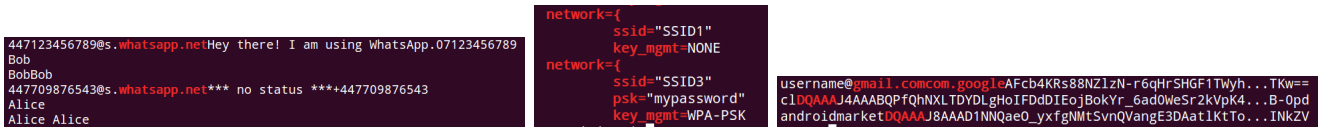
[13]A simple file carver would search for files' headers and footers. More advanced ones would also look for file fragment candidates and piece them together using certain properties of the underlying data.

B. Case Study: Hijacking Google Accounts

To improve usability and user engagement, most smartphone apps replace passwords with authentication tokens the first time a user enters his password. After the first password-based authentication, users are automatically logged-in with the authentication token; emails can be retrieved, calendar notifications downloaded, etc. without user intervention. These tokens are often stored on non-volatile flash storage on the data partition. Some Google tokens for the account *username@gmail.com*, are shown in Fig. 7(c). The first one, which we call the "master token", is the long random string starting with "AFc". It gives access to most Google user data. As a test, we Factory Reset our own phone, then recovered the master token. We then created the relevant files and rebooted the phone. After the reboot, the phone successfully re-synchronised contacts, emails, and so on. We recovered Google tokens in all devices with flawed Factory Reset, and the master token 80% of the time. Tokens for other apps such as Facebook can be recovered similarly. We stress that we have never attempted to use those tokens to access anyone's account.

C. Possible Attackers

Individuals buying devices on auction websites such as eBay are possible attackers. They need to spend a non-negligible time to bid and follow up on auctions. Furthermore, they have to pay a few dollars for commission and shipping fees for each device. So low-value data like contacts and email addresses do not seem profitable. Recovery and analysis of conversations and images (to blackmail victims) would generally require human intervention or more advanced tools, with the possible exception of browser history where simple keyword search can be effective. Blackmailing users requires enough devices to hit compromising data and enough users to hit a gullible mark. But this requires (*i*) a significant time investment to bet on/follow items and (*ii*) great logistics to buy, process, and re-sell devices. Therefore we think that only people with



(a) Whatsapp contacts with name and phone number. (b) WiFi passwords. (c) Android tokens.

Fig. 7. Example of pattern matching results.

enough time on their hands could make extra cash on top of an existing income this way. In general, high-value data like banking credentials appear likely to be the most profitable criminal option.

Smartphone salesmen in brick-and-mortar shops can reduce the cost of device acquisition, since it may be part of their job to receive second-hand devices and they can scan these devices without paying auction and shipping costs. So low-value data may be profitable for them. As it is common for merchants to talk to customers, they could also identify higher-than-average-value targets suitable for blackmail. Attackers who can add forensic software to the recycling chain may further increase the number of devices processed, and the amount of low-value data recovered. Attackers with access to corporate devices could also gain access to high-value data. On the other hand, shop staff will be easier for the police to identify and arrest once a complaint of blackmail is made. Thieves are yet another category of attackers: data are less likely to be deleted from a stolen device, so they are out of scope of this paper.

Although a lot of data can be recovered using pattern matching, this does not necessarily translate into actual profitability.

V. Alternative Sanitisation Methods

We considered the following methods to mitigate flawed Factory Resets.

Filling up the partition of interest with random-byte files, in the hope of overwriting all unallocated space, could be achieved via third-party non-privileged apps after the built-in Factory Reset. This would require the app to be installed manually by users after a Factory Reset is performed. Otherwise, Google credentials stored on the file system (necessary to install an app from Google Play) will not be erased by the procedure. This sanitisation procedure also adds an additional layer of uncertainty because it uses the file system rather than direct flash access. File systems also vary across devices and may be proprietary (such as Samsung's RFS). We therefore felt this option would not scale reliably across devices, and we discarded this method in our tests.

Overwriting the entire partition "bit-by-bit" once did provide logical sanitisation for all devices and all partitions we studied; it is therefore a reliable alternative. The drawback of this method is that it requires privileged (i.e. root – see Appendices) access to devices in practice. Therefore it is likely to put off ordinary users. This method does not provide thorough digital sanitisation, since the flash is over-provisioned – but an attacker cannot recover data using public APIs exposed by the Linux kernel. Furthermore, the over-provisioning could differ even for instances of the same device, for example if different grades of flash were used. Since we are concerned only with massively scalable attacks, we did not consider this issue further, but firms with high assurance requirements might have to unless they can use encryption, which we consider next.

