FROST

Forensic Recovery of Scrambled Telephones

Tilo Müller and Michael Spreitzenbarth

Department of Computer Science
Friedrich-Alexander University of Erlangen-Nuremberg
{tilo.mueller,michael.spreitzenbarth}@cs.fau.de

Abstract. At the end of 2011, Google released version 4.0 of its Android operating system for smartphones. For the first time, Android smartphone owners were supplied with a disk encryption feature that transparently encrypts user partitions. On the downside, encrypted smartphones are a nightmare for IT forensics and law enforcement, because brute force appears to be the only option to recover encrypted data by technical means. However, RAM contents are necessarily left unencrypted and, as we show, they can be acquired from live systems with physical access only. To this end, we present the data recovery tool FROST (*Forensic Recovery of Scrambled Telephones*). Using Galaxy Nexus devices from Samsung as an example, we show that it is possible to perform cold boot attacks against Android smartphones and to retrieve valuable information from RAM. This information includes personal messages, photos, passwords and the encryption key. Since smartphones get switched off only seldom, and since the tools that we provide must not be installed before the attack, our method can be applied in real cases.

1 Introduction

In 2011, 83 percent of the American adults had a cell phone from which 42 percent had a phone that can be classified as a *smartphone* [1]. Android is today the most common smartphone platform, followed by iOS, Blackberry OS, and Windows Phone. Since most consumers use their smartphones for both business and personal applications, missing devices often contain personal and corporate data. For example, the survey *The Lost Smartphone Problem* [2] on 439 U.S. organizations objectively determined that in a 12-month period 142,708 out of 3,297,569 employee smartphones were lost or stolen, i.e., 4.3 percent per year. 5,034 of these smartphones were known to be subject to theft, while the others were "missing". Only 9,298 smartphones were recovered within the time of the study. Results like those make clear that people must take precautions to secure their smartphones against physical loss. The most popular method to protect data against physical loss is encrypting it with AES. Android, for example, enables users to encrypt their user partition with AES since version 4.0, which was released in October 2011. However, encryption technologies are ambivalent as they also enable criminals to hide digital evidence, so that encrypted smartphones have a serious impact on digital forensics.

M. Jacobson et al. (Eds.): ACNS 2013, LNCS 7954, pp. 373–388, 2013.

Contributions. In this paper, we aim at recovering valuable information from encrypted smartphones. Roughly speaking, we analyze the characteristics of the *remanence effect* [3] on smartphones, prove that Android's boot sequence enable us to perform *cold boot attacks* [4], and show that valuable information can be retrieved from RAM. To this end, we present our recovery tool FROST (*Forensic Recovery of Scrambled Telephones*). FROST can be loaded to a smartphone *after* we got physical access to it, and *without* the need to have user privileges before. We carried out our experiments exemplarily for Galaxy Nexus devices. In detail, our contributions are:

1. *Evaluation of the Remanence Effect*: We analyze the characteristics of the remanence effect on smartphones for the first time. According to previous results on PCs, the decay of bits in RAM correlates with both the operating temperature of a device and its time without power. However, contradictory to previous results, we show that the remanence interval on smartphones is shorter. 50% of all bits are decayed after 2-4s, depending on the device temperature.
2. *Cold Boot Attacks*: The bootloader of Galaxy Nexus devices (and many other Android-driven smartphones) can be unlocked with physical access only. Unlocking the bootloader does *not* destroy RAM contents, but it requires us to reboot the smartphone. According to our results about the remanence effect, we can reboot a smartphone quickly while preserving a significant amount of RAM. After rebooting a Galaxy Nexus device, unlocking its bootloader, and booting up our recovery tool, we were still able to recover much sensitive information. Among others, we recovered emails, photos, contacts, calendar entries, WiFi credentials, and even the disk encryption key.
3. *Breaking Disk Encryption*: If a bootloader is already unlocked *before* we gain access to a device, we can break disk encryption. The keys that we recover from RAM then allow us to decrypt the user partition. However, if a bootloader is locked, we need to unlock it first in order to boot FROST, and the unlocking procedure wipes the user partition (but preserves RAM contents). Since bootloaders of Galaxy Nexus devices are locked by default, and since we conjecture that most people do not unlock them, disk encryption can mostly not be broken in real cases. In addition we integrated a brute force option that breaks disk encryption for short PINs.

The fact that user partitions are wiped out when unlocking the bootloader is a serious limitation of our method. Forensic experts from law enforcement might not be allowed to delete a user partition in order to retrieve digital evidence from RAM. Any data on disk would irretrievably be lost. However, this depends on the actual case and the respective legislation of the country. In any event, criminals do not care about this fact and it is therefore important to discuss the attack vector "RAM" irrespectively of its forensic application. With FROST, we are always able to acquire memory dumps from switched-on Galaxy Nexus devices, and we conjecture that our attack can be extended to a wider range of devices with the tools that we provide. A tutorial, a photo series, source codes, and precompiled binaries of our project are available at `www1.cs.fau.de/frost/`.

2 Background Information

We now provide necessary background information about the encryption support in Android 4.0 and subsequent versions (Sect. 2.1). We then give information about the remanence effect, and about cold boot attacks on PCs (Sect. 2.2). Finally, we give details about our device under test, namely the Samsung Galaxy Nexus (Sect. 2.3).

