Flash-Dateisysteme im Kontext der digitalen Forensik

Harald Baier und Matthias Deutschmann

Illegale Handlungen, die Geräte des Internets der Dinge (IoT) oder Embedded Devices wie Drohnen nutzen, nehmen stetig zu. Eine weit verbreitete Klasse von Dateisystemen zur Datenspeicherung auf eingebetteten Geräten ist die Klasse der Flash-Dateisysteme. Ein solches Flash File System (FFS) ist optimiert, um konzeptionelle Einschränkungen und Charakteristika von Flash-Speichern direkt zu verwalten, d.h. der Flash-Speicher wird nicht über einen zusätzlichen Hardware-Controller angesprochen, der einen Flash Translation Layer (FTL) bereitstellt. FTLs sind in verschiedenen Peripheriegeräten vorhanden, einschließlich SSDs, SD/MMC-Karten und eMMC-Chips.

Ein FFS enthält daher Mechanismen und Strukturen, die nicht Teil traditioneller blockbasierter Dateisysteme wie NTFS, APFS oder ExtX sind. Im Gegensatz zu FTLs, die ein solches Dateisystem obendrauf benötigen, sind FFS ausdrücklich für die Verwendung auf rohen Flash-Geräten konzipiert, wodurch die zusätzliche Übersetzungsschicht nicht mehr notwendig ist [4, 11].

Axis Communications AB führte eines der ersten FFS unter dem Namen **Journaling Flash File System** (JFFS) im Jahr 1999 unter der GNU General Public License ein [7]. Heute gibt es eine vielfältige Auswahl an FFS, was durch die rasche Entwicklung der Speichertechnologie erforderlich wurde. Mittlerweile findet sich ein FFS in verschiedenen Verbraucherprodukten, von Smartphones und Tablets bis hin zu Digitalkameras. Ein wichtiges Beispiel für ein FFS ist **Yet Another Flash File System** (YAFFS) [19].

Das **Unsorted Block Images File System** (UBIFS) stellt eine neuere Entwicklung im Bereich der FFS dar und wurde erstmals 2008 im Linux-Kernel 2.6.27 eingeführt und seitdem kontinuierlich aktualisiert. Benchmarks zeigen, dass es mehrere Vorteile gegenüber anderen FFS aufweist, insbesondere in Bezug auf die Leistung [16]. Ähnlich wie YAFFS findet sich UBIFS in einer Vielzahl von Geräten, von Roboterstaubsaugern, Internet-Sicherheitskameras bis hin zu Drohnen oder Routern [14]. Darüber hinaus ist UBIFS ein integraler Bestandteil von OpenWRT, einem auf Linux basierenden Betriebssystem, das für eingebettete Geräte entwickelt wurde [17].

Bezüglich der Analyse von FFS-basierten eingebetteten Geräten werden daher Forensik-Tools benötigt, die ein FFS handhaben können. Leider sind die derzeit verfügbaren IT-forensischen Tools oft nicht in der Lage, FFS oder allgemein rohe Flash-Images zu analysieren. Das

Open-Source Tool UBI Forensic Toolkit (UBIFT) schließt diese Lücke für das weit verbreitete UBI-Flash-Dateisystem. UBIFT kann über das öffentliche Repository unter https://github.com/matthias-deu/ubift/ heruntergeladen werden. Das Konzept von UBIFT ist inspiriert von dem Konzept der IT-forensischen Standardsoftware The SleuthKit. UBIFT ist in der Lage, Dateisystemstrukturen wie den Verzeichnisbaum oder das UBIFS-Journal zu parsen, um gelöschte Dateien einschließlich der jeweiligen Metadaten wiederherzustellen.

Flash Dateisysteme

Im Gegensatz zu traditionellen Dateisystemen wie ExtX haben FFS spezifische Anforderungen, die sich aus den Einschränkungen und Abnutzungserscheinungen von Flash-Speicher ergeben [12, 13]. Genauer gesagt besteht Flash-Speicher aus mehreren Blöcken von Speicherzellen. Diese Blöcke sind die kleinste Einheit für das Löschen und werden daher als Löschblöcke (erase blocks) bezeichnet. Die Löschoperation ist für Flash einzigartig und findet sich nicht auf traditionellen Festplatten.

Ein Block besteht aus aufeinanderfolgenden Seiten (pages), die die kleinste Einheit für Schreib- und Leseoperationen darstellen. Seiten enthalten auch einen zusätzlichen Bereich, der als Out-of-Band (OOB)-Bereich bezeichnet wird. Sein Layout unterscheidet sich je nach Hersteller und beinhaltet Metadaten wie Fehlerkorrekturcodes (ECC) oder Indikatoren, ob der Block defekt ist. Flash-Blöcke nutzen sich mit der Zeit ab, und wenn ein Block defekt wird, kann er aufgrund der physischen Bedingungen des Flashs nicht mehr verwendet werden.

