



Fakultät für Ingenieurwissenschaften

## **Bachelor-Thesis**

Forensische Sicherungsmöglichkeiten von Linux-Betriebssystemen  
auf Mobilgeräten

Abschlussarbeit zur Erlangung des Grades eines

## **Bachelor of Engineering**

der Hochschule Wismar

eingereicht von:

Christian Peter

Studiengang IT-Forensik

Erstgutachter:

Prof. Dr. Antje Raab-Düsterhöft

Zweitgutachter:

Dipl. Ing. Hans-Peter Merkel

eingereicht am 15.06.2023

# Aufgabenstellung

Ziel dieser Bachelor-Thesis ist die Ermittlung und Darstellung geeigneter Methoden zur Datenextraktion aus mobilen Endgeräten mit Linux-Betriebssystemen.

Dabei ist zu untersuchen, welche Ansätze der Android- und Desktop-Linux-Sicherung geeignet sind und ob neue Ansätze gefunden werden können.

Die Extraktionen sollen an einer vorbereiteten Auswahl von Testgeräten durchgeführt werden. Möglichkeiten zur Sicherung des RAM-Speichers sind zu erörtern und exemplarisch darzustellen.

Bei vorliegender Verschlüsselung des Gerätespeichers ist nach Möglichkeit eine unverschlüsselte Datensicherung durchzuführen. Verschlüsselte Datensicherungen sind zu entschlüsseln.

## Kurzreferat

Für die Sicherung von mobilen Endgeräten mit iOS oder Android-Betriebssystemen haben sich Methoden und kommerzielle Software-Lösungen etabliert, welche auch Anwendern ohne fundiertes Fachwissen die Datenextraktion dieser Geräte ermöglichen [1]. Zu Smartphones und Tablets mit Linux-Betriebssystemen liegen aktuell keine „Best Practice“ Methoden zur forensischen Datensicherung vor. Das wachsende Bewusstsein für Nachhaltigkeit und den Schutz der eigenen Daten lassen eine zunehmende Relevanz freier und quelloffener Betriebssysteme auch für Mobilgeräte erwarten. Diese Arbeit soll die Möglichkeiten der Datenakquise aus mobilen Endgeräten mit Linux-Betriebssystemen erfassen und für Praktizierende in der Mobilforensik geeignet darstellen.

## Abstract

Methods and commercial software solutions have been established for imaging mobile devices with iOS or Android operating systems, which also enable users without in-depth specialist knowledge to extract data from these devices [1]. There are currently no "best practice" methods for the forensic data acquisition from smartphones and tablets with Linux operating systems. The growing awareness of sustainability and the protection of one's own data suggest an increasing relevance of free and open-source operating systems also for mobile devices. This thesis aims to collect the possibilities of data acquisition from mobile devices with Linux operating systems and to present them in a suitable way for practitioners in mobile forensics.

---

# Inhaltsverzeichnis

Aufgabenstellung.....	2
Kurzreferat.....	3
Abstract.....	3
Abkürzungsverzeichnis.....	6
Glossar.....	8
1 Einleitung.....	9
1.1 Zielsetzung und Grenzen.....	9
1.2 Bedeutung.....	10
2 Linux auf Mobilgeräten.....	12
2.1 Ubuntu Touch.....	15
2.2 SailfishOS und AuroraOS.....	17
2.3 „klassische“ Linux-Systeme.....	19
2.4 „prebuild“ Images und Standardzugangsdaten.....	22
3 Testgeräte.....	23
4 Forensische Sicherung von Mobilgeräten.....	32
4.1 Strategische Vorbereitung.....	33
4.2 Operationale Vorbereitung.....	34
4.3 Datensammlung.....	38
4.3.1 Eingeschaltetes Gerät.....	38
4.3.2 Eingeschaltetes Gerät mit unbekannten Zugangsdaten.....	39
4.3.3 Ausgeschaltetes Gerät.....	40
4.3.4 Vorgehen bei der Datensammlung.....	41
5 RAM-Sicherung.....	44
5.1 LiME.....	44
5.2 AVML.....	47
5.3 SUC / sboot_dump.....	48
5.4 MTKClient.....	50
5.5 Weitere Methoden.....	51
6 Flash-Speicher-Sicherung.....	52
6.1 TCP-Live-Sicherung (SSH/Netcat).....	52
6.1.1 Suche nach der gemounteten Kryptopartition.....	57
6.1.2 Sicherung des entschlüsselten Volumes.....	58
6.1.3 Fazit der TCP-Live-Sicherungen.....	59
6.2 ADB-Live-Sicherung (Ubuntu Touch).....	61
6.2.1 Fazit der ADB-Live-Sicherungen.....	62
6.3 Recovery-Sicherung.....	63
6.3.1 Geräte mit A/B-Partitionen.....	66
6.3.2 Samsung Geräte.....	69
6.3.3 Fazit der Recovery-Sicherungen.....	70



---

6.4 Bootloader-Sicherung.....	72
6.4.1 EDL-Modus (Qualcomm SoCs).....	72
6.4.2 MTK-BROM-Modus (MediaTek SoCs).....	76
6.4.3 APX-Modus (Nvidia Tegra SoCs).....	78
6.4.4 FEL-Modus (Allwinner SoCs).....	87
6.4.5 Fazit der Bootloader-Sicherungen.....	90
6.5 Live-Linux Sicherungen (x86).....	92
6.5.1 Fazit der Live-Linux Sicherungen.....	94
6.6 Gerätespezifische Sicherungen.....	95
6.6.1 Jumpdrive.....	95
6.6.2 Nokia N9/N950 Rescue-Image.....	96
6.6.3 Fazit der gerätespezifischen Sicherungen.....	97
6.7 externe Flash-Speicher-Sicherungen.....	98
6.7.1 Fazit der externen Flash-Speicher-Sicherungen.....	99
7 Einsicht in das Dateisystem.....	100
8 Entschlüsselung der Sicherungen.....	103
8.1 Entschlüsselung von SailfishOS (Xperia XA2).....	104
8.2 Entschlüsselung von Droidian (Xperia 5).....	107
9 Bewertung der Ergebnisse.....	111
9.1 Bewertung der TCP-Live-Sicherung.....	112
9.2 Bewertung der ADB-Live-Sicherung.....	112
9.3 Bewertung der Recovery-Sicherung.....	113
9.4 Bewertung der Bootloader-Sicherung.....	114
9.5 Bewertung der Live-Linux-Sicherung.....	114
9.6 Bewertung der gerätespezifischen Sicherungen.....	115
9.7 Bewertung der externen Flash-Speicher-Sicherung.....	115
10 Zusammenfassung und Ausblick.....	116
Literaturverzeichnis.....	118
Abbildungsverzeichnis.....	126
Tabellenverzeichnis.....	129
Software.....	130
Selbstständigkeitserklärung.....	131
Anhangsverzeichnis.....	132

# Abkürzungsverzeichnis

<b>Abkürzung</b>	<b>Bedeutung</b>
ADB	Android Device Bridge
AFF	Advanced Forensic Format
APT	Advanced Packaging Tool
AVML	Acquire Volatile Memory for Linux
BCT	Boot Configuration Table
Blob	Binary Large Object
BPMP	Boot and Power Management Processor
BROM	Boot ROM
BSI	Bundesamt für Sicherheit in der Informationstechnik
CVE	Common Vulnerabilities and Exposures
DA	Download Agent
EABI	Embedded-Application Binary Interface
FDE	Full Disk Encryption
EDL	Emergency Download Mode (Qualcomm)
EWf	Expert Witness Format
FESCo	Fedora Engineering and Steering Committee
F(L)OSS	Free (Libre) Open Source Software
FSF	Free Software Foundation
GCC	GNU Compiler Collection
GNU	GNU's not Unix
GTK	Gimp Toolkit
GSI	Generic System Image
IMEI	International Mobile Equipment Identity
initrd	Initial Ramdisk
IPED	Indexador e Processador de Evidências Digitais (portugiesisch)
OMP	(Russian) Open Media Platform
LiME	Linux Memory Extractor
LUKS	Linux Unified Key Setup
LVM	Logical Volume Manager
MTK	MediaTek
NFC	Near Field Communication

<b>Abkürzung</b>	<b>Bedeutung</b>
NIST	National Institute Of Standards And Technology
NTP	Network Time Protocol
PGP	Pretty Good Privacy
Phosh	Phone Shell
PIN	Personal Identification Number
RAM	Random-Acess Memory
ROM	Read-Only Memory
SBK	Secure Boot Key
SEA	Symmetric Encryption Algorithm
SoC	System on Chip
SPL	Secondary Program Loader
SSH	Secure (Socket) Shell
ST	Suckless Terminal
SWMO	Simple Wayland Mobile
SXMO	Simple X Mobile
TAC	Type Allocation Code
TCP	Transmission Control Protocol
TWRP	Team Win Recovery Project
(U)EFI	(Unified) Extensible Firmware Interface
UMS	USB Mass Storage

---

# Glossar

Begriff	Bedeutung
BLOB	Proprietäre Komponente, die in binärer Form vorliegt und deren Quellcode öffentlich nicht zugänglich ist (z.B. Gerätetreiber) [2]
Branch	Entwicklungszweig innerhalb eines (Git-) Projekts
codename	Interne Herstellerbezeichnung der Mobilgeräte (z.B. „Hammerhead“)
coreutils	Anwendungen von denen erwartet wird, dass sie auf jedem (GNU)-Betriebssystem vorhanden sind. [3]
Downstream (Kernel)	Linux-Kernel mit Anpassungen für bestimmte Geräte / Komponenten
friendly name	Für Menschen gut lesbare Bezeichnung / Handelsbezeichnung der Mobilgeräte (z.B. „Nexus 5“)
Fork	Abspaltung eines Projekts auf Basis eines bereits bestehenden Projekts
libhybris	Kompatibilitätsschicht, die Bionic-C-Bibliotheken (Android) für Glibc-basierte Distributionen (klassische Linux-Derivate) nutzbar macht [4]
Mainline (Kernel)	Der offizielle Linux-Kernel, wie er auf <a href="http://www.kernel.org">www.kernel.org</a> veröffentlicht und von Linus Torvalds mittels PGP-Schlüssel signiert wird
OMAP	ARM-CPU-Familie von Texas Instruments
Project Treble	Hardware-Abstraktion von Mobilgeräten, welche mit Android 8 eingeführt wurde - Erleichtert die Aktualisierung des Systems unabhängig von der Hardware
Smudge Attack	Untersuchung des Gerätedisplays auf Fingerabdruckspuren um PIN- oder Pattern-Eingaben nachvollziehen zu können
Sunxi	ARM-CPU-Familie von Allwinner
Tegra	ARM-CPU-Familie von Nvidia
Treblizing	Befähigung eines Mobilgeräts für „Project Treble“ durch die Auslagerung von Gerätetreibern und -modulen auf eine „Vendor“-Partition

# 1 Einleitung

## 1.1 Zielsetzung und Grenzen

In Vorbereitung auf die geplanten Datenextraktionen wurden Mobiltelefone und Tablets verschiedener Hersteller beschafft und geeignete Linux-Systeme darauf installiert. Dabei wurde auch die Kompatibilität zu verschiedenen Betriebssystemen und der verbaute Chipsatz berücksichtigt, um ein umfassendes Bild dieses Themengebietes zu zeichnen und die geplanten Sicherungsmethoden auf ihre Allgemeingültigkeit hin zu überprüfen (siehe Abschnitt 3 – Seite 23).

Ziel dieser Bachelorarbeit ist die Ermittlung und Vorstellung geeigneter Sicherungsmethoden von Linux-Betriebssystemen auf Mobilgeräten. Dabei soll jeweils die Extraktion mit dem größtmöglichen Umfang erzeugt werden. Es soll dargelegt werden, welche Sicherungsmethode sich für welches Anwendungsszenario eignet und welchen Einfluss das verwendete Betriebssystem auf die Durchführung der Sicherung hat.

Dabei richtet sich die Arbeit an Mitarbeiterinnen und Mitarbeiter IT-forensischer Abteilungen, regionaler Beweissicherungseinheiten oder ähnlicher Einrichtungen, welche mit dem Umgang von Kommandozeilenbefehlen vertraut sind. Ein grundlegendes Verständnis für die Themengebiete: ADB, Fastboot, Verschlüsselung, Partitionierung und Mounting wird vorausgesetzt.

Es werden Software-basierte Extraktionsmöglichkeiten betrachtet. Hardware-gestützte Methoden wie Chip-Off oder JTAG sollen hier nicht untersucht werden. Es kann jedoch durch die Kenntnis der vorliegenden Verschlüsselungen die Prognose gegeben werden, dass sich diese Methoden zur Datenextraktion eignen. Eine Auswertung und Interpretation der gesicherten Daten soll nicht erfolgen. Über eine Einsicht in die Extraktion soll lediglich die erfolgreiche Datensicherung überprüft werden. Auch die RAM-Sicherung soll nur beispielhaft erläutert und exemplarisch dargestellt werden.

## 1.2 Bedeutung

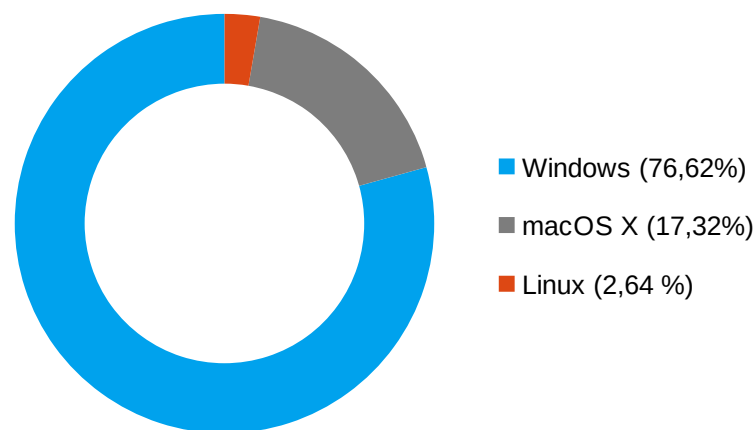
Bisherige Artikel über Ubuntu Touch [5] und SailfishOS [6] legten den Fokus auf die Analyse der jeweiligen nativen Anwendungen der Systeme und betrachten Sicherungsmöglichkeiten teils nur oberflächlich anhand eines einzelnen Gerätemodells. In der zu fertigenden Ausarbeitung sollen mehrere verschiedene Geräte betrachtet und so ein umfassenderes Bild der Möglichkeiten der Datensicherung gezeichnet werden.

Im Januar 2023 wurde von Statista weltweit ein Marktanteil von 1,77% für Desktop-Linux Installationen ausgewiesen [7]. In Deutschland betrug der Anteil 2,64% [8] (siehe Bild 1). Der Anteil von Linux auf Mobilgeräten wird aktuell nicht statistisch erfasst (siehe Bild 2) und liegt wahrscheinlich unter dem der Desktop-Installationen. Somit stellt Linux (insbesondere auf Mobilgeräten) eine Nische dar. Für Kriminelle kann eine solche Nische attraktiv sein, da Strafverfolgungsbehörden bisher nicht auf vorgefertigte Sicherungslösungen zurückgreifen können und Nutzer mobiler Linux-Systeme selbst darauf Einfluss nehmen können, wie das Gerät geschützt wird. Linux-First Geräte wie das Librem 5 oder das Pinephone bieten Hardware-Schalter um einzelne Gerätefunktionen wie das Mikrofon oder die Kamera zu deaktivieren. Die Firma „Phantom Secure“ modifizierte in der Vergangenheit Android-Geräte mit derartigen Features, um auch Kriminellen abhörsichere Mobiltelefone anbieten zu können [9].

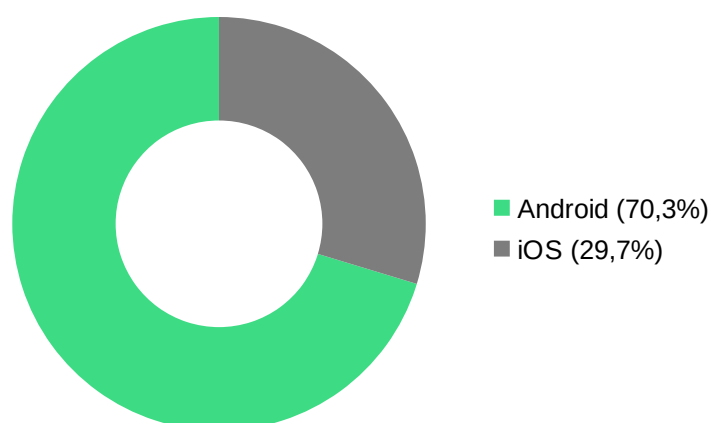
Die Entwickler mobiler Linux-Systeme verzichten zumeist aus Überzeugung darauf, ihre Nutzer zu tracken. Daher sind belegbare Werte für tatsächliche Nutzerzahlen der Systeme schwer zu ermitteln. Das Portal „jolla-devices.com“ erstellte Ende 2020 einen Analyse-Report über die Nutzerzahlen der eigenen Plattform und die Abonnenten der SailfishOS-Social-Media-Plattformen [10]. Daraus erging die Schätzung, dass es im Jahr 2020 zwischen 5.000 und 10.000 aktive SailfishOS-Nutzer gab. Das Online-Magazin Softpedia veröffentlichte im September 2015 eine Einschätzung des Ubuntu Touch Community Entwicklers Riccardo Padovani [11]. Demnach soll es in jenem Jahr ca. 25.000 Ubuntu

Touch-Nutzer gegeben haben.<sup>1</sup> An einer Umfrage des Pinephone-Herstellers Pine64 im Jahr 2022 [12] beteiligten sich 3.079 Nutzer. Dieses soll weniger als 5% aller Eigentümer eines Pinephones entsprechen. Hochgerechnet ergäben sich daraus ca. 60.000 Pinephone-Nutzer. Eine Auswertung der Crowdfunding Kampagne des Librem 5 durch das Portal „omg!ubuntu!“ im Februar 2020 [13] zeigte auf, dass zu diesem Zeitpunkt bereits mehr als 2.800 Librem 5 veräußert wurden.

Zunehmend relevant ist auch die Verwendung des SailfishOS-Forks „AuroraOS“ durch Mitarbeiter russischer Staatskonzerne und Angehörige der russischen Verwaltung [14] (siehe S. 12).



**Bild 1:** Marktanteile der führenden Betriebssysteme in Deutschland im Januar 2023 (Statista 2023)



**Bild 2:** Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in Deutschland im 3. Quartal des Jahres 2022 (Statista 2023)

<sup>1</sup> Die tatsächliche Nutzerzahl ist wahrscheinlich geringer. Über eine Anfrage bei dem UBPorts-Entwickler Florian Leeber wurde bekannt, dass täglich ca. 3000 Geräte Anfragen an den Update-Push-Server stellen.

## 2 Linux auf Mobilgeräten

Ist von Smartphone-Betriebssystemen die Rede, werden in der Regel lediglich zwei Betriebssysteme betrachtet. Für das dritte Quartal des Jahres 2022 wurde laut Statista in Deutschland ein Marktanteil von 29,7% für Apples iOS und 70,3% für Googles Android-System erhoben [15] (siehe Bild 2). Im Schatten der Marktführer können jedoch Entwicklungen beobachtet werden, welche die Nutzung klassischer Linux-Systeme auch auf Mobilgeräten ermöglichen oder auf der Basis etablierter Distributionen eigene Betriebssysteme bereitstellen. Einige dieser Entwicklungen reichen in eine Zeit zurück, in der die Marktmacht von Apple und Google noch nicht derart gefestigt war wie sie es heute ist. Bereits 2005 stellte Nokia Geräte mit Maemo, einem Mobilsystem auf Debian-Basis, vor [16]. Im Jahr 2011 folgte in Kooperation mit Intel das Nokia N9 mit dem MeeGo Betriebssystem [17]. Trotz überwiegend positiver Resonanz stellte Nokia die Entwicklung und weitere Pflege von MeeGo zu Gunsten der Kooperation mit Microsoft und der Verwendung von Windows Mobile ein [18]. Intel fand in der Folge mit Samsung einen neuen Partner und entwickelte das System zu TIZEN weiter [19]. Zunächst fand TIZEN auch auf Smartphones Anwendung. Heute setzt Samsung das System überwiegend auf seinen Smart-TV-Geräten ein. Ehemalige MeeGo-Entwickler verließen nach dem Microsoft-Deal Nokia und gründeten die finnische Firma Jolla Oy (Kurz Jolla) [20]. Jolla schuf aus der MeeGo-Codebasis das eigene Betriebssystem SailfishOS und stellte dieses zunächst auf selbst entwickelten Geräten wie dem Jolla Phone bereit. In Kooperation mit der Regierung von Bolivien und dem bolivischen Unternehmen „Jala“ wird SailfishOS seit 2017 in Lateinamerika vertrieben, um die technologische Abhängigkeit von den vereinigten Staaten zu minimieren [21]. Bereits 2016 erwarb OMP Russia als russisches Staatsunternehmen Anteile von Jolla und entwickelte auf Grundlage von SailfishOS ein eigenes Betriebssystem: AuroraOS. Zwar wurde die Kooperation von Jolla und OMP im Zuge des russischen Angriffskrieges in der Ukraine im Jahr 2022 ausgesetzt. Da OMP jedoch den kompletten Sourcecode von AuroraOS besitzt, wird das System auch ohne die Beteiligung aus Finnland weiter entwickelt [22]. Aktuell



wird AuroraOS von Mitarbeitenden der russischen Eisenbahngesellschaft, russischen Post und Rostelcom eingesetzt. Für die Durchführung einer Volkszählung im Jahr 2021 wurde das System auf hunderten Tablets installiert. Zukünftig sollen Mobiltelefone mit AuroraOS auch von Lehrern, Ärzten, Regierungsangestellten, Staatsbetrieben und Betrieben der kritischen Infrastruktur eingesetzt werden [14].

Im Jahr 2013 stellte Ubuntu-Entwickler Canonical Ubuntu Touch (zunächst Ubuntu Phone OS) vor [23] und fand im folgenden Jahr mit BQ (Spanien) und Meizu (China) zwei Partner, die ihre Geräte mit vorinstalliertem Ubuntu Touch anboten [24]. Im Jahr 2017 beendete Canonical das Projekt [25], jedoch wird Ubuntu Touch seither von der Ubports Foundation betreut und weiterentwickelt [26]. Ubports portierte das System zuvor für Android-Geräte wie das Nexus 5. Mit dem deutschen Startup „Hallo Welt“ fand auch Ubports einen Partner, der seine Smartphones der „Volla Phone“-Reihe wahlweise auch mit Ubuntu Touch anbietet. Zum Zeitpunkt der Anfertigung dieser Arbeit fand ein Wechsel der Ubuntu-Basis des Mobilsystems statt. Zuvor wurde Ubuntu 16.04 genutzt, nun sollte Ubuntu 20.04 Verwendung finden.

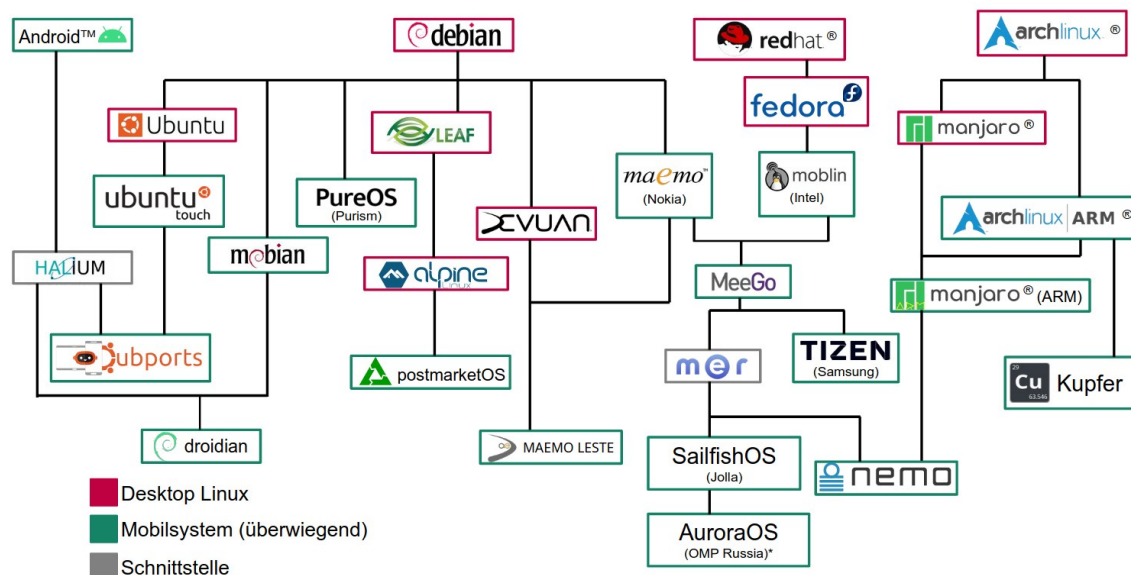
Einen erheblichen Einfluss auf die Weiterentwicklung mobiler Linux-Systeme haben „Linux First“ Geräte wie das Pine64 Pinephone (Pro) und das Purism Librem 5 [27]. Purism erstellte für sein PureOS auf Debian-Basis mit Phosh eine mobile Shell [28], die in der Folge auch von den Entwicklerteams anderer Distributionen als Oberfläche für Mobilgeräte übernommen wurde. Für das Pinephone erschienen bereits mehrere Ableger klassischer Desktop-Linux-Distributionen wie openSuse, OpenMandriva, Slackware oder Manjaro [29].

Das PostmarketOS-Projekt wurde 2017 ins Leben gerufen [30] und arbeitet seither am *Mainlining* von Mobiltelefonen. Aktuell werden 30 Android Geräte als kompatibel geführt. Darüber hinaus werden jedoch weitaus mehr Geräte vom Projekt unterstützt [31].

Das „Projekt Haliu“ hingegen arbeitet an Kernel Patches der jeweiligen Android-Kernels, um eine Hardwareabstraktion für die Ausführung von Linux auf dem Android-Kernel zu erreichen [32]. Auch Ubuntu Touch setzt für seine

Geräteports auf Haliu und fügt fehlende Funktionen über gerätespezifische Patches hinzu. Dieses Vorgehen erhöht die Auswahl kompatibler Geräte; diese verbleiben jedoch auf ihrem jeweiligen Kernel-Stand mitsamt bekannter Sicherheitslücken und Fehler. Ein weiterer Vertreter mit Haliu-Basis ist das Droidian Projekt, das den Debian-Ableger Mobian für Android Geräte verfügbar macht [33]. Das „Project Sandcastle“ [34] hingegen verfolgt das Ziel, Android und Linux auf Mobilgeräte von Apple zu portieren.

Eine Übersicht (ohne Anspruch auf Vollständigkeit) über die Entwicklung mobiler Linux Distributionen zeigt die folgende Darstellung<sup>2</sup>:



**Bild 3:** Übersicht der Entwicklung mobiler Linux-Distributionen

Darüber hinaus entwickeln sich stetig neue Ableger klassischer Linux-Distributionen. So billigte das Fedora Engineering and Steering Committee (FESCo) im Dezember '22 einen „Fedora Mobility“-Zweig auf Phosh-Basis [35].

<sup>2</sup> Die Abstammung der einzelnen Distributionen wurde den jeweiligen Projektdokumentationen entnommen. Android ist eine Marke von Google LLC, [Android-Roboter](#): „Creative Commons“ von Google LLC (CC BY 3.0); [Ubports Logo](#): „Creative Commons“ der Ubports Foundation (CC BY-SA 4.0); [PureOS Logo](#): „Creative Commons“ von Purism (CC BY-SA 4.0); [Mobian logo](#): „Creative Commons“ des Mobian Project (CC BY-SA 4.0); [postmarketOS Logo](#): „Creative Commons“ von „postmarketOS developer team“ (CC BY-SA 4.0); [Devuan emblem](#): „Creative Commons“ der „Dyne.org foundation“ (CC BY-SA 4.0); Debian ist ein registriertes Markenzeichen von Software in the Public Interest, Inc.; [Debian Logo](#): „Creative Commons“ von Software in the Public Interest, Inc. (CC BY-SA 3.0); [Leaf Logo](#): „Creative Commons“ von Leaf project (CC BY-SA 3.0); Manjaro® name and logo are registered trademarks of Manjaro GmbH & Co. KG. Some rights reserved; Arch Linux, copyright © 2002-2020 Judd Vinet and Aaron Griffin; Red Hat, Fedora und das Infinity design logo: Copyright © 2020 Red Hat, Inc., Red Hat und das Red Hat Logo sind Marken oder eingetragene Marken von Red Hat, Inc. oder dessen Tochterunternehmen in den USA und anderen Ländern. Alle anderen in diesem Dokument genannten Warenzeichen sind Eigentum der jeweiligen Inhaber; [Mer Logo](#): „Creative Commons“ von Mer Project (CC BY 3.0); [Tizen Logo](#): „Creative Commons“ von Tizen Project (CC BY 3.0); Maemo Logo: „Creative Commons“ von Maemo-Leste (CC BY 3.0); die Verwendung des [Ubuntu-Logos](#) erfolgt mit der Erlaubnis von Canonical Ltd.

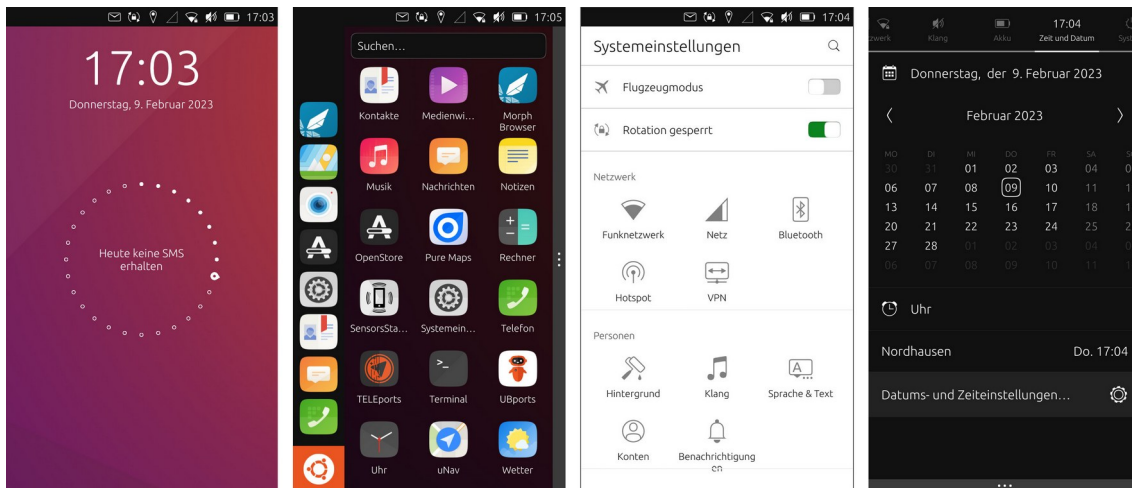
## 2.1 Ubuntu Touch

Aktuell werden kompatible Geräte im „stable“ Kanal noch mit Version 16.04 (Xenial Xerus) betrieben. Der Wechsel auf 20.04 (Focal Fossa) wird zum Zeitpunkt der Anfertigung der Arbeit jedoch bereits vorbereitet und kann auf einigen Geräten bereits über den „testing“ Kanal bezogen werden. Viele ältere Android-Geräte werden bei diesem Versionsprung nicht berücksichtigt, da der integrierte systemd-Service mindestens die Kernel-Version 3.18 voraussetzt [36] und ein Update des Kernels bei Haliu-Installationen nicht vorgesehen ist. Im Gegensatz zu einer Desktop-Ubuntu-Installation sind die Systemdaten schreibgeschützt im Speicher abgelegt. Eine Installation von Software über das Terminal und das Advanced Packaging Tool (APT) ist so nicht ohne Weiteres möglich. Studierende der Purdue University (USA) zeigten bereits 2021 auf, dass Ubuntu Touch keine Verschlüsselung des Gerätespeichers vorsieht [5]. Bisher hat auch der Wechsel auf 20.04 keine diesbezügliche Änderung bewirkt. Eine Möglichkeit, das Home-Verzeichnis einer Ubuntu Touch Installation zu verschlüsseln, wird im Ubports Blog beschrieben [37]. Hierbei wird jedoch auch hervorgehoben, dass dieses Vorgehen nicht nutzerfreundlich ist und nach jedem größeren Update wiederholt werden muss.

Zur Absicherung des Geräts kann wahlweise ein alphanumerisches Passwort beliebiger Länge oder ein vierstelliger PIN verwendet werden. Da die Verwendung einer PIN auf Mobilgeräten üblich ist und diese sehr kurze Ziffernfolge ebenfalls zur Erlangung der „Root“-Berechtigung über Sudo verwendet wird, wurde dieser Umstand von der NIST (National Institute Of Standards And Technology) als Verwundbarkeit eingestuft: (CVE-2022-40297). Der Penetrationstester Filip Karczewski demonstrierte im Jahr 2022 [38] einen Ansatz, diese Schwachstelle im Rahmen eines Bruteforce-Ansatzes auszunutzen und die PIN auf einem bereits entsperrten Gerät zu ermitteln. Dieses Vorgehen konnte auf Geräten mit Version 16.04 und 20.04 nachvollzogen werden. Da bei neueren Versionen von Ubuntu Touch jedoch bei dem Aufruf des Terminals das Nutzerpasswort abgefragt wird, erfordert der Angriff zudem, dass das Terminal bereits vom Nutzer geöffnet und nicht wieder

geschlossen wurde. Eine Terminal-App ist in der Standard-Installation enthalten und bereits nach dem ersten Gerätestart für den Nutzer verfügbar. Während der Bearbeitung dieser Arbeit erschien mit Ubuntu Touch 20.04 OTA-01 ein Update, welches die Vergabe einer bis zu 12-stelligen PIN gestattet.

Ubuntu Touch verwendet die mobile Oberfläche „Lomiri“ (ehemals Unity8):



**Bild 4:** Lomiri-Oberfläche in Ubuntu Touch 16.04

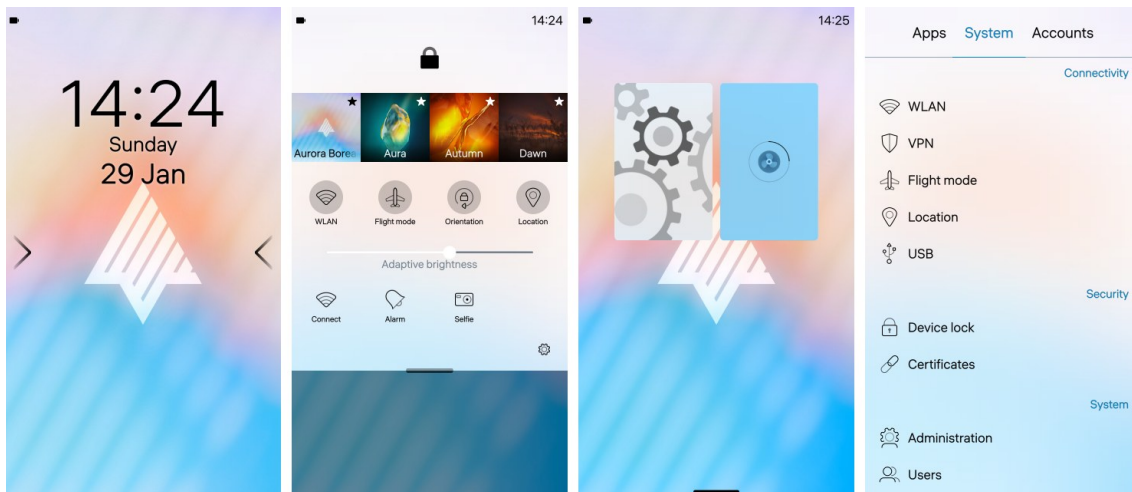
Im Zuge der Installation wird auch die eigene „Ubports“ Recovery installiert, die sich im Aufruf und im Aufbau je nach Gerät leicht unterscheidet. Wie auch bei der TWRP Recovery werden Verbindungen via ADB im Recovery-Modus mit Root-Berechtigungen erlaubt, ohne dass der Host-PC zuvor autorisiert werden muss. Für die Installation auf Android Geräten ist grundsätzlich zuvor das Entsperren des Bootloaders erforderlich. Partner-Geräte wie das Volla Phone werden bei werksseitiger Ubuntu Touch Installation jedoch auch mit geschlossenem Bootloader betrieben. Die offizielle Homepage des Projekts listet aktuell 56 kompatible Geräte [39]. Darüber hinaus sind Installationsdateien für weitaus mehr Geräte verfügbar. Auch im Rahmen dieser Arbeit werden zum Teil Geräte verwendet, welche nicht in der Liste kompatibler Geräte erfasst sind. Diese beinhalten nicht die Installation der Ubports-Recovery und verfügen zumeist über die TWRP-Recovery, welche auch im Rahmen der Installation des Systems Verwendung fand.

## 2.2 SailfishOS und AuroraOS

Im Zeitraum der Erstellung dieser Arbeit wurde SailfishOS 4.4 (Vanha Rauma) verwendet. Im Februar 2023 wurde das Update auf 4.5 (Struven ketju) auf ausgewählten Geräten ausgerollt. Wie auch bei Hailium kann SailfishOS Komponenten des Android-Kernels über die Hardware-Abstraktion „*libhybris*“ ansprechen [40]. Die freien Bestandteile des Meego Systems setzen als „Mer“ auf dem Kernel auf und bilden die Grundlage für das SailfishOS-Betriebssystem. Einen Artikel über die mögliche Datenextraktion und -analyse von SailfishOS veröffentlichte K. Tzvetanov im Jahr 2020 [6]. Zur Sicherung nutzte der Autor die alternative Recovery TWRP für eine ADB-Verbindung sowie eine SSH-Verbindung im Live-Betrieb. Die zu jener Zeit aktuelle Version SailfishOS 3.2 (Torronsuo) sah noch keine Verschlüsselung der Nutzerdaten vor. Diese wurde erst mit Version 3.3 (Rokua) hinzugefügt [41] und wird seither bei der Neuinstallation kompatibler Geräte im Rahmen der Erstinstallation angewandt. Dabei wird die unter Linux übliche LUKS Verschlüsselung eingesetzt. Bei AuroraOS (OC Abpopa) handelt es sich um einen Fork des Systems des russischen Staatsunternehmens OMP. Entsprechend unterscheidet sich der Versionsstand beider Systeme. Aktuell ist die AuroraOS-Version 4.0.2 (Balakovo). Beide Systeme verwenden die Closed-Source Oberfläche „Lipstick“:



Bild 5: "Lipstick"-Oberfläche unter SailfishOS



**Bild 6:** "Lipstick"-Oberfläche unter AuroraOS (AuroraOS IDE Emulator)

Da AuroraOS-Geräte nicht frei veräußert werden, erfolgten die Betrachtungen dieses Systems anhand der AuroraOS IDE<sup>3</sup>, welche einen Emulator des Systems bereitstellt und die Emulation verschiedener kompatibler Geräte erlaubt.

Bei beiden Systemen wird dem Nutzer gegenüber zunächst keine Terminal-Anwendung ausgewiesen. Diese wird verfügbar, nachdem unter Einstellungen → System → Administration der Entwicklermodus aktiviert wurde. Hier lässt sich auch (bei Kenntnis der PIN des Nutzers) ein beliebiges Passwort für die SSH-Verbindung zum Gerät festlegen. Eine Eigenheit, welche SailfishOS von seinem Vorgänger Meego übernommen hat, ist die Verwendung des Aufrufs „devel-su“ für die Erlangung von administrativen Berechtigungen. Auch in AuroraOS ist „devel-su“ enthalten, jedoch wird bei der Ausführung des Systems in der VM auch die Nutzung von „sudo“ ermöglicht. Das ursprüngliche Meego-System ist in der Bedienung sehr ähnlich. Auch hier wird zunächst keine Terminal-Anwendung zur Verfügung gestellt. Hier benötigt die Aktivierung des Entwicklermodus zusätzlich eine bestehende Internetverbindung (siehe Anhang Seite P).

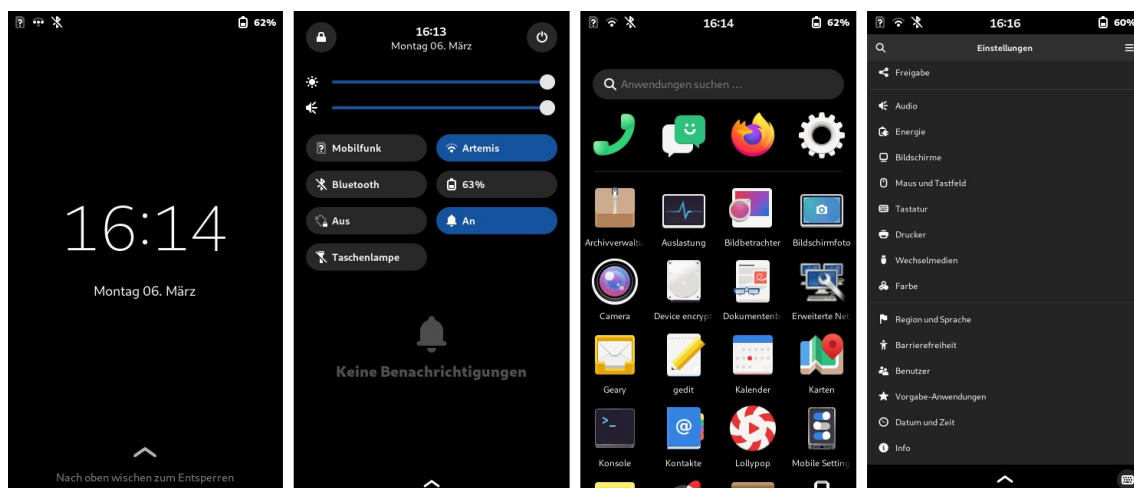
<sup>3</sup> In Folge des russischen Angriffskrieges in der Ukraine wurde der Zugang zu russischen Webseiten seitens Russland zum Teil erheblich eingeschränkt. Ein Aufruf der IDE-Homepage: <https://community-omprussia.ru> und ein Download der Entwicklungsumgebung (Version 4.0.2.175 / Qt Creator 4.15.2) erforderte die Verwendung eines russischen Proxy-Dienstes (hidemy.name).



## 2.3 „klassische“ Linux-Systeme

Neben dediziert für die Verwendung auf Smartphones geschaffenen Linux-Systemen werden zunehmend auch „klassische“ Desktop-Linux-Distributionen für den mobilen Betrieb angepasst. So handelt es sich bei PostmarketOS um einen Fork von Alpine Linux [42]. Mobile Debian-Ableger mit Mainline-Kernel sind Mobian [43] und PureOS [44]. Droidian macht Mobian auch als Halium-Installation verfügbar [33], während Arch Linux und Manjaro eigene ARM-Distributionen pflegen. Projekt Kupfer überträgt das gerätespezifische Bootstrap-Modell von PostmarketOS auf Arch Linux [45]. Auf Geräten, welche auf Mainline portiert wurden, ist theoretisch jede Distribution installierbar, die ein Image für die entsprechende Architektur bereitstellt. Klassische Desktopumgebungen wie Gnome, XFCE und KDE sind hierbei ebenfalls verwendbar. Diese sind jedoch nicht auf die Steuerung über Toucheingaben und die Darstellung auf meist sehr kleinen Bildschirmen angepasst. Daher wurden Umgebungen geschaffen, welche die Verwendung eines mobilen Geräts erleichtern und ein konsistentes Nutzererlebnis ermöglichen sollen. Dabei weisen drei Oberflächen aktuell die größte Verbreitung in mobilen Systemen auf und bieten jene Funktionen, die von einem Smartphone erwartet werden.

Phosh wurde von Purism für das eigene Librem 5 entwickelt und verwendet Elemente von GNOME und GTK [28]:



**Bild 7:** Phosh-Oberfläche unter Droidian

Nahezu alle aktuell verfügbaren mobilen Distributionen stellen eine Installation mit Phosh zur Verfügung. Die Bedienung ist hierbei intuitiv gehalten und ähnelt jener von iOS oder Android. Programme werden in einer Icon-Übersicht dargestellt, welche bei laufenden Anwendungen zugleich den Wechsel zwischen den Anwendungen erlaubt.

KDE Plasma Mobile wird von der KDE Community gepflegt und steht für Mainline-Installationen zur Verfügung [46]:

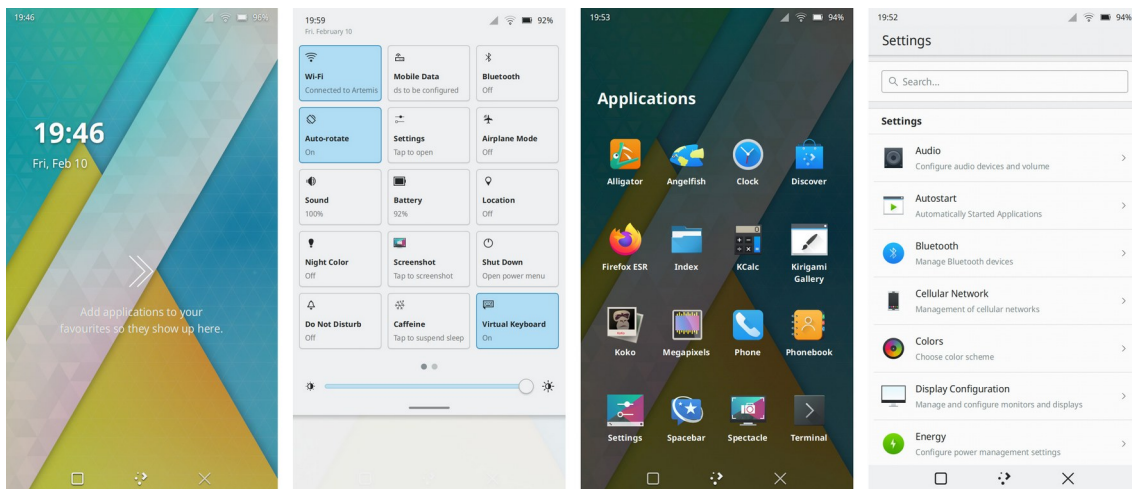


Bild 8: KDE Plasma Mobile unter PostmarketOS

Die Oberfläche nutzt das Qt-Framework. Mit Kirigami wurde ein Designstandard für mobile Anwendungen etabliert. Auch hier ist die Bedienung der von Android oder iOS nicht unähnlich.

