Towards Secure Deletion On Smartphones

Michael Spreitzenbarth[1] Thorsten Holz[2]

[1]Laboratory for Dependable Distributed Systems
University of Mannheim, Germany
spreitzenbarth@informatik.uni-mannheim.de

[2]Horst Görtz Institute for IT-Security
Ruhr-University Bochum
thorsten.holz@rub.de

Abstract: Nowadays, smartphones constitute one of the most commonly used electronic devices. Today's smartphones combine a variety of different technologies: they offer in addition to excellent mobile availability and connectivity also high-speed data transfer for the user. Moreover, they are multimedia capable due to their integrated digital camera or music player, and offer a wide variety of communication services like e-mail, SMS or MMS. Consequently, they are used increasingly as a "mobile office".

In this paper, we outline the possibilities and obstacles of *secure deletion*, namely the problem of deleting sensitive data on a smartphone in such a way that this data cannot be restored during a later forensic investigation. In order to guarantee the complete deletion of data, it would be necessary to access the memory chip directly such that we can overwrite the address space of existing data with arbitrary data. However, this approach is not possible when dealing with smartphones due to several reasons. On the one hand, the user's activities are restricted on the device, which implies that far-reaching system interventions cannot be conducted easily. On the other hand, writing on a specific physical address is hindered due to the use of "wear leveling" algorithms on flash chips, which are intended to optimize durability. We discuss these problems in detail and introduce an approach to more securely delete data under certain constraints.

1 Introduction

A *smartphone* is a mobile handset that offers a wide variety of different technologies which reassemble typical functionality of a PC: besides offering excellent mobile availability and connectivity, also high-speed data transfer via universal mobile telecommunications system (UMTS) and wireless local area network (WLAN) are available to a user such that she is always connected to the Internet. Moreover, smartphones are multi-media capable due to an integrated digital camera or music player, and – not to forget – they serve the function of text-based communication via e-mail, multimedia messaging service (MMS), and short message service (SMS). Consequently, they are used increasingly as a "mobile office", thanks to the various office applications available for mobile phones. Another aspect which should not be neglected is the fact that current smartphones are more fre-

quently equipped with GPS modules, that enable the user to make use of the mobile phone for navigation purposes.

One of the drawbacks of this development is the fact that smartphones also store a lot of sensitive data like for example contact information (e.g., phone numbers or full address details), e-mail messages that might contain sensitive information, or private photos. Unfortunately, there is typically no easy way to delete this information from a given phone. Due to the fact that the second market (e.g. ebay) for smartphones is increasing the deletion of these data becomes more and more important.

There are two main obstacles that complicate *secure deletion*, i.e., the problem of deleting sensitive data on a smartphone in such a way that this data cannot be restored during a later forensic investigation: limited interactions with the device and *wear leveling*.

On the one hand, smartphones offer only a limited user-interface and the user's activities are often restricted to interaction with (pre-)installed tools on the device. This often prohibits system interventions which require privileged access to the device. On the other hand, there is a subtle physical peculiarity of smartphones that needs to be taken into account: mobile devices typically use flash chips to store data and these components use a technique called *wear leveling* to evenly distribute write access across the flash chip. When implementing secure deletion, we thus need to take this physical requirement into account. In this paper, we outline the possibilities and obstacles of secure deletion and introduce an approach to more securely delete data under certain constraints.

This paper is outlined as follows: In Section 2, we discuss related work in the area of secure deletion. We provide a brief background on wear leveling and the implications of this approach in Section 3. In Section 4, we introduce an approach to securely delete data under certain constraints, which we evaluate in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

We are not the first to study the problem of secure deletion and thus we provide an overview of related work on this topic. One of the first papers in the area of secure deletion was published by Gutmann, who analyzed how data from magnetic and solid-state memory can be deleted in a secure way [Gut96]. He introduced the 35-pass overwrite technique to address the implications of different encoding schemes used by hard drives. Furthermore, he noted that "A good scrubbing with random data will do about as well as can be expected." [Gut96] which is true for magnetic and solid-state memory, but some obstacles need to be addressed when dealing with flash chips as we explain in the next section.

