h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

- Fachbereich Informatik -

Sicheres Löschen auf mobilen Endgeräten unter Android

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

vorgelegt von

Peter Lyko

Referent: Prof. Dr. Harald Baier

Korreferent: Björn Roos, M.Sc.

Ausgabedatum: 02.04.2013

Abgabedatum: 09.10.2013

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 8. Oktober 2013

Peter Lyko

Zusammenfassung

Diese Arbeit befasst sich mit dem Vorgang des sicheren Löschens auf Android-Smartphones. Ein Betriebssystem entfernt beim Löschen einer Datei meist nur den Verweis auf diese, nicht aber den Dateiinhalt. Eine solche Datei kann, solange sie nicht überschrieben wurde, wiederhergestellt werden. Android wird unter dem Betriebssystem Linux ausgeführt und unterliegt ebenfalls diesem Problem. Gibt der Besitzer eines Android Smartphones sein Endgerät weiter, können Daten dieses Nutzers auf dem Datenträger verbleiben. Eine Lösung dieses Problems wird in dieser Arbeit vorgestellt.

Zu Beginn werden drei Teilgebiete aufgearbeitet, welche dieser Arbeit zugrunde liegen. Benutzerdaten unter Android machen das erste Teilgebiet aus. Unter Verwendung des Referenzgerätes Samsung Galaxy S3 wurden die einzelnen Partitionen des Datenträgers untersucht, um die Speicherorte von Benutzerdaten bestimmen zu können. Das zweite Gebiet umfasst den Löschvorgang an sich und speziell auf NAND-Datenträgern. Das dritte Teilgebiet beinhaltet die Techniken der Mobilfunkforensik, die es ermöglichen, ein sicheres Löschen zu evaluieren.

Im Laufe der Nutzung eines Smartphones ändern sich die auf dem Datenträger gesicherten Daten oftmals. Die Gründe dafür liegen innerhalb des Lebenszyklus von Dateien. Das Löschen ist der letzte Abschnitt des Lebenszyklus einer Datei. Die verschiedenen Möglichkeiten des Löschens unter Android liefern einen Ansatz für das sichere Löschen.

Mithilfe der Endgeräte Samsung Galaxy S2 und S3 wurden Untersuchungen durchgeführt, die aufzeigen, ob ein sicheres Löschen ohne externe Software möglich ist. Dabei wurde sowohl das Zurücksetzen des Systems untersucht als auch die verfügbaren Optionen zum manuellen Löschen von Dateien. Diese Untersuchungen liefern unterschiedliche Ergebnisse. Zwar ist das Zurücksetzen des Systems eine mögliche Alternative zum sicheren Löschen, allerdings gilt dies nicht für jedes Endgerät. Die Optionen zum Löschen, welche die Referenzgeräte mitbringen, weisen Lücken in der Sicherheit auf.

Als Ergebnis der Arbeit wird ein Konzept vorgestellt, das es ermöglicht Benutzerdaten, wie sie auf einem Android Smartphone entstehen im laufenden System unwiderruflich zu löschen. Außerdem beinhaltet das Ergebnis eine Android-Applikation, die anhand des beschrieben Prinzips einen Teil des Konzeptes umsetzt.

Abstract

This Master Thesis deals with the process of secure deletion on Android-Smartphones. Deleting a file causes the operating system only to remove the reference to it, but not its content. As long as this file content is not overwritten, it can be recovered. Android runs under a Linux OS, so it also has this problem. If the user of a Smartphone sells his device, user data could remain in the memory. This work introduces a solution to this problem.

At the beginning, three subject areas, which are the bases of this work, will be discussed. The first one is user data under android. Using the reference device Samsung Galaxy S3, each partition has been investigated to find the location of user data. The second subject area includes the deletion in itself and especially on NAND memory. The third branch includes the mobile forensic techniques which allow evaluating a secure deletion. Over the usage of a smartphone the stored data change often. The reasons for this can be found within the life cycle of files.

Deletion is the last part of the life cycle of a file. The different ways of data deletion on Android provide an approach for secure deletion. Using the devices Samsung Galaxy S2 and S3, investigations have revealed whether secure deletion without the use of third party software is possible. In doing so, the system recovery as well as the available options of manual deletion was investigated. These investigations lead to differing results. Thought the system recovery delivers a possible alternative to secure deletion, this does not apply to each device. The options of deletion of the above-mentioned devices have gaps in security.

As a result of this work, a concept is presented that allows permanently deleting user data, as they occur on an Android smartphone, in a running system. Moreover, the result includes an Android application that implements a part of the introduced concept.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die überaus große Verbreitung von mobilen Endgeräten ist unumstritten. Das mobile Telefon gehört für viele Menschen zu einem zentralen Teil ihres Lebens. So ist es nicht von der Hand zu weisen, dass auf solch einem technischen Gerät Daten entstehen, die für den Benutzer sehr wichtig und zumeist privat sind. Da die heutige Technologie, nicht zuletzt wegen der industriellen Produktionsweise, einem sehr schnellen Wandel unterliegt, kommt es häufig vor, dass ein mobiles Endgerät den Benutzer wechselt. Betrachtet man diese Situation, stößt man auf folgenden Aspekt, welcher zum Kern der zugrunde liegenden wissenschaftlichen Arbeit gehört. Das Löschen von Daten wird zumeist nicht so vorgenommen, dass diese unwiederbringlich entfernt sind. Die übliche Strategie beim Löschen einer Datei besteht darin, den Verweis zu dieser innerhalb des Dateisystems zu entfernen. Somit kann der Benutzer des Endgerätes, über das Betriebssystem mit welchem er interagiert, die gelöschte Datei nicht mehr finden. Als sicher gelöscht kann man diese Datei jedoch nicht bezeichnen. Die zentrale Frage, die es zu beantworten gilt ist somit folgende. *Wie ist es möglich, sein Smartphone an eine dritte Person abzugeben ohne befürchten zu müssen, dass diese eventuell in der Lage ist private Daten vom Speicher des Gerätes zu laden?*

Ein Smartphone besitzt unterschiedliche Typen von Datenträgern auf denen Benutzerdaten abgelegt werden können. Zum einen ist da der persistente interne Speicher, welcher auch im ausgeschaltet Zustand Daten sichert. Zum anderen gibt es den semipersistenten Hauptspeicher, der wesentlich kleiner und schneller als der interne Datenträger ist und deswegen unter anderem Benutzerdaten zur Verarbeitung zwischenspeichert. Jedoch sind solche Daten bei einem Neustart des Endgerätes unwiederbringlich gelöscht. Ein dritter Typ ist der externe Speicher in Form einer Flash Karte. Auch dieser hält unabhängig vom Betriebszustand des Endgerätes Daten fest. Dieser Typ von Datenträger ist nicht an ein Endgerät gebunden und wird vom Benutzer bei der Abgabe seines Endgerätes meist einbehalten. Somit liegt der Fokus dieser Arbeit auf den Daten, die sich auf dem internen Speicher eines Smartphones befinden.

Benutzerdaten liegen in unterschiedlicher Form auf einem Smartphone zur Verfügung. Entweder sichern Applikationen in ihren Verzeichnisstrukturen Daten, oder der Nutzer selbst legt Daten in Form einer Datei direkt auf einem freien Bereich des

Datenträgers ab. Möchte der Nutzer solche Daten löschen müssen ebenso unterschiedliche Methoden verwendet werden.

Wie in [Idc13] zu sehen ist, teilen sich zwei Systeme nahezu den gesamten Markt für Smartphones, Android mit 75.0% und Apple iOS mit 17.3%[1]. In dieser Arbeit werden die Untersuchungen ausschließlich auf dem System des Marktführers Android durchgeführt. Dieses Software Paket ist frei verfügbar und ermöglicht das Studieren des Quellcodes, um Fragen zur Umsetzung, welche sich im Laufe der Ausarbeitung stellen werden, zu beantworten. Seit dem Erscheinen des ersten Android-Systems im September 2008 sind bereits 17 Versionen veröffentlicht worden [Adb13]. Des Weiteren stellen mehr als 25 Firmen Endgeräte für diese Plattform her [Anv13]. Daraus lässt sich schließen, dass eine große Anzahl unterschiedlicher Kombinationen, von Hersteller und Android Version, auf dem Markt vorhanden sind. Da jede Version Änderungen hervorbrachte und zudem jeder Hersteller sein Produkt nach seinen Vorstellungen konzipiert, können die Unterschiede von einem zum nächsten Android-Smartphone sehr groß sein. Wie die Verteilung in Abbildung 1-1 zeigt wird die Version 4 am häufigsten verwend[2]. Der zurzeit erfolgreichste Hersteller ist der südkoreanische Konzern Samsung[3] [Idc13]. Aufgrund der aufgeführten Fakten werden sich die Untersuchungen dieser Arbeit auf die aktuelle Version 4 und zwei Endgeräten des Herstellers Samsung konzentrieren. Diese sind, das Samsung Galaxy S3 GT-I9300 und der Samsung Galaxy S2 GT-I9100.

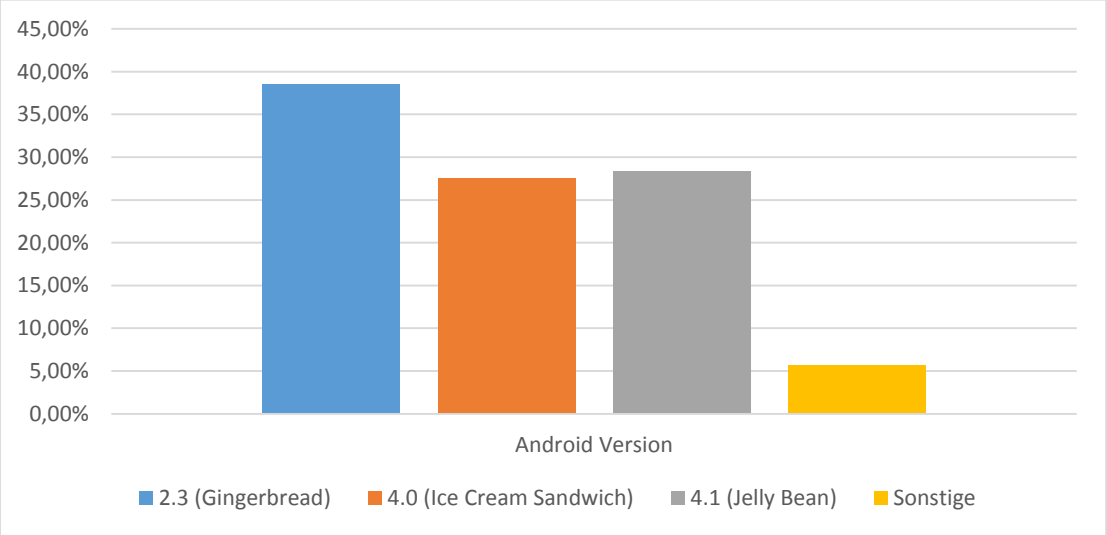


Abbildung 1-1 Verteilung der Android Versionen [Add13]

[1] Diese Zahlen gelten für das erste Quartal 2013 und beziehen sich auf den Marktanteil weltweit.
[2] Die Nutzung der Android Versionen, bezüglich der Aktualisierungsanfragen der Nutzer.
[3] Stand erstes Quartal 2013

1.1 Ziel der Arbeit

Das Ziel der Arbeit liegt darin, ein Konzept zu entwickeln, mit dem Benutzerdaten auf Android Smartphones unwiederbringlich gelöscht werden können. Dies soll auf der Grundlage des Wissens der folgenden drei Fachgebiete geschehen. Das erste betrifft das Android Softwarepaket, speziell die Sicherung von Benutzerdaten. Das zweite umfasst den Vorgang des Löschens auf Datenträgern, speziell jener mit NAND Technologie. Das Dritte schließlich beinhaltet die Techniken der Mobilfunk-Forensik, zur Sicherung von Daten. Bevor ein Konzept erstellt wird, soll im Vorhinein untersucht werden, wie sicher unterschiedliche Möglichkeiten des Löschens unter Android sind. Mit sicher ist hier gemeint, ob Daten nach der Durchführung eines Löschvorganges wiederhergestellt werden können oder nicht. Anhand von praktischen Beispielen soll aufgezeigt werden, wie ein Android Smartphone mit Daten umgeht. Eine Gegenüberstellung der Aktionen des Benutzers und den dazu parallel durchgeführten internen Funktionsabläufen zum Sichern, beziehungsweise Löschen der eingegebenen Daten, liefert dabei eine nachvollziehbare Sicht auf das System. Nach der theoretischen Aufarbeitung des Themas werden Teile des erstellten Konzeptes auf der Android Plattform implementiert, um eine praktisch anwendbare Lösung des in dieser Arbeit thematisierten Problems bereitstellen zu können.

1.2 Struktur und Inhalt

Das Kapitel 2 behandelt drei Teilgebiete, welche die Grundlagen dieser Arbeit ausmachen. Zunächst wird die Android Plattform thematisiert. Dabei wird beschrieben wie der Datenträger eines Endgerätes inhaltlich aufgebaut ist. Außerdem wird beantwortet wo Benutzerdaten gesichert werden, welche Arten davon existieren und wie diese gesichert werden. Ein weiteres Teilgebiet umfasst den Löschvorgang und den Datenträger, auf dem dieser durchgeführt wird. Wichtige Unterschiede hinsichtlich des Löschvorganges zwischen einem magnetischen Datenträger und einem NAND Flash Speicher werden betrachtet. Der Fokus liegt jedoch auf dem Datenträger, wie er in aktuellen Android-Endgeräten verbaut ist. Operationen wie das Schreiben, Lesen und Löschen werden dabei näher beschrieben. Das dritte Teilgebiet umfasst die Computer Forensik, bezogen auf mobile Endgeräte. Mit ihren Werkzeugen ist es möglich, Daten, die wie oben beschrieben nicht mehr vom Betriebssystem gefunden

werden können, wiederherzustellen. Es wird gezeigt, welche Vorbereitungen nötig sind und wie die Umsetzung der Techniken aussieht.

Das Kapitel 3 gibt einen Überblick über verwandte Arbeiten auf dem Gebiet des sicheren Löschens auf NAND-Datenträgern. Dabei wird klar, welche Ansätze für eine Lösung bereits vorhanden sind und wo diese Masterarbeit anknüpft.

In Kapitel 4 wird schrittweise der Lebenszyklus von Benutzerdaten, wie sie auf einem Android Smartphone auftreten, betrachtet. Dabei wird eine Applikation verwendet, die Benutzerdaten, wie in Kapitel 2.1.2 beschrieben, erzeugt. Zu solchen Benutzerdaten gehören standardmäßig installierte Applikationen, wie eine E-Mail App oder der Internet Browser. Bei der Installation der Software bestimmt das System wo diese physisch gesichert werden und erzeugt Verzeichnisstrukturen, welche Benutzerdaten, wie in Kapitel 2.1.2 beschrieben, beinhalten. Bei der Verwendung der Applikation entstehen im Laufe der Zeit immer größere Datenmengen. Dadurch kommt es zu einer automatischen Verteilung dieser auf dem Datenträger. Die Gründe und die Art der Verteilung sind wichtige Parameter, wenn es um ein sicheres Löschen geht. Am Ende eines Zyklus liegt das Löschen der erstellten Daten. Dafür gibt es mehrere Herangehensweisen, die es zu erörtern gilt.

Kapitel 5 greift die Strategien zum Löschen der Benutzerdaten auf und beschreibt, wie diese auf den Referenzgeräten umgesetzt werden. Ein wichtiger Punkt ist die Option des Werksrests auf einem Android-Endgerät. Obwohl nicht jedes Smartphone über diese Anwendung verfügt, ist sie sehr prominent und gehört zu dem standardmäßigen Vorgehen eines Nutzers vor dem Verkauf. Es ist wichtig, zu wissen, was genau dabei passiert und ob dadurch gelöschte Daten wiederhergestellt werden können. Die Ergebnisse dieser Untersuchungen dienen der Entwicklung eines Konzepts zum sicheren Löschen.

In Kapitel 6 wird die Implementierung des in Kapitel 5 entwickelten Konzeptes aufgeführt. Der Aufbau und die Funktionsweise sowie eine Evaluierung der Software sind der Inhalte dieses Abschnittes.

Kapitel 7 fasst diese Arbeit abschließend zusammen und gibt einen Ausblick für weitere Arbeiten auf diesem Gebiet.

2 Grundlagen

In diesem Kapitel werden die Grundlagen zu den drei Teilgebieten dieser Masterarbeit beschrieben. Dazu soll zunächst geklärt werden, was unter Benutzerdaten auf einem Android Smartphone zu verstehen ist. Des Weiteren wird der Löschvorgang an sich und speziell auf NAND-Datenträgern, wie sie in heutigen Smartphones verwendet werden thematisiert. Für das Verständnis im weiteren Verlauf der Ausarbeitung ist es ferner wichtig die Vorgehensweisen der Forensik auf einem mobilen Endgerät, mit Android System, zu kennen.

2.1 Benutzerdaten auf einem Android-System

Spricht man von Benutzerdaten ist nicht sofort klar, welche Daten unter diesen Begriff fallen. Speziell für diese Arbeit ist es wichtig, den Rahmen festzulegen, der definiert, wann Daten in diesem Zusammenhang wichtig sind und welche ignoriert werden können. Ein erster wichtiger Punkt ist, dass es um Daten geht, die auf einem mobilen Endgerät verwendet werden, allerdings nicht unbedingt ausschließlich darauf erstellt wurden. Der zweite Punkt unterscheidet den Urheber der Daten. Nimmt man beispielsweise eine Bilddatei her, kann diese durchaus unter die Kategorie Benutzerdaten fallen, allerdings nur wenn sie vom Nutzer des Endgerätes darauf geladen wurde. Eine Bilddatei hingegen, welche durch die Installation einer Android Standard Applikation vom Werk aus auf dem Speicher liegt, fällt nicht unter den Begriff der Benutzerdaten. Diese Unterscheidung ist vor allem wichtig, wenn man eine Analyse mittels File Carving vornimmt (siehe Kapitel 2.3.1). Sowohl die Applikations-Datei, als auch eine mögliche Datei des Nutzers würden in diesem Fall als Ergebnis anfallen und es gilt sie richtig einzuordnen.

Grundsätzlich kann man also von folgender Definition ausgehen. Benutzerdaten sind solche, die abgesehen von den System- und Anwenderdaten des Herstellers auf dem Endgerät gespeichert werden. Allerdings sind nicht alle Daten, welche durch die Benutzung entstehen als wichtig einzuordnen. Ausgehend von forensischen Untersuchungen, bei denen nach empfindlichen Daten gesucht wird [Hoo12], muss auch hier eine Unterscheidung gemacht werden. Eine durch den Benutzer installierte Applikation ist aus forensischer Sicht nicht unbedingt interessant [Hoo12]. Möchte der Benutzer jedoch seine Daten sicher löschen, wird er auch installierte Software

entfernen. Die Daten wiederum, die durch die Benutzung einer solchen Applikation entstehen sind zu einem großen Teil als empfindliche einzuordnen. Wie diese Daten aussehen wird in diesem Kapitel noch näher beschrieben. Stellt ein Benutzer Daten öffentlich zur Verfügung, sind diese für einen Forensiker uninteressant, da sie ohnehin zugänglich sind [Hoo12]. Ein Nutzer jedoch, der sein Handy weitergeben möchte, will auch solche Daten löschen. Geht es um das Löschen von Daten, kann also nicht zwischen empfindlichen und uninteressanten Daten unterschieden werden.

2.1.1 Partitionierung eines Datenträgers

Um hervorzuheben auf welche Bereiche eines Datenträges in einem Android Smartphone geachtet werden muss, wenn nach Benutzerdaten gesucht wird, soll in diesem Kapitel eine herkömmliche Partitionierungen betrachtet werden. Da eine Untersuchung mit forensischen Mitteln, bezüglich der Wiederherstellung von gelöschten Daten, sehr aufwendig sein kann, ist es von Vorteil die Datenmenge von vornherein zu reduzieren.

Das quelloffene Software Paket Android wird von vielen Herstellern verwendet [Anv13]. Einen Auszug von Smartphone Herstellern mit Android Systemen zeigt Abbildung 2-1. Diese folgen allerdings keinem standardisierten Verfahren, die Datenträger ihrer Endgeräte zu strukturieren[4]. Somit ist es nicht möglich eine allgemeingültige Aussage zu machen, die für jedes Smartphone zutrifft. Es existieren jedoch Ähnlichkeiten, die nötig sind, um eine Benutzung unter Android möglich zu machen. In diesem Unter-Kapitel soll deshalb anhand eines Referenzgerätes, dem Samsung Galaxy S3, geschildert werden, wie die Strukturierung auf einem Linux basierten Android Endgerät aussehen kann.

Um zur internen Aufteilung des Speichers auf einem Linux System zu gelangen muss zunächst die Schnittstelle zum Datenträger ausgemacht werden. In Linux-Systemen werden verbundene Geräte (**Dev**ices) im Verzeichnis dev aufgeführt [Cob13]. Auf einem Android Endgerät finden sich hier sämtliche Speichergeräte, mit denen das Betriebssystem agiert. Dazu gehören der Hauptspeicher, der externe Speicher, falls ein Datenträger eingelegt wurde und der interne Speicher. Diese

[4] Das zeigte der Vergleich der für diese Arbeit zur verfügung stehenden Endgeräte Samsung Galaxy S2 und S3.

werden als Verzeichnisse betrachtet und bilden somit einen Zugang zu den verbundenen Datenträgern [Cob13]. Ein solches Verzeichnis existiert ebenfalls auf dem Samsung Galaxy S3 unter dem Pfad /dev/block. Hier finden sich alle Partitionen, in die der interne Datenträger unterteilt ist (siehe Abbildung 2-2). Der Zugang zu den Partitionen kann sich von System zu System unterscheiden, dies hängt unter anderem vom verwendeten Speicher ab. Das Referenzgerät verfügt über einen Flash Datenträger vom Typ eMMC. Dieser Datenträgertyp ist mit einem internen Controller versehen[5], über welchen Zugriffe auf Daten erfolgen [Mcm10].

- Acer
- Google
- HTC
- Lenovo
- LG
- Motorola
- Panasonic
- Samsung
- Sony

Abbildung 2-1 Hersteller von Android Smartphones [Anv13]

Der gesamte Speicher des Referenzgerätes ist in zwei Teile und insgesamt vierzehn Partitionen gegliedert. Wobei Teil eins (mmcblk0p), den internen Speicher ausmacht und Teil zwei (mmcblk1p) die externe SD Karte, wie sie im Kapitel 2.1.2 beschrieben wird. Diese Partitionen dienen unterschiedlichen Aufgaben, welche es gilt zu untersuchen. Dadurch wird klar, welche Speicherabschnitte eventuell wichtige Benutzerdaten enthalten und welche Partitionen vernachlässigt werden können.

[5] Eine andere Variante ist es, eine Softwareschnittstelle zur Verfügung zustellen, die mit dem Speicher kommuniziert. Die Bezeichnung dafür lautet MTD.

cd /dev/block

interner Speicher	externer Speicher
mmcblk0p1	mmcblk1p1
mmcblk0p10	mmcblk1p5
mmcblk0p11	
mmcblk0p12	
mmcblk0p2	
mmcblk0p3	
mmcblk0p4	
mmcblk0p5	
mmcblk0p6	
mmcblk0p7	
mmcblk0p8	
mmcblk0p9	

Abbildung 2-2 Partitionen des Samsung Galaxy S3

Im Folgenden werden einige Partitionen beschrieben und zum Teil mit Beispielen, bezüglich des Referenzgerätes Samsung Galaxy S3, versehen. Dabei werden in Klammern jeweils die entsprechende Partition und die Größe in Megabyte aufgeführt. Die Bezeichnungen entsprechen, den vom System verwendeten. Auf dem Samsung Galaxy S3 können im Pfad /sys/devices/platform/dw_mmc/mmc_host/mmc0/mmc0:0001/block/mmcblk0 zu jeder Partition die oben genannten Informationen gefunden werden. Im Anschluss kann festgehalten werden, wieviel des internen Speichers vernachlässigt werden kann, wenn es um Benutzerdaten geht.

Die Boot Partitionen (mmcblk0p1 und mmcblk0p2 je 4,2, mmcblk0p5 8,4)

Je nach Konzipierung des Endgerätes existieren meist mehrere Partitionen, die im Zusammenhang mit der Bootsequenz eines Smartphones stehen. Das Referenzgerät Samsung Galaxy S3 besitzt gleich drei davon, welche als BOTA0 und BOTA1 und Boot bezeichnet werden.

Wie in [Hoo12] beschrieben besteht der Bootprozess aus sechs Phasen bei denen jeweils Daten wie der Boot-Loader und Initialisierungsskripte sowie weitere Informationen vom internen Speicher in den Hauptspeicher geladen werden. Für einen

tieferen Einblick in den Bootverlauf, vom Einschalten des Gerätes bis zur Betriebsbereitschaft, wird auf die angegebene Quelle verwiesen. Hier ist zunächst einmal nur wichtig, dass sich in diesem Bereich des Speichers keine Benutzerdaten befinden. Eine Änderung dieser Daten würde eventuell die Funktion des Endgerätes beeinflussen. Des Weiterern hat der Benutzer auf diesen Bereich des Speichers, ohne eine Änderung der Rechtevergabe, keinen Zugriff.

Die EFS Partition (mmcblk0p3 20,9 MB)

In der EFS Partition wurden beim Samsung Galaxy S3 folgende Inhalt gefunden. Die IMEI, die Identifikation Nummer, welche einem Smartphone eindeutig zugeordnet ist [Net13]. Des Weiteren werden hier die Bluetooth Verbindungs-Adresse und Daten für drm, der digitalen Rechtemanagement Software, gesichert [AdD13]. Die Wi-Fi Protokolle können hier ebenfalls gefunden werden.

Auf alle hier befindlichen Daten hat der Benutzer keinen Zugriff. Somit finden sich an dieser Stelle keine relevanten Daten, die es eventuell zu löschen gilt. Mehr noch dürfen diese Daten keinesfalls gelöscht werden, da ansonsten keine Verbindung mehr zu einem Netzwerk hergestellt werden kann [Net13].

Die PARAM Partition (mmcblk0p4 8,4)

Diese Partition hält Dateien bereit, welche vom System in einem Benutzer-Warte-Modus[6] gebraucht werden. Dies ergab die Untersuchung der PARAM Partition des Referenzgerätes. Vorwiegend handelt es sich um Bilddateien. Beispielsweise findet sich hier die Bilddatei, welche angezeigt wird, während der Akku geladen wird. Oder jene, die angezeigt wird während das Gerät einen Download von System Software durchführt. Auf dem Referenzgerät ist in dieser Partition das Bild aus Abbildung 2-3 zu finden. Dieses wird beim Starten des Gerätes auf dem Display angezeigt.

[6] Damit ist beispielsweise der Zustand beim Update der Firmware gemeint oder die Wartezeit beim Hochfahren des Endgerätes.

Abbildung 2-3 Displayausgabe beim Hochfahren des Samsung Galaxy S3 (Größe angepasst)

Die Recovery-Partiton (mmcblk0p6 8,4)

Die Recovery-Partition dient, wie die Bezeichnung vermuten lässt, dem Wiederherstellen des Endgerätes [Hoo12]. Es ist eine bootfähige Partition und deshalb, in internen Log Dateien, wie dem Recovery-Log[7], mit dem Dateisystem emmc ausgewiesen. Funktionen, die bei der Durchführung einer Recovery angewendet werden, befinden sich auf dieser Partition. Wie dieser Prozess abläuft, wird im Kapitel 5.1 detaillierter behandelt. An dieser Stelle ist nur wichtig, dass der Benutzer keinen Einfluss auf die Daten dieser Partition hat und somit keinerlei Benutzerdaten darauf zu finden sind.