Einen anderen Bedienansatz verfolgt das Team von SXMO/SWMO:

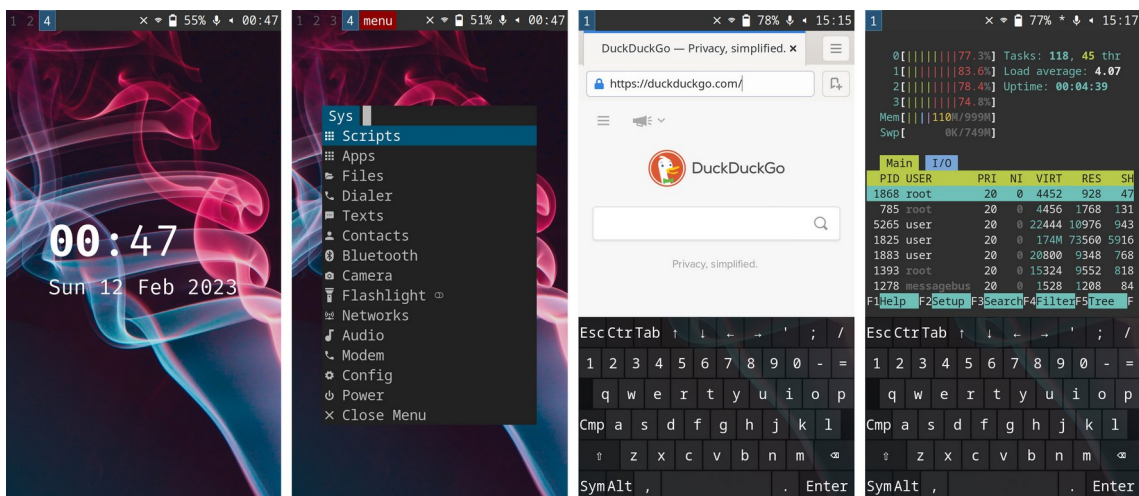


Bild 9: SWMO-Oberfläche unter PostmarketOS



Die Steuerung des Geräts erfolgt hier über ein Dialogsystem, das sowohl über den Touchscreen als auch über die Gerätetasten bedient werden kann. Dabei nutzt das Projekt Elemente der Desktop-Umgebung „suckless“ [47]. Häufig verwendete Funktionen können als Skript hinterlegt und über das Systemmenü aufgerufen werden. Bisher wird die Oberfläche überwiegend bei PostmarketOS-Installationen eingesetzt; es existieren jedoch auch Pakete für andere Distributionen.

Neben diesen Oberflächen werden auch noch weitere Projekte betreut, die zum Teil jedoch noch nicht komplett nutzbar sind. Die „GNOME Shell on Mobile“ befindet sich aktuell in Entwicklung und wird vom Bundesministerium für Bildung und Forschung gefördert [48]. „Glacier“ soll eine quelloffene Alternative zu „Lipstick“ von SailfishOS bieten [49]. Auch die „Cutie-Shell“ verfolgt diesen Ansatz und ist bereits unter Droidian installierbar [50]. „Pangolin Mobile“ wird für dahliaOS entwickelt, ist aber ebenfalls auf anderen Distributionen installierbar [51]. „Hildon“ wurde ursprünglich von Nokia für Maemo entwickelt und wird noch heute für Maemo Leste auf Devuan-Basis genutzt [52]. Auch wurde Lomiri bereits für andere Distributionen portiert [53]. Die Distribution AVMultiPhone verwendet PostmarketOS und einen leicht angepassten Mate-Desktop als Basis und soll damit eher einen „Hosentaschen-PC“ bereitstellen, mit dem auch telefoniert werden kann [54]. Für die reine Bedienung des Terminals über Toucheingaben ohne grafische Benutzeroberfläche eignet sich fbkeyboard (Framebuffer Keyboard) [55].

Eine Terminal-Anwendung ist für gewöhnlich bei klassischen Linux-Distributionen vorinstalliert und steht dem Nutzer direkt zur Verfügung. Unter SXMO lässt sich das Terminal unter „Apps → foot“ (bei Wayland-Installationen) bzw. unter „Apps → st“ (bei X11-Installationen) aufrufen.

Weitere mobile Oberflächen sind im Anhang der Arbeit abgebildet (Seite: AY).

## 2.4 „prebuild“ Images und Standardzugangsdaten

Einige Distributionen werden fertig konfiguriert zum Download angeboten. Dabei werden von den jeweiligen Projektbetreuern Standardzugangsdaten vorgegeben, welche bei Bedarf vom jeweiligen Nutzer nach der Installation zu ändern sind. Da jedoch die Chance besteht, dass der Nutzer des Geräts die Zugangsdaten nach der Installation nicht verändert hat, werden die bekannten Standardzugangsdaten nachfolgend dargestellt.

**Tabelle 1:** Standardzugangsdaten der mobilen Linux-Systeme

Distribution	User	Passwort
Arch Linux [29]	alarm	123456
Droidian [56]	droidian	1234
ExpidusOS [29]	expidus	expidus
Fedora Pinephone (Github) [29]	pine	123456
Fedora Pinephone (Nightly) [29]	pine	1111
Kali Phosh Unofficial [29]	kali	8888
Kali Nethunter Official [29]	kali	1234
Maemo Leste [57]	user	user
	root	toor
Maemo Leste (Pinephone) [29]	user	12345
	root	toor
Meego [58]	user	-
	root	rootme
Manjaro [29]	manjaro	123456
	root	root
Mobian [29]	mobian	1234
openSUSE (Pinephone) [29]	pine	1234
	root	root
openSUSE (Volla Phone) [59]	phablet	1234
SailfishOS 3.4 oder später [60]	defaultuser	-
SailfishOS 3.3 oder früher [60]	nemo	-
PostmarketOS [61]	user	147147
Ubuntu Touch [62]	phablet	-

Der Meego-Standarduser verfügt zunächst über kein Passwort, es kann jedoch über `passwd` eines vergeben werden. Bei SailfishOS ist ein Passwort bei der Ersteinrichtung festzulegen. Es konnte bei der Verwendung der AuroraOS-IDE nachvollzogen werden, dass auch bei AuroraOS der Nutzer „defaultuser“ verwendet wird.

### 3 Testgeräte

In Vorbereitung auf die geplanten Untersuchungen wurden Mobilgeräte beschafft, welche im Zeitraum der Ausarbeitung zur Ausführung von „FOSS“ (Free and Open Source Software) Betriebssystemen geeignet erschienen.

Dabei wurden Geräte präferiert, deren Funktionsumfang unter Verwendung der alternativen Betriebssysteme möglichst vollumfänglich gegeben ist. In Ermangelung freier Treiber ist jedoch die Funktion der Kamera bei vielen Geräten derzeit nicht gegeben. Eine Übersicht über den zu erwartenden Funktionsumfang geben die jeweiligen Projektseiten und Wikis der FOSS Systeme.

Nachfolgend sollen die verwendeten Mobilgeräte kurz vorgestellt werden. Es wird aufgezeigt mit welchem Betriebssystem sie betrieben werden, welchen Funktionsumfang sie aufweisen, in welchem Maße das Gerät von dem jeweiligen Projekt unterstützt wird und welchen SoC sie verwenden. Dieser wird im Rahmen der Sicherung von Relevanz sein, da sich je nach verwendetem SoC unterschiedliche Angriffspunkte für die Extraktion von Daten aus dem Gerät ergeben können.

#### Nokia N9 (Lankku)

**SoC:** Texas Instruments OMAP3630 (ARMv7)

**RAM / Speicher:** 1 GB / 16 GB

**OS (original):** MeeGo 1.2 "Harmattan"

**FOSS-OS:** MeeGo 1.2 "Harmattan"

**Support:** Durch Nokia eingestellt / Community

**Funktionen:** komplett

**Beschreibung:**

Das Nokia N9 ist das einzige Mobiltelefon, das von Nokia mit dem MeeGo Betriebssystem ausgeliefert wurde. Seitens der Community wird bis heute neue Software für das System entwickelt.



**Bild 10:** Nokia N9

## Pine64 Pinephone (Beta Edition)

**SoC:** Allwinner A64 4 x ARM Cortex-A53 (ARM64)

**RAM / Speicher:** 2 GB / 16 GB

**OS (original):** ohne / Manjaro KDE

**FOSS-OS:** Manjaro KDE „Ruah“

**Support:** Manjaro GmbH & Co. KG (Mainline)

**Funktionen:** komplett

### Beschreibung:

Pine64 entwickelte das Pinephone mit der Zielsetzung, ein Linux-Phone mit Mainline-Unterstützung bereitzustellen. Diverse Distributionen sind verfügbar. Es wird auch eine „Pro“ Variante mit leistungsfähigerer Hardware angeboten.



Bild 11: Pinephone

## Google Nexus 5 (hammerhead)

**SoC:** Qualcomm Snapdragon 800, MSM8974 (ARMv7)

**RAM / Speicher:** 2 GB / 16 GB

**OS (original):** Android 4.4.3 „KitKat“ bis 6.0.1 „Marshmallow“

**FOSS-OS:** Ubuntu Touch 16.04

**Support:** UBports (legacy)

**Funktionen:** überwiegend komplett, kein NFC

### Beschreibung:

Ubuntu Touch wurde hier über die Legacy Methode installiert. Das Halium Projekt verwendete das Nexus 5 als eines der ersten Referenzgeräte. Auch wird das Gerät vom Mainline-Kernel unterstützt.

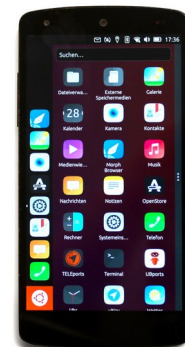


Bild 12: Nexus 5

## BQ Aquaris E5 HD (vegetahd)

**SoC:** MediaTek MT6582 (ARMv7)

**RAM / Speicher:** 1 GB / 16 GB

**OS (original):** Android 4.4.2 „KitKat“

**FOSS-OS:** Ubuntu Touch 16.04

**Support:** UBports (legacy)

**Funktionen:** komplett

### Beschreibung:

Hersteller BQ (Spanien) lieferte das Aquaris E5 HD sowie den Vorgänger E4.5 sowohl mit Android als auch mit Ubuntu Touch aus. Nach der Aufgabe des Projekts durch Canonical übernahm UBports die Betreuung der Software.

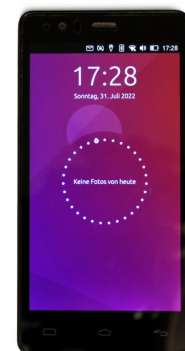


Bild 13: BQ Aquaris E5

## Sony Xperia XA2 (pioneer)

**SoC:** Qualcomm Snapdragon 630, SDM630 (ARM64)

**RAM / Speicher:** 3 GB / 32 GB

**OS (original):** Android 8 „Oreo“ – 9 „Pie“

**FOSS-OS:** Sailfish X (SailfishOS 4.4) „Vanha Rauma“

**Support:** Jolla

**Funktionen:** Komplett, FDE aktiv

### Beschreibung:

Diverse Geräte der Sony Xperia Reihe werden von Jolla offiziell unterstützt und bieten nach dem Erwerb einer Gerätelizenz u.a. den Support für Android-Apps und Microsoft Exchange. Eine Verschlüsselung erfolgte über die Installation.



Bild 14: Xperia XA2

## Motorola Moto G (titan)

**SoC:** Qualcomm Snapdragon 400, MSM8226 (ARMv7)

**RAM / Speicher:** 1 GB / 16 GB

**OS (original):** Android 4.4.4 „KitKat“

**FOSS-OS:** Sailfish OS 4.4.0.58 „Vanha Rauma“

**Support:** Community (VerdandiTeam / legacy)

**Funktionen:** komplett

### Beschreibung:

Community Ports von SailfishOS erhalten keinen Android-Support. Ein Community-Port von Ubuntu Touch ist ebenfalls für dieses Gerät verfügbar [XDA02]. Im Rahmen der Installation fand keine Verschlüsselung statt.



Bild 15: Moto G

## Lenovo A6000

**SoC:** Qualcomm Snapdragon 410, MSM8916 (ARM64)

**RAM / Speicher:** 1 GB / 8 GB

**OS (original):** Android 4.4.4 „KitKat“

**FOSS-OS:** PostmarketOS 22.06 (Phosh)

**Support:** Community (Mainline)

**Funktionen:** Keine Kamera, Kein GPS

### Beschreibung:

Seitens der PostmarketOS-Community gehört dieses Gerät zu den Android-Geräteports mit dem größten Funktionsumfang. Gefertigt wurde das Mobiltelefon von Wingtech.

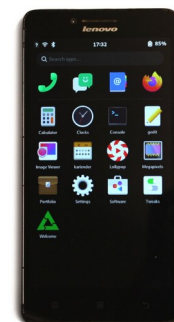


Bild 16: Lenovo A6000

## Motorola Droid 4 XT894 (maserati)

**SoC:** Texas Instruments OMAP4430 (ARMv7)

**RAM / Speicher:** 1 GB / 16 GB / 32GB µSD

**OS (original):** Android 2.3 „Gingerbread“ – 4.0 „ICS“

**FOSS-OS:** Maemo Leste 1.0

**Support:** Offiziell (Mainline)

**Funktionen:** Keine Kamera, Dual Boot über Bootmenü

### Beschreibung:

Die Installation des Systems auf eine MicroSD-Karte ist seitens der offiziellen Dokumentation [63] vorgesehen. Android verbleibt hier auf dem Gerät und kann über ein Bootmenü ebenfalls geladen werden.



Bild 17: Motorola Droid 4

## ASUS Transformer Pad TF300T

**SoC:** Nvidia Tegra 3 T30L (ARMv7)

**RAM / Speicher:** 1 GB / 32 GB / 32GB µSD

**OS (original):** Android 4.0 „ICS“ – 4.2 „Jelly Bean“

**FOSS-OS:** PostmarketOS edge (Mate)

**Support:** Community (Mainline)

**Funktionen:** Keine Kamera, GPU nur partiell

### Beschreibung:

Die verbaute Tegra-CPU wird nahezu vollständig vom Mainline-Kernel unterstützt. Auch hier ist die Installation auf MicroSD vorgesehen. Der Bootloader sieht kein Dual-Boot mit Android vor.



Bild 18: ASUS TF300T

## ASUS Transformer Book T100TA

**SoC:** Intel Atom Z3775 (x86\_64)

**RAM / Speicher:** 2 GB / 64 GB

**OS (original):** Windows 8.1 / Windows 10 (32 Bit)

**FOSS-OS:** Mobian Phosh / Debian 12 „Bookworm“

**Support:** Ohne (Mainline)

**Funktionen:** Keine Kamera

### Beschreibung:

Mit der Atom-Serie wagte Intel den Versuch, der überwiegenden Verbreitung von ARM-Chips in Mobilgeräten etwas entgegenzusetzen. Seitens des Mobian-Projekts werden auch Images für x86 Geräte bereitgestellt.



Bild 19: ASUS T100TA

## Volla Phone (yggdrasil) / Gigaset GS290

<b>SoC:</b>	MediaTek Helio P23 MT6763 (ARM64)
<b>RAM / Speicher:</b>	4 GB / 64 GB
<b>OS (original):</b>	Android 9 „Pie“ – Android 10 „Q“
<b>FOSS-OS:</b>	Ubuntu Touch 20.04
<b>Support:</b>	UBports (Haliu 9) / Hallo Welt Systeme UG
<b>Funktionen:</b>	komplett

### Beschreibung:

Das Startup „Hallo Welt“ lässt das Volla Phone von Gigaset produzieren und bietet neben dem Android „Volla OS“ auch Ubuntu Touch vorinstalliert an. Eine fast baugleiche „rugged“ Variante wird als Volla Phone X angeboten (GX290).



Bild 20: Volla Phone

## Xiaomi Redmi Note 8T (willow)

<b>SoC:</b>	Qualcomm Snapdragon 665, SM6125 (ARM64)
<b>RAM / Speicher:</b>	4 GB / 64 GB
<b>OS (original):</b>	Android 9 „Pie“ – Android 10 „Q“
<b>FOSS-OS:</b>	Ubuntu Touch 16.04
<b>Support:</b>	Community (Haliu 9)
<b>Funktionen:</b>	Keine Kamera

### Beschreibung:

Ubuntu Touch wurde als generisches Image auf dem Gerät installiert. Die Funktionspatches wurden für das fast baugleiche Modell Note 8 (Ginko) erstellt und erfordern das Flashen der Vendor Partition des Ginko Modells.

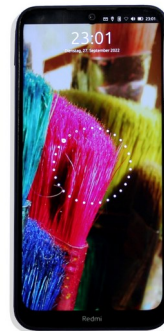


Bild 21: Redmi Note 8T

## Samsung Galaxy S5 SM-G900F (klte)

<b>SoC:</b>	Qualcomm Snapdragon 801, MSM8974AC (ARMv7)
<b>RAM / Speicher:</b>	2 GB / 16 GB
<b>OS (original):</b>	Android 4.4.2 „KitKat“ bis 6.0.1 „Marshmallow“
<b>FOSS-OS:</b>	PostmarketOS edge (Gnome-Mobile)
<b>Support:</b>	Community (Mainline)
<b>Funktionen:</b>	Keine Kamera, kein Sound, keine Telefonie

### Beschreibung:

Die Unterstützung durch PostmarketOS ist hier noch rudimentär. Zum Verfassen von SMS oder Emails eignet sich das Gerät jedoch. Den größten Funktionsumfang bietet auf dem S5 aktuell Ubuntu-Touch.

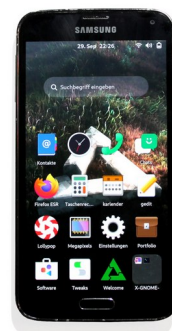


Bild 22: Galaxy S5



## Xiaomi Mi A2 lite (daisy)

**SoC:** Qualcomm Snapdragon 625, MSM8953 (ARM64)

**RAM / Speicher:** 3 GB / 32 GB

**OS (original):** Android 8.1 „Oreo“ – Android 10 „Q“

**FOSS-OS:** Droidian Phosh / Debian 11 „Bullseye“

**Support:** Community (Halium 9)

**Funktionen:** Keine Kamera, kein Suspend

### Beschreibung:

Die Installation erfolgt hier mittels des generischen Halium-Images. Neuere Versionen von Droidian erfordern weitere Patches. Ubuntu Touch wird ebenfalls angeboten. Das Gerät ist nahezu baugleich mit dem Redmi 6 Pro.



Bild 23: Mi A2 lite

## Oneplus 3T (A3003)

**SoC:** Qualcomm Snapdragon 821, MSM8996 (ARM64)

**RAM / Speicher:** 6 GB / 64 GB

**OS (original):** Android 6.0.1 „Marshmallow“ – Android 9 „Pie“

**FOSS-OS:** Droidian Phosh / Debian 12 „Bookworm“

**Support:** Community (Halium 9)

**Funktionen:** Keine Kamera

### Beschreibung:

Droidian wurde auch hier als generisches Image installiert. Dazu wurde eine Vendor-Partition angelegt (Treblizing) und über die Installation einer kompatiblen LineageOS-Version befüllt.



Bild 24: Oneplus 3T

## Samsung Galaxy S9 SM-G960F (starlte)

**SoC:** Samsung Exynos 9810 (ARM64)

**RAM / Speicher:** 4 GB / 64 GB

**OS (original):** Android 8 „Oreo“ – Android 10 „Q“

**FOSS-OS:** Ubuntu Touch 16.04

**Support:** Community (Halium 10)

**Funktionen:** Kein ADB oder MTP, Kein SD-Karten-Automount

### Beschreibung:

Auch dieser Port findet sich nicht in der offiziellen Gerätedatenbank von Upports und wird über einen Telegram-Kanal betreut. Das Fehlen von ADB und MTP lässt Schwierigkeiten bei der Sicherung erwarten.



Bild 25: Galaxy S9



## HTC One M7

<b>SoC:</b>	Qualcomm Snapdragon 600, APQ8064T (ARMv7)
<b>RAM / Speicher:</b>	2 GB / 32 GB
<b>OS (original):</b>	Android 4.1.2 „Jelly Bean“ – Android 5.1 „Lollipop“
<b>FOSS-OS:</b>	Ubuntu Touch 14.04
<b>Support:</b>	Community (Legacy)
<b>Funktionen:</b>	Kein Ton, keine Telefonie

### Beschreibung:

Ein sehr altes Ubuntu Touch Image konnte über das XDA-Forum bezogen werden. Auch für PostmarketOS und SailfishOS wird eine Geräteunterstützung angeboten.



Bild 26: HTC One

## Samsung Galaxy S III GT-I9300 (m0)

<b>SoC:</b>	Samsung Exynos 4412 (ARMv7)
<b>RAM / Speicher:</b>	1 GB / 16 GB / 32 GB µSD
<b>OS (original):</b>	Android 4.0 „ICS“ – Android 4.3 „Jelly Bean“
<b>FOSS-OS:</b>	PostmarketOS edge (SXMO)
<b>Support:</b>	Community (Mainline)
<b>Funktionen:</b>	Keine Kamera, keine Telefonie

### Beschreibung:

Fertige Images werden auf der PostmarketOS Projektseite angeboten. Die Installation erfolgt auf eine MicroSD-Karte. Für das Galaxy SIII steht auch ein freies Open-Source-Android namens Replikant zur Verfügung.



Bild 27: Galaxy S III

## Xiaomi Redmi 2 Pro (wt88047)

<b>SoC:</b>	Qualcomm Snapdragon 410, MSM8916 (ARM64)
<b>RAM / Speicher:</b>	2 GB / 16 GB
<b>OS (original):</b>	Android 4.4.4 „Kitkat“ – Android 5.1 „Lollipop“
<b>FOSS-OS:</b>	PostmarketOS 22.06 (Plasma Mobile)
<b>Support:</b>	Community (Mainline)
<b>Funktionen:</b>	Keine Kamera

### Beschreibung:

Das Redmi 2 erreicht unter PostmarketOS einen nahezu vollständigen Funktionsumfang. Xiaomi ließ das Gerät von der Firma Wingtech entwickeln und produzieren.



Bild 28: Redmi 2

## Xiaomi Pocophone F1 (beryllium)

**SoC:** Qualcomm Snapdragon 845, SDM845 (ARM64)

**RAM / Speicher:** 6 GB / 64 GB

**OS (original):** Android 8.1 „Oreo“ – Android 10 „Q“

**FOSS-OS:** Mobian Phosh / Debian 12 „Bookworm“

**Support:** Offiziell (Mainline)

**Funktionen:** Keine Kamera, kein GPS

### Beschreibung:

Der SDM845 verfügt über acht Kerne und ist der aktuell leistungsfähigste SoC mit Mainline Unterstützung. Images für Arch Linux (Kupfer), PostmarketOS, Ubuntu Touch und Droidian werden ebenfalls für dieses Gerät angeboten.



Bild 29: Pocophone F1

## Xiaomi Redmi 4X (santoni)

**SoC:** Qualcomm Snapdragon 435, MSM8940 (ARM64)

**RAM / Speicher:** 3 GB / 32 GB

**OS (original):** Android 6.0 „Marshmallow“ – Android 7.1 „Nougat“

**FOSS-OS:** Sailfish OS 3.0.3.10 „Hossa“

**Support:** Community (Dreemurrs Embedded Labs / legacy)

**Funktionen:** Kein Fingerprint, fehlende Video-Codecs

### Beschreibung:

Auch dieser Sailfish-Port erreicht einen nahezu vollständigen Funktionsumfang. Das Porting-Team stellt auch eine native Arch-Linux Installation für das Redmi 4X bereit, welche jedoch keine Telefonie ermöglicht.



Bild 30: Redmi 4x

## Dell Venue 8 Pro (3845)

**SoC:** Intel Atom Z3735G (x86\_64)

**RAM / Speicher:** 1 GB / 32 GB

**OS (original):** Windows 8.1 / Windows 10 (32 Bit)

**FOSS-OS:** Sailfish x86 (Sailfish 4.0.1.48 „Koli“)

**Support:** Community (Sailfish x86 Project)

**Funktionen:** Keine Kamera, kein Sound

### Beschreibung:

Der Sailfish Port für x86 Geräte setzt auf Ubuntu 20.04 auf. Auch Jolla selbst bietet eine x86 Version ihres Systems für das Jolla Tablet an. Der Funktionsumfang entspricht hier der Ubuntu-Desktop Installation.



Bild 31: Dell Venue 8 Pro

## Xiaomi Redmi 9C (angelica)

**SoC:** MediaTek Helio G35 MT6765G (ARM64)

**RAM / Speicher:** 2 GB / 32 GB

**OS (original):** Android 10 „Q“ – Android 11 „R“

**FOSS-OS:** Droidian Phosh / Debian 12 „Bookworm“

**Support:** Community (Haliu 10)

**Funktionen:** Keine Kamera

### Beschreibung:

Das fast baugleiche Redmi 9A wird ebenfalls von der Droidian Community unterstützt. Zum Zeitpunkt der Anfertigung der Arbeit bestand ein Bug, welcher das Aufwecken des Displays bei MTK-SoC's nach dem Suspend verhinderte.



Bild 32: Redmi 9C

## Sony Xperia 5 (bahamut)

**SoC:** Qualcomm Snapdragon 855, SDM855 (ARM64)

**RAM / Speicher:** 6 GB / 128 GB

**OS (original):** Android 9 „Pie“ – 11 „R“

**FOSS-OS:** Droidian Phosh / Debian 12 „Bookworm“

**Support:** Offiziell (Haliu 11)

**Funktionen:** FDE aktiv

### Beschreibung:

Das Xperia 5 wird vom Droidian-Team offiziell unterstützt. Die Installation erfolgte über ein bereitgestelltes Fastboot-Installationsskript. Für diese Form der Installation wird die Verschlüsselung des Gerätespeichers ermöglicht.



Bild 33: Xperia 5

## Microsoft Surface 2

**SoC:** Nvidia Tegra 4 T114 (ARMv7)

**RAM / Speicher:** 2 GB / 32 GB

**OS (original):** Windows RT 8.1 (32Bit ARM)

**FOSS-OS:** Ubuntu Mate 22.04

**Support:** Grate Kernel

**Funktionen:** Keine Kamera, GPU nur partiell

### Beschreibung:

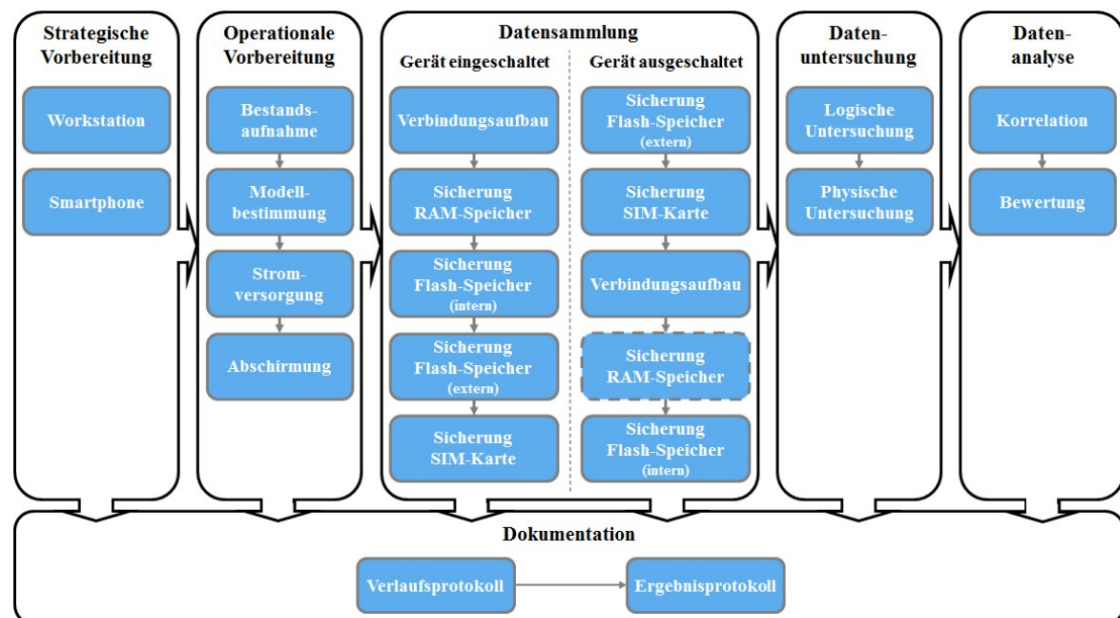
Der Support des ARM Windows RT wurde durch Microsoft eingestellt. Die Installation des Systems erfolgte über die „UEFI-Methode“, die nur eine geringe Schnittstellenkompatibilität bietet.



Bild 34: Surface 2

## 4 Forensische Sicherung von Mobilgeräten

Ungeachtet des verwendeten Betriebssystems haben sich Vorgehensmodelle etabliert, welche den Ablauf einer IT-forensischen Untersuchung normieren und somit nachvollziehbar gestalten sollen. Das BSI veröffentlichte bereits 2011 die aktuelle Fassung des Leitfadens „IT-Forensik“ [64]. Dieser beinhaltet auch Instruktionen zur Sicherung klassischer Linux-Systeme. Auf die Eigenheiten von Mobilgeräten und deren Einfluss auf die Datensicherung geht der Leitfaden hingegen nicht ein. Diesem Umstand trug Muth [65] im Jahr 2013 Rechnung und kombinierte die Vorgaben des BSI mit dem Sicherungsleitfaden der NIST [66] und dem Lehrmodell nach Geschonneck [67]. Das erstellte Modell zielt zwar auf die Sicherung von Android-Geräten, lässt sich jedoch analog auch auf Linux-Mobilgeräte übertragen, da sich die Fähigkeiten der Systeme ähneln und auch Android auf einem modifizierten Linux-Kernel aufbaut. Das forensische Prozessmodell nach Muth ist in Bild 35 dargestellt:



**Bild 35:** forensisches Prozessmodell nach Muth (Bildquelle: MT Muth [65 S. 38])

Dabei entsprechen die Kategorien „strategische Vorbereitung“, „operationale Vorbereitung“, „Datensammlung“, „Datenuntersuchung“, „Datenanalyse“ und

„Dokumentation“ jenen aus dem Leitfaden „IT-Forensik“ des BSI. Die Prozessschritte „Datenuntersuchung“ und „Datenanalyse“ sollen im Rahmen dieser Ausarbeitung nicht betrachtet werden. Eine geeignete Dokumentation ist Teil einer jeden IT-forensischen Untersuchung und wurde in diversen wissenschaftlichen Arbeiten hinreichend betrachtet. Kernaspekt der durchzuführenden Untersuchungen soll die Datensammlung sein. Ohne eine entsprechende Vorbereitung kann die Datensammlung jedoch nicht erfolgen. Daher werden notwendige Vorbereitungshandlungen speziell im Hinblick auf vorliegende Linux-Systeme nachfolgend betrachtet.

## 4.1 Strategische Vorbereitung

Im Sinne des BSI beinhaltet die strategische Vorbereitung Maßnahmen, welche vor dem Eintritt eines Sicherheitsvorfalls zu erfolgen haben. Dies wird auf die Sicherung von Mobiltelefonen besonders im Rahmen der Strafverfolgung für gewöhnlich nicht anwendbar sein. Bezogen auf die forensische Workstation können hingegen Vorbereitungen getroffen werden, um eine sachgerechte Datensammlung zu ermöglichen. Dies beinhaltet die Aktualisierung des Betriebssystems und der zu verwendenden Software, die Bereitstellung von nötigen Datenkabeln und Writeblockern, sowie die Beschaffung der erforderlichen Software-Lösungen, welche sich für die Sicherung der jeweiligen Mobilgeräte eignen.

Im Rahmen dieser Arbeit wird folgende Workstation verwendet:

---

### **Lenovo Thinkpad T440p**

CPU: Intel® Core™ i7-4712MQ

RAM: 16GB DDR3

SSD: Sandisk SDSSDH3 512GB (Manjaro KDE 22.0.2)

Transcend TS512GMTS430S (Windows 11)

Samsung SSD 850 EVO 1TB (für durchgeführte Sicherungen)

Als Sicherungssystem kommt Manjaro KDE 22.0.2 zum Einsatz. Erforderliche Software zum Herstellen einer SSH-Verbindung mit den Mobilgeräten ist im System bereits vorinstalliert. Weiterhin wurden die „android-tools“ installiert um auf die Protokolle adb und fastboot zugreifen zu können. Der Bezug der Systemzeit erfolgt über das NTP Protokoll und die Manjaro-Zeitserver. Writeblocker für USB-Datenträger (Tableau Forensic USB Bridge) und Speicherkarten (Digital Intelligence USB 3.0 Forensic Card Reader) stehen für die Sicherungen bereit. Auf geeignete Software zur Gerätekommunikation auf Bootloader-Ebene wird im Rahmen der Bootloader-Sicherung eingegangen. Sofern proprietäre Forensik-Software zur Sicherung erforderlich ist, steht ebenfalls eine Windows 11 Installation zur Verfügung.

## 4.2 Operationale Vorbereitung

Im Rahmen der **Bestandsaufnahme** gilt es, den Zustand des zu untersuchenden Mobilgeräts festzustellen. Dabei ergeben sich die relevanten Fragestellungen:

- Ist das Gerät ein- oder ausgeschaltet?
- Welchen Ladestand weist der Akku auf?
- Welche Datendienste (Wifi, mobile Daten, Bluetooth) sind aktiv?
- Stimmt die angezeigte Zeit mit der Referenzzeit überein? Wenn nein: Wie groß ist die Abweichung?
- Ist eine Bildschirmsperre eingerichtet?

Da die vorliegenden Testgeräte selbst vorbereitet wurden, sind diese Aspekte bereits bekannt. In der Praxis sind darüber hinaus auch Maßnahmen der klassischen Forensik in Betracht zu ziehen. So können z.B. aufgefundene Fingerabdruckspuren auf dem Display des Gerätes für einen „Smudge“-Angriff genutzt werden [68].

Ebenfalls kann die **Bestimmung des Gerätemodells** entscheidend für den Erfolg der Datensicherung sein. Ist das genaue Modell des Mobilgeräts

bekannt, kann eine Recherche in einer Gerätedatenbank wie z.B. [phonedb.net](http://phonedb.net) erfolgen, und es können so Informationen zum verwendeten SoC oder zur möglichen Speichergröße erlangt werden. Die Kenntnis des SoC ist für möglicherweise anwendbare Bootloader-Angriffe relevant. Darüber hinaus ist forensische Software, die auf dem Gerät angewendet werden soll, auf die zugrunde liegende Architektur des Systems abzustimmen. Informationen zum Modell können oftmals über Aufschriften am Gerät selbst erlangt werden. Häufig findet sich auch eine Modellnummer im geöffneten Gerät unter dem Akku. Zu beachten ist, dass die Handelsbezeichnung eines Geräts nicht notwendigerweise eindeutig ist. So existieren z.B. mehrere Varianten des Samsung Galaxy S9. Die in Europa übliche Variante mit der Modellnummer G960F trägt den Codenamen „samsung-starlte“ und wird von einem Exynos 9810 angetrieben, die amerikanische Variante G960U trägt den Codenamen „samsung-starqlte“ und verfügt über einen Snapdragon 845. Eindeutig lässt sich das Gerät auch über die IMEI Nummer identifizieren. Die ersten acht Ziffern der 15-stelligen IMEI bilden die TAC Nummer, die eindeutig einem Gerätemodell zugeordnet ist. Auch hier existieren Online-Datenbanken (z.B. [imei.info](http://imei.info)), welche zur Identifizierung herangezogen werden können. Üblicherweise kann die IMEI auf der Geräterückseite oder dem SIM-Kartenträger abgelesen werden. Ergänzend zur Modellbestimmung ist die **Bestimmung des verwendeten Betriebssystems** vorzunehmen, um mögliche Angriffspunkte für eine Datensicherung zu ermitteln. In einigen Fällen lässt die verwendete „Desktop“-Oberfläche einen Rückschluss auf das Betriebssystem zu (siehe: Seiten 15, 17 und 19). Besteht bereits ein Zugriff auf das Terminal können Informationen zum Betriebssystem unter anderem über die Befehle:

```
cat /etc/os-release oder lsb_release -d
```

ausgegeben werden. Informationen zum verwendeten Kernel lassen sich über den Aufruf:

```
uname -r
```

anzeigen.



Diese Information kann für eine RAM-Sicherung relevant sein, da das LiME-Modul, welches für die Arbeitsspeichersicherung verwendet werden kann, für den jeweiligen Kernel des Zielsystems vorkompiliert werden muss. Ist auch das Betriebssystem bekannt, kann eine Recherche nach zu erwartenden Funktionen für das jeweilige Gerät auf der Projektseite des Systems erfolgen. Beispielhaft sei der Funktionsumfang eines Xiaomi Redmi 7 unter Ubuntu Touch gezeigt, wie er unter [devices.ubuntu-touch.io](https://devices.ubuntu-touch.io) aufgerufen werden kann (Bild 36).

Actors:	✓ 24+ hours battery lifetime	✓ Bluetooth
✓ Manual brightness	✗ 7+ days stability	✓ Flight mode
✓ Torchlight		🌐 🌐 Hotspot
✓ Vibration	GPU:	✓ WiFi
	✓ Boot into UI	
Camera:	🌐 Hardware video playback	Sensors:
✓ Flashlight	Misc:	✗ Automatic brightness
✓ Photo	✓ Anbox patches (deprecated)	✗ Fingerprint reader
✓ Video	✓ AppArmor patches	✓ GPS
✓ Switching between cameras	✓ Battery percentage	✓ Proximity
	✗ Offline charging	✓ Rotation
Cellular:	✓ Online charging	✓ Touchscreen
✓ Carrier info, signal strength	✗ Recovery image	
✓ Data connection	✗ Reset to factory defaults	Sound:
🌐 Dual SIM functionality	✓ SD card detection and access	✓ Earphones
✓ Incoming, outgoing calls	✓ RTC time	✓ Loudspeaker
✓ MMS in, out	✓ Shutdown / Reboot	✓ Microphone
✓ PIN unlock	✗ Wireless External monitor	✓ Volume control
✓ SMS in, out	? Waydroid	
✓ Change audio routings	Network:	USB:
✓ Voice in calls		✗ MTP access
		✗ ADB access
Endurance:		

**Bild 36:** Funktionsumfang eines Redmi 7 unter Ubuntu Touch (Quelle: <https://devices.ubuntu-touch.io/device/onclite>, Aufruf: 23.02.2023)

Hierbei wird ersichtlich, dass auf diesem Gerät kein „Offline charging“ verfügbar ist. Dies hat zur Folge, dass das ausgeschaltete Gerät bei der Verbindung mit einer Stromquelle in das Betriebssystem bootet. Da MTP und ADB nicht verfügbar sind, können diese Protokolle zur Datensicherung nicht verwendet werden. Die Überprüfung der Projektseite spart Zeit, die bei untauglichen Sicherungsversuchen verloren ginge. Häufig kann aus der Gerätedokumentation heraus eine Installationsanleitung aufgerufen werden. So lässt sich nachvollziehen, wie das System nach den Vorstellungen der Entwickler zu installieren ist. Ist etwa die Installation auf eine MicroSD-Karte vorgesehen, ist dies bei der späteren Sicherung zu beachten, da das Entfernen



der SD-Karte im Betrieb zu Datenverlust und Beschädigungen am System führen kann. Anlaufpunkte für Gerätedaten stellen beispielsweise die folgenden Projektseiten dar:

**Tabelle 2:** Projektseiten der mobilen Linux-Distributionen

Droidian	<a href="https://devices.droidian.org/devices/">https://devices.droidian.org/devices/</a>
Maemo Leste	<a href="https://leste.maemo.org/Category:Device">https://leste.maemo.org/Category:Device</a>
Mobian	<a href="https://wiki.mobian-project.org/doku.php?id=devices">https://wiki.mobian-project.org/doku.php?id=devices</a>
PostmarketOS	<a href="https://wiki.postmarketos.org/wiki/Devices">https://wiki.postmarketos.org/wiki/Devices</a>
SailfishOS	<a href="https://wiki.merproject.org/wiki/Adaptations/libhybris">https://wiki.merproject.org/wiki/Adaptations/libhybris</a>
Ubuntu Touch	<a href="https://devices.ubuntu-touch.io/">https://devices.ubuntu-touch.io/</a>

Sollen flüchtige Daten (RAM, tmp-Verzeichnis) gesichert werden, ist für die **Stromversorgung** des Geräts Sorge zu tragen um eine Notabschaltung zu verhindern. Da auch andere Sicherungsformen den Betrieb des Geräts erforderlich machen, sollte der Akku vor der Sicherung vollständig geladen werden. Steht dabei „Offline-Charging“ wie im gezeigten Beispiel nicht zur Verfügung, kann das Laden auch im Fastboot- oder Recovery-Modus erfolgen.

Unter Android- und iOS sind bereits im Betriebssystem Methoden integriert, um eine Sperrung des Displays oder gar die Löschung der Nutzerdaten aus der Ferne zu veranlassen [66 S. 28]. Auch für Mobilgeräte mit Linux-Betriebssystemen ist dieser Aspekt relevant. Neben professionellen Diensteanbietern, welche „remote wipe“-Lösungen für Linux-Systeme anbieten, existieren auch Fernzugriffslösungen wie tmate [69], die vollumfänglichen Zugriff auf die Konsole des Linux-Systems erlauben. Zur Unterbindung möglicher Fernzugriffe ist der sogenannte „Flugmodus“ oder „Offline Modus“ zu aktivieren bzw. sind die Interfaces zu deaktivieren, welche aktiv für eine Internetverbindung verwendet werden. Ist ein Zugriff auf die Bedienelemente zur Deaktivierung der Datenverbindung nicht möglich, kann eine **Abschirmung** durch die Verwendung von Schirmungstaschen (Faraday Bags) oder die Entnahme der SIM-Karte erfolgen. Letzteres kann jedoch den Verlust des Zugriffs auf die SIM-Daten bedeuten, wenn die SIM-PIN im Rahmen der Sicherung nicht bekannt ist.

## 4.3 Datensammlung

Das Vorgehensmodell nach Muth [65 S. 38] kann auch auf Linux-Mobilgeräte angewandt werden, da bisherige Erkenntnisse (siehe 4.2 Operationale Vorbereitung – Seite 34) dem nicht entgegenstehen. Dennoch sind Details im Ablauf zu beachten. Das Vorgehen bei der Datensammlung unterscheidet sich im Ablauf; je nachdem, ob das Gerät bereits eingeschaltet ist oder nicht und ob die Zugangsdaten zum Gerät bereits bekannt sind.

### 4.3.1 Eingeschaltetes Gerät

Der **Verbindungsaufbau** zum laufenden Gerät erfolgt für gewöhnlich über die USB-Schnittstelle. SSH- oder ADB-Verbindungen sind zwar auch über Wifi oder Bluetooth möglich, diese bergen aber ein höheres Risiko für Störungen, Übertragungsfehler oder bereits initiierte Remote-Wipe-Aktionen.

Jede Aktion, die nun auf dem zu sichernden Gerät durchgeführt wird, führt zu potentiellen Veränderungen am Gerätespeicher. Insbesondere droht der Verlust von Informationen, die im Arbeitsspeicher vorgehalten werden. Die **Sicherung des RAM-Speichers** sollte aufgrund seiner Flüchtigkeit vor weiteren Maßnahmen erfolgen [64 S. 34]. Im Anschluss kann eine **Sicherung des internen Flash-Speichers** erfolgen. Hierbei sollten geeignete systemeigene Programme und Funktionen genutzt werden, um die Beeinflussung des Speicherinhalts möglichst gering zu halten [70 S. 38]. Geeignet erscheinen dabei die Programme „dd“ und „cat“, welche in nahezu jedem unixoiden Betriebssystem enthalten sind [71 S. 25], da sie zu den „*coreutils*“ gehören. Dabei wird sich der Umstand zu Nutze gemacht, dass unter Linux alles als Datei betrachtet wird („everything is a file“) [72 S. 111] und so auch Block-Devices wie Speichergeräte an Kopierbefehle übergeben werden können. Auch die **Sicherung des externen Flash-Speichers** kann so über eine bestehende SSH-Verbindung oder direkt über die Terminal-Anwendung des Mobilgeräts auf

einen angeschlossenen USB-Speicher erfolgen. Bei einer vorliegenden Verschlüsselung des externen Speichers ist dabei auch eine unverschlüsselte Sicherung des eingebundenen Speichergeräts möglich. Im Falle einer Installation des Systems auf dem externen Speicher lassen sich temporäre Daten oft nur direkt über das laufende Gerät sichern.

#### **4.3.2 Eingeschaltetes Gerät mit unbekannten Zugangsdaten**

Ist das Gerät mit einem Sperrbildschirm gesichert und konnten die Zugangsdaten im Rahmen der operationalen Vorbereitung nicht ermittelt werden, wird der Verbindungsaufbau erwartungsgemäß erschwert. Dennoch sollte der Versuch unternommen werden, eine USB-Verbindung herzustellen und verfügbare Protokolle abzufragen. Ist das verwendete System bekannt, kann eine Recherche nach bekannten Sicherheitslücken erfolgen, welche einen Zugriff auf das Gerät erlauben. Möglich ist, dass der Gerätespeicher bereits als Massenspeichergerät über USB freigegeben wird. So kann bereits teilweise eine logische Datensicherung erfolgen. Weitere Zugriffsversuche können über eine eventuell aktive SSH-Verbindung durchgeführt werden. Dabei kann auf bekannte Standardzugangsdaten (siehe Seite 22) zurückgegriffen werden, sofern diese durch den Nutzer nicht geändert wurden. Je nach Gerät kann auch über einen Neustart in den Bootloader-Modus eine Sicherung des RAM-Speichers möglich sein. Diese Möglichkeit wird im Abschnitt zur RAM-Sicherung näher erläutert. Sind die Ansätze der Live-Sicherung ausgeschöpft, kann das Gerät ausgeschaltet werden und die Sicherung entsprechend des Vorgehensmodells für ausgeschaltete Geräte erfolgen. Für gewöhnlich wird der Inhalt des /tmp Verzeichnisses bei einem Neustart in das Betriebssystem geleert [72 S. 101]. Es besteht jedoch auch die Möglichkeit, dass der Nutzer ein Löschskript an das Herunterfahren des Gerätes gekoppelt hat. Daher ist vom Herunterfahren des Systems abzusehen und das Gerät über die Entnahme des Akkus oder durch längere Betätigung der {Power} Taste auszuschalten.

### 4.3.3 Ausgeschaltetes Gerät

Die Möglichkeit der Sicherung flüchtiger Daten ist bei ausgeschalteten Geräten zwar wahrscheinlich nicht gegeben, doch ein Start des Geräts in das Betriebssystem kann die Löschung temporärer Dateien auslösen und sollte daher vermieden werden [72 S. 101].