2.1 Disk Encryption Since Android 4.0

With Android 4.0, support for AES-based disk encryption was introduced. While third party apps that extend the functionality of Android smartphones are primarily written in Java, disk encryption resides entirely in system space and is written in C. Android's encryption feature builds upon *dm-crypt*, which has been available in Linux kernels for years. Dm-crypt relies on the *device-mapper* infrastructure and the *Crypto API* of the Linux kernel. It provides a flexible way to encrypt block devices by creating a virtual encryption layer on top of all kinds of abstract block devices, including real devices, logical partitions, loop devices, and swap partitions. Writing to a mapped device gets encrypted and reading from it gets decrypted. Although dm-crypt is suitable for *full disk encryption* (FDE), Android does not encrypt full disks but only user partitions.

Dm-crypt is kept modular and supports different ciphers and modes of operation, including AES, Twofish and Serpent, as well as CBC and XTS. Android 4.0 makes use of the cipher mode `aes-cbc-essiv:sha256` with 128-bit keys [5]. The AES-128 *data encryption key* (DEK) is encrypted with an AES-128 *key encryption key* (KEK), which is in turn derived from the user PIN through the *password-based key derivation function 2* (PBKDF2) [6]. Using two different keys, namely the DEK and the KEK, renders cumbersome reencryption in the case of PIN changes unnecessary. The encrypted DEK as well as the *initialization vector* (IV) for PBKDF2 are random numbers taken from `/dev/urandom`. These values are stored inside a *crypto footer* of the disk. The crypto footer can either be an own partition or it can be placed at the last 16 kilobytes of an encrypted partition. The crypto footer becomes important for our implementation because it holds necessary information to decrypt encrypted partitions.

Unlike iOS, which automatically activates disk encryption when a PIN is set, Android's encryption is disabled by default. Activating it manually takes up to an hour for the initial process and cannot be undone. Furthermore, it can only be activated if PIN-locks or passwords are in use. In Android, PINs consist of 4 to 16 numeric characters, and passwords consists of 4 to 16 alphanumeric characters with at least one letter. New screen locking mechanisms like pattern-locks and face recognition are less secure, and so Google forbids them in combination with disk encryption. Pattern-locks, for example, can be broken by *Smudge Attacks* [7], and face recognition can simply be tricked by showing a photo of the smartphone owner [8].

2.2 Remanence Effect and Cold Boot Attacks

Adversaries with physical access to their target can perform *cold boot attacks* against encrypted PCs. Cold boot attacks have become publicly known in 2008, when Halderman et al. [4] proved that the *remanence effect* can be exploited to recover disk encryption keys from RAM. The remanence effect, however, has already been known since decades and is neither specific to encryption keys nor to memory chips of PCs [3,9]. The remanence effect says that contents of volatile memory fade away gradually over time, rather than disappearing immediately after power is cut. It also says that low temperatures slow down the fading process. Anderson and Kuhn first outlined attacks exploiting the remanence effect of cooled down memory chips [10]. In applied cryptography, the remanence effect is also used as a timing source [11], and as an entropy source [12].

On PCs, secret keys can be traced in RAM after a reboot from malicious USB drives, due to the remanence effect. Above that, cooled down RAM chips can physically be replugged into another PC. The replug variant is more generic than the reboot variant, because it works irrespectively of BIOS and boot sequence settings. With a recovered secret key, adversaries can decrypt the hard disk and eventually access all data. Cold boot attacks are generic and constitute a threat to all disk encryption solutions. However, it has not been reported yet if, and *how*, cold boot attacks are applicable against ARM-based devices such as smartphones and tablets. According to Halderman et al., by cooling down RAM chips the *remanence interval* is extended from 30 seconds up to ten minutes. According to our results, the remanence interval on smartphones is much shorter (see Sect. 3.2). An interesting question at the beginning was if we can obtain a physical RAM dump from smartphones at all? Unlike x86 PCs, Android devices have soldered RAM chips that we cannot unplug, and no bootable USB ports. Hence, we had to find another way to boot system code. The popular trend towards open bootloaders in recent Android devices opened more avenues for attack. Galaxy Nexus devices (and many other Android-driven smartphones) have now bootloaders that can be manipulated with physical access only.

2.3 Samsung Galaxy Nexus

For our purpose, we have chosen the Galaxy Nexus from Samsung because it was the first device with Android 4.0 and consequently, it was the first Android-based smartphone with encryption support. Moreover, it is an official Google phone, meaning that it comes with an official Android version from Google which is not modified by the phone manufacturer. Official Google releases are most amenable for an in-depth security analysis, and flaws can be generalized best to a wider class of devices.