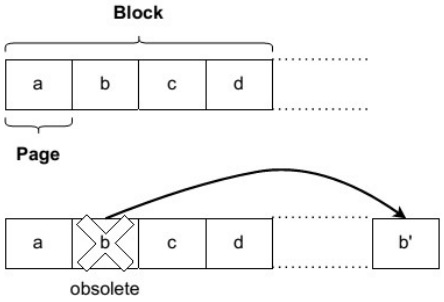

Abbildung 7: Out-of-Place Update Ansatz

Nicht-leere Seiten können aufgrund der physischen Einschränkungen nicht direkt lokal geändert werden, daher muss ihr gesamter entsprechender Block zuerst gelöscht werden, was dazu führt, dass alle Daten dieses Blocks verloren gehen, wenn sie nicht gespeichert werden. Ein naiver Ansatz, Daten vor dem Löschen zu speichern, besteht darin,

einen ganzen Block in einen Puffer zu lesen, den Inhalt einer bestimmten Seite zu ersetzen und alles zurück auf den Flash zu schreiben. Dieser Ansatz dauert ungefähr 100 Mal länger als das direkte Schreiben der aktualisierten Seite an einen anderen, leeren Platz [10]. Daher verwenden FFS einen Out-of-Place Update Ansatz, den wir in Abbildung 1 veranschaulichen. Seitenänderungen werden durchgeführt, indem aktualisierte Seiten an einen anderen, leeren Platz geschrieben werden, die alte Seite wird dann als veraltet markiert.

Dieser Mechanismus ist im Kontext der digitalen Forensik wichtig, da alte Seiten typischerweise nicht sofort gelöscht werden, sodass ihre Inhalte auch später ausgelesen werden können. Da der Gesamtspeicherplatz des Flash-Speichers aber begrenzt ist und die Anzahl veraltete Daten mit der Zeit zunimmt, nutzt ein FFS einen Garbage Collector (GC), der die Aufgabe hat, veraltete Seiten freizugeben. Die Implementierung eines GC ist spezifisch für ein einzelnes FFS, daher kann keine allgemeine Schlussfolgerung hinsichtlich der Wiederherstellbarkeit von Daten gezogen werden.

UBIFS

Abbildung 2 zeigt die UBIFS-Architektur innerhalb des Linux-Kernels. UBIFS stützt sich auf zusätzliche Schichten, die als **UBI** und **MTD** [15] bezeichnet werden.

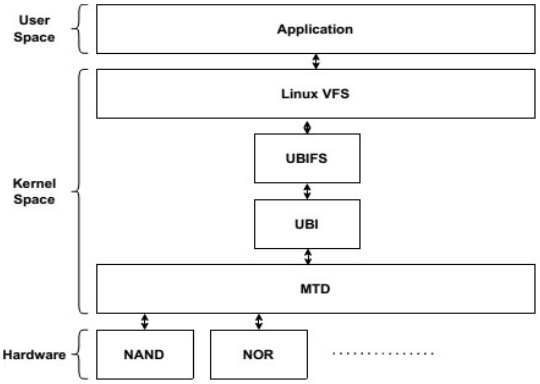

Abbildung 8: UBIFS Architektur [2]

Das **Memory Technology Device (MTD)** ist eine Abstraktionsschicht für rohe Flash-Geräte [15], sie ist als ein Linux-Subsystem implementiert. Ihr primäres Ziel ist es, eine generische Schnittstelle zwischen Hardware-Treibern und höheren Schichten innerhalb der Linux-Architektur bereitzustellen. Als solches bietet ein MTD-Gerät einheitlichen Zugriff auf den Flash-Speicher. MTD ermöglicht weiterhin die Partitionierung eines Geräts in mehrere, statische MTD-Partitionen.

Das **Unsorted Block Image (UBI)** ist eine Schicht über der MTD-Schicht [3, 18]. UBI bietet die Verwaltung von mehreren flexiblen Volumes, die aus mehreren aufeinanderfolgenden logischen Löschblöcken (LEB) bestehen. LEBs werden dynamisch auf physische Löschblöcke (PEB) innerhalb der MTD-Schicht abgebildet und bieten somit eine Adressübersetzungsfunktionalität. Verschiedene andere flash-bezogene Mechanismen wie Wear-Leveling, Bad-Block-Handling und Scrubbing sind ebenfalls innerhalb der UBI-Schicht implementiert.