Zunächst kann hier die **Sicherung des externen Flash-Speichers** erfolgen. Dazu wird das Medium (in der Regel eine µSD-Karte) aus dem Gerät entfernt und unter Verwendung eines forensischen Write-Blockers gesichert. Im Anschluss kann auch die **Sicherung des internen Flash-Speichers** durchgeführt werden. Dabei sollten Methoden bevorzugt werden, welche die Ausführung des Betriebssystems nicht erfordern. Yang [73 S. 70] erläutert dazu, dass die Datenintegrität der Nutzerdaten bei Sicherungen über den Bootloader erhalten bleibt, selbst wenn die physische Extraktion mehrfach durchgeführt wird. Bei der **Bootloader-Sicherung** wird der Umstand genutzt, dass die Hersteller der SoCs Protokolle und Schnittstellen vorsehen, die Firmware-Updates und Software-Reparaturen zulassen, auch wenn ein Zugriff auf das Betriebssystem nicht mehr möglich ist. Die anwendbaren Methoden unterscheiden sich entsprechend, je nachdem welcher SoC im Mobilgerät verbaut wurde und wie der Aufruf des jeweiligen Bootloader/BootROM-Modus<sup>4</sup> durch den Hersteller implementiert wurde. Auf die Möglichkeiten der Bootloader-Sicherung wird im entsprechenden Abschnitt eingegangen. Ist eine Sicherung über den BootROM-Modus nicht möglich, kann der Versuch einer **Recovery-Sicherung** unternommen werden [75 S. 139]. Die überwiegende Anzahl verfügbarer Mobilgeräte mit Linux-Betriebssystem wurde vormals mit Android betrieben. Zur Installation des Linux-Systems war regelmäßig die Entsperrung des Bootloaders erforderlich (siehe Anhang Seite A). Für die forensische Sicherung hat dies den Vorteil, dass kein Datenverlust durch das

---

4 Die Verwendung des Begriffs „Bootloader“ ist mehrdeutig. Bei Start des Geräts wird zunächst der Primary Bootloader geladen (BootROM). Der anschließend geladene Secondary Bootloader stellt Protokolle wie Fastboot oder den Samsung-Odin-Mode zur Verfügung. Der Secondary Bootloader ist auch gemeint, wenn vom „Bootloader Unlock“ die Rede ist. Auch der Fastboot-Modus wird häufig als Bootloader-Modus bezeichnet, so auch bei dem Aufruf „adb reboot bootloader“. Im Rahmen dieser Ausarbeitung werden Methoden, die auf den BootROM abzielen als Bootloader-Sicherung bezeichnet. In anderen Quellen findet sich der Begriff „Flasher Tool“ für Anwendungen, die mit dem BootROM kommunizieren [74 S. 4].

nachträgliche Entsperrn des Bootloaders zu erwarten ist und eine Custom Recovery auf das Gerät geladen werden kann, welche einen Zugriff auf den Gerätespeicher gestattet [76 S. 276]. In einigen Fällen ist bereits eine Custom Recovery installiert. Bei Geräten mit x86/64-Architektur kann eine **Sicherung über ein Live-System** erfolgen [77 S. 279].

Zu einigen Geräte existieren auch spezifische Methoden, die nicht eindeutig einer der vorangegangenen Sicherungs-Kategorien zugeordnet werden können. Diese Methoden werden im Abschnitt „gerätespezifische Sicherungsmethoden“ vorgestellt. Darunter fallen auch Linux-First Geräte, die keine Implementierung von Fastboot oder einem Recovery-Modus vorsehen.

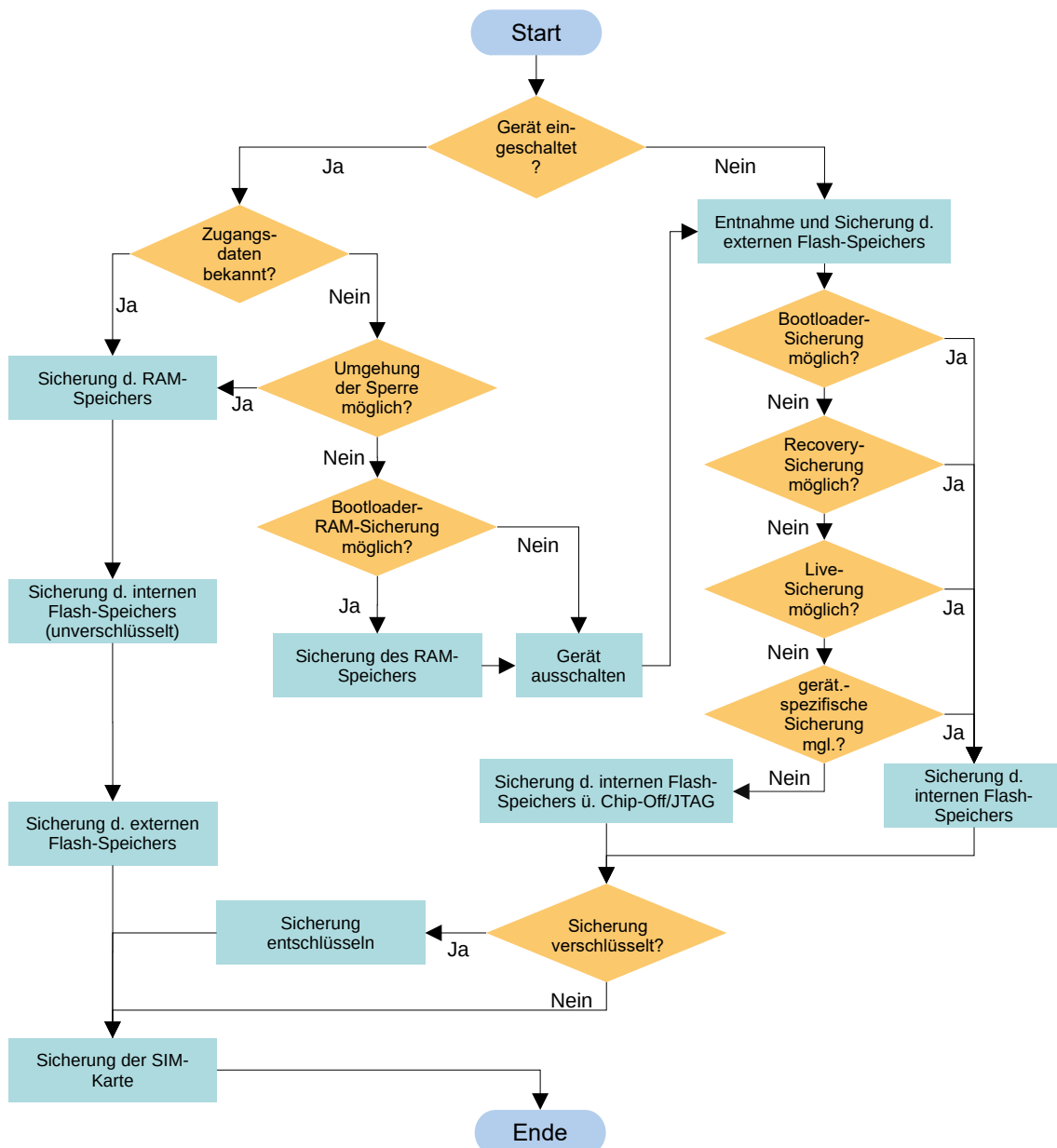
#### 4.3.4 Vorgehen bei der Datensammlung

Daten von Mobilgeräten können in unterschiedlichem Umfang gesichert werden. So lässt sich die Datenmenge an die jeweilige Zielsetzung der Untersuchung anpassen. Das NIST unterscheidet hier sechs Stufen der Datenakquise [66 S. 17]:

**Tabelle 3:** Stufen der Datensicherung (nach NIST)

<b>Manuelle Extraktion:</b>	Betrachten und Dokumentieren der Inhalte des Geräts über das Display (z.B. durch Fotografie).
<b>Logische Extraktion:</b>	Übertragen von Daten zwischen dem Mobilgerät und der forensischen Workstation. Ein vollständiges Dateisystem lässt sich nur mit Systemverwalter-Rechten übertragen.
<b>Physische Extraktion:</b>	Bitweise Kopie des Speicherinhalts (auch Hex-Dump).
<b>JTAG</b>	Extraktion von Daten über ein standardisiertes Test-Interface auf der Platine des Geräts.
<b>Chip-Off</b>	Entfernen des Flash-Speichers von der Platine des Geräts zur direkten Auslesung.
<b>Micro Read</b>	Erfassen der Statusänderungen direkt am Chip mittels eines Elektronenmikroskops.

Im Rahmen dieser Ausarbeitung wird die physische Extraktion angestrebt, da hierbei Daten im größtmöglichen Umfang gesichert werden. Entsprechend der vorangegangenen Erläuterungen wurde ein Vorgehensmodell der Datensammlung erstellt, welches die Flüchtigkeit der Daten berücksichtigt und somit einen maximalen Extraktionsumfang erreichen soll:



**Bild 37:** Ablauf der Datensammlung

Die Sicherung der SIM-Karte und die Möglichkeiten der JTAG- und Chip-Off-Sicherungen wurden nur der Vollständigkeit halber erfasst. Das verwendete Betriebssystem hat bei diesen Maßnahmen keinen Einfluss auf die Sicherung.

Auch das Verfahren „Micro Read“ als höchste Stufe des NIST-Modells soll nicht weiter betrachtet werden, da das verwendete Betriebssystem hierbei ebenfalls nachrangig ist.

Aktuell werden physische Datenextraktionen von Mobilgeräten überwiegend im RAW-Format erzeugt [78 S. 4],[73 S.74]. Diese Praxis geht auf eine Zeit zurück, in der mobile Endgeräte über eine eher geringe Speichergröße verfügten. Neuere Smartphones werden hingegen mit bis zu 1TB Speicher vertrieben [79]. Auch die Unterstützung für µSD-Karten mit mehreren TB Speicher ist bei vielen aktuellen Geräten gegeben. Für die Sicherung von Festplatten und SSDs aus klassischen Rechner-Systemen ist die komprimierte Sicherung als forensisches Image üblich. Hierbei hat sich das EWF (Expert-Witness-Format) als „Quasistandard“ [80 S. 129] etabliert. Bereits 1998 als proprietäres Dateiformat der Firma EnCase eingeführt [81 S.186], existiert mit „libewf“ [82] mittlerweile eine freie Bibliothek zum Umgang mit EWF-Dateien. Das Format bietet im forensischen Kontext auch weitere Vorteile. So können Informationen zum Vorgang (z.B. Case Number), zum Gerät / Datenträger oder zur sichernden Person direkt im Abbild hinterlegt werden. Marktübliche forensische Analysetools (Cellebrite Physical Analyzer, Magnet Axion, Oxygen Forensic Detective) und freie Software-Lösungen wie Autopsy oder IPED können das Format interpretieren. Das gänzlich quelloffene AFF (Advanced Forensics Format) bietet einen vergleichbaren Funktionsumfang, konnte sich aber bisher nicht gegen das stärker verbreitete EWF durchsetzen [80 S. 129].

Wo möglich, sollen die entsprechenden Anwendungen der libewf-Bibliothek im Rahmen der Flash-Speicher-Sicherung Anwendung finden:

`ewfacquire`

für externe Flash-Speicher und Geräte im Massenspeichermodus

`ewfacquiestream`

für Ausgaben auf stdin, z.B. über SSH oder ADB

Die Nutzung der Anwendungen dd und cat auf den Ziel-Geräten steht diesem Vorgehen nicht entgegen, da der RAW-Output dieser Befehle an ewfacquiestream übergeben werden kann.

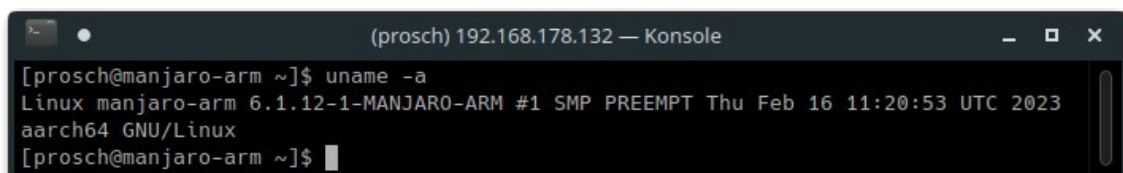
## 5 RAM-Sicherung

Im Arbeitsspeicher des Geräts werden neben aktuell verwendeter Daten potentiell auch Passwörter und Kryptoschlüssel vorgehalten [83 S. 17-22]. Diese Informationen können im weiteren Verlauf der Untersuchung von entscheidender Bedeutung sein. Einige ermittelte Methoden, welche die Extraktion von Inhalten aus dem Arbeitsspeicher mobiler Geräte erlauben, sollen folgend vorgestellt werden.

### 5.1 LiME

Bei dem Linux Memory Extractor (LiME) handelt es sich um ein Loadable Kernel-Module (LKM), das die Arbeitsspeichersicherung unter Linux und Android ermöglicht [84]. Hierbei wird vorausgesetzt, dass die Zugangsdaten zum Gerät bekannt sind. Das Modul muss dabei für den Kernel kompiliert werden, welcher aktuell auf dem Gerät verwendet wird. Um den Einfluss auf den Untersuchungsgegenstand möglichst gering zu halten, sollte das Modul zudem extern erzeugt werden. Dieser Vorgang wurde für das Pine64 Pinephone (siehe Seite 24) nachvollzogen.

Dazu wird zunächst eine SSH-Verbindung zum Gerät aufgebaut und der aktuell verwendete Kernel abgefragt:

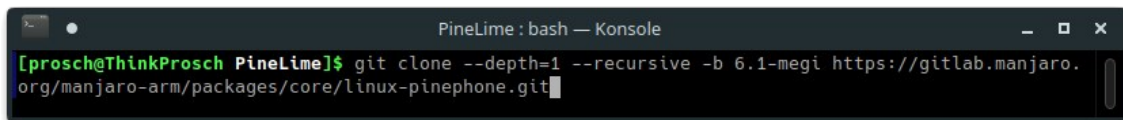
A screenshot of a terminal window titled "(prosch) 192.168.178.132 — Konsole". The terminal shows the command "[prosch@manjaro-arm ~]\$ uname -a" and its output: "Linux manjaro-arm 6.1.12-1-MANJARO-ARM #1 SMP PREEMPT Thu Feb 16 11:20:53 UTC 2023 aarch64 GNU/Linux". The prompt "[prosch@manjaro-arm ~]\$ " is visible at the bottom.

```
(prosch) 192.168.178.132 — Konsole
[prosch@manjaro-arm ~]$ uname -a
Linux manjaro-arm 6.1.12-1-MANJARO-ARM #1 SMP PREEMPT Thu Feb 16 11:20:53 UTC 2023
aarch64 GNU/Linux
[prosch@manjaro-arm ~]$
```

**Bild 38:** Abfrage des verwendeten Kernels über SSH

Die verwendete Kernel-Version lautet hier: 6.1.12-1-MANJARO-ARM. Dieser Kernel kann aus dem Manjaro Gitlab-Portal entsprechend der verwendeten Version über „git“ bezogen werden:





```
PineLime: bash — Konsole
[prosch@ThinkProsch PineLime]$ git clone --depth=1 --recursive -b 6.1-megi https://gitlab.manjaro.org/manjaro-arm/packages/core/linux-pinephone.git
```

Bild 39: Klonen der Kernel-Quelldateien

Zum Kompilieren von ARM-Paketen auf dem x86\_64-Hostsystem sind zudem die folgenden Pakete erforderlich: `aarch64-linux-gnu-gcc`, `flex` und `bison`.

Entsprechend der Instruktionen des Entwicklers „Danct12“ [85] ist die Zielarchitektur in der Datei `makepkg.conf` (Kopie der Datei des Hostsystems) in der Zeile: „`CARCH=`“ anzupassen. Abweichend von diesen Instruktionen ist jedoch nicht „`ARM64`“, sondern „`aarch64`“ in der Datei zu hinterlegen. Die passenden Umgebungsvariablen werden über die Befehle:

```
export ARCH=aarch64
```

und

```
export CROSS_COMPILE=aarch64-linux-gnu-
```

gesetzt. Anschließend kann der Kernel über `makepkg` unter Verweis auf die angepasste `makepkg.conf` erzeugt werden. Benötigte Abhängigkeiten werden dabei automatisch erfüllt:



```
linux-pinephone: makepkg — Konsole
[prosch@ThinkProsch linux-pinephone]$ makepkg --config ./makepkg.conf -s
==> Erstelle Paket: linux-pinephone 6.1.12-1 (Sa 18 Mär 2023 21:59:54 CET)
==> Prüfe Laufzeit-Abhängigkeiten...
==> Prüfe Buildtime-Abhängigkeiten...
==> Installiere fehlende Abhängigkeiten...
[sudo] Passwort für prosch:
Abhängigkeiten werden aufgelöst ...
Nach in Konflikt stehenden Paketen wird gesucht ...

Pakete (3) dtc-1.6.1-4  uboot-tools-2023.01-1  xmlto-0.0.28-4

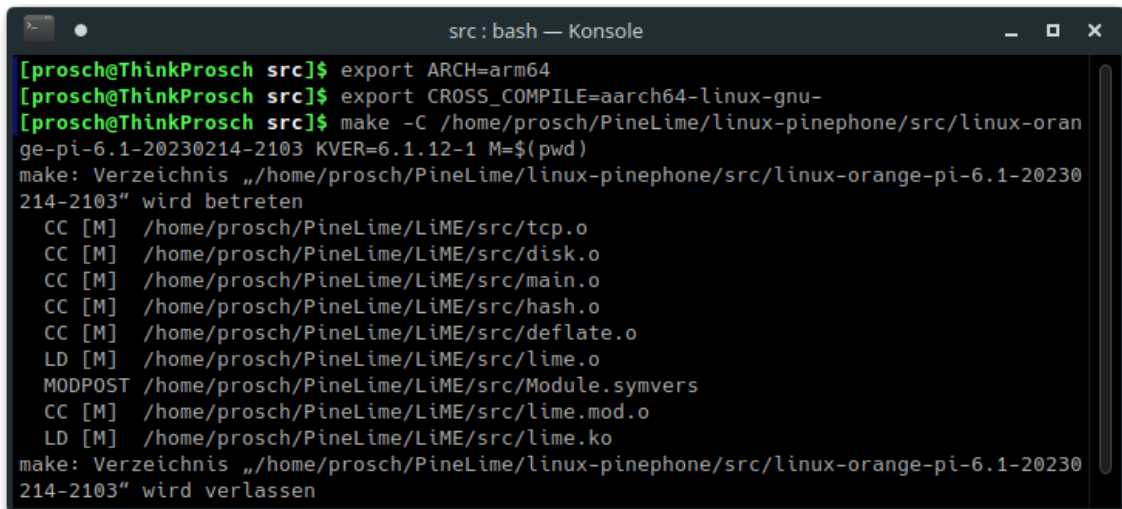
Gesamtgröße des Downloads:      0,35 MiB
Gesamtgröße der installierten Pakete: 1,38 MiB

:: Installation fortsetzen? [J/n] J
```

Bild 40: Erstellen des Linux-Kernels

Auch die Quelldaten des LiME-Modus können über „git“ bezogen werden [84]. Aus dem geklonten `src`-Verzeichnis heraus werden hierbei wieder die ARM-

Umgebungsvariablen gesetzt und der Pfad des erstellten Kernels an den „make“-Befehl übergeben<sup>5</sup>:



```
src: bash — Konsole
[prosch@ThinkProsch src]$ export ARCH=arm64
[prosch@ThinkProsch src]$ export CROSS_COMPILE=aarch64-linux-gnu-
[prosch@ThinkProsch src]$ make -C /home/prosch/PineLime/linux-pinephone/src/linux-oran
ge-pi-6.1-20230214-2103 KVER=6.1.12-1 M=$(pwd)
make: Verzeichnis „/home/prosch/PineLime/linux-pinephone/src/linux-orange-pi-6.1-20230
214-2103“ wird betreten
CC [M] /home/prosch/PineLime/LiME/src/tcp.o
CC [M] /home/prosch/PineLime/LiME/src/disk.o
CC [M] /home/prosch/PineLime/LiME/src/main.o
CC [M] /home/prosch/PineLime/LiME/src/hash.o
CC [M] /home/prosch/PineLime/LiME/src/deflate.o
LD [M] /home/prosch/PineLime/LiME/src/lime.o
MODPOST /home/prosch/PineLime/LiME/src/Module.symvers
CC [M] /home/prosch/PineLime/LiME/src/lime.mod.o
LD [M] /home/prosch/PineLime/LiME/src/lime.ko
make: Verzeichnis „/home/prosch/PineLime/linux-pinephone/src/linux-orange-pi-6.1-20230
214-2103“ wird verlassen
```

**Bild 41:** Kompilieren des LiME-Moduls gegen den erstellten Kernel

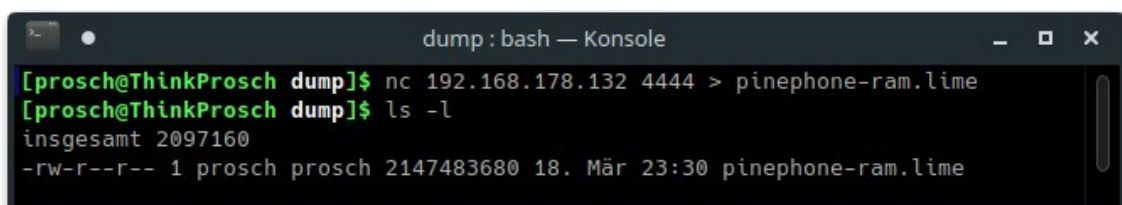
Das erstellte Modul lime.ko wird im Anschluss auf das Pinephone übertragen:

```
scp lime.ko prosch@192.168.178.132:/home/prosch/.
```

Auf der Kommandozeile des Pinephone kann dieses Modul nun geladen werden. Vorliegend wird die Sicherung über TCP gewählt:

```
sudo insmod ./lime.ko „path=tcp:4444 format=lime“
```

Auf der forensischen Workstation wird dieser Stream entgegengenommen:



```
dump: bash — Konsole
[prosch@ThinkProsch dump]$ nc 192.168.178.132 4444 > pinephone-ram.lime
[prosch@ThinkProsch dump]$ ls -l
insgesamt 2097160
-rw-r--r-- 1 prosch prosch 2147483680 18. Mär 23:30 pinephone-ram.lime
```

**Bild 42:** RAM-Sicherung des Pinephones über LiME

Hierbei wird ein 2GiB-Dump erzeugt. Dies entspricht auch dem tatsächlichen RAM-Speicher des verwendeten Pinephones.

<sup>5</sup> Die Bezeichnung „linux-orange-pi“ wird verwendet, da Manjaro den „megi-Kernel“ als Quelle nutzt. Dieser wird für diverse Geräte mit Allwinner SoC wie auch dem Orange Pi bereit gestellt.

## 5.2 AVML

Unabhängig von der verwendeten Kernel Version erlaubt AVML (Acquire Volatile Memory for Linux) [86] die Sicherung des Arbeitsspeichers unter Linux. Mögliche Datenquellen für die Sicherung sind hier gemäß der offiziellen Dokumentation die Einbindepunkte:

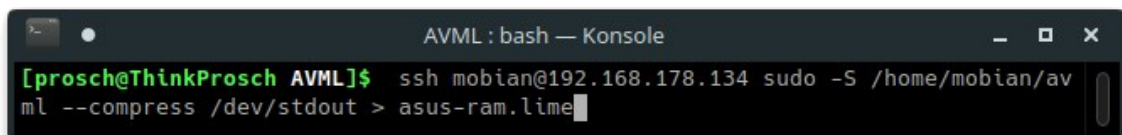
- /dev/crash
- /dev/kcore
- /dev/mem

Die Anwendung kann vorkompiliert für die x86\_64 Architektur von dem Github-Portal bezogen werden. Die Erstellung für ARM64 ist ebenfalls möglich [87]. Nachfolgend wird die Sicherung des Arbeitsspeichers des ASUS Transformer Book T100TA (siehe Seite 26) dargestellt.

Zunächst wird die Anwendung AVML auf das Zielgerät übertragen:

```
scp avml mobian@192.168.178.134:/home/mobian/.
```

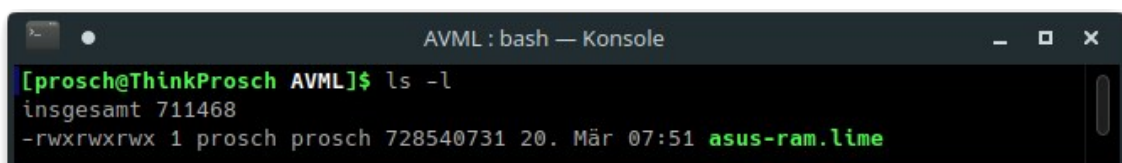
Über SSH kann anschließend eine komprimierte Sicherung des Arbeitsspeichers direkt auf die forensische Workstation erfolgen:



```
AVML : bash — Konsole
[prosch@ThinkProsch AVML]$ ssh mobian@192.168.178.134 sudo -S /home/mobian/avml --compress /dev/stdout > asus-ram.lime
```

**Bild 43:** RAM-Sicherung des Asus T100TA über AVML

Da AVML nur die Speicherung unter einem lokalen Pfad vorsieht, wird als „Zieldatei“ die Standardausgabe stdout gewählt. Aus den 4GiB Arbeitsspeicher des Geräts wurde ein komprimierter Dump von ca. 729MB erzeugt.



```
AVML : bash — Konsole
[prosch@ThinkProsch AVML]$ ls -l
insgesamt 711468
-rwxrwxrwx 1 prosch prosch 728540731 20. Mär 07:51 asus-ram.lime
```

**Bild 44:** Erstellte komprimierte AVML-Sicherung im Lime-Format

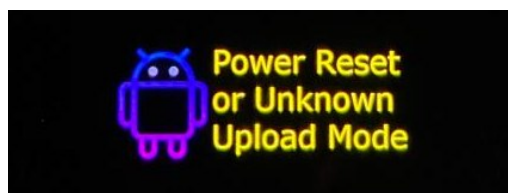
### 5.3 SUC / sbboot\_dump

Der „Download“-Mode von Samsung-Mobilgeräten ist im Rahmen forensischer Untersuchungen schon mehrfach betrachtet worden [88 S. 483]. Weniger bekannt ist die Implementierung des „Upload“-Modus. Dieser wird z.B. bei einem Gerätedefekt aufgerufen und gibt Entwicklern Zugriff auf den Arbeitsspeicher des Geräts, um eine Fehleranalyse zu ermöglichen. Das Tool SUC (Samsung Upload Client) erlaubt die Kommunikation mit Geräten in diesem Modus [89]. Unter Android lässt sich der Upload-Mode über einen Dialer-Code erzwingen [90] Diese Möglichkeit besteht bei der Verwendung von mobilen Linux-Betriebssystemen bisher nicht. Jedoch existieren Tastenkombinationen, welche den Aufruf des Upload-Modus triggern [91].

Für das vorliegende Samsung S9 (siehe Seite 28) konnte die folgende Kombination ermittelt<sup>6</sup> werden, die den Upload-Mode aus dem laufenden System heraus erzwingt:

- Halten der {leiser} Taste über die gesamte Dauer des Vorgangs,
- neunmaliges Betätigen der {Power} Taste,
- fünfmaliges Betätigen der {lauter} Taste und
- dreimaliges Betätigen der {Power} Taste

Nach der letzten Betätigung der Power Taste schaltet das Display des Geräts abrupt ab und es erscheint in der Folge die Meldung „Upload Mode“



**Bild 45:** Upload Mode auf dem Samsung Galaxy S9

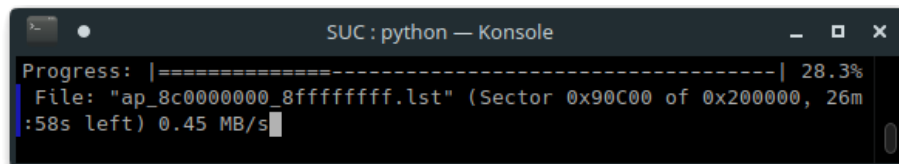
In diesem Modus kann das Telefon über USB mit der Workstation verbunden werden. Unter Verwendung des Python-Skripts: samupload.py wird eine Übersicht der allgemeinen Geräteinformationen und erkannter

<sup>6</sup> Die Kombination wird bei der Verwendung kommerzieller Forensik-Software für andere Samsung-Modelle beschrieben. Über einen Versuch ließ sich die Funktion auch bei dem vorliegenden Gerät nachweisen.

Speicherbereiche ausgegeben. Diese Speicherbereiche können über ihre jeweiligen Adressen oder nach Namen extrahiert werden. Der Aufruf:

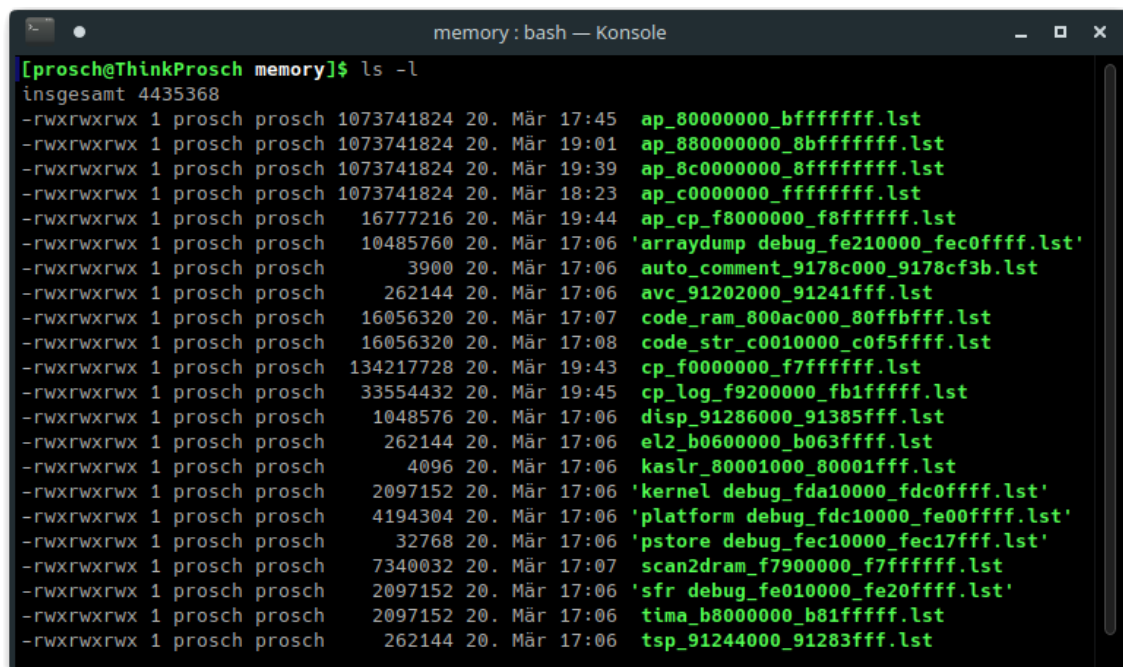
```
python samupload.py all
```

stößt eine Sicherung aller aufgefundenen Speicherbereiche an.



**Bild 46:** RAM-Sicherung des Samsung Galaxy S9 über den SUC

Die dabei extrahierten Daten überschreiten im Umfang die Speichergröße des Geräts, da über die Adressierung Dopplungen entstehen.



**Bild 47:** Extrahierte Speicherbereiche des Samsung Galaxy S9

Die vier „ap“-Bereiche von je einem GiB Größe entsprechen hier den vier GiB des Arbeitsspeichers des Geräts. Der Versuch, den kompletten Speicher des Geräts über den „range“ Parameter als eine Datei zu sichern, erzeugte regelmäßig nur einen Dump von zwei GiB.

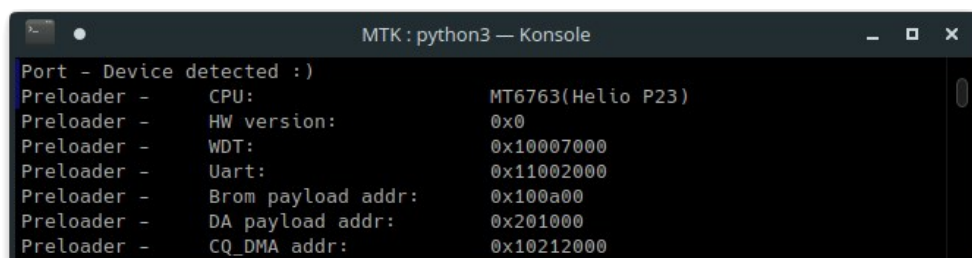
## 5.4 MTKClient

Geräte mit MTK (Mediatek) SoC stellen über den Bootloader verschiedene Bootmodi bereit, welche die Diagnose und Fehlerbeseitigung durch die Gerätehersteller erlauben. Für die Sicherung relevant sind der Preloader- und der Bootrom-Modus (BROM-Modus). Diese Modi erfordern eine USB-Verbindung und ein wartendes Client-Programm, um das Gerät im Bootloader-Modus abzufangen und zu halten [92]. Ein Python-Tool, das mit MTK-Geräten im BROM- oder Preloader-Modus kommuniziert, ist der MTKClient [93]. Auf die Fähigkeiten des MTKClients wird im Rahmen der Sicherung des internen Flash-Speichers näher eingegangen (siehe Seite 76). Für die RAM-Sicherung ist das Tool ebenfalls relevant, da mit dem undokumentierten Parameter „memdump“ Abbilder der Arbeitsspeicherbereiche des Mobilgeräts erstellt werden können.

Die Sicherung wird hier am Beispiel des Volla Phones / Gigaset GS290 (siehe Seite 27) demonstriert. Dazu wird das eingeschaltete Gerät über USB mit der forensischen Workstation verbunden und der Befehl:

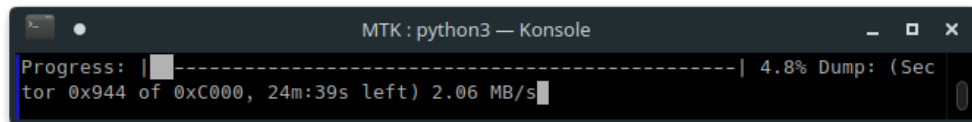
```
python mtk memdump volla-mem.dump
```

auf der Workstation abgesetzt. Das Tool wartet nun auf ein Gerät im BROM-Modus. Dazu wird im Power-Menü des Volla Phones ein Neustart ausgelöst und unmittelbar die Tastenkombination **{leiser}** und **{lauter}** betätigt und gehalten. Bei korrekter Ausführung erscheint keine Ausgabe auf dem Gerätedisplay und der MTKClient meldet die Verbindung im BROM-Modus:



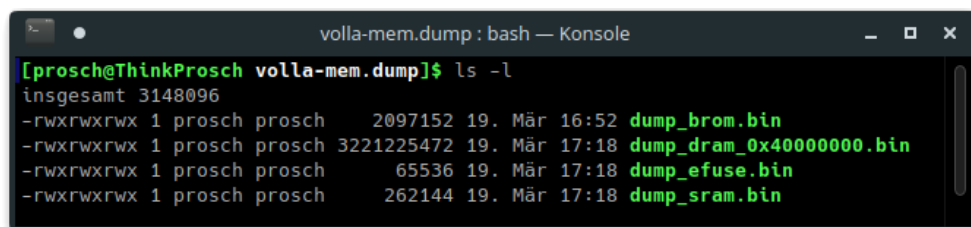
**Bild 48:** MTKClient - Verbindung im BROM-Modus

Der abgesetzte Memdump-Befehl erzeugt daraufhin eine Sicherung der Speicherbereiche in das angegebene Verzeichnis „volla-mem.dump“



**Bild 49:** MTKClient - Sicherung der Arbeitsspeicherbereiche

Die relevante DRAM-Extraktion umfasst hier 3GiB der insgesamt 4GiB des Geräts. Die verbleibenden 1GiB Speicher werden vom System reserviert und beinhalten unter anderem auch den bereits extrahierten SRAM.



**Bild 50:** Extrahierte Speicherbereiche des Volla Phones

## 5.5 Weitere Methoden

Neben den bereits genannten Methoden konnten auch weitere Möglichkeiten der RAM Sicherung ermittelt werden, die sich bei den vorliegenden Geräten jedoch nicht nachvollziehen ließen. Für einige Motorola- und Nexus-Geräte wird beschrieben [94], dass eine RAM-Sicherung im Fastboot-Modus über den Aufruf:

```
fastboot oem ramdump
```

möglich sei. Weiterhin wird für das Tool EDLClient in der zugehörigen Github-Dokumentation [95] die RAM-Sicherung im „900E“-Modus erläutert. Die Möglichkeit, diesen Modus über Fastboot auf einem Google Nexus 5X zu forcieren, wird als kritische Sicherheitslücke beschrieben [96].



## 6 Flash-Speicher-Sicherung

Die nachfolgend vorgestellten Methoden bieten zum Teil Zugriff auf den internen und den externen Flash-Speicher. Zunächst wird davon ausgegangen, dass die zu untersuchenden Geräte im eingeschalteten Zustand mit bekannten Zugangsdaten übergeben wurden. In diesem Fall ist häufig eine Sicherung des Speichers über eine TCP-Verbindung möglich. Einige Ubuntu Touch Geräte erlauben auch eine Verbindung über die Android-Device-Bridge (ADB). Folgend werden auch Methoden vorgestellt, die eine Sicherung bei unbekannten Zugangsdaten erlauben.

### 6.1 TCP-Live-Sicherung (SSH/Netcat)

Diese Form der Sicherung kommt vorwiegend bei eingeschalteten Geräten mit bekannten Zugangsdaten in Betracht (siehe Punkt 4.3.1 - Seite 38). Idealerweise wird durch die installierten Systeme der Mobilgeräte der USB-Gadget-Modus unterstützt. In diesem Fall erscheint das angeschlossene System als USB-Netzwerkkarte an der Workstation und erlaubt einen Zugriff auf das Gerät über TCP. Stellt ein untersuchtes Gerät zum Zeitpunkt der Untersuchung die USB-Gadget-Funktion nicht bereit, kann eine TCP-Verbindung über WLAN hergestellt werden. Im forensischen Kontext sollte hierbei jedoch beachtet werden, dass die Geräte über das Netzwerk nicht mit dem Internet verbunden werden. Bereits 2014 stellte der niederländische IT-Forensiker Bert Busstra fest, dass sich eine WiFi-Verbindung zur Datenextraktion aus Mobilgeräten eignet [97 S. 5]. Bei den dort untersuchten Android-Geräten gelang jedoch nur eine logische Datensicherung. Vorliegend soll eine bitidentische Kopie der Gerätespeicher erzeugt werden. Für eine Verbindung über WLAN ist zunächst die zugewiesene IP-Adresse zu ermitteln:

```
ip a | grep inet
```

(auf der Konsole des untersuchten Gerätes)



Die ermittelte Adresse im Adressbereich der Workstation wird für den späteren Aufbau der Verbindung zum Gerät benötigt. Auch die zugewiesene IP-Adresse einer USB-Verbindung lässt sich so ermitteln. Seitens der Betreiber der jeweiligen Distributionen wurden jedoch vereinzelt bereits IP-Adressen für den USB-Zugriff definiert:

**Tabelle 4:** definierte USB-IP-Adressen der mobilen Distributionen

PostmarketOS [98]	172.16.42.1
Droidian [99 S. 23]	10.15.19.82
MeeGo / SailfishOS [100]	192.168.2.15
Maemo Leste [101]	192.168.42.2
Ubuntu Touch	nicht konfiguriert

Über eine Einsicht in die Geräte wurde festgestellt, dass auf 22 der 25 Testgeräte bereits im Rahmen der Standardinstallation ein Open-SSH-Server installiert war. Auf 12 Geräten war bereits Netcat vorinstalliert. Dabei handelte es sich um Mobilgeräte mit PostmarketOS, Ubuntu Touch und Ubuntu. Wegen der höheren Verbreitung auf den Testgeräten wurde eine Verbindung über SSH zur Sicherung bevorzugt. Insbesondere Sicherungen über WLAN profitieren hierbei von der bei SSH verwendeten Verschlüsselung [102] der Datenübertragung zur Workstation, um einen potentiellen Abfluss der Daten zu verhindern. In einem Artikel des SANS Institute aus dem Jahr 2005 werden die Vorzüge von SSH im forensischen Kontext näher beschrieben [103 S. 48 bis 52]. Der Autor kommt dabei zu dem Ergebnis, dass SSH ein geeignetes Tool zur forensischen Datensicherung darstellt. Der Befehlsaufruf auf der forensischen Workstation ist dabei im Testszenario wie folgt aufgebaut:

```
sshpass -p '753159'
```

Übergabe des **Nutzerpassworts** an SSH

```
ssh user@172.16.42.1
```

Aufbau der SSH-Verbindung mit dem **Nutzerprofil**

```
"sudo -S dd if=/dev/mmcblk0"
```

Aufruf des Sicherungsbefehls auf dem mobilen Endgerät. Bei der **Erlangung der Root-Rechte** wird das Passwort über **stdin** abgefragt.

```
| ewfacquirestream ...
```

Pipe der Ausgabe des abgesetzten Befehls (dd) an die Workstation und Entgegennahme des Streams durch ewfacquirestream

```
| tee log.txt
```

Schreiben der Terminalausgabe in die Datei „log.txt“

Zuvor ist einmalig eine Verbindung über SSH mit dem Gerät herzustellen, um den Schlüsselaustausch mit der Workstation zu genehmigen. Diese Verbindung kann zudem genutzt werden um über: `lsblk`

eine Übersicht der vorhandenen Block-Devices zu erhalten und den zu sichernden Speicher auszuwählen. Der komplette Befehlsaufruf für das im Beispiel gesicherte Xiaomi Redmi 2 mit PostmarketOS ist in Bild 51 dargestellt:

**Bild 51:** Befehlsaufruf für die SSH-Sicherung des Xiaomi Redmi 2

Über diesen Aufruf wurde von dem 14 GiB Gerätespeicher eine Sicherung mit einer Größe von 2,4 GiB im EWF-Format erstellt.

Der Aufruf von `ewfacquirestream` wurde um die folgenden Parameter ergänzt:

**Tabelle 5:** Sicherungsparameter für ewfacquirestream

-C BT-2023-CP	Fallnummer „BT-2023-CP“
-D "Xiaomi Redmi 2 SSH Backup"	Beschreibung der Sicherung
-e "Christian Peter"	Name des Untersuchenden
-E "01-19"	Spur- / Beweisnummer
-N „SSH-Backup des ...“	Notizen
-c best	Bestmögliche Kompression (Standard: deflate)
-l redmi_2_ssh	Name der Logdatei für Hashwerte und Fehler (.txt)
-t redmi_2_ssh	Name der Zieldatei(en) im Format .E*

Nichtgesetzte Parameter werden nicht abgefragt und mit Standardwerten besetzt. Gemäß der zugehörigen Manpage des Programms [104] wird so eine Sicherung im EnCase6-Format mit einer Dateisplittung bei 1,4 GiB über den gesamten Speicherbereich erzeugt.

Je nach System sind für die Herstellung einer SSH-Verbindung zusätzliche Schritte erforderlich. Die Geräte Nokia N9 (MeeGo) und HTC One (Ubuntu Touch 14.04) verwenden eine DSA-Verschlüsselung (Digital Signature Algorithm) für den SSH-Schlüsselaustausch. OpenSSH unterstützt diesen Algorithmus noch immer; die Ausführung muss hierbei jedoch über den Parameter `-oHostKeyAlgorithms=+ssh-dss` forciert werden. Bei SailfishOS-Geräten ist der Aufruf: `sudo -S` durch `devel-su` zu ersetzen.

Ubuntu Touch verlangt für eine Verbindung den zuvorigen Schlüsselaustausch über MTP oder ADB [105]. Alternativ kann bei Geräten mit Ubuntu Touch 16.04 die Authentifizierung über Passwort aktiviert werden [106]:

Wechsel auf das „root“ Konto:

```
sudo su
```

Anlegen eines Arbeitsverzeichnis und Ändern der Ordnerberechtigung:

```
mkdir /var/run/sshd
```

```
chmod 755 /var/run/sshd
```

Aktivieren der Passwort-Authentifizierung:

```
echo "PasswordAuthentication yes" >> /etc/ssh/sshd_config
```

```
echo "AllowUsers phablet" >> /etc/ssh/sshd_config
```

Erzeugen der SSH-Schlüssel:

```
/usr/bin/ssh-keygen -A
```

Start von SSH:

```
/usr/sbin/sshd
```

Unter Ubuntu Touch 20.04 konnte bei Verwendung dieser Befehle keine SSH-Verbindung mittels Passwort-Authentifizierung hergestellt werden. Hier ist das Hinterlegen der auf der Workstation erzeugten Schlüssel erforderlich. Ubuntu Touch 14.04 lässt SSH-Verbindungen unter Verwendung eines Passworts zu, ohne dass Veränderungen an Konfigurationsdateien vorgenommen werden müssen.

Im Rahmen der Untersuchung des Microsoft Surface 2 (Ubuntu Mate 22.04 armhf) konnte keine OpenSSH-Installation vorgefunden werden. Hier wird auf die vorhandene Netcat-Installation zurückgegriffen und der Aufruf zum Dump des internen Speichers im Terminal des Geräts per Pipe an Netcat übergeben:

```
sudo dd if=/dev/mmcblk0
```

Aufruf des Sicherungsbefehls auf der Konsole des mobilen Endgeräts

```
| nc -l -p 8888
```

Pipe-Übergabe an Netcat im [Listener-Modus](#) auf dem [Port 8888](#)

Auf diesen Port kann nun von der Workstation aus zugegriffen werden:

```
nc 192.168.178.118 8888
```

Zugriff auf den [gewählten Port](#) auf die ermittelte IP des Geräts

```
| ewfacquirestream ...
```

Pipe-Übergabe des Netcat-Streams an ewfacquirestream

```
| tee log.txt
```

Schreiben der Terminalausgabe in die Datei „log.txt“

Der Aufruf von `ewfacquirestream` wird folgend nach dem bereits beschriebenen Muster aufgebaut.

Im Versuchsaufbau wurde so aus den vorgefundenen 29 GiB Speicher des Geräts eine Extraktion von 5 GiB erzeugt.

### 6.1.1 Suche nach der gemounteten Kryptopartition

Die Geräte Sony Xperia XA2 (SailfishOS) und Sony Xperia 5 (Droidian) nutzen eine Full Disk Encryption (FDE) zur Verschlüsselung des Home-Verzeichnisses. Die vorangegangene physikalische Sicherung des Gerätespeichers lässt so keine Einsicht in die gesicherten Nutzerdaten zu, da diese zunächst entschlüsselt werden müssen (siehe Seite 103).

Bei vollem Zugriff auf das Gerät und Kenntnis der Zugangsdaten können diese Daten jedoch auch unverschlüsselt extrahiert werden. Dies wird am Beispiel des Xperia XA2 demonstriert.

Es besteht ein SSH-Zugriff auf das Terminal des Geräts.