The Galaxy Nexus family comes with an OMAP4 chip from Texas Instruments (4460) which has a Cortex-A9 CPU implementing ARMv7. The partition layout of an encrypted Galaxy Nexus device is given in Fig. 1. Most of the thirteen partitions can be ignored for our purpose, except userdata, metadata and recovery. The userdata partition contains the encrypted filesystem, the metadata partition is the crypto footer that holds necessary information for decryption, and

block device	partition name	description
/dev/block/mmcblk0p1	xloader	bootloader code
/dev/block/mmcblk0p2	sbl	bootloader code
/dev/block/mmcblk0p3	efs	static information like IMEI
/dev/block/mmcblk0p4	param	boot parameters
/dev/block/mmcblk0p5	misc	system settings like carrier ID
/dev/block/mmcblk0p6	dgs	unknown *(zero filled on all devices)*
/dev/block/mmcblk0p7	boot	boot code
/dev/block/mmcblk0p8	**recovery**	recovery image
/dev/block/mmcblk0p9	radio	radio firmware (GSM)
/dev/block/mmcblk0p10	system	Android operating system
/dev/block/mmcblk0p11	cache	cache (e.g., for user apps)
/dev/block/mmcblk0p12	**userdata**	user data (encrypted)
/dev/block/mmcblk0p13	**metadata**	crypto footer

Fig. 1. Partition layout of an encrypted Samsung Galaxy Nexus device

the recovery partition is a partition that holds a *second* bootable Linux. The recovery partition is different from the main Android system (which is stored on the system partition). It can be compared best with a rescue system of ordinary PCs and allows basic operations on the hard disk without booting into full Android. The recovery partition plays a vital role in our cold boot attack, because we make use of it to boot our own system code.

3 Cold Boot Attacks on Galaxy Nexus Smartphones

We now give an evaluation about the remanence effect on Galaxy Nexus devices and probe the effectiveness of cold boot attacks. To this end, we rely on our recovery tool FROST; we describe the technical details of FROST in Sect. 4. We now describe how FROST can be booted (Sect. 3.1). Based on FROST, we then examine how the operating temperature of a Galaxy Nexus device correlates with the decay of bits (Sect. 3.2). Afterwards, we have a look at personal data that we can gain from RAM when the phone is encrypted (Sect. 3.3). Finally, we have a look to the special case when bootloaders are already unlocked before accessing the phone (Sect. 3.4). If so, we can break Android's encryption feature entirely and decrypt all data on the phone.

3.1 Booting the FROST Recovery Image

The question we answer in this section is, how do we reboot a smartphone and run FROST if physical access to it has just been gained? An important point at the beginning is to ensure that the device has sufficient power for a live analysis. Otherwise, it must be charged, because once an encrypted device loses power, all possibilities other than brute force are lost to gain data from it. After charging, the device must be *cooled down* in order to increase the remanence interval (see

Sect. 3.2). As a rule of thumb, we experienced good results when putting the device into a $-15°C$ freezer for 60 minutes. Before that, it should be packed up in a freezer bag in order to protect it against water condensation.

After the phone has been charged and cooled down, we can reboot it. Since the Galaxy Nexus device has no reset button (like most other smartphones), we have to reboot it by unplugging the battery briefly. Shutting the device down from the lock screen is too slow and valuable information in RAM would get lost. In order to boot up the device quickly after reinserting the battery, the power button must already been held *before* removing the battery. The entire process has to happen so quickly that the phone is without power only for a few hundred milliseconds. Once a smartphone is up again, the risk of losing RAM contents is defeated, because neither unlocking the bootloader nor booting into FROST destroys any important memory lines according to our tests.

Additionally, the buttons *volume up* and *volume down* must be held during boot to enter the *fastboot* mode. Once the phone is in fastboot mode, it can be connected to a PC via USB. First, we assume the bootloader is locked. If so, we have to run `fastboot oem unlock` first. This command requires us to confirm the following warning on the phone: "To prevent unauthorized access to your personal data, unlocking the bootloader will also delete all personal data from your phone". Once we confirm this warning, the encrypted user partition gets wiped. However, "all personal data" is not deleted from the phone – RAM contents are preserved.

Next, FROST must be booted. This can be done in two different ways. Either we run `fastboot flash recovery frost.img` to install it persistently on the recovery partition, or we run `fastboot boot <kernel> [ramdisk]` to start FROST temporarily. The latter is interesting for forensic investigations in the case that the bootloader was already unlocked, because it then prevents the forensic examiner from modifying the state of the phone (which might be illegal depending on the case and/or country). However, if the bootloader must get unlocked first, the state of the phone must be modified anyway. In that case, either the first or the second command can be used interchangeable. Again, criminals most likely do not care about changing the phone state, and thus it is important to discuss both attack vectors, irrespectively of their forensic applicability.

After installing FROST to the recovery partition of a phone, the *recovery mode* option must be selected from the phone's boot menu in order to launch FROST. With the help of FROST, personal data and even encryption keys can now be recovered (see Sect. 3.3 and Sect. 3.4). We strongly recommend to practice the entire procedure several times before carrying it out in real cases. The time of battery removal is critical and the entire procedure must happen quickly (see Sect. 3.2).

3.2 The Remanence Effect

We now analyze the remanence effect of RAM on Galaxy Nexus devices. That is, we analyze the number of decayed bits in RAM after power is cut, in dependence of the operating temperature of a phone and the time of battery removal. Earlier

	ε	$0.5-1s$	$1-2s$	$3-4s$	$5-6s$
$5-10$ °C	0 (0%)	2 (0%)	1911 (5%)	8327 (25%)	24181 (73%)
$10-15$ °C	0 (0%)	976 (2%)	2792 (8%)	18083 (55%)	25041 (76%)
$15-20$ °C	0 (0%)	497 (1%)	4575 (13%)	20095 (61%)	25433 (77%)
$20-25$ °C	0 (0%)	421 (1%)	16461 (50%)	23983 (73%)	27845 (84%)
$25-30$ °C	1 (0%)	2204 (6%)	16177 (49%)	27454 (83%)	28661 (87%)

Fig. 2. Number of bit flipping errors per physical page (in total and percentage) in dependence of the phone temperature and the time of battery removal

in our investigations, we recognized that the chance to recover personal data with FROST increases considerably if the phone is cold. We then experimented with putting the phone into a fridge and into a freezer, and we got even better success rates. In the following we give exact benchmarks for this effect.