Die Adressübersetzungsfunktion von UBI wird in Abbildung 3 gezeigt: LEBs innerhalb der UBI-Schicht werden dynamisch auf PEBs in der MTD-Schicht abgebildet. Defekte Blöcke (bad blocks) werden von UBI transparent gehandhabt, und sein Wear-Leveling-Mechanismus zielt darauf ab, Löschoperationen gleichmäßig über PEBs zu verteilen.

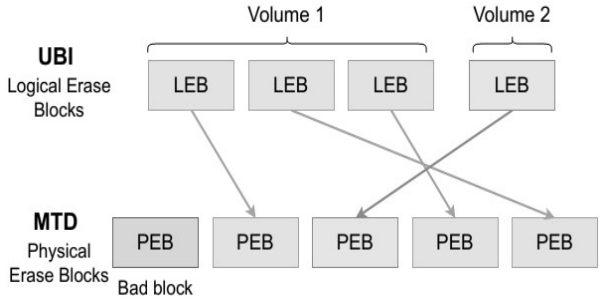

Abbildung 9: Zuordnung von LEBs zu PEBs in der UBI Schicht

UBI fügt einem PEB zwei zusätzliche Header hinzu, um seine Mechanismen zu implementieren (siehe Abbildung 4): einen Erase-Counter-Header (EC) und einen Volume-Identifier-Header (VID).

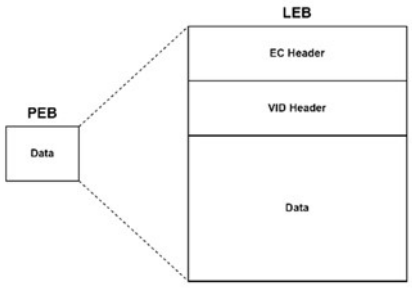

Abbildung 10: UBI Header

Der EC-Header wird vom Wear-Leveling-Subsystem von UBI zur ungefähr gleichmäßigen Allokation der Erase Blocks verwendet. Seine Magic Bytes mit dem Wert 0x55424923 (ASCII „UBI#“) sind von

besonderer Bedeutung, da sie den Beginn eines LEB markieren und somit die Identifizierung aller verfügbaren LEBs innerhalb eines Geräts ermöglichen. Weitere Informationen können dann aus dem zusätzlichen VID-Header abgeleitet werden, der eine Volume-ID (Datenfeld vol_id) speichert, um anzugeben, zu welchem Volume der LEB gehört. Weiterhin enthält der VID-Header eine LEB-Nummer (Datenfeld lnum), um die spezifische Zuordnung zwischen LEB und PEB anzuzeigen.

Ähnlich wie der EC-Header hat der VID-Header Magic Bytes mit dem Wert 0x55424921 (ASCII „UBI!“). Während der EC-Header immer verfügbar ist (es sei denn, es liegen fehlerhafte Bedingungen vor), kann der VID-Header fehlen, wenn er aktuell nicht einem UBI-Volume zugeordnet ist. UBIs Volumenverwaltung wird implementiert, indem ein reserviertes Volume mit der ID 0x7fffefff und dem Namen Layout-Volume bereitgestellt wird. Daher kann es identifiziert werden, indem nach LEBs gescannt und ihre vol_id verglichen wird. Auf diese Weise können alle verfügbaren UBI-Volumes identifiziert werden.

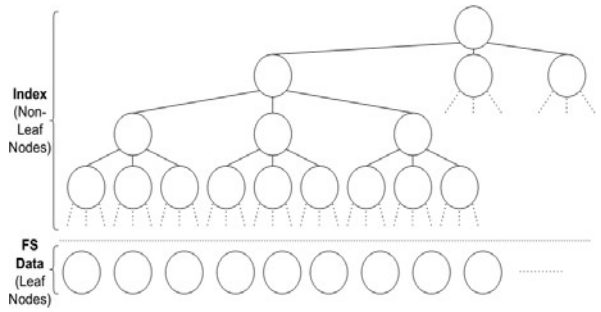

Abbildung 11: UBIFS B+-Baum

Das **UBI-Dateisystem (UBIFS)** baut auf der UBI-Schicht auf und stützt sich auf deren Mechanismen. Im Gegensatz zu seinem Vorgänger JFFS2 speichert UBIFS den Dateiindex auf dem Flash. Die Dateiindexstruktur wird durch einen B+-Baum realisiert, der eine spezifische Art von ausgeglichenem Suchbaum ist, die Anzahl der Kindknoten wird durch den Verzweigungsfaktor (Fanout) festgelegt. Abbildung 5 veranschaulicht die Struktur eines B+-Baums in UBIFS mit einem Fanout von drei.