Der Aufruf:

```
lsblk
```

zeigt neben den vorhandenen Block-Devices auch deren jeweilige Einhängepunkte im System an. Relevant für die Sicherung ist neben dem Root-Verzeichnis („/“) das Home-Verzeichnis („/home“).

```
(defaultuser) 192.168.2.15 — Konsole
mmcblk0p76          259:44    0   20G    0 part
├── sailfish-root    252:0     0   2,5G    0 lvm  /
├── sailfish-home    252:1     0  17,6G    0 lvm
│   └── luks-5da0b92b-46d0-4445-9ef7-5b89c10bd104 252:2     0  17,6G    0 crypt /home
├── mmcblk0p77       259:45     0   20M    0 part
├── mmcblk0p78       259:46     0   2,8G    0 part
├── mmcblk0p79       259:47     0   2,8G    0 part
├── mmcblk0rpb       179:32     0    4M    0 disk
└── zram0            253:0      0  512M    0 disk  [SWAP]
[defaultuser@XperiaXA2 ~]$
```

**Bild 52:** Aufruf von lsblk zum Auffinden von Krypto-Partitionen

In Bild 52 ist ersichtlich, dass der Userdata-Partition (mmcblk0p76) zwei logische Volumes zugewiesen wurden. Hierbei handelt es sich um ein dynamisches Speicherkonzept, welches über das Logical Volume Management (LVM) eine Abstraktion des physikalischen Speichers vorsieht. Ein oder mehrere physikalische Speichergeräte oder Partitionen (hier Userdata) werden zu einer Gruppe zusammengefasst und der Speicher anschließend beliebig vielen logischen Volumes (vergleichbar Partitionen) zugewiesen [107 S. 396].

Das Volume „sailfish-home“ ist mittels LUKS verschlüsselt und wurde bereits über dm-crypt (Device Manager „crypt“ Modul) nach „luks-5da0b92b-46d0-4445-9ef7-5b89c10bd104“ entschlüsselt. Diesem Volume ist der Einhängepunkt „/home“ zugewiesen. Eine unverschlüsselte Sicherung kann entsprechend für dieses Device erfolgen. LVM- und LUKS-Volumes werden über den Device Mapper verwaltet, welcher die Grundlage für LVM und dm-crypt bildet [108]. Ein Zugriff kann daher über den Pfad „/dev/mapper/“ erfolgen.

### 6.1.2 Sicherung des entschlüsselten Volumes

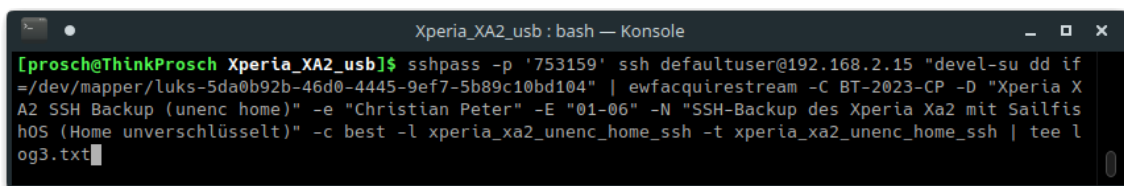
Der Aufruf der Sicherung unterscheidet sich nur im angegebenen Pfad des zu sichernden Devices. Statt über

```
devel-su dd if=/dev/mmcblk0
```

den gesamten Gerätespeicher zu sichern<sup>7</sup>, wird das Device unter dem Pfad „/dev/mapper/“ über den Aufruf:

```
devel-su dd if=/dev/mapper/luks-5da0b92b-46d0-4445-9ef7-5b89c10bd104
```

an ewfacquirestream übergeben.



**Bild 53:** Sicherung des unverschlüsselten Home-Verzeichnisses des Sony Xperia XA2

Im Gegensatz zur Sicherung des kompletten Speichers erreicht ewfacquirestream hierbei kaum eine Kompression, so dass aus 17 GiB Daten auch eine 17 GiB Sicherung erzeugt wurde. Dies begründet sich in dem

<sup>7</sup> Da es sich hier um eine SailfishOS-Installation handelt wird „devel-su“ statt „sudo -S“ verwendet.

Umstand, dass im Rahmen der Entschlüsselung des „sailfish-home“-Volumes der komplette Speicherbereich (bei einer Full Disk Encrytion) der kryptografischen Funktion ausgesetzt wird. Das logische Volume „sailfish-root“ wurde nicht verschlüsselt, so dass über den an `ewfacquirestream` übergebenen Aufruf:

```
"sudo -S dd if=/dev/mapper/sailfish-root"
```

aus den zugewiesenen 2,4 GiB Speicher ein Dump von 674 MiB erzeugt wurde.

### 6.1.3 Fazit der TCP-Live-Sicherungen

Die Systeme der Geräte: Oneplus 3T (Droidian) und Pocophone F1 (Mobian) beinhalteten weder OpenSSH noch Netcat und wurden daher nicht gesichert. In diesem Fall ist eine Sicherung des Speichers unter Verwendung von `dd` oder `cat` direkt über das Geräteterminal auf einen externen Speicher (USB-OTG, µSD-Karte) möglich. Eine Kompression der Sicherung kann im Anschluss auf der Workstation erfolgen. Bei der Droidian-Installation des Oneplus 3T handelt es sich um einen Community Port, welcher nicht offiziell vom Projekt unterstützt wird. Auf Geräten mit offiziellen Installationen war SSH vorhanden und nutzbar. Die Mobian-Installation des Pocophone F1 wird direkt vom Mobian-Projekt bereitgestellt. Die Bereitstellung eines SSH-Servers ist projektseitig nicht vorgesehen. Diesbezüglich findet sich ein Hinweis in der Projektdokumentation, wonach ein SSH-Server durch den Nutzer selbst zu installieren sei [109]. Auch Ubuntu Mate beinhaltet keinen vorinstallierten SSH-Server. Hier kann jedoch auf das bereitgestellte Tool `netcat` zurückgegriffen werden. Bei dem Gerät ASUS Transformer Pad TF300T gelang eine Sicherung erst nach mehreren Versuchen, da das System in unregelmäßigen Abständen einfror.

Die durchgeführten Sicherungen wurden dokumentiert und die Sicherungszeiten erfasst. Eine vollständige Übersicht der TCP-Sicherungen ist

im Anhang der Arbeit (Seite BB) einzusehen. Diese folgt dabei dem dargestellten Schema:

**Tabelle 6:** TCP-Live-Sicherungen (Auszug)

Gerät	System	Protokoll	Medium	Speicher	Backup	Dauer
Redmi 2	PostmarketOS	SSH	USB	14 GiB	2,4 GiB	22min 29s
Surface 2	Ubuntu Mate	Netcat	WiFi	29 GiB	5 GiB	2h 58min 9s

Zusammenfassend lässt sich festhalten: Von den 25 vorhandenen Testgeräten ließen sich 23 Geräte über eine TCP-Verbindung sichern. Unter Verwendung einer USB-Verbindung erlaubten 10 Geräte einen Zugriff auf den Gerätespeicher. 13 Geräte wurden über ein kabelloses Netzwerk gesichert. Auf keines der Geräte mit Ubuntu Touch konnte via USB-Netzwerk zugegriffen werden. Überwiegend ließen sich unter Verwendung von USB höhere Transferraten erzielen. Auch die AuroraOS-VM konnte über SSH gesichert werden. Der Zugang über das Netzwerk erlaubt sowohl die Sicherung des internen als auch des externen Flash-Speichers. Da die Einbindepunkte des Systems im Betrieb erkennbar sind, konnte der verwendete Speicher gezielt gewählt werden. Durch die Verwendung von Ewfacquirestream konnte eine Verringerung der Sicherungsgröße um durchschnittlich ca. 89% erreicht werden. Da die Testgeräte nur wenig Nutzerdaten beinhalteten, ist im Realeinsatz je nach Speichernutzung eine weniger starke Kompression zu erwarten.

Die Sicherung der eingebundenen LUKS-Partitionen gelang und erlaubte den unverschlüsselten Einblick in die Nutzerdaten (siehe Seite 100). Hierbei wird durch Ewfacquirestream kaum eine Kompression erreicht. Die durchschnittliche Platzersparnis liegt hier nur etwas über 3%.



## 6.2 ADB-Live-Sicherung (Ubuntu Touch)

Die offizielle Dokumentation des UBports-Projekts beschreibt die Möglichkeit, per ADB (Android Device Bridge) auf die Geräteshell zuzugreifen [110]. Dazu ist im System der Entwicklermodus zu aktivieren. Der entsprechende Schalter ist unter „Systemeinstellungen“ → „Info“ (die Dokumentation benennt hier „Über“) → „Entwicklermodus“ zu erreichen. Abweichend von der Beschreibung der Dokumentation war für das Herstellen einer ADB-Verbindung über die Workstation kein Neustart der Geräte erforderlich.

Das Vorgehen ähnelt hierbei stark der Sicherung über eine SSH-Verbindung. Zunächst kann über:

```
adb shell und lsblk
```

Einsicht in die Speicherzuweisung des Geräts genommen werden. Am Gerät selbst ist dazu lediglich die USB-Verbindung über ADB zu bestätigen (siehe Anhang Seite G). Vorteilhaft ist hier, dass ein Schlüsselaustausch über ADB erfolgt und kein gesonderter Kanal dazu erforderlich ist. Der Befehlsaufbau für die ADB-Sicherung wurde wie folgt gewählt:

```
adb shell
```

Aufruf der ADB-Shell zur Übergabe des folgenden Befehls

```
"echo 753159"
```

Ausgabe des Nutzerpassworts

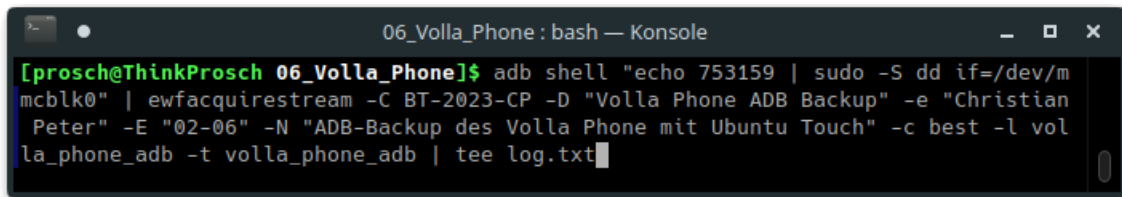
```
| sudo -S dd if=/dev/mmcblk0"
```

Pipe des ausgegebenen Nutzerpassworts an sudo. Aufruf des Sicherungsbefehls auf dem mobilen Endgerät.

Im Gegensatz zur SSH-Sicherung war eine Abfrage des Nutzerpassworts zur Erlangung von Systemverwalterrechten über stdin nicht möglich. Daher wurde das Passwort über den „echo“ Befehl übergeben.

Die Übergabe an Ewfacquirestream gleicht der SSH-Sicherung.

Die Befehlskette zur Sicherung des Volla Phone (GS290) ist in Bild 54 dargestellt:



```
06_Volla_Phone : bash — Konsole
[prosch@ThinkProsch 06_Volla_Phone]$ adb shell "echo 753159 | sudo -S dd if=/dev/m
mcblk0" | ewfacquirestream -C BT-2023-CP -D "Volla Phone ADB Backup" -e "Christian
Peter" -E "02-06" -N "ADB-Backup des Volla Phone mit Ubuntu Touch" -c best -l vol
la_phone_adb -t volla_phone_adb | tee log.txt
```

**Bild 54:** Befehlsaufruf für die ADB-Sicherung des Volla Phone (GS290)

### 6.2.1 Fazit der ADB-Live-Sicherungen

Von den sechs vorhandenen Mobilgeräten mit einer Ubuntu Touch Installation konnten vier Geräte eine ADB-Verbindung über USB herstellen. Dabei handelte es sich um die drei vom Projekt unterstützten Geräte (GS290, Nexus 5 und Aquaris E5HD) und das veraltete System des HTC One. Eine Sicherung des Gerätespeichers gelang in der Folge lediglich bei dem HTC Gerät nicht, da es zu wiederholten Verbindungsabbrüchen der USB-Verbindung kam. Eine tabellarische Übersicht der erfolgten Sicherungen ist im Anhang der Arbeit einzusehen (Seite BC). Die im Rahmen der Kompression erzielten Dateigrößen der Sicherungen sind mit denen der SSH-Sicherung vergleichbar, jedoch nicht identisch. Dies ist bei einer Live-Sicherung des laufenden Systems auch nicht zu erwarten. Die Sicherungszeiten der beiden Geräte mit Ubuntu Touch 16.04 (Aquaris E5HD und Nexus 5) liegen trotz der Verwendung von USB zur Übertragung mit einem Plus von ca. 11% leicht über denen der SSH-Sicherung.

Die Sicherungszeit des Volla Phone (GS290) verkürzte sich hingegen um 4 Stunden 27 Minuten und 12 Sekunden auf ca. 25% der Dauer der SSH-Sicherung.

Eine Einsicht in die Nutzerdaten war bei den drei erfolgten Sicherungen möglich.

## 6.3 Recovery-Sicherung

Bereits 2011 wurde in einem Artikel der Fachzeitschrift „Digital Investigation“ die Möglichkeit der Recovery-Sicherung bei bestimmten Android-Geräten dokumentiert [111 S. 14-24]. Dabei wurde demonstriert, dass eine Custom Recovery auf das zu untersuchende Mobiltelefon aufgespielt werden kann, um eine Datensicherung zu ermöglichen. Die damit einhergehenden Veränderung des Gerätespeichers wurde in Kauf genommen, da die relevante Userdata-Partition unangetastet blieb. Einige Geräte unterstützen darüber hinaus den Live-Boot eines Recovery-Images über das Fastboot-Protokoll [112]. Im forensischen Sinne ist diese Möglichkeit zu bevorzugen, da Manipulationen am Untersuchungsgegenstand auf ein Mindestmaß begrenzt werden und die verwendete Recovery für die Dauer der Sicherung im RAM vorgehalten wird. Entsprechend des erarbeiteten Vorgehensmodells (siehe Seite: 42) eignet sich die Methode vorwiegend für ausgeschaltete Geräte oder für eingeschaltete Geräte mit unbekannten Zugangsdaten, für die keine Möglichkeit zur Umgehung der Sperre gefunden wurde. Besteht die Möglichkeit der RAM-Sicherung trotz bestehender Sperre, ist diese vor einer Recovery-Sicherung durchzuführen.

Anwendbar ist diese Methode bei einer Vielzahl von Geräten, die ursprünglich mit Android vertrieben wurden und für die bereits eine Custom Recovery verfügbar ist. Eine hohe Akzeptanz in der Community genießt das Team Win Recovery Project [113], das Images für „offiziell“ vom Projekt unterstützte Geräte auf der Projektseite zum Download anbietet. Darüber hinaus existieren „inoffizielle“ Ports für weitere Geräte und Modifikationen, die sich vorwiegend in der Benutzeroberfläche unterscheiden.

Am Beispiel des Xiaomi Redmi 4X soll der Live-Recovery Boot und die anschließende Datenextraktion dargestellt werden.

Offizielle Images für dieses Gerät können von der Projektseite ([dl.twrp.me/santoni/](http://dl.twrp.me/santoni/)) bezogen werden. Dabei ist das Redmi 4X über die Suchmaske der Projektseite nicht auffindbar. Es empfiehlt sich, benötigte

Images über einschlägige Suchmaschinen zu recherchieren (Suchaufruf: <Gerätecodename> + twrp). Anschließend wird das Gerät in den Fastboot-Modus versetzt. Im Falle des Redmi 4X erfolgt dies über die Betätigung der {leiser} Taste während des Anschlusses an die Workstation über USB. Bei Erfolg wird auf dem Gerätebildschirm die Meldung „Fastboot“<sup>8</sup> ausgegeben.

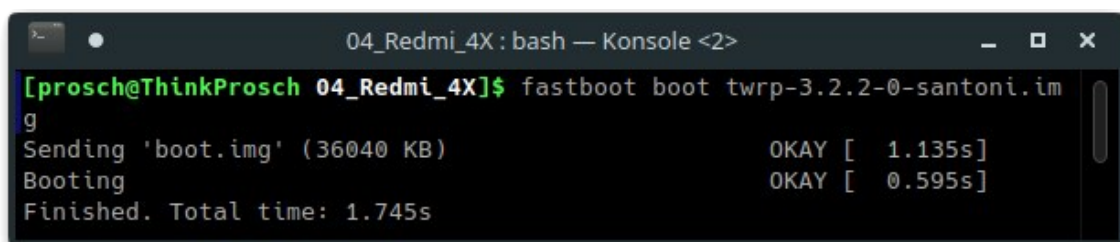
Ob eine Fastboot-Verbindung von der Workstation zum Gerät besteht, kann über den Aufruf:

```
fastboot devices
```

nachvollzogen werden. Liegt eine Verbindung vor, wird die Seriennummer des Geräts ausgegeben. TWRP kann wie folgt über den „boot“ Parameter geladen werden:

```
fastboot boot <twrp-image.img>
```

Gegebenenfalls sind diverse Versuche mit verschiedenen angebotenen TWRP-Versionen nötig, um einen Live-Boot zu erreichen. Das Redmi 4X konnte mit der Version 3.2.2-0 über den „boot“ Parameter geladen werden. Die aktuelle Version 3.6.2\_9-0 ist für den Live-Boot nicht geeignet.



**Bild 55:** Live-Boot von TWRP auf dem Redmi 4X über Fastboot

Nach dem Start von TWRP ist das Mobilgerät über ADB ansprechbar. Im Gegensatz zu der ADB-Verbindung unter Ubuntu Touch oder klassischen Android-Systemen wird dabei direkt mit Systemverwalterberechtigung gearbeitet. Klassische Linux-Befehle wie „lsblk“ sind in der TWRP-Shell nicht verfügbar. Auch der Pfad der vorhandenen Blockdevices unterscheidet sich

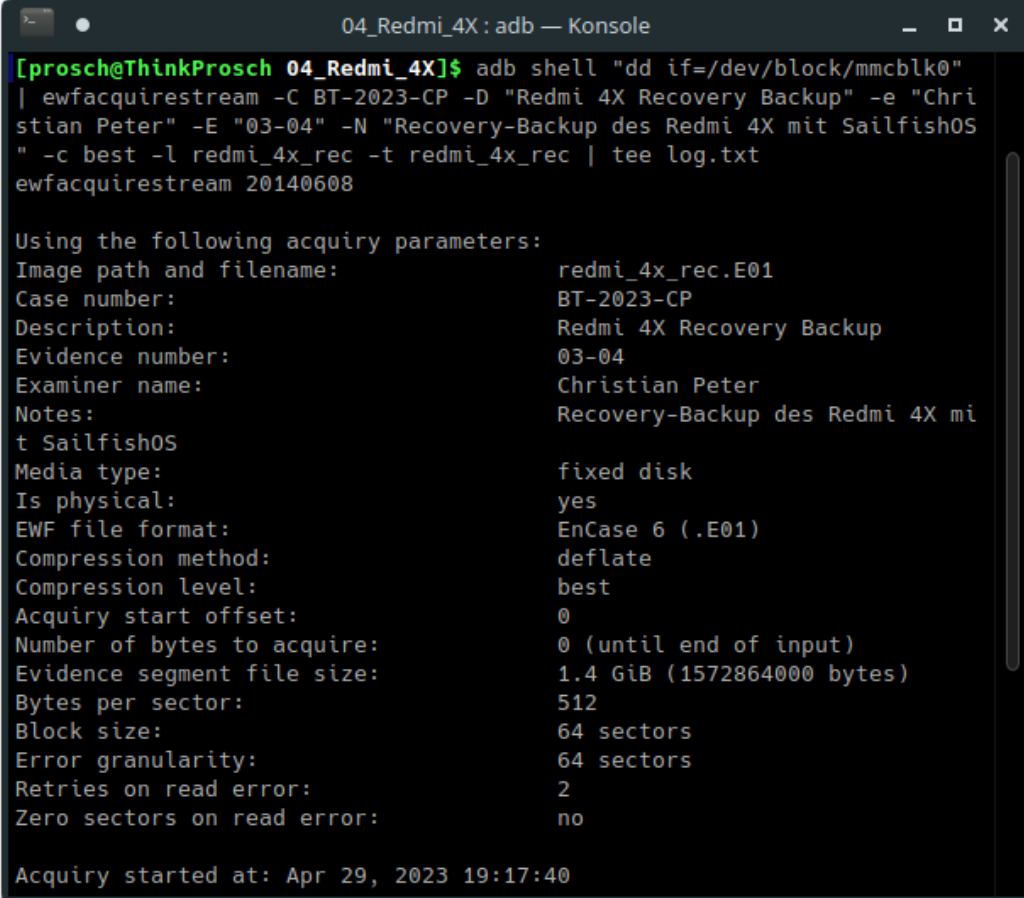
<sup>8</sup> Das Redmi 4X und einige weitere Xiaomi Geräte zeigen im Fastboot-Modus das Xiaomi-Maskottchen „Mitu“ (ein Kaninchen mit chinesischer Uschanka-Mütze), welches Reparaturen am Android-Maskottchen „Bugdroid“ durchführt.

gegenüber den bisherigen Betrachtungen. Ein Blockdevice (Emmc), welches im gestarteten System unter: `/dev/mmcblk0` angesprochen werden konnte, wird nun unter `/dev/block/mmcblk0` angeboten. Sind mehrere Speicher verfügbar, beispielsweise bei einer zusätzlich eingelegten  $\mu$ SD-Karte, kann sich die Nummerierung der Blockdevices von der des Zielsystems unterscheiden.

Um eine Übersicht über die vorhandenen Speichergeräte zu erhalten, eignet sich der folgende Befehl:

```
ls /dev/block
```

Der eigentliche Sicherungsaufwurf kann dann analog der ADB-Sicherungen (siehe Seite 61) erfolgen. Da bereits mit Systemverwalter-Rechten gearbeitet wird, ist die Übergabe eines Passworts und die Verwendung von „sudo“ hier nicht erforderlich.



```
04_Redmi_4X: adb — Konsole
[prosch@ThinkProsch 04_Redmi_4X]$ adb shell "dd if=/dev/block/mmcblk0
| ewf_acquirestream -C BT-2023-CP -D "Redmi 4X Recovery Backup" -e "Chri
stian Peter" -E "03-04" -N "Recovery-Backup des Redmi 4X mit SailfishOS
" -c best -l redmi_4x_rec -t redmi_4x_rec | tee log.txt
ewf_acquirestream 20140608

Using the following acquiry parameters:
Image path and filename:      redmi_4x_rec.E01
Case number:                  BT-2023-CP
Description:                  Redmi 4X Recovery Backup
Evidence number:              03-04
Examiner name:                Christian Peter
Notes:                        Recovery-Backup des Redmi 4X mi
t SailfishOS
Media type:                   fixed disk
Is physical:                  yes
EWF file format:              EnCase 6 (.E01)
Compression method:           deflate
Compression level:            best
Acquiry start offset:         0
Number of bytes to acquire:   0 (until end of input)
Evidence segment file size:   1.4 GiB (1572864000 bytes)
Bytes per sector:             512
Block size:                   64 sectors
Error granularity:            64 sectors
Retries on read error:        2
Zero sectors on read error:   no

Acquiry started at: Apr 29, 2023 19:17:40
```

**Bild 56:** Recovery-Sicherung des Xiaomi Redmi 4X über ADB

Der überwiegende Teil der vormaligen Android-Geräte wurde bereits im Rahmen der Installation des jeweiligen Linux-Systems mit einer Custom-Recovery ausgestattet. Abhängig vom jeweiligen Gerät lässt sich diese Recovery über den Befehl:

```
fastboot reboot recovery
```

oder über eine Tastenkombination (meist **{Lauter}** und **{Power}**) aufrufen. Einige Geräte (z.B. das OnePlus 3T) bieten auch die Möglichkeit, die Recovery über einen Auswahlbildschirm im Fastboot Modus zu starten. Dazu werden die Lautstärketasten für die Wahl des gewünschten Modus verwendet. Informationen zum Aufruf der Recovery für das jeweilige Gerät finden sich im Anhang der Arbeit (Seite A ff.). Ubuntu Touch Geräte, die von Ubports als unterstützt gelistet werden, sind nach der Installation über den Ubports Installer mit der Ubports-Recovery ausgestattet. Da auch diese Recovery eine Systemverwalter-Shell über ADB anbietet, ergeben sich hierbei für die Sicherung keine Unterschiede. Dies trifft auch auf das vorinstallierte System des Volla Phones zu. Zwar wird das Gerät mit einem gesperrten Bootloader ausgeliefert; die vorhandene Ubports Recovery erlaubt aber auch hier den privilegierten Zugriff auf den Gerätespeicher.

### 6.3.1 Geräte mit A/B-Partitionen

Viele der aktuellen Android-Geräte verfügen über keine dedizierte Recovery-Partition. Dies begründet sich in einem Wandel des Update-Systems seitens Google. Bereits mit Android 7.0 führte Google das Feature „Seamless System Updates“ ein [114], stellte den Herstellern von Mobilgeräten die Nutzung jedoch zunächst frei. Erst für Geräte, die mit Android 13 ausgeliefert werden, wird dieses Feature als zwingend angesehen, um eine „Google Mobile Services (GMS)“ Zertifizierung zu erhalten [115]. Eine Ausnahme stellen aktuell Mobilgeräte von Samsung dar, welche auch 2023 trotz werksseitiger Android 13

Installation keine Seamless Updates unterstützen [116]. Zum Konzept dieser Update-Strategie gehört, dass Systempartitionen doppelt auf dem Gerät vorhanden sind und über die Slots A und B angesprochen werden. Im Falle eines fehlerhaften Updates verbleibt eine funktionierende Boot-Umgebung auf dem Gerät und kann vom jeweils anderen Slot geladen werden. Eine Recovery-Partition zur Installation von Updates ist hierbei nicht mehr erforderlich. Ungeachtet dessen wird von einigen A/B-Geräten auch der Live-Boot einer Custom Recovery unterstützt. So handelt es sich bei den Testgeräten Xiaomi Mi A2 Lite und Sony Xperia XA2 um A/B-Geräte, welche den Live-Boot einer Recovery erlauben.

Bei dem vorliegenden Sony Xperia 5 (siehe Seite 31) lag ebenfalls ein A/B-Partitionsschema vor. Der Live-Boot einer Recovery wird von diesem Gerät nicht unterstützt. Ein offizielles TWRP-Image ist für das Xperia 5 ebenfalls nicht verfügbar. Es konnte jedoch eine vorkompilierte Version der „Lineage Recovery“ über einen XDA-Foreneintrag [117] bezogen werden. Das bezogene Archiv beinhaltete die drei Imagedateien: boot.img, dtbo.img und vbmeta.img. Das Flashen dieser Image-Dateien über Fastboot ohne Beachtung der Boot-Slots führt zwangsläufig dazu, dass die Droidian-Installation des Gerätes nicht mehr geladen wird. Es kann jedoch der Umstand genutzt werden, dass sich das Boot-Slot-System über Fastboot ansprechen lässt. Im Falle des Sony Xperia 5 erfolgt der Aufruf von Fastboot unter Betätigung der **{lauter}** Taste im ausgeschalteten Zustand während des Anschlusses über USB an die Workstation. Die **{lauter}** Taste ist dabei so lange gedrückt zu halten, bis die Farbe der Status-LED des Geräts von rot auf blau wechselt.

Der Befehl:

```
fastboot getvar current-slot
```

gibt an, welcher Slot aktuell genutzt wird. Folgend kann über:

```
fastboot --set-active=a      bzw:      fastboot --set-active=b
```

der jeweils andere Bootslot gewählt werden. Bei dem vorliegenden Xperia 5 war zunächst Slot B aktiv, so dass auf Slot A gewechselt wurde (siehe Bild 57).

```

15_Xperia_V: bash — Konsole
[prosch@ThinkProsch 15_Xperia_V]$ fastboot getvar current-slot
current-slot: b
Finished. Total time: 0.003s
[prosch@ThinkProsch 15_Xperia_V]$ fastboot --set-active=a
Setting current slot to 'a' OKAY [ 0.033s]
Finished. Total time: 0.036s

```

Bild 57: Anzeige und Auswahl des Bootslots auf dem Xperia 5

Die Installation der Image-Dateien erfolgt im Anschluss ebenfalls über Fastboot:

```

15_Xperia_V: bash — Konsole <2>
[prosch@ThinkProsch 15_Xperia_V]$ fastboot flash boot boot.img
Sending 'boot_a' (65536 KB) OKAY [ 1.948s]
Writing 'boot_a' OKAY [ 0.255s]
Finished. Total time: 2.222s
[prosch@ThinkProsch 15_Xperia_V]$ fastboot flash dtbo dtbo.img
Sending 'dtbo' (8192 KB) OKAY [ 0.241s]
Writing 'dtbo' OKAY [ 0.042s]
Finished. Total time: 0.294s
[prosch@ThinkProsch 15_Xperia_V]$ fastboot --disable-verity --disable-ver
ification flash vbmeta vbmeta.img
Rewriting vbmeta struct at offset: 0
Sending 'vbmeta' (4 KB) OKAY [ 0.005s]
Writing 'vbmeta' OKAY [ 0.003s]
Finished. Total time: 0.019s

```

Bild 58: Installation der Lineage Recovery in Bootslot A

Die Parameter „--disable-verity“ und „--disable-verification“ werden dabei benötigt, um den Integritätscheck des Android-Kernels zu deaktivieren, welcher den Start einer Custom Recovery verhindern würde.

Die installierte Recovery kann nun über die Tastenkombination **{leiser}** und **{Power}** aufgerufen und die Sicherung nach dem beschriebenen Prinzip (siehe Seite 65) durchgeführt werden. Da es sich um ein Gerät mit UFS-Speicher<sup>9</sup> handelt, lautet der Pfad des zu sichernden Block-Devices hier: `/dev/block/sda`. Es werden auch die Pfade: `sdb` bis `sdf` angeboten; `sda` nimmt jedoch den Großteil des verfügbaren Speicherbereichs ein und beinhaltet die relevante Userdata-Partition.

<sup>9</sup> Im Gegensatz zum halb-duplex eMMC-Speicher verwendet UFS eine voll-duplex Schnittstelle für simultane Lese- und Schreibzugriffe [118]. Da UFS-Speicher teurer ist, wird in günstigeren Smartphones oft noch eMMC-Speicher eingesetzt.



Nach der erfolgreichen Datensicherung wird der ursprüngliche Bootslot wieder ausgewählt:

```
fastboot --set-active=b
```

und die Startfähigkeit des Systems wieder hergestellt. Die Recovery verbleibt hier zunächst auf dem Gerät, wird jedoch im Rahmen eines künftigen Systemupdates automatisch überschrieben.

### 6.3.2 Samsung Geräte

Abweichend von der üblichen Bereitstellung des Fastboot-Modus wird auf Samsung Mobilgeräten der Download-Modus implementiert [119]. Ein Windows-Tool zur Kommunikation mit Geräten im Download-Modus ist Odin [120], weshalb auch die Bezeichnung: „Odin-Modus“ geläufig ist. Für Linux steht eine quelloffene Software-Alternative namens Heimdall zur Verfügung, die mit dem verwendeten Protokoll „Loke“ kommunizieren kann.

Die Möglichkeit des Live-Boots einer Recovery besteht im Download-Modus nicht. Es wird jedoch eine vom Betriebssystem unabhängige Recovery-Partition bereitgestellt. Auch TWRP bietet offizielle Images für zahlreiche Samsung-Geräte an.

Das Vorgehen soll anhand des vorliegenden Samsung Galaxy S5 demonstriert werden:

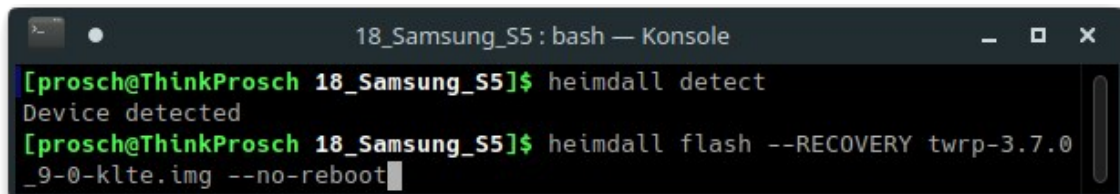
Das offizielle TWRP-Image wird auf der Projektseite ([dl.twrp.me/klte/](http://dl.twrp.me/klte/)) zum Download angeboten. Auch in der Gerätedatenbank ist das Galaxy S5 vertreten. Der Download-Mode wird bei diesem Gerät im ausgeschalteten Zustand über die Tastenkombination: **{leiser}**, **{Home}** und **{Power}** erzwungen. Die folgende Warnmeldung ist mit der **{lauter}** Taste zu bestätigen. Der Befehl:

```
heimdall detect
```

gibt Auskunft, ob ein Samsung-Gerät im Download Modus verbunden ist. TWRP lässt sich wie folgt über den Befehl:

```
heimdall flash --RECOVERY <twrp-image.img> --no-reboot
```

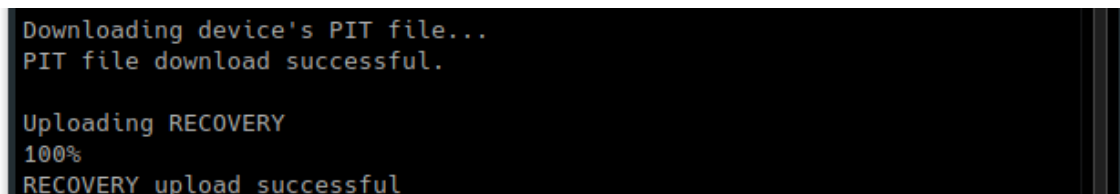
installieren. Bild 59 zeigt den Ablauf der Installation:



```
18_Samsung_S5 : bash — Konsole
[prosch@ThinkProsch 18_Samsung_S5]$ heimdall detect
Device detected
[prosch@ThinkProsch 18_Samsung_S5]$ heimdall flash --RECOVERY twrp-3.7.0
_9-0-klte.img --no-reboot
```

**Bild 59:** Flashen von TWRP auf dem Samsung Galaxy S5

Wurde ein passendes Image geladen, wird kurz darauf der erfolgreiche Upload der Recovery vermeldet:



```
Downloading device's PIT file...
PIT file download successful.

Uploading RECOVERY
100%
RECOVERY upload successful
```

**Bild 60:** Galaxy S5 - TWRP wurde erfolgreich geflasht

Nach der Installation kann TWRP mittels der Tastenkombination **{lauter}**, **{Home}** und **{Power}** aufgerufen werden. Die eigentliche Sicherung des Gerätespeichers unterscheidet sich in der Folge nicht von der anderer Android-Geräte (siehe Seite 65).

### 6.3.3 Fazit der Recovery-Sicherungen

Für die 20 Testgeräte, die zuvor mit Android betrieben wurden, konnten Custom-Recovery-Images gefunden werden. Elf dieser Geräte wurden bereits im Rahmen der Betriebssystem-Installation mit einer Custom-Recovery bespielt. Auf drei Geräten wurde im Rahmen der Sicherung eine geeignete

Recovery installiert. Dabei handelt es sich um die Installation der Lineage Recovery auf dem A/B-Gerät Sony Xperia 5 im ungenutzten Bootslot und die Samsung-Geräte Galaxy S III und Galaxy S5. Vier Geräte (Xiaomi Redmi 4X, Xiaomi Mi A2 Lite, Sony Xperia XA2 und Lenovo A6000) ließen den Live-Boot einer Custom Recovery zu. Hierbei konnte keine ADB-Verbindung zu der TWRP-Recovery auf dem Lenovo A6000 hergestellt werden. Versuche, eine Recovery auf weiteren Geräten live zu booten, welche bereits installationsseitig eine Custom-Recovery vorweisen, wurden nicht unternommen. Eine TWRP-Recovery für das Motorola Droid 4 konnte nicht gefunden werden. Die verfügbare „Safestrap“ Recovery [121] verlangt die Installation aus einem laufenden Android-System heraus und wurde entsprechend nicht betrachtet. Insgesamt wurden Speicherabbilder von 18 Geräten erzeugt.

Die detaillierte Übersicht der Recovery-Sicherungen ist im Anhang der Arbeit (Seite BC) einzusehen. Einen Auszug zeigt das folgend dargestellte Schema:

**Tabelle 7:** Recovery-Sicherungen (Auszug)

Gerät	System	Installation	Recovery	Speicher	Backup	Dauer
Redmi 4X	SailfishOS	live	TWRP	29 GiB	1,5 GiB	19min 44s
Xperia 5	Droidian	Installiert	Lineage	119 GiB	12,2 GiB	1h 4min 27s

Auch hierbei ist ersichtlich, dass die Größe der Speicherabbilder mit denen der vorangegangenen Sicherungen vergleichbar ist. Im Vergleich mit den TCP-Live-Sicherungen werden überwiegend kürzere Sicherungszeiten erreicht. Von 16 Geräten, die über beide Methoden gesichert werden konnten, unterschreiten 12 die Sicherungszeiten der TCP-Live-Sicherungen. Vier Geräte benötigen mehr Zeit für die Sicherung. Gegenüber den ADB-Live-Sicherungen werden bei zwei von drei vergleichbaren Geräten kürzere Sicherungszeiten erreicht. Die Sicherungszeit des Gigaset GS290 ist hingegen bei der Recovery-Sicherung leicht erhöht.

Ein Einblick in das Dateisystem ist bei den erzeugten Systemabbildern möglich. Die Nutzerdaten der Geräte Sony Xperia XA2 und Xperia 5 liegen hingegen verschlüsselt vor und sind vor der weiteren Betrachtung zu entschlüsseln (siehe: 8 Entschlüsselung der Sicherung – Seite 103).

## 6.4 Bootloader-Sicherung

Die Sicherung über ein Recovery-Image konnte zum Teil durchgeführt werden, ohne Veränderungen am Flash-Speicher des Geräts zu verursachen. In einigen Fällen war jedoch auch die Installation der Recovery auf dem Gerät erforderlich. Hierdurch wurden die relevanten Nutzerdaten nicht verändert; es findet aber eine Veränderung von Daten auf dem Gerät statt. Im Leitfaden IT-Forensik des BSI wird festgehalten, „dass idealerweise [...] keine Daten durch [...] Sicherungsmaßnahmen verfälscht werden.“ [64 S. 25]

Einige Geräte bieten die Möglichkeit, über eine Kommunikation mit dem Bootloader Daten zu extrahieren, ohne permanente Veränderungen am Speicherinhalt des Geräts zu verursachen. Bei Mobilgeräten mit ARM-SoC sind diese Möglichkeiten abhängig vom Chipsatz-Hersteller und der Implementierung verfügbarer Bootmodi durch die Hersteller der Mobilgeräte. Die recherchierten Möglichkeiten der Bootloader-Sicherung für die vorliegenden Testgeräte sollen nachfolgend vorgestellt werden.

### 6.4.1 EDL-Modus (Qualcomm SoCs)

In einem Blogpost der Aleph Security Gruppe [122] wird die Möglichkeit erläutert, über den Emergency Download Mode (EDL) Zugriff auf den Bootloader von Mobilgeräten mit Qualcomm-SoC zu erhalten. Dabei wird vorausgesetzt, dass für das jeweilige Gerät ein passender und vom Hersteller signierter „Programmer“ oder „Firehose-Loader“<sup>10</sup> vorliegt, welcher vom „Sahara“-Protokoll des Geräts überprüft und geladen wird. Nach dem Laden des Programmers können Befehle entsprechend der Fähigkeiten des Loaders an das Gerät gesendet werden. Für forensische Zwecke kann dazu das Tool „Qualcomm Sahara / Firehose Attack Client / Diag Tools“ (folgend: EDL Client)

---

<sup>10</sup> Die Herkunft der aktuell verfügbaren Loader ist nicht bekannt. Vermutlich wurden sie durch Reparaturdienstleister veröffentlicht.

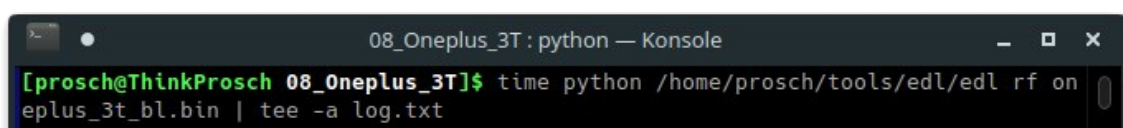
[95] genutzt werden. Dieses Tool erlaubt das automatisierte Uploaden eines Firehose-Loaders an ein Gerät im EDL-Modus und das Ausführen von Sicherungsbefehlen über entsprechende Payloads. Eine umfangreiche Sammlung verfügbarer Loader ist in einem Submodul des Tools bereits enthalten. Darüber hinaus können selbst gefundene Loader über ein Skript in die Sammlung des Tools übertragen werden. Voraussetzung für die Kommunikation mit dem Bootloader ist zunächst der Aufruf des EDL-Modus. Über USB identifiziert sich ein Gerät im EDL-Modus anschließend als „Qualcomm HS-USB 9008“ oder „QDLoader HS-USB“.

Der Aufruf des EDL-Modus unterscheidet sich je nach Gerät. Einige Geräte lassen sich über die Tastenkombination `{leiser}` und `{lauter}` in den EDL-Modus starten, andere erfordern das Kurzschließen definierter „Test Points“ [123]. Xiaomi Geräte gestatten zum Teil auch den Start in den EDL-Modus aus dem Fastboot-Modus heraus. Ein entsprechendes Boot-Skript (`fastpwn`) ist in der Repository des EDL Clients enthalten.

Bei den vorliegenden Testgeräten konnten verschiedene Möglichkeiten des EDL-Boots beobachtet werden. Die genannte Tastenkombination während des Anschlusses an die Workstation über USB ließ sich z.B. für das Oneplus 3T anwenden. Am Gerät selbst kann dabei keine Reaktion beobachtet werden. Der anschließende Aufruf des Sicherungstools folgt dabei der Syntax:

```
python edl rf <image.bin>
```

Der Parameter „rf“ wurde gewählt, um den gesamten Flash-Speicher in eine Datei zu sichern. Bei den durchgeführten Sicherungen wurde zudem über „time“ die Laufzeit des Tools während der Sicherung erfasst und die Ausgabe über „tee“ in eine Datei geschrieben:



```
08_Oneplus_3T: python — Konsole
[prosch@ThinkProsch 08_Oneplus_3T]$ time python /home/prosch/tools/edl/edl rf on
eplus_3t_bl.bin | tee -a log.txt
```

**Bild 61:** Sicherungsaufbau des EDL Clients bei dem Oneplus 3T

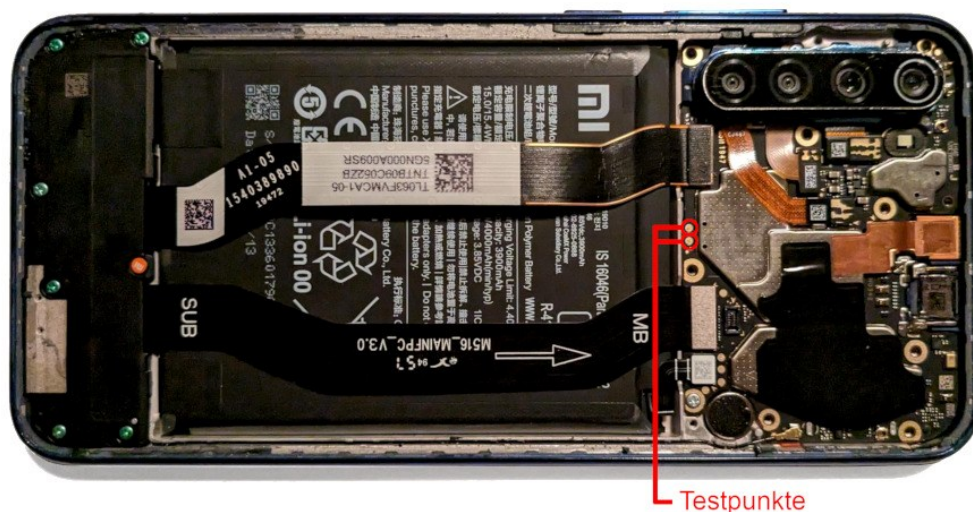
Das Tool lässt sich wahlweise nach dem Anschluss des Gerätes oder bereits davor aufrufen. In diesem Fall wartet das Programm auf die Verbindung eines Geräts und gibt Hinweise zum Triggern des EDL-Modus. Die erfolgreiche Geräteverbindung wird anschließend über das Terminal ausgegeben:

```
....main - Device detected :)
sahara - Protocol version: 2, Version supported: 1
main - Mode detected: sahara
sahara -
-----
HWID:          0x0005f0e12a703db9 (MSM_ID:0x0005f0e1,OEM_ID:0x2a70,MODEL_ID:
0x3db9)
CPU detected:   "MSM8996Pro"
PK_HASH:       0xc0c66e278fe81226585252b851370eabf8d4192f0f335576c3028190d49
d14d4
Serial:        0x210f6164
```

**Bild 62:** Erkennen des sahara-Modus am Oneplus 3T über den EDL Client

Ein passender Loader wurde bereits über die Sammlung des Tools geladen. Nach einer kurzen Initialisierungsphase startete die Sicherung des Speichers.

Das Testgerät Xiaomi Redmi Note 8T wechselt bei Verwendung der Tastenkombination **{leiser}** und **{lauter}** nicht in den EDL-Modus. Stattdessen ist hier eine Verbindung zweier Testpunkte auf der Hauptplatine des Mobiltelefons herzustellen, während das Gerät per USB mit der Workstation verbunden wird. Die Position der Testpunkte konnte im Rahmen einer Internetrecherche ermittelt werden [124]. Das verklebte Backcover war zuvor zu entfernen, um Zugriff auf die Hauptplatine zu erhalten.



**Bild 63:** EDL-Testpunkte des Redmi Note 8T

Über das Brücken der beiden Pads während des Anschlusses an die Workstation ließ sich anschließend eine EDL-Verbindung herstellen:

```

.....main - Device detected :)
sahara - Protocol version: 2, Version supported: 1
main - Mode detected: sahara
sahara -
-----
HWID:           0x0010a0e100720000 (MSM_ID:0x0010a0e1,OEM_ID:0x0072,MODEL_ID:0x0000)
CPU detected:   "nicobar"
PK_HASH:       0x1bebe3863a6781db4b01086063007334de9e5ca14971c7c4f4358ec9d79cda4692ce
5e948c6fd409408f4c919fcadfe3
Serial:        0xc30bb166

```

**Bild 64:** Erkennen des sahara-Modus am Redmi Note 8T über den EDL Client

Auch bei diesem Gerät wurde über den EDL-Client eine Sicherung des internen Flash-Speichers erzeugt.

Der Fastboot-Aufruf des EDL-Modus über das „fastpwn“-Bootskript gelang bei dem Pocophone F1. Hier erfolgte jedoch keine Sicherung, da die Authentifizierung gegenüber dem Bootloader scheiterte.