Die Modem Partition (mmcblk0p7 33,6)

Die Modem oder Radio Partition ist wie bei der zuvor beschriebenen Partition, mit dem Dateisystem emmc ausgewiesen. Ein Zugriff auf die Daten ist somit nicht mit gewöhnlichen Mitteln möglich. Dies hat ein Versuch mittels Einhängen in eine Ubuntu Workstation ergeben. Im Forum der XDA Developer wird im Bezug zu dieser Partition vorwiegend über das Herstellen von Rootrechten diskutiert. Bei fehlerhafter Durchführung kann somit die Funktion des Smartphones verloren gehen. Die genaue Funktion und der genaue Inhalt dieser Partition erfordern weiterer Recherche.

Die Cache Partition (mmcblk0p8 1073,7)

Ein Cache wird verwendet, um Dateien zwischenzuspeichern [Hei13]. Diese Partition ist mit dem gleichen Dateisystem beschrieben, wie die System und Benutzerdaten Partition (EXT4). Es kann direkt darauf zugegriffen werden. Laut Foreneinträgen

[7] Diese Datei befindet sich in der Cache Partition und wird bei jedem Zurücksetzen automatisch vom System erstellt. Dies ergaben die Untersuchungen im Kapitel 5.1

werden hier heruntergeladene Applikationen zwischengespeichert. Zwar befinden sich hier keine empfindlichen Benutzerdaten wie Namen, Adresse oder Ähnliches, dennoch könnten Rückschlüsse auf die Benutzung des Endgerätes gezogen werden.

Die System Partition (mmcblk0p9 1610,6)

Auf der Systempartition sind das Betriebssystem Linux sowie die Android Software installiert [Hoo12]. Auf dem Samsung Galaxy S3 finden sich hier Applikationen und Medien-Daten wie Klingeltöne, die durch Android vorinstalliert sind. Diese Daten sind den Benutzerdaten, die es zu löschen gilt, ähnlich jedoch vernachlässigbar. Eine Übersicht der Daten, welche fälschlicherweise als Benutzerdaten identifiziert werden könnten, sollte der komplette interne Speicher mittels File Carving durchsucht werden, findet sich in der folgenden Tabelle.

Verzeichnis	Inhalt
App	apks sämtlicher Standard Apps
Fonts	Installierte Schriftarten
Media	Systemtöne und Herstellervideos
Usr	Benutzersystem Einstellungen wie die Tastaturvariante
Wallpaper	Standard Hintergrundbilder

Tabelle 2-1 Systeminhalte des Referenzgerätes, welche als Benutzerdaten interpretiert werden könnten.

Für die Untersuchungen in Kapitel 5 und die Implementierung sind die Dateien in den Verzeichnissen bin und xbin noch zu erwähnen. In diesen befinden sich die verfügbaren Anwendungen aus dem Linux Repertoire, wie beispielsweise „dd“. Des Weiteren kann hier im Falle eines gerouteten Gerätes der Super User Befehl „su“ gefunden werden.

Die Preload Partition (mmcblk0p10 587,2)

Diese Partition wird zumeist als hidden bezeichnet. Sie wird vom Hersteller dafür benutzt Daten, wie Applikationen, Bilder und Videos zu sichern, die bei einer neuen

Installation des Systemes immer vorhanden sind. Die folgende Tabelle zeigt die Daten am Beispiel des Samsung S3 Referenzgerätes.

Bezeichnung	Typ
Lieferheld	App
newsap	App
hsr	App
kaufda	App
mytaxi	App
ntv	App
over the horizon	Musik
wonder of nature	Video

Tabelle 2-2 Inhalt der Partition HIDDEN.

Hier werden Daten gespeichert, die vom Typ her denen gleichen, die als Benutzerdaten bezeichnet werden können. Da der Benutzer aber keinen Einfluss auf diese hat, können sie beim Löschen vernachlässigt werden.

Die OTA Partition (mmcblk0p11 8,3)

OTA ist die Abkürzung für Over The Air. Möchte man ein Update der Android Version vornehmen, so geschieht das über diese Partition. Aus diesem Fakt lässt sich einfach schließen, dass sich keine Benutzerdaten darauf befinden können. Der Benutzer hat keinen direkten Zugriff auf diese Daten, sondern bestimmt nur den Zeitpunkt des Updates, bei dem die Daten auf dieser Partition verarbeitet werden. Da ein Update der Android Software den gewöhnlichen Betrieb ausschließt, muss das Update auf einem anderen System ablaufen. Dafür wird diese bootfähige Partition verwendet. Sie ist wie zuvor bei der Recovery und Modem Partition mit dem Dateisystem emmc ausgewiesen.

Die data Partition (mmcblk0p12 12381)

Der größte Teil des internen Speichers, beim Referenzgerät Samsung Galaxy S3, fällt auf diese Partition. In Abbildung 2-4 ist zu erkennen, dass 79% der gesamten Speicherkapazität durch mmcblk0p12 belegt ist. Die Bezeichnung dieser Partition, deutet darauf hin, dass hier der entscheidende Anteil, der vom Benutzer erzeugten Daten liegen muss. Die vom Benutzer installierten Applikationen und die damit erzeugten Daten werden auf dieser Partition gespeichert. Des Weiteren findet man Fotos, Videos und weitere aktiv gespeicherte Dateien, die nicht explizit auf die externe SD Karte geladen wurden. Hier befinden sich, wie in mmcblk0p9 und mmcblk0p10 viele Daten, die nicht von Interesse sind, wenn es um das sichere Löschen von Benutzerdaten geht. Dazu gehört solche Software, die zur Erzeugung von Benutzerdaten nötig ist. Damit ist beispielsweise für Benutzerdaten in Form eines Textes, ein Editor gemeint.

Damit ist klar, dass das Hauptinteresse der Untersuchungen in Kapitel 5 auf die Partition mmcblk0p12 gerichtet ist. Des Weiteren lassen sich Rückschlüsse zur Benutzung aus den Daten auf Partition mmcblk0p8 ziehen. Somit sind wie in Abbildung 2-4 zu erkennen ist, 14% des internen Speichers von vornherein zu vernachlässigen, wenn es um Benutzerdaten geht. Alle anderen Partitionen werden in dieser Verteilung vernachlässigt, da sie weniger als 1% des gesamten Speicherbereiches ausmachen.

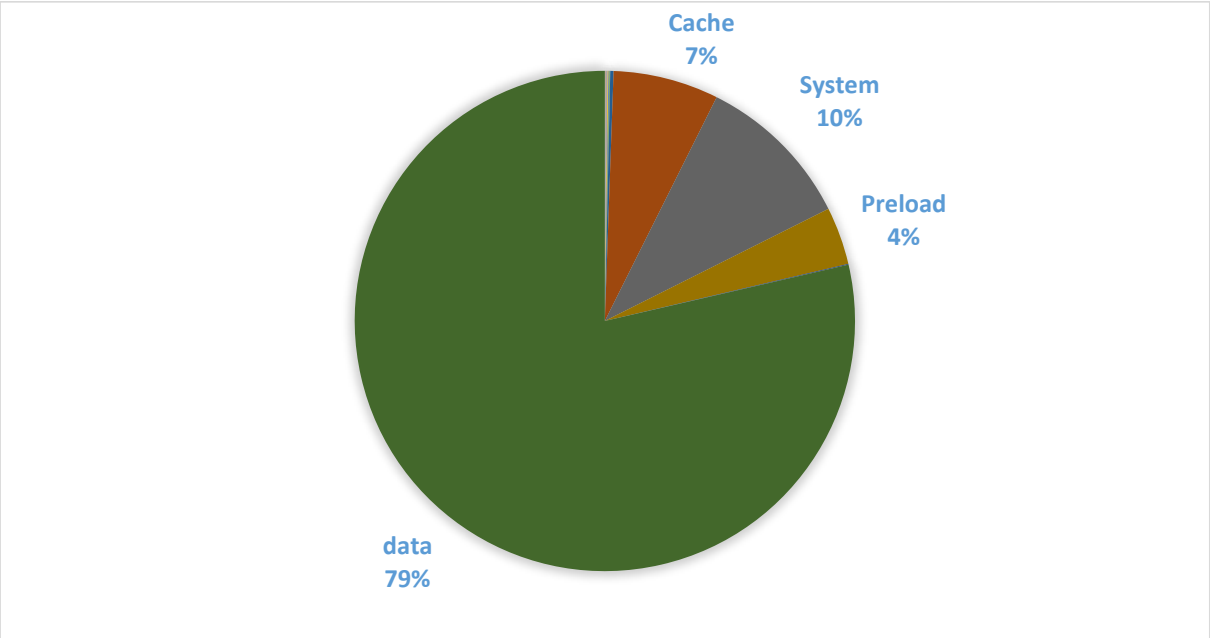


Abbildung 2-4 Verteilung des internen Speichers des Samsung Galaxy S3

Der Mount Befehl

Der Mount Befehl ist ein Standard Befehl, der auf jedem Linux System verfügbar ist [Hoo12]. Unter Android befindet er sich in der System Partition im Verzeichnis bin. Führt man diesen Befehl aus, werden alle Verzeichnisse, die in das System eingehängt sind aufgelistet (siehe Abbildung 2-5). Nicht alle der oben genannten Partitionen werden zu jeder Zeit genutzt. Wie sich gezeigt hat, dienen sie speziellen Zwecken. Anhand der Partitionen, die im laufenden Betrieb tatsächlich in das System eingebunden sind, lässt sich erkennen, ob auf diesen überhaupt Benutzerdaten entstehen können. Ist das nicht der Fall, kann nicht auf diesen Speicher geschrieben werden, während der Nutzer Daten eingibt oder das Endgerät auf andere Art und Weise nutzt. Somit entstehen auch keine Benutzerdaten auf diesen Partitionen.

Abbildung 2-5 Mount-Befehl beim Referenzgerät während normalem Betrieb

Mittels der Datei Filesystems, die sich auf jedem Android-Endgerät befindet, lässt sich bestimmen, welche Dateisysteme auf dem Endgerät verwendet werden [Hoo12]. Folgende Auflistung gibt Filesystems für das Samsung Galaxy S3 aus. Diejenigen Dateisysteme mit der Bezeichnung nodev sind emulierte Dateisysteme und befinden sich nicht auf einem physischen Speicher [Hoo12]. Auf dem Hauptspeicher des Endgerätes geladene Benutzerdaten können ignoriert werden, da nach einem Neustart alle Daten dieses Speichers gelöscht sind [Hoo12].

```
shell@android:/proc $ cat filesystems
nodev    sysfs          nodev  sockfs          nodev  fusectl
nodev    rootfs         nodev  usbfs                  ext2
nodev    bdev           nodev  pipefs                 ext3
nodev    proc           nodev  anon_inodefs           ext4
nodev    cgroup         nodev  devpts                 vfat
nodev    tmpfs          nodev  ramfs                  msdos
nodev    binfmt_misc    nodev  ecryptfs               fuseblk
nodev    debugfs        nodev  fuse                   exfat
```

Somit ist klar, dass die ersten neun und das vorletzte Dateisystem aus Abbildung 2-5 Mount-Befehl beim Referenzgerät während normalem Betriebvon vornherein vernachlässigbar sind. Die letzte Zeile beschreibt die emulierte externe SD Karte wie sie in Kapitel 2.1.2 beschrieben wird. Der Zugriff auf die System Partition ist nur in lesender Form möglich, was durch die Bezeichnung ro belegt ist. Die Partition efs ist nötig, um Rechte für Medien Daten zu prüfen[8]. Das System verifiziert damit ob geschützen Mediendateien abgespielt werden dürfen. Somit verbleiben die beiden wichtigsten Partitionen im Bezug auf Benutzerdaten Cache und Data. Diese können im Betrieb als einzige mit Daten, welche vom Nutzer erzeugt wurden, beschrieben werden.

Nachdem nun aufgeführt wurde, welche Teile des persistenten Speichers wichtig und welche zu vernachlässigen sind, soll nun ein genauerer Blick auf die wichtigen Partitionen geworfen werden. In der Verzeichnisstruktur der wichtigen Partitionen ist es relevant, zu wissen, wo genau das System Benutzerdaten ablegt.

Cache-Partition (cache)

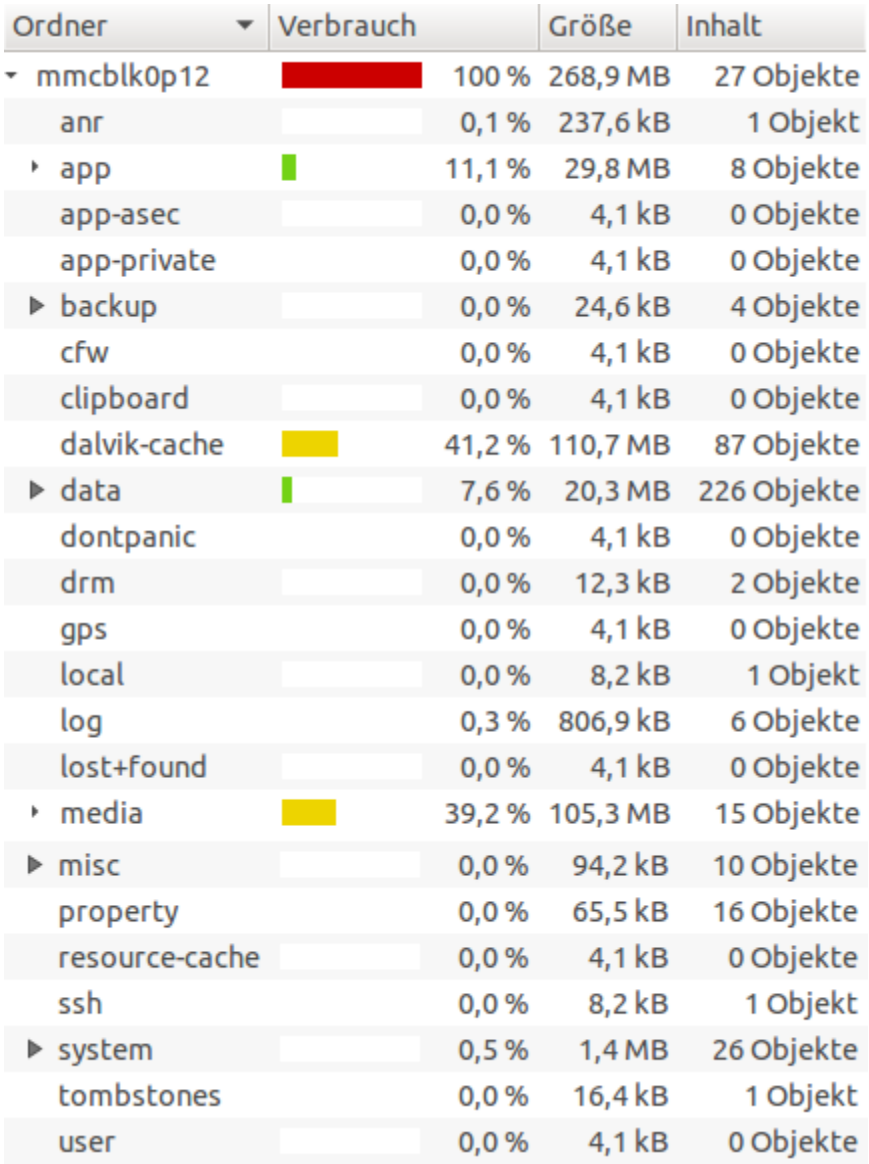
Beim Referenzgerät enthielt die Cache Partition einen Lost&Found Ordner und außerdem den Ordner Recovery. Aus Letzterem kann sowohl der Zeitpunkt des letzten Zurücksetzens des Systems, als auch die dabei ausgeführten Schritte bestimmt werden.

Laut XDA Developer verwendet das System diesen Speicherplatz zum Auslagern von Applikation. Jedoch geschieht dies nicht bei der Nutzung durch den Anwender, sondern beim Herunterladen und Installieren einer App. Ist eine Applikation besonders groß, kann es passieren, dass der Cache überläuft und die App nicht installierbar ist. Es wurde überprüfen, ob beim Installieren von Apps, Daten auf dieser Partition zwischengespeichert werden. Dafür wurden beim Samsung Galaxy S3 mehrere Applikationen heruntergeladen. Anschließend wurde die Cache Partition mittels Hexeditor durchsucht. Bis auf die im vorigen Absatz beschrieben konnten keinerlei Daten gefunden werden.

[8] Vgl. Seite 9

Benutzerdaten-Partition (data)

Die Untersuchung von data liefert die meisten Ergebnisse bei der Suche nach Benutzerdaten. Die Verzeichnisstruktur soll nun Aufschluss darüber geben, wo solche Daten zu finden sind. Die oberste Ebene ist in Abbildung 2-6 zu sehen. Die wichtigsten Ordner werden im Folgenden genauer betrachtet, um aufzuzeigen welche Benutzerdaten dort zu finden sind.



Ordner	Verbrauch	Größe	Inhalt
mmcblk0p12	100 %	268,9 MB	27 Objekte
anr	0,1 %	237,6 kB	1 Objekt
app	11,1 %	29,8 MB	8 Objekte
app-asec	0,0 %	4,1 kB	0 Objekte
app-private	0,0 %	4,1 kB	0 Objekte
backup	0,0 %	24,6 kB	4 Objekte
cfw	0,0 %	4,1 kB	0 Objekte
clipboard	0,0 %	4,1 kB	0 Objekte
dalvik-cache	41,2 %	110,7 MB	87 Objekte
data	7,6 %	20,3 MB	226 Objekte
dontpanic	0,0 %	4,1 kB	0 Objekte
drm	0,0 %	12,3 kB	2 Objekte
gps	0,0 %	4,1 kB	0 Objekte
local	0,0 %	8,2 kB	1 Objekt
log	0,3 %	806,9 kB	6 Objekte
lost+found	0,0 %	4,1 kB	0 Objekte
media	39,2 %	105,3 MB	15 Objekte
misc	0,0 %	94,2 kB	10 Objekte
property	0,0 %	65,5 kB	16 Objekte
resource-cache	0,0 %	4,1 kB	0 Objekte
ssh	0,0 %	8,2 kB	1 Objekt
system	0,5 %	1,4 MB	26 Objekte
tombstones	0,0 %	16,4 kB	1 Objekt
user	0,0 %	4,1 kB	0 Objekte

Abbildung 2-6 Verzeichnisstruktur der Data Partition des Samsung Galaxy S3

data: Dieser Ordner beinhaltet die vom System als Daten bezeichnete Inhalte der installierten Applikationen. Diese Art von Daten werden im folgenden Kapitel 2.1.2 beschrieben. Wie Abbildung 2-6 zeigt sind 226 Ordner in diesem Verzeichnis zu finden.

Diese stehen für die vorinstallierten Strukturen, in welche die jeweiligen Applikationen, die während ihrer Benutzung erzeugten Daten ablegen. Beispiele hierfür sind Nachrichten der SMS App, Kontakte der Telefon App oder der Surfverlauf der Browser App.

media: Enthält einen sogenannten symbolischen Link [9] auf einen emulierten externen Speicher, wie er in Kapitel 2.1.2 beschrieben wird. Hier werden Dateien abgelegt, die der Nutzer durch eine USB Verbindung von einem Rechner, auf dem internen Speicher des Endgerätes gespeichert hat. Diese können laut den vorinstallierten Verzeichnissen Sounds, Notizen, Videos, Podcasts, Playlisten, Bilder etc. sein. Außerdem könnte jede weitere denkbare Datei hier gesichert sein, die der Nutzer entweder von seinem Rechner übertragen hat oder über ein Netzwerk geladen hat. Ist kein externer Speicher in Form einer SD Karte eingelegt, wird die Kamera App Aufnahmen in diesem emulierten externen Speicher sichern.

system: Die in diesem Verzeichnis wichtigen Benutzerdaten sind die Account-Daten. Auf einem Android Endgerät wird der Nutzer dazu aufgefordert einen Google Play Account anzulegen, um Applikationen herunterladen zu können. Dieser Account fungiert ab der ersten Anmeldung als Benutzer-Account und wird hier unter User gesichert.

misc: Dieses Verzeichnis enthält Netzwerkverbindungsdaten unter anderem auch jene für Drahtlosverbindungen. Die Datei /wifi/wpa_supplicant.conf listet alle Wi-Fi Verbindungen inklusive SID und Passwort auf und das absolut unverschlüsselt [Hoo12].

2.1.2 Datensicherung auf dem Android-System

Android ist kein Betriebssystem, sondern vielmehr ein Software Paket, das unter Linux betrieben wird [Hoo12]. Die Inhalte dieses Softwarepaketes liefern die Grundlagen für die Sicherung von Benutzerdaten auf einem Android-Endgerät. Zum einen sind dies, in Bezug auf die Android Schichten Architektur, das Framework zur Erzeugung von Applikationen, zum anderen die SQlite Bibliothek [Hoo12].

Die Android Entwicklungssoftware bietet unterschiedliche Möglichkeiten, um Daten zu speichern. Zunächst kann zwischen dem internen oder dem externen Speicher

[9] Eine Datei, die einen Verweis zu einem anderen Pfad hält [Kal13]

entschieden werden [Ado13]. Des Weiteren können Datenbanken, Schlüsselpaare und direkte Datei Sicherungen vorgenommen werden [Ado13]. Da es dem Autor einer Applikation freigestellt ist, wie er seine Software strukturiert, ist es schwierig, eine allgemeine Aussage über die Sicherung der Daten innerhalb einer Applikation zu treffen. In diesem Kapitel werden deshalb die Methoden, mit denen Benutzerdaten gesichert werden im Einzelnen beschrieben.

Wie aus dem ersten Abschnitt dieses Kapitels hervorgeht, ist der Ort an dem Applikationen Daten sichern festgelegt. Dieser Punkt wird bewiesen, wenn man sich die Methode ansieht, mit der ein Programmierer Daten für seine App speichern kann. [Ado13]. Da das Android System alle Daten, welche intern gespeichert wurden beim Deinstallieren einer App löscht, muss es festlegen wo diese gespeichert werden. Die Möglichkeiten eines Entwicklers Daten zu sichern werden deshalb insofern eingeschränkt, dass er beim Erstellen eines Verzeichnisses oder einer Datei mit der Methode getFilesDir() arbeiten muss [Ado13]. Diese liefert eine Datei beziehungsweise ein Verzeichnis in dem die gewünschten Daten gesichert werden können. Somit ist es zwar möglich innerhalb des App-Verzeichnisses, die Struktur frei zu gestalten, jedoch nicht an beliebiger Stelle im System. Dem Benutzer selbst ist es ebenfalls möglich sowohl auf dem internen als auch auf dem externen Speicher Benutzerdaten direkt abzulegen [Ado13].

Externer Speicher

Dieser Datenträger ist meist mit dem Dateisystem FAT32 formatiert [Hoo12]. Die Wahl dieses Dateisystems beruht auf der Kompatibilität zu vielen Systemen. Um Daten einfach auf ein Handy übertragen zu können, werden diese frei zugänglich gesichert [Hoo12]. Eine Ausnahme stellen auf diesen Speicher verlagerte Applikationen dar. Android bietet die Möglichkeit die installierte Software auf diesen Speicher auszulagern [Ade13]. Gewöhnlicherweise geht das über die App „Einstellungen“ und den darin vorhandenen Anwendungs-Manager, den Android mitliefert. Hier können sämtliche installierte und aktive Apps gefunden werden. Nutzt man diese Option, löscht das System nur die Rahmen-Daten der Applikation und speichert diese verschlüsselt auf der externen SD Karte [Hoo12]. Die Benutzerdaten, die bislang über diese Software erstellt wurden, verbleiben auf der Data Partition im internen Speicher [Hoo12]. Einige Applikationen erzeugen bei der Nutzung auch direkt auf dem externen

Speicher Benutzerdaten. Der größte Anteil an relevanten Daten wird vom Benutzer selbst in Form von Dateien auf dem externen Speicher abgelegt. Das können Fotos, Videos, explizit gesicherte Textdateien, PDFs und weitere Dateien dieser Art sein. Solche können auf zwei Wegen auf den Datenträger gelangen. Entweder sie werden direkt auf dem Endgerät erstellt oder durch das Laden über ein Netzwerk, beziehungsweise eine USB-Verbindung auf dem Endgerät gesichert. In letzterem Fall hängt das Betriebssystem des Rechners, mit dem die USB-Verbindung besteht, zwei Datenspeicher ein. Diese sind die externe SD Karte im FAT Dateisystem und die emulierte SD Karte (siehe 2.1.2.6) im gphoto2 Dateisystem. Dadurch ist ein Zugriff auf diese Speicherbereiche möglich.

Die meisten Benutzer neigen dazu, ihre externen SD Karte zu behalten, wenn sie ihr Mobiltelefon abgeben. Deswegen wird sich diese Arbeit nicht damit beschäftigen, wie sich externe Daten innerhalb des Endgerätes löschen lassen.

Benutzerdaten innerhalb einer Applikation (interner Speicher)

Apps sind zweifelsfrei der wichtigste Bestandteil von Android. Mit ihrer Hilfe wird praktisch die komplette Nutzung über den Anwender abgedeckt. Somit enthalten ihre Verzeichnis-Strukturen auch einen entsprechenden Anteil an empfindlichen Benutzerdaten. Nicht jede App enthält jedoch relevante Benutzerdaten. Ein Spiel beispielsweise könnte eventuell nur den Spielernamen enthalten, der Rückschlüsse auf den Nutzer zulässt. Standardmäßig vorinstallierte Apps wie die E-Mail App oder die Browser App hingegen enthalten eine Menge relevanter Benutzerdaten [Hoo12], weswegen in dieser Arbeit mit einer Standard Applikation der aktuellen Android Version gearbeitet wird. Die Möglichkeiten jedoch, wie Daten gesichert werden, sind für alle Applikationen gleich und werden in den folgenden Unterpunkten beschrieben. Wie diese Methoden explizit angewendet werden, um Benutzerdaten zu speichern, kann in den Untersuchungen in Kapitel 5 nachvollzogen werden.

SQLite Datenbanken

Im Schichtenmodel der Android Architektur findet sich ein Paket mit dem Namen SQLite [Hoo12]. Es steht für die Standardbibliothek des Datenbankmanagements unter Android. Viele der in Android gesicherten Benutzerdaten werden in SQLite

Datenbanken organisiert [Hoo12]. Betrachtet man die Standard Applikation aus den Untersuchungen in Kapitel 5, speichert diese nahezu alle Benutzerdaten in mehreren SQLite Datenbanken. Über die Android Schnittstelle android.database.sql ist es einem App-Entwickler möglich, eine Datenbank auf dem Speicher des Endgeräts anzulegen, die aus einer einzigen Datei bestehen [Ado13] [Hoo12]. Diese SQLite Dateien speichern ihre Datensätze in doppelt verketteten Listen, deren Objekte als Seiten bezeichnet werden [Hwa13]. Informationen zu einer SQLite Datei lassen sich aus dem Header[10] extrahieren. Abbildung 2-7 zeigt den Header einer *.db Datei aus dem Database Verzeichnis einer Android Applikation.

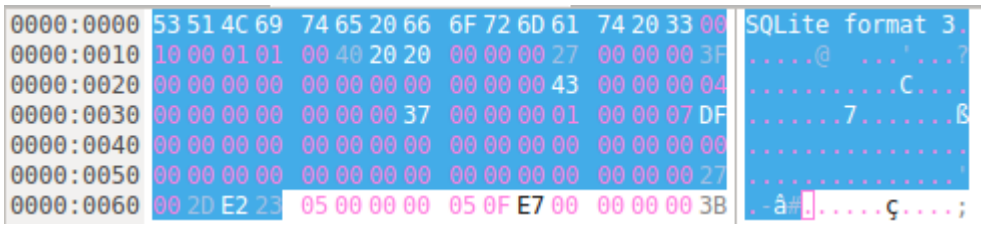


Abbildung 2-7 Header einer SQLite Datei

Die Größe der Seiten ist durch den Anwender wählbar [Hwa13]. In der Android Version 4.1.2 wurde die Größe der SQLite Seiten den Dateisystem Blöcken unter EXT4 angepasst[11]. Somit ist jede Seite 4096 Byte groß, siehe Byte 16-17 in Abbildung 2-7.