Bauer and Priyantha proposed a technique to perform secure deletion at the filesystem level: they implemented an extension to the *ext2* filesystem that asynchronously deletes file data and meta-data by overwriting this data according to best known practices [BP01]. Joukov et al. extend this approach to also handle *ext3* filesystems and discuss in detail the problem of secure deletion [JPZ06]. We implemented a similar approach that can be used

on mobile phones. Leet et al. discuss a method that can handle NAND flash file systems and the authors provide a proof-of-concept implementation for YAFFS [LHC+08]. The main idea behind their approach is to encrypt files and forces all keys of a specific file to be stored in the same block, thus only a single erase operation is need to delete the key – as a result, the file can not be accessed anymore. Our approach is more light-weight and we overwrite the data according to best know practices.

Specific aspects of the secure deletion problem like secure deletion for a versioning filesystem [PBH+05] or a secure peer-to-peer backup system [BBST01] have been discussed in the literature. Typically, all these approaches implement different techniques to overwrite data and relevant meta-data with random data to impede forensic analysis. We perform a similar approach when dealing with flash chips.

3 Background: Wear leveling

In this section, we explain the different wear leveling techniques and the implied consequences for our approach. First, we outline general approaches of wear leveling followed by a technique which is used in more recent Nokia smartphones.

Flash chips have the drawback that their content can only be changed a limited number of times: at some point, after 100,000 or more cycles depending on the actual chip, so much voltage or time is required to either program or erase the cell (or both) that it becomes impractical to use it any further. The lifetime of that cell has ended at this point. To address this problem, the technique of wear leveling for a memory controller has been introduced: it provides a method to distribute the access the cells at times when it is detected that they are receiving significantly uneven use. The exact procedure of those operations is described in the following.

Figure 1 schematically depicts the memory operation techniques [LNTG08]. Here, the storage space is separated into single blocks whereby those blocks again consist of several blocks. If the user now sends data which are attached to a logical address, this logical address is converted with the help of an address translation table into a physical memory address, i.e., to a block within the bank.

This address translation table can be reprogrammed by a processing unit in order to change the physical address of the block in which data are intended to be stored. The process of reprogramming is used to lade the whole memory equally often with writing operations. In order to do such reprogramming, information on storage characteristics and storage occupancy are collected. Therefore, the number of writing operations on each storage block is stored. If it is the case that this number exceeds the average value of the other blocks by a certain amount, the wear leveling process is started.

When wear leveling is accomplished, two main events occur. First of all, data of the heavily used blocks are swapped with the data of those blocks which are least used. In a second step, the address translation table is modified such that the new physical addresses are connected to the old logical ones. Consequently, a data block having the same logical address as prior to wear leveling is now written on another physical address than beforehand.

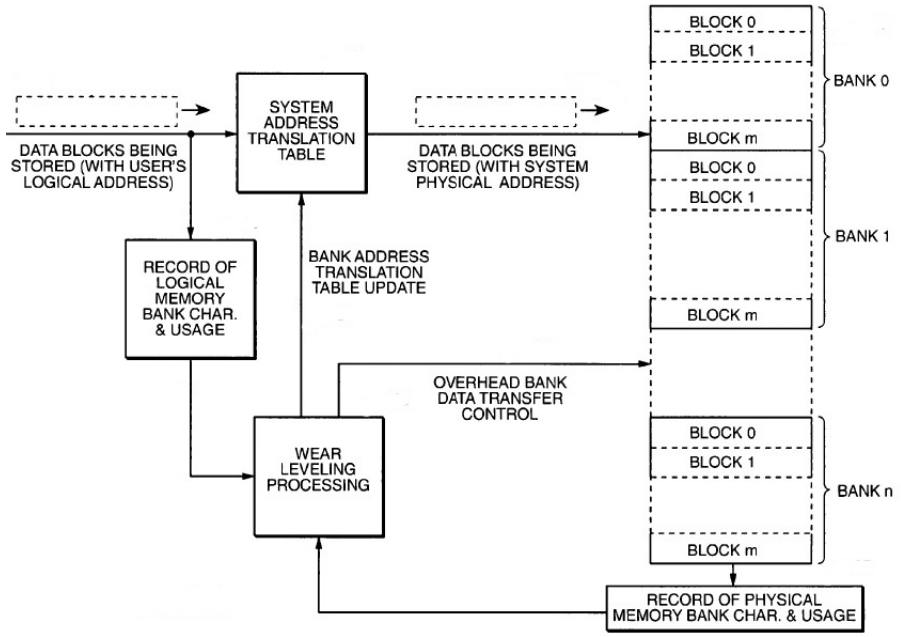

Figure 1: Schematic illustration of ways in which the solid state memory may be operated [LNTG08]