Unterschiede in der Ausgabe des EDL Clients ergeben sich aus der Verwendung von eMMC- oder UFS-Speicher. Liegt ein eMMC-Chip vor, erfolgt der Dump über den gesamten Speicherinhalt in eine Datei. Bei UFS-Speicherchips wird der Speicher in mehrere „LUN“s (Logical Unit Number) aufgeteilt. Die LUN Nr. 0 entspricht dem Blockdevice „sda“ der vorangegangenen Sicherungen und beinhaltet die Userdata-Partition. Erzeugte Speicherabbilder können nach der Sicherung über ewfacquire komprimiert und segmentiert werden. So wurde aus den 53 GiB der LUN0 des Oneplus 3T ein EWF-Backup von 5,6 GiB erzeugt.

```

08_Oneplus_3T: bash — Konsole
[prosch@ThinkProsch 08_Oneplus_3T]$ ewfacquire -C BT-2023-CP -D "Oneplus 3T Boot
loader Backup" -e "Christian Peter" -E "06-08" -N "Bootloader-Backup des Oneplus
3T mit Droidian (EDL)" -c best -l oneplus_3t_bl -t oneplus_3t_bl -u oneplus_3t_
bl.bin.lun0 | tee -a log.txt

```

**Bild 65:** Komprimieren der EDL-Sicherung des Oneplus 3T über ewfacquire

Der Erfolg dieser Methode ist erheblich davon abhängig, ob es bekannte Ansätze zum Aufruf des EDL-Modus gibt. Für die überwiegende Zahl der Qualcomm-Geräte wurden geeignete Ansätze bisher nicht bekannt.

## 6.4.2 MTK-BROM-Modus (MediaTek SoCs)

MediaTek integrierte ebenfalls ein Bootloader-Protokoll zur Reparatur und Fehlerdiagnose in seinen SoCs. Bereits 2015 wurde der forensische Nutzen des BROM-Modus<sup>11</sup> (Boot ROM Modus) in einem Artikel des Journal of Digital Forensics dargestellt [125]. Auch aktuelle MediaTek Geräte sind über diesen Modus ansprechbar. Dabei wird ein Chip-abhängiger Download Agent (DA) in den RAM des Gerät geladen oder über einen Payload die Authentifizierung des DA deaktiviert, um einen universellen DA an das Gerät zu senden [126]. Im Rahmen dieser Ausarbeitung wird das Tool „MTKClient“ [93] verwendet, welches bereits eine umfangreiche Sammlung an Preloader-Dateien beinhaltet.

Im Gegensatz zum Aufruf des EDL-Modus bei Qualcomm SoCs ist bei MTK Geräten entscheidend, dass bereits ein geeignetes Programm auf der forensischen Workstation läuft und auf den Anschluss eines Gerätes im BROM- oder Preloader-Modus wartet.

Die übliche Methode für den Aufruf des BROM-Modus ist die Tastenkombination: {leiser} und {lauter} während das ausgeschaltete Gerät per USB mit der Workstation verbunden wird. Einige Geräte benötigen jedoch den Masseschluss eines Testpunkts oder die Brückung zweier Testpunkte, um den BROM Modus aufzurufen. Die drei vorliegenden Testgeräte mit MediaTek Chipsatz konnten jeweils über die genannte Tastenkombination in den BROM-Modus gestartet werden. Der Programmaufruf ähnelt hierbei dem des EDL Clients:

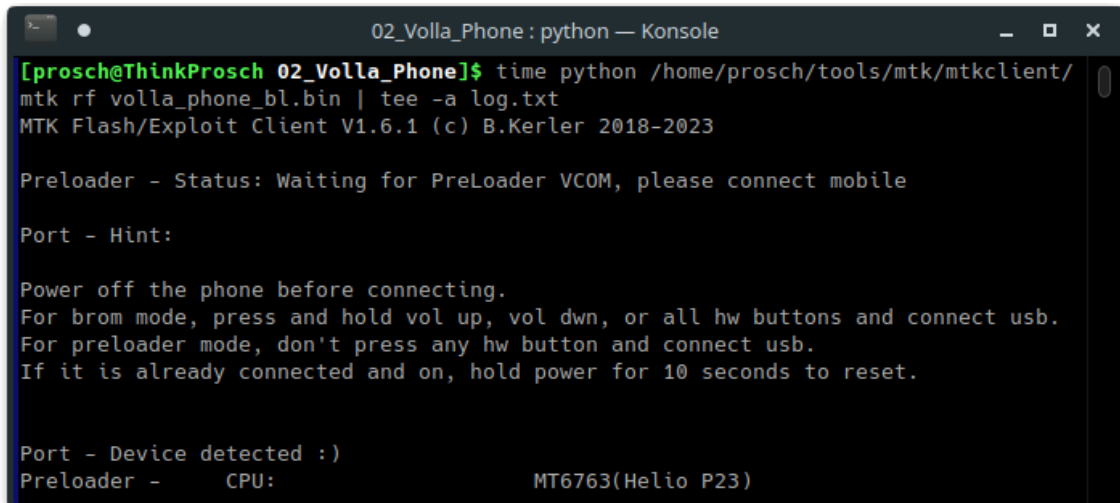
```
python mtk rf <image.bin>
```

Wie schon bei den Qualcomm-Sicherungen bewirkt der „rf“ Parameter die Sicherung des kompletten Speicherinhalts. Auch hier wurde der „time“ Operator vorangesetzt, um die Ausführungszeit des Programms zu ermitteln. Der Pipe-Aufruf von „tee“ schreibt die Konsolenausgabe in eine Textdatei.

<sup>11</sup> Der Begriff „BROM-Mode“ wird nicht exklusiv von MediaTek verwendet. Auch Geräte mit Unisoc/Spreadtrum-, Allwinner- und Texas Instruments-SoC verwenden diese Bezeichnung.



Bild 66 zeigt den Aufruf des Tools für das Volla Phone / GS290. Nach einer kurzen Initialisierungsphase erfolgt die Sicherung des gesamten Speichers in eine Datei im unkomprimierten RAW-Format.



```

[prosch@ThinkProsch 02_Volla_Phone]$ time python /home/prosch/tools/mtk/mtkclient/
mtk rf volla_phone_bl.bin | tee -a log.txt
MTK Flash/Exploit Client V1.6.1 (c) B.Kerler 2018-2023

Preloader - Status: Waiting for PreLoader VCOM, please connect mobile

Port - Hint:

Power off the phone before connecting.
For brom mode, press and hold vol up, vol dwn, or all hw buttons and connect usb.
For preloader mode, don't press any hw button and connect usb.
If it is already connected and on, hold power for 10 seconds to reset.

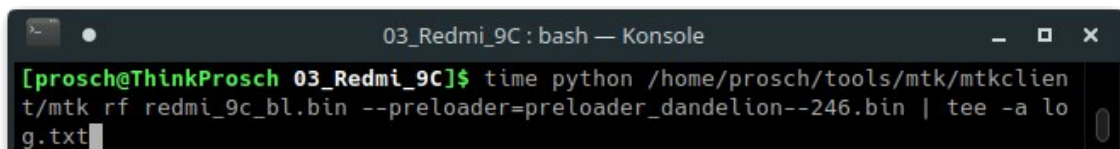
Port - Device detected :)
Preloader - CPU: MT6763(Helio P23)

```

**Bild 66:** Sicherung des Volla Phones / GS290 über den MTKClient

Dieser Speicher-Dump kann über ewacquire in das EWF-Format überführt werden, wodurch eine erhebliche Speicherersparnis ermöglicht wird. So wurde aus dem 58 GiB RAW-Dump eine 2,4 GiB große EWF-Sicherung erzeugt.

Das Gerät Xiaomi Redmi 9C wechselte ebenfalls zuverlässig über die genannte Tastenkombination in den BROM-Modus. Eine Sicherung über den universellen Preloader scheiterte jedoch mit der Fehlermeldung: „Error on sending emi“<sup>12</sup>. Unter Verwendung des Preloaders des nahezu baugleichen Redmi 9A (Codename: Dandelion), welches über das XDA-Forum [127] bezogen werden konnte, gelang auch die Sicherung des Flash-Speichers bei diesem Modell.



```

[prosch@ThinkProsch 03_Redmi_9C]$ time python /home/prosch/tools/mtk/mtkclient/
mtk rf redmi_9c_bl.bin --preloader=preloader_dandelion--246.bin | tee -a log.txt

```

**Bild 67:** Verwenden eines geeigneten Preloaders bei der Ausführung von MTKClient

<sup>12</sup> EMI ist die Abkürzung für External Memory Interface. Dieses ist verantwortlich für die Kommunikation zwischen einem Schaltkreis und einem externen Speichermedium (z.B. RAM)

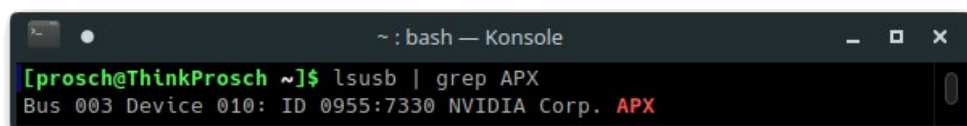
### 6.4.3 APX-Modus (Nvidia Tegra SoCs)

Der APX-Modus<sup>13</sup> [128] oder RCM-Modus (Nvidia Recovery Modus) ist im Boot-ROM von Nvidia Tegra SoCs integriert und erlaubt eine low-level Kommunikation mit diesen Geräten. Über einen Exploit namens „Fusée Gelée“ kann der USB Software Stack des APX-Modus genutzt werden, um Kontrolle über den Boot and Power Management Processor (BPMP) zu erlangen [129]. Über diesen Angriff kann eigener, unsignierter Code auf dem Gerät ausgeführt werden. Eine größere Bekanntheit erlangte der Hack über die Videospielkonsole Nintendo Switch, die diese Schwachstelle ebenfalls aufweist [130].

Das Vorgehen bei der Ausnutzung dieser Schwachstelle für die forensische Sicherung der vorliegenden Geräte mit Nvidia SoC unterscheidet sich bei der Verwendung von Software und Payloads. Daher wird folgend die Sicherung beider Geräte erläutert.

#### ASUS Transformer Pad TF300T:

Diverse Möglichkeiten für den Aufruf des APX-Modus werden in einem Foreneintrag im XDA-Forum benannt [131]. Als zuverlässige Methode bei dem vorliegenden Gerät erwies sich das Halten der **{lauter}** Taste während des Anschlusses des ausgeschalteten Geräts an die Workstation über USB. Bei Erfolg wird ein neues Gerät im APX-Modus an der Workstation erkannt:



**Bild 68:** Identifizierung des TF300T im APX-Modus

Die ausgegebene Device-ID (Hier: 7330) ist für den folgenden Angriff als Parameter relevant. Eine geeignete Version des Nvidia Tools „nvflash“, welches

<sup>13</sup> Es konnte keine offizielle Dokumentation über die Bedeutung von „APX“ gefunden werden. Ob es sich hierbei um eine Abkürzung handelt, ist nicht bekannt.

für die Kommunikation mit Geräten im APX-Modus genutzt werden kann, wurde durch das Team „AndroidRoot.mobi“ bereitgestellt. Zum Zeitpunkt der Anfertigung der Arbeit war diese Domain nicht mehr erreichbar. Eine Kopie der Binaries und der Dokumentation des Projekts konnte auf dem Portal „readynests.com“ [132] vorgefunden werden. Die ursprüngliche Dokumentation sah die Erzeugung notwendiger Bootloader-Blobs über ein spezielles Recovery-Image vor. Diese Blobs enthalten den Secure Boot Key (SBK), welcher für jedes Gerät individuell ist und bei dem Aufruf von nvflash die Ausführung weiterer Anweisungen legitimiert. Da das Aufspielen einer Recovery eine potentiell vermeidbare Veränderung am Gerätespeicher verursacht und unvorhersehbare Umstände den Aufruf von Fastboot oder den Gerätestart verhindern können, wurde eine alternative Methode zur Generierung der Blobs gewählt. Dieses Vorgehen wird in einem weiteren XDA-Foreneintrag erläutert [133].

Zunächst waren dazu die geforkten „nvcrypttools“ des Github-Users GeorgMato4 in der Branch „forN7“<sup>14</sup> [134] mitsamt enthaltener Submodule zu beziehen und die darin enthaltene Binary „mknvfblob“ unter Verwendung der Umgebungsvariable „export CROSS\_COMPILE=“ zu kompilieren.

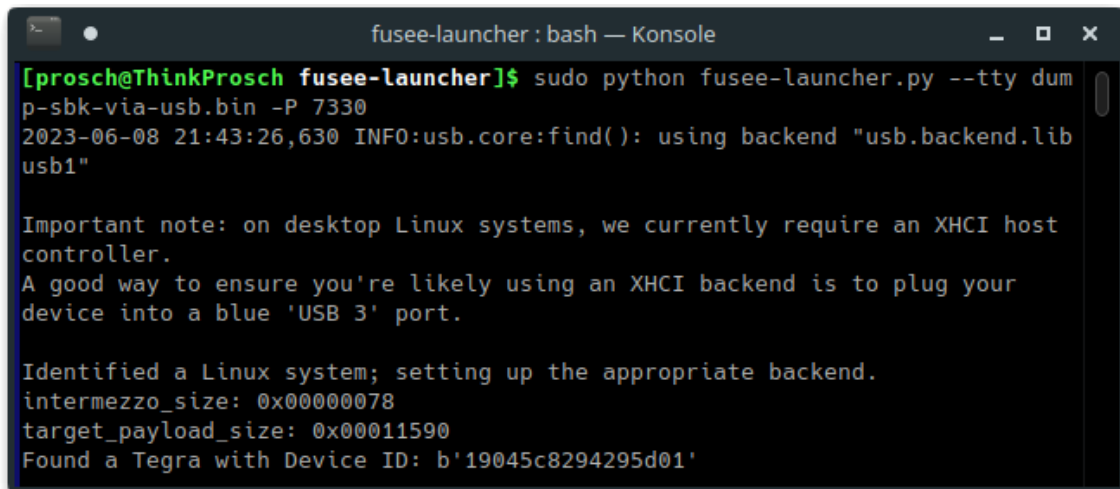
Bereits im Download enthalten waren entschlüsselte Bootloader-Dateien ohne SBK und Boot Configuration Tables (BCT) für verschiedene geeignete Tegra3 Geräte. Der für die Erstellung des Blob-Files erforderliche SBK konnte unter Ausnutzung des Fusée Gelée Exploits ermittelt werden. Es wurde dabei die Repository „tegra30\_debrick“ des Github-Users „tofurky“ verwendet. Diese wurde zunächst inklusive der enthaltenen Submodule per Git bezogen:

```
git clone --recurse-submodules https://github.com/tofurky/tegra30_debrick/
```

Im enthaltenen Verzeichnis „fusee-launcher“ waren im Anschluss die Payload-Dateien über eine geeignete Toolchain (gcc-arm-none-eabi 12.2) zu kompilieren. Zur Extraktion des SBK konnte nun der Payload „dump-sbk-via-

14 N7 bezeichnet das Google Nexus 7 Tablet. Dieses verwendet ebenfalls den Tegra3 SoC. Geeignete Binaries für den Angriff des Transformer Pads waren in dieser Branch der Repository ebenfalls enthalten.

usb.bin“ über den Fusée Gelée Exploit übermittelt werden. Die zuvor ermittelte Device-ID wurde ebenfalls für den Aufruf verwendet (siehe Bild 69).



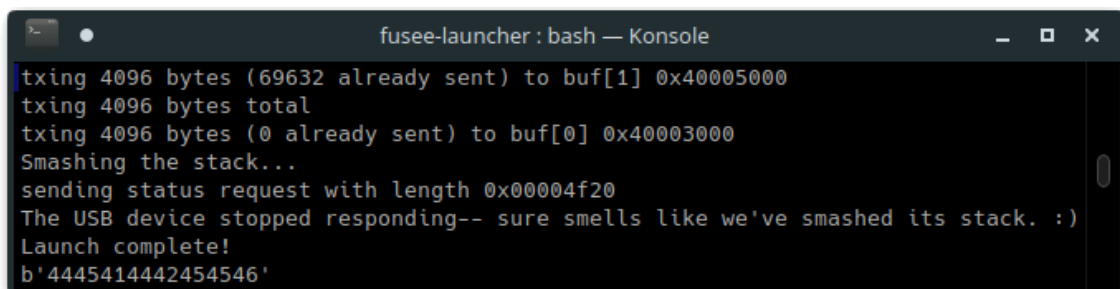
```
fusee-launcher : bash — Konsole
[prosch@ThinkProsch fusee-launcher]$ sudo python fusee-launcher.py --tty dum
p-sbk-via-usb.bin -P 7330
2023-06-08 21:43:26,630 INFO:usb.core:find(): using backend "usb.backend.lib
usb1"

Important note: on desktop Linux systems, we currently require an XHCI host
controller.
A good way to ensure you're likely using an XHCI backend is to plug your
device into a blue 'USB 3' port.

Identified a Linux system; setting up the appropriate backend.
intermezzo_size: 0x00000078
target_payload_size: 0x00011590
Found a Tegra with Device ID: b'19045c8294295d01'
```

**Bild 69:** Senden des Fusée-Payloads an das Transformer Pad TF300T

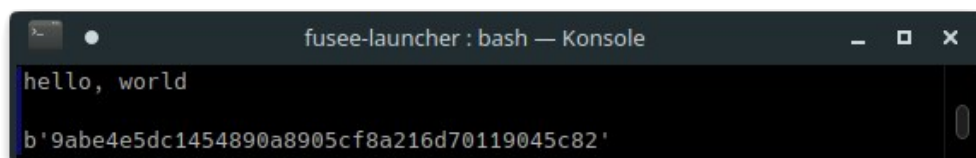
Über den Aufruf wird ein Buffer Overflow herbeigeführt, welcher die Ausführung des Payloads ermöglicht:



```
fusee-launcher : bash — Konsole
txing 4096 bytes (69632 already sent) to buf[1] 0x40005000
txing 4096 bytes total
txing 4096 bytes (0 already sent) to buf[0] 0x40003000
Smashing the stack...
sending status request with length 0x00004f20
The USB device stopped responding-- sure smells like we've smashed its stack. :)
Launch complete!
b'4445414442454546'
```

**Bild 70:** Erfolgreicher Buffer Overflow Angriff auf den Bootloader des TF300T

Der Payload vermeldet schließlich „hello, world“ gefolgt von der Ausgabe des gerätespezifischen SBK (siehe Bild 71).



```
fusee-launcher : bash — Konsole
hello, world
b'9abe4e5dc1454890a8905cf8a216d70119045c82'
```

**Bild 71:** Ausgabe des SBK des Transformer Pads TF300T

Gemäß der Beschreibung des Projekts besteht der SBK aus den ersten 32 Zeichen zwischen den beiden Hochkommata. Für das vorliegende Gerät lautet der SBK: 9abe4e5dc1454890a8905cf8a216d701

Dieser SBK kann nun zum Erzeugen und Signieren des erforderlichen Blobs über die `nvcrypttools` genutzt werden:

```
nvcrypttools : bash — Konsole <3>
[prosch@ThinkProsch nvcrypttools]$ ./mknvfblob -W -K 9abe4e5dc1454890a8905cf8a216d701 --blob ./tf300.blob --bctin ./bct/300.bct --bctr ./testr.bct --bctc ./testc.bct --blin ./bootloaders/bootloader.tf300.XBT --blout .test.ebt -c 0x30
mknvfblob
-----
Encrypting BL './bootloaders/bootloader.tf300.XBT' to '.test.ebt'...done.
Generating encrypted recovery BCT from './bct/300.bct' to '.testr.bct'...done.
Generating encrypted create BCT from './bct/300.bct' to '.testc.bct'...done.
Generating blob file './tf300.blob'...done.
Adding create BCT to blob file [wheelie]...done.
Adding recovery BCT to blob file [wheelie]...done.
Adding bootloader to blob file [wheelie]...done.
Adding odmdata to blob file [wheelie]...done.
Adding chip id to blob file [wheelie]...done.
```

**Bild 72:** Erzeugen der Blob-Datei des TF300T für den Aufruf von `nvflash`

Hierbei wird über:

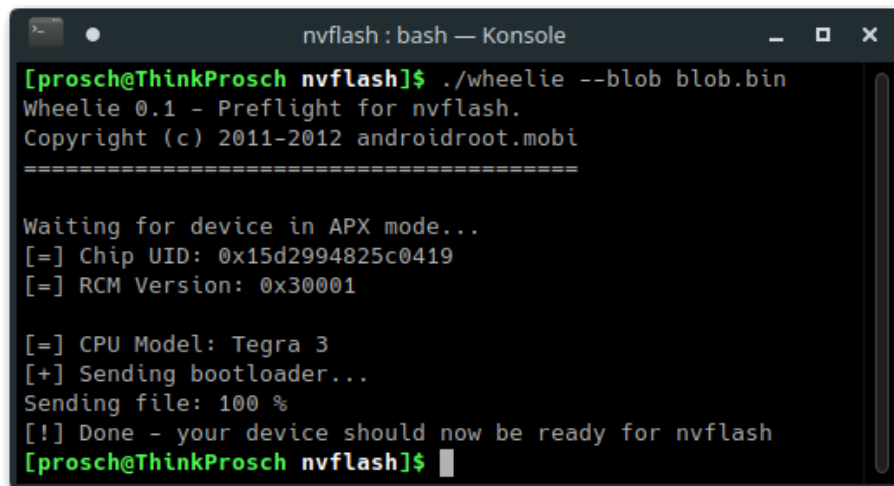
- „-K“ der ermittelte SBK,
- „--bctin“ die heruntergeladene unverschlüsselte BCT und
- „--blin“ der heruntergeladene unverschlüsselte Bootloader

übergeben. Der Parameter „-c 0x30“ beschreibt ein vorliegendes Tegra 3 Gerät. Die Ausgabedateien werden über die folgenden Parameter definiert:

- „--bctr“ → verschlüsselte Recovery BCT
- „--bctc“ → verschlüsselte Create BCT
- „--blout“ → verschlüsselter Bootloader
- „--blob“ → Blob-Datei bestehend aus `bctr`, `bctc` und `blout`

Die erzeugte Blob-Datei (Hier: `tf300.blob`) wird in der Folge an das Tool „wheelie“ aus dem `nvflash`-Verzeichnis übergeben. Da wheelie für die Blob-Datei die Dateiendung „.bin“ erwartet, wurde die Datei in „blob.bin“ umbenannt.

Der Aufruf von wheelie dient der Vorbereitung des Geräts für die folgende Kommunikation über nvflash. Dazu ist das Gerät erneut in den APX-Modus zu versetzen.



```
nvflash : bash — Konsole
[prosch@ThinkProsch nvflash]$ ./wheelie --blob blob.bin
Wheelie 0.1 - Preflight for nvflash.
Copyright (c) 2011-2012 androidroot.mobi
=====

Waiting for device in APX mode...
[=] Chip UID: 0x15d2994825c0419
[=] RCM Version: 0x30001

[=] CPU Model: Tegra 3
[+] Sending bootloader...
Sending file: 100 %
[!] Done - your device should now be ready for nvflash
[prosch@ThinkProsch nvflash]$
```

**Bild 73:** Versetzen des TF300T in den nvflash-Modus über wheelie

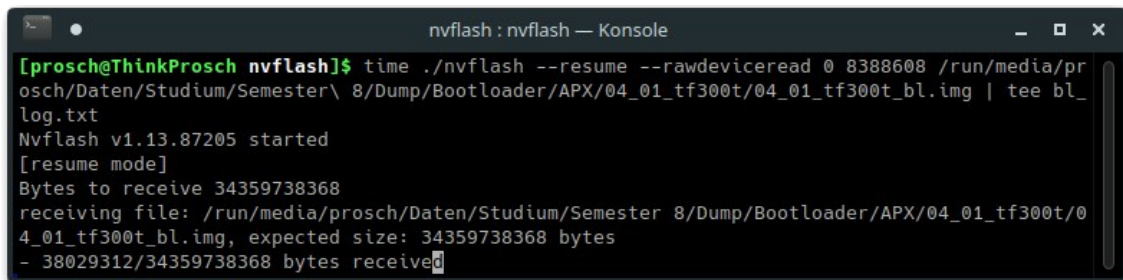
Die Sicherung des Gerätespeichers kann im Anschluss über nvflash erfolgen. Der Aufruf lautet:

```
nvflash --resume --rawdeviceread <Startadresse> <Endadresse> <image.bin>
```

Über „--resume“ wird dabei angegeben, dass bereits ein Gerät im nvflash-Modus wartet. Der Parameter „--rawdeviceread“ lässt das Lesen des eMMC-Speichers zu. Die Adressen werden dabei in Sektoren mit einer Größe von 4096 Bytes<sup>15</sup> angegeben. Da das Gerät mit 32GB internen Speicher beworben wird, werden für die Sicherung zunächst 34.359.738.368 Bytes (32 GiB) angenommen<sup>16</sup>, woraus sich bei genannter Sektorengröße eine Gesamtsektorenzahl von 8388608 ergibt. Bild 74 zeigt den Aufruf der Sicherung über den Bereich 0 bis 8388608. Die Sicherung stoppte bei dem Testgerät nach 31788630016 Bytes. Dies entspricht 29,6 GiB oder einer Sektorenzahl von 7760896. Für die Sicherung des internen Flash-Speichers war die Wahl eines zu großen Speicherbereichs nicht mit Problemen verbunden.

<sup>15</sup> Bei älteren Firmware-Versionen kann die Blockgröße 2048 Bytes betragen.

<sup>16</sup> Die Größen GiB und GB sind zwar klar definiert, werden durch verschiedene Hersteller jedoch falsch verwendet. Davon ausgehend, dass 32 GiB angegeben sind, wird ein größerer Speicherbereich für die Sicherung angenommen. Für die Sicherung ist ein zu groß gewählter Speicherbereich unschädlich, da die Sicherung nach dem letzten Sektor stoppt.



```
nvflash : nvflash — Konsole
[prosch@ThinkProsch nvflash]$ time ./nvflash --resume --rawdeviceread 0 8388608 /run/media/prosch/Daten/Studium/Semester\ 8/Dump/Bootloader/APX/04_01_tf300t/04_01_tf300t_bl.img | tee bl_log.txt
Nvflash v1.13.87205 started
[resume mode]
Bytes to receive 34359738368
receiving file: /run/media/prosch/Daten/Studium/Semester 8/Dump/Bootloader/APX/04_01_tf300t/04_01_tf300t_bl.img, expected size: 34359738368 bytes
- 38029312/34359738368 bytes receive
```

**Bild 74:** Sicherung des internen Flash-Speichers des TF300T über nvflash

Bei dem vorliegenden Gerät befinden sich auf dem internen Flash-Speicher neben dem Kernel noch Reste des Android-Systems. Das relevante RootFS mit dem Nutzerverzeichnis (/home) wurde auf einer  $\mu$ SD-Karte installiert (siehe Anhang Seite W). Die Möglichkeit der Installation des vollständigen Systems auf den internen Flash-Speicher wird aber ebenfalls auf der PostmarketOS-Projektseite beschrieben [135].

## Microsoft Surface 2:

Bei dem Surface 2 handelt es sich ursprünglich um ein Windows RT Tablet. Das OpenRT-Projekt beschreibt die Möglichkeiten des Betriebs von Linux auf dem Gerät. Vorgesehen ist dabei das Vorhalten der Boot-Partition auf einem USB-Stick und des RootFS auf einer  $\mu$ SD-Karte<sup>17</sup> [136][137]. Im Rahmen der Installation (siehe Anhang Seite AW) wurden die benötigten Daten jedoch auf den Gerätespeicher kopiert.

Das Vorgehen bei der Sicherung unterscheidet sich von der vorangegangenen Tegra 3 Methode und wird nachfolgend erläutert.

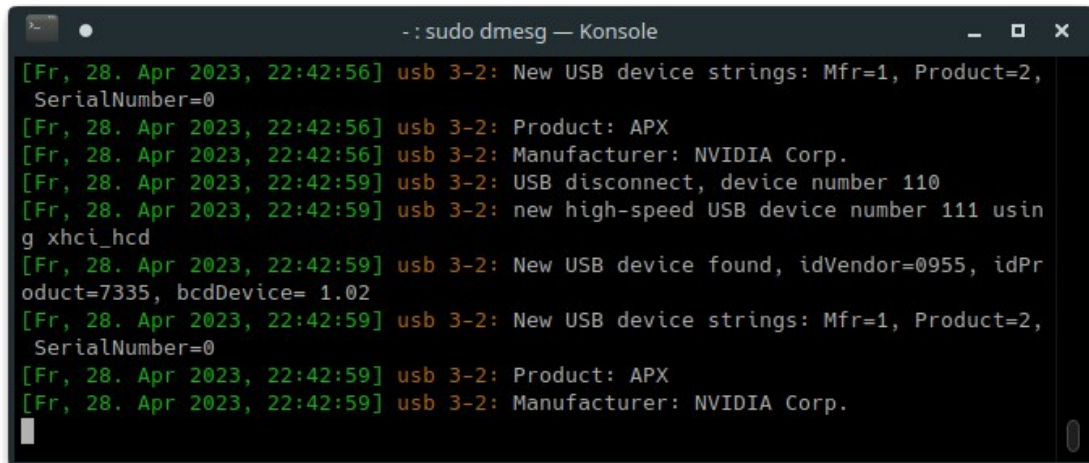
Das vorliegende Surface 2 kann über eine Tastenkombination in den APX-Modus gestartet werden.<sup>18</sup> Dazu wird das ausgeschaltete Gerät über ein USB-Typ-A zu USB-Typ-A-Kabel mit der Workstation verbunden. Anschließend wird die **{lauter}** Taste betätigt und gehalten und einmalig die **{Power}** Taste betätigt.

<sup>17</sup> Im Bearbeitungszeitraum war die Energieversorgung des USB-Ports treiberseitig für das Surface 2 nicht implementiert, so dass nach der Übergabe des EFI-Systems an den Linux Kernel keine weiteren Daten von USB geladen wurden. Daher war die zusätzliche Verwendung einer  $\mu$ SD-Karte erforderlich.

<sup>18</sup> In der OpenRT-Community wird von Geräten berichtet, die den Aufruf des APX-Modus nicht gestatten.



Um nachzuvollziehen, ob der Anschluss im APX-Modus gelungen ist, können über `sudo dmesg -w` Kernel-Meldungen live ausgegeben werden.



```

[Fr, 28. Apr 2023, 22:42:56] usb 3-2: New USB device strings: Mfr=1, Product=2,
  SerialNumber=0
[Fr, 28. Apr 2023, 22:42:56] usb 3-2: Product: APX
[Fr, 28. Apr 2023, 22:42:56] usb 3-2: Manufacturer: NVIDIA Corp.
[Fr, 28. Apr 2023, 22:42:59] usb 3-2: USB disconnect, device number 110
[Fr, 28. Apr 2023, 22:42:59] usb 3-2: new high-speed USB device number 111 usin
g xhci_hcd
[Fr, 28. Apr 2023, 22:42:59] usb 3-2: New USB device found, idVendor=0955, idPr
oduct=7335, bcdDevice= 1.02
[Fr, 28. Apr 2023, 22:42:59] usb 3-2: New USB device strings: Mfr=1, Product=2,
  SerialNumber=0
[Fr, 28. Apr 2023, 22:42:59] usb 3-2: Product: APX
[Fr, 28. Apr 2023, 22:42:59] usb 3-2: Manufacturer: NVIDIA Corp.

```

**Bild 75:** Kernel-Ausgabe beim Aufruf des APX-Modus am Surface 2


Für die weitere Kommunikation mit dem Gerät wurde das „fusee-tools“ Projekt der Open Surface RT Github Instanz mitsamt der enthaltenen Submodule bezogen [138] und die enthaltenen Payloads kompiliert.

Anschließend konnte das Skript „get\_bct.sh“ ausgeführt werden, um die BCT des vorliegenden Geräts zu extrahieren. Im Skript wird auf den allgemeinen Payload „patch\_irom.bin“ verwiesen. Für das vorliegende Tegra 4 Gerät musste dieser Eintrag entsprechend angepasst werden, um den Payload „T114\_patch\_irom.bin“ zu verwenden. Die extrahierte BCT lag nach der Ausführung als „current.bct“ vor und wurde in „surface-2.bct“ umbenannt.

Aus einem vorliegenden physical Dump kann über die BCT die Adresse des Bootloaders für dessen Extraktion ermittelt werden. Da im Rahmen der TCP-Live-Sicherung bereits ein Physical Dump erstellt wurde, wurde der Bootloader an der Adresse 0x20000 mit einer Länge von 787808 Bytes extrahiert und für den folgenden Aufruf von nvflash verwendet. Dabei zeigte sich, dass der Parameter „--rawdeviceread“ bei dem Surface 2 nicht zur Verfügung steht. Für einen Angriff ohne vorliegende Speicherextraktion wurde daher ein anderes Vorgehen gewählt.



Über Github wird seitens der Open Surface RT Community ein Fork des freien Bootloaders U-Boot betreut. Unter der Branch „microsoft-surface-2“ [139] finden sich bereits Konfigurationsdateien zur Erstellung des Bootloaders für das Surface 2. Da eine Bildausgabe auf dem Tablet bei der Verwendung von U-Boot aktuell nicht unterstützt wird, wurde die Konfigurationsdatei „uboot/include/configs/surface-2.h“ entsprechend angepasst, um das Gerät direkt in den Mass Storage Modus zu starten und den internen Flash-Speicher freizugeben (siehe Bild 76).

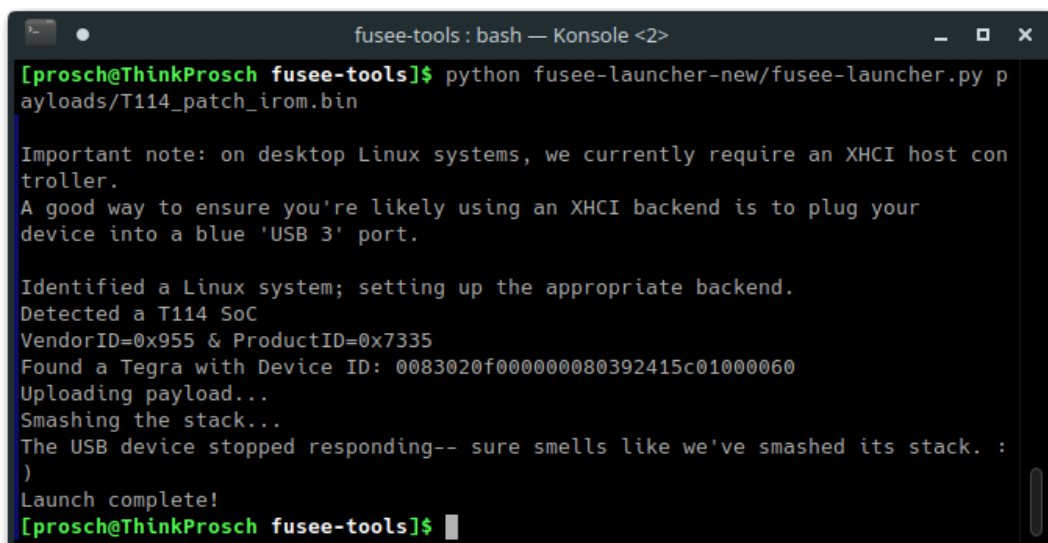


<pre> 192  /* 193  // directly boot to fastboot 194  #undef CONFIG_BOOTCOMMAND 195  #define CONFIG_BOOTCOMMAND \ 196      "env set serial# 0123456789123456;" \ 197      "env set platform \"Tegra 4 T114\";" \ 198      "fastboot usb 0;" 199  */ 200 201  #endif /* __CONFIG_H */ </pre>	<pre> 192  // directly boot to mass storage mode 193  #undef CONFIG_BOOTCOMMAND 194  #define CONFIG_BOOTCOMMAND \ 195      "env set serial# 0123456789123456;" \ 196      "env set platform \"Tegra 4 T114\";" \ 197      "ums 0 mmc 0;" 198 199 200 201  #endif /* __CONFIG_H */ </pre>
--	--

**Bild 76:** Modifikation der U-Boot Konfiguration für den Start in den Mass Storage Mode

Bei der Kompilierung von U-Boot (make surface-2\_defconfig) wurden mehrere Binaries erstellt. Für die Übergabe an das Gerät ist die Datei: „u-boot-dtb-tegra.bin“ geeignet.

Über den erneuten Aufruf des Payloads „T114\_patch\_irom.bin“ wird das Surface 2 für die Entgegnahme des Bootloaders vorbereitet.



```

fusee-tools : bash — Konsole <2>
[prosch@ThinkProsch fusee-tools]$ python fusee-launcher-new/fusee-launcher.py p
ayloads/T114_patch_irom.bin

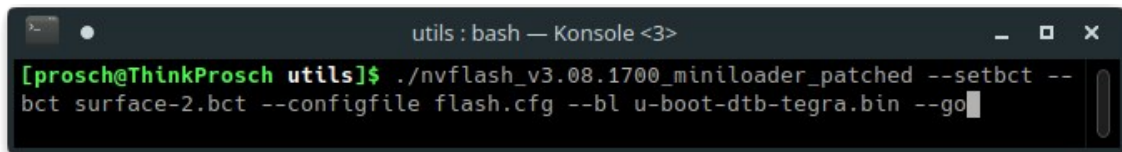
Important note: on desktop Linux systems, we currently require an XHCI host con
troller.
A good way to ensure you're likely using an XHCI backend is to plug your
device into a blue 'USB 3' port.

Identified a Linux system; setting up the appropriate backend.
Detected a T114 SoC
VendorID=0x955 & ProductID=0x7335
Found a Tegra with Device ID: 0083020f000000080392415c01000060
Uploading payload...
Smashing the stack...
The USB device stopped responding-- sure smells like we've smashed its stack. :
)
Launch complete!
[prosch@ThinkProsch fusee-tools]$

```

**Bild 77:** Anwendung des Fusée Gelée Exploits auf das Surface 2

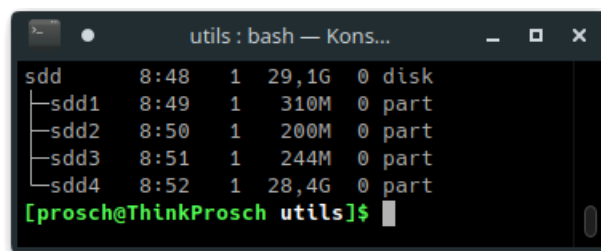
U-Boot kann anschließend über eine modifizierte Version von nvflash (enthalten in den bezogenen fusee-tools) geladen werden:



```
utils : bash — Konsole <3>
[prosch@ThinkProsch utils]$ ./nvflash_v3.08.1700_miniloader_patched --setbct --
bct surface-2.bct --configfile flash.cfg --bl u-boot-dtb-tegra.bin --go
```

**Bild 78:** Starten des erstellten U-Boot Images auf dem Surface 2 über nvflash

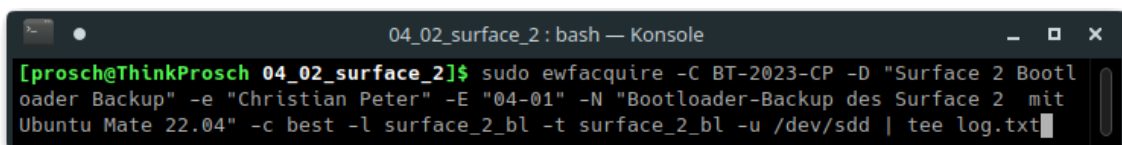
Dabei wird die extrahierte BCT übergeben, um den Bootloader an die korrekte Speicheradresse zu senden. Die Config-Datei „flash.cfg“ ist ebenfalls bereits im Projektverzeichnis enthalten. Nach der Ausführung meldet nvflash einen Kommunikationsfehler. Dies ist zu erwarten, da U-Boot statt eines nvflash-fähigen Bootloaders übergeben wurde. Nach dem Absetzen des Bootbefehls ist der eMMC-Speicher des Surface 2 an der Workstation einsehbar.



```
utils : bash — Kons...
sdd      8:48   1  29,1G   0 disk
├--sdd1   8:49   1   310M   0 part
├--sdd2   8:50   1   200M   0 part
├--sdd3   8:51   1   244M   0 part
└--sdd4   8:52   1  28,4G   0 part
[prosch@ThinkProsch utils]$
```

**Bild 79:** Surface 2 im Mass Storage Mode

Der Gerätespeicher kann nun nach dem beschriebenen Prinzip über ewfacquire gesichert werden.



```
04_02_surface_2 : bash — Konsole
[prosch@ThinkProsch 04_02_surface_2]$ sudo ewfacquire -C BT-2023-CP -D "Surface 2 Bootl
oader Backup" -e "Christian Peter" -E "04-01" -N "Bootloader-Backup des Surface 2 mit
Ubuntu Mate 22.04" -c best -l surface_2_bl -t surface_2_bl -u /dev/sdd | tee log.txt
```

**Bild 80:** Sicherung des Surface 2 im Mass Storage Mode über ewfacquire

Dieses Vorgehen erfordert keine Veränderung am Gerätespeicher und ist unabhängig vom verwendeten Betriebssystem. Bisherige Ansätze der Sicherung von Windows-RT-Geräten sahen umfangreiche Modifikationen am Betriebssystem (Jailbreak) zur Ausführung unsignierter Software vor [140].

#### 6.4.4 FEL-Modus (Allwinner SoCs)

SoCs des Herstellers Allwinner Technology finden sich überwiegend in preisgünstigen Einplatinencomputern wie dem Banana Pi, Orange Pi oder Cubieboard. Die Verwendung von Linux-Systemen auf diesen Plattformen ist üblich und die Community arbeitet aktiv am Mainlining der SoCs [141].

Als Rückfallebene auf Allwinner Geräten wurde der FEL-Mode integriert. Dieser Modus wird vom Bootrom aufgerufen, wenn kein gültiges Bootmedium vorgefunden wird [142]. Zur Kommunikation mit dem BROM kann das Tool „sunxi-fel“ aus dem Paket „sunxi-tools“ [143] genutzt werden. Der Name Sunxi bezeichnet die ARM-SoC-Familie von Allwinner.

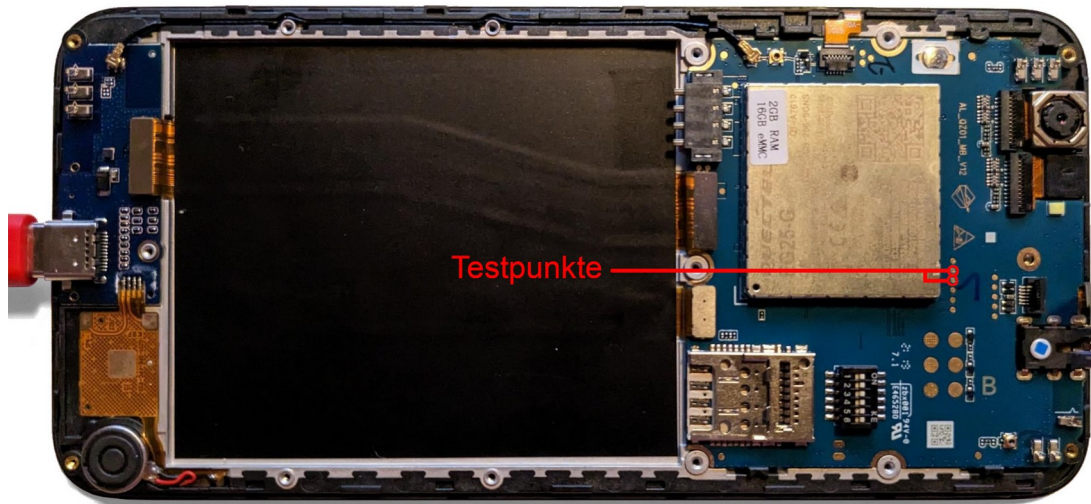
Das BROM-Protokoll ist zwar proprietär, erfordert jedoch aktuell keine weitere Legitimierung und erlaubt so die Ausführung von eigenem Code. Das Linux-Sunxi-Portal listet verschiedene Methoden auf, den FEL-Modus manuell zu erzwingen [144]:

- Betätigen eines definierten FEL-Schalters beim Boot (meist bezeichnet als „recovery“, „uboot“ oder „fel“)
- Halten einer Standard-Taste (z.b. `{lauter}`)
- Verwenden eines speziellen SD-Karten-Images
- Senden bestimmter UART Befehle

Für das vorliegende Pine64 Pinephone wird als verfügbare Methode die Verwendung des SD-Karten-Images benannt [145]. Da jedoch mit „Jumpdrive“ (siehe Seite 95) eine komfortablere Möglichkeit der Gerätesicherung über eine vorbereitete µSD-Karte vorliegt, wurde von dieser Methode Abstand genommen. Im Fall eines defekten µSD-Karten-Slots kann der Aufruf des FEL-Modus die letzte verbleibende Möglichkeit der Datensicherung darstellen, ohne das Gerät dauerhaft zu beschädigen oder zu zerstören.

In einem Forenbeitrag des Pine64-Forums [146] werden mehrere Pads beschrieben, die als Test Point für den Start des FEL-Modus in Betracht

kommen. Über Versuche konnten die Testpunkte ermittelt werden, die zuverlässig den FEL Modus am vorliegenden Gerät triggerten. Diese sind in Bild 81 dargestellt.



**Bild 81:** FEL-Testpunkte des Pine64 Pinephone (Hardware Revision v1.2b)

An der Workstation wird das angeschlossene Pinephone nach der Verbindung der Testpunkte vom Tool sunxi-fel erkannt.

```
01_Pinephone: bash — Konsole
[prosch@ThinkProsch 01_Pinephone]$ sunxi-fel -l
USB device 003:012 Allwinner A64 92c095ba:94104620:79028414:101a0290
```

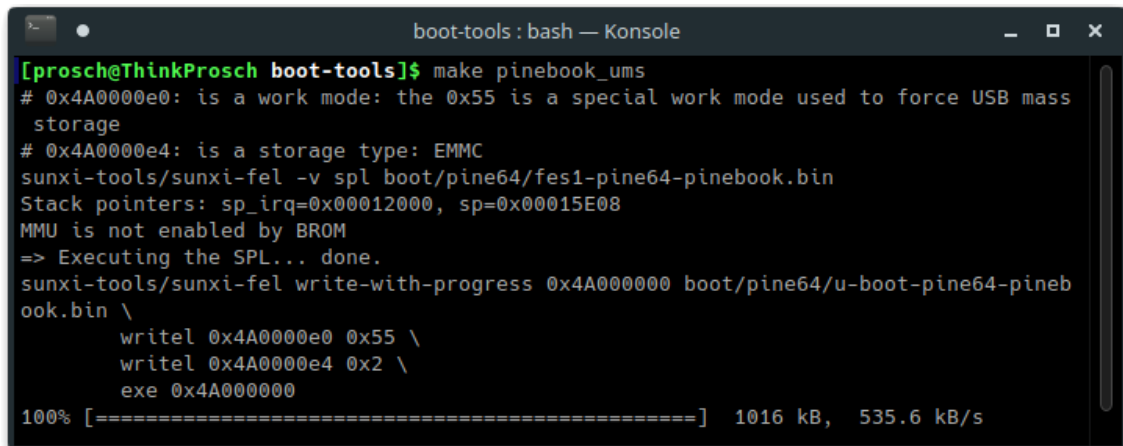
**Bild 82:** Angeschlossenes Pinephone im FEL-Modus

Zur Freigabe des internen eMMC Speichers als Mass Storage Device kann erneut U-Boot genutzt werden. Ein komfortables Boot-Skript und ein entsprechend modifiziertes U-Boot Image ist im „boot-tools“ Repository des Github-Nutzers Kamil Trzcinski abrufbar [147]. Dieses wurde ursprünglich für das Pine64 Pinebook erstellt; aufgrund der ähnlichen Hardware funktioniert das U-Boot-Image aber auch auf dem Pinephone.