Für das Entfernen von Datensätzen aus einer SQLite Datenbank sind folgende Funktionen für diese Arbeit wichtig. Der auto_vacuum Modus bestimmt den Umgang mit gelöschten Datensätzen [Hwa13]. Hier gibt es drei Modi. Hat dieser den Wert 0, wird ein gelöschter Datensatz in eine sogenannte freelist gespeichert und an das Ende der Datenbank verschoben [Hwa13]. Im Modus 1 wird beim Ausführen eines commit Befehls die ans Ende verlagerte Seite mit gelöschten Datensätzen abgeschnitten und gehört nicht mehr zur Datei [Hwa13]. Im Modus 2 wird wie in 1 vorgegangen, allerdings löst nicht jeder commit Befehl das abtrennen der gelöschten Datensätze aus. Stattdessen muss dieser Schritt explizit ausgeführt werden [Hwa13].

Die zweite wichtige Funktion einer SQLite Datenbank wird von Reardon, Basin und Capkun in ihren Ausführungen zum sicheren Löschen erwähnt. Sie wird als Pragma

[10] Der Header in einer SQLite Datenbank entspricht den ersten 100 Byte dieser Datei.
[11] Dies wurde innerhalb der Untersuchungen auf der Android Version 4.1.2 für die vorinstallierten Applikationen überprüft.

Statement bezeichnet [Rbc13]. Gelöschte Datensätze werden dadurch mit Nullen überschrieben [Rbc13]. Allerdings reicht dieses Vorgehen nicht aus, da die Sicherheit vom verwendeten Dateisystem abhängt [Rbc13]. In den Untersuchungen im Kapitel 5 wird gezeigt, ob das Löschen von Datensätzen innerhalb einer SQLite Datei unter Android, zu einem Sicheren Löschen führt oder nicht.

Shared Preferences

Diese Methode bietet zwar nicht so viele Benutzerdaten, wie in einer SQLite Datenbank zu finden sind, jedoch sollte sie nicht ignoriert werden. Je nach Vorgehen des Entwicklers können sich hier Benutzerdaten befinden, die gelöscht werden sollten.

Bei Shared Preferences spricht man innerhalb einer Applikation von Schlüssel-Wert Paaren [Ado13]. Beispielsweise der Schlüssel istEsWahr vom Typ boolean und dem dazugehörigen Wert true. Der eigentliche Sinn dieser Daten ist es, Einstellungen zu sichern, um diese nach dem Schließen und Neustarten einer Applikation erneut zur Verfügung zu haben [Ado13]. Solche Daten werden in den jeweiligen Verzeichnissen im XML Format gespeichert und machen normalerweise nur einen sehr kleinen Teil der genutzten Datenkapazität einer Applikation aus.

Online (Cloud)

Die letzte Variante wie Daten in einer Applikation gesichert werden können, besteht in der Möglichkeit sie in eine Cloud zu schieben [Hoo12]. Im Zusammenhang mit dieser Arbeit ist es selbstverständlich nicht das Ziel diese Benutzerdaten auf irgendeine Weise zu manipulieren oder gar zu löschen. Der Aspekt, um den es geht, ist, die Beziehungen zu diesen Daten zu löschen. Das heißt, dass eine Applikation, welche Daten in einer Cloud verwaltet, die Adresse zu diesen Daten kennt und eventuell auch die Zugangsdaten. Genau diese Daten sind das primäre Ziel. Ein weiteres Ziel könnte sich ergeben, wenn eine Applikation nicht nur die Daten in einer Cloud verwaltet, sondern ebenso auf das Endgerät überträgt. In diesem Fall werden Benutzerdaten nach den bereits beschrieben Möglichkeiten, auch in solch einer Cloud-Applikation zu finden sein.

Emulierter externer Speicher

Solch ein Speicher fungiert wie ein externer Speicher, wird jedoch physisch auf dem internen Speicher gesichert. Standardmäßig dient er dem Zweck, dem Benutzer die Möglichkeit zu bieten, Daten zwischen einem Rechner und dem Smartphone auszutauschen. Bei der Verbindung über den USB Port, wird der emulierte externe Speicher als Massenspeicher auf dem Rechner eingehängt. Fortan können Dateien vom Endgerät auf den Rechner, zum Beispiel Aufnahmen der Kamera App, geladen werden. Oder es können umgekehrt, Dateien vom Rechner auf dem Endgerät gesichert werden.

2.2 Das Löschen von Daten

Die Bezeichnung Löschen im Zusammenhang mit Dateien auf einem Computer mit einer herkömmlichen Festplatte trifft nicht zu. Weist ein Benutzer das Betriebssystem seines Rechners an eine Datei zu löschen, wird dieses nicht die Datei selbst von der Festplatte löschen, sondern vielmehr den Verweis darauf [Fox09]. Den Beweis dafür liefern die Untersuchungen in Kapitel 5.2.2 für das Android System unter Linux. Auch bei anderen Kombinationen aus Hardware und Betriebssystem ist dieses Vorgehen üblich, um den Vorgang des Löschens möglichst schnell durchführen zu können [Fox09]. Aus diesem Grund ist es möglich, mit den Mitteln der Computer-Forensik gelöschte Daten wiederherzustellen. Mehr dazu im Kapitel 2.3.1, welches Techniken der Mobilfunkforensik behandelt.

Um ein tatsächliches Löschen durchzuführen ist zusätzliche Software nötig, welche die Speicherbereiche, auf denen betroffene Dateien gesichert sind, überschreibt. Bereits 1996 beschrieb Peter Gutmann wie Daten sicher gelöscht werden können, um zumindest davon auszugehen, dass es nur mit größtem Aufwand möglich ist, diese wiederherstellen zu können [Gut96]. Mehrfaches Überschreiben mit speziellen Bytemustern stellt hierbei das sichere Löschen her [Gut96]. Dieses und weitere Verfahren wie

- 5220.22-M-Standard des US amerikanischen Verteidigungsministeriums,
- VSITR-Standard der BSI,
- Bruce-Schneier-Algorithmus

zielen dabei auf Daten ab, welche auf einem magnetischen Datenträger gesichert wurden [Fox09]. Durch die Ladung[12] kleinster Bereiche auf einem solchen Datenträger wird der Wert eines Bits gespeichert [Fox09]. Die Ladung in der Umgebung des Bereiches, welcher für ein Bit steht kann Rückschlüsse über frühere Ladungszustände erlauben [Bit08]. Deswegen ist es nötig Bereiche auf denen zu löschenden Daten gesichert waren mehrfach zu überschreiben, um sicher zu gehen, dass diese nicht wiederhergestellt werden können. Da die Dichte der Datensicherung auf magnetischen Datenträgern, seit den Untersuchungen von Peter Gutmann, stets zunahm reduzierte sich auch die Anzahl an nötigen Überschreibungsdurchläufen [Fox09]. Dies liegt wiederum daran, dass der Bereich um ein Bit herum, aus dem ein Rückschluss auf die zuvor gültige Ladung gezogen werden kann, kleiner wurde [Bit08]. Die oben genannten Vorgehensweisen beruhen also auf den gleichen Hintergrund wie Gutmann, haben jedoch ihrer Strategie den Entwicklungen der Hardware angepasst. Laut Peter Gutmann gibt es stets die Möglichkeit Daten wiederherzustellen, selbst von einem gelöschten und mehrfach überschrieben Bereich eines Datenträgers. *„it is effectively impossible to sanitise storage locations by simple overwriting them, no matter how many overwrite passes are made or what data patterns are written. […]the task of an attacker can be made significantly more difficult, if not prohibitively expensive."* [Gut96]. Bis heute werden Programme angeboten, die nach den empfehlungen von Gutmann Daten sicher Löschen[13].

Ein Android Smartphone wird mit dem Betriebssystem Linux betrieben und unterliegt ebenso dem oben beschrieben Verfahren zum „Löschen" von Daten. Das heißt es werden zunächst einmal nur Metadaten gelöst, welche Aufschluss über den Speicherbereich geben, an dem eine Datei gespeichert ist. Dies führt dazu, dass dieser Bereich nicht mehr als belegt bezeichnet wird und somit andere Daten dort gespeichert werden dürfen [Fox09]. Ein sicheres Löschen wie Gutmann es beschreibt ist hier jedoch nicht anwendbar, da in einem mobilen Endgerät kein magnetischer Datenträger verbaut ist, sondern ein Flash Datenträger [Hoo12]. Dieser führt grundsätzlich andere Operationen zum Beschreiben, Lesen und Löschen aus. Dafür werden Daten nicht durch magnetische, sondern durch elektrische Ladung dauerhaft gespeichert [Mcm10]. Diesbezüglich liefern die bisher dargelegten Punkte nur das Verständnis für

[12] Mit Ladung ist hier die magnetische Ladung gemeint, die positiv, negativ oder neutral sein kann.
[13] Dazu gehört beispielsweise die Software Tune Up Utilitys. Siehe http://www.tuneup.de/products/tuneup-utilities/features/delete-files-safely/

das Problem im Zusammenhang mit dem sicheren Löschen. Die Aspekte, welche für ein Smartphone wichtig sind sollen im Folgenden klar werden.

2.2.1 Der Löschvorgang auf einem NAND-Datenträger

In aktuellen mobilen Endgeräten sind Datenträger verbaut, welche die sogenannte NAND Technologie beinhalten [Mcm10]. Drei Operationen innerhalb dieser Hardware werden im Folgenden näher beschrieben, um ein Verständnis für die Probleme im Zusammenhang mit dem sichern Löschen auf Smartphones zu bekommen. Das Beschreiben oder Programmieren, das Lesen und das Löschen von Speicherzellen. Dafür ist zunächst der Aufbau einer solchen Zelle wichtig.

Ein NAND-Datenträger besteht aus einer großen Menge von Zellen, die Ladungen über einen langen Zeitraum speichern können [Mcm10]. Abbildung 2-8 zeigt eine solche Zelle.

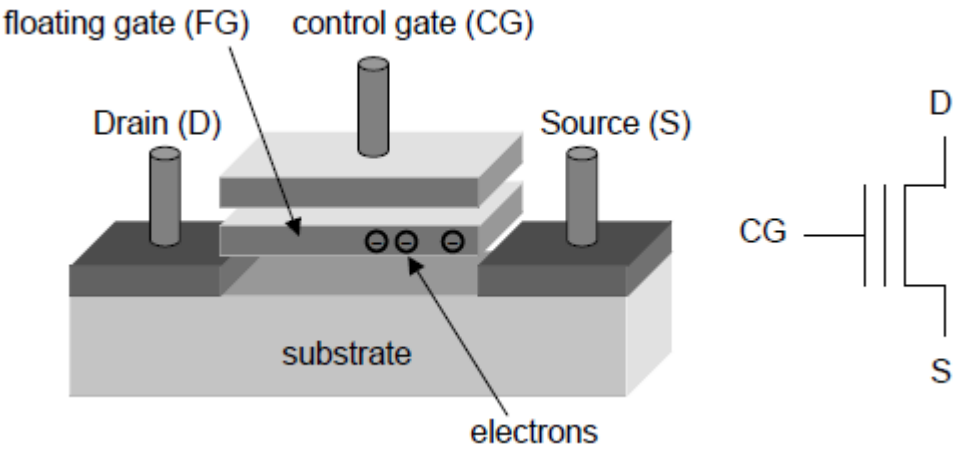


Abbildung 2-8 Der Querschnitt einer Zelle aus einem NAND-Datenträger [Mcm10]

Diese Zelle gehört zu den Bausteinen, die man allgemein als Transistor bezeichnet [Mcm10]. Die Besonderheit dieses Transistors sind zwei sogenannte Gates, wodurch er zur Verwendung in einem persistenten Datenträger geeignet ist. Das floating Gate ist von einer elektrisch isolierenden Schicht umgeben und ist in der Lage Ladungen zu speichern [Mcm10]. Das control Gate liegt darüber und wirkt als Verbindung zu dem isolierten Gate [Mcm10]. Das floating Gate kann durch diese Konstruktion einerseits angesteuert werden und bietet andererseits den darauf geladenen Elektronen keine Möglichkeit abzufließen [Mcm10]. Dadurch hält sie ohne weitere Zufuhr von Energie

über einen langen Zeitraum hinweg den gespeicherten Zustand fest [Mcm10]. Andere Baugruppen, die als sens amp bezeichnet werden übersetzen den Ladungszustand einer Zelle in einen digitalen Wert [Mcm10]. Dafür wird eine Schranke festgelegt ab welcher Ladungsmenge entweder der Wert eins oder der Wert null gilt [Mcm10].

Die Anordnung und Verschaltung dieser Zellen miteinander macht den resultierenden Speicher zu einem NAND-Datenträger [Mcm10]. Abbildung 2-9 zeigt die Verschaltung der Speicher Zellen in einem NAND-Datenträger. Die Zellen bilden in Reihenschaltung einen String [Mcm10], im Folgenden als Spalte bezeichnet. Die einzelnen Zellen einer Spalte werden jeweils parallel mit den Zellen der benachbarten Spalten in einer Wordline angesteuert [Mcm10], im Folgenden als Zeile bezeichnet. Eine Zeile wird im Zusammenhang mit NAND-Datenträgern allgemein als Seite bezeichnet [Hwa13], wobei mehrere dieser Seiten einen Block bilden [Mcm10]. Die Zeilen sind außerdem mit der sogenannten Bitline verbunden, die beim Schreiben und Lesen den „Wert“ der angesteuerten Zellen führt [Mcm10].

Soll nun ein Wert geschrieben oder gelesen werden, muss eine Spannung an eine Zelle angelegt werde, um diese zu markieren [Mcm10]. Da die Zellen jedoch Spaltenweise verbunden sind, erhält entweder die gesamte Spalte eine Markierung oder gar keine Zelle. Der Wert der angelegten Spannung bestimmt dabei, ob die Zellen einer Spalte übergangen werden sollen oder betroffen sind [Mcm10]. So wird eindeutig bestimmt, welche Seite eines Blocks für die jeweilige Operation bestimmt ist. Aus diesem Grund kann jeweils nur eine komplette Seite eines Blockes gelesen oder beschrieben werden. Des Weiteren ist es durch die Schreib-Operation nur möglich den Wert der Zellen von eins auf null zu ändern nicht umgekehrt [Hoo12] [Mcm10] [Sub09]. Um eine Änderung von null zu eins zu leisten muss die Lösch-Operation durchgeführt werden.

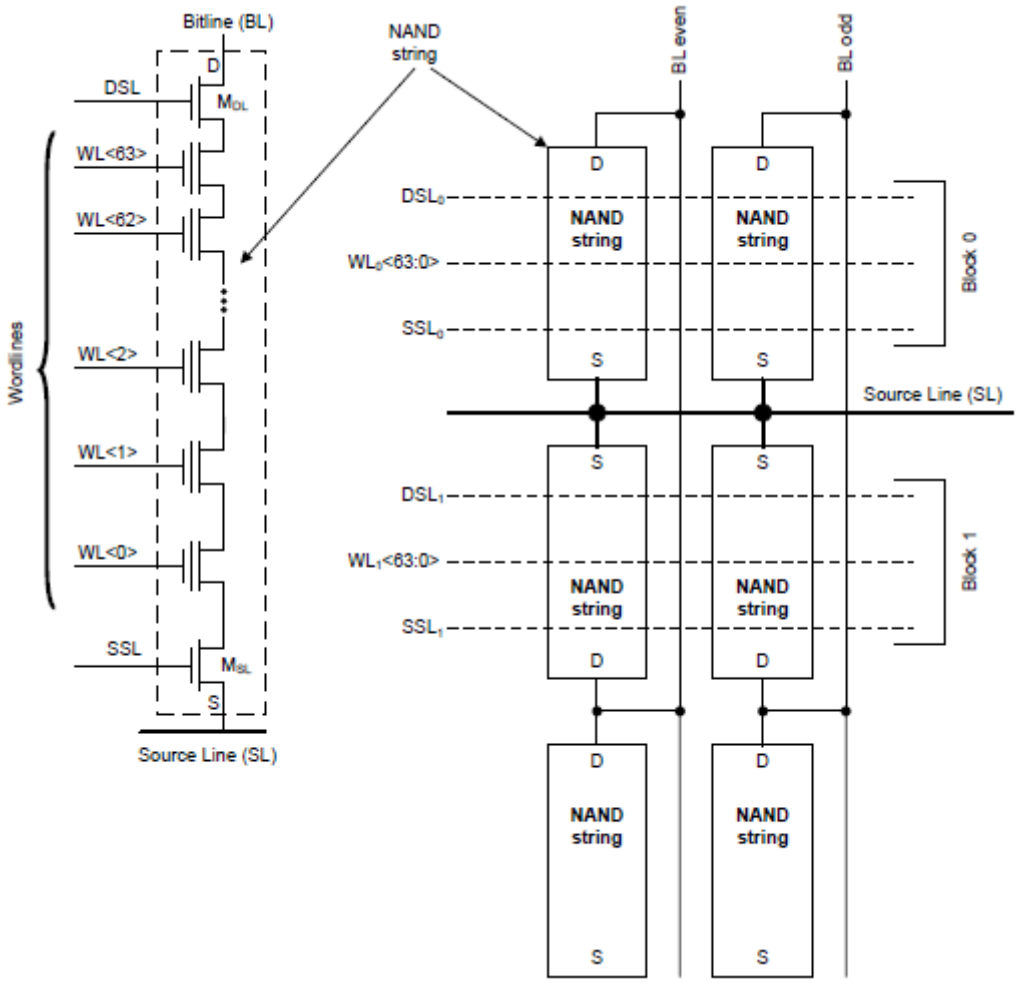


Abbildung 2-9 Innerer Aufbau eines NAND-Datenträgers [Mcm10]

Die Zellen eines NAND Bausteines sind so konzipiert worden, dass sie ihre Ladung über mehrere Jahre ohne Versorgungsspannung halten können [Mcm10]. Deswegen bedarf es auch beim Setzen dieser Ladung eines größeren Aufwands [Mcm10]. Um den Schutz, der die Zelle vor dem Verlust ihrer Ladung schützen soll, zu überwinden ist es nötig eine hohe Spannung anzulegen [Mcm10]. Die Ansteuerung mit einer solch hohen Spannung ist nicht über die Steueranschlüsse der Zeilen, wie oben beschrieben möglich, sondern erfolgt auf anderem Wege auf alle Zellen eines Blocks. Somit kann die Ladung aller Zellen „gefüllt“ werden und führt dazu, dass der gesamte Block mit einsen „beschrieben“ ist.

Wie nun gezeigt wurde, unterliegen das Schreiben und das Löschen einer unterschiedlichen Größenordnung. Dies führt zu einem Problem im Zusammenhang mit dem sichern Löschen, das wie in [Rbc13] beschrieben mit dem Löschen von Daten auf einer Compact Disk verglichen werden kann. Befinden sich sowohl zu löschende und noch benötigte Daten auf einer CD, müssen die benötigten Daten anderweitig

gesichert und die gesamte CD zerstört werden. Im übertragenen Sinne entspricht die CD einem Block im NAND-Datenträger. Befinden sich zu löschende Daten auf einer Seite, muss der entsprechende gesamte Block gelöscht werden. Allerdings ist zu beachten, dass sich keine noch benötigten Daten auf anderen Seiten des gleichen Blocks befinden. Diese Eigenschaft erschwert das sichere Löschen auf einem NAND-Datenträger. Durch das Speichern von noch benötigten Daten eines zu löschenden Blocks entsteht eine Verteilung von Daten. Beim Löschen solcher verteilten Daten müssen alle Fragmente berücksichtigt werden. Dieser Punkt wird noch genauer im Kapitel 4.2.1 beschrieben.

Wie beim Löschvorgang auf magnetischen Datenträgern wird auch auf Smartphones mit NAND-Datenträgern, beim Löschen einer Datei nicht sofort sicher gelöscht. Stattdessen wird beim Löschen einer Datei, was einer Änderung entspricht, der entsprechende Bereich als invalid bezeichnet und somit zum Löschen freigegeben [Ilh12]. Des Weiteren kommt hinzu, dass beim Ändern einer Datei der entsprechende Speicherbereich auf dem NAND-Datenträger nicht überschrieben wird. Dies ist wie oben gezeigt wurde physisch gar nicht möglich, da vor dem Beschreiben (Einsen in Nullen ändern) zunächst eine Löschoperation durchgeführt werden muss (Nullen in Einsen ändern). Somit wird beim Ändern einfach ein freier Block mit den geänderten Daten beschrieben. Die alte Version verbleibt auf dem Datenträger und kann wiederhergestellt werden sollange sie nicht überschreiben wurde. Eine Verbindung der oben genannten Löschverfahren für magnetische Datenträger und NAND-Datenträgern ist wie in [Ywd13] gezeigt möglich, siehe dazu Kapitel 3.1 im Abschnitt Löschen einer Datei.

Das Löschen eines Blockes auf einem NAND-Datenträger reicht aus, um ein Wiederherstellen der dort gesicherten Daten auszuschließen. Jedoch muss sichergestellt sein, dass alle Kopien der zu löschenden Daten berücksichtigt werden.

2.2.2 Behandlung von Bad Blocks

Da Zellen in einem NAND-Datenträger nur eine gewissen Anzahl von Schreib und Löschoperationen leisten können, bevor sie nicht mehr korrekt funktionieren, versehen Hersteller ihre Produkte mit überschüssigen Blöcken [Mcm10]. Im Falle eines Ausfalls von Zellen kann der betroffene Block durch einen der überschüssigen Blöcke ersetzt

werden. Dies geschieht mittels Bad Block Management (BBM). Wie in Abbildung 2-10 zu sehen ist, referenziert ein logischer Block einen physischen. Bei Bedarf wird diese Referenz auf einen reservierten Block geändert [Mcm10]. Das BBM prüft nicht ob ein Block fehlerhaft ist sondern tauscht ihn nach einer statistisch sinnvollen Anzahl an schreib und löschzugriffe aus [Mcm10]. Somit ist klar, dass ein ausgemusterter Block noch lesbar sein kann. Daten die darauf gespeichert wurden, können eventuell ausgelesen werden.

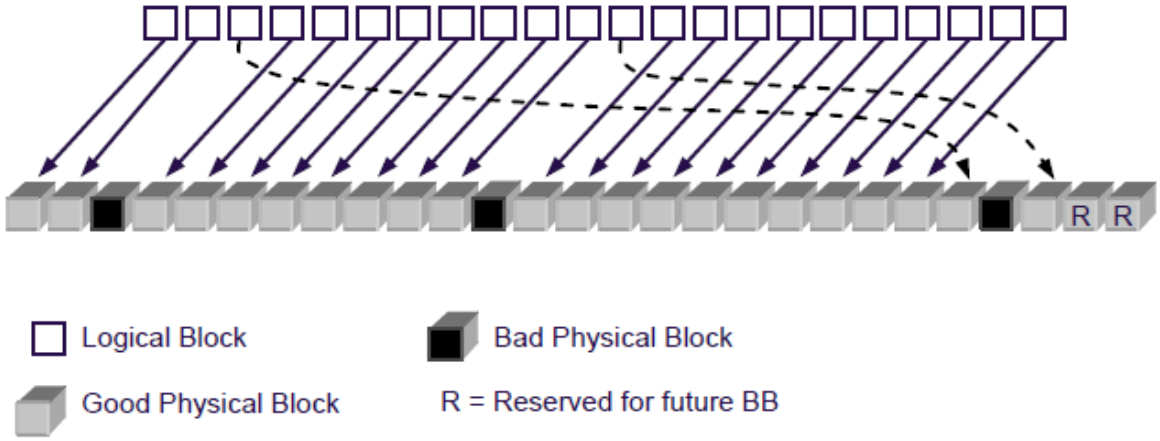


Abbildung 2-10 Bad Block Management

2.3 Mobilfunk-Forensik

Unter dem Begriff Forensik versteht man das Nachvollziehen eines Tatherganges auf der Grundlage von ermittelten Spuren [Car05]. Der Tathergang muss dabei dem Auftraggeber der Untersuchungen, unter Verwendung anerkannter Methoden bewiesen werden [Car05]. Eine Disziplin der Forensik ist die Digitale oder Computer Forensik. Diese wiederum ist ein Überbegriff, unter den die Mobilfunk Forensik fällt. In dieser Arbeit ist es wichtig, das sichere Löschen von Benutzerdaten nachzuvollziehen. Die Techniken der mobilfunk Forensik bieten dafür die passenden Werkzeuge. Deswegen wird hier nur der Teil dieser Disziplin behandelt, bei dem es um die Sicherung von Daten geht.

In diesem Abschnitt sollen die Methoden beschrieben werden, die es möglich machen, Daten von einem mobilen Endgerät sicher zu kopieren, um eine Untersuchung dieser Daten durchführen zu können. Mit sicher ist in diesem Falle möglichst unverändert gemeint. Dies ist ein Leitprinzip der Forensik und erschwert die Arbeit im Allgemeinen [Hoo12]. Es ist jedoch meist unumgänglich eine Änderung der

Daten auf einem Mobilen Endgerät vorzunehmen bevor es überhaupt möglich ist Inhalten zu kopieren [Hoo12].

Wie Andrew Hoog in [Hoo12] beschreibt, unterscheidet man bei den Techniken der Datensicherung zwischen logischen und physischen Zugriffen. Wobei Logische Zugriffe bezogen auf eine spezielle Datei unter der Verwendung des zugrunde liegenden Dateisystems abzielen. Physische Zugriffe hingegen haben einen kompletten Datenträger als Ziel und sind unabhängig von der Beschaffenheit des Dateisystems. Beide Varianten werden Bestandteil dieser Arbeit sein.

Obwohl hier Techniken der Forensik verwendet werden, ist es nicht das Ziel einen Tathergang (Das Vorgehen eines Nutzers beim Erzeugen von Daten) zu rekonstruieren. Vielmehr sollen diese genutzt werden, um Spuren des Nutzers zu entfernen beziehungsweise zu prüfen, ob diese erfolgreich entfernt wurden. Deswegen werden weitere Regeln die hier noch nicht erwähnt wurden außer Acht gelassen und auf tiefergehende Literatur aus diesem Gebiet verwiesen.

2.3.1 Techniken der Mobilfunk-Forensik

Um eine forensische Analyse durchführen zu können ist es zunächst wichtig einen Rechner als sogenannte Workstation zu konfigurieren, um gesicherte Daten darauf zu speichern und analysieren zu können [Hoo12]. Dazu wurde für diese Arbeit eine Ubuntu 12.10 Workstation konfiguriert und mit den folgenden Tools bestückt.

- Sleuthkit
- SQLite Browser
- Android SDK
- Scalpel
- Strings
- Okteta

Sleuthkit ist ein bekanntes Tool in der digitalen Forensik und enthält eine Vielzahl nützlicher Eigenschaften zum Analysieren von gespeicherten Daten [Car05].

Der SQLite Browser machte es möglich, Daten aus einer Datenbank wie in 2.1.2 beschrieben über eine grafische Oberfläche zu durchsuchen.

Das Android SDK ist die Entwicklungsoftware von Android und bietet neben der API auch das Tool adb, die Android Debug Bridge. Damit ist es mögliche über eine Workstation mit einem Android-Endgerät kommunizieren zu können [Mei12].

Scalpel ist ein File Carving Tool [Hoo12]. Mit solch einem Werkzeug ist es möglich, innerhalb eines festgelegten Datenbereiches nach Dateien zu suchen und diese extern zu speichern [Hoo12]. Jedes Dateiformat besitzt eine bestimmte Zeichenfolge, die es identifiziert [Hoo12]. Nach diesen Zeichenfolgen sucht ein File Carving Tool und liest eine zuvor bestimmte Menge an Daten ab dieser Identifizierungsmarke aus [Hoo12]. So ist es möglich, Daten zu finden, die durch das Löschen des Verweises innerhalb des Dateisystems „gelöscht“ worden sind.