Fig. 2 lists the bit error rate of memory pages as a function of the device temperature and the time without power before reboot. To determine the device temperature we utilized an infrared thermometer and pointed it to the exactly same position on the phone's motherboard each test run. To cool down the phone, we put it into a -15 °C freezer. 25–30 °C is the normal operating temperature of a Galaxy Nexus, 20–25 °C is reached after 10 minutes, 15–20 °C after 20 minutes, 10–15 °C after 40 minutes, and 5–10 °C after 60 minutes inside the freezer. In several test cases, we never observed damage to the phone when putting it into the freezer for 60 minutes or less (longer periods have not been tested).

To determine the bit error rate, we used FROST to fill memory pages at fixed physical addresses entirely with 0xff. The page size in Android is $4,096$ and so we filled each page with $4,096 \cdot 8 = 32,768$ bits. After booting into FROST, as described in Sect. 3.1, we reconsidered the pages that we recently filled and counted the bits that were now zero. By this means, we got the total number of decayed bits and we were able to estimate the overall bit error rate, as listed in Fig. 2. Note that the highest possible bit error rate is 87.5%, and not 100%, because the passive state of 50% of RAM lines is 0xc0, and not 0x00. We reproduced our test for different physical addresses, and all pages exhibited the same behavior.

The most inaccurate measures in our test set-up are the times that a device is without power. According to Sect. 3.1, for rebooting a Galaxy Nexus quickly the battery must be removed manually. Milliseconds are crucial for the number of decayed bits, but the mechanic task of battery removal cannot be handled exactly. Therefore, with ε we define the quickest unplugging/replugging procedure that we "were able to perform"; we claim this was consistently below 500 ms. Moreover, we define four intervals up to six seconds, and say that we replugged the battery "somewhen" within these intervals. We explain inconsistencies of our results given in Fig. 2 and 3 mostly with inaccurate timings.

In Fig. 3, we visualized the data set from Fig. 2. It becomes clear that the bit error rate of RAM increases with both the temperature and the time without power. For example, at a temperature of approximately 25 $°C$ we have a bit error rate of 50% after two seconds, whereas the corresponding bit error rate at

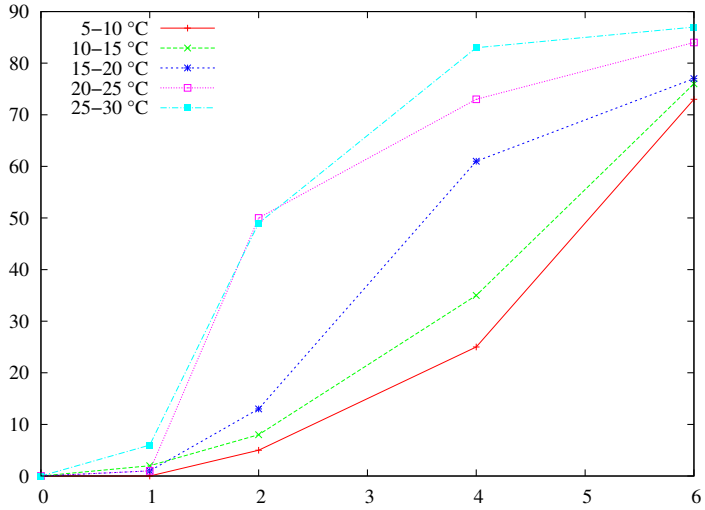

Fig. 3. Bit error ratio (y-axis) in dependence of time (x-axis) and temperatures. The bit error rate decreases with both lower temperatures and shorter times without power.

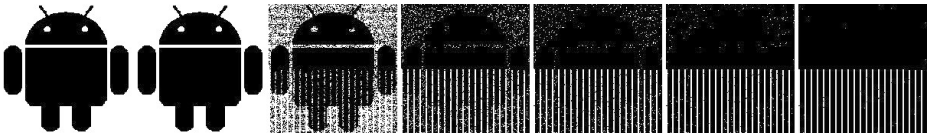

Fig. 4. A Droid-bitmap in RAM of a Galaxy Nexus device after 0, ε, 0.5s, 1s, 2s, 4s, and 6s without power. The cold boot attacks have been deployed at room temperature.

temperatures around 10 °C is only 5%. Hence, besides replugging the battery quickly, putting a device into a freezer increases the chance to recover personal data from RAM notably.

In Fig. 4, we visualized the remanence effect on Galaxy Nexus devices by visualizing decayed bits as a series of Droid bitmaps. For this series, we used 4096-byte bitmaps that exactly fit into one physical page. We used bitmaps rather than JPEGs to visualize bit errors, because using JPEGs entire blocks get destroyed rather than single pixels. We then increased the interval that the phone was without power during boot successively from ε to 6 seconds. Whenever the bitmap header got destroyed, we fixed it manually in order to display the image. Fig. 4 graphically shows the remanence effect and the distribution of bit errors. It also shows that the passive state of the first half of a physical RAM page is 0x00, while the passive state of the second half is 0xc0.