Enabling Full Disk Encryption (FDE) on first use of the device would be more appropriate for ordinary users if devices support it. Enabling FDE only before performing a Factory Reset (as suggested by Google[14]) may only provide logical sanitisation, not thorough digital sanitisation (plain-text data could still be present on the flash as it is over-provisioned). FDE was introduced in ICS (v4.0.x) so it cannot help the large number of affected GB (v2.3.x) devices. On one HTC phone running GB (v2.3.x), we found an encryption option, but it left all the data behind. We assume this was a vendor customisation and may only encrypt allocated space. FDE for the internal SD card is not supported on all phones, and not all v4.x devices support FDE on the data partition despite AOSP's support. As a rule of thumb, only devices with an emulated internal SD card inherit the "encryption support" from the data partition when supported. On supported Android versions, the encryption key is stored encrypted with a key derived from a salt and a user-provided PIN (or password). This encrypted blob is referred to as the "crypto footer" in the AOSP source code. An attacker who gains access to the crypto footer has enough information to brute-force the user's PIN offline. The footer is stored in a dedicated partition or in the last 16KB of the data partition – the exact location is configured by vendors through the "encryptable=" option of the Android fstab file. In either case, we found that the footer was not erased after a flawed Factory Reset. Consequently, to logically sanitise a device with encryption, it is essential to select a strong password to thwart offline brute-force attacks. As most people just use a 4-6 digit PIN, it would usually be trivial to brute-force.

Mobile Anti-Virus (MAV) apps have a "remote wipe" feature to sanitise data on lost or stolen smartphones. We refer the reader to a study by Simon and Anderson [5]: at the time of the publication (2015), they found that remote wipe functions "are not an alternative to a flawed built-in

[14] www.bbc.com/news/technology-28264446

Factory Reset".

VI. Recommendations

For vendors, our recommendations are to use a recent eMMC with support for digital sanitisation, and to properly expose it in the Bootloader, Recovery and Android kernels. More generally, previous research has shown that vendors' customisations are a source of security problems [6], [7], [8]. Therefore, we provide the following guidelines to the AOSP developers, hoping they can reduce the chance of slip-ups in the future:

1) Use an emulated primary SD card: this ensures that only one partition needs to be properly sanitised on the phone, reducing the space for mistakes.
2) Erase the entire partition, not only the part explicitly used by the file system. This reduces the chance of unfortunate surprises due to eMMC wear-levelling block management and deletion implementation problems.
3) Implement sanitisation of all partitions in one place only; for example in Recovery mode or Bootloader mode; and have Settings simply have the phone reboot into the appropriate mode with the right parameters. Having the relevant code in one place eases testing and reduces possible mistakes.
4) Expose an option to have the Recovery mode perform a sanitisation validation, by reading back the entire partition and checking it.
5) Provide test units for vendors to test sanitisation in the Android Compliance Suite Test (CST). Have the tests fail if secure sanitisation fails, e.g. if not supported or if the verification step 4 fails.
6) Do not resort to an insecure sanitisation if the secure one fails - as it is currently the case[15].
7) Before a Factory Reset takes place, a broadcast Intent could be sent to apps, so that they could take necessary steps to invalidate their credentials – assuming that Internet connection is available.
8) Store the encryption metadata at the start of the data partition in a crypto header, rather than at the end in a crypto footer. This reduces the risk of dictionary attacks in the event of flawed sanitisation, since the first blocks are generally overwritten during partition formatting. Storing the metadata on the data partition also ensures that there is only one partition to take care of, as above.

VII. Related Work

Some previous studies looked at data recovery and users' practices. Garfinkel and Shelat [9] studied sanitisation practices of second-hand magnetic hard disks and found no standard practice in 2003, with only 9% of disk properly sanitised. Breeuwsma et al. [10] discussed data acquisition using low-level imaging techniques. Luke and Stokes [11], Billard and Hauri [12] and Lewis and Kuhn [13, Chapter 5] devised techniques to recover respectively generic files and video files from flash-based media. Walls et al. [14], [15] devised algorithms for forensic triage of non-sanitised feature phones and smartphones.