Standardmäßig verwendet UBIFS einen Fanout von acht. Im Vergleich zu einem regulären B-Baum hält ein B+-Baum Daten ausschließlich in Blattknoten, die inneren Knoten (Indexierungsknoten) speichern Verweise auf die Kindknoten durch Schlüssel und Zeiger. UBIFS-Dateisystemobjekte können daher in den Blättern des B+-Baums gefunden werden.

UBIFS unterscheidet zwischen 14 verschiedenen Knotentypen. Jeder Knotentyp hat einen eigenen Zweck mit unterschiedlicher forensischer Relevanz. Alle Knoten haben einen gemeinsamen Header (ubifs_ch), der in seinem Feld magic die Magic Bytes mit dem Wert 0x06101831 enthält. Dies ermöglicht einen einfachen Scan-Ansatz: Jeder verfügbare Knoten kann gefunden werden, indem nach gemeinsamen Headern gesucht wird, der dann je nach konkretem Knotentyp (im Feld node_type gespeichert) weiterverarbeitet werden kann.

UBIFS hält sich an die Abstraktion des Linux Virtual File System (VFS) und bietet daher Metadaten in Form von Inodes an. Ein Inode wird von UBIFS in seinem ubifs_inode_node Knoten gespeichert, dort sind viele für die digitale Forensik relevante Artefakte wie MAC (Modification, Access, Change) Zeitstempel verfügbar. Bezüglich der Zeitstempel ist wichtig, dass der Access-Zeitstempel standardmäßig nicht geschrieben wird, um die Lebensdauer des Flashs zu verlängern. Obwohl der Zeitstempel nicht aktualisiert wird, hat ein Test gezeigt, dass er ursprünglich bei der Dateierstellung geschrieben wird, sodass er verwendet werden kann, um die Erstellungszeit einer Datei abzuleiten.

Neben dem Inode-Knoten gibt es mehrere andere forensisch relevante Knotentypen. Ein **Verzeichniseintragsknoten** (directory entry node, ubifs_dent_node) verknüpft eine Inode-Nummer mit einem Namen für eine Datei bzw. Verzeichnis. In UBIFS ist das Wurzelverzeichnis mit der Inode-Nummer 1 verknüpft. Im Allgemeinen sind zwei Inode-Nummern mit einem Dent-Knoten verknüpft: eine inum bezieht sich auf die Inode-Nummer der Datei (oder des Verzeichnisses), mit der der Dent-Knoten verknüpft ist, während die in den ersten 32 Bits des Schlüssels kodierte Inode-Nummer die Inode-Nummer des übergeordneten Verzeichnisses ist (was die rekursive Bestimmung des vollständigen Pfades ermöglicht).

Ein **Datenknoten** (data node, ubifs_data_node) ähnelt einem Datenblock, der mit einem Inode verknüpft ist. Standardmäßig kann ein einzelner Datenknoten die maximale Menge von 4096 Bytes speichern (definiert durch die Konstante UBIFS_BLOCK_SIZE in linux/fs/ubifs/ubifs-media.h). Ein **Masterknoten** (master node, ubifs_mst_node) gibt wichtige Positionen an, insbesondere die Position des Wurzelindexknotens des B+-Baums. UBIFS ist in Bereiche unterteilt, Abbildung 6 bietet einen Überblick über alle Bereiche. Für die digitale Forensik ist der Log-Bereich von besonderer Bedeutung, er ist Teil des Journals. Das Hauptziel des Journals ist es, die Anzahl der Schreibzugriffe auf den Flash zu reduzieren. Allgemein verwendet UBIFS einen wandernden Baummechanismus, um seinen Out-of-Place-Update-Mechanismus zu implementieren: Jedes Mal, wenn ein

Knoten innerhalb des Baums aktualisiert wird, wird er an einen anderen, leeren Platz geschrieben.

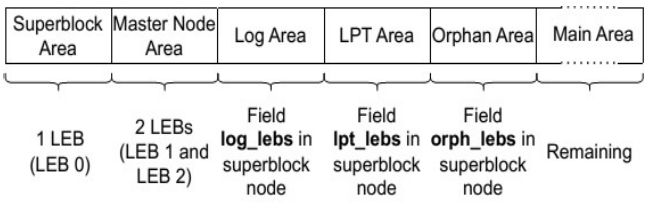

Abbildung 12: Bereiche von UBIFS

Da die Referenz des Elternknotens auf den Knoten nun veraltet ist, wird auch sie in ähnlicher Weise aktualisiert. Daher werden Knoten Out-of-Place aktualisiert, indem auch alle Knoten entlang ihrer Elternkette aktualisiert werden.