Figure 2 illustrates another approach of wear leveling by means of a process flow diagram [LNTG08]. In this diagram, the process of wear leveling is initialized by the system. This initialization can either be started by a system service or by a memory controller. The rest of the process is mostly identical to the already described one, except for the fact that in this case an additional *spare bank* is used. Data of the least used bank is written to the spare bank and data of the heavy used bank is now written to the least used bank, which in turn is now empty. In another step, the heavy used bank is converted to the new spare bank. The following process is now again identical to the previously described one.

Those two figures illustrate that there are various approaches for the execution of wear leveling. However, all approaches have the intention of even wear of the storage spaces. According to Symbian OS Internals [sym09] and Samsung [Fla07], Nokia uses in its current device series (e.g.: N-Series) Samsung OneNAND [sam09] storage with a modification of the already described wear leveling techniques. In this case, no swapping of data on the blocks is conducted due to time reasons; instead, a delete- and write-operation is saved. This happens by the storage of the erase-count of each block. If we would now like to write to a block having a higher count than one of the non-used blocks, the block which has not been used up to now is deleted and the write operation is done on that block. The real selected block is now marked as non-used and shifted to the so-called *garbage queue*, which is ordered descending according to the erase-count. If at a later point in time a block for writing is needed, its erase-count is compared to the block on first position of the queue, and the block with the lowest erase-count of those two is chosen for the writing

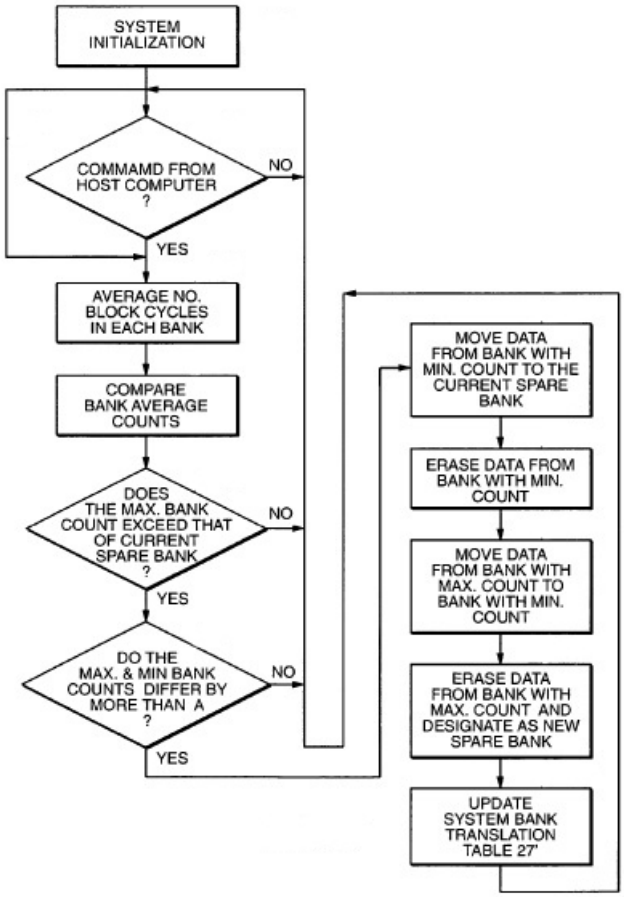

Figure 2: Flow diagram showing a preferred operation of the memory system [LNTG08]

operation. There are two methods of deleting data stored on blocks within the garbage queue: the first approach is to delete data when the block is inserted into the queue, while in the second approach data will be deleted as soon as the block leaves this queue and is then re-used for storing new data. When inquiring Samsung which approach is used in OneNAND chips, the company did unfortunately not answer our request.