In contrast to Halderman et al., who considered the remanence effect on PCs, we cannot cool down RAM chips below 0°C without risking serious damage to the

phone's hardware. Particularly the display and the battery are likely to suffer damage from temperatures below 0°C. Nevertheless, for temperatures above 0°C our experiments reveal shorter remanence intervals than those identified by Halderman et al. [4]. But, as we see in the next section, the shorter remanence intervals still enable us to perform cold boot attacks against smartphones.

3.3 Recovery of Personal Data

We now investigate which data we can forensically recover from encrypted smartphones through cold boot attacks. Specifically, we are after personal data and digital evidence such as address book contacts, documents, messages, photos, and calendar entries. For our main case, we set up a Galaxy Nexus as personal phone and used it for everyday communications over a week. We then took a photo and did a phone call immediately before the attack. Our goal was to recover as much personal data from the entire week as possible, and the time before the attack in particular. To this end, we attacked the phone by means of FROST and took a memory dump. The memory dump of our test case was near optimal, i.e., we cooled down the phone below 10°C and replugged the battery so quickly that we had a bit error ratio of about 0% according to Fig. 2. Hence, we conjecture we recovered nearly everything that was available in RAM.

We then examined the memory dump with known system utilities like *strings* and *hexdump*, and made use of data recovery programs like *PhotoRec*. Besides photos, PhotoRec can recover websites, text files, databases, sound files, source codes, and binary programs from raw memory images. From the memory dump, we were able to recover 68 JPEG and 199 PNG pictures, 36 OGG tracks, 295 HTML and 386 XML files, 215 SQlite databases, 28 ZIP and 105 JAR archives, 1214 ELF binaries, 485 JAVA source codes, and 6,331 text files.

We then analyzed all recovered data sets thoroughly. While most PNG images that we recovered were system images and logos (and hence, of no interest for us) many JPEG files were *personal photos*. We were able to recover both the picture that was recently taken and older pictures. We were surprised when we even recovered pictures that were taken with another smartphone weeks before the attack. The reason was that these pictures got synchronized in the background via Dropbox (a common filehoster). For the photo we took immediately before the attack, we could recover two variants, a small thumbnail and a high-resolution variant. For the other photos, we could only retrieve the small thumbnail.

As stated above, most PNG images that we recovered were system files, but also the Wikipedia and Wikimedia logos were available. Indeed, we surfed to *wikipedia.org* in the week before the attack, and it was one of the webpages we accessed last, but we did not access it immediately before the attack. Even though, we could also trace its HTML source in RAM. Moreover, we found residues of other webpages in RAM, too. Besides that, we found personal text files and recent emails in RAM. And we found the *entire chat-history* of WhatsApp (a popular messenger). We also explicitly searched for names of our contact list, and we found each name to be present in RAM several times. Near the memory locations where we identified a name, we found respective phone numbers,

Personal information	fully recovered	partly recovered	not recovered
Address book contacts	✓		
Calendar entries		✓	
Emails and messaging		✓	
GPS coordinates			✓
High resolution pictures		✓	
Recent phone calls			✓
Thumbnail pictures	✓		
Web browsing history		✓	
WhatsApp history	✓		
WiFi credentials	✓		

Fig. 5. Set of personal information that we exemplarily searched for. Most of the data we search for could at least partly be recovered.

email addresses, and other contact details. We also found the remaining entries of our contact list that we did not explicitly search for, indicating that the entire address book is in RAM. Additionally, we recovered dates like birthdays from Jorte Calendar, indicating that also the calendar is in RAM. Interestingly, we even found plaintext passwords. Actually, we did search for the SSID of our department WiFi and we could easily locate the according username and password in plaintext. We did not enter the password right before the attack but days before; the password is probably loaded into RAM each time before connecting to the WiFi.

Overall, we recovered dozens of personal information from RAM with known recovery tools and common system utilities. However, we could not locate all information that we were looking for. We tried to find the call history, i.e., we wanted to find out which number has been dialed last, but we were not successful. Likely, this information is in RAM but we failed to identify the respective memory structure. We also failed to recover GPS coordinates when we wanted to construct a movement profile. However, we are confident that more information can be retrieved from RAM with more efforts in the future. Fig. 5 summarizes our results.

3.4 Recovery of the Disk Encryption Key

Apart from personal data, we were also able to recover the disk encryption keys (given that no or only a few bits were decayed). However, on devices where the bootloader is locked, the bootloader must get unlocked first (see Sect. 3.1). On current Galaxy Nexus devices, the unlocking process deletes the userdata and cache partition. We verified that Google actually *wipes* the userdata and cache partition, meaning that these partitions get zero-filled. As a consequence, it becomes pointless to retrieve encryption keys from RAM, although this is still possible.