Der Aufruf des Boot-Skripts über:

```
make pinebook_ums
```

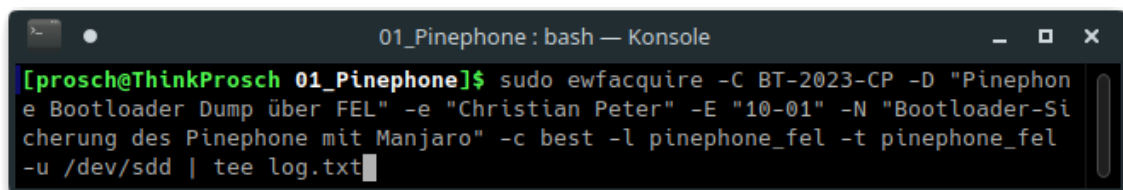
lädt zunächst den passenden Secondary Program Loader (SPL), welchem anschließend das eigentliche U-Boot Image übergeben wird.



```
boot-tools : bash — Konsole
[prosche@ThinkProsche boot-tools]$ make pinebook_ums
# 0x4A0000e0: is a work mode: the 0x55 is a special work mode used to force USB mass
storage
# 0x4A0000e4: is a storage type: EMMC
sunxi-tools/sunxi-fel -v spl boot/pine64/fes1-pine64-pinebook.bin
Stack pointers: sp_irq=0x00012000, sp=0x00015E08
MMU is not enabled by BR0M
=> Executing the SPL... done.
sunxi-tools/sunxi-fel write-with-progress 0x4A000000 boot/pine64/u-boot-pine64-pineb
ook.bin \
    writel 0x4A0000e0 0x55 \
    writel 0x4A0000e4 0x2 \
    exe 0x4A000000
100% [=====] 1016 kB, 535.6 kB/s
```

**Bild 83:** Booten des Pinephones in den U-Boot Mass Storage Mode

Nach der Ausführung des Skripts ist der Speicher des Pinephones an der Workstation verfügbar und kann über ewfacquire im EWF-Format gesichert werden.



```
01_Pinephone : bash — Konsole
[prosche@ThinkProsche 01_Pinephone]$ sudo ewfacquire -C BT-2023-CP -D "Pinephon
e Bootloader Dump über FEL" -e "Christian Peter" -E "10-01" -N "Bootloader-Si
cherung des Pinephone mit Manjaro" -c best -l pinephone_fel -t pinephone_fel
-u /dev/sdd | tee log.txt
```

**Bild 84:** Sicherung des eMMC-Speichers des Pinephone über ewfacquire

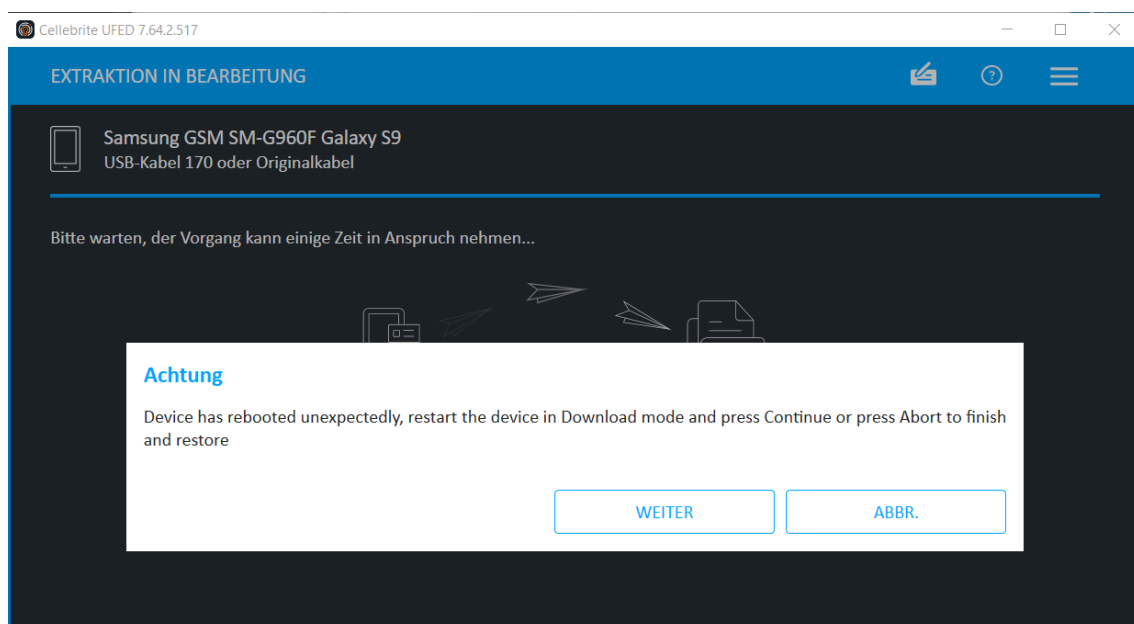
Das erzeugte EWF Image ist mit 2,6 GiB kleiner als das im laufenden System erstellte Image (2,8 GiB). Eine Einsicht in das Dateisystem ist möglich.

Nach der Trennung der USB-Verbindung und dem Einsetzen des Geräteakkus startete das Pinephone wie erwartet in das installierte System.

### 6.4.5 Fazit der Bootloader-Sicherungen

Die dargestellten Möglichkeiten sind nicht abschließend. Auch andere Hersteller von ARM-SoCs stellen Protokolle bereit, welche die Kommunikation mit dem Bootloader erlauben. Für die Testgeräte mit Texas Instruments SoC wurde keine Methode vorgefunden, die zuverlässig die Kommunikation mit dem Bootloader ermöglicht. Für Geräte, die vormals mit Maemo betrieben wurden, kann der „Open Free Fiasco Firmware Flasher“ (0xFFFF) eine Möglichkeit darstellen, einen alternativen Bootloader an entsprechende Geräte zu senden.

Daten der Samsung-Testgeräte konnten nicht im Rahmen der Bootloader-Sicherung extrahiert werden. Freie Software zur Kommunikation mit Exynos-SoCs wurde nicht gefunden. Sicherungsversuche über die kommerzielle Windows-Software UFED4PC im Modus „Bootloader“ oder „decrypting Bootloader“ schlugen fehl, obwohl diese Methoden bei den unmodifizierten Android-Installationen der Testgeräte (Galaxy S III und Galaxy S9) zuverlässig zur physikalischen Sicherung des Gerätespeichers eingesetzt werden können. Auch das Galaxy S5 mit Qualcomm SoC wurde nicht gesichert, da keine Methode zum Aufruf des EDL-Modus ermittelt werden konnte.



**Bild 85:** Fehlschlag der Bootloader-Sicherung des Galaxy S9 über UFED4PC

Die durchgeführten Sicherungen wurden dokumentiert und tabellarisch erfasst. Eine komplette Übersicht der durchgeführten Sicherungen ist im Anhang der Arbeit (Seite BD) einzusehen. Dabei entspricht die Darstellung dem Muster:

**Tabelle 8:** Bootloader-Sicherungen (Auszug)

Gerät	SoC	Modus	Aufruf	Speicher	Backup	Dauer
Redmi 2	Qualcomm	EDL	laut + leise + USBin	14 GiB	2,4 GiB	18min 57s
Surface 2	Nvidia	APX	laut halten + power	29 GiB	5,4 GiB	53min 52s

Bei den 13 Testgeräten mit Qualcomm-SoC konnten für acht Geräte Methoden gefunden werden, den EDL-Modus aufrufen zu können. Für fünf dieser Geräte gelang eine Extraktion des Gerätespeichers über den Bootloader mit dem EDL Client. Die drei Testgeräte mit MediaTek-Chip konnten über das Tool MTKClient physikalisch gesichert werden. Das Testgerät Redmi 9C erforderte dabei jedoch die Verwendung eines geeigneten Preloaders.

Die Sicherung der beiden Tablets mit Nvidia-Tegra-Prozessor im APX-Modus gelang über die Anwendung des „Fusée Gelée“ Exploits. Die Verwendung des Exploits im forensischen Kontext stellt auch für unmodifizierte Geräte eine geeignete Sicherungsmethode dar, die den bisherigen Sicherungsansätzen nach den Anforderungen des Leitfadens IT-Forensik des BSI überlegen ist.

Für das Testgerät Pine64 Pinephone konnte eine Sicherung des internen eMMC-Speichers im FEL-Modus erfolgen. Die bisher undokumentierten Testpunkte zum Aufruf des FEL-Modus bei diesem Gerät konnten gefunden werden. Über diese ist der Aufruf des Modus auch ohne die Möglichkeit des Boots über eine µSD-Karte gegeben.

Insgesamt konnten von den 23 Geräten mit ARM-CPU elf Geräte im Bootloader-Modus gesichert werden. Da keine Interaktion mit der Ausgabe der Geräte über den Touchscreen erfolgt, eignet sich die Bootloader-Sicherung auch für Geräte mit defekten Displays. Auch ist keine funktionierende Installation eines Betriebssystems erforderlich, so dass auch Daten von nicht (softwareseitig) bootfähigen Mobilgeräten gesichert werden können. Die Verwendung von U-Boot bei der Sicherung erlaubte die komprimierte Sicherung der Gerätespeicher ohne den Umweg eines RAW-Dumps.

## 6.5 Live-Linux Sicherungen (x86)

Die zuvor erläuterten Methoden der Sicherung des internen Flash-Speichers bezogen sich überwiegend auf Mobilgeräte mit ARM-SoC. Tablet-PCs, Convertibles oder Mobiltelefone wurden jedoch auch auf Basis der klassischen x86-Architektur gefertigt. Im Gegensatz zu Tower-PC-Systemen oder einer Vielzahl an Notebooks können bei Tablet-PCs die Datenspeicher nicht ohne erheblichen Aufwand aus den Geräten entnommen werden.

Da der Start des installierten Systems potentielle Veränderungen der Daten nach sich ziehen kann und zunächst nicht bekannt ist, ob Schutzmaßnahmen (Antiforensics) gegen die Sicherung integriert wurden, sollte die Möglichkeit genutzt werden, ein nach forensischen Gesichtspunkten geprüftes Betriebssystem zu laden [148 S. 75 – 80]. Für die Sicherung der vorliegenden Testgeräte wurde Tsurugi Acquire [149] gewählt. Diese Variante des Systems basiert auf der 32-Bit Architektur und bietet damit eine höhere Kompatibilität zu verschiedenen Mobilgeräten. Beide x86-Testgeräte (Asus Transformer Book T100TA und Dell Venue 8 Pro) haben 64-Bit kompatible Atom-Prozessoren verbaut. Das UEFI (Unified Extensible Firmware Interface) beider Geräte unterstützt hingegen nur 32-Bit EFI Boot Blobs [150].

Der Ablauf der Sicherung wird am Beispiel des Asus T100TA erläutert. Ein Tsurugi Aquire USB-Image kann von der Projektseite bezogen und mittels Gnome Disks auf einen USB-Stick geschrieben werden. Da das T100TA ohne angedockte Tastatur nicht über einen USB-Typ-A-Anschluss verfügt, ist die Verwendung eines Micro-USB-auf-USB-Typ-A-Adapters („On-The-Go“/OTG-Adapter) erforderlich, um den Bootstick an das Gerät anschließen zu können. Es sollte zudem noch ein Zielmedium und eine Tastatur verwendet werden. Daher wurde ein ausreichend dimensionierter USB-Hub mit dem OTG-Adapter verbunden. Der Start des installierten Betriebssystems war durch entsprechende Änderungen der Bootreihenfolge in den UEFI Einstellungen zu verhindern. Hierbei wurde der Tsurugi Bootstick als erstes Medium gesetzt. Das



UEFI kann bei dem T100TA unter Verwendung der angeschlossenen Tastatur und Betätigung der **{F1}** Taste während des Systemstarts aufgerufen werden.



**Bild 86:** Aufbau der Peripherie für die x86-Live-Sicherung mit Tsurugi Acquire

Tsurugi Acquire hängt gemäß der Voreinstellungen des Systems keine Laufwerke automatisch schreibend ein. Dies ist für das Zielmedium im Filebrowser einmalig manuell durchzuführen. Anschließend konnte die Sicherung des internen Flash-Speichers erfolgen. Das Live-System beinhaltet bereits mehrere Tools, welche zur Datenakquise verwendet werden können. Mit Guymager ist auch eine grafische Oberfläche für die Libewf-Bibliothek enthalten.

GUYMAGER 0.8.12				
Devices Misc Help				
Rescan				
Serial nr.	Linux device	Model	State	Size
01010668e28501e7e...	/dev/sda	USB SanDisk_3.2Gen1	Idle	30.8GB
24441007D6D1B96E	/dev/sdb	SanDisk U3_Cruzer_Micro	Idle	16.0GB
0x00f400ee	/dev/mmcblk2		Running	62.5GB
	/dev/mmcblk2boot1		Idle	4.2MB
	/dev/loop0	filesystem.squashfs	Idle	823.0MB
Size 62,545,461,248 bytes (58.3GiB / 62.5GB)				
Sector size 512				
Image file /media/root/55CC90692CF0BD1E/T100TA_Tsurugi/t100ta_tsurugi.Exx				
Info file /media/root/55CC90692CF0BD1E/T100TA_Tsurugi/t100ta_tsurugi.info				
Current speed 20.22 MB/s				
Started 14. May 13:23:39 (00:00:51)				
Hash calculation MD5				
Source verification off				
Image verification on				
Overall speed (all acquisitions) 20.22 MB/s				

**Bild 87:** Sicherung des internen Flash-Speichers des T100TA über Guymager

### 6.5.1 Fazit der Live-Linux Sicherungen

Beide Testsysteme konnten auf die beschriebene Weise über Tsurugi Acquire gesichert werden. Die Verwendung eines USB-Sticks als Zielspeicher führte bei dem Testgerät Dell Venue 8 Pro zu einem vorzeitigen Abschalten des Geräts und somit zu einer unvollständigen Sicherung. Dies ist auf die zusätzliche Leistungsaufnahme eines USB-Geräts und die altersbedingt geringe Akkuleistung des Geräts zurückzuführen. Bei Verwendung einer µSD-Karte für die Extraktion konnte die Sicherung erfolgreich beendet werden.

Tsurugi Acquire bietet die Möglichkeit des RAM-Boots. Dabei soll das Betriebssystem in den RAM des Geräts geladen werden, so dass der Bootstick nach dem Start des Systems entfernt werden kann. Diese Möglichkeit wurde getestet, um den Akku des Geräts während der Sicherung über USB zu laden. Auf diese Weise konnte keine Sicherung erzeugt werden. Das System stürzte ab, nachdem der Boot-Stick entfernt wurde.

Die Sicherungen wurden protokolliert und sind in der nachfolgenden Tabelle dargestellt:

**Tabelle 9:** Live-Linux-Sicherungen (x86)

Gerät	System	Zielfatenträger	Aufruf UEFI	Speicher	Backup	Dauer
Asus T100TA	Mobian	USB-Stick (32GB)	F1	58 GiB	9,3 GiB	24min 30s
Dell Venue 8 Pro	SailfishOS	µSD-Karte (32 GB)	F2	29 GiB	7 GiB	57min 11s

Gegenüber den TCP-Live-Sicherungen dieser Geräte wurden deutlich kürzere Sicherungszeiten erreicht. Die Sicherung des T100TA über WiFi benötigte mehr als sechs, die des Dell Venue 8 Pro mehr als neun Stunden.

Die Extraktionsgröße des Dell Venue 8 Pro über das Live-System ist mit der TCP-Live-Sicherung vergleichbar. Bei dem Asus T100TA wurde über die Verwendung von Tsurugi eine ca. 600 MiB größere Sicherung erzeugt.

Eine Einsicht in das Dateisystem ist bei den erzeugten Sicherungen möglich.

## 6.6 Gerätespezifische Sicherungen

Neben den bereits erläuterten Methoden gibt es Ansätze, die nur für einzelne oder sehr wenige Geräte anwendbar sind. Dabei werden zum Teil Mechanismen genutzt, die auch bei der Recovery- oder Bootloader-Sicherung Verwendung fanden. Nachfolgend werden zwei Ansätze vorgestellt, die für drei der vorliegenden Testgeräte geeignet sind.

### 6.6.1 Jumpdrive

Für eine wachsende Zahl an „Mainline“-Linux Mobiltelefonen wird das Bootimage „Jumpdrive“ angeboten [151]. Das selbsterklärte „Schweizer Taschenmesser für Mobilgeräte“ gibt nach dem Start direkt den internen eMMC-Speicher des Mobilgeräts frei und ähnelt damit im Ablauf der Bootloader-Sicherung über ein angepasstes U-Boot Image. Der Start von Jumpdrive unterscheidet sich je nach Gerät.

Für das Testgerät Pine64 Pinephone wird das angebotene Jumpdrive-Image über eine geeignete Software auf eine µSD-Karte geschrieben und diese in das Gerät eingesetzt. Da das Pinephone den Start eines Betriebssystems von einer µSD-Karte präferiert [152], wird das installierte Betriebssystem nicht geladen und statt dessen Jumpdrive ausgeführt. Nach dem Anschluss des Geräts per USB an die Workstation ist der Gerätespeicher als Massenspeichergerät verfügbar und kann gesichert werden. Um Veränderungen an den Gerätedaten zu vermeiden, kann bei der Verwendung von Jumpdrive auch ein USB-Writeblocker eingesetzt werden. Auch für das Pocophone F1 steht ein Jumpdrive Image zur Verfügung. Dieses kann wie bei der Recovery Sicherung bei bestehender USB-Verbindung über Fastboot geladen werden:

```
fastboot boot boot-xiaomi-beryllium-tianma.img
```

Nach dem Absetzen des Fastboot-Befehls erscheint ein neues Massenspeichergerät an der Workstation. Da die Verwendung eines Writeblockers das Senden von Fastboot-Befehlen unterbindet, kann dieser erst nach dem Absetzen des Befehls verwendet werden.

Beide Geräte können im Anschluss über ewfacquire gesichert werden.

### 6.6.2 Nokia N9/N950 Rescue-Image

In einem Blogpost des Entwicklers Tobias Brunner [153] wird erläutert, wie der Gerätespeicher des Nokia N9 auch ohne Kenntnis des Gerätesperrcodes eingesehen werden kann. Dazu ist der Kernel des Nokia N950 zusammen mit einem Rescue initrd (initial ramdisk) Image über die Software Flasher auf das Gerät zu laden. Wegen der sehr ähnlichen Hardware soll das Image auch für das Nokia N9 geeignet sein. Die benötigten Images werden über den Blogbeitrag zum Download angeboten. Diese werden über die Software „Flasher“ an das Gerät übertragen. Der Befehl lautet:

```
sudo flasher --load -k vmlinuz-2.6.32.20112201-11.2-adaptation-n950-  
bootloader -n initrd.img-rescue-2.6.32.20112201-11.2-n950 --boot
```

Im Anschluss sollte der Gerätespeicher bestehend aus drei Partitionen an der Workstation einsehbar sein. Dieses Verhalten konnte über mehrere Versuche nicht beobachtet werden. Nach dem Absetzen des Befehls wurde eine Fehlermeldung am Gerät über ein modifiziertes Betriebssystem ausgegeben. Anschließend startete das Gerät in die MeeGo Oberfläche. Der zuvor aktive Gerätesperrcode wurde dabei nicht abgefragt. In diesem Zustand kann eine TCP-Live-Sicherung (siehe Seite 52) nach dem beschriebenen Prinzip durchgeführt werden.

### 6.6.3 Fazit der gerätespezifischen Sicherungen

Die beiden Geräte, für die ein Jumpdrive-Image zur Verfügung steht, konnten an der Workstation eingesehen und gesichert werden. Die Sicherungen wurden protokolliert und tabellarisch erfasst.

**Tabelle 10:** gerätespezifische Sicherungen

Gerät	System	Methode	Aufruf	Speicher	Backup	Dauer
Pinephone	Manjaro	Jumpdrive	µSD-Image	14 GiB	2,6 GiB	24min 58s
Pocophone F1	Mobian	Jumpdrive	Fastboot	53 GiB	10,6 GiB	1h 15min 8s

Der zu sichernde Speicherbereich und die Größe des Abbilds entsprachen bei dem Pine64 Pinephone der Größe vorangegangener Sicherungen. Bei dem Pocophone F1 wurde an der Workstation nur ein Massenspeichergerät mit 53 GiB Speicher vermeldet. Im Rahmen der Recovery-Sicherung (siehe Seite 63) konnte bei diesem Gerät ein Speicherbereich von 59 GiB gesichert werden. Auch die komprimierte EWF-Sicherung ist mit 10,6 GiB kleiner als die Recovery-Sicherung mit 13,2 GiB. Eine Einsicht in die gesicherten Daten lässt den Schluss zu, dass über Jumpdrive lediglich die Userdata-Partition als Massenspeichergerät eingebunden wurde. Bei beiden durchgeführten Extraktionen war eine anschließende Einsicht in das Dateisystem möglich.

Die Verwendung eines Rescue-Images bei dem vorliegenden Nokia N9 führte nicht zu der erwarteten Freigabe des Gerätespeichers ohne die Ausführung des Betriebssystems. Da die eingerichtete Gerätesperre in Form eines PIN-Codes beim Start des Systems umgangen wurde, können jedoch andere Ansätze zur Sicherung des Gerätespeichers Anwendung finden.

## 6.7 externe Flash-Speicher-Sicherungen

Im Rahmen der Vorbereitungen wurde das eigentliche rootFS ( $\approx$  Wurzeldateisystem) dreier Geräte gemäß der jeweiligen Projektleitung auf  $\mu$ SD-Karten installiert. Dabei wurden 32 GB  $\mu$ SD-Karten der Marke SanDisk verwendet.

Die Möglichkeit der Sicherung dieser externen Flash-Speicher bestand bereits im Rahmen der TCP-Live-Sicherung (siehe Seite 52) und der Recovery-Sicherung (siehe Seite 63). Davon ausgehend, dass bei einem Gerät mit unbekannten Zugangsdaten keine weiteren Ansätze für die Live-Datensicherung gefunden wurden und das Gerät ausgeschaltet wurde, oder dass bereits ein ausgeschaltetes Gerät vorliegt, ist gemäß des erarbeiteten Vorgehensmodells (siehe Seite 41) die Entnahme des externen Flash-Speichers zur Datensicherung vorgesehen. Hierbei handelt es sich um die klassische „dead“-Forensik<sup>19</sup>, für die der Leitfaden IT-Forensik der Einsatz eines geeigneten Hardware-Writeblockers zwingend vorgesehen ist [64 S. 26].



**Bild 88:** eingesetzter Writeblocker bei der Sicherung einer  $\mu$ SD-Karte

Für die Sicherung der drei  $\mu$ SD-Karten aus den Geräten: Motorola Droid 4, Asus Transformer Pad TF300T und Samsung Galaxy SIII wurde ein Writeblocker vom Typ „Digital Intelligence USB 3.0 Forensic Card Reader“ verwendet. Der Schreibschutzschalter befand sich in der Stellung „RO“ (Read Only)

<sup>19</sup> Im Gegensatz zur „live“-Forensik wird bei der „dead“-Forensik von ausgeschalteten Geräten mit entnehmbaren Datenträgern ausgegangen.

### 6.7.1 Fazit der externen Flash-Speicher-Sicherungen

Über ewfacquire konnten komprimierte Speicherabbilder der Datenträger erstellt werden. Die durchgeführten Sicherungen wurden protokolliert und tabellarisch erfasst.

**Tabelle 11:** externe Flash-Speicher-Sicherungen

Gerät	System	Speicher	Backup	Dauer
Motorola Droid 4	Maemo Leste	29 GiB	1,2 GiB	10min 16s
Asus Transformer Pad TF300T	PostmarketOS	29 GiB	726,3 MiB	7min 22s
Samsung Galaxy S III	PostmarketOS	29 GiB	991,5 MiB	7min 58s

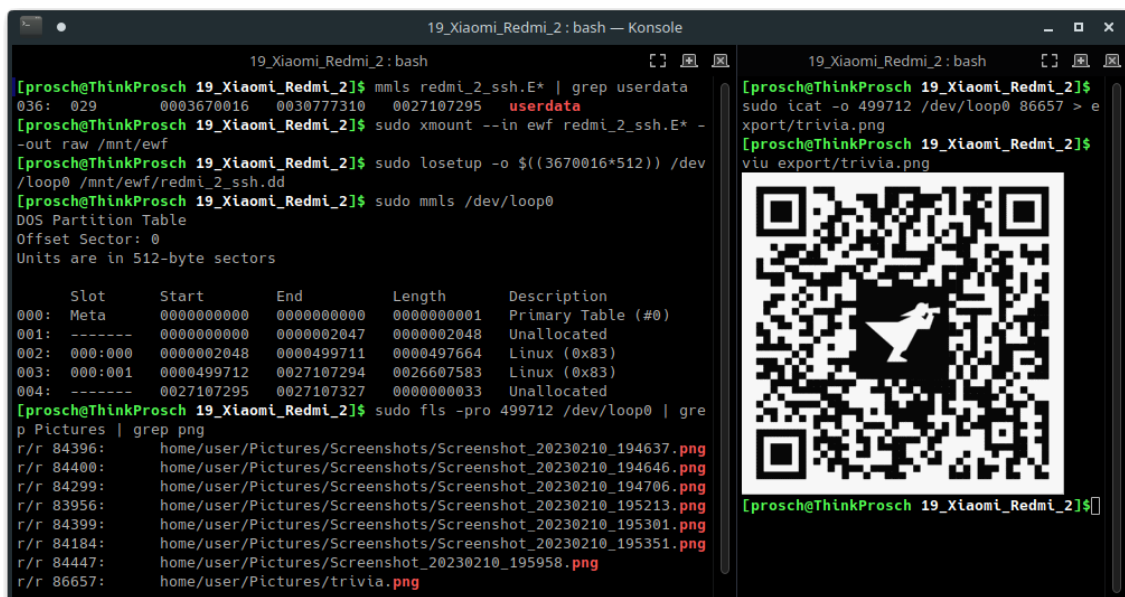
Verglichen mit den TCP-Live-Sicherungen und den Recovery-Sicherungen unterscheiden sich die Dateigrößen der EWF-Images nur um wenige MiB. Die Verwendung eines Write-Blockers ermöglicht hier jedoch tatsächlich bitidentische Sicherungen bei wiederholten Durchführungen der Sicherungen.

Bezüglich des Zeitaufwandes konnte eine erhebliche Zeitersparnis beobachtet werden. Bei der externen Sicherung wurden Zeiten von 7 bis 10 Minuten erreicht. Vorangegangene Sicherungen über SSH oder eine Custom Recovery benötigten Zeiten von 40 Minuten bis über drei Stunden. Da die Recovery-Sicherung bei der Datenakquise des externen Speichers keine Vorteile bietet und zudem ein höheres Risiko potentieller Datenveränderungen in sich trägt, ist der externe Flash-Speicher bei ausgeschalteten Geräten grundsätzlich zu entnehmen und extern zu sichern. Bei eingeschalteten Geräten ist entsprechend vorangegangener Überlegungen (temporäre Dateien, vorliegende Verschlüsselung – siehe Seite 38) bei gegebener Möglichkeit eine Live-Sicherung zu bevorzugen.

Die Datenträgerabbilder konnten nach der Sicherung eingesehen werden und sind somit für weitere Untersuchungen geeignet.

## 7 Einsicht in das Dateisystem

Die Auswertung und Interpretation der gesicherten Daten ist nicht Teil dieser Ausarbeitung. Da die Datensicherung jedoch dem Zweck der späteren Auswertung dient, soll sichergestellt werden, dass diese mit den vorliegenden Daten möglich ist. Eine Einsicht in die Sicherung über das Sleuthkit ermöglicht die direkte Betrachtung der vorliegenden Daten. Bei einem Großteil der Sicherungen können die extrahierten Daten so betrachtet werden, wie es auch bei Desktop-Linux-Systemen der Fall gewesen wäre [72 S. 64]. Einige Geräte weisen jedoch Eigenheiten auf, die so weder bei Desktop-Systemen noch bei Android-Installationen zu beobachten sind. Dieses soll anhand der TCP-Live-Sicherung des Xiaomi Redmi 2 Pro demonstriert werden (siehe Bild 89 und nachfolgende Erläuterung).



**Bild 89:** Einsicht in die Datensicherung des Redmi 2 und Extraktion einer Bilddatei

Über `ewfacquirestream` wurde eine gesplittete EWF-Sicherung bestehend aus den Dateien `redmi_2_ssh.E01` und `redmi_2_ssh.E02` erzeugt.

Diese Sicherung wurde mittels `mmls` eingesehen und zeigte die üblichen Strukturen eines Android-Geräts. Ein zusätzlicher Aufruf von `grep` erlaubt die Anzeige des Offsets der Userdata-Partition:



```
mmls redmi_2_ssh.E* | grep userdata
```

Dabei ist ersichtlich, dass die Partition am Startoffset 3670016 zu finden ist. Es wurde der Versuch unternommen, per `fls` in das Dateisystem der Partition einzusehen:

```
fls -o 3670016 redmi_2_ssh.E*
```

Woraufhin `fls` ein unbekanntes Dateisystem vermeldete.

Um die Userdata-Partition direkt als Blockgerät ansprechen zu können, wurde das Image zunächst über `xmount` als virtuelles RAW-Image bereitgestellt:

```
sudo xmount --in ewf redmi_2_ssh.E* --out raw /mnt/ewf
```

und die Partition anschließend als Loop-Device eingebunden:

```
sudo losetup -o $((3670016*512)) /dev/loop0 /mnt/ewf/redmi_2_ssh.dd
```

Es wurde erneut `mmls` genutzt, um das eingebundene Loop-Device einzusehen:

```
sudo mmls /dev/loop0
```

wobei wieder eine Partitionstabelle gefunden wurde:

```
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
```

	Slot	Start	End	Length	Description
000:	Meta	0000000000	0000000000	0000000001	Primary Table (#0)
001:	-----	0000000000	0000002047	0000002048	Unallocated
002:	000:000	0000002048	0000499711	0000497664	Linux (0x83)
003:	000:001	0000499712	0027107294	0026607583	Linux (0x83)
004:	-----	0027107295	0027107327	0000000033	Unallocated

**Bild 90:** Partitionstabelle der eingebundenen Userdata-Partition

Als größte Partition konnte nun die Nr. 003 am Offset 499712 ausgemacht werden. Weitere Informationen zu dieser Partition wurden über `fsstat` ausgegeben:

```
sudo fsstat -o 499712 /dev/loop0 | head
```

Es zeigte sich ein Ext4 Dateisystem mit dem Label „pmOS\_root“.

```
FILE SYSTEM INFORMATION
-----
File System Type: Ext4
Volume Name: pmOS_root
Volume ID: 7d3ff04c6d8e7b93e8400e8de52870ab

Last Written at: 2022-10-25 19:18:22 (CEST)
Last Checked at: 2022-10-19 16:25:39 (CEST)

Last Mounted at: 2023-04-09 22:55:48 (CEST)
```

**Bild 91:** Ausgabe von Dateisysteminformationen über fsstat

Die Bezeichnung lässt darauf schließen, dass es sich um das Wurzeldateisystem der PostmarketOS (pmOS) Installation handelt. Dieses kann nun über `fls` eingesehen werden. Auch die Extraktion einzelner Dateien über `icat` ist möglich. Bild 89 (Seite 100) zeigt das Einbinden der Userdata-Partition und die gezielte Extraktion einer Bilddatei aus der Sicherung.

Andere Geräte nutzten logische Volumes zur Verwaltung verschiedener Systembereiche. Die Interpretation logischer Volumes wird aktuell durch das Sleuthkit nicht unterstützt, so dass Boardmittel der forensischen Workstation zu nutzen sind um diese einzusehen. Die Einbindung logischer Volumes wird im Abschnitt 8 – Entschlüsselung der Sicherung (Seite 103) erläutert, da die Testgeräte mit aktiver Speicherverschlüsselung gemäß der Standardkonfiguration der Verschlüsselung auch logische Volumes verwenden.

## 8 Entschlüsselung der Sicherungen

Die beiden Testgeräte Sony Xperia XA2 und Sony Xperia 5 weisen eine aktivierte Full Disc Encryption (FDE) auf. Bei dem Xperia XA2 erfolgte die Verschlüsselung im Rahmen der Installation von SailfishOS während des ersten Gerätestarts, ohne dass eine aktive Entscheidung des Nutzers gefordert war (siehe Anhang Seite Q). Die SailfishOS-Dokumentation beschreibt, dass zur Verschlüsselung des /home-Verzeichnisses LUKS (Linux Unified Key Setup) verwendet wird [41]. Erst mit Version 4.5.0.16 wurde im Februar 2023 die Möglichkeit implementiert, alphanumerische Passwörter für die Verschlüsselung zu verwenden [154]. Zuvor war die Nutzung eines PIN-Codes vorgesehen.

Die Droidian-Installation des Sony Xperia 5 wurde erst nach der Ersteinrichtung des Systems verschlüsselt. Die Github-Dokumentation des Projekts benennt auch hier LUKS als Verschlüsselungsmethode [155]. Konkret wird die Nutzung von LUKS2 und die Auslagerung des LUKS-Headers in ein eigenes logisches Volume beschrieben.

LUKS gilt als Standardverfahren der Linux-Datenträgerverschlüsselung. Das Format ist im Paket Cryptsetup [156] enthalten und setzt mit dm-crypt auf dem Device-Mapper-Subsystem des Linux-Kernels auf [158]. Für die eigentliche Verschlüsselung wird ein Master-Key generiert, der für ein symmetrisches Verschlüsselungsverfahren (SEA – Symmetric Encryption Algorithm) eingesetzt wird. Dieser Master-Key wird über das gleiche SEA mit einem vom Nutzer gewählten Passwort (Password-Based Key) verschlüsselt. Zusammen mit Informationen zu den eingesetzten Algorithmen wird der verschlüsselte Master-Key im LUKS-Header abgelegt [157].<sup>20</sup>

Da bei der Entschlüsselung beider Geräte unterschiedliche Vorgehensweisen erforderlich sind, wird der Ablauf für beide Geräte erläutert.

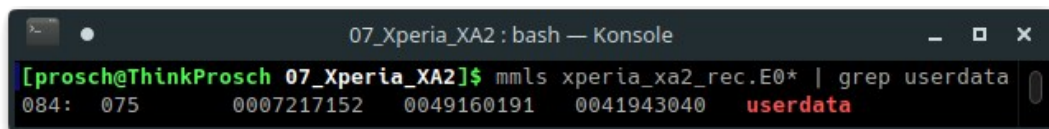
---

<sup>20</sup> Die Funktionsweise ist im Rahmen dieser Arbeit stark vereinfacht dargestellt. Der Einsatz von Key-Slots, Master-Key-Stripes und Key-Kandidaten wird im Detail unter [157] erläutert. Für das Verständnis der nachfolgenden Dokumentation der Entschlüsselung genügt die vereinfachte Darstellung.

## 8.1 Entschlüsselung von SailfishOS (Xperia XA2)

In einem Artikel des Online-Magazins „Forensic Focus“ wird eine Möglichkeit beschrieben, das LUKS-Passwort über einen Brute-force-Ansatz mittels Hashcat zu ermitteln und die vorliegende Sicherung im Anschluss zu entschlüsseln [159]. Dieses Vorgehen konnte bei der Entschlüsselung der Recovery-Sicherung des Sony Xperia XA2 zu großen Teilen angewandt werden.

Wie bereits bei der Einsicht in das Dateisystem des Redmi 2 (siehe Seite 100), wurde zunächst das Offset der Userdata-Partition über `mm1s` ermittelt:



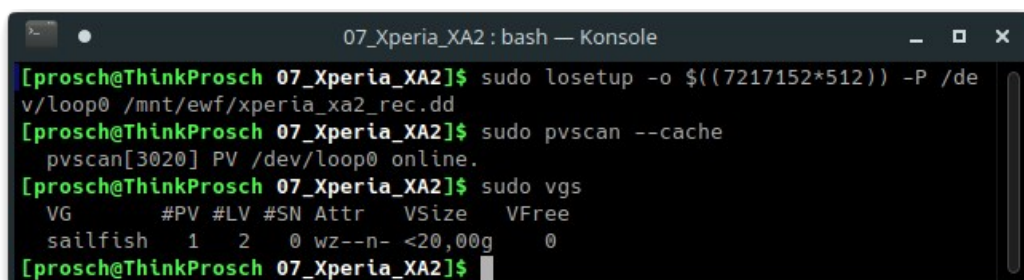
```
07_Xperia_XA2 : bash — Konsole
[prosch@ThinkProsch 07_Xperia_XA2]$ mm1s xperia_xa2_rec.E0* | grep userdata
084: 075      0007217152  0049160191  0041943040  userdata
```

**Bild 92:** Offset der Userdata-Partition des Sony Xperia XA2

Für das weitere Vorgehen war über `xmount` ein RAW-Image bereitzustellen:

```
sudo xmount --in ewf xperia_xa2_rec.E* --out raw /mnt/ewf
```

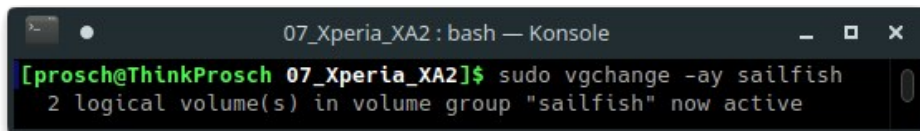
Die Userdata-Partition konnte über das ermittelte Offset (7217152) als Loop-Device eingebunden werden. Da das Sleuthkit die Interpretation logischer Volumes aktuell nicht unterstützt, wurden Befehle des Logical Volume Managers (`lvm`) der Manjaro-Installation der Workstation genutzt. Dabei sucht das Tool `pvscan` nach neuen physischen Volumes. Über `vgs` wurden die zugehörigen Volume-Gruppen und die darin enthaltenen logischen Volumes aufgefunden.



```
07_Xperia_XA2 : bash — Konsole
[prosch@ThinkProsch 07_Xperia_XA2]$ sudo losetup -o $((7217152*512)) -P /dev/loop0 /mnt/ewf/xperia_xa2_rec.dd
[prosch@ThinkProsch 07_Xperia_XA2]$ sudo pvscan --cache
pvscan[3020] PV /dev/loop0 online.
[prosch@ThinkProsch 07_Xperia_XA2]$ sudo vgs
VG      #PV #LV #SN Attr   VSize   VFree
sailfish 1  2  0 wz--n- <20,00g  0
[prosch@ThinkProsch 07_Xperia_XA2]$
```

**Bild 93:** Einbinden der Userdata-Partition und darin enthaltener logischer Volumes

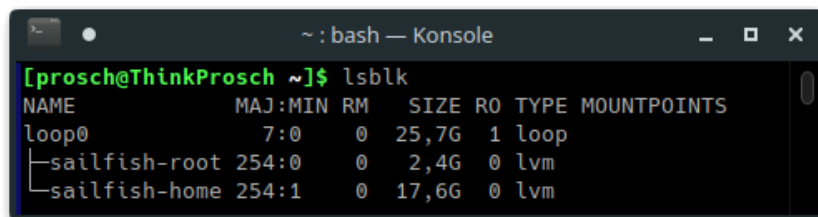
Aus Bild 93 geht hervor: Die Userdata-Partition (loop0) wurde als Physical Volume (PV) eingerichtet und der Volume Group (VG) „sailfish“ zugewiesen. Diese enthält zwei Logical Volumes (LV). Häufig werden die Volumes nach dem Absetzen des `lv` Befehls bereits über den Device Mapper des Systems eingebunden. Geschieht dies nicht automatisch, können die logischen Volumes über `vgchange` manuell aktiviert werden:



```
07_Xperia_XA2: bash — Konsole
[prosche@ThinkProsch 07_Xperia_XA2]$ sudo vgchange -ay sailfish
2 logical volume(s) in volume group "sailfish" now active
```

**Bild 94:** Aktivieren der logischen Volumes der Gruppe "sailfish"

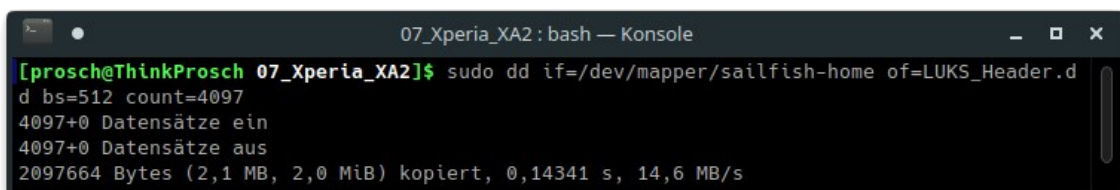
Nach dem Aufruf waren die Volumes „sailfish-root“ und „sailfish-home“ im System eingebunden.



```
~: bash — Konsole
[prosche@ThinkProsch ~]$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
loop0         7:0    0 25,7G  1 loop
└─sailfish-root 254:0    0   2,4G  0 lvm
└─sailfish-home 254:1    0 17,6G  0 lvm
```

**Bild 95:** eingebundene lvm-Volumes im System der Workstation

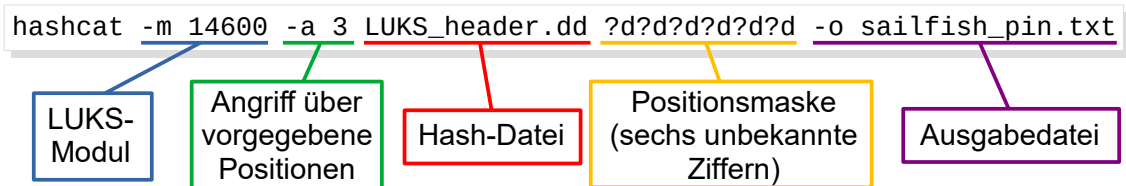
Bei „sailfish-home“ handelt es sich um das LUKS-verschlüsselte Home-Verzeichnis. Hashcat benötigt für den Brute-force den LUKS-Header und eine kleine Menge der verschlüsselten Daten [160]. Der Header befindet sich standardmäßig am Beginn des verschlüsselten Volumes und kann mittels `dd` für die Übergabe an Hashcat extrahiert werden (siehe Bild 96). Für LUKS1 hat sich eine Datenmenge von 2 MiB etabliert ( $4097 \cdot 512$  Bytes). Im erwähnten Artikel des Magazins Forensic Focus werden fälschlich  $4079 \cdot 512$  Bytes beschrieben.



```
07_Xperia_XA2: bash — Konsole
[prosche@ThinkProsch 07_Xperia_XA2]$ sudo dd if=/dev/mapper/sailfish-home of=LUKS_Header.d
d bs=512 count=4097
4097+0 Datensätze ein
4097+0 Datensätze aus
2097664 Bytes (2,1 MB, 2,0 MiB) kopiert, 0,14341 s, 14,6 MB/s
```

**Bild 96:** Extraktion des LUKS-Headers mittels `dd`

Der extrahierte Header konnte im Anschluss an Hashcat übergeben werden. Dabei wurden die folgenden Parameter genutzt:



Unter Verwendung der CPU der Workstation konnte die Nutzer-PIN (753159) nach ca. sechs Stunden ermittelt werden. So war die Entschlüsselung der Home-Partition mittels `cryptsetup` und der korrekten Passphrase möglich. Als Bezeichnung für das neue Blockdevice wurde „`sailfish_unencrypted`“ gewählt.

```
07_Xperia_XA2: bash — Konsole
[prosch@ThinkProsch 07_Xperia_XA2]$ sudo cryptsetup luksOpen /dev/mapper/sailfish-home sailfish_unencrypted
Geben Sie die Passphrase für »/dev/mapper/sailfish-home« ein:
```

**Bild 97:** Entschlüsselung der Home-Partition des Xperia XA2 über `cryptsetup luksOpen`

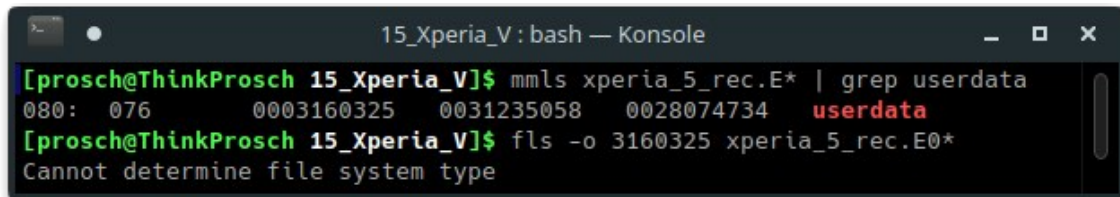
Anschließend konnte das entschlüsselte Volume über `fls` eingesehen werden.

```
07_Xperia_XA2 : bash — Konsole <2>
[prosch@ThinkProsch 07_Xperia_XA2]$ sudo fls /dev/mapper/sailfish_unencrypted
[sudo] Passwort für prosch:
d/d 11: lost+found
d/d 783361: .system
r/r 13: aquota.user
d/d 652801: .zypp-cache
d/d 261121: defaultuser
r/r * 13(realloc): aquota.user.new
V/V 1150561: $OrphanFiles
```

**Bild 98:** Einsicht in das entschlüsselte Home-Verzeichnis des Xperia XA2

## 8.2 Entschlüsselung von Droidian (Xperia 5)

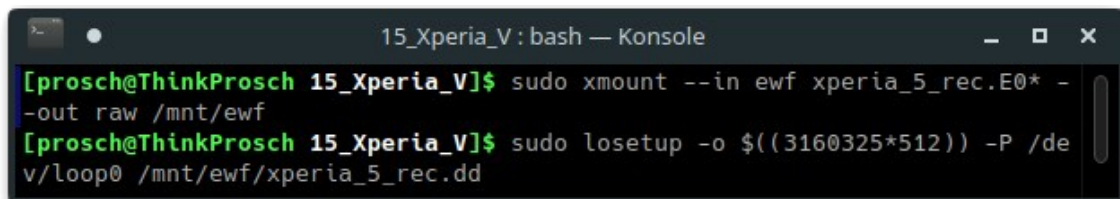
Wie bereits bei den vorangegangenen Einsichten wurde die Sicherung zunächst über `mmls` untersucht und das Offset der Userdata-Partition ermittelt. Sleuthkit-Befehle können das Dateisystem der Partition nicht interpretieren.



```
15_Xperia_V: bash — Konsole
[prosch@ThinkProsch 15_Xperia_V]$ mmls xperia_5_rec.E* | grep userdata
080: 076      0003160325    0031235058    0028074734    userdata
[prosch@ThinkProsch 15_Xperia_V]$ fls -o 3160325 xperia_5_rec.E0*
Cannot determine file system type
```

**Bild 99:** Ermittlung des Offsets der Userdata-Partition des Xperia 5 über `mmls`

Auch bei dieser Sicherung wurde über `xmount` ein RAW-Image für die weitere Analyse bereitgestellt und die Userdata-Partition als Loop-Device eingebunden:



```
15_Xperia_V: bash — Konsole
[prosch@ThinkProsch 15_Xperia_V]$ sudo xmount --in ewf xperia_5_rec.E0* -
-out raw /mnt/ewf
[prosch@ThinkProsch 15_Xperia_V]$ sudo losetup -o $((3160325*512)) -P /de
v/loop0 /mnt/ewf/xperia_5_rec.dd
```

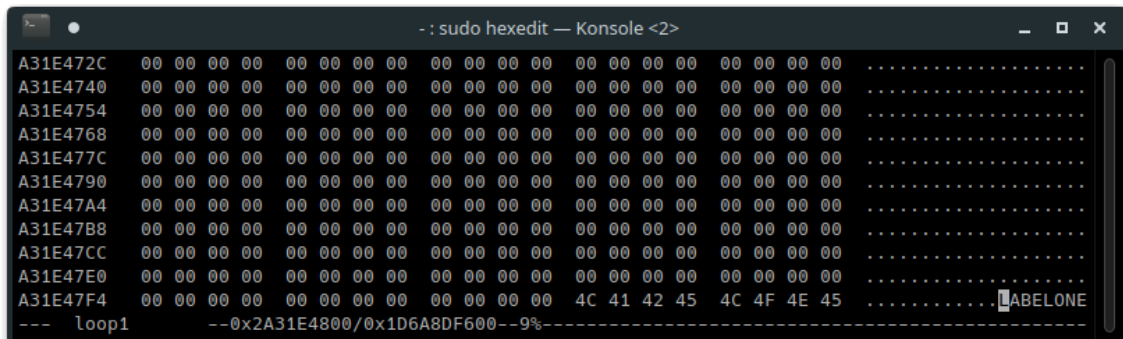
**Bild 100:** Einbinden der Userdata-Partition des Xperia 5 als Loop-Device.