Die Software Strings ermöglicht es Zeichenketten eines festgelegten Speicherbereiches auszulesen und in einer Datei zu sichern [Hoo12].

Okteta ist ein Hex Editor, der es ermöglicht den Code einer Datei oder eines bestimmten Speicherbereiches anzuzeigen und zu verändern [Hoo12].

Des Weiteren wurden folgende Anwendungen verwendet, die bereits auf dem Ubuntu beziehungsweise auf dem Android System des Endgerätes vorinstalliert sind:

- dd
- su
- df
- mount

dd bietet die Möglichkeit, exakte Kopien von Datenträgern zu erstellen [Hoo12]. In dieser Arbeit wird die Version verwendet, welche sich auf dem Betriebssystem des jeweiligen Endgerätes befindet nicht die auf der Workstation.

df bietet einen Überblick des genutzten Speichers aller Datenträger eines Systems [Hoo12]. Darüber hinaus zeigt es die Größe der jeweiligen Datenträger an.

Der Befehl mount erzeugt eine Liste aller eingehängten Dateisysteme innerhalb eines Betriebssystems. Dabei werden sowohl die physisch auf dem Datenträger gespeicherten Dateisysteme ausgeben, als auch virtuelle, die beispielsweise auf den Hauptspeicher geschrieben werden [Hoo12].

Auf dieser Grundlage ist es nun möglich Partitionen eines Endgerätes wie sie in 2.1.1 aufgeführt sind auf einen Rechner zu laden und zu analysieren. Diese

Möglichkeit wird im Kapitel 5, wenn es darum geht unterschiedliche Löschstrategien zu untersuchen, zum Tragen kommen.

Um Daten wie sie auf einem Android Smartphone gesichert sind untersuchen zu können muss eine Lösung für das Extrahieren dieser Daten auf eine Workstation gewählt werden. Für diese Arbeit wurde stets das extrahieren auf eine in das Endgerät eingelegt microSD Karte gewählt und somit ein exaktes Abbild von Endgerätpartitionen nach folgender Vorgehensweise erzeugt. Eine microSD Karte wurde mit dem Linux Dateisystem EXT 4 formatiert und in das Gerät eingelegt. Über ADB, wurde eine Verbindung von der Workstation zum Endgerät hergestellt[14], um folgende Befehle auszuführen und somit eine Kopie von einer Partition auf der eingelegten SD Karte zu erhalten.

1. **adb shell** → Starten je eines Daemons auf Rechner und Endgerät [Hoo12]
2. **su** → Erlangen von Superuser Rechten
3. **cd dev/block/vold** → Navigation zu der SD Karten-Schnittstelle
4. **mount –t ext4** ***SDKartenBezeichner*** ***PfadDerSDKarte***
   → Einhängen der mit EXT4 formatierten microSD Karte
5. **cd dev/block** → Navigation zu der Partitions-Schnittstelle
6. **dd if=mmcblk0p12 of=** ***PfadDerSDKarte*****/mmcblk0p12.dd bs=4096**
   → Kopieren der gesamten Partition auf die SD Karte

Da eine SD Karte im normalen Betrieb beim Samsung Galaxy S2 und S3 mit dem Dateisystem FAT32 formatiert ist, kann das System sie unter EXT4 nicht selbstständig einhängen. Somit wird dieser Schritt manuell in Zeile 4 durchgeführt. Der Grund für diesen Umstand liegt in der maximalen Dateigröße bei den zwei genannten Dateisystemen. FAT32 unterstützt eine maximale Dateigröße von circa 4 Gigabyte [Car05]. Mit EXT4 ist es möglich Dateien mit eine Größe von 2 Terabyte zu speichern [Mat07]. Wie in 2.1.1 gezeigt ist die data Partition des Galaxy S3 circa 12,3 GB groß. Somit kann mit dieser hier angewandten Technik eine Teilung der Partition in vier Teile und eine anschließende Zusammenführung dieser Teile vermieden werde.

In einem zweiten Schritt kann das so erzeugte Image entweder in die Workstation eingebunden oder mittels File Carving untersucht werde. Ersteres macht Sinn, wenn

[14] Dafür ist es nötig die Option USB-Debugging auf dem Smartphone zu aktivieren [Hoo12]

nach Benutzerdaten in einem nicht gelöschten System gesucht wird. Im Falle von veränderten Metadaten durch Löschvorgänge macht der zweite Punkt Sinn. Es müssen nämlich solche Daten gefunden werden, deren Verbleib nicht mehr durch das System zurückzuverfolgen ist, jedoch noch auf dem internen Speicher gesichert sind. Dieser Vorgang kann mittels den oben erwähnten Tools Scalpel oder Strings vorgenommen werden.

2.4 Zusammenfassung

Dieses Kapitel führte die drei Teilgebiete, dieser Ausarbeitung ein. Im Abschnitt über Benutzerdaten unter Android wurde beschrieben was unter Benutzerdaten zu verstehen ist und wo diese auf einem Android Smartphone zu finden sind.

Das Löschen wurde über bekannte Löschalgorithmen für magnetische Datenträger eingeführt. Anschließend wurde auf die hier wichtigen NAND-Datenträger eingegangen. Die Funktionsweise dieser Datenträger wurde beschrieben und liefert die Grundlage für das Verständnis in den folgenden Kapiteln.

Zuletzt wurde beschrieben inwiefern die Mobilfunk-Forensik für das erlangen der Ziele dieser Arbeit wichtig ist. Die Methoden, um Benutzerdaten von einem Endgerät zu laden, wurden dabei hervorgeheben. Die Untersuchungen in Kapitel 5 werden es erforderlich machen Techniken der Mobilfunk-Forensik anzuwenden, um unterschiedliche Lösch-Strategien, welche in Kapitel 4.3 aufgeführt werden zu evaluieren.

3 Verwandte Arbeiten

In der Vergangenheit beschäftigten sich andere Arbeiten ebenfalls mit dem Thema sicheres Löschen auf Flash-Datenträgern. Es besteht ein grundlegender Unterschied zwischen Flash- und Hard Disk Datenträgern, welcher bereits im Kapitel 2.2 beschrieben wurde. Dadurch ist es nötig andere Lösungen zum sicheren Löschen anzuwenden. Fünf Arbeiten auf diesem Gebiet, werden in diesem Kapitel thematisiert, um mögliche Ansätze unter den Bedingungen eines NAND-Datenträgers zu veranschaulichen.

3.1 Sicheres Löschen

In [Scl08] wurden bereits 2008 zwei generelle Möglichkeiten zum Löschen von Daten auf NAND-Datenträgern vorgestellt. Diese sind zum einen das sogenannte Block Cleaning, bei dem das aus Kapitel 2.2.1 bekannte Löschen eines Blockes durchgeführt wird. Zum anderen das Zero Overwriting, bei dem alle Daten eines Speicherbereiches mit Nullen überschrieben werden. Die zweite Methode ist allerdings nicht ohne Weiteres umsetzbar. Hier wird ein bereits beschriebener Block, der als gelöscht markiert wurde, mit Nullen überschrieben [Scl08]. Wie im Kapitel 2.2.1 beschreiben, ist beim Schreiben von Seiten, eine Änderung der Ladung in den Speicherzellen nur in eine Richtung möglich. Das heißt der Wert einer Speicherzelle kann von einer Eins zu einer Null geändert werden, nicht umgekehrt. Nicht bei jedem Datenträger ist es möglich eine bereits beschriebe Seite zu beschreiben [Rbc13].

Diese beiden Methoden werden in [Scl08] bezüglich ihren Vor- und Nachteilen verglichen. Beim Löschen von kleinen Speicherbereichen in der Größenordnung einer Seite ist es effizienter das Zero Overwriting zu wählen [Scl08]. Beim Löschen größerer Bereiche empfiehlt es sich die Variante des Block Cleaning zu wählen [Scl08]. Beim Löschen mittels Block Cleaning muss jedoch beachtet werden, das sich im zu löschenden Block Daten befinden können, die nicht gelöscht werden sollen. Diese müssen vor dem Löschen auf einer andere Stelle gesichert werden, bevor das Block Cleaning durchgeführt werden kann [Scl08].

Shuba präsentierte 2009 eine Lösung, welche Dateien nicht direkt löscht, jedoch unlesbar macht [Sub09]. In jedem Block auf einem NAND-Datenträger befinden sich

einige Bits zur fehlerkorrektur [Sub09]. Dieser Bereich wird als Error Correcting Code (ECC) bezeichnet. Er umfasst einen redundanten Anteil an Daten, der dafür genutzt wird, Fehler im entsprechenden Block zu korrigieren, falls Solche auftreten [Sub09]. Um Dateien im entsprechenden Block unlesbar zu machen, werden drei Bits im ECC Speicherbereich gekippt [Sub09]. Des Weiteren werden per Zufall weitere drei Bits im Speicherbereich des Blocks gekippt [Sub09]. Durch die zweite Änderung entsteht ein Fehler, der mittels ECC korrigiert werden soll. Durch die Änderung im ECC ist das allerdings nicht möglich. Somit sind die Daten nicht mehr lesbar und können als gelöscht bezeichnet werden [Sub09]. Shuba gibt eine Performanzsteigerung von 92.88% im Vergleich zu Algorithmen unter der Methoden des Zero Overwriting und eine Steigerung von 92,98% im Vergleich zur Block Cleaning Methode an.

In [Ilh12] wurde 2012 eine Lösung für das Problem von nicht überschreibbaren Seiten geliefert. Damit ist das sogenannte out-of-place Update gemeint. Ein Flash-Datenträger überschreibt bei einer Änderung von Daten diese nicht, sonder erzeugt an einer anderen Stelle des Speichers eine neue Version der Daten [Ilh12]. Mittels Änderungen am internen Buffer und einer Manipulation im Flash Translation Layer wird in [Ilh12] ein sicheres Löschen gewährleistet.

Dabei wird beim Übergang von logischen Blöcken zu physischen Blöcken im internen Buffer ein Zähler integriert [Ilh12]. Dieser zählt, wie oft jede Seite geändert wurde. Ab einem bestimmten Wert werden alle alten Versionen einer Seite gelöscht [Ilh12]. Um dies umzusetzen, wird in der FTL[15] eine Methode implementiert, die nach allen gezählten Versionen sucht und diese mittels Block Cleaning löscht [Ilh12].

In [Ywd13] wurde 2013 eine Lösung geliefert, wie durch einfaches Überschreiben ein sicheres Löschen möglich ist. Diese Arbeit bezieht sich auf SSD Datenträger, welche ebenfalls mit NAND-Speicher versehen sind [Ywd13]. Durch eine Änderung der Treiber Software wird erreicht, dass geänderte Seiten nicht wie üblich ignoriert, sondern mit Nullen überschrieben werden [Ywd13]. Dadurch wird bei jeder Änderung einer Seite vermieden, dass eine alte Version dieser Seite auf dem Speicher verbleibt. Durch diese Manipulation der Charakteristik eines NAND-Datenträgers kann mittels

[15] Flash Translation Layer

Software, die zum Löschen auf magnetischen Datenträgern konzipiert wurde, ein sicheres Löschen auf NAND-Datenträgern gewährleistet werden [Ywd13].

Reardon, Basin und Capkun lieferten 2013 ein System of Knowledge in dem sie Arbeiten auf dem Gebiet des sicheren Löschens beschreiben [Rbc13]. Diese werden dabei nach dem Interface und dem Datenträgertyp sortiert dargestellt. Ein Abschnitt beschäftigt sich mit dem sicheren Löschen auf Benutzerebene und liefert dabei fünf Ansätze zum sicheren Löschen. Davon können zwei als sicher für die Voraussetzungen in einem NAND-Datenträger betrachtet werden [Rbc13]. Diese entsprechen den folgenden Punkten.

Das *secure erase* Verfahren umfasst eine Software, die über die Kommunikation mit dem Hardware-Controller ein sicheres Löschen aller Daten auslöst [Rbc13]. Auf einer tieferen Ebene ist also eine Möglichkeit gegeben alle Daten auf dem physischen Datenträger zu löschen, welche von der Benutzerschicht aus initiiert wird [Rbc13]. Vergleichbar ist dies mit dem Ausführen eines Werksrests unter Android.

Durch das *freespace filling*, wird der nicht belegte Speicherbereich beschrieben, um die darauf unter Umständen noch gesicherten Daten zu löschen. Da bei NAND-Datenträgern dabei ein hoher Aufwand durch Wear Leveling entsteht, existieren Ansätze, die darauf abzielen den freien Speicherbereich permanent zu warten. Hierbei wird nur eine bestimmte Menge an gelöschten Daten zugelassen.

3.2 Zusammenfassung

Zusammenfassend können folgende Schlüsse, aus den in diesem Kapitel vorgestellten Arbeiten, für diese Arbeit gezogen werden. Es existieren zwei grundsätzliche Möglichkeiten Daten auf einem NAND-Datenträger zu entfernen, das Löschen von Blöcken und das überschreiben von Datenbereichen mit Nullen.

Es wurden einige Lösungen vorgestellt, die es ermöglichen auf tieferen als der Benutzerebene ein sicheres Löschen auf Flash-Datenträgern umzusetzen. Eine Lösung, welche mittles Androidapplikation umgesetzt wird erfordert ein Konzept, das an die möglichen Ansätze für NAND-Datenträger aus [Rbc13] anknüpft. Im Kapitel 5.4 wird ein Konzept vorgestellt, welches das Prinzip des freespace filling für das Android System adaptiert und somit eine Lösung zum sicheren Löschen von Benutzerdaten liefert.

4 Lebenszyklus von Benutzerdaten

Die im Kapitel 2.1 beschriebenen Grundlagen zur Art und dem Ort der Sicherung von Benutzerdaten, sollen im Folgenden verfeinert werden. Es wurde bereits zwischen Benutzerdaten innerhalb einer Applikation und solchen unterschieden, die direkt vom Nutzer auf das Endgerät geladen werden. Diese Unterscheidung soll im ersten Abschnitt dieses Kapitels genauer behandelt werden. Anschließend wird die Verteilung solcher Daten beschrieben. Da sich die Daten des Nutzers auf einem mobilen Endgerät ständig ändern, das heißt, sich vermehren, gelöscht oder verschoben werden, kommt es zu einer Fragmentierung. Da, wie in Kapitel 2.2.1 beschrieben, die Speicherzellen eines NAND-Datenträgers nur eine begrenzte Anzahl von Schaltvorgängen überstehen, ist es nötig, die Nutzung aller Speicherzellen gleichmäßig zu verteilen. Dadurch entsteht ebenfalls eine Fragmentierung. So kann sich eine Datei vom Zeitpunkt ihrer Erstellung bis sie gelöscht wird, auf unterschiedlichen Bereichen des Speichers verteilt haben. Im letzten Abschnitt werden die Strategien des Android Systems thematisiert, mit denen Benutzerdaten entfernt werden können.

4.1 Datenentstehung

Der erste Schritt im Lebenszyklus von Benutzerdaten ist die Entstehung. Zu den Methoden der Sicherung von Benutzerdaten aus Kapitel 2.1.2 lassen sich zwei Möglichkeiten für ihren Ursprung definieren. Zum einen kann eine Applikation, welche standardmäßig zum System gehört oder eine die vom Benutzer installiert worden ist, die Wurzel von Benutzerdaten sein. Zum anderen kann der, bei Android allgemein als Media bezeichnete Bereich, der Ursprung für, auf dem gesamten Speicher verteilte, Daten des Nutzers sein. Dieser Bereich wurde bereits in Kapitel 2.1.2 als emulierter externer Speicher beschrieben.

4.1.1 Installieren von Applikationen

Der standardmäßige Weg eine Applikation auf einem Android-Endgerät zu installieren läuft über die Software Google Play Store. Das ist eine Applikation, die den Zugang zu

*.apk Dateien von registrierten Android Entwicklern darstellt [Hoo12]. Eine weitere Möglichkeit besteht darin, eine *.apk Datei über die Android Debug Bridge auf dem Endgerät zu speichern und von dort aus mittels der adb-Anwendung „install“ zu installieren. Die Dateiendung apk weist auf eine gepackte Android Applikation hin. Hierin liegen alle Inhalte einer Applikation gebündelt in einer Datei vor [Ado13]. Im Werkszustand sind bereits Applikationen installiert, welche durch Android oder durch den Hersteller des Endgerätes bestimmt wurden. Diese werden als native Apps bezeichnet [Mei12], ein Beispiel dafür ist die App Google Play Store. Solche Applikationen werden im Falle eines Zurücksetzens neuinstalliert und können nicht durch den Nutzer deinstalliert werden. Wie in Abbildung 4-1 zu sehen ist, kann diese native App nicht deinstalliert werden. Stattdessen kann sie auf eine ältere Version zurückgesetzt werden.

Vor der Installation einer App legt das System fest, wo diese installiert werden soll. Seit API 8 [16] ist es möglich innerhalb der Rechtevergabe einer Applikation zu bestimmen, wo diese Installiert werden soll [Ado13]. Das sogenannte manifest Attribut, welches dafür verantwortlich ist, heißt android:installLocation [Ado13]. Der Autor einer App wählt in diesem Punkt entweder den Wert „preferExternal“ oder „auto“. Wird keine der Optionen gesetzt, installiert das System alle Daten auf dem internen Speicher, da ohne dieses Attribut, keine Rechte für den externen Speicher vergeben werden können [Ade13]. Ist der interne Speicher voll kann die App nicht installiert werden.

Die Option preferExternal führt dazu, bei der Installation möglichst auf dem externen Speicher zu installieren [Ade13]. Besitzt dieser keinen freien Speicher, wird intern oder überhaupt nicht installiert [Ade13]. Unter auto bestimmt das System, wo die App installiert wird [Ade13]. Der Benutzer hat dann die Möglichkeit zu einem Zeitpunkt nach der Installation den Ort zu ändern und somit Daten vom internen auf den externen Speicher oder umgekehrt zu verschieben (siehe Abbildung 4-1). Die App com.android.settings, aus dem Android Source Code, beinhaltet diese Optionen. Jedoch unterstützt nicht jedes Endgerät alle zur Verfügung gestellten Optionen. Das Samsung Galaxy S2 verfügt beispielsweise darüber, das Samsung Galaxy S3 nicht. Dieses Faktum könnte im Zusammenhang mit der Kapazität des internen Speichers liegen. Das S2 verfügt dabei über circa zwei Gigabyte, wohingegen das S3 sechszehn

[16] Android Version 2.

Gigabyte bereitstellt. Da die SD-Karte bereits in Kapitel 2 als irrelevant für diese Arbeit ausgegeben wurde, wird nicht weiter auf diese Option eingegangen.

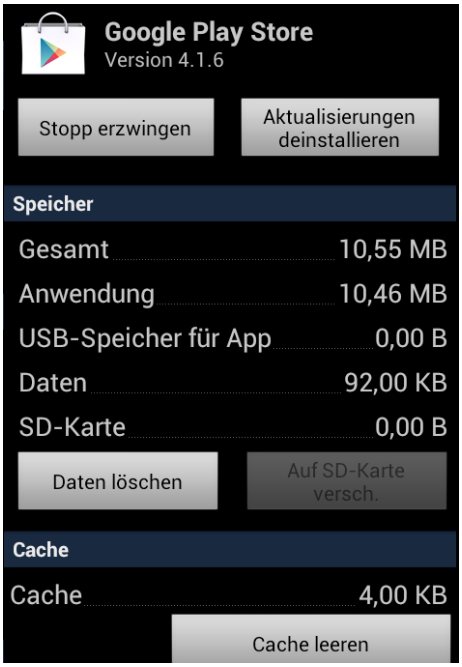


Abbildung 4-1 Optionen für die App Google Play Store im Anwendungs-Manager innerhalb der com.android.settings Applikation

Unabhängig von den soeben genannten Optionen zur Wahl das Installationsortes der Applikation werden die Benutzerdaten stets auf dem internen Speicher gesichert [Ade13]. Damit sind die in Kapitel 2.1.2 beschriebenen Varianten von Daten, SQlite Datenbanken, Schlüsselpaaren oder die direkte Sicherung von Daten, wie in Form des Caches, gemeint. Nur die apk selbst, wird eventuell extern gesichert, um bei großen Applikationen ein Überlauf des internen Speichers zu verhindern.

4.1.2 Speichern von Dateien

Der Ursprung, der von dem Nutzer direkt gespeicherten Dateien ist der externe Speicher. Dieser kann sowohl der Speicherbereich auf einer eingebundenen microSD Karte sein, als auch als emulierter externer Speicher im internen Speicher liegen. Wie der externe Speicher organisiert ist, kann von einem zum nächsten Endgerät sehr unterschiedlich sein. Selbst wenn es Endgeräte vom gleichen Hersteller sind. Betrachtet man die Umsetzung dieses Bereiches beim Samsung Galaxy S2 im Vergleich zum Galaxy S3 wird Folgendes klar. Beim S2 existiert eine eigene Partition

für den emulierten externen Speicher mit einer Größe von elf Gigabyte. Im Gegensatz dazu ist beim S3 der emulierte externe Speicher variabel und greift je nach Bedarf auf die Speicherkapazität des internen Speichers zu. Der externe Speicher entspricht hingegen bei beiden Modellen der Kapazität der eingelegter microSD Karte.

Auf diese Speicherbereiche kann mittels USB Verbindung zu einem Rechner zugergriffen werden. In diesem Fall werden beide in das System des verbundenen Rechners eingehängt. Der Lebenzyklus von Benutzerdaten in Form einer Datei beginnt somit mit dem Kopieren einer solchen auf diesem Wege.

Ein weiterer Weg, auf den gleichen Bereich Dateien abzulegen, besteht darin, diese über ein Netzwerk auf das Endgerät zu sichern. Unter der Verwendung von Applikationen können ebenfalls Benutzerdaten dieser Art entstehen. Beispielsweise beim Herunterladen einer Datei über einen Browser. In diesem Fall wird die geholte Datei nicht im internen Applikationsverzeichnis des Browsers gespeichert. Stattdessen wählt der Benutzer einen Ort im externen Speicher oder im emulierten externen Speicher.

4.2 Verteilung von Daten

Im Laufe der Zeit verändert sich sowohl die Struktur als auch die Menge der Daten auf einem Endgerät. Benutzerdaten, werden an unterschiedlichen Orten auf dem Datenträger abgelegt. Das stellt ein Problem dar, wenn man beim Löschen verteilter Daten sicher gehen will, dass kein Fragment unangetastet bleibt.

Eine Fragmentierung könnte beispielsweise bei einer SQLite Datenbankdatei auftreten. Die Applikation com.android.email, welche in der Android Version 4 standardmäßig installiert ist, beinhaltet zu jeder Zeit mindestens zwei solcher Dateien. Diese stellen vor der ersten Verwendung der Applikation zwei unbefüllte Datenbanken mit jeweils mehreren Tabellen dar. Die Dateigrößen liegen bei 233472 Byte und 12288 Byte. Diese Größe kann bereits eine Fragmentierung dieser Dateien erzeugen. Die Blöcke im verwendeten Dateisystem EXT4, in denen Dateien gespeichert werden, umfassen eine Größe von 4096 Byte [Mat07]. Somit ist es nötig, die oben erwähnte Datenbankdatei in einem Speicherbereich zu sichern, der aufgerundet über 233472/4096 freie Blöcke hintereinander verfügt. Anderenfalls entsteht eine Fragmentierung der Datei. Durch die Nutzung der Applikation entstehen Benutzerdaten, die zum Teil in den genannten Datenbanken gesichert werden.

Dadurch wächst der Speicherbedarf der SQLite Dateien und eine Fragmentierung wird wahrscheinlicher.

Ein Beispiel aus den Untersuchungen von Kapitel 5.2.1 zeigt, dass bereits kleine Dateien fragmentiert werden (siehe Abbildung 4-2). Der grün markierte Bereich weist eine Dateigröße [Ext13] von 53248 Byte aus. Somit werden dreizehn Blöcke beschrieben, um diese Datei speichern zu können. Die blau markierten Bereiche zeigen wie diese Datei gespeichert wurde. Die vierte, fünfte und sechste Zeile der hexadezimalen Ausgabe des Inodes gibt, an den markierten Stellen, jeweils die Metadaten eines Fragmentes der Datei aus [Ext13]. Wobei im ersten Fragment (Zeile vier) drei Blöcke verwendet werden. Im zweiten Fragment (Zeile fünf) ebenfalls. Das letzte Fragment(Zeile sechs) umfasst die übrigen sieben Blöcke.

```
FF 81 68 27    00 D0 00 00    66 B3 B8 51    EA DD C6 51
5B E3 C5 51    00 00 00 00    68 27 01 00    68 00 00 00
80 00 08 00    01 00 00 00    0A F3 03 00    04 00 00 00
00 00 00 00    00 00 00 00    03 00 00 00    7B 06 1E 00
03 00 00 00    03 00 00 00    81 06 1E 00    06 00 00 00
07 00 00 00    69 87 01 00    00 00 00 00    00 00 00 00
00 00 00 00    87 84 8B 2B    00 00 00 00    00 00 00 00
00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
1C 00 00 00    F4 AB 57 77    2C B5 66 D7    A0 2E E3 C5
66 B3 B8 51    A0 2E E3 C5
```

Abbildung 4-2 Inode einer SQLite Datenbankdatei

Die stetige Änderung einer Datenbankdatei muss im Zusammenhang mit dieser Arbeit jedoch nicht permanent verfolgt werden. Der Inode liefert, wie gezeigt, alle Metadaten, die nötig sind, um alle Fragmente einer Datei finden zu können. Somit ist beim Vorgang des sicheren Löschens nur die letzte Dokumentation im entsprechenden Inode zu beachten, wenn es darum geht die Fragmentierung einer Datei zu berücksichtigen. Weitere Probleme, die durch die Umsetzung der Hardware wie im Kapitel 2.2.1 beschrieben auftreten werden im nächsten Kapitel im Bereich des Konzeptes behandelt.

Benutzerdaten, deren Größe statisch ist sind hier genau so zu betrachten, wie die soeben beschriebene dynamische Datei. Darunter fallen beispielsweise Dateien aus dem Cache-Verzeichnis einer Applikation, oder Bilddateien im externen Speicher. Solche Dateien verändern ihren Speicherbedarf zwar nicht, können jedoch ebenfalls

auf unterschiedliche Speicherbereiche auf dem Datenträger verteilt werden. Dabei gilt der gleiche Umstand wie bei der SQLite Datenbankdatei.

4.2.1 Wear Leveling

Eine Speicherzelle wie sie in den Grundlagen beschrieben wurde, unterliegt einer bestimmten Lebensdauer. Der Zyklus von Löschen und Beschreiben kann circa hunderttausendmal durchgeführt werden, bevor eine Zelle und damit der gesamte Block in der diese liegt nicht mehr einwandfrei funktioniert [Mcm10]. Dies gilt für eine Zelle, die ein Bit speichern kann und als Single Level Cell bezeichnet wird [Mcm10]. Bei einer Multi Level Cell, die mehrer Bits speichern kann, geht man nach zehntausend Zyklen davon aus, dass ihr Block ausgetauscht werden muss (siehe Kapitel 2.2.2) [Mcm10].