4 Towards Secure Deletion on Symbian Phones

In order to develop a tool that helps to securely deleting data on a smartphone, we have resorted to the programming language Python. Concretely, we used Python for S60 [Nok08]

(also known as pyS60) available in Version 1.4.5. The Python for S60 platform simplifies application development and provides a scripting solution for the Symbian C++ APIs.

Within this development platform, we have developed a tool named *SecDel* which currently possesses the ability to delete SMS messages, telephone directories, as well as, calendar entries from a given phone. Moreover, it contains an update-function, which allows the user to load modules, which are developed in the future, via the Internet.

In the following, we discuss the secure deletion of telephone book entries exemplarily and in detail. When deleting calendar entries and SMS messages, we have only focused on specific aspects of these methods, as the remaining procedure is very similar in all cases. The contacts module offers an API to address book services allowing the creation of contact information databases. The contacts module represents a Symbian contact database as a dictionary-like *ContactDB* object, which contains *Contact* objects and which is indexed using the unique IDs of those objects [ST07].

As shown in the upper part of Listing 1, the *ContactDB* is opened and the unique ID and the entry title is read out of all entries with the help of a loop, both parts are then consolidated into one list element.

```
def searchContacts():
    appuifw.note(u"searching Contacts...", "info")
    db = contacts.open()
    list3 = []
    for db_id in db:
        id = str(db[db_id].id)
        title = db[db_id].title
        element = (id + " - " + title)
        list.append(element)
    selection = appuifw.multi_selection_list(list, 'checkbox', 1)
    for k in selection:
        contact_id = list[k].split(" ")[0]
        contact_id = int(contact_id)
        # change value
        kind = ['city', 'company_name', 'country', 'date', '
            dtmf_string', 'email_address', 'fax_number', '
            job_title', 'note', 'pager_number', 'phone_number', '
            po_box', 'postal_address', 'postal-code', 'state', '
            street_address', 'url', 'video_number', 'picture', '
            second_name', 'voip', 'sip_id', 'personal_ringtone', '
            share_view', 'prefix', 'suffix', 'push_to_talk', '
            locationid_indication', 'last_name', 'first_name', '
            mobile_number']
        for type in kind:
          for i in range(0,10):
            try:
                if db[contact_id].find(type)[i].value:
                    db[contact_id].find(type)[i].value = u"
                        XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
            except:
```

```
        continue
    # delete contact
    db.__delitem__(contact_id)
    appuifw.note(u"Contact deleted!", "conf")
```

Listing 1: Secure deletion of contacts used in SecDel

After such a list element has been created from each of the telephone book entries, it is saved in a list. Then a *Multi-Selection-Listbox* is created with the help of this list. This listbox (see Figure 3) enables the search for entries and marking of several entries in order to delete all these entries at once.

Figure 3: SecDel select contacts which should be deleted

In the next step, the script separates the ID of the selected list entries from the title in order to search the whole entries in the database again. If the corresponding entry is found, all existing objects within this entry, no matter how often they occur, are overwritten with a sign chain, consisting of 59 times "X". We have chosen the length of 59 due to the fact that this is the maximum allowed length of characters which is available for the longest of the available objects. We conduct this step in order to make sure that none of the characters of the initially entry is still available afterwards. In Figure 4, we depict an entry of the telephone's address book which has been overwritten in the above described way.

Figure 4: SecDel contact entry after renaming it

After successful overwriting, the objects of all previously chosen entries are deleted from the database. This is communicated to the user in another message.

A particularity exists when we deal with calendar entries or notes: here the stored dates and times as well as the point of time of reminders are either deleted completely or reset to the 01.01.1970, 00:00 o'clock by the code fragment shown in Listing 2. In addition, repeated events are converted to unique events before they are deleted from the database, such that this fact does not reveal any information.

```
db2[calendar_id].set_repeat(None)
if (db2[calendar_id].type == "note"):
        # clear date and time
        db2[calendar_id].set_time(None)
else:
        # set date to 01.01.1970
        db2[calendar_id].set_time(0, 0)
```