Other studies looked at how secure deletion can be performed. Wei et al. [2] empirically assessed the reliability of hard drive techniques and of the SSDs' built-in sanitisation commands. They found that all existing hard drive techniques for individual file sanitisation fail; this was also reported by Freeman and Woodward [16]. Gutmann [17] looked at techniques to achieve analog sanitisation on hard drives, and devised the well-known 35-pass overwrite technique. File sanitisation techniques generally rely on data encryption: storage sanitisation is then performed by securely erasing keys using built-in commands or raw flash access [18], [19], [20].

On Android, Mahajan et al. [21] used commercial software on 5 different devices to recover Viber and Whatsapp chats from non-sanitised Android smartphones. Simon and Anderson [5] studied the reliability of remote wipe functions provided by mobile anti-virus apps.

To the best of our knowledge, this is the first comprehensive study of Android Factory Reset functions.

VIII. Conclusion and Future Work

We presented the first thorough analysis of Android factory reset functions by studying 21 Android smartphones from 5 vendors running Android versions v2.3.x to v4.3. We presented a detailed and chronological analysis of flaws across Android versions. We tracked these issues to (*i*) Android failures, (*ii*) inadequate vendor upgrade practices, and (*iii*) improper vendor integration and testing practices.

Future research should continue to investigate the level of security provided by smartphones' built-in sanitisation functions, to see whether the situation improves following the disclosures reported here. It could also investigate the level of security provided by these, i.e. whether they provide digital sanitisation or not.

IX. Acknowledgements

The authors thank the anonymous reviewers for their valuable suggestions and comments. This work was supported by the Samsung Electronics Research Institute (SERI) [grant number RG67002].

[15] source.android.com/devices/tech/security/best-practices.html

References

[1] R. Schwamm and N. C. Rowe, "Effects of the factory reset on mobile devices," in *The Journal of Digital Forensics, Security and Law (JDFSL)*, 2014.
[2] M. Y. C. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably erasing data from flash-based solid state drives.," in *FAST*, vol. 11, pp. 8–8, 2011.
[3] "Photorec." http://www.cgsecurity.org/wiki/PhotoRec.
[4] G. G. Richard III and V. Roussev, "Scalpel: A frugal, high performance file carver.," in *DFRWS*, 2005.
[5] L. Simon and R. Anderson, "Security analysis of consumer-grade anti-theft solutions provided by android mobile anti-virus apps," in *3rd Mobile Security Technologies Workshop (MoST)*, 2015.
[6] A. Pereira, M. Correia, and P. Brandão, "Usb connection vulnerabilities on android smartphones: Default and vendors customizations," in *Communications and Multimedia Security*, pp. 19–32, Springer, 2014.

[7] X. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The peril of fragmentation: Security hazards in android device driver customizations," in *IEEE Symposium on Security and Privacy*, 2014.
[8] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on android security," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 623–634, ACM, 2013.
[9] S. L. Garfinkel and A. Shelat, "Remembrance of data passed: A study of disk sanitization practices," *IEEE Security & Privacy*, vol. 1, no. 1, pp. 17–27, 2003.
[10] M. Breeuwsma, M. De Jongh, C. Klaver, R. Van Der Knijff, and M. Roeloffs, "Forensic data recovery from flash memory," *Small Scale Digital Device Forensics Journal*, vol. 1, no. 1, pp. 1–17, 2007.
[11] J. Luck and M. Stokes, "An integrated approach to recovering deleted files from nand flash data," *Small Scale Digital Device Forensics Journal*, vol. 2, no. 1, pp. 1941–6164, 2008.
[12] D. Billard and R. Hauri, "Making sense of unstructured flash-memory dumps," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1579–1583, ACM, 2010.
[13] A. B. Lewis, "Reconstructing compressed photo and video data," Tech. Rep. UCAM-CL-TR-813, University of Cambridge, Computer Laboratory, Feb 2012.
[14] R. J. Walls, E. Learned-Miller, and B. N. Levine, "Forensic Triage for Mobile Phones with DEC0DE," in *Proc. USENIX Security Symposium*, Aug 2011.
[15] S. Varma, R. J. Walls, B. Lynn, and B. N. Levine, "Efficient Smart Phone Forensics Based on Relevance Feedback," in *Proc. ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, Nov 2014.
[16] M. Freeman and A. Woodward, "Secure state deletion: Testing the efficacy and integrity of secure deletion tools on solid state drives," in *Australian Digital Forensics Conference*, p. 65, 2009.
[17] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proceedings of the Sixth USENIX Security Symposium, San Jose, CA*, vol. 14, 1996.
[18] J. Lee, J. Heo, Y. Cho, J. Hong, and S. Y. Shin, "Secure deletion for nand flash file system," in *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1710–1714, ACM, 2008.
[19] B. Lee, K. Son, D. Won, and S. Kim, "Secure data deletion for usb flash memory," *J. Inf. Sci. Eng.*, vol. 27, no. 3, pp. 933–952, 2011.
[20] J. Reardon, S. Capkun, and D. A. Basin, "Data node encrypted file system: Efficient secure deletion for flash memory," in *USENIX Security Symposium*, pp. 333–348, 2012.
[21] A. Mahajan, M. Dahiya, and H. Sanghvi, "Forensic analysis of instant messenger applications on android devices," *arXiv preprint arXiv:1304.4915*, 2013.
[22] T. Vidas, C. Zhang, and N. Christin, "Toward a general collection methodology for android devices," *Digit. Investig.*, vol. 8, pp. S14–S24, Aug. 2011.