Im Gegensatz zum dargestellten Vorgehen für die SailfishOS-Installation des Xperia XA2, wurde über das Tool `pvsan` nach der Einbindung des Loop-Devices bei der Xperia 5 Sicherung kein Physical Volume aufgefunden. Auch über `mmls` wurden keine weiteren Partitionen ausgegeben.

Die Dokumentation des Logical Volume Managers beschreibt, dass der Physical Volume Label Header eines physischen Volumes in dessen zweiten Sektor zu finden ist und die Bezeichnung „LABELONE“ trägt [161]. Zur Einsicht in das Loop-Device wurde der Linux-Hexeditor „Hexedit“ verwendet. Eine Suche nach dem Text „LABELONE“ fand den Eintrag an der Adresse `0x2A31E4800` (siehe Bild 101). Umgerechnet in Dezimalzahlen entspricht dies

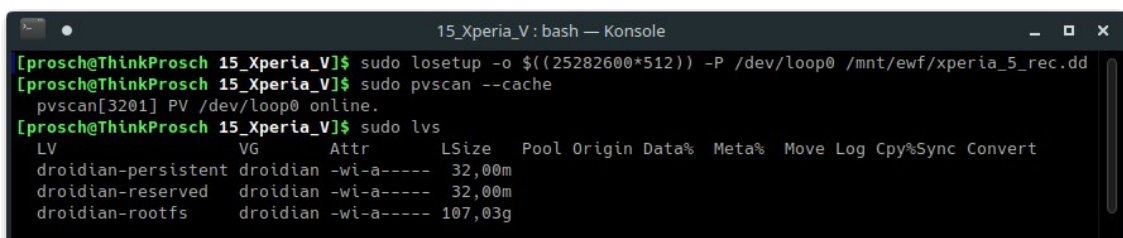


11326605312 Bytes. Bei einer Sektorengröße von 512 Bytes ergibt sich die Position des Headers bei Sektor 22122276.



**Bild 101:** Suche nach dem Physical Volume Label Header mittels Hexedit

Der Angabe der Dokumentation folgend, wird ein weiterer Sektor abgezogen und der Beginn des Physical Volumes bei Sektor 22122275 verortet. Da auch das Sleuthkit bei der Angabe der Partitionsoffsets von 512 Byte großen Sektoren ausgeht, wurde das ermittelte Offset auf das ausgegebene Offset der Userdata-Partition (3160325) aufaddiert. Das neue Partitionsoffset lautet 25282600. Nachdem das Loop-Device unter Verwendung des neuen Offsets erneut eingebunden wurde, konnten auch die logischen Volumes über die LVM-Kommandos aufgefunden werden.



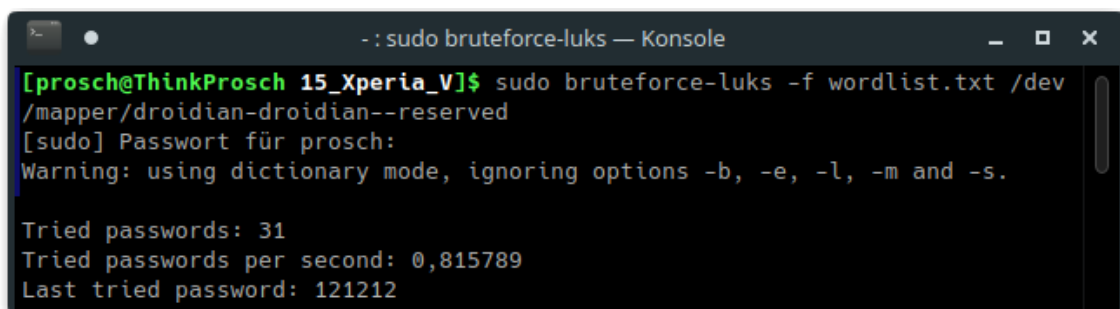
**Bild 102:** Auffinden der logischen Volumes am ermittelten Offset

In der Dokumentation der Droidian-Verschlüsselung [155] wird beschrieben, dass LUKS2 für die Verschlüsselung des rootFS-Volumes verwendet wird. Weiter wird erläutert, dass sich der LUKS-Header nicht am Anfang des verschlüsselten Volumes sondern im separaten Volume „droidian-reserved“ befindet.



Hashcat ist bisher nicht mit LUKS2 kompatibel und vermeldet bei einem Versuch, den Header für einen Bruteforce zu verwenden, eine inkompatible LUKS-Version.

Um einen Angriff auf das Nutzerpasswort durchzuführen, wurde das Tool `bruteforce-luks` [162] bezogen. Dazu wurde eine Passwortliste über 500 Einträge erzeugt und das korrekte Passwort (753159) an das Ende der Liste gesetzt. Bei dem Aufruf von `bruteforce-luks` wird das eingebundene Volume „droidian-reserved“ und die erstellte Wortliste übergeben.

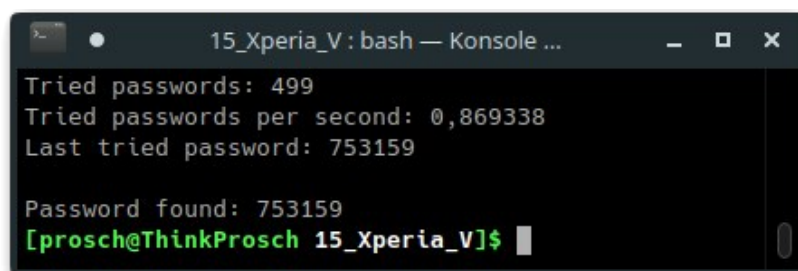


```
- : sudo bruteforce-luks — Konsole
[prosch@ThinkProsch 15_Xperia_V]$ sudo bruteforce-luks -f wordlist.txt /dev
/mapper/droidian-droidian--reserved
[sudo] Passwort für prosch:
Warning: using dictionary mode, ignoring options -b, -e, -l, -m and -s.

Tried passwords: 31
Tried passwords per second: 0,815789
Last tried password: 121212
```

**Bild 103:** Bruteforce der LUKS2-Verschlüsselung über "bruteforce-luks" und eine Wortliste

Nach 9 Minuten und 34 Sekunden wurde das korrekte Passwort ausgegeben:



```
15_Xperia_V: bash — Konsole ...
Tried passwords: 499
Tried passwords per second: 0,869338
Last tried password: 753159

Password found: 753159
[prosch@ThinkProsch 15_Xperia_V]$
```

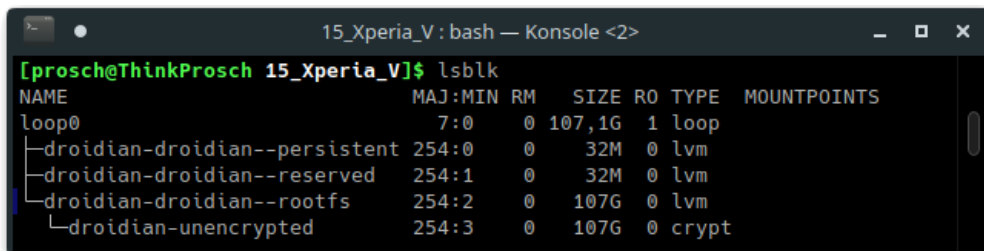
**Bild 104:** Ausgabe des gefundenen LUKS-Nutzerpassworts

Es zeigt sich, dass der Bruteforce-Ansatz des LUKS2-verschlüsselten Geräts wesentlich zeitintensiver ist als die vorangegangene Nutzung von Hashcat für den Angriff auf das Xperia XA2. Da das Tool `bruteforce-luks` mehrere Aufrufe von `cryptsetup` parallelisiert, kann der Einsatz leistungsfähigerer Hardware die Ausführungszeit erheblich verringern.

Nach der Ermittlung des Passworts kann auch die Sicherung des Xperia 5 über cryptsetup entschlüsselt werden:

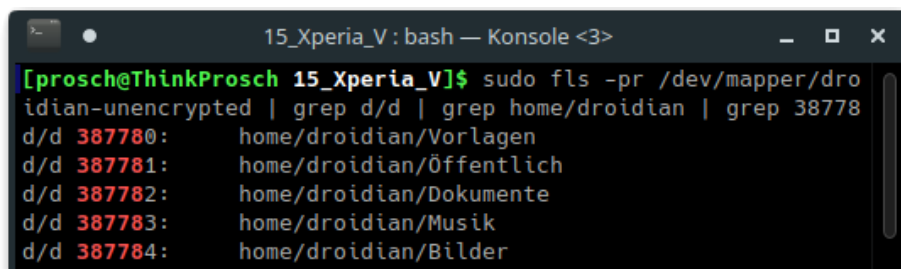
```
sudo cryptsetup luksOpen /dev/mapper/droidian-droidian--rootfs »  
droidian-unencrypted --header /dev/mapper/droidian-droidian--reserved
```

Anschließend ist das unverschlüsselte Wurzeldateisystem im System der Workstation eingebunden (siehe Bild 105) und kann mittels fls eingesehen werden (siehe Bild 106).



```
15_Xperia_V: bash — Konsole <2>  
[prosch@ThinkProsch 15_Xperia_V]$ lsblk  
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS  
loop0                              7:0      0 107,1G  1 loop  
└─droidian-droidian--persistent    254:0      0   32M  0 lvm  
└─droidian-droidian--reserved      254:1      0   32M  0 lvm  
└─droidian-droidian--rootfs        254:2      0  107G  0 lvm  
    └─droidian-unencrypted          254:3      0  107G  0 crypt
```

**Bild 105:** Eingebundenes und entschlüsseltes Wurzeldateisystem des Xperia 5



```
15_Xperia_V: bash — Konsole <3>  
[prosch@ThinkProsch 15_Xperia_V]$ sudo fls -pr /dev/mapper/droidian-unencrypted | grep d/d | grep home/droidian | grep 38778  
d/d 387780:    home/droidian/Vorlagen  
d/d 387781:    home/droidian/Öffentlich  
d/d 387782:    home/droidian/Dokumente  
d/d 387783:    home/droidian/Musik  
d/d 387784:    home/droidian/Bilder
```

**Bild 106:** Einsicht in die entschlüsselte Sicherung über fls

## 9 Bewertung der Ergebnisse

Es wurden übliche Methoden der Datenakquise bei Desktop-Linux-Installationen und Android-Geräten zusammengetragen und anhand der vorbereiteten Testgeräte auf ihre Nutzbarkeit bei mobilen Linux-Installationen hin untersucht. Darüber hinaus konnten Möglichkeiten in den Bereichen der Bootloader-Sicherungen und gerätespezifischer Sicherungen aufgezeigt werden, die bisher im forensischen Kontext keine Verwendung fanden. Die erzielten Sicherungszeiten wurden erfasst und können den erstellten Sicherungsübersichten (siehe Anhang Seite BB ff.) entnommen werden. Da es bei den verschiedenen Sicherungsmethoden Ausreißer bei der Dauer der Sicherung gab, ist eine allgemeingültige Aussage, welche der Methoden diesbezüglich zu bevorzugen ist, nicht zu treffen. Tabelle 12 zeigt eine Übersicht über die ermittelten Sicherungsmethoden. Dabei wird dargestellt, auf wie viele Testgeräte sich die Methode entsprechend ihrer Beschreibung oder Dokumentation theoretisch anwenden lassen sollte, ob für diese Geräte im Versuch tatsächlich ein Ansatz zur Anwendung der Methode gefunden wurde und wie viele Sicherungen erfolgreich durchgeführt werden konnten. Weiter wurde erfasst, wie viele Geräte unter Umgehung des Sperrcodes gesichert werden konnten und bei wie vielen dies nur bei der jeweiligen Methode gelang.

**Tabelle 12:** Übersicht der erfolgten Sicherungen nach Methode

Insgesamt 25 Testgeräte	TCP-Live	ADB-Live	Recovery	Bootloader	Live-Linux	spezifisch	extern
theoretisch anwendbar	25	6	20	23	2	3	3
im Versuch anwendbar	23	4	19	14	2	3	3
erfolgreiche Sicherungen	23	3	18	11	2	3	3
Sicherungserfolg (in %)	100	75	94,7	78,6	100	100	100
gesicherte Geräte (in %)	92	12	72	44	8	12	12
Umgehung der Sperre	0	0	18	11	2	3	3
davon exklusiv bei der Methode	0	0	8	2	2	1	1

## 9.1 Bewertung der TCP-Live-Sicherung

Diese Methode eignet sich für Geräte mit bekannten Zugangsdaten, die sich zum Zeitpunkt der Untersuchung im eingeschalteten Zustand befinden. Erforderliche Software zur Durchführung der Sicherung (OpenSSH oder Netcat) war auf 23 der 25 Testgeräte bereits vorinstalliert. Das nachträgliche Aufspielen dieser Software auf den zwei Geräten ohne entsprechende Installation ist möglich, stellt aber eine Modifikation des Systems dar, die vermieden werden sollte. Daher wurde bei den durchgeführten Versuchen darauf verzichtet. Die unverschlüsselte Sicherung der Nutzerdaten kann auch bei verschlüsselten Geräten erfolgen. Auch die Sicherung des RAM mittels LiME (siehe Seite 44) oder AVML (siehe Seite 47) kann über eine TCP-Verbindung erfolgen. Es kann sowohl der interne Flash-Speicher als auch die eingelegte µSD-Karte physikalisch gesichert werden. Die Sicherung über SSH bietet den Vorteil, den Befehl für die Datenakquise auf der Workstation formulieren und absetzen zu können und so bei einigen Geräten der Erfolg der Sicherung nicht von der Bedienbarkeit des zu sichernden Systems abhängig ist. Netcat hingegen erlaubt die Sicherung ohne vorherigen Schlüsselaustausch.

## 9.2 Bewertung der ADB-Live-Sicherung

ADB ist auf Ubuntu-Touch-Installationen des UBPorts-Projekts verfügbar. Hierbei wird ebenfalls eine TCP-Verbindung genutzt. Community-Ports des Systems beinhalten oft keine ADB-Installation. Auch diese Methode eignet sich für eingeschaltete Geräte mit bekannten Zugangsdaten. Auf kompatiblen Geräten ist eine Sicherung über ADB gegenüber der SSH-Sicherung zu bevorzugen, da der Schlüsselaustausch automatisiert bei der ersten Verbindungs-Herstellung erfolgt und nur über eine Abfrage auf dem Gerätebildschirm zu bestätigen ist. Gegenüber der Nutzung von ADB auf Android-Geräten sind Systemverwalter-Rechte unter Ubuntu Touch erheblich

einfacher zu erlangen. Bei Kenntnis des Nutzerpassworts genügt das Voranstellen von „sudo“ bei Befehlsaufrufen. Unter Android sind dazu erhebliche Eingriffe in das System nötig. Auch ADB ist geeignet, um erstellte Binaries für die Sicherung des RAM-Speichers auf das Gerät zu übertragen. Möglich sind Sicherungen des internen Flash-Speichers und der eingelegten µSD-Karte. Drei der sechs Testgeräte mit Ubuntu-Touch-Installation konnten über ADB gesichert werden.

### **9.3 Bewertung der Recovery-Sicherung**

Das Konzept der Recovery-Partition ist bei Geräten mit vormaliger Android-Installation zu finden. Oft wurde eine Custom-Recovery bereits für die Installation des Linux-Systems verwendet. Diese Recovery-Installationen können komfortabel für die Sicherung des Gerätespeichers genutzt werden. Da Nutzerdaten vorwiegend nicht in der Recovery zu finden sind, ist die Installation einer geeigneten Custom-Recovery eine Veränderung des Gerätespeichers, die im Rahmen der Datensicherung vertretbar ist. Veränderungen am Gerätespeicher sind jedoch zwingend zu dokumentieren. Diese Methode erlaubt auch die Sicherung von Geräten, zu denen bisher keine Zugangsdaten bekannt sind. Auch im Rahmen der Recovery-Sicherung kann neben dem internen auch der externe Flash-Speicher gesichert werden, sofern die eingelegte µSD-Karte von der Recovery erkannt wird. Eine Sicherung des Speichers gelang bei 18 der 20 Testgeräte, welche zuvor mit Android betrieben wurden. Acht Geräte konnten bei unbekannter Displaysperre nur über diese Methode gesichert werden. Es handelte sich um die Geräte: Google Nexus 5, Sony Xperia XA2, Motorola Moto G, Samsung Galaxy S5, Xiaomi Mi A2, Samsung Galaxy S9, HTC One und Xperia 5. Das Gerät Lenovo A6000 konnte zwar in eine Custom Recovery gestartet werden; diese erlaubte jedoch keinen Zugriff über die Workstation. Bei den vorliegenden Testgeräten bot Recovery-Sicherung ohne Kenntnis der Zugangsdaten des Geräts die größte Erfolgsaussicht.

## 9.4 Bewertung der Bootloader-Sicherung

Steht geeignete Software für das zu sichernde Gerät bereit, bietet die Bootloader-Sicherung eine forensisch sichere Methode der Datenakquise, bei der die Ausführung des installierten Betriebssystems nicht erforderlich ist und bei der keine Veränderungen am Gerätespeicher erfolgen. So ist diese Form der Sicherung für ausgeschaltete Geräte ohne bekannte Zugangsdaten geeignet. Zum Teil ergeben sich aus dem Bootloader-Zugriff auch Möglichkeiten der RAM-Sicherung bei eingeschalteten Geräten, die aus dem laufenden Betrieb heraus den Bootloader-Modus aufrufen können (siehe Seiten 48 bis 51). Limitierende Faktoren sind hier die bekannten Möglichkeiten zum Aufruf des jeweiligen Bootloader-Modus und die Verfügbarkeit von Software zur Kommunikation mit den Geräten in diesen Modi. Überwiegend wird über den Bootloader nur der Zugriff auf den internen Flash-Speicher gewährt. Über die Verwendung eines alternativen Bootloaders wie „das U-Boot“ kann jedoch auch der Zugriff auf eine eingelegte  $\mu$ SD-Karte erlangt werden. Anwendung findet die Methode bei Geräten mit ARM-Prozessoren. Von den 23 Testgeräten mit ARM-Chip konnten elf Geräte über den Bootloader gesichert werden. Dabei wurden zwei Geräte ohne Kenntnis des Sperrcodes nur über diese Methode gesichert.

## 9.5 Bewertung der Live-Linux-Sicherung

Forensische Live-Linux-Systeme sind bisher nur für x86-Systeme verfügbar. Dabei wird das System über ein externes Medium (meist USB-Stick) geladen und auf den internen Gerätespeicher erfolgt nur ein lesender Zugriff. Geeignet ist diese Methode für ausgeschaltete Geräte ohne bekannte Zugangsdaten. Neben dem internen Speicher können auch externe Speichergeräte gesichert werden, sofern diese nicht zum Start des Live-Systems zu entfernen waren. Beide Testgeräte mit Intel-x86-CPU konnten über das verwendete „Tsurugi-Linux“ physikalisch gesichert werden.

## 9.6 Bewertung der gerätespezifischen Sicherungen

Es konnten zwei Sicherungsmethoden aufgefunden werden, die keiner der vorangegangenen Ansätze eindeutig zuzuordnen waren.

Zunächst wurde der Versuch unternommen, den Gerätespeicher des Nokia N9 über ein „Rescue-Image“ als Massenspeicher freizugeben. Dieser Ansatz funktionierte nicht wie erwartet. Eine Freigabe des Speichers erfolgte nicht. Stattdessen startete das Gerät in das System ohne die zuvor eingerichtete Abfrage des Nutzerpassworts, so dass die Methode dennoch in Vorbereitung für eine Live-Sicherung verwendet werden kann. Nach dem erneuten Einspielen des zuvor installierten Systemimages war die Abfrage der Nutzerpin erneut aktiviert.

Weiterhin wurde mit „Jumpdrive“ eine Methode gefunden, um den internen Speicher von einigen Geräten mit hoher Mainline-Unterstützung als Massenspeicher freizugeben. Für das Pine64 Pinephone stellt dies eine komfortable Möglichkeit der Datensicherung dar. Bei vormalig mit Android betriebenen Geräten (Pocophone F1) wird lediglich die Userdata-Partition freigegeben. Hier erlauben die zuvor beschriebenen Ansätze eine umfassendere Datensicherung.

## 9.7 Bewertung der externen Flash-Speicher-Sicherung

Die drei Geräte mit Systeminstallationen auf externen Flash-Speichern (µSD-Karten) konnten erwartungsgemäß unter Entnahme des Speichers und Verwendung eines Write-Blockers gesichert werden. Dieses Vorgehen ist bei ausgeschalteten Geräten immer zu bevorzugen, da Veränderungen an den zu sichernden Daten wirksam verhindert werden. Für das Motorola Droid 4 wurde keine weitere Möglichkeit recherchiert, den Systemspeicher ohne Kenntnis des Sperrcodes sichern zu können.

## 10 Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit sollten geeignete Methoden der Datenextraktion aus mobilen Endgeräten mit Linux-Betriebssystemen ermittelt werden. Dazu wurden 25 Testgeräte vorbereitet und verschiedene mobile Linux-Systeme darauf installiert. Die Testgeräte wiesen zum Teil unterschiedliche Architekturen auf und wurden von verschiedenen Herstellern gefertigt. So konnte auf mögliche Unterschiede bei der Sicherung in Abhängigkeit vom Hersteller der Geräte und der Chipsätze eingegangen werden.

Als übliche Möglichkeit der Desktop-Linux-Sicherung wurde die Datenextraktion über SSH oder Netcat ermittelt. Diese konnte auch bei einem Großteil der Testgeräte angewendet werden.

Übliche Methoden der Android-Sicherung (wie die Verwendung von ADB, einer Custom-Recovery oder Bootloader-Angriffe), wurden ebenfalls bei verschiedenen Geräten nachvollzogen. Für viele Geräte konnte keine Möglichkeit des Aufrufs des Chipsatz-typischen Bootloader-Modus aufgefunden werden. Die Option des forensischen Einsatzes des APX-Modus bei Nvidia-Geräten und des FEL-Modus bei dem Linux-First-Gerät Pine64 Pinephone wurde beschrieben. Diese Methoden waren in einschlägiger Fachliteratur bisher nicht zu finden. Mit der Boot-Umgebung „Jumpdrive“ wurde eine weitere Möglichkeit der Sicherung des Pine64 Pinephone aufgezeigt.

Die externe Sicherung enthaltener µSD-Karten ist im Ablauf unabhängig vom verwendeten Betriebssystem und stellt auch bei mobilen Linux-Systemen eine geeignete Methode der Datensicherung dar.

Insgesamt wird der Erfolg der durchgeführten Sicherungen durch den Umstand begünstigt, dass keine der untersuchten Linux-Systeme von der möglichen Hardware-Verschlüsselung der Mobilgeräte Gebrauch macht. Für jedes Mobilgerät konnte mindestens eine Sicherung auch ohne Kenntnis des Gerätesperrcodes erfolgreich durchgeführt werden.



Mit LiME, AVML und den Bootloader-Methoden der vorliegenden Samsung- und MediaTek-Geräte konnten Möglichkeiten der RAM-Sicherung aufgezeigt werden. Diese wurden exemplarisch (ein Gerät pro Methode) demonstriert.

Die Datensicherungen zweier Mobilgeräte mit aktiver Full-Disk-Verschlüsselung konnten entschlüsselt werden. Dabei zeigte sich, dass der Erfolg der Entschlüsselung erheblich von der Komplexität der verwendeten Passphrase abhängt. Beide Geräte verwendeten die Linux-Verschlüsselung „LUKS“. Der dargestellte Angriff auf die LUKS1-Verschlüsselung des Mobilgeräts Sony Xperia XA2 mit SailfishOS bei der bisher üblichen Verwendung einer PIN als Passphrase führt in verhältnismäßig kurzer Zeit zum Erfolg. Die LUKS2-Verschlüsselung des Droidian-Systems auf dem Xperia 5 stellt hingegen einen wirksamen Schutz der Nutzerdaten dar. Eine Angriffsmethode wurde auch hier aufgezeigt. Diese bindet Zeit und Ressourcen allerdings in einem erheblich höherem Maße. Ohne Hinweise auf die Passphrase (z.B. über Social-Engineering-Angriffe) ist ein Bruteforce-Ansatz zur Entschlüsselung in einer relevanten Zeitspanne hier nicht wahrscheinlich.

Die dargestellten Methoden stellen nur eine Momentaufnahme dar. Es zeigt sich, dass geeignete Ansätze für die Datensicherung stets neu zu suchen sind und dass diese Recherche einen erheblichen Teil der forensischen Arbeit am Gerät ausmacht. Bereits im Zeitraum der Vorbereitung und Bearbeitung dieser Arbeit konnten relevante Veränderungen bei den installierten Systemen beobachtet werden. Beispielsweise wurde die Möglichkeit der Verschlüsselung für Droidian-Geräte erst im August 2022 integriert. Ein Versionssprung von Ubuntu Touch im Jahr 2023 ermöglicht die Verwendung längerer PIN-Codes. Ein aktuelles SailfishOS-Update sieht die Festlegung einer alphanumerischen Passphrase für die Verschlüsselung des Gerätespeichers vor.

Analysen der Besonderheiten und angepasster Software der Mobilsysteme können Inhalt weiterer Ausarbeitungen sein.

# Literaturverzeichnis

- [1] Locker, Theresa; Vice.com - Wie deutsche Ermittler beschlagnahmte Smartphones knacken; 2017; <https://www.vice.com/de/article/8qmxqx/wie-deutsche-ermittler-beschlagnahmte-smartphones-knacken>; Aufruf: 27.01.2023
- [2] Openos.at; Das Open Source Glossar - BLOB; 2023; <http://www.openos.at/pages/basics/glossary.php?terms=b#blob>; Aufruf: 05.06.2023
- [3] the Free Software Foundation; Coreutils - GNU core utilities; 2016; <https://www.gnu.org/software/coreutils/>; Aufruf: 06.03.2023
- [4] Grüner, Sebastian; Glibc statt Bionic auf Android-Hardware; 2012; <https://www.golem.de/news/libhybris-glibc-statt-bionic-auf-android-hardware-1209-94797.html>; Aufruf: 21.02.2023
- [5] Keim, Y.; Yoon, Y.H.; Karabiyik, U. ; Digital Forensics Analysis of Ubuntu Touch on PinePhone; 2021; <https://www.mdpi.com/2079-9292/10/3/343>;
- [6] Tzvetanov, Krassimir; Karabiyik, Umit; A First Look at Forensic Analysis of sailfishos; 2020; [https://docs.lib.purdue.edu/cit\\_articles/51](https://docs.lib.purdue.edu/cit_articles/51); Aufruf: 09.02.2023
- [7] NetMarketShare; Marktanteile der führenden Betriebssysteme weltweit im Januar 2023; 2023; <https://de.statista.com/statistik/daten/studie/828610/umfrage/marktanteile-der-fuehrenden-betriebssystemversionen-weltweit/?locale=de>; Aufruf: 18.02.2023
- [8] StatCounter; Marktanteile der führenden Betriebssysteme in Deutschland von Januar 2009 bis Januar 2023; 2023; <https://de.statista.com/statistik/daten/studie/158102/umfrage/marktanteile-von-betriebssystemen-in-deutschland-seit-2009/?locale=de>; Aufruf: 05.04.2023
- [9] Bube, Lars; Abhörsichere Smartphones für Drogenbosse; 2018; <https://www.ict-channel.com/telekommunikation/chef-von-phantom-secure-verhaftet.116536.html>; Aufruf: 28.02.2023
- [10] Giovanni; The 2020 J-D analytics report; 2020; <https://jolla-devices.com/news/2020-analytics-report/>; Aufruf: 28.02.2023
- [11] Stahie, Silviu; Ubuntu Community Developer Estimates Number of Ubuntu Phones at 25K; 2015; <https://news.softpedia.com/news/ubuntu-community-developer-estimates-number-of-ubuntu-phones-at-25k-492897.shtml>; Aufruf: 28.02.2023
- [12] Pine64; PinePhone community poll results; 2022; <https://www.pine64.org/2022/01/31/pinephone-community-poll-results/>; Aufruf: 28.02.2023
- [13] Sneddon, Joey; Purism Librem 5 Crowdfunding Ends With Over \$2 million Raised; 2020; <https://www.omgubuntu.co.uk/2017/10/purism-librem-5-crowdfunding-ends-raised-2-million>; Aufruf: 28.02.2023
- [14] Epifanova; Alena, Dietrich, Philipp ; Russia's Quest for Digital Sovereignty; 2022; <https://dgap.org/en/research/publications/russias-quest-digital-sovereignty>; Aufruf: 03.02.2023
- [15] Kantar; Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in Deutschland im 3. Quartal der Jahre 2021 und 2022; 2022; <https://de.statista.com/statistik/daten/studie/198435/umfrage/marktanteile-der-smartphone-betriebssysteme-am-absatz-in-deutschland/>; Aufruf: 01.02.2022
- [16] Ihlenfeld, Jens; Maemo - Linux-Plattform für mobile Endgeräte von Nokia; 2005; <https://www.golem.de/0505/38247.html>; Aufruf: 01.02.2023
- [17] Kluczniok, Jan; Nokia N9 im Test: Das MeeGo-Experiment; 2011; <https://www.netzwelt.de/news/89709-mee-go-experiment-nokia-n9-test.html>; Aufruf: 01.02.2023
- [18] Biermann, Kai; Nokia und Microsoft verbünden sich; 2011; <https://www.zeit.de/digital/mobil/2011-02/nokia-microsoft-wp7>; Aufruf: 01.02.2023
- [19] Wedekind, Klaus; Ex-Nokia-Leute planen N9-Nachfolger - MeeGo lebt als Jolla weiter; 2012; <https://www.n-tv.de/technik/MeeGo-lebt-als-Jolla-weiter-article6681721.html>; Aufruf: 01.02.2023
- [20] Färber, Nils; Aus den Ruinen von Maemo und MeeGo entstanden Jolla und Sailfish OS; 2014; <https://www.linux-magazin.de/ausgaben/2014/05/jolla-und-sailfish/>; Aufruf: 02.02.2023

- 
- [21] Jolla Ltd.; Cases; 2023; <https://sailfishos.org/cases/>; Aufruf: 03.02.2023
  - [22] Ordulj, Marin; Aurora OS is based on Sailfish but owned by OMP; 2021; <https://nokiamob.net/2021/11/16/aurora-os-is-based-on-sailfish-but-owned-by-omp/>; Aufruf: 03.02.2023
  - [23] Sneddon, Joey; Ubuntu Phone OS Unveiled by Canonical; 2013; <https://www.omgubuntu.co.uk/2013/01/ubuntu-phone-os-unveiled-by-canonical>; Aufruf: 03.02.2023
  - [24] Filippi, Angelica; Erste Ubuntu-Smartphones von BQ und Meizu; 2014; [https://www.itmagazine.ch/artikel/55325/Erste\\_Ubuntu-Smartphones\\_von\\_BQ\\_und\\_Meizu.html](https://www.itmagazine.ch/artikel/55325/Erste_Ubuntu-Smartphones_von_BQ_und_Meizu.html); Aufruf: 03.02.2023
  - [25] Donauer, Jürgen; Unity 8, Konvergenz und Ubuntu Phone eingestampft – Ubuntu 18.04 geht zurück zu GNOME; 2017; <https://www.bitblokes.de/unity-8-konvergenz-und-ubuntu-phone-eingestampft-ubuntu-18-04-geht-zurueck-zu-gnome/>; Aufruf: 04.02.2023
  - [26] Thommes, Ferdinand; Ubuntu: UBports will Ubuntu Touch am Leben erhalten; 2017; <https://www.computerbase.de/2017-04/ubuntu-ubports-ubuntu-touch/>; Aufruf: 04.02.2023
  - [27] Thommes, Ferdinand; Das Jahr der Linux-Phones; 2020; <https://www.linux-community.de/ausgaben/linuxuser/2020/05/librem-5-und-pinephone/>; Aufruf: 04.02.2023
  - [28] Günther, Guido; Phosh Overview; 2020; <https://puri.sm/posts/phosh-overview/>; Aufruf: 04.02.2023
  - [29] Pine64; PinePhone Software Releases; 2023; [https://wiki.pine64.org/wiki/PinePhone\\_Software\\_Releases](https://wiki.pine64.org/wiki/PinePhone_Software_Releases); Aufruf: 04.02.2023
  - [30] PostmarketOS; Aiming for a 10 year life-cycle for smartphones; 2017; <https://postmarketos.org/blog/2017/05/26/intro/>; Aufruf: 04.02.2023
  - [31] PostmarketOS; Devices; 2023; <https://wiki.postmarketos.org/wiki/Devices>; Aufruf: 04.02.2023
  - [32] Halium; Halium Project; 2023; <https://halium.org/>; Aufruf: 04.02.2023
  - [33] Droidian; hybris-mobian Is Now Droidian; 2021; <https://droidian.org/blog/hybris-mobian-is-now-droidian/>; Aufruf: 04.02.2023
  - [34] Corelium; Projecz Sandcastle - What's Currently supported; 2022; <https://projectsandcastle.org/status>; Aufruf: 06.02.2023
  - [35] Larabel, Michael; Fedora 38 Cleared To Produce "Mobility Phosh" Spins; 2022; <https://www.phoronix.com/news/Fedora-Mobility-Phosh-Approved>; Aufruf: 06.02.2023
  - [36] Neumayer, Alfred E.; Planning for snapd support on Ubuntu Touch; 2023; <https://forum.snapcraft.io/t/planning-for-snapd-support-on-ubuntu-touch/33560>; Aufruf: 08.02.2023
  - [37] Croome, Chris; This time, I have tried to set out the basic privacy and safety features of Ubuntu Touch!; 2018; <https://ubports.com/de/blog/ubports-blogs-nachrichten-1/post/ubuntu-touch-safety-architecture-25>; Aufruf: 08.02.2023
  - [38] Karczewski, Filip; Proof of Concept: Privilege escalation in Ubuntu Touch 16.04 - by Passcode Brute-force; 2022; <https://github.com/filipkarc/PoC-ubuntutouch-pin-privesc>; Aufruf: 08.02.2022
  - [39] Ubports; Find the Linux phone that suits you; 2023; <https://devices.ubuntu-touch.io/>; Aufruf: 08.02.2023
  - [40] Jolla; SailfishOS - Architecture; 2023; <https://docs.sailfishos.org/Reference/Architecture/>; Aufruf: 09.02.2023
  - [41] Jolla; Encryption of User Data; 2023; [https://docs.sailfishos.org/Support/Help\\_Articles/Encryption\\_of\\_User\\_Data/#introduction-to-encryption-in-sailfish-os](https://docs.sailfishos.org/Support/Help_Articles/Encryption_of_User_Data/#introduction-to-encryption-in-sailfish-os); Aufruf: 09.02.2023
  - [42] Thommes, Ferdinand; Mobiler Alpine-Linux-Fork PostmarketOS; 2021; <https://www.linux-magazin.de/ausgaben/2021/10/postmarketos/>; Aufruf: 13.02.2023
  - [43] Mobian Team; What is Mobian?; 2021; [https://blog.mobian-project.org/posts/2021/02/18/what\\_is\\_mobian/](https://blog.mobian-project.org/posts/2021/02/18/what_is_mobian/); Aufruf: 13.02.2023
  - [44] Pejaković, Mladen; PureOS Frequently Asked Questions FAQ; 2021; <https://tracker.pureos.net/w/faq/>; Aufruf: 13.02.2023
  - [45] Projekt Kupfer; Kupfer Linux - What is Kupfer?; 2022; <https://kupfer.gitlab.io/>; Aufruf: 13.02.2023

- 
- [46] Larabel, Michael; KDE Plasma Mobile Drops Halium Support In Favor Of Open Devices; 2020; <https://www.phoronix.com/news/Plasma-Mobile-Drops-Halium>; Aufruf: 13.02.2023
- [47] Sxmo Contributors; Sxmo: Simple X Mobile; 2022; <https://sxmo.org/>; Aufruf: 13.02.2023
- [48] Dreßler, Jonas; GNOME Shell on mobile: An update; 2022; <https://blogs.gnome.org/shell-dev/2022/09/09/gnome-shell-on-mobile-an-update/>; Aufruf: 13.02.2023
- [49] Mlích, Jozef; Nemomobile in April 2022; 2022; <https://nemomobile.net/pages/nemomobile-in-april-2022/>; Aufruf: 14.02.2023
- [50] Cutie Shell; Cutie Shell; 2022; <https://cutie-shell.org/>; Aufruf: 14.02.2023
- [51] Linder, Brad; Pangolin Mobile is a new open source user interface for smartphones; 2021; <https://liliputing.com/pangolin-mobile-is-a-new-open-source-user-interface-for-smartphones/>; Aufruf: 14.02.2023
- [52] Maemo Leste; Maemo Leste; 2023; <https://maemo-leste.github.io/>; Aufruf: 14.02.2023
- [53] PostmarketOS; Lomiri - The convergent desktop environment. ; 2021; <https://wiki.postmarketos.org/wiki/Lomiri>; Aufruf: 14.02.2023
- [54] Archivista.ch; Darum gibt es das Archivista-Telefon in zweiter Generation; 2022; <https://archivista.ch/cms/de/aktuell-blog/blogs-2022/avmultiphone-v2/>; Aufruf: 15.02.2023
- [55] julianwi; fbkeyboard; 2017; <https://github.com/julianwi/fbkeyboard>;
- [56] Droidian; Droidian; ; <https://github.com/droidian-images/droidian>; Aufruf: 15.02.2023
- [57] Maemo Leste; Getting Started; 2023; [https://leste.maemo.org/Getting\\_Started](https://leste.maemo.org/Getting_Started); Aufruf: 15.02.2023
- [58] Khurram Ali; Tutorial Nokia N9: Files transfer via WiFi / SSH and Terminal Root Access; 2011; <https://techprolonged.com/2011/10/tutorial-nokia-n9-files-transfer-via-wifi-ssh/>; Aufruf: 15.02.2023
- [59] Open Suse; HCL:VollaPhone; 2023; <https://en.opensuse.org/HCL:VollaPhone>; Aufruf: 23.02.2023
- [60] Adel Nouredine; Sailfish OS Tips & Tricks; 2022; <https://www.nouredine.org/articles/sailfish-os-tips-tricks>; Aufruf: 15.02.2023
- [61] PostmarketOS; Installation; 2022; <https://wiki.postmarketos.org/wiki/Installation>; Aufruf: 15.02.2023
- [62] Sturmflut's Blog; Hacking Ubuntu Touch, Part 5: adb shell vs. phablet-shell; 2015; <https://sturmflut.github.io/ubuntu/touch/2015/05/08/hacking-ubuntu-touch-part-5-adb-shell-vs-phablet-shell/>; Aufruf: 15.02.2023
- [63] Maemo Leste Wiki; Maemo Leste - Motorola Droid 4; 2022; [https://leste.maemo.org/Motorola\\_Droid\\_4](https://leste.maemo.org/Motorola_Droid_4); Aufruf: 07.09.2022
- [64] Bundesamt für Sicherheit in der Informationstechnik; Leitfaden „IT-Forensik“; 2011; [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden\\_IT-Forensik.pdf?\\_\\_blob=publicationFile&v=1](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf?__blob=publicationFile&v=1);
- [65] Muth, Denise; Leitfaden zur forensischen Untersuchung von Android-Smartphones; 2013; Masterarbeit - Hochschule Darmstadt
- [66] NIST; Guidelines on Mobile Device Forensics; 2014; <https://www.nist.gov/publications/guidelines-mobile-device-forensics>;
- [67] Geschonneck, Alexander; Computer-Forensik : Systemeinbrüche erkennen, ermitteln, aufklären ; 2006; dpunkt-Verlag; 978-3-89864-379-5
- [68] J. Aviv, Adam et al.; Smudge Attacks on Smartphone Touch Screens; 2010; [https://www.researchgate.net/publication/228644465\\_Smudge\\_Attacks\\_on\\_Smartphone\\_Touch\\_Screens](https://www.researchgate.net/publication/228644465_Smudge_Attacks_on_Smartphone_Touch_Screens);
- [69] Viennot, Nicolas; Instant terminal sharing; 2023; <https://tmate.io/>; Aufruf: 23.02.2023
- [70] Heinson, Dennis; IT-Forensik; 2015; Mohr Siebeck Tübingen; 978-3-16-153701-1
- [71] Lin, Xiaodong; Introductory Computer Forensics - A Hands-on Practical Approach; 2018; Springer Nature Switzerland; 978-3-030-00580-1
- [72] Nikkel, Bruce; Practical Linux Forensics - A Guide For Digital Investigators; 2021; William Pollock; 978-1-7185-0196-6
- [73] Yang, Seung Jei et al.; New acquisition method based on firmware update protocols for Android smartphoness; 2015; Elsevier Ltd.
- [74] Dasgupta, Rhythm Kr; Mobile Forensic: Investigation of Dead or Damage Smart Phone - An Overview, Tools & Technique Challenges from Law Enforcement Perspective.; 2020;
- [75] Tahiri, Soufiane; Mastering Mobile Forensics; 2016; Packt Publishing Ltd.; ISBN 978-1-78528-781-7

- 
- [76] Mahalik, H. et al.; Practical Mobile Forensics (Second Edition); 2016; Packt Publishing Ltd.; 978-1-78646-420-0
- [77] Kalsi, Tajinder; Practical Linux Security Cookbook ; 2018; Packt Publishing Ltd.; 978-1-78913-839-9
- [78] Lwin, Htar Htar et al.; Comparative Analysis of Android Mobile Forensics Tools; 2020; IEEE
- [79] Trinko, Miroslav; Speicher ist nie genug: ZTE hat das weltweit erste Smartphone mit 18 GB RAM und 1 TB Speicher vorgestellt ; 2021; <https://gagadget.com/de/91101-speicher-ist-nie-genug-zte-hat-das-weltweit-erste-smartphone-mit-18-gb-ram-und-1-tb-speicher-vorgestellt/>; Aufruf: 20.03.2023
- [80] Labudde, Dirk; Forensik in der digitalen Welt; 2016; University of Applied Sciences Mittweida; 978-3-662-53800-5
- [81] Dolle, Wilhelm; Computer-Forensik in der Praxis; 2009; Springer Gabler
- [82] Metz, Joachim; libewf is a library to access the Expert Witness Compression Format (EWF); 2023; <https://github.com/libyal/libewf>; Aufruf: 21.03.2023
- [83] Karayianni, Stavroula; Katos, Vasilios; Global Security, Safety and Sustainability & e-Democracy - Practical Password Harvesting from Volatile Memory; 2011; Springer, Berlin, Heidelberg; 978-3-642-33447-4
- [84] Sylve, Joe; LiME ~ Linux Memory Extractor; 2022; <https://github.com/504ensicsLabs/LiME>; Aufruf: 22.03.2023
- [85] Danct12; Arch Linux ARM on Pine64's PineTab; 2019; <https://danct12.github.io/PineTab-Arch-Linux-ARM/>; Aufruf: 23.03.2023
- [86] Microsoft; AVML (Acquire Volatile Memory for Linux); 2023; <https://github.com/microsoft/avml>; Aufruf: 23.03.2023
- [87] kingground, bmc-msft; Any plan to support ARM / ARM64 architecture?; 2021; <https://github.com/microsoft/avml/issues/28>; Aufruf: 23.03.2023
- [88] Moreb, Mohammed; Practical Forensic Analysis of Artifacts on iOS and Android Devices; 2022; Apress Media LLC; 978-1-4842-8025-6
- [89] Kerler, Björn; SUC - Samsung Upload Client v1.0; 2023; [https://github.com/bkerler/sboot\\_dump](https://github.com/bkerler/sboot_dump); Aufruf: 23.03.2023
- [90] MSAB; Capturing RAM in Android systems simplified; 2022; <https://www.msab.com/blog/capturing-ram-in-android-systems-simplified/>; Aufruf: 23.03.2023
- [91] Hardreset.info; Upload-Modus SAMSUNG Galaxy S10 SD855; 2023; <https://www.hardreset.info/de/devices/samsung/samsung-galaxy-s10-sd855/upload-modus/>; Aufruf: 23.03.2023
- [92] MSAB; MediaTek Chipsets Guide: How to Use XRY to Extract Data from MTK-Based Devices ; 2023; <https://www.msab.com/de/blog/mediatek-chipsets-xry-extraction/>; Aufruf: 24.03.2023
- [93] Kerler, Björn; MTKClient; 2023; <https://github.com/bkerler/mtkclient>; Aufruf: 24.03.2023
- [94] Gromanowski, Lukasz; Fastboot oem commands in MotoG (3rd ed.); 2016; <https://gromanowski.net/2016/07/18/fastboot-oem-commands-in-motog-3rd-gen/>; Aufruf: 27.03.2023
- [95] Kerler, Björn; Qualcomm Sahara / Firehose Attack Client / Diag Tools; 2023; <https://github.com/bkerler/edl>; Aufruf: 27.03.2023
- [96] Spring, Tom; Android Patch Fixes Nexus 5X Critical Vulnerability; 2016; <https://threatpost.com/android-patch-fixes-nexus-5x-critical-vulnerability/120346/>; Aufruf: 27.03.2023
- [97] Busstra, Bert; Android and Wireless data-extraction using Wi-Fi; 2014; IEEE
- [98] PostmarketOS; USB Network; 2021; [https://wiki.postmarketos.org/wiki/USB\\_Network](https://wiki.postmarketos.org/wiki/USB_Network); Aufruf: 20.04.2023
- [99] Halium Documentation Contributors; Halium Documentation; 2023; <https://readthedocs.org/projects/halium-docs/downloads/pdf/latest/>; Aufruf: 20.04.2023
- [100] Leppänen, Tomi; USB\_MODED use and API description/overview; 2020; [https://github.com/sailfishos/usb-moded/blob/master/docs/usb\\_moded-doc.txt](https://github.com/sailfishos/usb-moded/blob/master/docs/usb_moded-doc.txt); Aufruf: 20.04.2023
- [101] Maemo Leste; Status/USB Peripheral; 2021; [https://leste.maemo.org/Status/USB\\_Peripheral](https://leste.maemo.org/Status/USB_Peripheral); Aufruf: 20.04.2023