Since the wiping process is induced by the telephone software rather than the PC, it cannot easily be bypassed. And since the bootloader of a new Galaxy Nexus is locked by default, we conjecture that most Galaxy Nexus devices have

locked bootloaders. However, it is generally a device-dependent property whether a bootloader is locked or unlocked by default, and whether partitions get wiped during unlocking or not. The first series of Galaxy Nexus devices, did *not* delete user partitions when unlocking the bootloader [13]. Later versions of the Galaxy Nexus apparently delete userdata partitions but do not wipe them if the phone is not encrypted [14]. Other devices, like the Samsung Galaxy SII, are shipped with unlocked bootloaders even by default [15], such that unlocking is never necessary.

If we find a bootloader to be unlocked, then FROST can even be applied to break disk encryption, i.e., to decrypt the entire user partition. In that case, it is pointless to discuss which personal data apart from the key can be recovered from RAM (see Sect. 3.3), because we have access to the entire disk. We built necessary key recovery and decryption tools into FROST. In short, we go over all physical memory pages in order to trace AES key schedules. For details, see Sect. 4.

To conclude, for all Android-based smartphones with an unlocked bootloader, or those that can get unlocked without wiping the user partition, we can perform cold boot attacks on the disk encryption key. For all other devices, we can "only" perform cold boot attacks to retrieve valuable information from RAM.

4 Implementation of the FROST Recovery Image

We now present details on the implementation of the FROST recovery image. Technically, FROST is a set of recovery tools that we developed and compounded together into an easy-to-use GUI. Notably, FROST displays a GUI that allows forensic examiners to acquire full memory dumps, to recover encryption keys directly on the phone, to unlock the encrypted user partition with recently recovered keys, and to crack weak PINs with brute force. We come back to these points in the subsequent sections.

4.1 Linux Kernel Module and GUI

The centerpiece of FROST are its loadable Linux kernel modules (LKMs). Accessing physical memory requires system level privileges, and to gain system level privileges we load LKMs. As a basis for our recovery image, we chose the recovery image from *ClockworkMod*, which is a known provider for custom Android ROMs. We integrated our FROST LKM, as well as user mode utilities and third party tools, into the ClockwordMod recovery image and modified it's GUI such that forensic data recovery can be operated comfortably. Users can choose between one of the following options in the FROST GUI:

- *Telephone encryption state*: To check the encryption state of the phone, we try to mount the userdata partition and check whether that succeeds.
- *Key recovery*: This option searches for AES keys (see Sect. 4.2). On success, the recovered key is displayed to the user and saved internally for later use.

- *RAM dump via USB*: This option saves a full memory dump of the smartphone to the PC for offline analysis.
- *Crack 4-digit PINs*: Performs brute force attacks against weak PINs (see Sect. 4.3). Recovered PINs and keys are displayed and the key is saved for later use.
- *Decrypt and mount data*: Decrypts the user partition with recently recovered keys.

The key recovery mode optimized for Galaxy Nexus devices finishes in about $9s$. To create a full memory dump of 700 MB (which is the RAM size of a Galaxy Nexus) takes $3m$ $9s$. To load memory dumps to the PC, we make use of the LiME module [16]. LiME parses a kernel structure to learn physical memory addresses and each physical page is then transferred over TCP to the computer. Alternatively, LiME allows to save physical memory dumps to user partitions, but this is not an option in FROST because we assume user partitions are encrypted. To decrypt the userdata partition, we integrated a statically linked ARM binary of the *dmsetup* utility [17]. This option becomes available only if one of the key recovery methods or the brute force approach were successful, i.e., if the decryption key is known.

A precompiled version of the FROST recovery image for Galaxy Nexus devices is on our website (`http://www1.cs.fau.de/frost`). Above that, we provide the code of the FROST LKM and our user mode utilities as open source, such that similar images can easily be built for a wider class of devices. You may use these components independently of the recovery image.

4.2 AES Key Recovery

Our key recovery algorithm in FROST is based on the known utility *aeskeyfind* [4]. Aeskeyfind searches for AES keys in a given memory image from x86 PCs by identifying AES key schedule patterns in RAM. Contrary to aeskeyfind, FROST is implemented for ARM and searches for AES keys *on-the-fly*, i.e., directly on the phone. In comparison to x86, the endianness of key bytes in ARM is reversed, for example, such that exisiting algorithms had to be adapted. Our optimized code recovers AES keys in less than 10 seconds directly on the phone (whereas aeskeyfind requires always about 10 minutes). An exemplary FROST output is given in Fig. 6.

Our key recovery LKM basically supports two search modes: *quick search* and *full search*. Quick search is highly optimized for Galaxy Nexus devices and looks for AES keys at certain RAM addresses. In detail, we have chosen the address space 0xc5000000 to 0xd0000000 because all our tests revealed that AES key schedules are placed in this range. In quick search mode, the recovery process finishes within seconds. This mode, however, might fail on other devices because the search space might be too specific. Therefore, we implemented the full search mode that considers the entire physical RAM. The full search mode uses a sliding window mechanism that looks at each physical RAM page twice. In quick search mode, AES key schedules which are spread over multiple pages, are missed. In

```
adb> insmod frost.ko fullsearch=0 ; dmesg

key-32: 4ee35476397b76905828a89f3d9b872f
      : ccb6671af6eebffe94ea1bc87c0948e4
key-32: 4ee35476397b76905828a89f3d9b872f
      : ccb6671af6eebffe94ea1bc87c0948e4
key-16: bcdbc55cf809cb5989e58a40ecbb7164
key-16: bcdbc55cf809cb5989e58a40ecbb7164

Summarizing 4 keys found.
```

Fig. 6. Keys recovered with the FROST LKM

```
adb> ./crackpin

  magic: D0B5B1C4
 encdek: 3c4ac402c6095ed46cf4f1e2281a1f3e
   salt: 19043211840adfde95110c7f99263d6c

>>  KEK: 2165534cc66099714a8226753d70b576
>>   IV: 05cb47cf3a98d77e563bb4cfcde791aa
>>  DEK: bcdbc55cf809cb5989e58a40ecbb7164
>>  PIN: [2323]
```

Fig. 7. Key and PIN recovered with brute fore

practice, however, this is unlikely; in 50 test cases, we never observed an AES key schedule that was spread over two pages (the page size in Android is 4096 bytes).