Um diese kaskadierenden Schreibzugriffe auf den Flash zu minimieren, fungiert das Journal als Schreibcache. Alle Änderungen am Dateisystem werden im Journal gepuffert, bevor der Baum aktualisiert wird. Sobald das Journal eine bestimmte Größenschwelle erreicht, wird eine Commit-Operation durchgeführt, die alle Knoten innerhalb des Dateiindexbaums basierend auf dem Inhalt des Journals aktualisiert. Dies hat eine wichtige Bedeutung für die Forensik, da das bloße Durchschauen des Dateiindexbaums auf dem Flash Dateien verpassen würde, die derzeit nur im Journal gepuffert sind.

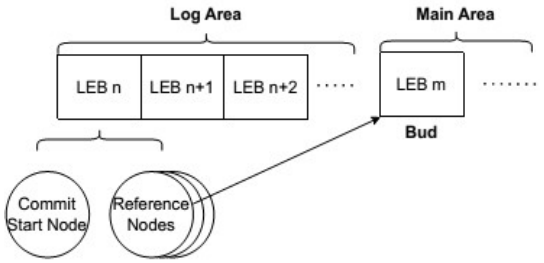

Abbildung 13: UBIFS Journal

Der Log-Bereich als Teil des Journals enthält sogenannte Referenzknoten. Referenzknoten verweisen auf LEBs im Hauptbereich (Main Area), die Knoten des Journals enthalten. Solche Knoten können Inode-Knoten, Datenknoten und so weiter sein. Die referenzierten LEBs werden auch als Buds bezeichnet. Abbildung 7 veranschaulicht dieses Konzept.

Dateilöschungen werden ebenfalls im Journal gepuffert. Die Handhabung von Dateilöschungen variiert je nachdem, ob es die Löschung eines Inodes oder eines Verzeichniseintrags betrifft. Wenn ein Inode

gelöscht wird, wird ein Inode-Knoten mit einer Verknüpfungszahl von null ins Journal geschrieben. Bei einem Journal-Commit werden während der Verarbeitung die entsprechende Inode und Verknüpfungen, die auf sie zeigen, gelöscht. Verzeichniseintraglöschungen werden ähnlich behandelt. Im Verzeichniseintragsknoten wird die Inode-Nummer auf null gesetzt und ins Journal geschrieben.

Im Allgemeinen können zwei Ansätze verwendet werden, um solche Marker zu finden. Zunächst kann der Dateiindex vollständig ignoriert werden, indem das gesamte Image gescannt wird. Dieser Brute-Force-Ansatz ist unkompliziert, da er einfach jeden Knoten im Hauptbereich parsen würde. Ein geparster Knoten kann dann daraufhin überprüft werden, ob es sich um einen Inode-Knoten mit einer Verknüpfungszahl von null oder um einen Verzeichniseintragsknoten mit einer Inode-Nummer von null handelt.

Der zweite Ansatz nutzt Strukturen des Dateisystems, um veraltete Knoten zu finden. Beispielsweise können alte Referenzknoten innerhalb des Logs verwendet werden, um bereits übertragene Buds zu finden. Dieser Ansatz ist jedoch nicht sehr vielversprechend, da nach einem Commit ein neuer, leerer Log-LEB verwendet wird. Daher ist der alte Log-LEB, der veraltete Referenzknoten enthält, sehr wahrscheinlich bereits nicht mehr zugeordnet und zur Löschung vorgesehen.

Die Menge der verbleibenden relevanten veralteten Knoten hängt vom Garbage Colletor (GC) ab. Wenn UBIFS keinen Speicherplatz mehr hat, löscht der GC veraltete Knoten und stellt dadurch freien Speicherplatz zur Verfügung.

UBI Forensik Toolkit

UBIFT bietet verschiedene Funktionen zur Analyse des UBI-Dateisystems, einschließlich Methoden zur Wiederherstellung gelöschter Dateien basierend auf den zuvor genannten Strukturen. Seine Architektur basiert auf den etablierten Konzepten von TSK (The SleuthKit von Brian Carrier [6]), insbesondere der Verwendung von Abstraktionsschichten. UBIFT ist frei verfügbar über das Repository https://github.com/matthias-deu/ubift/.

Carriers ursprüngliches Konzept ist stark auf traditionelle blockbasierte Dateisysteme ausgerichtet, daher wurde es für UBIFT angepasst. Ein besonderes Problem war die Notwendigkeit einer zusätzlichen Schicht, um UBI zu kapseln, da eine solche Schicht bei traditionellen Dateisystemen nicht vorhanden ist. Abbildung 8 zeigt das angepasste Konzept. Ähnlich wie Carriers TSK organisiert UBIFT seine Befehle in Schichten unter Verwendung von Präfixen und Suffixen.