Auf einem Datenträger befinden sich stets sowohl Daten, die selten verändert oder gelöscht werden als auch Daten, die sehr häufig verändert werden [Cwe12]. Wie in Kapitel 2.2.1 gezeigt wurde, ist für eine Änderung von Daten auf einem NAND-Datenträger das Löschen und Wiederbeschreiben der betroffenen Blöcke nötig. Um nun eine gleichmäßige Verteilung dieser Zyklen für alle Blöcke zu gewährleisten wird das sogenannte Wear Leveling eingesetzt. Dabei existiert eine Schicht[17] zwischen Betriebssystem und Datenträger, die beim Zugriff auf Daten zwischen logischen und physischen Blöcken unterscheidet [Chs12]. Das operierende System greift auf logische Blöcke zu und ein Controller bildet diese auf den physischen Speicherort auf dem Datenträger ab [Mcm10]. Dadurch kann bei häufigem Zugriff auf einen logischen Block, sein Inhalt auf einem anderen physischen Block gesichert werden, um den zuvor gültigen physischen Block zu entlasten. Durch diesen Vorgang entsteht allerdings eine gewisse Redundanz. Daten, die durch die Wear Leveling Methode auf einen anderen physischen Speicher Ort kopiert werden, existieren ebenfalls an dem zuvor genutzten Speicherort [Cwe12]. Diese Tatsache wird ignoriert bis eine gewisse Größe an freiem Speicher unterschritten wird [Cwe12]. Das sogenannte Garbage Collector Modul lokalisiert dann nicht benötigte Blöcke und löscht diese [Mcm10]. Dadurch wird sichergestellt, dass es freie und damit beschreibbare Blöcke gibt.

[17] Diese Schicht wird als Flash Translation Layer bezeichnet und mit FTL abgekürzt.

Die Umsetzung dieser Methode ist nicht in jedem Android Endgerät gleich. Zum einen kann die Abbildung von logischem zu physischem Block sowohl mittels Software Lösung (MTD) als auch über die Hardware mittels Controller im Datenträger Umgesetzt werden[18] [Chs12]. Zum anderen gibt es eine Vielzahl von Algorithmen, die eine möglichst gute Verteilung der Blöcke gewährleisten sollen.
Die Verteilung hinsichtlich des Wear Leveling stellt kein Problem für diese Arbeit dar. Da aus der Sicht des Benutzers stets die gleichen logischen Blöcke angesteuert werden, existiert aus diesem Blickwinkel keine Verschiebung der Daten. Wenn bei den Untersuchungen im Kapitel 5 oder bei einer Operation der Applikation aus Kapitel 6 auf Benutzerdaten zugegriffen wird, geschieht das stets über die FTL [Mcm10]. Der Controller des physischen Datenträgers zeigt dann bei einem Zugriff auf einen logischen Block auf den richtigen physischen Block [Mcm10].

4.3 Strategien zum Entfernen von Benutzerdaten

Wie in Kapitel 2.1.2 beschrieben, gibt es verschiedene Methoden unter Android Benutzerdaten auf einem Smartphone abzulegen. Welche Möglichkeiten unter Android existieren diese nun wieder zu Löschen soll hier thematisiert werden. Dabei ist es möglich, sowohl Teile der erzeugten Daten zu löschen, als auch komplette Verzeichnisse. Angelehnt an die bisher beschriebenen Formen von Benutzerdaten, wie Datensätze (Inhalte innerhalb einer SQLite Datei) und Dateien (eine Bilddatei) existieren in diesem Kapitel entsprechende Abschnitte. Des Weiteren werden die Möglichkeiten thematisiert, welche die Android Applikation „Einstellungen“ zum Löschen von Benutzerdaten bietet. Ein Beispiel zu den jeweiligen Strategien sorgt dabei für ein besseres Verständnis.

4.3.1 Löschen von Datensätzen innerhalb einer Applikation

Innerhalb einer Applikation liefert der Entwickler Möglichkeiten, gesicherte Daten wieder zu löschen. In diesem Fall werden also Inhalte einer Datei entfernt, nicht aber komplette Dateien. Meistens sind SQLite Datenbanken bei dieser Art von

[18] Diese Variante kommt sowohl beim Samsung Galaxy S2 als auch beim Samsung Galaxy S3 zum Einsatz.

Löschvorgang betroffen. Die folgenden Punkte fallen unter die Kategorie Datensätze innerhalb von Applikationen.

- Nachricht in Form einer E-Mail
- Kontaktdaten
- Link innerhalb eines Browsers
- Kalendereinträge
- Anruflisten
- Routen in einer Navigations-App

In der Android-Version 4.1.2 können solche Benutzerdaten innerhalb der standardmäßig installierten Applikationen anfallen. Tabelle 4-1 zeigt, welche Dateien dabei betroffen sind, wo diese gesichert sind und welche Dateitypen ein solcher Löschvorgang betrifft.

Beispiel	Datei	Dateityp	Pfad(data/data…)
E-Mail	EmailProvider.db, EmailProviderBody.db	SQLite DB	com.android.email
Kontakt	contacts2.db	SQLite DB	com.android.providers.contacts
Kalendereintrag	calendar.db	SQLite DB	com.android.providers.calendar

Tabelle 4-1 Benutzerdaten in Form von Datensätzen in Standard Applikationen unter Android 4.1.2

4.3.2 Löschen von Dateien

Das Löschen einer kompletten Datei funktioniert über einen Dateimanager. Unter Android gibt es dafür eine native Applikation, die mit "Eigene Dateien" bezeichnet wird. Damit lassen sich Dateien auf der externen Speicherkarte und auf der emulierten externen Speicherkarte öffnen, verschieben und auch löschen. Es existieren viele weitere Varianten solcher Applikationen, die es erlauben, den kompletten Verzeichnisbaum eines Endgerätes zu durchsuchen. Allerdings haben diese Applikationen ohne Superuser Rechte, das bedeutet bei einem nicht gerooteten Endgerät, ebenfalls nur Zugriff auf die Dateien im externen Speicher. Abbildung 4-3

zeigt den Standard Dateimanager aus der Android-Version 4.1.2, bei dem das Löschen einer Datei vorgenommen wird.

Abbildung 4-3 Das Löschen einer Bilddatei mit dem Android Dateimanager

4.3.3 Löschen über den Android Anwendungs-Manager

Die Applikation „Einstellungen“ ist in jeder Android-Version 4 vorhanden. In der Version 4.1.2 bietet sie eine Reihe von Möglichkeiten, die Funktionen des Endgerätes zu steuern oder Informationen darüber zu erhalten. Hier nur einige der Möglichkeiten, die diese App bietet.

- Wireless LAN aktivieren/deaktivieren, Konfigurieren
- Bluetooth aktivieren/deaktivieren, Konfigurieren
- Speicherplatz Überblick
- Android Version anzeigen
- Identifikationsnummer des Endgerätes anzeigen
- Datennutzung Netz ausgeben
- Home-Bildschirmmodus einstellen
- Ruhemodus aktiviert/deaktiviert
- Ton Einstellungen

- Display Einstellungen
- Akku Informationen
- Energiesparmodus aktiviert/deaktiviert
- Sichern und zurücksetzen
- Konto Verwaltung
- Entwickler-Optionen
- Anwendungs-Manager

Der letzte Punkt der Liste liefert die Möglichkeit Benutzerdaten mithilfe dieser Applikation zu löschen. Um dies durchführen zu können, lassen sich in besagtem Menüpunkt, alle Anwendungen anzeigen, die ausgeführt werden, beziehungsweise installiert sind. Eine Anwendung, die ausgeführt wird lässt sich über den Anwendungs-Manager durch folgende Funktionen beeinflussen. Zunächst einmal kann ein stoppen der Anwendung durchgeführt werden, wodurch diese terminiert wird. Wichtiger in diesem Zusammenhang sind die Funktionen zur Manipulation der Benutzerdaten einer Anwendung. Es ist möglich nur den Cache zu löschen oder alle Daten der Anwendung zu entfernen. Innerhalb des speziellen Anwendungsmenüs wird der aktuelle Speicherbedarf des Caches, der Daten und der Applikation selbst angezeigt (siehe Abbildung 4-1).

4.4 Zusammenfassung

In diesem Kapitel wurden die einzelnen Phasen beschrieben, die Benutzerdaten auf einem Android System vom Installieren bis zum Entfernen durchlaufen. Diese Phasen wurden als Entstehung, als Verteilung und zuletzt als Entfernung bezeichnet. Dabei wurde zwischen Benutzerdaten unterschieden, die durch Applikationen auf den Datenträger gelangen und solchen, die direkt vom Benutzer gesichert werden.

Bei der Verteilung der Daten wurde ein besonderes Augenmerk auf das Prinzip des Wear Leveling gelegt. Mit dem Ergebnis, dass diese Funktion die Aufgaben beim Löschen von Benutzerdaten nicht beeinflussen. Ein wichtiger Aspekt war die Betrachtung der Fragmentierung von Benutzerdaten. Wenn im weiteren Verlauf dieser Arbeit das sichere Löschen von Benutzerdaten durchgeführt oder geprüft werden soll, ist es entscheidend, diesen Punkt zu beachten.

Die Möglichkeiten, welche das Android System bietet, um Daten zu löschen, wurden zuletzt in diesem Kapitel behandelt. In den nun folgenden Untersuchungen, sollen diese auf ihre Sicherheit hin geprüft werden, um eine Grundlage für das angestrebte Konzept zum sicheren Löschen auf der Benutzerebene zu schaffen.

5 Untersuchungen auf mobilen Endgeräten

In diesem Kapitel werden die gezeigten Löschstrategien näher betrachtet, um ihre Sicherheit hinsichtlich einer möglichen Wiederherstellung festzustellen. Dafür werden die Inhalte des Grundlagen Kapitels aufgegriffen. Dies bedeutet, die Techniken der Mobilfunk Forensik zur Sicherstellung von Daten wird angewendet. Außerdem werden nur die Bereiche auf einem Datenträger berücksichtigt in denen Benutzerdaten auftreten. Zuletzt werden die Kenntnisse über das physische Löschen auf einem NAND-Datenträger mit einbezogen. Zuvor wird der Vorgang „Zurücksetzen des Systems“ behandelt. Hier ist es wichtig, zu klären, was diese Funktion leistet und ob beim Zurücksetzen alle Benutzerdaten sicher gelöscht werden.

Die Untersuchungen wurden auf den Endgeräten Samsung Galaxy S3 GT-I9300 und Samsung Galaxy S2 GT-I9100 durchgeführt. Beide verfügen über einen sechszehn Gigabyte großen NAND-Datenträger, der über einen Controller in der Hardware angesteuert wird [Sam13]. Das bedeutet, über Linux wird auf den Speicher zugegriffen ohne die Software Schnittstelle MTD verwenden zu müssen. Das S3 wird mit der Android Version 4.1.2 betrieben, wohingegen das S2 die vorangegangene Version 4.0.1 verwendet.

Aus den Erkenntnissen dieser Untersuchungen kann anschließend ein Konzept erzeugt werden, das beschreibt, durch welche Schritte innerhalb des Android Systems es möglich ist, von einem sicheren Löschvorgang auszugehen.

5.1 Zurücksetzen des Systems

Der Werksreset oder unter Android 4.1.2 auf dem Samsung Galaxy S3 „Sichern und zurücksetzen“ genannt, wird von Nutzern eines Smartphones vor dem Verkauf häufig verwendet. Beim Aufrufen dieser Funktion warnt das System davor, dass alle Benutzerdaten darauf unwiederbringlich gelöscht werden. Im Folgenden soll der Beweis dieser These, für die oben genannten Endgeräte, erbracht werden.

Um zu ermitteln, welche Schritte das System beim Zurücksetzen durchführt wird sowohl der Quelltext von [Ads13], als auch eine vom System erstellte Logdatei eines Resetvorgangs, des Samsung Galaxy S3 hergenommen. Es spielt keine Rolle auf welchem Wege das Zurücksetzen stattfindet. Sowohl über das Android-System, als

auch über den Recovery-Modus[19], wird der gleiche Code ausgeführt [Ads13]. In der App Einstellungen wird unter dem Punkt „Sichern und zurücksetzen“ dem System mittels „wipedata“ Flag zu verstehen gegeben, dass das System zurückgesetzt werden soll. Dadurch schreibt das System, je nach den Angaben des Nutzers, welche Datenträgerbereiche gelöscht werden sollen, in die Cache-Partition Befehle, die beim Zurücksetzen durchgeführt werden. Die Möglichkeiten des Nutzers sind dabei folgende.

- Zurücksetzen
- Formatieren des USB Speichers
- Löschen des externen Speichers

Die Auswahl veranlasst das Setzten von Werten für die entsprechenden Bereiche des Speichers. In der Logdatei protokolliert das System welche sogenannten Flags gesetzt wurden, wenn diese aus den oben genannten Befehlen aus dem Cache gelesen werden. Dieser Vorgang wird als „collecting command“ bezeichnet (siehe Abbildung 5-1)

```
[collecting command]
stat() of /dev/block/mmcblk0p8 succeeded on try 1
I:Got arguments from /cache/recovery/command
Command: "/sbin/recovery" "--wipe_data"
previous_runs = 0
send_intent = (null)
update_package = (null)
att_fota_update = 0
tmo_fota_update = 0
wipe_data = 1, wipe_cache = 1,wipe_sdcard = 0,delete_data = 0
show_ui_text = 0
update_factory_csc = 0,update_home_csc = 0
carry_out = (null)
```

Abbildung 5-1 Aus dem Cache gelesene Befehle zum Zurücksetzen des Samsung Galaxy S3

Anschließend wird das System heruntergefahren und der Recovery-Modus im, auf der Recovery-Partiton gesicherten, System gestartet. In diesem Modus wird nun der Inhalt der Data und Cache Partition gelöscht. Wie das im Einzelnen abläuft lässt sich nicht allgemein beantworten, da für jedes Endgerät eine eigene Methode dafür

[19] Ein Android Smartphone startet mittels spezieller Tastenkombination in den recovery Modus. Das ist ein eigenes kleines Betriebssystem (beim S3 unter Partition 6 installiert)

implementiert werden muss. Im Android-Quellcode findet sich deshalb die virtuelle Methode WipeData() inklusive folgender Dokumentation.

```
virtual int WipeData() { return 0; }
// Called when we do a wipe data/factory reset operation (either via a
// reboot from the main system with the --wipe_data flag, or when the
// user boots into recovery manually and selects the option from the
// menu.)  Can perform whatever device-specific wiping actions are
// needed.  Return 0 on success.  The userdata and cache partitions
// are erased AFTER this returns (whether it returns success or not).
```

Je nachdem ob der Nutzer zusätzlich die Option zum Löschen des externen Speichers und des emulierten externen Speichers gewählt hat, werden diese ebenso gelöscht. Nach dieser Prozedur müssen die Partitionen neu formatiert werden. Im Falle des Endgerätes Galaxy S3 werden die Partitionen Data und Cache mit dem Dateisystem EXT4 formatiert. Dies geschieht über die Methode erase_volume(), welche zunächst nur prüft, welche Partition ihr übergeben wurde. Dementsprechend gibt sie eine Anzeige für den Nutzer auf dem Display aus und schreibt Einträge in die Logdatei. Anschließend gibt sie diese Partition, unter der Anwendung der Methode format_volume(), zurück. Diese verfügt über die Möglichkeit, jedes der auf dem Endgerät möglichen Dateiformate (siehe Kapitel 2.1.1) zu schreiben. Zum Schluss werden noch spezielle System Einstellungen vorgenommen und vom Hersteller gewählte Software installiert. Dieser letzte Schritt ist hier nicht wichtig und wird nicht weiter verfolgt.

Das Zurücksetzen des Systems veranlasst also das Löschen der Data und Cache Partition. Wie sicher dieses Vorgehen ist, soll im Folgenden mit den Test Endgeräten untersucht werden.

Ein Speicherbereich auf einem persistenten Datenträger, der als Partition bezeichnet wird, kann mit einem unabhängigen Dateisystem beschrieben werden. Die Partition mmcblk0p12 (siehe Kapitel 2.1.2) ist mit EXT4 formatiert und besteht somit aus einer bestimmten Anzahl von Blockgruppen. Im Falle der hier genannten data Partition des Samsung Galaxy S3, gibt es 93 davon. Diese Aufteilung ist wichtig, um die Adressierung von gespeicherten Daten zu gewährleisten. Innerhalb einer Blockgruppe sind folgende Attribute wichtig, um gesicherte Daten ausfindig machen

zu können. Der Inode, der Filesystemblock (Im Folgenden als FS-Block bezeichnet) und das Verzeichnis.

Ein Inode steht für eine Datei und gibt an, in welchen FS-Blöcken diese gesichert ist [Ext13]. Die FS-Blöcke haben eine feste Größe, im Falle von EXT4 sind das genau 4096 Byte, und unterteilen den gesamten Speicherbereich. Die oben erwähnten Blockgruppen fassen bis auf die Letzte, immer die gleiche Menge FS-Blöcke zusammen. Die Adressierung in einem Inode zu der referenzierenden Datei erfolgt über die Angabe des FS-Blocks. Da diese über den kompletten Speicher hinweg durchnummeriert sind, kann damit ab dem Beginn der Partition der genaue physische Speichertort ermittelt werden.

Verzeichnisse sind in einer Baumstruktur aufgebaut und enthalten den Inode jeder darin enthaltenen Datei, sowie den jeweiligen Dateinamen. Um nun den Speicherbereich zu untersuchen, auf dem eine bestimmte Datei gesichert ist, wurden in dieser Arbeit folgende Schritte durchgeführt.

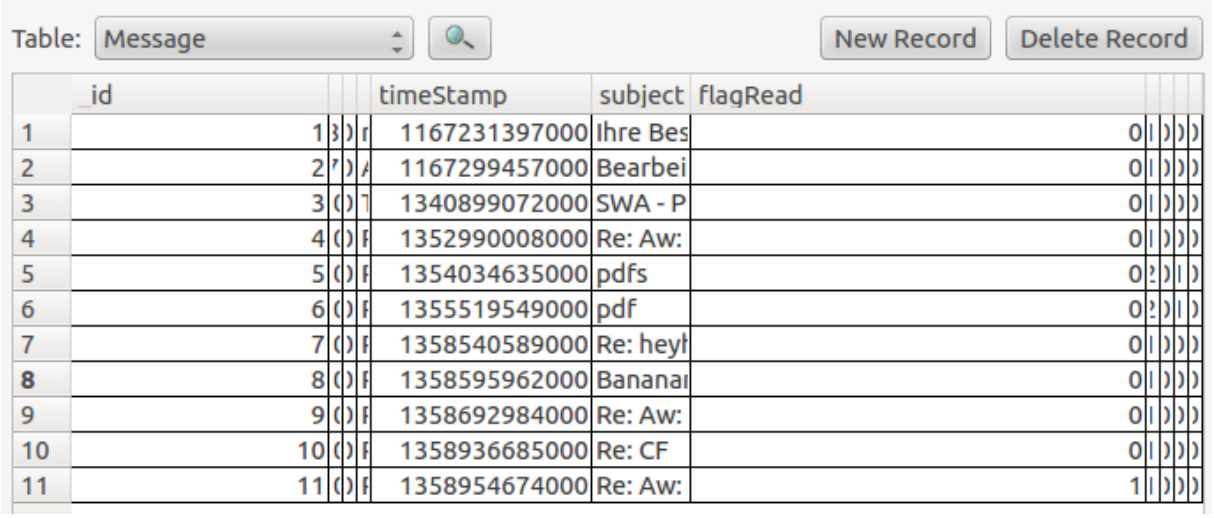
- Ein Image der Partition oder des Datenträgers erstellen (exakte Kopie)
- Image in eine Ubunut-Workstation einhängen
- Ins Verzeichnis der fraglichen Datei navigieren
- Alle enthaltenen Inodes anzeigen lassen
- Mittels fsstat[20] die Blockgruppe des fraglichen Inodes ermitteln
- Im Inode den FS-Block ermitteln, in dem die Datei geischert ist
- FS-Block untersuchen

Dieses Vorgehen kommt bei allen nun folgenden Untersuchungen zum Tragen. Innerhalb dieses Unterkapitels soll klar werden, ob Dateien bei einem Zurücksetzen des Systems gelöscht und neu angelegt werden oder bearbeitet werden. Des Weiteren soll geklärt werden ob Daten die gelöscht wurden als sicher gelöscht gelten oder ob sie wiederhergestellt werden können.

[20] Vgl. Seite 61

Beobachtung einer Datei über ein Werksreset hinweg

Die ausgewählte Datei befindet sich in der Benutzerdaten-Partition (data) und ist auf jedem Smartphone, dass die Android Version 4 enthält zu finden. Ihr Dateiformat ist SQLite, die Dateiendung ist folglich .db. Der Dateiname lautet EmailProvider.db, der Pfad der Datei ist /data/data/com.android.email/databases. Um nachvollziehen zu können, ob und wie diese Datei beim Werksreset verändert wurde, wird diese mittels erzeugen eines E-Mail Kontos verändert. Mit der Software SQLite Browser können die Einträge in dieser Datei angezeigt werden (siehe Abbildung 5-2).



	id	timeStamp	subject	flagRead
1	1	1167231397000	Ihre Bes	0
2	2	1167299457000	Bearbei	0
3	3	1340899072000	SWA - P	0
4	4	1352990008000	Re: Aw:	0
5	5	1354034635000	pdfs	0
6	6	1355519549000	pdf	0
7	7	1358540589000	Re: heyl	0
8	8	1358595962000	Bananar	0
9	9	1358692984000	Re: Aw:	0
10	10	1358936685000	Re: CF	0
11	11	1358954674000	Re: Aw:	1

Abbildung 5-2 Die Tabelle Messages in der Datenbank EmailProvider.db, angezeigt im SQLite Browser

Das Postfach enthält elf E-Mails. Außer der ID, dem Zeitstempel, dem Betreff und dem gelesen Flag, enthält die Tabelle Messages noch weitere Details zu den jeweiligen E-Mails. Hier ist allerdings nur wichtig, dass die, seit der Installation der Applikation vorhandene, leere Datenbankdatei verändert wurde.

Bevor nun das System zurückgesetzt wird, muss festgestellt werden, wo diese Datei auf dem Datenträger gespeichert ist. Dazu werden die oben gezeigten Schritte durchgeführt. Wird der ermittelte Speicherort mit einem Hex Editor untersucht findet sich, wie in Kapitel 2.1.2 gezeigt, der SQLite Header. Im Anschluss stehen unter anderem die Einträge zu den elf gesicherten E-Mails.

Da nun sowohl der Speicherort, als auch der Inhalt der Zieldatei sichergestellt wurden, kann das Endgerät zurückgesetzt werden. Im Anschluss muss erneut ein Image erzeugt werden, bei dem direkt der zuvor ermittelte Speicherort untersucht

werden kann. Die zuvor an dieser Stelle gesicherte Datei ist nicht mehr vorhanden. Der Bereich vom Beginn des Headers, um die Dateigröße addiert, enthält keine der zuvor gesicherten Inhalte mehr. Stattdessen befindet sich an dieser Stelle eine SQLite Datei ohne jeglichen Inhalt. Der Inode, welcher zuvor die Datei referenzierte, ist in der dazugehörigen Blockgruppe keiner Datei mehr zugeordnet.

Man kann davon ausgehen, dass die zuvor gesicherte Datei nicht bearbeitet wurde. Es gibt zwei Möglichkeiten, was stattdessen passiert ist. Entweder wurde EmailProvider.db komplett gelöscht und durch die Installation der Android Applikationen während des Zurücksetzens überschrieben oder als freier Speicherbereich markiert und ebenfalls überschrieben. Ein weiterer Versuch soll zeigen ob Daten, die nicht durch eine Neuinstallation überschrieben werden sicher entfernt werden.

Wird der komplette Inhalt des Datenbereiches gelöscht

Nachdem nun ein sehr kleiner spezieller Teil des internen Speichers betrachtet wurde, soll nun der gesamte Speicherbereich betrachtet werden. Dadurch soll gezeigt werden, ob Benutzerdaten im kompletten internen Speicher nach dem Werksreset wiederhergestellt werden können oder restlos gelöscht sind. Dazu wurde sowohl beim Modell S3 als auch beim Modell S2 eine Datei, die den String „Peter“ enthält im Data Bereich gesichert. Dies führt dazu, dass eine Strings Analyse als Ergebnis eine .txt Datei ausgibt, welche größer als der interne Speicherbereich selbst ist. Somit kann man davon ausgehen das nahezu jeder nicht vom System belegt Speicherbereich mit diesem String beschrieben wurde. Abbildung 5-3 zeigt die Verteilung der belegten Blöcke auf der Data Partition vor dem Speichern der Test Datei, im Anschluss daran und nach dem Zurücksetzen des Systems.

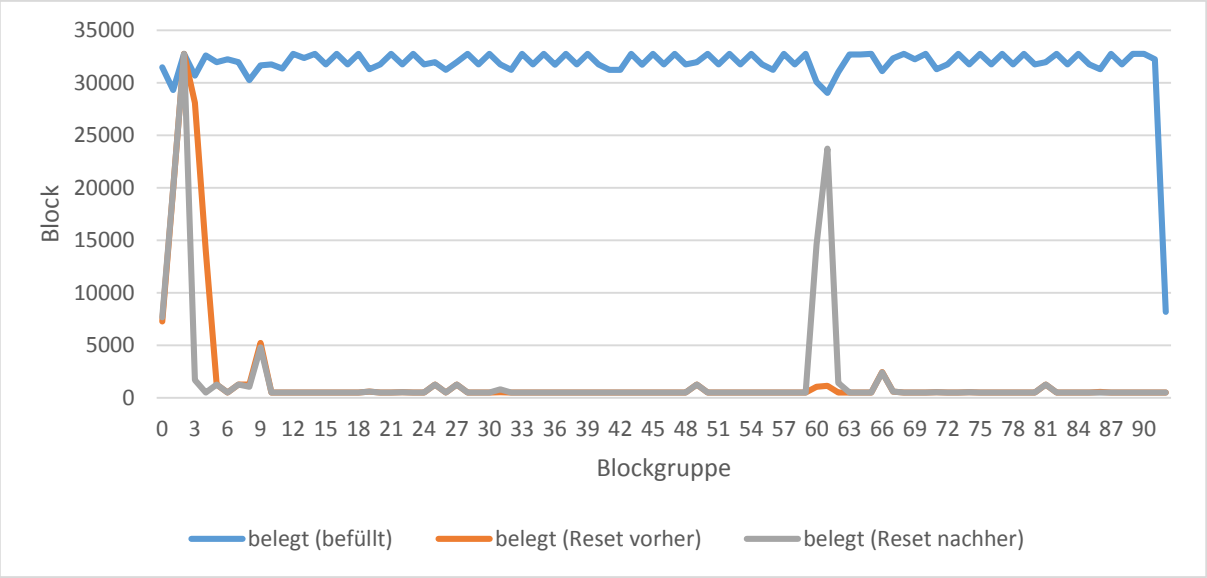


Abbildung 5-3 Belegte Blöcke innerhalb aller Blockgruppen der Data Partition des Samsung Galaxy S3

Dies alleine beweist allerdings noch nicht, dass der Inhalt der Test Datei nicht wiederhergestellt werden kann. Es könnten nur die Metadaten gelöscht worden sein. Eine Strings Analyse bestätigt jedoch, dass der komplette Speicherbereich überschrieben worden sein muss. Die Ausgabe Datei ist kleiner als 1% des gesamten Speicherbereiches und der String „Peter“ kommt so wie er gesichert wurde, nicht ein einziges Mal vor.

Beim Samsung Galaxy S2 ergibt derselbe Test ein anderes Ergebnis. Die Data Partition dieses Endgerätes ist nur 2 Gigabyte groß. Dieser Speicher wurde nahezu komplett mit dem Test String beschrieben. Nach dem Zurücksetzen des Systems wurde, wie zuvor beim Modell S3 die Ausgabe der Strings Analyse vor und nach der Befüllung verglichen. Der Test String konnte in beiden gefunden werden und die Dateigröße ist nahezu gleich.