Listing 2: Code excerpt of SecDel which changes the date and time values

As a further module, we have included an update-function by which it is possible to load future modules and extensions via an Internet connection belatedly. As a second additional update, we have included a remote service function. It happens sometimes that one loses his smartphone, or that it is stolen. Typically a user can only hope that the important data stored on the phone are not used by the finder or thief. To address this problem, we have developed the following function: if SecDel is kept active in the background, it analyzes every SMS message in the inbox regarding a certain message sequence. As an authentication process, the user can specify a password which has to be in that message, too. If the phone now receives a SMS message with the sequence "run_secdel" in the first line and a password in the second line, the tool will validate the password and start with the immediate deletion of all data as described above. The validation of the password should prevent a denial-of-service attack. This process is done with the help of a callback function which triggers this behavior. The inbox object's *bind()* function binds a callback function to an event that is generated by an incoming message. In our case, the function *msg_rec()* is called up as soon as a new SMS is received on the smartphone. Apple's iPhone implements a similar protection technique that enables a user to remotely wipe the phone, which erases all personal data and restores the phone to the factory settings.

5 Evaluation

The operations we have discussed in the previous sections have been tested on a Nokia E90 phone with Symbian OS version 9.2 installed. For the verification of secure deletion on this generation of smartphones, currently only few technical possibilities exist. The reason lies in the fact that Symbian has introduced security restrictions (also called *Capabilities* [Symb]), which disable any foreign access to all important private files. In order to get to those storage spaces, we need an *All Files* capability by which our software agent becomes certified (compare [Syma]). The *All Files* capability is one of the Manufacturer-approved capabilities, which are highly sensitive capabilities that can only be obtained from the device manufacturer, even if it is just for experimenting on a phone under our

control. Getting these capabilities requires a user to pass a complex certification process, and to have an ACS Publisher ID from Verisign [ST07].

Due to the reasons stated above, we had to resort to a smartphone using an older Symbian version for the purpose of verification. When using older Symbian versions, the stated security restrictions had not been implemented yet. In this case, it is also not possible to create a memory dump with the help of *flasher-tools* like Twister Box. The single solution which is currently realizable, apart from de-soldering the memory element, is constituted by the solution with the help of a software agent. This agent is installed on the memory card of the mobile device and creates a dump of saved files from this location. Installing the agent on the memory card rather than on the smartphone itself provides the advantage that nearly no data will be changed during the installation process. This could be realized on condition of dumping the original inserted memory card on a bigger one and then installing the software agent on the new card. In Figure 5 we show the logical paradigm workflow of these tools.

Figure 5: Workflow of a software agent by means of Panoptes

For this purpose, we used a Nokia N70 phone and the software agents MIAT (of the University of Rome [MO07, DM08]) and Panoptes (which has been developed within a diploma thesis [Spr09]). After comprehensive tests, we received notice that these tools are not able to restore any deleted data. Within the data, which had been saved with the help of MIAT and Panoptes, we could neither find data which had been deleted by the Symbian interface, nor were data detectable which had been deleted by the tool SecDel. To the best of our knowledge, SecDel thus securely deletes data on the tested phone by overwriting the relevant data with garbage.

Without any doubt, the most perfect solution for the verification of the depicted operations and the throughout validation of the implemented wear leveling technique of the memory element, would be the de-soldering of the flash storage chip and the subsequent direct analysis of this element. Unfortunately, we were not given the possibility to do so which is why we resorted to the approaches presented beforehand. Another drawback of de-soldering is the fact that this approach is questionable from a forensic perspective since evidence might be destroyed during the process.

6 Conclusion and Future Work

In this paper, we presented the various influences which are related to secure deletion of data on mobile phones. In the beginning, we explained the different kinds of wear leveling used for flash storage. In general, wear leveling describes the possibility of extending the economic life-time of a flash chip by exchanging blocks on which data are used very frequently with those on which data are used less frequently. With the help of this process, the whole storage element is equally burdened with write operations in order to prevent deficiencies within single sections. In this context, various approaches exist to implement this concept. In most cases, those differ only in the handling of data which are stored on the blocks which should be swapped. When dealing with current Nokia smartphones, these blocks are deleted completely before data is written on these blocks again. As a result, the problem of reading out file fragments in the so-called *slack space* does not exist on these phones. Slack space is the unused space in a disk cluster: some file systems use fixed-size clusters and even if the actual data being stored requires less storage than the cluster size, an entire cluster is reserved for the file. Since always complete blocks are deleted, we can ignore this problem on mobile phones.