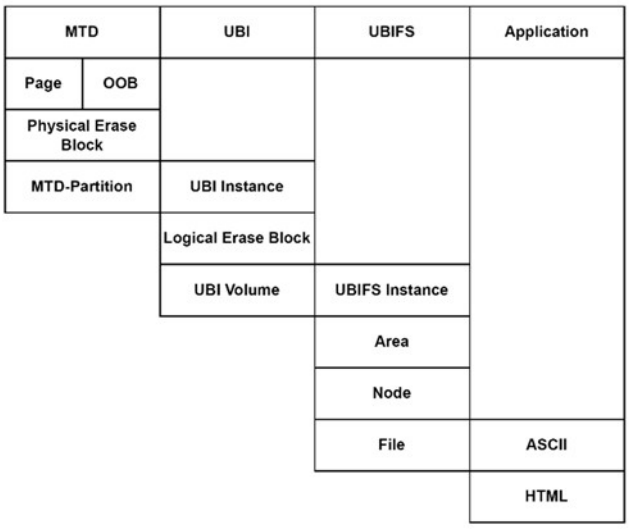

Abbildung 14: UBIFT Abstraktionsschichten, inspiriert von [6]

Das Präfix kennzeichnet die Schicht, während das Suffix die Funktionalität des auszuführenden Befehls festlegt. Die derzeit verfügbaren Befehle sind detailliert in Tabelle 1 aufgeführt.

Präfix	Suffix	Präfix	Suffix
mtd	ls, cat	fs	stat
peb	cat	i	ls, cat, stat
ubi	ls, cat	f	ls, find
leb	ls, cat	j	ls

Tabelle 4: UBIFT Befehle

Die meisten Befehle unterstützen die Angabe eines zusätzlichen Scan- oder Gelöscht-Parameters. Diese Parameter veranlassen UBIFT, den zuvor erwähnten Scan-Mechanismus zum Auffinden von Index-Knoten zu nutzen, anstatt den Dateiindex selbst zu durchlaufen. Dadurch findet UBIFT auch Knoten, die nicht mehr Teil des B+-Baums sind, also gelöschten Inhalt. Der Gelöscht-Parameter unterscheidet sich vom Scan-Parameter darin, dass er ausschließlich Knoten anzeigt, die nicht im B+-Baum gefunden werden. Wichtige Befehle im Kontext der Dateiwiederherstellung sind fls und ils. Wie bei TSK gibt fls Informationen über Dateinamen und Metadatenadressen aus. Abbildung 9 zeigt eine beispielhafte Verwendung von fls, um gelöschte Dateien aus einem Beispiel-Flash-Dump zu extrahieren.

```
$ python ./ubift.py fls flash_dump.bin -o 0 -n data
↪  --deleted
Type    Inode   Parent  Name
file    0       105     secret.txt
dir     0       104     secret_folder
file    0       107     secret_image1.jpg
file    0       107     secret_image4.jpg
```

Abbildung 15: Beispielhafte Ausgabe des fls Befehls.

In Abbildung 9 ist neben dem Flashdump auch das Offset der UBI-Instanz über den Switch -o anzugeben (hier also PEB 0, d.h. am Anfang des Flash-Dumps) sowie der Name des UBI-Volumes über den Switch -n (hier also das Volume data). Die Ausgabe von fls enthält dann Typ, Metadatenadresse (also Inode), übergeordnetes Verzeichnis (Parent) und den Dateinamen. Da alle extrahierten Dateien gelöscht sind, wird ihnen eine Inode-Adresse von null zugewiesen.

```
$ python ./ubift.py ils foscam.bin -o 213 -i 0
↪  --scan
[...]
inum|uid|gid|timestamps|mode|nlink|size|data|dent
4023|0|0|2000|2000|2000|LINK|rwxrwxrwx|1|35B|0|1
4024|0|0|2000|2000|2000|LINK|rwxrwxrwx|1|43B|0|1
4025|0|0|1970|1970|1970|FILE|rw-------|1|530B|18|1
4026|0|0|1970|1970|1970|DIR|rwxr-xr-x|0|160B|0|1
111|0|0|2017|2019|2019|FILE|rw-r--r--|1|17.1KiB|5|1
[...]
```

Abbildung 16: Beispielhafte ils Ausgabe

ils bietet einen umfassenden Überblick über alle verfügbaren Inodes innerhalb von UBIFS und zeigt wichtige forensische Metadaten wie Benutzer-/Gruppen-IDs, Zeitstempel und Dateitypen an. Abbildung 10 zeigt eine beispielhafte Ausgabe von ils für eine Internetkamera. Die Nutzung von UBIFT wird durch ein bereitgestelltes Autopsy-Plugin vereinfacht. Somit können Ermittler der digitalen Forensik den vertrauten Autopsy-Dateibrowser für eine benutzerfreundliche Untersuchung nutzen. In einer Fallstudie wurde UBIFT auf einen öffentlich verfügbaren Flash-Dump einer Foscam R2-Kamera angewendet, um die Bedeutung von UBIFT für eine digital-forensische Untersuchung zu demonstrieren. Die Ergebnisse der Evaluation sind in dem Artikel [21] verfügbar.