Werden nur allozierte Dateien überschrieben

Zuletzt soll gezeigt werden, ob bei einem Reset nur allozierte Bereiche des Datenträgers berücksichtigt werden. Wäre das der Fall, würde das System nur Benutzerdaten löschen, die es kennt. Anderenfalls kann man inklusive der bereits gemachten Erkenntnisse beweisen, dass der komplette Speicherbereich zurückgesetzt wird.

Dieser Versuch wurde auf dem Endgerät Samsung Galaxy S3 vorgenommen. Wie oben gezeigt, wird bei einem Reset sowohl die Data als auch die Cache Partition

gelöscht. Letztere entspricht auf dem genannten Smartphone einer Speicherkapazität von rund einem Gigabyte. Um diesen Speicherbereich beschreiben zu können, ohne eine Allokation der betroffenen Blöcke zu verursachen, muss dieser vom System entkoppelt werden. Diese Voraussetzung ist für die Cache Partition im Betrieb möglich, was es leichter macht diesen Versuch durchzuführen.
Die Cache Partition wurde zunächst mittels der Anwendung dd auf den externen Speicher kopiert. Dadurch entsteht ein exaktes Abbild der Cache Partition auf der eingelegten SD Karte. Dieses erzeugte Image kann somit auf eine Ubuntu Workstation geladen werden und mittel Hex Editor so verändert werden, dass keine Allokation des beschriebenen Blocks stattfindet. Hier wurde ans Ende der Partition der String „Peter“ in den nicht allozierten und zuvor mit ausschließlich 00 beschriebenen Block eingefügt. Der dafür benötigte Platz von 5 Byte wurde zuvor gelöscht, damit die Größe des Images gleich bleibt. Das veränderte Image wurde anschließend auf den externen Speicher kopiert und dieser in das System eingehängt. Über die Android Debug Bridge wurde mittels folgender Befehle die Cache Partition durch das veränderte Image überschrieben und als neue Cache Partition eingehängt.

```
su
mount
ausgabe

.
.
.

umount /cache
dd if=/dev/block/mmcblk0p8 of=/storage/extSdCard/mmcblk0p8A.dd bs=4096

.kopieren von SD-Karte auf Workstation
.bearbeitung durch Hex Editor
.kopieren von Workstation auf SD-Karte

dd if=/storage/extSdCard/mmcblk0p8B.dd of=/dev/block/mmcblk0p8 bs=4096
mount -t ext4 /dev/block/mmcblk0p8 /cache

dd if=/dev/block/mmcblk0p8 of=/storage/extSdCard/mmcblk0p8C.dd bs=4096
```

Abbildung 5-4 Befehle zum Überschreiben der Cache Partition

Um sicherzustellen, dass dieser Schritt erfolgreich war, wurde erneut ein Image der Cache Partition erzeugt. Eine Analyse des letzten nicht allozierten Blocks ergab, dass der String Peter nun auf dem internen Speicher in dieser Partition steht.
Als letzter Schritt wurde das Zurücksetzen des Systems durchgeführt. Die Analyse des im Anschluss erzeugte Images der Cache Partition ergab, dass der String Peter nicht

in dieser Partition vorhanden ist. Der letzte Block ist wie zuvor ausschließlich mit dem Byte 00 beschrieben und der letzte Block ist nicht alloziert.

5.2 Anwendung verschiedener Löschstrategien

Das Android System verfügt wie in Kapitel 4.3 beschrieben über Möglichkeiten, auf dem Datenträger gesicherte Benutzerdaten wieder zu löschen. In diesem Kapitel werden diese Möglichkeiten aufgegriffen und ihre Sicherheit geprüft. Das heißt, es ist zu klären, ob Benutzerdaten mit den Mitteln des Systems unwiderruflich gelöscht werden können. Dazu werden wie gehabt die beiden Referenz-Endgeräte der Firma Samsung verwendet. Für die Löschstrategien, welche innerhalb einer Applikation durchgeführt werden können, wird wie oben die Applikation E-Mail verwendet. Da diese eine Standard-App ist, kann sie nicht deinstalliert werden. Dem Nutzer ist es nur möglich Applikationen zu deinstallieren, die er selbst installiert hat. Für die Deinstallations-Strategie wird deshalb die E-Mail Applikation Web.de Mail verwendet, die mit der Standard App Google Play Store heruntergeladen werden kann.

Die Applikation com.android.email

Diese Software ist auf Android-Endgeräten mit der Version 4 vorinstalliert und macht es möglich ein bestehendes E-Mail Konto eines beliebigen Anbieters zu verwalten. Die Benutzerdaten, welche über diese Applikation erzeugt werden legt das System im Verzeichnis /data/data/com.android.email ab (siehe Abbildung 5-5).

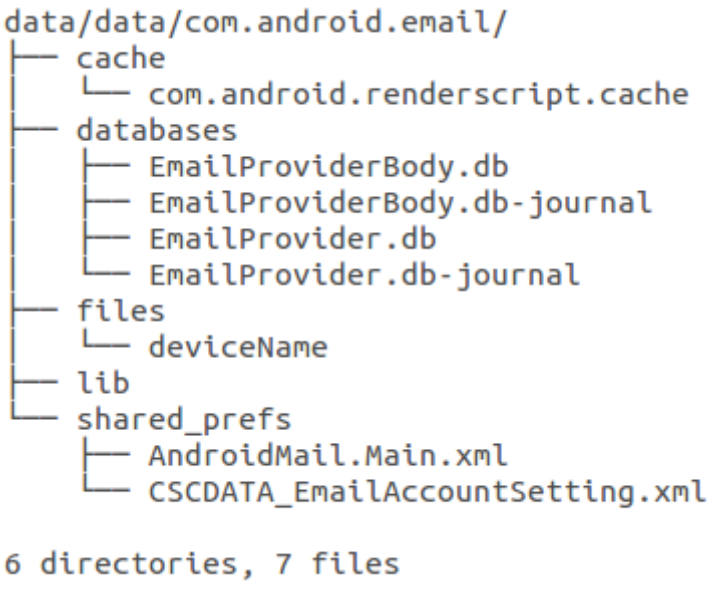


Abbildung 5-5 Verzeichnisbaum der Standard E-Mail App unter Android 4 nach der Installation

Wie man sehen kann, besteht das Verzeichnis vor der ersten Nutzung aus lediglich 7 Dateien. Diese sind auf die fünf Unterverzeichnis verteilt, welche die Methoden der Datensicherung, wie in Kapitel 2.1.2 beschrieben, abgesehen von dem Punkt der Sicherung auf der externen SD Karte, wiederspiegelt. Die Ordner Cache und File sind als Dateisicherung auf dem internen Speicher zu verstehen. Shared_prefs enthält Schlüssel und Wertepaare. Databases beinhaltet SQLite Datenbanken. Die Verbindung zu Benutzerdaten auf einem online Server liegen in Form der Zugangs Daten ebenfalls innerhalb der SQLite-Datenbanken in der Datei EmailProvider.db vor. Um nun die Sicherung der Daten in diesem Verzeichnis und auf dem Datenträger zu veranschaulichen, wurden mit einem Benutzerkonto E-Mails aus dem Internet geladen. Folgende Benutzerdaten, die hierbei auftreten, sind von Interesse.

- Kontodaten das Kontos
- Kontakte des Kontos
- Nachrichten
- Dateien aus der Anlage

All diese Daten wurden durch die Anmeldung des Nutzers auf das Endgerät übertragen und in das oben genannte Verzeichnis gespeichert. Hier die Verzeichnisstruktur nach der automatischen Synchronisierung der E-Mail-Ordner Eingang, Entwurf, Ausgang etc.

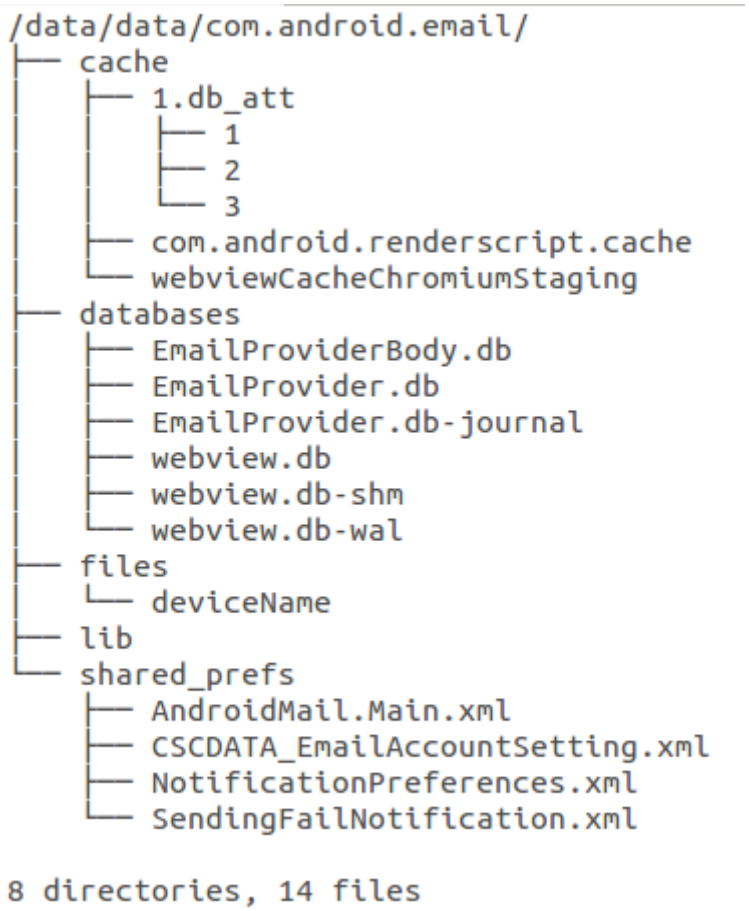


Abbildung 5-6 Verzeichnisbaum der Standard E-Mail App unter Android 4 nach der Synchronisierung

Im Allgemeinen ist der Begriff Cache im Zusammenhang mit der Zwischenspeicherung zu verstehen. Daten werden im Cache gesichert, um sie bei Bedarf schneller laden zu können. Aus diesem Grund legt die Applikation Dateien, die sich in der Anlage einer E-Mail befinden in den Cache, um sie bei Bedarf schnell von dort laden zu können und nicht erneut aus dem Web herunterladen zu müssen.

In den Shared Preferences werden zum größten Teil nicht empfindlich Benutzer Daten gespeichert. Allerdings sind auch Daten das Benutzer Kontos hier gesichert. Einige Beispiel aus der Datei androidmail.main.xml veranschaulichen die Art der darin liegenden Daten:

<string name =
„5ffc….d27.ringtone“>/system/media/audio/notification/S_Postman.ogg
</string> Der Verweis zur Sounddatei beim Empfangen einer E-Mail.

<int name =

5ff2c…d27.automatic.CheckIntervalMinutes" value = "15"/>

Das Intervall in Minuten in dem E-Mails automatisch abgerufen werden.

<string name =

„5ff2c…d27.signatur"> Von Samsung Mobile gesendet

</string> String am Ende jeder gesendeten E-Mail Nachricht.

<string name=

„5ff2c….d27.description> xxxxxx@xxxxx.xx

Die E-Mail-Adresse, welche für diesen Account gültig ist.

<string name=

„5ff2c….d27.name> xxxxxx

Der Name, den der Empfänger als Absender jeder Nachricht sieht.

Zuletzt die Daten aus den SQLite Datenbanken. Hier wird der größte Teil aller Benutzerdaten festgehalten. Es werden die Account Daten und sämtliche Nachrichten gesichert. Dabei sind die Dateien EmailProvider.db und EmailProviderBody.db wichtig, da sie folgenden Inhalt speichern.

EmailProvider.db:

- Kontakte
- Empfangszeit
- Betreff
- Textausschnitt
- Benutzername, Passwort (verschlüsselt), Serveradresse

EmailProviderBody.db:

- Komplette Nachricht
  - HTML
  - Text Form

5.2.1 Samsung Galaxy S3

Löschen von Datensätzen

Das Löschen von Datensätzen in der Android E-Mail Applikation betrifft Nachrichten. Jede Nachricht besitzt jeweils den zu ihr gehörenden Datensatz in den oben gezeigten Datenbank Dateien. Es gilt nun zu klären, ob beim Löschen einer Nachricht im normalen Betrieb der Applikation, die Datensätze in den Datenbanken sicher gelöscht werden. Es liegt auf der Hand, dass nach dem Löschen einer Nachricht, diese innerhalb der App nicht mehr verfügbar ist. Jedoch muss die betroffene SQLite-Datei nach dem Löschen genauer untersucht werden, um festzustellen, ob eine Wiederherstellung der Informationen möglich ist. Wie in Kapitel 2.1.2 beschrieben wurde, sind die Datenbanken mit dem Vacuum Modus konfiguriert. Folglich ist zu erwarten, dass ein gelöschter Datensatz nicht wiederhergestellt werden kann.

In den Benutzerdaten-Verzeichnissen wurden die Datensätze zu insgesamt elf E-Mails gesichert (siehe Abbildung 5-2). Über das grafische User Interface wurden die Nachrichten mit der ID neun und zehn gelöscht. Eine Untersuchung der Datenbank EmailProviderBody.db ergibt, dass die Datensätze gelöscht worden sind (siehe Abbildung 5-7). Die Dateigröße dieser Datei hat sich im Vergleich zum Zeitpunkt vor dem Löschen verkleinert. Vorher belegte die Datei 69623 Byte nach dem Löschen belegt sie 53248 Byte.

Table: Body

New Record | Delete Record

	_id	messageKey	htmlContent	textContent	htmlReply	textReply	sourceMessageK
1	1	1		---------- ALTE			
2	2	2		Sehr geehrte Ku			
3	3	3	Hey Peter, <	Hey Peter, ...			
4	4	4	<html><head><r	Wir können uns			
5	5	7	<html><head><r	Gude, ...			
6	6	8	<html><head></	Moin, ich lad gra			
7	7	9	<html><head><r	Hab das gestern			
8	8	10	<html><head></	Welch daten me			
9	11	5	<html><head></	text			
10	12	6					

Abbildung 5-7 Datenbank Datei EmailProviderBody.db nach dem Löschen der Datensätze 9 und 10

Betrachtet man nun den Inhalt des Speichers direkt nach dem Ende der Datei, findet man dort den gelöschten Datensatz.
Der Vacuum Modus veranlasst erst nach einem Commit Befehl innerhalb der Datenbank das Löschen von Datensätzen [Hwa13]. Um sicherzugehen, dass es zu einem Commit kommt, wurde die App Neugestartet bevor erneut ein Image zum Auslesen der Datenbank erstellt wurde. Die Untersuchung der fraglichen Datenbankdatei ergab, dass keine Veränderungen aufgetreten sind. Die Datensätze, welche gelöscht wurden, sind auch nach einem Neustart der Applikation noch immer am Ende der Datei zu finden.

Löschen des App-Caches

Wie oben erwähnt enthält die Applikation com.android.email in ihrem Cache Verzeichnis Benutzerdaten. Diese Dateien speichert die Software beim Laden einer Datei aus dem Anhang einer E-Mail. Dadurch wird gewährleistet, dass beim Anzeigen dieser Nachricht, der Anhang nicht jedes Mal aus dem Web heruntergeladen werden muss. Was passiert nun, wenn die Option Cache löschen im Anwendungen-Manager von Android gewählt wird. Um dies zu klären wird zunächst der Speicherbereich auf dem Datenträger mittels Inode, nach dem oben beschrieben Verfahren, festgehalten und eine exakte Kopie dieses Bereiches erzeugt. Anschließend wird die Option Cache löschen ausgeführt und erneut eine Kopie des gleichen Bereiches erstellt. Dazu ist es nötig, den Speicherort eines Inodes bestimmen zu können. Dies soll im Folgenden beschrieben werden.

Wenn die Inode-Nummer ermittelt wurde, kann mittels des Softwarepaketes Sleutchkit und dem darin enthaltenen Befehl fsstat die gesamte Data-Partition untersucht werden. Als Ergebnis liefert diese, zahlreiche Informationen zur Strukturierung der Partition. Die wichtigsten Werte für diese Berechnung sind folgende.

Inode Range: Gibt an, welche Inodes in jeder Blockgruppe (BG) stehen
Block Range: Gibt an, welche FS-Blöcke zu welcher BG gehören
Inode Tabelle: Gibt an, in welchen FS-Blöcken die Inodes einer BG stehen

Außerdem ist es wichtig zu wissen, dass die Größe eines Inodes, beim hier verwendeten EXT4 Dateisystem, 256 Byte entspricht [Mat07]. Mithilfe dieser

Informationen lässt sich jeder Inode in der gesamten Partition lokalisieren. Zunächst wird die Blockgruppe des gesuchten Inodes ermittelt. Darin kann der Beginn, in Form der FS-Block Nummer, der Inode Tabelle ausgemacht werden. In dieser Arbeit wurde stets innerhalb einer Blockgruppe gearbeitet. Der Grund dafür ist, dass eine Blockgruppe eine kleinere Datenmenge umfasst als die gesamte Partition und sich somit besser in einem Hex Editor untersuchen lässt. In diesem Fall muss der Beginn der Inode Tabelle relativ zum Beginn der Blockgruppe gesehen werden. Der Befehl fsstat gibt die Adresse der Inode Tabelle jedoch relativ zum Beginn der Partition an. Deshalb ist zunächst folgender Schritt nötig.

Byteoffset Inode Tabelle = Beginn Inode Tabelle - Beginn BG * 4096 Byte

Somit kann mittels Hex Editor an den Anfang der Inode Tabelle in der untersuchten Blockgruppe gesprungen werden. Nun ist es nötig, den Offset des gesuchten Inode zu berechnen.

Byteoffset ges. Inode = (ges. Inode – Beginn Inode Range) * 256 Byte

Durch die Addition beider Ergebnisse erhält man den Offset in Byte vom Beginn der Blockgruppe. Dieses Vorgehen wird nun verwendet, um den Inode einer Datei im Cache Verzeichnis der E-Mail Applikation zu finden. Die Dateien, die dafür zur Verfügung stehen sind in Tabelle 5-1 zu sehen.

Dateiname	Dateiformat	Dateigröße in Byte	FS-Blöcke
1	PNG	28415	7
2	PDF	9063	3
3	PDF	37734	10

Tabelle 5-1 Dateien im Cache Verzeichnis für die Untersuchung der Cache löschen Funktion

Die Datei 1.png wird durch den Inode 74024 referenziert. Dieser liegt in Blockgruppe neun. Seine Inode Range umfasst 8127 Inodes, wobei der Erste die Nummer 73153 und der Letzte die Nummer 81280 trägt. Die oben gezeigte Berechnung ergibt den Byteoffset des Inodes 74024. Er lautet 222976 Byte. Nun muss noch der Byte Offset bis zum Beginn der Inode Tabelle berechnet werden. Der Anfang der Inode Tabelle

steht in Blockgruppe 9 im FS-Block 295659. Die Blockgruppe selbst beginnt mit FS-Block 294912. Nach der oben gezeigten Rechnung ergibt das einen Offset von 3059712 Byte. Addiert mit dem zuvor berechneten Offset des Inodes ergibt sich der Offset in Bytes für den Inode 74024 ab dem Beginn der Blockgruppe 9. Durch extrahieren dieser Blockgruppe aus der gesamten Partition kann nun jeder Inode, aus der oben erwähnten Inode Range, auf diese Weise gefunden werden. In Abbildung 5-8 sind je die ersten 64 Byte zu sehen, die sich an der zuvor berechnet Stelle auf dem Datenträger befinden.

```
FF 81 68 27     FF 6E 00 00     D2 36 C5 51     24 A9 C5 51
D2 36 C5 51     00 00 00 00     68 27 01 00     38 00 00 00
80 00 08 00     01 00 00 00     0A F3 01 00     04 00 00 00
00 00 00 00     00 00 00 00     07 00 00 00     38 90 04 00

Nach dem Löschen des Caches--------------------------------

80 81 68 27     00 00 00 00     D2 36 C5 51     8C 60 C5 51
8C 60 C5 51     8C 60 C5 51     68 27 00 00     00 00 00 00
80 00 08 00     01 00 00 00     0A F3 00 00     04 00 00 00
00 00 00 00     00 00 00 00     00 00 00 00     00 00 00 00
```

Abbildung 5-8 Die jeweils ersten 64 Byte des Inodes 74024, vor und nach dem Löschen des Caches

```
4-7   : Dateigröße (Anzahl der FS-Blöcke),
56-57 : Anzahl der Blöcke,
60-63 : FS-Block des Anfangs der referenzierten Datei [Ext13]
```

Wie klar zu erkennen ist, wird die Datei 1.png nach dem Löschen des Caches nicht mehr durch die Metadaten im Inode referenziert. Um nun den Verbleib des tatsächlichen Dateiinhaltes zu prüfen, muss der Datenträger unter FS-Block 38 90 04 betrachtet werden. Dieser Wert ist in den Bytes 60-63 im Inode vor dem Löschen des Caches zu finden. Dieser ist in Hexadezimal und Little Endian[21] angegeben. Somit muss bei der Umrechnung in einen dezimalen Wert von rechts gelesen werden.

38 90 04 -> 04 90 38 –Umrechnung in dezimal> 299064

In diesem FS-Block und in den sieben darauffolgenden, wird die Datei 1.png auf dem Datenträger gesichert. Sieben Blöcke entsprechen exakt 28672 Byte. Da die Datei selbst 28416 Byte groß ist, bleiben 256 Byte ungenutzt.

[21] Beschreibt die Reihenfolge in der die Stellen in einer Zahl gesichert werden. Bei Little Endian wird die Stelle mit dem höchsten Wert als erstes gesichert.

Nach dem Löschen ist die Datei nicht mehr über das Betriebssystem zu finden. Da wie oben gezeigt die Metadaten zur Größe und Speicherort der Datei gelöscht worden sind. Der Verzeichnisbaum der Applikation zeigt das keine Datei mehr im Cache liegt (siehe Abbildung 5-9). Keiner der oben gezeigten Inodes, welche die drei Testdateien referenzieren, liegen mehr in diesem Ordner. Sie referenzieren fortan entweder eine andere Datei oder sie sind frei. Da der Speicherort der Datei noch bekannt ist, kann nachvollzogen werden, ob auch der Inhalt der Datei entfernt wurde.

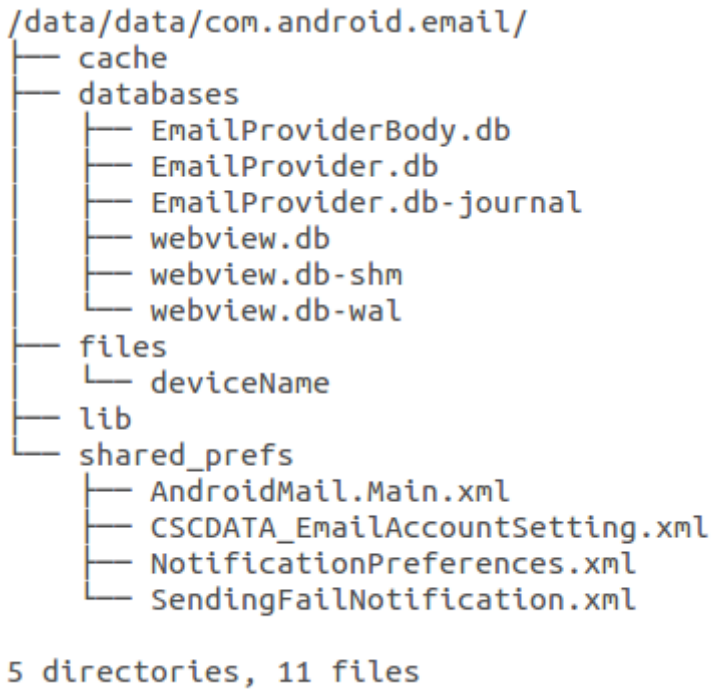


Abbildung 5-9 Verzeichnisbaum der E-Mail Applikation nach dem Löschen des Caches

Über einen Hex Editor zeigt sich, dass die ersten sechs Blöcke komplett gelöscht wurden, und zwar unwiderruflich, da sie ausschließlich mit dem Byte 00 beschrieben sind. Sechs Blöcke entsprechen 24576 Byte. Die übrigen 3840 Byte sind unverändert auf dem Datenträger geblieben.

Die anderen beiden Testdateien zeigen bezüglich der Metadaten das gleiche Ergebnis. Beim Dateiinhalt ist das Ergebnis bei 2.pdf ähnlich dem Ergebnis bei 1.png. Die ersten beiden Blöcke sind gelöscht (8192 Byte), im dritten Block sind die letzten 871 Byte (siehe Tabelle 5-1) der Datei vorhanden. Die Datei 3.pdf wurde ohne jeglichen Rest gelöscht, das heißt alle Bytes wurden mit 00 überschrieben.

Löschen der Daten

Die Möglichkeit alle Benutzerdaten einer Applikation mit einem Knopfdruck zu löschen scheint eine einfache Strategie zu sein. Jedoch muss klar sein, ob diese Strategie sicher ist. Wie sich oben gezeigt hat, ist das Löschen einzelner Datensätze aus den Datenbanken der E-Mail Applikation nicht sicher. Wie die Umsetzung beim Löschen aller Datensätze ist, soll nun betrachtet werden.

Wie zuvor wird dafür der Inode einer betroffenen Datei untersucht. Dadurch kann auf den Speicherort der Datei geschlossen und dieser über die Durchführung des Löschvorgangs hinweg betrachten werden. Die betroffene Datei ist die bereits bekannte SQLite Datei EmailProviderBody.db. Diese Datei existiert immer, egal ob Benutzerdaten gesichert wurden oder die Applikation unbenutzt ist. Dadurch kann ein guter Vergleich, vor und nach dem Vorgang des Löschens durchgeführt werden.

Der Inode 65585 referenziert die Datei. Ihre Größe beträgt 53248 Byte, wie in Abbildung 5-10 zu sehen ist, welche den genannten Inode zeigt. EmailProviderBody.db ist in drei Fragmente geteilt. Das Erste beginnt bei FS-Block 1967739, dieser Bereich liegt in Blockgruppe 60. Das zweite Fragment beginnt sechs Blöcke weiter in der gleichen Blockgruppe. Das Letzte beginnt physisch gesehen weit entfernt in Blockgruppe 3.

```
B0 81 68 27    00 D0 00 00    66 B3 B8 51    04 80 EF 51
5B E3 C5 51    00 00 00 00    68 27 01 00    68 00 00 00
80 00 08 00    01 00 00 00    0A F3 03 00    04 00 00 00
00 00 00 00    00 00 00 00    03 00 00 00    7B 06 1E 00
03 00 00 00    03 00 00 00    81 06 1E 00    06 00 00 00
07 00 00 00    69 87 01 00    00 00 00 00    00 00 00 00
```

Abbildung 5-10 Erste 96 Byte des Inode 65585 vor dem Löschen der Daten

Nachdem die Option Daten Löschen durchgeführt wurde, referenziert der Inode 65585 noch immer die beobachtete Datenbankdatei. Eine Untersuchung des Inodes zeigt jedoch, dass dieser nun auf andere Bereiche des Datenträgers verweist (siehe Abbildung 5-11). Die Dateigröße hat sich von zuvor 53248 Byte auf 24576 Byte reduziert. Der Speicherbereich, der zuvor von diesem Inode referenziert wurde, ist in allen drei Fragmenten unverändert geblieben. Das heißt, es wurde eine neue Datenbankdatei erstellt. Die alte Datenbank ist unverändert geblieben, wird jedoch nicht mehr von einem Inode referenziert und kann somit nicht mehr über das

Dateisystem gefunden werden. Die Wiederherstellung ist möglich bis die Fragmente der Datei durch andere Daten überschrieben werden.

```
FF 81 68 27     00 60 00 00     AE 9F EF 51     91 B6 FC 51
AE 9F EF 51     00 00 00 00     68 27 01 00     30 00 00 00
80 00 08 00     01 00 00 00     0A F3 02 00     04 00 00 00
00 00 00 00     00 00 00 00     03 00 00 00     22 56 1E 00
03 00 00 00     03 00 00 00     28 56 1E 00     00 00 00 00
```

Abbildung 5-11 Erste 80 Byte des Inode 65585 nach dem Löschen der Daten

Deinstallieren einer Applikation

Die Applikation Web.de Mail speichert wie die vergleichbare Standard Software von Android, Benutzerdaten aus dem E-Mail Konto des Nutzers. Wobei die Umsetzung etwas anders ist. Dateien etwa, die als Anhang heruntergeladen werden, sichert das Programm in einem separaten Ordner im Database Verzeichnis. Im Cache Ordner, den die Android Software für Dateien aus dem Anhang verwendet, sichert Web.de Mail bei diesem Test keine Benutzerdaten. Eine Datenbank, welche die geladenen Nachrichten in einer Datei enthält, existiert hier[22]. Diese soll im Folgenden, vor und nach der Installation von Web.de Mail betrachtet werden. Von einer weiterführenden Beschreibung der Applikation wird hier abgesehen, da nur die Sicherheit des Löschvorgangs wichtig ist.