Subsequently, we presented a tool which tries to delete personal data on the smartphone in a secure way. This means that data is deleted in such a way that restoring them is not possible. We implemented this technique by overwriting the existing data with known garbage and thus erasing the information. Our evaluation suggests that this overwriting is successful and all relevant information is deleted. Unfortunately, we were not able to thoroughly verify the depicted deleting operations in a forensically correct way due to the fact that we did not possess the needed means to de-solder and examine the flash chip. We could only prove with the help of the tools MIAT [MO07, DM08] and Panoptes [Spr09] that deleted data are not contained in the files intended for storage any longer. Due to this reason, the tool can not guarantee “secure deletion”, but rather “more secure deletion”. The combination of overwriting of the data and the subsequent deletion of them on the operating system level on the one hand, as done by SecDel, and the wear leveling process on the memory level on the other hand, as done by the build in flash modules, significantly increase the difficulty to restore this data.

Our future work will focus on the development and improvement of forensically sound analysis tools for Symbian OS and other operating systems for mobile devices, e.g., Google Android or iPhone OS. At the same time, we will also work on improving the secure deletion tool which we have introduced in this paper, especially on improving the evaluation to verify that the deleted data can indeed not be restored.

References

[BBST01] Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A Secure Peer-to-Peer Backup System. Technical report, MIT, 2001.

[BP01] Steven Bauer and Nissanka B. Priyantha. Secure data deletion for Linux file systems. In *USENIX Security Symposium*, 2001.

[DM08] Alessandro Distefano and Gianluigi Me. An overall assessment of Mobile Internal Acquisition Tool. *Digital Investigation*, 5:121–127, 2008.

[Fla07] Flash Software Group. XSR1.5 WEAR LEVELING. Technical report, Samsung Electronics Co., Ltd, 2007.

[Gut96] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *USENIX Security Symposium*, 1996.

[JPZ06] Nikolai Joukov, Harry Papaxenopoulos, and Erez Zadok. Secure deletion myths, issues, and solutions. In *Second ACM Workshop on Storage Security and Survivability*, 2006.

[LHC+08] Jaeheung Lee, Junyoung Heo, Yookun Cho, Jiman Hong, and Sung Y. Shin. Secure deletion for NAND flash file system. In *ACM Symposium on Applied Computing*, 2008.

[LNTG08] Karl MJ Lofgren, Robert D Norman, Gregory B Thelin, and Anil Gupta. Wear Leveling techniques for flash EEPROM systems. United States Patent, April 2008.

[MO07] Pontjho M Mokhonoana and Martin S Olivier. Acquisition of a Symbian Smart phone's Content with an On-Phone Forensic Tool. Technical report, Information and Computer Security Architectures Research Group, 2007.

[Nok08] Nokia. *PyS60 Library Reference*, 1.4.5 edition, Dezember 2008.

[PBH+05] Zachary N. J. Peterson, Randal Burns, Joe Herring, Adam Stubblefield, and Aviel D. Rubin. Secure deletion for a versioning file system. In *USENIX Conference on File and Storage Technologies (FAST)*, 2005.

[sam09] SAMSUNG OneNAND™, September 2009. `http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_OneNAND.html`.

[Spr09] Michael Spreitzenbarth. Mobile Phone Forensics. Diploma Thesis, University of Mannheim, 2009.

[ST07] Jürgen Scheible and Ville Tuulos. *Mobile Python: Rapid Prototyping of Applications on the Mobile Plattform*, volume 1st. Wiley, October 2007.

[Syma] Symbian Press. Complete Guide to Symbian Signed. `http://developer.symbian.org/wiki/index.php/Complete_Guide_To_Symbian_Signed`.

[Symb] Symbian Press. Plattform Security. `http://developer.symbian.org/wiki/index.php/Platform_Security_(Fundamentals_of_Symbian_C%2B%2B)`.

[sym09] Symbian OS Internals, September 2009. `http://developer.symbian.org/wiki/index.php/Symbian_OS_Internals/`.