X. Appendices

A. List of devices

The list of devices for which we show results in Section III:

Froyo (v2.2.x): HTC Nexus One, Motorola Defy.

Gingerbread (GB, v2.3.x): Galaxy S Plus, HTC Wildfire S, HTC Desire S, Galaxy S, Galaxy S2, Galaxy ACE, LG Optimus L3, Nexus S.

Ice Cream Sandwich (ICS, v4.0.x): HTC Sensation, Galaxy S3, HTC Desire C, Galaxy S2, LG Optimus L5.

Jelly Bean (JB, v4.[1-3]): Nexus 4 (x2), Motorola Razr I, LG Optimus L7, Nexus S, Galaxy Note 1, HTC One S, HTC One X.

Other devices in our sample for which we do not present results (too few devices) include the Motorola Defy (Eclair, v2.1.x), Nexus 4 and Nexus 5 (KitKat, v4.4).

B. Pattern writing and validation from Section III-A

The following steps used to test each partition of interest are detailed below:

Root Access: To be able to write to a partition bit-by-bit, we first needed low-level access to the flash storage. In the case of yaffs2, this means access to the raw flash, while for an eMMC it means access to the logical blocks. Android does not give such low-level access to apps. Rather, one needs to "root" the device. We achieved this with known root exploits or by booting a custom Recovery – in the latter we would backup the stock Recovery first. Previous work [22] suggested loading custom code via the Bootloader without requiring root access within the Android OS. However this only works on a handful of devices today.

Writing Patterns: We wrote identifying patterns on the entire partition. Each pattern was $1/4^{th}$ the device block size. For external storage, we wrote the patterns with the Android OS. For the data partition, devices would sometimes crash and reboot: in this case we resorted to using a custom Recovery booted in the previous step. All patterns are delimited with common 10-byte HEADERs and FOOTERs (Fig. 9), and uniquely identified by a 4-byte counter (ID). We filled each pattern with random bytes (RANDOM) to avoid the underlying chip using compression, and added a 16-byte md5 DIGEST over the RANDOM bytes for verification. Pattern generation was done on a laptop, and sent to the phone via USB with the Android Debug Bridge (adb) utility shipped with the Android SDK (Fig. 8).

```
# laptop: forward port 12345 to device
$ adb forward tcp:12345 tcp:12345

# device
$ /dev/busybox nc -l -p 12345 | \
 /dev/busybox dd of=/dev/block/mmcblk0p2

# laptop: pipe pattern to local port
$ ./echo_pattern | nc localhost 12345
```

Fig. 8. Pattern writing for Galaxy S's data partition (i9000). Reading a partition is achieved with similar commands.

Fig. 9. Pattern written to a partition of interest.

Sanitisation: We performed a wipe with one of the recommended options given by vendors[16], i.e. (*i*) the Factory Reset in Settings, and (*ii*) in the Recovery/Bootloader mode. Because of the lack of formatting and file system introduced by our patterns, some devices would fail to perform the wipe on external storage. In this case, we first re-formatted them with the built-in Settings option. When we had installed a custom Recovery, we would also take care of re-installing the original one prior to the wipe (with the backup).

[16]Alcatel, BlackBerry, Apple, HTC, Samsung, Huawei, Nokia, Motorola, Lenovo, Sony, LG

Imaging: We imaged the entire partition with similar commands as presented in Fig. 8.

Validation: The last step is the recovery and validation of patterns. We searched for non-deleted pattern candidates using their known header and footer, then validated each candidate by verifying the digest over the random data. We also investigated which percentage of the disk was properly sanitised (i.e. zeros for an eMMC or ones for a raw flash), in order to confirm that it was consistent with the number of patterns recovered.