Für die Bestimmung des Inodes wurde das, durch die Installation entstandene Verzeichnis de.web.mobile.android.mail angewählt. Anschließend wurde nach dem gleichen Muster wie im Kapitel 5.1 beschrieben vorgegangen, um den Inode und den Speicherbereich der untersuchten Datei zu finden. Die Fragmentierung dieser Datei beläuft sich auf drei Teile, bei einer Dateigröße von 90112 Byte. Die tatsächliche Größe beläuft sich allerdings auf 90113 Byte. Dieses eine Byte wird nach dem Deinstallieren wichtig sein.

Nachdem die Deinstallation durchgeführt wurde, findet sich im Pfad data/data das Verzeichnis de.web.mobile.android.mail nicht mehr, welches zuvor alle Benutzerdaten enthielt. Der referenzierende Inode verweist noch immer auf die FS-Blöcke, welche vor der Deinstallation ausgewiesen wurden. Die Dateigröße jedoch ist auf null gesetzt worden. Abbildung 5-12 zeigt die Ausgabe eines Hex Editors, welche den

[22] Da sich die Entwickler bei der Benennung dieser Datei auf eine 32 Zeichen umfassende Kombination aus Zahlen und Buchstaben entschieden haben, wird dieser nicht aufgeführt.

Speicherbereich der Testdatei beim Übergang vom vierten zum fünften FS-Block, des dritten und letzten Fragmentes enthält. Es wurde ein Löschvorgang durchgeführt, wodurch bis auf den letzten Block der Datei jeglicher Inhalt davon mit dem Byte 00 überschrieben wurde. Zu beachten ist, dass im fünften FS-Block nur ein einziges Byte gesichert wurde. Exakt dieses Byte ist nach dem Deinstallieren der Applikation auf dem Datenträger zurückgeblieben.

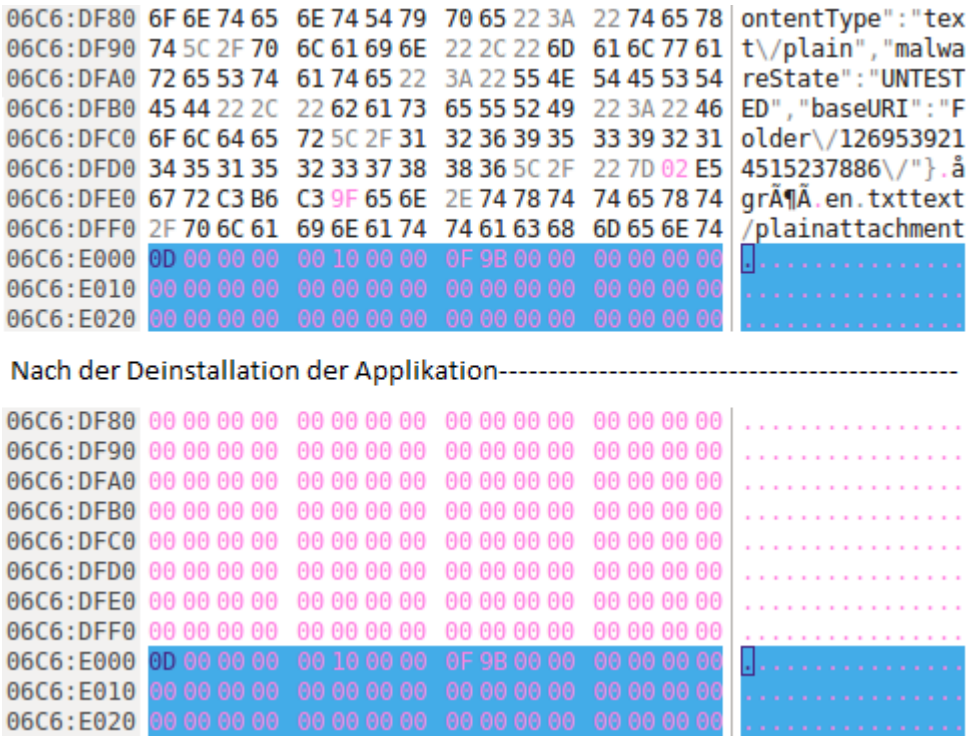


Abbildung 5-12 Übergang von FS-Block 4 zu 5 des dritten Fragmentes der Testdatei, vor und nach der Deinstallation der Applikation

Löschen einer Datei

Wie in Kapitel 2.1.2 beschrieben, ist es beim Samsung Galaxy S3 so, dass vom Benutzer direkt gesicherte Dateien den Speicherbereich der Data Partition belegen. Der emulierte externe Speicher, welcher bei der Verbindung des Android-Endgerätes mit einem Rechner, in das System des Rechners eingehängt wird, hat seinen Ursprung unter /data/media. Dieser Abschnitt beschäftigt sich mit dem Löschvorgang der dort abgelegten Dateien. Dafür wurde eine circa ein Gigabyte große Testdatei im beschriebenen Bereich gesichert und wieder gelöscht. Ziel war es, herauszufinden, ob der Löschvorgang als sicher bezeichnet werden kann. Abbildung 5-13 zeigt den Inode der 1040100352 Byte großen Testdatei, vor und nach dem Löschen. Wie man sehen

kann, wird die Datei an drei Stellen des Datenträgers gesichert. Auch nach dem Löschen der Datei behält der Inode die Verweise zu diesen Bereichen, jedoch wird die Dateigröße auf null gesetzt. Der Inhalte der Datei, welcher wie zuvor ausschließlich aus einer Aneinanderkettung des Strings „Peter“ besteht, wurde in allen Bereichen komplett mit dem Byte 00 überschrieben.

Anders als bei den zuvor gezeigt Dateien, die allesamt eine Dateigröße unter einem Megabyte aufweisen, ist hier ein Unterschied im Inode auszumachen. Das erste Fragment wird nicht direkt adressiert, sondern mit den, in EXT4 vorkommenden, Extends behandelt [Ext13].

```
B4 81 FF 03     00 AC FE 3D     FF D4 EF 51     6F D8 EF 51
6F D8 EF 51     00 00 00 00     FF 03 01 00     60 FF 1E 00
80 00 08 00     01 00 00 00     0A F3 01 00     04 00 01 00
00 00 00 00     00 00 00 00     FE 01 05 00     00 00 03 00
00 68 00 00     00 10 00 00     00 F0 03 00     00 78 00 00
00 70 00 00     00 10 04 00     00 E8 00 00     00 60 00 00
00 A0 04 00     86 0D EB AC     00 00 00 00     00 00 00 00
00 00 00 00     00 00 00 00     00 00 00 00     00 00 00 00
1C 00 00 00     50 43 5E 0E     44 08 06 06     B8 85 DD 1D
FF D4 EF 51     B8 85 DD 1D

Nach dem Löschen----------------------------------------

B4 81 FF 03     00 00 00 00     FF D4 EF 51     20 FD F0 51
20 FD F0 51     20 FD F0 51     FF 03 00 00     00 00 00 00
80 00 08 00     01 00 00 00     0A F3 00 00     04 00 00 00
00 00 00 00     00 00 00 00     FE 01 05 00     00 00 03 00
00 68 00 00     00 10 00 00     00 F0 03 00     00 78 00 00
00 70 00 00     00 10 04 00     00 E8 00 00     00 60 00 00
00 A0 04 00     86 0D EB AC     00 00 00 00     00 00 00 00
00 00 00 00     00 00 00 00     00 00 00 00     00 00 00 00
1C 00 00 00     3C DC EB 6E     3C DC EB 6E     B8 85 DD 1D
FF D4 EF 51     B8 85 DD 1D
```

Abbildung 5-13 Erste 152 Byte des Inode 536737 vor und nach dem Löschen der Testdatei

5.2.2 Samsung Galaxy S2

Löschen des Caches

Die Durchführung dieser Untersuchung gleicht der aus Kapitel 5.2.1. Der Inode, der hier betrachteten Datei lautet 115443. Aus Abbildung 5-14 ist zu entnehmen, dass die referenzierte Datei 5748 Byte groß ist, zwei Blöcke belegt und auf dem Datenträger des Endgeräts in den FS-Blöcken 460918 und 460919 zu finden ist.

Nach dem Löschen des Caches ist im Inode sowohl die Dateigröße als der Verweis zu den genannten Blöcken gelöscht worden. Auf dem Datenträger befindet sich der

Dateiinhalt, der von Inode 115443 referenziert wurde, in absolut unveränderter Form. Dies gilt ebenso für alle anderen Dateien, welche zuvor im Cache-Verzeichnis abgelegt waren.

```
FF 81 24 27     74 16 00 00     C5 32 F0 51     04 EB FB 51
C5 32 F0 51     00 00 00 00     24 27 01 00     10 00 00 00
80 00 08 00     01 00 00 00     0A F3 01 00     04 00 00 00
00 00 00 00     00 00 00 00     02 00 00 00     76 08 07 00

Nach dem Löschen des Caches---------------------------------

80 81 24 27     00 00 00 00     C5 32 F0 51     51 34 F0 51
51 34 F0 51     51 34 F0 51     24 27 00 00     00 00 00 00
80 00 08 00     01 00 00 00     0A F3 00 00     04 00 00 00
00 00 00 00     00 00 00 00     00 00 00 00     00 00 00 00
```

Abbildung 5-14 Erste 64 Byte des Inodes 115443 vor und nach dem Löschen des Caches

Löschen der Daten

Unter Linux Systemen ist es möglich, innerhalb des Terminals, mit dem Befehl ls alle Ordner und Dateien des aktuellen Verzeichnisses auflisten zu lassen. Dieser Befehl wurde bei den Untersuchungen in diesem Kapitel stets verwendet, um den Inhalt der Benutzerdaten-Verzeichnisse anzuzeigen. Mit dem Zusatz –i wird in der Auflistung außerdem der Inode zur jeweiligen Datei angezeigt. Abbildung 5-15 zeigt die Ausgabe unter Verwendung des genannten Befehls, für die Ordner Cache und Databases. Wie zu sehen ist, befinden sich fünf Dateien im Cache, welche durch das Laden des Anhangs aus E-Mail Nachrichten entstanden sind. Außerdem sind die bekannten Dateien EmailProvider.db und EmailProviderBody.db vorhanden.

```
peter@peter-slame:/media/mmc10/data/com.android.email/cache/1.db_att$ ls -li

115543 -rwxrwxrwx 1 10020 10020   741 Aug  5 10:03 1
115541 -rwxrwxrwx 1 10020 10020  5748 Aug  5 10:03 2
115542 -rwxrwxrwx 1 10020 10020 28415 Aug  5 10:03 3
115545 -rwxrwxrwx 1 10020 10020 16758 Aug  5 10:03 4
115544 -rwxrwxrwx 1 10020 10020 45429 Aug  5 10:03 5

peter@peter-slame:/media/mmc10/data/com.android.email/databases$ ls -li

115136 -rwxrwxrwx 1 10020 10020  12288 Jul 25 16:28 EmailProviderBody.db
115138 -rw------- 1 10020 10020  32768 Jan  1  2000 EmailProviderBody.db-shm
115137 -rw------- 1 10020 10020 391432 Aug  5 10:03 EmailProviderBody.db-wal
115134 -rwxrwxrwx 1 10020 10020 241664 Aug  5 10:04 EmailProvider.db
115135 -rw-rw---- 1 10020 10020      0 Aug  5 10:04 EmailProvider.db-journal
115527 -rw-rw---- 1 10020 10020  28672 Aug  5 10:02 webviewCookiesChromium.db
115222 -rw-rw---- 1 10020 10020      0 Aug  5 10:07 webviewCookiesChromiumPrivate.db
115524 -rw-rw---- 1 10020 10020  12288 Aug  5 10:02 webview.db
115526 -rw------- 1 10020 10020  32768 Aug  5 10:02 webview.db-shm
115525 -rw------- 1 10020 10020  45352 Aug  5 10:02 webview.db-wal
```

Abbildung 5-15 Liste der Benutzerdatendateien aus dem Cache und Database Verzeichnis vor dem Löschen

Nachdem die Option Daten löschen ausgeführt wurde, sind wie zuvor beim Ausführen der Cache löschen Option die Inodes der Dateien verändert worden. Die Dateien selbst liegen unverändert auf dem Datenträger vor. Ein erneutes Ausführen des ls Befehls zeigt, dass der Inhalt des Benutzerdaten Verzeichnisses auf den Stand zurückgesetzt wurde, wie man ihn nach einer Neuinstallation vorfindet (siehe Abbildung 5-16). Teilweise wurden für diese Dateien Inodes verwendet, welche vor dem Löschen der Daten auf die nun verwaisten Benutzerdaten referenzierten (vgl. Abbildung 5-15 und Abbildung 5-16)

```
peter@peter-slame:/media/mmc10/data/com.android.email/cache$ ls -li

peter@peter-slame:/media/mmc10/data/com.android.email/databases$ ls -li

115412 -rw-rw---- 1 10020 10020  12288 Jan  1  2000 EmailProviderBody.db
115422 -rw------- 1 10020 10020  32768 Jan  1  2000 EmailProviderBody.db-shm
115384 -rw------- 1 10020 10020  24752 Jan  1  2000 EmailProviderBody.db-wal
115171 -rw-rw---- 1 10020 10020 233472 Jan  1  2000 EmailProvider.db
115138 -rw-rw---- 1 10020 10020      0 Jan  1  2000 EmailProvider.db-journal
```

Abbildung 5-16 Liste der Benutzerdatendateien aus dem Cache und Database Verzeichnis nach dem Löschen

Deinstallieren einer Applikation

In Kapitel 5.2.1 wurde bereits getestet, wie sicher das Deinstallieren der Test Applikation Web.de Mail in Bezug auf eine mögliche Wiederherstellung ist. Der gleiche Test wurde ebenfalls für das Samsung Galaxy S2 durchgeführt. Das Ergebnis unterscheidet sich zu dem des Samsung Galaxy S3. Im Inode der untersuchten Datenbank (siehe Kapitel 5.2.1) ändert sich nach dem Deinstallieren die Dateigröße auf null. Der Verweis zu den Blöcken, welche die Dateifragmente beinhalten, bleiben unverändert. Der Inode selbst wird nach dem deinstallieren der Applikation als frei ausgewiesen. Die komplette Datei und ihr Inhalt ist in den zuvor referenzierten drei Fragmenten noch vorhanden.

5.3 Ergebnisse der Untersuchungen

Die Ergebnisse der Untersuchungen zeigen, dass der Grad an Sicherheit der Android internen Möglichkeiten zum Löschen von Benutzerdaten unterschiedlich ausfällt. Sowohl unter den verwendeten Löschstrategien als auch bei den zwei Endgeräten konnten Unterschiede ausgemacht werden.

Im Kapitel 5.1 konnte gezeigt werden, dass ein Werksreset beim Samsung Galaxy S3 den kompletten Speicherbereich der Cache- und Data-Partition sicher löscht. Allerdings spricht dieses Ergebnis nicht allgemein für die Option des Zurücksetzens. Beim Samsung Galaxy S2 konnten nach dem Werksreset nahezu alle zuvor gesicherten Daten wieder gefunden werden. Nur solche Daten, die bei der Neuinstallation von Software überschrieben wurden, konnten als sicher gelöscht bezeichnet werden. Da dies vom Zufall abhängt, kann man nicht von einem gezielten sicheren Löschen sprechen.

Kapitel 5.2.1 beweist, dass es beim Samsung Galaxy S3 vereinzelt möglich ist, ohne die Verwendung von externer Software, Benutzerdaten sicher zu löschen. Allerdings konnten auch Lücken ausgemacht werden. Innerhalb der Testapplikation konnten Datensätze nicht sicher gelöscht werden. Die Strategie über den Anwendungs-Manager von Android alle Daten einer App zu löschen ergab, dass diese das Verzeichnis der Applikation sozusagen nur zurücksetzt und nötige Datenbanken neu erstellt. Die zuvor hier gesicherten Dateien verbleiben jedoch auf dem Speicher. Betrachtet man die Ergebnisse der Untersuchungen beim Löschen der Cachedateien einer Applikation ist zu erkennen, dass in einigen Fällen die komplette Datei gelöscht wird. In den meisten Fällen wird jedoch ein Rest der Datei nicht gelöscht. Ähnlich sieht es beim Deinstallieren einer App aus. Auch hier konnten Reste, der vor der Deinstallation gesicherten Benutzerdaten wiederhergestellt werden. Nur beim Löschen von Dateien konnte ein sicheres Löschen festgestellt werden.

Die Untersuchungen mit dem Samsung Galaxy S2 zeigten, dass zum sicheren Löschen die Mittel des Systems in keiner Weise ausreichen. Alle Tests ergaben, dass der Speicherbereich, welcher die Inhalte einer betrachteten Datei enthielt, unverändert blieb. Somit war eine Wiederherstellung der Testdaten möglich. Des Weiteren konnte festgestellt werden, dass sich die Änderungen in den Inodes beim Löschen einer Datei nicht auf die Verweise selbst beziehen. Das heißt, wenn der Inode vor dem Löschen bekannt ist, kann der Verweis zu den „gelöschten“ Daten extrahiert werden. Dies ist selbstverständlich nur solange möglich, bis der Inode für die Referenz einer anderen Datei verwendet und damit überschrieben wird.

Die Ergebnisse zeigen das es stark vom Endgerät abhängt, ob Daten sicher gelöscht werden können. Um eine Lösung zu erhalten, welche unabhängig von der verwendeten Hardware ist, wird im Folgenden ein Konzept vorgestellt, welches ein sicheres Löschen von Benutzerdaten ermöglicht.

5.4 Konzept zum sicheren Löschen

Das hier vorgestellte Konzept basiert auf der folgenden Überlegung. Wenn die zu entfernenden Benutzerdaten gelöscht werden und anschließend der freie Speicher der entsprechenden Partition mit Dateien gefüllt wird, bis er komplett beschrieben ist, kann die gelöschte Datei nicht wiederhergestellt werden. Das Konzept lässt sich somit in zwei Schritte unterteilen, das Löschen und das Füllen.

Das Löschen der Dateien wird über eine Bibliothek festegelegt. Diese Bibliothek enthält Listen die bestimmten Benutzerdaten zugeordnet werden. Das heißt, es können entweder alle Dateien, die von einer Applikation erstellt wurden (siehe Kapitel 2.1.2), gelöscht werden oder es existiert eine spezielle Liste. Um vom Benutzer erstellte Dateien zu berücksichtigen, beinhaltet die Bibliothek dafür den Pfad der emulierten externen SD Karte. Dadurch ist es möglich alle, der Bibliothek bekannten Benutzerdaten zu löschen.

Die Inodes und die Dateinamen aller ausgewählten Dateien werden gesichert. So kann der Speicherbereich der entsprechenden Dateien sowohl vor als auch nach dem Löschen ermittelt werden. Einerseits kann damit das Löschen selbst vorgenommen werden, andererseits kann im Nachhinein überprüft werden, ob der Speicherbereich tatsächlich gelöscht wurde.

Jedes Objekt dieser Liste wird folgendermaßen verwendet. Mittels Dateiname kann eine Referenz auf die zu löschende Datei erzeugt werden. Über diese wird geprüft, ob eine Datei mit diesem Namen existiert und welche Speicherkapazität diese besitzt. Im Anschluss kann diese mithilfe des Dateisystems überschrieben werden. Daraufhin wird die nun unbrauchbare und nicht mehr lesbare Datei gelöscht.

Bei einem NAND-Datenträger kann das Überschreiben der Datei weggelassen werden. Wie in Kapitel 2.2.1 beschrieben kann ein bereits beschriebener Block nicht überschrieben werden. Somit würde der Versuch eine Datei zu überschreiben, das Erstellen einer neuen Datei ergeben. Abbildung 5-17 zeigt, dass ein einfaches Löschen in diesem Fall ausreicht und zudem performanter ist.

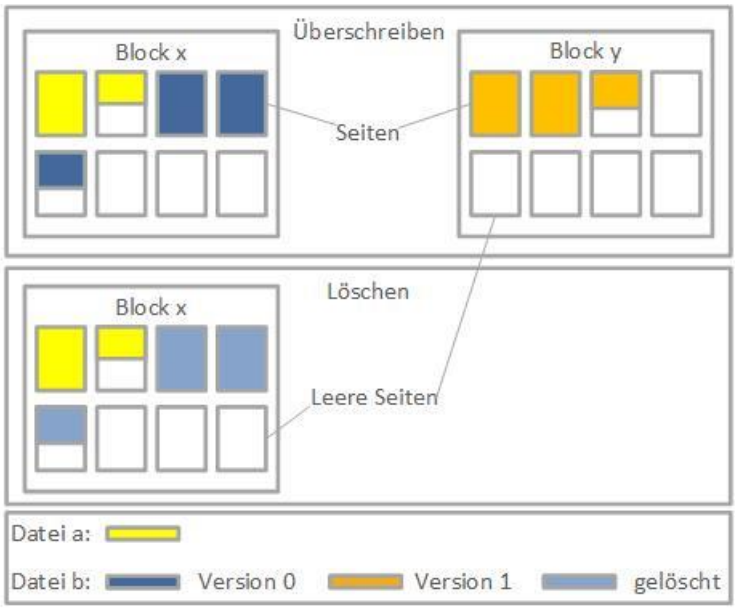


Abbildung 5-17 Der Vergleich eines Lösch- und Überschreibvorgangs bei einem NAND-Datenträger

Das Löschen ändert die Meta-Informationen der betroffenen Datei. Dadurch wird aus dem Dateiinhalt freier Speicherplatz. Beim Überschreiben geschieht das ebenfalls, jedoch auf einem anderen Weg. Der Inode, welcher zuvor auf die zu löschende Datei wies, referenziert nun eine neue Datei, an einer zuvor freien Stelle des Speichers.

Die bisher durchgeführten Schritte führen im Falle einer vom Benutzer erstellten Datei dazu, dass ihr Speicherbereich wieder frei gegeben wird. Bei einer Applikation wird diese in den Zustand vor der ersten Nutzung versetzt, da keine gesicherten Daten mehr vorliegen. Benutzerdaten einer Applikation sind ausschließlich auf dem persistenten Speicher des Endgerätes gesichert. Bei einem Neustart muss die Applikation diese zunächst wieder laden. Liegen keine Benutzerdaten vor, wird die App wie bei der ersten Nutzung ausgeführt.

Füllen Methode 1

Um alle gelöschten Dateien unwiderruflich zu entfernen, wird der gesamte freie Speicher der Partition beschrieben. Damit kann sichergestellt werden, dass keine ältere Version einer gelöschten Datei oder die gelöschte Datei selbst, zurückbleiben.

Zunächst wird der vorhandene freie Speicherplatz der Benutzerdaten-Partition und das verwendete Dateisystem ermittelt. Wie in Kapitel 2.1.1 beschrieben kann für die Erfassung aller wichtigen Benutzerdaten nur die Benutzerdaten-Partition betrachtet werden. Durch den ermittelten Wert kann eine Einschätzung der zu erzeugenden

Dateien gemacht werden, welche den Speicherbereich füllen. Der Inhalt dieser Dateien besteht ausschließlich aus Nullen. Die Größe der Dateien wird über den Divisor der Gesamtkapazität bestimmt.

Bei den Dateisystemen ist zu beachten, dass unter FAT die Dateigröße unter vier Gigabyte bleibt, da das die maximal zu speicherende Dateigröße ist [Car05]. Unter EXT4 liegt sie bei 2 Terabyte [Mat07] und ist daher vernachlässigbar. Um die Dateien möglichst groß werden zu lassen wird der Divisor zunächst klein gewählt und anschließend nach oben hin korrigiert.

Nachdem die Größe feststeht werden alle Dateien in einem expliziten Verzeichnis erstellt. Somit kann im Anschluss dieses Verzeichnis komplett gelöscht werden um den Speicherbereich, der nun ausschließlich mit Nullen beschrieben ist wieder freizugeben.

Damit wurde erreicht, ein sicheres Löschen durchzuführen. Allerdings erzeugt dieses Vorgehen eine hohe Abnutzung [23] des NAND-Datenträgers. Um dies zu verhindern wird durch das folgende, aufwendigere Verfahren nur der nötigen Speicherbereich beschrieben.

Füllen Methode 2

Dieses Verfahren bezieht sich auf das bei aktuellen Android Smartphones übliche Dateisystem EXT4. Es wird zunächst eine Datei erstellt, die der Blockgröße des Dateisystems entspricht. Anschließend wird für jede Blockgruppe der Benutzerdaten-Partition folgende Prozedur durchgeführt. Aus der Databitmap[24] wird ermittelt welche Blöcke nicht alloziert sind. Diese Blöcke werden nach einander mit der erstellten Datei verglichen. Sind diese gleich, wird der Block als alloziert markiert und es wird fortgefahren anderenfalls wird der Block überschrieben und ebenfalls als belegt markiert. Beide Markierung müssen in einer Liste gesichert werden. Nachdem der gesamte freie Speicherbereich behandelt wurde, werden alle gemerkten Markierung wieder entfernt, um den Speicherbereich freizugeben.

Des Ergebnis ist dem der Methode 1 identisch jedoch wird der Datenträger geschont. Der Grad der Schonung hängt Dabei von der Größe des unbeschriebenen

[23] Wie im Kapitel 4.2.1 beschrieben, entsteht durch die Nutzung eines NAND-Datenträgers eine Abnutzung, welche seine Lebensdauer verkürzt.

[24] Ein Bereich der in jeder Blockgruppe den Zustand der Allokation für jeden Block beinhaltet.

Speicherbereiches ab. Je mehr Speicher unberührt ist desto weniger Abnutzung findet durch das Füllen statt. Im Gegensatz zu Methode 1 ist diese Methode sehr viel aufwendiger. Abbildung 5-18 zeigt den Algorithmus zu Methode 2.

```
------------------------------------------------------------------------
Eingabe : Ext4 Partition, Datei x (Inhalt:Nullen, Größe:FS-Blockgröße)
Ergebnis: Der gesamte freie Speicher ist mit NUllen beschrieben
------------------------------------------------------------------------

1. Vergleiche jeden nicht allozierten FS-Block mit x
   und sichere FS-Blocknummer.

2. Wenn wahr, dann FS-Block allozieren
   sonst, überschreibe FS-Block mit x.

3. Setze gemachte Allokationen zurück.

------------------------------------------------------------------------
```

Abbildung 5-18 Algorithmus zu Methode 2 des Füllens

Fragmente einer sicher zu löschenden Datei werden beim Füllen ebenfalls entfernt, wenn sie auf einem Block mit noch benötigtem Inhalt gesichert sind. Wie im Kapitel 4.2 beschrieben, speichert der Garbage Collector benötigte Seiten an anderer Stelle und macht den freigewordenen Block wieder beschreibbar. Das Problem Fragmente einer gelöschten Datei im Fileslack einer nicht gelöschten Datei wiederherstellen zu können besteht hier ebenfalls nicht. Da eine Seite zunächst gelöscht sein muss, bevor sie beschrieben werden kann, entsteht erst gar kein Fileslack.