Ähnliche Tools

Zusätzlich zu UBIFT gibt es weitere Tools zur Analyse von UBIFS, die eine ähnliche, jedoch teilweise stark eingeschränkte Funktionalität bieten. Der UBI Reader [9] ist ein Python-Modul und eine Sammlung von Skripten, die für das Extrahieren von Daten und die Analyse von UBI/UBIFS-Images konzipiert sind. Die umfangreiche Auswahl an eigenständigen Skripten innerhalb des UBI Readers ist eher wenig benutzerfreundlich. Um spezifische Aufgaben zu erfüllen, müssen Benutzer zunächst das relevante Skript finden und dessen Funktionalität verstehen, was potenzielle Komplexität einführt. Wird der UBI Reader auf einen künstlich generierten Flash-Dump ohne strukturelle Fehler angewendet, erzielt er Ergebnisse, die denen von UBIFT ähneln. Dennoch fehlen dem UBI Reader Funktionen zur Wiederherstellung gelöschter Dateien, und er berücksichtigt nicht das Dateisystem-Journal – ein bekanntes Problem, das auf der GitHub-Seite des Autors dokumentiert ist. Folglich bleiben Daten innerhalb des Journals, die für digitale forensische Untersuchungen potenziell entscheidend sind, unbeachtet, was möglicherweise zum Auslassen von Beweisen führt, die für den Erfolg einer Untersuchung entscheidend sind.

Der UBIFS Dumper [8] wird als einzelnes Python-Skript bereitgestellt, das als CLI implementiert ist. Es bietet die Funktionalität, den Inhalt von UBIFS-Images anzuzeigen oder zu extrahieren. Wird das Werkzeug auf eine einzelne MTD-Partition angewendet, funktioniert es häufig problemlos. Jedoch stößt es bei der Handhabung mehrerer UBI-Instanzen auf Probleme und meldet einen unbekannten Dateityp. Außerdem fehlt dem Werkzeug, ähnlich wie dem UBI Reader, die Unterstützung für das Journal, wodurch es unfähig ist, gelöschte Dateien wiederherzustellen. Zusammenfassend unterscheidet sich UBIFT von Werkzeugen wie dem UBI Reader und UBIFS Dumper durch die Einführung neuer Funktionen und Fähigkeiten:

- **UBIFT basiert auf dem etablierten TSK-Schichtkonzept**: Die Übernahme von Konzepten aus Brian Carriers Sleuth Kit zielt darauf ab, die Akzeptanz von UBIFT innerhalb der digitalen Forensik-Community zu erhöhen. Die Integration in das SleuthKit erscheint jedoch gegenwärtig fraglich, angesichts der zusätzlichen Schichten, die durch das UBI-Ökosystem eingeführt wurden.
- **Umfassende Wiederherstellung gelöschter Dateien**: UBIFT ist das einzige derzeit verfügbare Werkzeug, das die Wiederherstellung gelöschter Dateien durch Verwendung des UBIFS-Journals ermöglicht. UBIFTs einzigartiger Scan-Mechanismus ermöglicht die Wiederherstellung von Daten, die andere Methoden übersehen.
- **Benutzerfreundlichkeit**: UBIFT legt Wert auf

Benutzerfreundlichkeit, insbesondere durch eine detaillierte Hilfefunktion. Das Autopsy-Plugin ermöglicht darüberhinaus eine GUI-basierte Nutzung von UBIFT.

Zusammenfassung und Ausblick

Im Rahmen dieses Beitrags wurde UBIFT vorgestellt, ein Python-Toolkit, das die Möglichkeit bietet, tiefgreifende digital-forensische Analysen von UBIFS durchzuführen. Es wurde gezeigt, dass UBIFT eine strukturierte Analyse eines Images ermöglicht, ähnlich den etablierten Konzepten von Brian Carrier. Weiterhin wurde eine Analyse und Bewertung von UBIFS bereitgestellt, die hervorhebt, dass gelöschte Daten aufgrund seines Journals und des allgemein bei Flash-Dateisystemen verwendeten Out-of-Place-Update-Ansatzes wiederhergestellt werden können.