If neither the quick search mode nor the full search mode succeeds, the memory image is too noisy, meaning that too many bits decayed during cold booting (see Sect 3.2). As stated above, our key recovery algorithm is based on the utility aeskindfind. Aeskeyfind discards key candidates as soon as a given threshold of bits is reached that are not in line with a typical key schedule structure. If this threshold is too high, pseudo keys are identified from irrelevant memory regions. But if the threshold is too low, the recovery algorithm becomes prone to decayed bits from cold booting. As a tradeoff value that is based on results of our experiments, we have chosen 64 as default threshold in FROST. That means, 64 out of 1280 bits can be disturbed per key schedule at maximum. In other words, the bit error ratio is not allowed to exceed 5% to be able to identify key schedules in FROST.

To overcome the situation that no key bits can be recovered due to noisy images, we implemented a third search mode. During our RAM analysis, we recognized that AES keys are typically present in memory *five times*: once in the context of an AES forward schedule, once in the context of an AES backward schedule, and three times as a stand-alone bit sequence. Stand-alone bit sequences are commonly hard to identify as keys, because keys themselves have no structure. Only their corresponding key schedules have a structure. To exploit these occurrences, we implemented a search mode that is less generic but offers good results in practice: Rather than searching for key schedule patterns, we look after "magic strings" that appear near the desired key. From several test runs we know fixed offsets from magic strings to key locations. Given these offsets are correct, we can recover key bits independently of key schedules. However, this procedure is optimized for Galaxy Nexus devices and specific Android versions; offsets may change in upcoming releases.

The key recovery code of FROST was developed and tested on Galaxy Nexus devices, but it works for other Android-based smartphones, too. It is a part of our project which is *platform-independent*, meaning that it even runs on non-

Android systems. For example, we have successfully tested parts of our module on a PandaBoard with Ubuntu. In general, FROST's key recovery code can be used on all ARM devices where you have a Linux shell with root access.

4.3 PIN Cracking through Brute Force

PINs are still the most frequent screen lock in use today. But long PINs are too inconvenient for most people that work on their phones on a daily basis, because they must be entered for each interaction with the device, e.g., for giving a call, for writing a message, and for taking a photo. Consequently, people commonly use short PINs of only 4-8 digits. That is a concern, because in Android the screen lock PIN necessarily equals the PIN that is used to derive the disk encryption key. Consequently, besides cold boot attacks, short PINs are another weak point of Android's encryption feature. (Note that we find the restriction that encryption passwords must equal screen lock PINs in Android unnecessary and dangerous. In practice, it is more time consuming to crack a visual PIN prompt than to perform automated brute force attacks against encrypted filesystems.)

In 2012, Cannon and Bradford [18] presented details about Android's encryption system and gave instructions on how to break it with brute force attacks against the PIN. They published their findings in form of a Python script that breaks Android encryption offline, meaning that it runs on an x86 PC after the userdata and metadata partition have been retrieved "somehow". Basically, we reimplemented their Python script in C and cross-compiled it for the ARM architecture, so that we can perform efficient attacks directly on the phone, without the need to download the user partition. To this end, we cross-compiled the *PolarSSL* library for Android, an open source library similar to OpenSSL which is more light-weight and easier to use and integrate. We then statically linked our PIN cracking program with the PolarSSL library, because Android does not support dynamic linking. Both the source code and the statically linked binary are available on our webpage; an exemplary output of it is given in Fig. 7.

PINs with four digits are cracked within $2m\ 58s$ at maximum, i.e., in one and a half minute on average. Although we only implemented the important 4-digit case yet, we can estimate that 5-digit PINs are cracked in about $15m$, 6-digit PINs in $2h\ 30m$, and 7-digit PINs in about $25h$.

5 Future Work and Conclusions

To defeat physical access attacks, disk encryption has become an essential security mechanism for mobile devices. Virtually all PC operating systems support disk encryption since years, and smartphones now provide encryption, too. However, when losing an Android-based smartphone, chances are to lose valuable information even though encryption was used. The remanence effect shows up on smartphones, and as we have proven, it can be exploited with cold boot attacks to retrieve personal data. We believe that our study about Android's encryption is important for two reasons: First, it reveals a significant security gap

that users should be aware of. Since smartphones are switched off only seldom, the severity of this gap is more concerning than on PCs. Second, we provide the recovery utility FROST which allows law enforcement to recover data from encrypted smartphones comfortably.