Das Prinzip des vorgestellten Konzeptes ermöglicht es, ein sicheres Löschen unabhängig vom Typ des Endgerätes durchzuführen. Bei der Umsetzung kann eine Lösung angestrebt werden, die es dem Nutzer ermöglicht Benutzerdaten auszusuchen, die er als sicher gelöscht wissen möchte. Danach wird beim Prozess des Füllens sichergestellt, dass es unmöglich ist diese gewählten Daten wiederherzustellen.

Im nächsten Kapitel wird das Prinzip des Löschens von Benutzerdaten einer Applikation und das Füllen nach Methode 1 implementiert und evaluiert. Die Applikation ist dabei die in dieser Arbeit durchweg als Beispiel herangezogene com.android.email.

6 Implementierung und Evaluierung der Applikation

In diesem Kapitel wird die Umsetzung eines Teils des vorgestellten Konzeptes, aus dem vorhergehenden Kapitel beschrieben. Es wurde eine Lösung in Form einer Applikation auf dem Android System implementiert. Die Struktur dieser Applikation macht den Inhalt des ersten Abschnittes in diesem Kapitel aus. Im Anschluss wird auf der Grundlage der bisherigen Untersuchungen eine Evaluierung der Software durchgeführt. Das heißt, es wird nach dem gleichen Schema vorgegangen, wie bei den Tests zur Untersuchung der Sicherheit von Löschstrategien im Kapitel 5.2. Dadurch kann verglichen werden, wie sich die Auswirkungen beim Löschen von Daten unterscheiden. Des Weiteren dient das zur Verfügung stehende Endgeräte Samsung Galaxy S2 dazu, das sichere Löschen der zugrunde liegenden Applikation zu testen. Da die Untersuchungen aus Kapitel 5.2.2 ergaben, dass kein sicheres Löschen mit den zur Verfügung stehenden Mitteln möglich ist, wurde dieses Endgerät für die Evaluierung ausgewählt.

6.1 Inhalt und Struktur der Applikation

Die Applikation umfasst das Löschen von Benutzerdaten, welche durch die Nutzung einer Applikation erzeugt wurden. Dabei wurde das sichere Löschen der Daten für die zuvor in dieser Arbeit als Beispiel herangezogene App com.android.email [25] umgesetzt. Des Weiteren ist es möglich die standardmäßigen Daten, wie Datenbanken, Cachedateien und Einstellungen für jede beliebige installierte Applikation auf dem verwendeten Endgerät sicher zu löschen[26].

Das Füllen des Datenträgers ist ebenfalls eine Funktion der hier beschrieben Applikation. Der Anwender kann diese Funktion ausführen, um sicher zu gehen, dass sich keine verwaisten Daten auf der Data Partition befinden.

Die Applikation besteht aus vier Klassen, welche im Folgenden einzeln genauer beschreiben werden.

Die Klasse Home_Activity liefert die Benutzeroberfläche, die beim Starten der Applikation angezeigt wird. Hier hat der Nutzer die Möglichkeit eine Liste aller installierten Applikationen, mittles Aktionbutton anzuzeigen. Des Weiteren wird die

[25] Vgl.Seite 56
[26] Vgl.Seite 18

Funktion zum Füllen des Speichers über einen Button auf dem Hintergrundlayout zur Verfügung gestellt. Außerdem wird beim Starten sichergestellt, dass eine Applikationsbibliothek im App-Verzeichnis angelegt wird. In der Applikationsbibliothek befinden sich Listen mit den Pfaden aller Dateien, die Benutzerdaten enthalten können. Für diese Implementierung liegen hier die Liste für die App com.android.email und eine Weitere für Standardordner innerhalb eines Android Appverzeichnisses vor. Bei jedem Start der Applikation wird geprüft, ob diese Listen im Applikationsverzeichnis vorhanden sind. Ist das nicht der Fall, werden sie dorthin kopiert. Die Funktion dieser Dateien spielt in der Klasse Userdata_Activity eine Rolle, dazu mehr im entsprechenden Abschnitt.

Die Klasse Apps_Activity wird mittels Intent[27] von Home_Activity aufgerufen und erzeugt eine Liste aller auf dem Endgerät befindlichen Applikationen. Für jede Applikation wird der eindeutige Name ermittelt und beim Anwählen einer App per Intent an die Klasse Userdata_Activity gesendet. Der Benutzer ist somit in der Lage alle installierten Apps per Klick anzuwählen.

Die Klasse Userdata_Activity erhält einen Intent mit dem Namen der angewählten Applikation von der Klasse Apps_Activity. Anhand dieser Information werden die folgenden zwei zusätzlichen Informationen zur ausgewählten App ermittelt und ausgegeben. Die User ID, welche eine Applikation eindeutig referenziert und der Pfad des Applikations-Verzeichnisses. Des Weiteren werden die Benutzerdaten dieser Applikation ausgegeben, wobei die bereits erwähnte Applikationsbibliothek zum Tragen kommt. Anhand der übermittelten Daten zu der ausgewählten App wird die Bibliothek auf Informationen zu Benutzerdaten durchsucht. Liegen Informationen vor, wie im Fall der App com.android.email, werden die dort hinterlegten Pfade für die weitere Verarbeitung in einer Liste gespeichert und zurückgegeben. Existiert keine spezielle Bibliotheksdatei für die vom Benutzer gewählte App wird die Standarddatei ausgewählt. Dadurch werden die Standardverzeichnisse Cache, Database und Shared_prefs zurückgegeben. Die ermittelten Pfade werden im Anschluss nach ihrem Inhalt durchsucht. Die gefunden Dateien werden in einer Liste gesichert und zurückgegeben. Das Ergebnis ist eine Liste von Dateien, die auf der Grundlage der in

[27] Android Klasse, die es ermöglicht andere Klassen anzusprechen und Informationen zu übermitteln [Mei12].

der Bibliothek angegebenen Pfade, alle Benutzerdaten dieser Applikation enthält. Diese Liste wird nun mit einer Methode verarbeitet, die dafür sorgt, dass die einzelnen Dateien gelöscht werden. Der Kopf der Methode sieht folgendermaßen aus:

```
userdata(String app, ArrayList<String> folder, boolean delete)
```

Durch den Parameter *delete* ist es zum einen, durch den Wert false, möglich die Dateien anzuzeigen, zum anderen wird, durch den Wert true, das Löschen aller Dateien initiiert. Diese Lösung macht es möglich, die gleiche Methode für das Anzeigen und das Löschen verwenden zu können. Des Weiteren wird dem Nutzer dadurch, mittles Button, das Ausführen des Löschvorganges der ermittelten Dateien überlassen. Beim Klicken dieses Buttons wird die Methode userdata mit dem Parameter delete gleich true ausgeführt. Dadurch wird eine weitere Methode ausgeführt, die deleteFile heißt. Die zuvor ermittelten Pfade und Dateien werden an diese Methode übergeben. Diese wiederum löst, über einen Prozess in der Kommandozeile, das Löschen der entsprechenden Datei aus. Zunächst wird zu dem entsprechenden Pfad navigiert. Anschließend werden die Rechte für einen Zugriff auf die betroffene Datei manipuliert, um diese schließlich löschen zu können.

Nachdem alle Dateien auf die beschriebene Weise gelöscht wurden, muss ein Neustart des Endgerätes durchgeführt werden, um einen Absturz des Systems beim Ausführen einer betroffenen App zu verhindern. Der Grund dafür liegt in den nun nicht mehr lesbaren Dateien. Ein Neustart bewirkt, dass die Dateien zum Sichern der Applikationsdaten neu geladen werden müssen. Liegen sie nicht vor, werden sie neu erstellt.

Die Klasse Fill_Activity wird durch einen Intent aus der Home_Activity Klasse ausgeführt und hat keine eigene Anzeige. Sie veranlasst, dass der freie interne Speicher des Endgerätes vollkommen beschrieben wird. Dafür wird mittels der Methode fillAndErase(), der zur Verfügung stehende Speicherplatz in der Data Partition vollkommen befüllt. Dafür werden Dateien, deren Inhalte nur aus Nullen bestehen im eigens dafür erstellten Ordner fill erzeugt. Im Anschluss wird der Inhalt dieses Ordners gelöscht. Wobei durch das Löschen der überschriebene Speicherbereich wieder freigegeben wird. Dieser Vorgang kann abhängig vom freien Speicher ein wenig Zeit in Anspruch nehmen. Nachdem der komplette Vorgang beendet ist, wird auf dem Display erneut das Layout der Klasse Home_Activity angezeigt. Abbildung 6-1 zeigt eine Übersicht der beschriebenen Applikation SLamE.

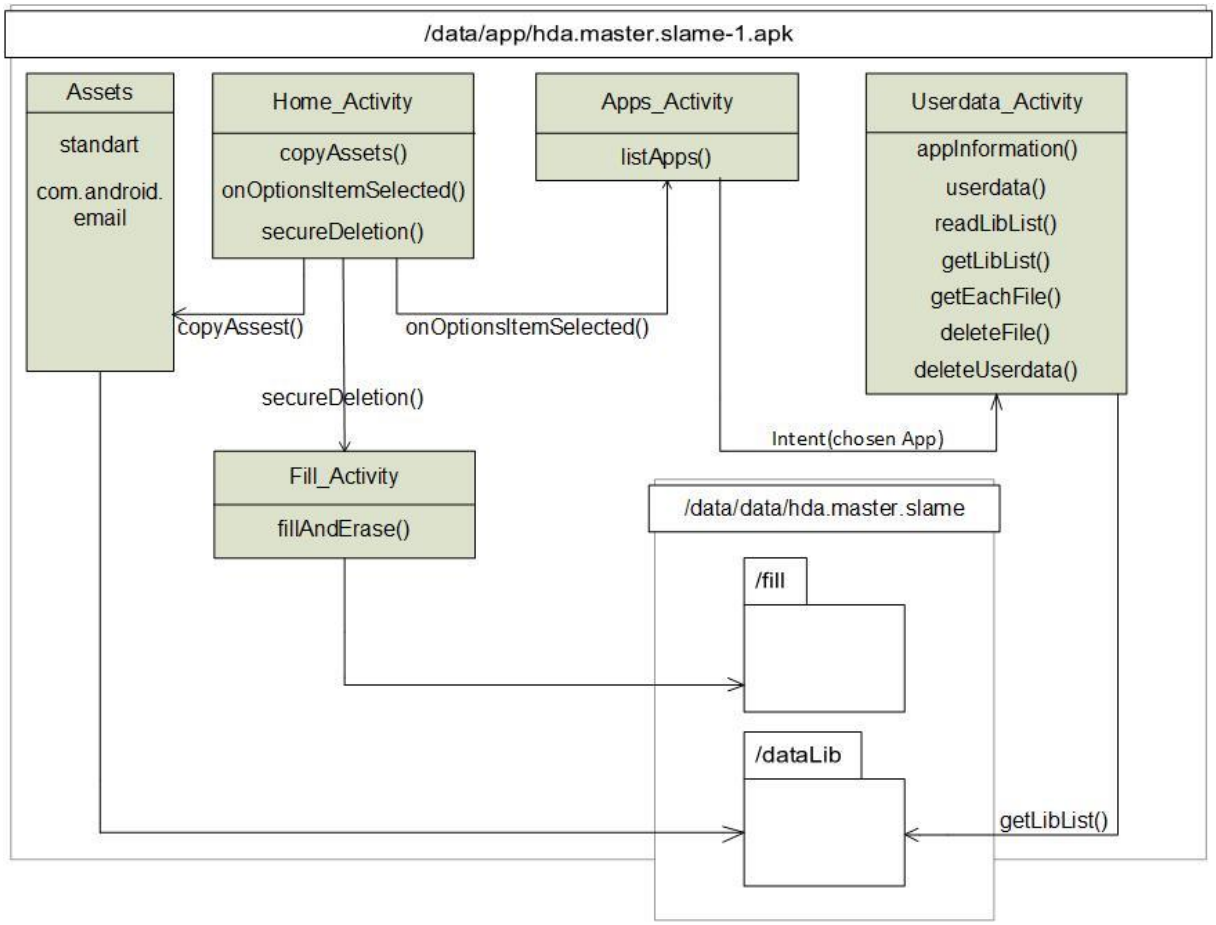


Abbildung 6-1 Übersicht zur Applikation SLamE

6.2 Evaluierung der Software

Die Evaluierung der Software wird am Beispiel der Applikation com.android.email durchgeführt. Wie im Kapitel 6.1 beschrieben, existiert dafür eine Liste von Pfaden in denen Benutzerdaten dieser Applikation gesichert werden. Die Applikation wurde durch die Eingabe von Accountdaten eines E-Mailproviders mit Benutzerdaten versehen. Die Applikation sichert dadurch automatisch alle, in diesem Account vorhanden, E-Mails im entsprechenden Applikationsverzeichnis. Anschließend wurde eine exakte Kopie der Benutzerdaten-Partition erzeugt, um den Zustand vor dem Löschvorgang festzuhalten. Daraufhin wurde mit der Applikation SLamE die entsprechende App ausgewählt und ein sicheres Löschen durchgeführt. Zuletzt wurde erneut eine exakte Kopie der Benutzerdaten-Partition erzeugt. Dadurch kann mittels Vergleich von bestimmten Speicherbereichen aus beiden Partitionsabbildern gezeigt

werden, ob Dateinhalte nach dem Löschen wiederhergestellt werden können oder nicht.

6.2.1 Ergebnis

Der Test zeigt, dass mittels Anwendung der SLamE App auf die E-Mail-Applikation com.android.email, alle Benutzerdaten, die durch das Erstellen des E-Mail-Accounts auf den Datenträger geladen wurden, sicher und damit unwiderruflich gelöscht wurden. Durch die spezielle Liste aus dem dataLib Verzeichnis, für diese App werden alle Dateien, die Benutzerdaten enthalten berücksichtigt.

Wie in den Untersuchungen im Kapitel 5.2 wurden die Inodes der entsprechenden Dateien aus dem Partitionsabbild extrahiert. SLamE listet für die Pfade cache, cache/1.db_att, databases, und shared_prefs die gefundenen Dateien auf. Die Speicherbereiche der dort aufgeführten Fragemente[28] einer jeden Datei wurden mit denen nach dem Ausführen der Anwendung verglichen. Das Ergebnis ist, dass der zuvor gesicherte Dateiinhalt, im Anschluss mit Nullen überschrieben ist. Abbildung 6-2 zeigt, dass das Problem mit nicht gelöschten Dateiresten[29] nicht auftritt. Auch sehr kleine Dateien, wie eine 295 Byte große XML-Datei wurden entfernt.

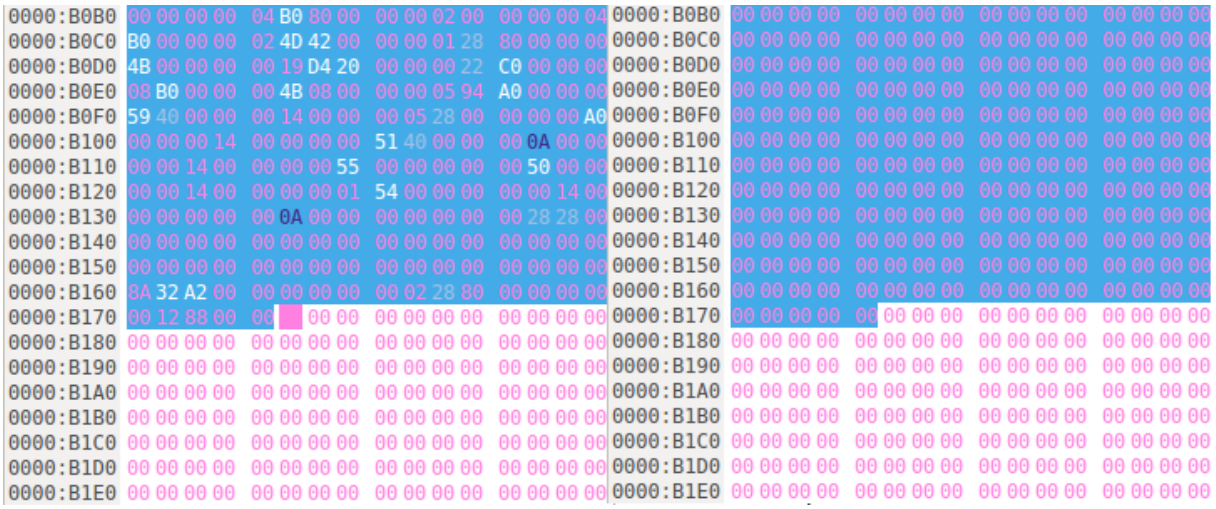


Abbildung 6-2 Dateiende einer PDF-Datei aus dem Cacheverzeichnis vor (links) und nach (rechts) dem Löschen durch die App SLamE

[28] Vgl. Seite 42
[29] Vgl Abbildung 5-12

6.3 Zusammenfassung

Dieses Kapitel präsentierte die Umsetzung einer Lösung zum sicheren Löschen von Benutzerdaten durch eine Android Applikation. Diese ermöglicht es dem Nutzer, im laufenden Betrieb des Endgerätes, installierte Applikationen anzuwählen, sich eine Liste der durch sie erzeugten Dateien ausgeben zu lassen und diese zu löschen. Durch die Ausführung der Option sicher Löschen ist es dem Nutzer der Applikation SlamE möglich, zu jeder Zeit alle gelöschten Inhalte auf dem Datenträger unwiderruflich zu entfernen.

Es wurde eine Evaluierung diese Software auf dem Android Smartphone Samsung Galaxy S2 GT-I9100 durchgeführt. Dabei wurden alle, durch die Testapplikation com.android.email erzeugten Daten, sicher und unwiderruflich gelöscht.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Lösung erarbeitet, die es ermöglicht Benutzerdaten auf einem Android Smartphone sicher zu löschen. Es wurde gezeigt, welche Dateien als Benutzerdaten interpretiert werden können und an welchen Stellen auf dem Datenträger eines Android-Gerätes diese zu finden sind. Die Verteilung von Dateien innerhalb des internen Speichers, durch Datenträgereigenschaften, konnte als irrelevant im Bezug auf die vorgestellte Lösung aufgetan werden. Im Falle eines NAND-Datenträgers, der das Überschreiben von Blöcken zulässt, kann hingegen ein direktes Löschen von Dateien umgesetzt werden. Unter Berücksichtigung der Verteilung innerhalb des Lebenszyklus einer Datei kann in diesem Fall ein direktes Löschen durchgeführt werden. Die gezeigt Lösung kann unabhängig vom Datenträgertyp angewendet werden. Der Grund dafür ist, dass jeder Datenträger beim Löschen von Inhalten auf seinem Speicher, wenigstens den betroffenen Bereich als frei für neue Inhalte markiert.

Um die Notwendigkeit von externer Software in Form einer App zu begründen, wurde für die Referenzgeräte Samsung Galaxy S3 und Samsung Galaxy S2 gezeigt, welche Möglichkeiten das jeweilige System bietet, um erzeugte Benutzerdaten zu löschen. Des Weiteren wurde die Sicherheit dieser Möglichkeiten aufgezeigt. Ein sicheres Löschen konnte nur auf dem Gerät Galaxy S3 festgestellt werden. Durch das Ausführen des Werksresets löscht dieses Smartphone alle auf der Benutzerdaten-Partition gesicherten Inhalte, und zwar unwiederbringlich. Im laufenden System konnte nur für die Option des manuellen Löschens, von den vom Nutzer gesicherten Dateien, ein sicheres Löschen festgestellt werden. Da der Nutzer jedoch nicht die Rechte besitzt, jede Datei auf der Benutzerdaten-Partition zu löschen, ist dies nicht ausreichend. Das Samsung Galaxy S2 hingegen bringt keine Option mit, ein sicheres Löschen durchzuführen. Alle durchgeführten Tests führten zu dem Ergebnis, dass gelöschte Dateien wiederhergestellt werden können.

Das vorgestellte Konzept bietet die Möglichkeit, auf beiden Referenzgeräten ein sicheres Löschen im laufenden Betrieb durchzuführen. Mehr noch kann der Nutzer wählen, welche seiner erzeugten Daten gelöscht werden sollen. Die Umsetzung für die Android Version 4 bietet das sichere Löschen von Benutzerdaten innerhalb von Applikationen an. Ein gezieltes Entfernen von Dateien, die Benutzerdaten enthalten, wurde speziell für die Applikation com.android.email umgesetzt. Dies kann für jede

beliebige Applikation erweitert werden. Dazu muss eine Liste hinzugefügt werden, welche alle Pfade zu Benutzerdaten innerhalb der Applikation enthält. Für den Fall, dass keine solche Liste vorliegt, werden die Android üblichen Ordner cache, databases und shared_prefs innerhalb des Applikationsverzeichnisses nach Dateien durchsucht. Somit können Benutzerdaten einer beliebigen Applikation auf einem Android System sicher gelöscht werden.

Änderungen, die durch jede neue Version von Android einhergehen, liefern stets neue Ansätze für die Arbeit auf dem hier behandelten Gebiet. Beispielsweise, dass neue Technologien die Möglichkeit mit sich bringen, neue Arten von Benutzerdaten zu erzeugen. Auch die Weiterentwicklung von NAND-Datenträgern liefert eventuell die Möglichkeit, eine performantere Lösung für mobile Endgeräte im Bezug auf sicheres Löschen im laufenden Betrieb umzusetzen. Wenn die Kapazität von Datenträgern zunimmt wird das Konzept des Füllens eine langwierige Option zum sicheren Löschen sein.

Eine Umsetzung des vorgestellten Konzeptes, ohne die Nutzung von Superuser Rechten würde eine Lösung für jedes beliebige Endgerät liefern. Mehr noch könnte es auf Endgeräten mit anderen Betreibsystemen, wie das der Firma Apple unter iOS umgesetzt werden.

Literaturverzeichnis

[Adb13] Developer, Andoird: *Andoird Developers Blog.* http://android-developers.blogspot.de/, zuletzt besucht am 15. Juni 2013.

[Add13] Developer, Android: *Dashboards.* http://developer.android.com/about/dashboards/index.html, erstellt am: 21. Juni 2013, zuletzt besucht am 21. Juni 2013.

[AdD13] Developers, Android: *Dokumentation android.drm.* http://developer.android.com/reference/android/drm/package-summary.html, zuletzt besucht am 12. Mai 2013.

[Ade13] Developer, Android: *App Install Location.* http://developer.android.com/guide/topics/data/install-location.html, zuletzt besucht am 27. Juli 2013.

[Ado13] Developer, Android: *Storage Options.* http://developer.android.com/guide/topics/data/data-storage.html, zuletzt besucht am 23. April 2013.

[Ads13] Developers, Android: *The Android Source Code.* http://source.android.com/source/downloading.html, zuletzt besucht am 21. Juni 2013.

[Anv13] Developer, Android: *Using Hardware Devices.* http://developer.android.com/tools/device.html#VendorIds, zuletzt besucht am 07. Mai 2013.

[Bit08] BITKOM: *Leitfaden zum Sicheren Löschen Version 2.0.* Berlin: BITKOM 2008.

[Car05] Carrier, B.: *File System Forensic Analysis.* 9th ed. Boston: Addision-Wesley, 2005.

[Chs12] Ching-Che, C.; Ning-Mi, H.: *A Low-Complexity High-Performance Wear-Leveling.* Taiwan: Circuits and Systems, 2012 IEEE, Asia Pacific Conference, S. 595 - 598.

[Cob13] Cobbaut, P.: *Linux Fundamentals.* http://linux-training.be/files/books/LinuxFun.pdf, erstellt am: 05. Mai 2013, zuletzt besucht am 2013. Juli 05.

[Cwe12] Chundong, W.; Weng-Fai, W.: *Observational Wear Leveling.* San Francisco, California, USA.: 2012 ACM/EDAC/IEEE, Design Automation Conference, S. 253 - 242.

[Ext13] Wiki, Ext4: *Ext4 Disk Layout.* https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Inline_Data, zuletzt besucht am 17. September 2013.

[Fox09] Fox, D.: *Sicheres Löschen von Daten auf Festplatten. In: Datenschutz und Datensicherheit (2009) 2, S. 110 - 113.*

[Gut96] Gutmann, P.: *Secure Deletion of Data from Magnetic and Solid-State Memory.* Auckland: 1996 Sixth USENIX Security Symposium Proceedings, San Jose, S. 77 - 90.

[Hei13] Online, Heise: *Heise Glossar.* http://www.heise.de/glossar/entry/Cache-395216.html, zuletzt besucht am 19. September 2013.

[Hoo12] Hoog, A.: *Android Forensik Datenrecherche, Analyse und mobile Sicherheit bei Android.* München: Franzis Verlag GmbH, 2012.

[Hwa13] Hwaci, Applied Software Research: *The SQLite Database File Format.* http://www.sqlite.org/fileformat2.html#database_header, zuletzt besucht am 20. Juli 2013.

[Idc13] IDC: *Top Five Smartphone Operating Systems.* http://www.idc.com/getdoc.jsp?containerId=prUS24108913, erstellt im: Mai 2013, zuletzt besucht am 18. Juni 2013.

[Ilh12] Ilhoon, S.: *Implementing Secure File Deletion in NAND-based Block Devices with Internal Buffers.* IEEE 2012, Transactions on Consumer Electronics, Vol. 58, No. 4, S. 1219 - 1224.

[Kal13] Kalhammer, F.: *Linux-Praxis.de.* http://www.linux-praxis.de/lpic1/lpi101/1.104.7.html, zuletzt besucht am 17. September 2013.

[Mat07] Mathur, A.; Cao, M.; Bhattachary, S.: *The new ext4 Filesystem: current status and future plans.* Ottawa, Ontario Canada 2007: IBM, Linux Symposium, Volume Two, S. 21 - 34.

[Mcm10] Micheloni, R.; Crippa, L.; Marelli, A.: *Inside NAND Flash Memories.* Dordrecht Heidelberg London New York: Springer, 2010.

[Mei12] Meier, R.: *Profesional Android 4 Applikation Development.* Indianapolis, Indiana: John Wiley & Sons, Inc., 2012.

[Mos13] Mobile Studien: *Verkaufszahlen/Marktanteile Q1 2013.* http://mobile-studien.de/marktanteile-endgeraete/q1-2013/, zuletzt besucht am 18. Juni 2013.

[Net13] Netzwelt GmbH*: Netzwelt Wissen.* http://www.netzwelt.de/news/101321-netzwelt-wissen-imei-nummer.html, zuletzt besucht am 01. September 2013.

[Rbc13] Reardon, J.; Basin, D.; Capkun, S.: *SoK: Secure Data Deletion.* Zürich: 2013 IEEE, Symposium on Security and Privacy, S. 301 - 315.

[Sam13] Samsung,: *Samsung Semiconductors.* http://www.samsung.com/global/business/semiconductor/product/flash-emmc/detail?productId=7639&iaId=2324, zuletzt besucht am 02. September 2013.

[Scl08] Kyoungmoon, S.; Jongmoo, C.; Donghee, L.; Noh, S.: *Models and Design of an Adaptive Hybrid Scheme for Secure Deletion of Data in Consumer Electronics.* 2008 IEEE, Transactions on Consumer Electronics, Vol. 54, No. 1 S. 100 - 104.

[Sub09] S.Subha,: *An Algorithm for Secure Deletion in Flash Memories.* CA, USA: 2009 IEEE, Sixth International Conference on Information Technology: New Generations, S. 1705 - 1706.

[Ywd13] Yi, Q.; Wei, T.; Dan, F.; Jingning, L.; Hu, Y.; Zhiming, Z.: *Per-File Secure Deletion Combining with Enhanced.* Berlin Heidelberg, Springer-Verlag 2013, S. 509-516.