Allerdings gibt es mehrere Einschränkungen. Zunächst ist UBIFT nicht in der Lage, alle Arten von Flash-Dumps zu verarbeiten, z. B. können Flash-Dumps, die fehlerhafte Strukturen oder Bit-Flips enthalten, unvorhergesehene Probleme verursachen. Dies ist eine Folge davon, dass UBIFT hauptsächlich mit selbst erstellten Dumps getestet wurde, die keinerlei fehlerhafte Strukturen enthalten. Solche Spezialfälle, wie verschlüsselte Instanzen von UBIFS, wurden ebenfalls nicht getestet. Daher wird angestrebt, die Robustheit und Vielseitigkeit von UBIFT zu verbessern, indem es einer umfassenderen Palette von Datensätzen unterzogen wird.

Referenzen

[1] Abbott, D. (Ed.), 2013. Linux for Embedded and Real-time Applications. Embedded technology series. 3rd ed. ed., Elsevier Science, Burlington.

[2] Bharadwaj, N.K., Singh, U., 2019. Acquisition and Analysis of Forensic Artifacts from Raspberry Pi an Internet of Things Prototype Platform. Recent Findings in Intelligent Computing Techniques 707, 311–322.

[3] Bityutskiy, A., 2007. UBI - Unsorted Block Images. Titel: http://www.linux-mtd.infradead.org/doc/ubi.ppt. Accessed: 2023-09-30.

[4] Boukhobza, J., 2017. Flash Memory Integration: Performance and Energy Issues. Energy management in embedded systems set, Elsevier Science, San Diego. Titel: http: //www.sciencedirect.com/science/book/9781785481246.

[5] Brewer, J.E., Gill, M., 2008. Nonvolatile Memory Technologies With Emphasis on Flash: A Comprehensive Guide to Understanding and Using NVM Devices.

[6] Carrier, B., 2003. Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers Titel: https: //www.utica.edu/academic/institutes/ecii/publications/articles/A04C3F91-AFBB-FC13-4A2E0F13203BA980.pdf.

[7] David Woodhouse, 2001. JFFS : The Journalling Flash File System Titel: https://sourceware.org/jffs2/jffs2.pdf.

[8] GitHub, 2023-08-13a. ubidump: Tool for viewing and extracting files from an UBIFS image. URL: https://github.com/nlitsme/ubidump.

[9] GitHub, 2023-08-13b. ubi_reader: Collection of Python scripts for reading information about and extracting data from UBI and UBIFS images. Titel: https://github.com/onekey-sec/ubi_reader.

[10] Hunter, A., 2008. A Brief Introduction to the Design of UBIFS: UBIFS Whitepaper Titel: http://www.linux-mtd.infradead. org/doc/ubifs_whitepaper.pdf.

[11] Liu, S., Guan, X., Tong, D., Cheng, X., 2010. Analysis and Comparison of NAND Flash Specific File Systems. Chinese Journal of Electronics 19, 403–408. doi:10.23919/CJE.2010. 10167491.

[12] Micheloni, R., Crippa, L., Marelli, A., 2010. Inside NAND Flash Memories. Springer Netherlands, Dordrecht. doi:10. 1007/978-90-481-9431-5.

[13] Micron Technology, Inc. jcooke, dberrett (TW), vschulthies, 2010. Nand flash 101: An introduction to nand flash and how to design it into your next product. URL: https://user.eng.umd.edu/~blj/CS-590.26/micron-tn2919.pdf.

[14] Moonbeom Park, S.J., 2017. Iot hacking & forensic with 0-day Titel: https://troopers.de/downloads/troopers17/TR17_ What_happened_to_your_home.pdf.

[15] MTD, 2023. Memory Technology Device Subsystem for Linux. URL: http://www.linux-mtd.infradead.org/. Accessed: 2023- 09-30.

[16] Olivier, P., Boukhobza, J., Senn, E., 2012. On benchmarking embedded linux flash file systems. ACM SIGBED Review 9(2) 43-47 9 URL: https://arxiv.org/pdf/1208.6391.pdf

[17] OpenWrt, 2023. Openwrt project. URL: https://openwrt.org/docs/techref/filesystems#ubifs. Accessed: 2023-09-30.

[18] UBI, 2020. Unsorted block images. URL: http://www.linux-mtd.infradead.org/doc/ubi.html. Accessed: 2023-09- 30.

[19] YAFFS, 2023. Yet Another Flash File System. URL: https: //yaffs.net/. Accessed: 2023-09-25.

[20] Zeifman, I., Rossi, B. 2023. OpenWrt OS: How It Works, Challenges, Security Concerns and Alternatives. URL: https://sternumiot.com/iot-blog/openwrt-how-it-works-challenges-and-alternatives. Letzter Zugriff: 2023-09-30.

[21] Deutschmann, M., Baier, H. 2024. Ubi est indicium? On forensic analysis of the UBI file system. Forensic Science International: Digital Investigation, 48.