We have several plans for future improvements of FROST. First, we want to make our recovery image available for more Android devices than just the Galaxy Nexus. We already provide device independent system utilities like the FROST LKM and the PIN cracking program on our webpage, so that forensic examiners can compose recovery images for other devices also on their own. We provide appropriate howtos and source codes of our project for that. From an academic point of view, it is more important to analyze Android's memory structures in-depth in future. For example, we were not able to recover GPS coordinates and the list of recent phone calls yet, but we believe that this information is present in RAM.

To conclude, we have proven that smartphones can be attacked by cold boot attacks. To this end, we have shown that on Galaxy Nexus devices low temperatures raise the success rate of cold boot attacks (remanence effect). We also presented FROST, a tool that recovers personal data from encrypted smartphones. The biggest limitation of FROST to date, however, is that it requires an unlocked bootloader for breaking encryption entirely. Recovering the disk encryption key is always possible, but searching for it becomes pointless when the bootloader was locked (because the user partition gets wiped during unlocking). Nevertheless, personal data can *always* be recovered from RAM.

Countermeasures against cold boot attacks are difficult. On x86 PCs, solutions like TRESOR and TreVisor [19,20] perform encryption on CPU registers only, thereby thwarting attempts to reveal sensitive key material from RAM. However, such solutions are limited to encryption keys and cannot protect RAM contents in general. Protecting all information in RAM is assumed to be infeasible, which in turn proves the severity of tools like FROST.

Acknowledgments. We would like to thank Felix Freiling, Johannes Götzfried, Andreas Kurtz, Sven Schmitt, Johannes Stüttgen, and Stefan Vömel for helpful comments on the topic of our study.

References

1. Smith, A.: 35% of American adults own a smartphone. Pew Internet and American Life Project. Pew Research Center (July 2011)
2. Ponemon Institute LLC. The Lost Smartphone Problem: Benchmark study of U.S. organizations. In: Ponemon Institute Research Report. sponsored by McAfee (October 2011)
3. Gutmann, P.: Data Remanence in Semiconductor Devices. In: Proceedings of the 10th USENIX Security Symposium, Washington, D.C. USENIX Association (August 2001)

4. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest We Remember: Cold Boot Attacks on Encryptions Keys. In: Proceedings of the 17th USENIX Security Symposium, San Jose, CA, August 2008, pp. 45–60. Princeton University, USENIX Association (August 2008)
5. Android Open Source Project (AOSP). Notes on the implementation of encryption in Android 3.0, source.android.com/tech/encryption/
6. Turan, M., Barker, E., Burr, W., Chen, L.: Special Publication 800-132: Recommendation for Password-Based Key Derivation. Technical report, NIST, Computer Security Division, Information Technology Laboratory (December 2010)
7. Aviv, A.J., Gibson, K., Mossop, E., Blaze, M., Smith, J.M.: Smudge Attacks on Smartphone Touch Screens. In: WOOT 2010, 4th USENIX Workshop on Offensive Technologies. Department of Computer and Information Science, University of Pennsylvania (August 2010)
8. Kumar, M.: Android facial recognition based unlocking can be fooled with photo. The Hacker News (November 2011), http://thehackernews.com/
9. Skorobogatov, S.: Data Remanence in Flash Memory Devices. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 339–353. Springer, Heidelberg (2005)
10. Anderson, R., Kuhn, M.: Tamper Resistance – a Cautionary Note. In: The Second USENIX Workshop on Electronic Commerce Proceedings, Oakland, California, pp. 1–11. USENIX Association (November 1996)
11. Rahmati, A., Salajegheh, M., Holcomb, D., Sorber, J., Burleson, W., Fu, K.: TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices withou Clocks. In: 21st USENIX Security Symposium, Bellevue, WA, UMass Amherst, USENIX Association (August 2012)
12. Saxena, N., Voris, J.: We Can Remember It for You Wholesale: Implications of Data Remanence on the Use of RAM for True Random Number Generation on RFID Tags. In: 5th Workshop on RFID Security (RFIDSec), Leuven, Belgium, Polytechnic Institute of New York University (July 2009)
13. xdadevelopers. Google Play Nexus not wiping after Bootloader Unlock. Thread 1650830 (April 2012), http://forum.xda-developers.com
14. xdadevelopers. Internal Memory Data Recovery - Yes We Can! Thread 1994705 (November 2012), http://forum.xda-developers.com
15. xdadevelopers. GT-i9100 Galaxy SII FAQ. Thread 1046748 (April 2011), http://forum.xda-developers.com
16. Sylve, J.: LiME - Linux Memory Extractor. In: ShmooCon 2012, Washington, D.C. Digital Forensics Solutions, LLC (January 2012)
17. Zugelder, M.: androidcrypt.py (April 2012), https://github.com/michael42/androidcrypt.py/
18. Cannon, T., Bradford, S.: Into the Droid: Gaining Access to Android User Data. In: DefCon 2012. VIA Forensics (July 2012)
19. Müller, T., Freiling, F., Dewald, A.: TRESOR Runs Encryption Securely Outside RAM. In: 20th USENIX Security Symposium, San Francisco, California. University of Erlangen-Nuremberg, USENIX Association (August 2011)
20. Müller, T., Taubmann, B., Freiling, F.C.: TreVisor: OS-Independent Software-Based Full Disk Encryption Secure Against Main Memory Attacks. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 66–83. Springer, Heidelberg (2012)