- 
- [102] Luber, Stefan; Was ist Secure Shell (SSH / SSH-1 / SSH-2 / OpenSSH)?; 2018; <https://www.ip-insider.de/was-ist-secure-shell-ssh-ssh-1-ssh-2-openssh-a-691217/>; Aufruf: 20.04.2023
- [103] SANS Institute; GIAC Certified Forensic Analyst Practical Assignment Analysis of a floppy disk and use of SSH as a forensic tool ; 2005; SANS Institute
- [104] Metz, Joachim; ewfacquirestream(1) - Linux man page; 2009; <https://linux.die.net/man/1/ewfacquirestream>; Aufruf: 25.04.2023
- [105] UBports Docs; Shell access via SSH; 2022; <https://docs.ubports.com/de/latest/userguide/advanceduse/ssh.html>; Aufruf: 26.04.2023
- [106] Sapegin, Andrey; Answer to "How can I access my Ubuntu phone over ssh?"; 2016; <https://askubuntu.com/a/804166>; Aufruf: 26.04.2023
- [107] Matotek, D. et al.; Pro Linux System Administration: Learn to Build Systems for Your Business Using Free and Open Source Software; 2017; Apress; 978-1-4842-2007-8
- [108] Fischer, Werner; Device Mapper Targets; 2023; [https://www.thomas-krenn.com/de/wiki/Device\\_Mapper\\_Targets](https://www.thomas-krenn.com/de/wiki/Device_Mapper_Targets); Aufruf: 27.04.2023
- [109] Mobian Wiki; Networking HowTos; 2022; <https://wiki.mobian-project.org/doku.php?id=howto:networking#connect-to-a-running-ssh-server>; Aufruf: 12.05.2023
- [110] UBports; Shell access via ADB; 2023; <https://docs.ubports.com/de/latest/userguide/advanceduse/adb.html>; Aufruf: 03.05.2023
- [111] Vidas, Timothy et al.; The Proceedings of the Eleventh Annual DFRWS Conference - Toward a general collection methodology for Android devices; 2011; Elsevier
- [112] MobileEdit; Android recovery data acquisition; 2020; <https://support.mobileedit.com/portal/en/kb/articles/twrp-android-recovery-data-acquisition>; Aufruf: 17.05.2023
- [113] Team Win Recovery Project; About; 2022; <https://twrp.me/about/>; Aufruf: 17.05.2023
- [114] AOSP; A/B (Seamless) System Updates; 2022; <https://source.android.com/docs/core/ota/ab>; Aufruf: 22.05.2023
- [115] XDA Staff; Google mandates virtual A/B system updates for devices launching with Android 13; 2022; <https://www.xda-developers.com/google-mandate-virtual-a-b-system-updates-android-13/>; Aufruf: 22.05.2023
- [116] Fäcknitz, Felix; Galaxy S23: Samsung ignoriert wichtiges Android-Feature schon wieder; 2023; <https://www.netzwelt.de/news/213567-galaxy-s23-samsung-ignoriert-wichtiges-android-feature.html>; Aufruf: 22.05.2023
- [117] Sjll; [ROM][UNOFFICIAL] LineageOS 18.1 for Xperia 5 [EOL]; 2021; <https://forum.xda-developers.com/t/rom-unofficial-lineageos-18-1-for-xperia-5-eol.4350091/>; Aufruf: 22.05.2023
- [118] Mohammed, Abubakar; What Is UFS Storage? How Is It Better Than eMMC?; 2022; <https://fossbytes.com/what-is-ufs-storage/>; Aufruf: 22.05.2023
- [119] Ballein, Maurice; Android: So schaltet ihr den Recovery- und Download-Modus ein; 2019; <https://www.netzwelt.de/anleitung/158481-android-so-schaltet-recovery-download-modus.html>; Aufruf: 22.05.2023
- [120] Betts, Andy; What Is Samsung's Odin Firmware Flashing Software?; 2022; <https://www.makeuseof.com/samsung-odin-overview/>; Aufruf: 22.05.2023
- [121] PostmarketOS; Motorola Droid 4 (motorola-maserati); 2023; [https://wiki.postmarketos.org/index.php?title=Motorola\\_Droid\\_4\\_\(motorola-maserati\)](https://wiki.postmarketos.org/index.php?title=Motorola_Droid_4_(motorola-maserati)); Aufruf: 24.05.2023
- [122] Hay, Roe; Hadad, Noam; Exploiting Qualcomm EDL Programmers (1): Gaining Access & PBL Internals; 2018; <https://alephsecurity.com/2018/01/22/qualcomm-edl-1/>; Aufruf: 26.05.2023
- [123] Privacy International; A technical look at Phone Extraction; 2019; <https://privacyinternational.org/long-read/3256/technical-look-phone-extraction>; Aufruf: 26.05.2023
- [124] ISP Pinout; Redmi Note 8T EDL Mode Test Point ; 2021; <https://www.isppinout.com/2021/12/redmi-note-8t-edl-mode-test-point.html>; Aufruf: 27.05.2023
- [125] Kong, Joe; Data Extraction on MTK-based Android Mobile Phone Forensics; 2015; Purdue University

- 
- [126] OxygenForensics; Support of MediaTek devices in Oxygen Forensic® Detective; 2022; <https://www.forensicfocus.com/news/support-of-mediatek-devices-in-oxygen-forensic-detective/>; Aufruf: 27.05.2023
  - [127] VD171; [PRELOADER][IMG][STOCK] PRELOADER Partition Binary for DANDELION (Redmi 9A / Redmi 9AT); 2021; <https://forum.xda-developers.com/t/preloader-img-stock-preloader-partition-binary-for-dandelion-redmi-9a-redmi-9at.4330759/>; Aufruf: 28.05.2023
  - [128] XDA Developers; APX mode; 2014; [https://web.archive.org/web/20190527223958/https://forum.xda-developers.com/wiki/APX\\_mode](https://web.archive.org/web/20190527223958/https://forum.xda-developers.com/wiki/APX_mode); Aufruf: 30.05.2023; Archiviert am 27.05.2019
  - [129] Open Surface RT; Fusée Gelée; 2022; <https://openrt.gitbook.io/open-surfacert/common/boot-sequence/tegra-soc-boot-process/fusee-gelee>; Aufruf: 30.05.2023
  - [130] Söldner, Michael; Fusée Gelée: Nintendo Switch gehackt; 2018; <https://www.pcwelt.de/article/1170192/fusee-gelee-nintendo-switch-gehackt.html>; Aufruf: 30.05.2023
  - [131] tobdaryl; [GUIDE] How to enter APX mode or Bootloader menu; 2012; <https://forum.xda-developers.com/t/guide-how-to-enter-apx-mode-or-bootloader-menu.2034866/>; Aufruf: 30.05.2023
  - [132] KC McCoy; Tegra3 Guide: nvflash NEXUS 7 (and Transformer Jellybeans) ; 2020; <http://www.readynests.com/tegra3-guide-nvflash-nexus-7-and-transformer-jellybeans/>; Aufruf: 30.05.2023
  - [133] fxsheep; [TUTORIAL] Generate blobs for a bricked Nexus 7 without another N7 or Tegra30 device; 2021; <https://forum.xda-developers.com/t/tutorial-generate-blobs-for-a-bricked-nexus-7-without-another-n7-or-tegra30-device.4305955/>; Aufruf: 30.05.2023
  - [134] GeorgeMato4; nvcrypttools; 2020; <https://github.com/GeorgeMato4/nvcrypttools/tree/forN7>; Aufruf: 30.05.2023
  - [135] PostmarketOS; ASUS Transformer Pad (asus-tf300t); 2022; [https://wiki.postmarketos.org/wiki/ASUS\\_Transformer\\_Pad\\_\(asus-tf300t\)](https://wiki.postmarketos.org/wiki/ASUS_Transformer_Pad_(asus-tf300t));
  - [136] Open RT Community; Open Surface RT; 2022; <https://openrt.gitbook.io/open-surfacert/>; Aufruf: 31.01.2023
  - [137] mineman; even better 4 step install; 2022; <https://discord.com/channels/710026735294349322/710044936862564384/965771898548027452>;
  - [138] Open Surface RT; fusee tools; 2023; <https://github.com/Open-Surface-RT/fusee-tools>; Aufruf: 31.05.2023
  - [139] Open Surface RT; u-boot (microsoft-surface-2); 2023; <https://github.com/Open-Surface-RT/u-boot/tree/microsoft-surface-2>; Aufruf: 31.05.2023
  - [140] Iqbal, Asif et al.; Windows Surface RT tablet forensics; 2014; Elsevier
  - [141] Linux Sunxi; Linux mainlining effort; 2023; [https://linux-sunxi.org/Linux\\_mainlining\\_effort](https://linux-sunxi.org/Linux_mainlining_effort); Aufruf: 01.06.2023
  - [142] The U-Boot development community; Allwinner SoC based boards - Booting via the USB(-OTG) FEL mode; 2021; <https://u-boot.readthedocs.io/en/latest/board/allwinner/sunxi.html#booting-via-the-usb-otg-fel-mode>;
  - [143] Mery, Alejandro; sunxi-tools; 2023; <https://github.com/linux-sunxi/sunxi-tools>; Aufruf: 01.06.2023
  - [144] Linux Sunxi; FEL - Triggering FEL Mode; 2021; <https://linux-sunxi.org/FEL>; Aufruf: 01.06.2023
  - [145] Linux Sunxi; PinePhone - FEL Mode; 2022; [https://linux-sunxi.org/PinePhone#FEL\\_mode](https://linux-sunxi.org/PinePhone#FEL_mode); Aufruf: 01.06.2023
  - [146] zetabeta, kqlnut; what are these pins above modem chip?; 2021; <https://forum.pine64.org/showthread.php?tid=15541>; Aufruf: 01.06.2023
  - [147] Trzciński, Kamil; boot-tools; 2018; <https://github.com/ayufan-pine64/boot-tools>; Aufruf: 01.06.2023
  - [148] Reddy, Niranjani; Practical Cyber Forensics - Linux Distributions Used for Forensic Analysis; 2019; Apress; 978-1-4842-4460-9
  - [149] Tsurugi-Linux; Tsurugi Acquire; 2023; [https://tsurugi-linux.org/tsurugi\\_acquire.php](https://tsurugi-linux.org/tsurugi_acquire.php); Aufruf: 04.06.2023

- 
- [150] Kaps, Reik; Linux auf Intel Atom Tablet (z3735g); 2017; <https://www.leinelab.org/projekt:atom-tablet-linux:start>; Aufruf: 04.06.2023
  - [151] dreemurrs-embedded; Jumpdrive; 2022; <https://github.com/dreemurrs-embedded/Jumpdrive>; Aufruf: 06.06.2023
  - [152] Pine64; PinePhone Installation Instructions; ; [https://wiki.pine64.org/index.php/PinePhone\\_Installation\\_Instructions](https://wiki.pine64.org/index.php/PinePhone_Installation_Instructions); Aufruf: 13.09.2022
  - [153] Brunner, Tobias; Access a locked Nokia N9 MeeGo phone; 2013; <https://tobru.ch/access-a-locked-nokia-n9-mee-go-phone/>; Aufruf: 06.06.2023
  - [154] jorvikku; [Release notes] Struven ketju 4.5.0.16; 2023; <https://forum.sailfishos.org/t/release-notes-struven-ketju-4-5-0-16/14290>; Aufruf: 08.06.2023
  - [155] Paolantonio, Eugenio; droidian-encryption-service; 2022; <https://github.com/droidian/droidian-encryption-service>; Aufruf: 08.06.2023
  - [156] Wagner, Arno; cryptsetup; 2023; <https://gitlab.com/cryptsetup/cryptsetup>; Aufruf: 08.06.2023
  - [157] Mönchmeyer, Ralph; dm-crypt/Luks – Begriffe, Funktionsweise und die Rolle des Hash-Verfahrens – I; 2018; <https://linux-blog.anracom.com/2018/11/13/dm-crypt-luks-begriffe-funktionsweise-und-die-rolle-des-hash-verfahrens-i/>; Aufruf: 08.06.2023
  - [158] Scherf, Torsten; Reisesicherheit für Daten; 2015; <https://www.admin-magazin.de/Das-Heft/2015/01/Workshop-Open-Source-Tipp>; Aufruf: 08.06.2023
  - [159] Bell, Patrick; Bruteforcing Linux Full Disk Encryption (LUKS) With Hashcat; 2018; <https://www.forensicfocus.com/articles/bruteforcing-linux-full-disk-encryption-luks-with-hashcat/>; Aufruf: 08.06.2023
  - [160] jabrown; Cracking Linux Disk Encryption LUKS; 2020; <https://penguin-systems.com/2020/10/cracking-luks/>; Aufruf: 08.06.2023
  - [161] Metz, Joachim; LVM format specification; 2020; [https://github.com/libyal/libvslvm/blob/main/documentation/Logical%20Volume%20Manager%20\(LVM\)%20format.asciidoc](https://github.com/libyal/libvslvm/blob/main/documentation/Logical%20Volume%20Manager%20(LVM)%20format.asciidoc); Aufruf: 09.06.2023
  - [162] glv2; bruteforce-luks; 2019; <https://github.com/glv2/bruteforce-luks>; Aufruf: 10.06.2023
  - [163] Motorola Mobility LLC.; Unlocking the Bootloader; 2022; <https://motorola-global-portal.custhelp.com/app/standalone/bootloader/unlock-your-device-a>; Aufruf: 13.07.2022
  - [164] Mister\_Magister; SailfishOS for Moto G 2014 by VerdandiTeam; 2021; <https://forum.xda-developers.com/t/sailfish-os.3204245/>; Aufruf: 13.07.2022
  - [165] The UBports project; Install Ubuntu Touch; 2022; <https://docs.ubports.com/en/latest/userguide/install.html>; Aufruf: 13.07.2022
  - [166] Software in the Public Interest; Unofficial non-free images including firmware packages; 2022; <https://cdimage.debian.org/cdimage/unofficial/non-free/cd-including-firmware/>; Aufruf: 04.07.2022
  - [167] Mobian; Squashing a Book Full of Worms; 2021; <https://blog.mobian.org/posts/2021/12/24/bullseye-to-bookworm/>; Aufruf: 13.07.2022
  - [168] PostmarketOS; Lenovo A6000 (lenovo-a6000); 2022; [https://wiki.postmarketos.org/wiki/Lenovo\\_A6000\\_\(lenovo-a6000\)](https://wiki.postmarketos.org/wiki/Lenovo_A6000_(lenovo-a6000)); Aufruf: 13.07.2021
  - [169] postmarketos-mainline; msm8916-mainline / lk2nd; ; <https://github.com/msm8916-mainline/%20lk2nd>; Aufruf: 13.07.2022
  - [170] Jeschke, Malte; Nokia - Support für Symbian und MeeGo endet 2013; 2013; <https://www.tecchannel.de/a/support-fuer-symbian-und-mee-go-endet-2013,2046555>; Aufruf: 19.07.2022
  - [171] OpenRepos; N9 RepoMirror; 2015; <https://openrepos.net/content/ancelad/n9-repomirror>; Aufruf: 19.07.2022
  - [172] Jolla; Installing Sailfish X on XA2 using Linux; 2022; <https://jolla.com/sailfishx-linux-instructions-xa2/>; Aufruf: 29.01.2022
  - [173] Sony; Unlock Bootloader; 2022; <https://developer.sony.com/develop/open-devices/get-started/unlock-bootloader/>; Aufruf: 29.07.2022
  - [174] Sony; Software binaries for AOSP Oreo (Android 8.1) – Kernel 4.4 – Nile (v16); 2022; <https://developer.sony.com/file/download/software-binaries-for-aosp-oreo-android-8-1-kernel-4-4-nile-v16/>; Aufruf: 29.07.2022
  - [175] Maemo Leste; Maemo Leste Images - Droid 4; 2022; <https://maedevu.maemo.org/images/droid4/>; Aufruf: 06.09.2022



- 
- [176] Lindgren, Tony; Github - droid4-kexecboot; 2022; <https://github.com/tmlind/droid4-kexecboot.git>; Aufruf: 06.09.2022
  - [177] Peter, Christian; IT-Dad: Android 7.1 auf dem Asus Transformer Pad TF300T; 2019; <https://it-dad.de/2019/03/02/android-7-1-auf-dem-asus-transformer-pad-tf300t/>; Aufruf: 21.09.2022
  - [178] Ryhel, Svyatoslav ; Github - Grate Driver - Linux; 2022; <https://github.com/clamor-s/linux>; Aufruf: 21.09.2022
  - [179] Wietlisbach, Oliver; Dieses Smartphone gibt dir die Kontrolle über deine Daten zurück – und so funktioniert es; 2021; <https://www.watson.ch/digital/review/985327442-das-volla-phone-beweist-ein-smartphone-ganz-ohne-google-oder-apple-geht>; Aufruf: 23.09.2022
  - [180] Carneiro, Rúben; Volla Phone Repair; 2021; <https://rubencarneiro.github.io/rubencarneiro.io/Volla-phone-Repair/>;
  - [181] Ubports Forum - User: kaiyes; Flash halium vendor on redmi note 8t (willow) ; 2021; <https://forums.ubports.com/topic/6986/flash-halium-vendor-on-redmi-note-8t-willow>; Aufruf: 27.09.2022
  - [182] xda-developers - kasjan321; [ROM] [Mi A2 Lite] Droidian GSI; 2021; <https://forum.xda-developers.com/t/rom-mi-a2-lite-droidian-gsi.4335679/>; Aufruf: 09.10.2022
  - [183] xda-developers - RoiArthurB; [Alpha][oneplus3/t][Unofficial] Droidian (Debian Bookworm) for OnePlus 3/T; 2022; <https://forum.xda-developers.com/t/alpha-oneplus3-t-unofficial-droidian-debian-bookworm-for-oneplus-3-t.4482691/>; Aufruf: 11.10.2022
  - [184] UBports - SolAZDev; Gitlab - Samsung Galaxy 9 Series Ubuntu Touch Port; 2022; <https://gitlab.com/ubports/porting/community-ports/android10/samsung-galaxy-s9/samsung-exynos9810/>; Aufruf: 13.10.2022
  - [185] Ubuntu Wiki; Devices - M7; 2013; <https://wiki.ubuntu.com/Touch/Devices/M7>; Aufruf 19.10.2022
  - [186] PostmarketOS; Samsung Galaxy S III (samsung-m0); 2022; [https://wiki.postmarketos.org/wiki/Samsung\\_Galaxy\\_S\\_III\\_\(samsung-m0\)](https://wiki.postmarketos.org/wiki/Samsung_Galaxy_S_III_(samsung-m0)); Aufruf: 24.10.22
  - [187] PostmarketOS; Xiaomi Redmi 2 (xiaomi-wt88047); 2022; [https://wiki.postmarketos.org/wiki/Xiaomi\\_Redmi\\_2\\_\(xiaomi-wt88047\)](https://wiki.postmarketos.org/wiki/Xiaomi_Redmi_2_(xiaomi-wt88047)); Aufruf: 25.10.2022
  - [188] Mobian; Installing Mobian (Android devices); 2022; <https://wiki.mobian-project.org/doku.php?id=install-android>; Aufruf: 15.11.2022
  - [189] Danct12, Asriel Dreemurr; SailfishOS 3.x for Redmi 4X (santoni); 2020; <https://github.com/sailfish-santoni/projectmanagement>; Aufruf: 05.12.2022
  - [190] HengYeDev; Sailfish x86 amd64 v0.3 (Sailfish 4.0.1.48); 2021; <https://github.com/sailfish-x86/rootfs/releases/tag/0.3>; Aufruf: 12.12.2022
  - [191] XDA-Forum "FakeShell"; [Linux OS] Droidian Redmi 9C [dandelion][angelica]; 2022; <https://forum.xda-developers.com/t/linux-os-droidian-redmi-9c-dandelion-angelica.4471425/>; Aufruf: 01.01.2023
  - [192] Ubuntu Mate; Raspberry Pi Download Options; 2022; <https://ubuntu-mate.org/raspberry-pi/download/>; Aufruf: 31.01.2023

# Abbildungsverzeichnis

Bild 1: Marktanteile der führenden Betriebssysteme in Deutschland im Januar 2023.....	11
Bild 2: Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in Deutschland im 3. Quartal des Jahres 2022.....	11
Bild 3: Übersicht der Entwicklung mobiler Linux-Distributionen.....	14
Bild 4: Lomiri-Oberfläche in Ubuntu Touch 16.04.....	16
Bild 5: "Lipstick"-Oberfläche unter SailfishOS.....	17
Bild 6: "Lipstick"-Oberfläche unter AuroraOS (AuroraOS IDE Emulator).....	18
Bild 7: Phosh-Oberfläche unter Droidian.....	19
Bild 8: KDE Plasma Mobile unter PostmarketOS.....	20
Bild 9: SWMO-Oberfläche unter PostmarketOS.....	20
Bild 10: Nokia N9.....	23
Bild 11: Pinephone.....	24
Bild 12: Nexus 5.....	24
Bild 13: BQ Aquaris E5.....	24
Bild 14: Xperia XA2.....	25
Bild 15: Moto G.....	25
Bild 16: Lenovo A6000.....	25
Bild 17: Motorola Droid 4.....	26
Bild 18: ASUS TF300T.....	26
Bild 19: ASUS T100TA.....	26
Bild 20: Volla Phone.....	27
Bild 21: Redmi Note 8T.....	27
Bild 22: Galaxy S5.....	27
Bild 23: Mi A2 lite.....	28
Bild 24: Oneplus 3T.....	28
Bild 25: Galaxy S9.....	28
Bild 26: HTC One.....	29
Bild 27: Galaxy S III.....	29
Bild 28: Redmi 2.....	29
Bild 29: Pocophone F1.....	30
Bild 30: Redmi 4x.....	30
Bild 31: Dell Venue 8 Pro.....	30
Bild 32: Redmi 9C.....	31
Bild 33: Xperia 5.....	31
Bild 34: Surface 2.....	31
Bild 35: forensisches Prozessmodell nach Muth (Bildquelle: MT Muth [65 S. 38]).....	32
Bild 36: Funktionsumfang eines Redmi 7 unter Ubuntu Touch.....	36
Bild 37: Ablauf der Datensammlung.....	42
Bild 38: Abfrage des verwendeten Kernels über SSH.....	44
Bild 39: Klonen der Kernel-Quelldateien.....	45
Bild 40: Erstellen des Linux-Kernels.....	45
Bild 41: Kompilieren des LiME-Moduls gegen den erstellten Kernel.....	46
Bild 42: RAM-Sicherung des Pinephones über LiME.....	46
Bild 43: RAM-Sicherung des Asus T100TA über AVML.....	47
Bild 44: Erstellte komprimierte AVML-Sicherung im Lime-Format.....	47
Bild 45: Upload Mode auf dem Samsung Galaxy S9.....	48
Bild 46: RAM-Sicherung des Samsung Galaxy S9 über den SUC.....	49
Bild 47: Extrahierte Speicherbereiche des Samsung Galaxy S9.....	49
Bild 48: MTKClient - Verbindung im BROM-Modus.....	50
Bild 49: MTKClient - Sicherung der Arbeitsspeicherbereiche.....	51
Bild 50: Extrahierte Speicherbereiche des Volla Phones.....	51
Bild 51: Befehlsaufruf für die SSH-Sicherung des Xiaomi Redmi 2.....	54
Bild 52: Aufruf von lsblk zum Auffinden von Krypto-Partitionen.....	57

---

Bild 53: Sicherung des unverschlüsselten Home-Verzeichnisses des Sony Xperia XA2.....	58
Bild 54: Befehlsaufruf für die ADB-Sicherung des Volla Phone (GS290).....	62
Bild 55: Live-Boot von TWRP auf dem Redmi 4X über Fastboot.....	64
Bild 56: Recovery-Sicherung des Xiaomi Redmi 4X über ADB.....	65
Bild 57: Anzeige und Auswahl des Bootslots auf dem Xperia 5.....	68
Bild 58: Installation der Lineage Recovery in Bootslot A.....	68
Bild 59: Flashen von TWRP auf dem Samsung Galaxy S5.....	70
Bild 60: Galaxy S5 - TWRP wurde erfolgreich geflasht.....	70
Bild 61: Sicherungsaufbau des EDL Clients bei dem Oneplus 3T.....	73
Bild 62: Erkennen des sahara-Modus am Oneplus 3T über den EDL Client.....	74
Bild 63: EDL-Testpunkte des Redmi Note 8T.....	74
Bild 64: Erkennen des sahara-Modus am Redmi Note 8T über den EDL Client.....	75
Bild 65: Komprimieren der EDL-Sicherung des Oneplus 3T über ewfacquire.....	75
Bild 66: Sicherung des Volla Phones / GS290 über den MTKClient.....	77
Bild 67: Verwenden eines geeigneten Preloaders bei der Ausführung von MTKClient.....	77
Bild 68: Identifizierung des TF300T im APX-Modus.....	78
Bild 69: Senden des Fusée-Payloads an das Transformer Pad TF300T.....	80
Bild 70: Erfolgreicher Buffer Overflow Angriff auf den Bootloader des TF300T.....	80
Bild 71: Ausgabe des SBK des Transformer Pads TF300T.....	80
Bild 72: Erzeugen der Blob-Datei des TF300T für den Aufruf von nvflash.....	81
Bild 73: Versetzen des TF300T in den nvflash-Modus über wheelie.....	82
Bild 74: Sicherung des internen Flash-Speichers des TF300T über nvflash.....	83
Bild 75: Kernel-Ausgabe beim Aufruf des APX-Modus am Surface 2.....	84
Bild 76: Modifikation der U-Boot Konfiguration für den Start in den Mass Storage Mode.....	85
Bild 77: Anwendung des Fusée Gelée Exploits auf das Surface 2.....	85
Bild 78: Starten des erstellten U-Boot Images auf dem Surface 2 über nvflash.....	86
Bild 79: Surface 2 im Mass Storage Mode.....	86
Bild 80: Sicherung des Surface 2 im Mass Storage Mode über ewfacquire.....	86
Bild 81: FEL-Testpunkte des Pine64 Pinephone (Hardware Revision v1.2b).....	88
Bild 82: Angeschlossenes Pinephone im FEL-Modus.....	88
Bild 83: Booten des Pinephones in den U-Boot Mass Storage Mode.....	89
Bild 84: Sicherung des eMMC-Speichers des Pinephone über ewfacquire.....	89
Bild 85: Fehlschlag der Bootloader-Sicherung des Galaxy S9 über UFED4PC.....	90
Bild 86: Aufbau der Peripherie für die x86-Live-Sicherung mit Tsurugi Acquire.....	93
Bild 87: Sicherung des internen Flash-Speichers des T100TA über Guymager.....	93
Bild 88: eingesetzter Writeblocker bei der Sicherung einer µSD-Karte.....	98
Bild 89: Einsicht in die Datensicherung des Redmi 2 und Extraktion einer Bilddatei.....	100
Bild 90: Partitionstabelle der eingebundenen Userdata-Partition.....	101
Bild 91: Ausgabe von Dateisysteminformationen über fsstat.....	102
Bild 92: Offset der Userdata-Partition des Sony Xperia XA2.....	104
Bild 93: Einbinden der Userdata-Partition und darin enthaltener logischer Volumes.....	104
Bild 94: Aktivieren der logischen Volumes der Gruppe "sailfish".....	105
Bild 95: eingebundene lvm-Volumes im System der Workstation.....	105
Bild 96: Extraktion des LUKS-Headers mittels dd.....	105
Bild 97: Entschlüsselung der Home-Partition des Xperia XA2 über cryptsetup luksOpen.....	106
Bild 98: Einsicht in das entschlüsselte Home-Verzeichnis des Xperia XA2.....	106
Bild 99: Ermittlung des Offsets der Userdata-Partition des Xperia 5 über mmls.....	107
Bild 100: Einbinden der Userdata-Partition des Xperia 5 als Loop-Device.....	107
Bild 101: Suche nach dem Physical Volume Label Header mittels Hexedit.....	108
Bild 102: Auffinden der logischen Volumes am ermittelten Offset.....	108
Bild 103: Bruteforce der LUKS2-Verschlüsselung über "brute-force-luks" und eine Wortliste... ..	109
Bild 104: Ausgabe des gefundenen LUKS-Nutzerpassworts.....	109
Bild 105: Eingebundenes und entschlüsseltes Wurzeldateisystem des Xperia 5.....	110
Bild 106: Einsicht in die entschlüsselte Sicherung über fls.....	110
Bild 107: Moto G - Abfrage der Unlock-Daten über Fastboot.....	A
Bild 108: Moto G - Entsperren des Bootloaders über Fastboot.....	A
Bild 109: Moto G - Unlock Befehl.....	B
Bild 110: Moto G - Aufspielen von TWRP über Fastboot.....	B

Bild 111: Moto G - Installation von SailfishOS über TWRP.....	C
Bild 112: Nexus 5 - Aktivieren des Entwicklermodus.....	D
Bild 113: Nexus 5 - USB-Debugging aktivieren.....	D
Bild 114: Aquaris E5 - Beschreiben des Flash-Speichers mit dem SP Flash Tool.....	G
Bild 115: Aquaris E5 - Aktivieren des Entwicklermodus.....	G
Bild 116: Aquaris E5 - USB-Debugging erlauben.....	G
Bild 117: Aquaris E5 - Verwendung des Ubports Installers.....	H
Bild 118: T100TA - Aufruf des erweiterten Windows-Starts.....	I
Bild 119: T100TA - Reparatur des Systems über den Rettungsmodus.....	J
Bild 120: T100TA - Ändern der Release-Version auf Bookworm (testing).....	K
Bild 121: T100TA - Anlegen der Mobian Paketquellen.....	K
Bild 122: T100TA - Hinterlegen der korrekten Auflösung für Phosh.....	L
Bild 123: Lenovo A6000 - Installation von lk2nd.....	N
Bild 124: Lenovo A6000 - Aufruf von lk2nd.....	N
Bild 125: Nokia N9 - Aufspielen des Systemimages mittels flasher.....	O
Bild 126: Nokia N9 - gespiegeltes Repository.....	P
Bild 127: Xperia XA2 - Das Unlocken des Bootloaders ist gestattet.....	Q
Bild 128: Xperia XA2 - Unlocken des Bootloaders via Fastboot.....	R
Bild 129: Xperia XA2 - Ausführung des SailfishOS-Installationsskripts.....	S
Bild 130: Droid 4 - Aktualisierung von Android.....	T
Bild 131: Droid 4 - Installation von kexecboot.....	U
Bild 132: Pinephone - Beschreiben des eMMC über Jumpdrive.....	V
Bild 133: TF300T - Erstellen des Images über pmbootstrap.....	X
Bild 134: TF300T - Beschreiben der MicroSD Karte mit PostmarketOS.....	Y
Bild 135: TF300T - Beschreiben der Boot-Partition mit dem Linux-Kernel.....	Y
Bild 136: Volla Phone - Beschreiben des Gerätes mit dem Herstellerimage.....	Z
Bild 137: Redmi Note 8T - Frist zum Entsperren des Bootloaders.....	AA
Bild 138: Redmi Note 8T - Entsperren des Bootloaders.....	AA
Bild 139: Redmi Note 8T - Aufspielen des Android 9 Herstellerimages.....	AB
Bild 140: Redmi Note 8T - Flashen der Vendor-Partition des Redmi Note 8.....	AC
Bild 141: Galaxy S5 - Download Modus.....	AD
Bild 142: Mi A2 lite - Installation von Droidian.....	AF
Bild 143: Oneplus 3T - Entsperren des Bootloaders.....	AG
Bild 144: Oneplus 3T - Installation von TWRP und LineageOS.....	AI
Bild 145: Galaxy S9 - Flashen von Android 10 mittels Odin.....	AJ
Bild 146: Galaxy S9 - Flashen von TWRP über Heimdall.....	AK
Bild 147: Galaxy S9 - Beschreiben mit Ubuntu Touch über TWRP und ADB.....	AL
Bild 148: HTC One - Entsperren des Bootloaders.....	AM
Bild 149: HTC One - Einspielen des Ubuntu Touch Backup Images.....	AN
Bild 150: Samsung S III - Beschreiben der MicroSD-Karte.....	AO
Bild 151: Poco F1 - Ermittlung des Paneltyps via TWRP.....	AQ
Bild 152: Poco F1 - Installation von Mobian über Fastboot.....	AR
Bild 153: Redmi 4x - Übertragen der Image-Dateien.....	AS
Bild 154: Venue 8 Pro - Installation von Sailfish über Clonezilla.....	AT
Bild 155: Xperia 5 - Installation von Droidian über Fastboot.....	AV
Bild 156: Surface 2 - Secure Boot wurde deaktiviert.....	AX
Bild 157: Hildon-Oberfläche unter Maemo Leste.....	AY
Bild 158: Phosh-Oberfläche unter Droidian (Phosh Version 0.24).....	AY
Bild 159: Gnome Mobile unter PostmarketOS.....	AY
Bild 160: Cutie-Shell (Bildquelle: <a href="https://cutie-shell.org/screenshots">https://cutie-shell.org/screenshots</a> ).....	AZ
Bild 161: Glacier-Oberfläche (Bildquelle: <a href="https://nemomobile.net/glacier-home/">https://nemomobile.net/glacier-home/</a> ).....	AZ
Bild 162: angepasster Mate-Desktop (Bildquelle: <a href="https://archivista.ch">https://archivista.ch</a> ).....	AZ
Bild 163: Pangolin UI unter dahliaOS (VM).....	BA
Bild 164: Maui Shell (Bildquelle: <a href="https://github.com/Nitrox/maui-shell">https://github.com/Nitrox/maui-shell</a> ).....	BA
Bild 165: tuna (Bildquelle: <a href="https://github.com/Gusaroo/tuna/blob/main/screenshots/">https://github.com/Gusaroo/tuna/blob/main/screenshots/</a> ).....	BA
Bild 166: EDL-Testpunkte des Redmi 4X.....	BE
Bild 167: EDL-Testpunkte des Mi A2 Lite.....	BE

# Tabellenverzeichnis

Tabelle 1: Standardzugangsdaten der mobilen Linux-Systeme.....	22
Tabelle 2: Projektseiten der mobilen Linux-Distributionen.....	37
Tabelle 3: Stufen der Datensicherung (nach NIST).....	41
Tabelle 4: definierte USB-IP-Adressen der mobilen Distributionen.....	53
Tabelle 5: Sicherungsparameter für ewfacquirestream.....	54
Tabelle 6: TCP-Live-Sicherungen (Auszug).....	60
Tabelle 7: Recovery-Sicherungen (Auszug).....	71
Tabelle 8: Bootloader-Sicherungen (Auszug).....	91
Tabelle 9: Live-Linux-Sicherungen (x86).....	94
Tabelle 10: gerätespezifische Sicherungen.....	97
Tabelle 11: externe Flash-Speicher-Sicherungen.....	99
Tabelle 12: Übersicht der erfolgten Sicherungen nach Methode.....	111

# Software

Die folgende Software wurde zur Bearbeitung des Themas der Bachelorarbeit verwendet:

Software	Quelle
Android Tools 33.0.3-3	<a href="http://tools.android.com/">http://tools.android.com/</a>
arm-linux-gnueabi-hf-gcc13-linaro-bin 13.0-0	<a href="https://aur.archlinux.org/arm-linux-gnueabi-hf-gcc13-linaro-bin.git">https://aur.archlinux.org/arm-linux-gnueabi-hf-gcc13-linaro-bin.git</a>
AuroraOS-IDE 4.0.2.175	<a href="https://community-omprussia.ru">https://community-omprussia.ru</a>
AVML 0.11.0	<a href="https://github.com/microsoft/avml">https://github.com/microsoft/avml</a>
bruteforce-luks 1.4.0	<a href="https://github.com/glv2/bruteforce-luks">https://github.com/glv2/bruteforce-luks</a>
Clonezilla 3.0.2-21	<a href="https://clonezilla.org/downloads.php">https://clonezilla.org/downloads.php</a>
cryptsetup 2.6.1-3.4	<a href="https://gitlab.com/cryptsetup/cryptsetup">https://gitlab.com/cryptsetup/cryptsetup</a>
flasher 3.12.1	<a href="https://aur.archlinux.org/flasher-harmattan.git">https://aur.archlinux.org/flasher-harmattan.git</a>
gnome-disk-utility 42.0	<a href="https://gitlab.gnome.org/GNOME/gnome-disk-utility">https://gitlab.gnome.org/GNOME/gnome-disk-utility</a>
Gparted Live 1.4.0-6-amd64	<a href="https://gparted.org/download.php">https://gparted.org/download.php</a>
Hashcat 6.2.6	<a href="https://github.com/hashcat/hashcat">https://github.com/hashcat/hashcat</a>
Heimdall 1.4.2	<a href="https://glassechidna.com.au/heimdall/">https://glassechidna.com.au/heimdall/</a>
Hexedit 1.6-1	<a href="http://rigaux.org/hexedit">http://rigaux.org/hexedit</a>
Libewf 20140608-6	<a href="https://github.com/libyal/libewf">https://github.com/libyal/libewf</a>
LiME 1.9.1	<a href="https://github.com/504ensicsLabs/LiME">https://github.com/504ensicsLabs/LiME</a>
MTKClient 1.52	<a href="https://github.com/bkerler/mtkclient">https://github.com/bkerler/mtkclient</a>
nvflash 1.13.87205	<a href="https://github.com/tofurky/tegra30_debrick/tree/master/utlis">https://github.com/tofurky/tegra30_debrick/tree/master/utlis</a>
nvflash 3.08.1700	<a href="https://github.com/Open-Surface-RT/fusee-tools/tree/master/utlis">https://github.com/Open-Surface-RT/fusee-tools/tree/master/utlis</a>
qdl 1.0	<a href="https://git.linaro.org/landing-teams/working/qualcomm/qdl.git">https://git.linaro.org/landing-teams/working/qualcomm/qdl.git</a>
Qualcomm Sahara / Firehose Attack Client 3.52	<a href="https://github.com/bkerler/edl">https://github.com/bkerler/edl</a>
Rufus 3.21	<a href="https://rufus.ie/downloads/">https://rufus.ie/downloads/</a>
Sleuthkit 4.12.0	<a href="https://github.com/sleuthkit/sleuthkit">https://github.com/sleuthkit/sleuthkit</a>
SUC – Samsung Upload Client 1.0	<a href="https://github.com/bkerler/sboot_dump">https://github.com/bkerler/sboot_dump</a>
sunxi-tools 1.4	<a href="https://github.com/linux-sunxi/sunxi-tools">https://github.com/linux-sunxi/sunxi-tools</a>
Tsurugi Acquire 2021.1	<a href="https://tsurugi-linux.org/downloads.php">https://tsurugi-linux.org/downloads.php</a>
TWRP	<a href="https://twrp.me/Devices/">https://twrp.me/Devices/</a>
UBports Installer 0.9.5-beta	<a href="https://github.com/ubports/ubports-installer/releases/tag/0.9.5-beta">https://github.com/ubports/ubports-installer/releases/tag/0.9.5-beta</a>
UFED4PC 7.61.0.12	<a href="https://cellebrite.com/de/cellebrite-ufed-de/">https://cellebrite.com/de/cellebrite-ufed-de/</a>

## **Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ich erkläre ferner, dass ich die vorliegende Arbeit in keinem anderen Prüfungsverfahren als Prüfungsarbeit eingereicht habe oder einreichen werde.

Die eingereichte schriftliche Fassung entspricht der auf dem Medium gespeicherten Fassung.

XXX, 15.06.2023

# Anhangsverzeichnis

Anhang 1: Vorbereitung der Mobilgeräte.....	A
Motorola Moto G (2014) – SailfishOS.....	A
1. Bootloader unlocken.....	A
2. TWRP aufspielen.....	B
3. Installation von SailfishOS.....	C
4. Update von SailfishOS.....	C
Google Nexus 5 – Ubuntu Touch.....	D
1. Bootloader unlocken.....	D
3. Installation von Ubuntu Touch.....	E
BQ Aquaris E5 HD – Ubuntu Touch.....	F
1. Ubuntu Rom flashen.....	F
Asus Transformer Book T100TA – Mobian.....	I
1. Secure Boot deaktivieren.....	I
2. Installation von Mobian.....	J
Lenovo A6000 – PostmarketOS.....	M
1. Flash des aktuellen Systemimages.....	M
2. Installation von lk2nd und PostmarketOS.....	M
Nokia N9 – MeeGo.....	O
1. Flash des aktuellen Systemimages.....	O
2. Installation des N9 RepoMirrors.....	P
Sony Xperia XA2 – SailfishOS X.....	Q
1. Bootloader unlocken.....	Q
2. Installation von SailfishOS.....	R
Motorola Droid 4 – Maemo Leste.....	T
1. Aktualisierung von Android.....	T
2. Installation von kexecboot.....	U
3. Installation von Maemo.....	U
Pine64 Pinephone – Manjaro ARM KDE.....	V
1. Erstellen einer Jumpdrive-MicroSD-Karte.....	V
2. Installation von Manjaro auf den internen eMMC Speicher.....	V
Asus Transformer Pad TF300T – PostmarketOS.....	W
1. Unlocken des Bootloaders.....	W
2. Kompilieren des Linux-Kernels.....	W
3. Installation von PostmarketOS (pmbootstrap).....	W
Gigaset GS290 / Volla Phone – Ubuntu Touch.....	Z
1. Flash des Hersteller-Images.....	Z
Xiaomi Redmi Note 8T – Ubuntu Touch.....	AA
1. Entsperren des Bootloaders.....	AA
2. Installation von Ubuntu Touch und notwendiger Pakete.....	AB
Samsung Galaxy S5 – PostmarketOS.....	AD
1. Installation von PostmarketOS.....	AD
Xiaomi Mi A2 lite – Droidian.....	AE
1. Installation des Android 9 Images.....	AE
2. Installation von Droidian.....	AE
Oneplus 3T – Droidian.....	AG
1. Entsperren des Bootloaders.....	AG
2. „Treblizing“ des Geräts und Installation von Droidian.....	AH



Samsung Galaxy S9 – Ubuntu Touch.....	AJ
1. Entsperren des Bootloaders.....	AJ
2. Flashen der erforderlichen Software-Version.....	AJ
3. Installation von Ubuntu Touch.....	AK
HTC One M7 – Ubuntu Touch.....	AM
1. Entsperren des Bootloaders.....	AM
2. Installation von Ubuntu Touch.....	AN
Samsung Galaxy S III – PostmarketOS.....	AO
1. Beschreiben einer MicroSD-Karte.....	AO
2. Beschreiben der Boot-Partition.....	AO
Xiaomi Redmi 2 Pro – PostmarketOS.....	AP
1. Flash der aktuellen Firmware.....	AP
2. Flash von TRWP, LineageOS, lk2nd und PostmarketOS.....	AP
Xiaomi Pocophone F1 – Mobian.....	AQ
1. Entsperren des Bootloaders.....	AQ
2. Bestimmung des Paneltyps und Installation von Mobian.....	AQ
Xiaomi Redmi 4X – SailfishOS.....	AS
1. Entsperren des Bootloaders.....	AS
2. Flash von TWRP, LineageOS und Sailfish.....	AS
Dell Venue 8 Pro – SailfishOS.....	AT
1. Secure Boot deaktivieren.....	AT
2. Installation von SailfishOS.....	AT
Xiaomi Redmi 9C – Droidian.....	AU
1. Entsperren des Bootloaders.....	AU
2. Installation von Droidian.....	AU
Sony Xperia 5 – Droidian.....	AV
1. Bootloader unlocken.....	AV
2. Installation von Droidian.....	AV
Microsoft Surface 2 – Ubuntu Mate 22.04.....	AW
1. Secure Boot deaktivieren.....	AW
2. Installation von Ubuntu.....	AX
Anhang 2: mobile Oberflächen.....	AY
Anhang 3: Übersicht der Sicherungen.....	BB
TCP-Live-Sicherungen.....	BB
TCP-Live Sicherung (unverschlüsselt).....	BB
ADB-Sicherungen (Ubuntu Touch).....	BC
Recovery-Sicherungen.....	BC
Bootloader-Sicherungen.....	BD
Anhang 4: Testpunkte.....	BE
Xiaomi Redmi 4X (EDL).....	BE
Xiaomi Mi A2 Lite (EDL).....	BE
Anhang 5: digitale Anlagen.....	BF

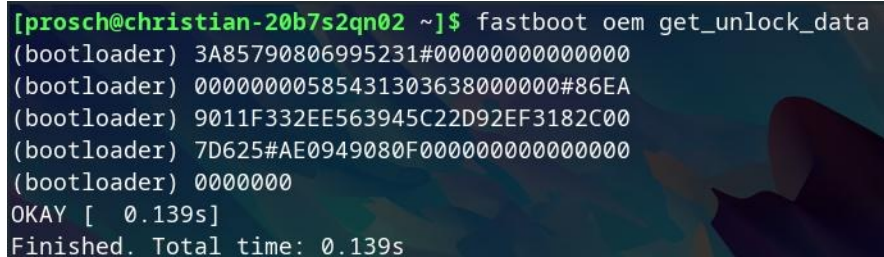
## Anhang 1: Vorbereitung der Mobilgeräte

### Motorola Moto G (2014) – SailfishOS

#### 1. Bootloader unlocken

Das Mobiltelefon befand sich noch im Originalzustand und wurde mit Android 6 betrieben. Für den Entsperrprozess war es notwendig, ein Google-Konto mit dem Gerät zu verknüpfen, um später eine Anfrage bei Motorola stellen zu können.

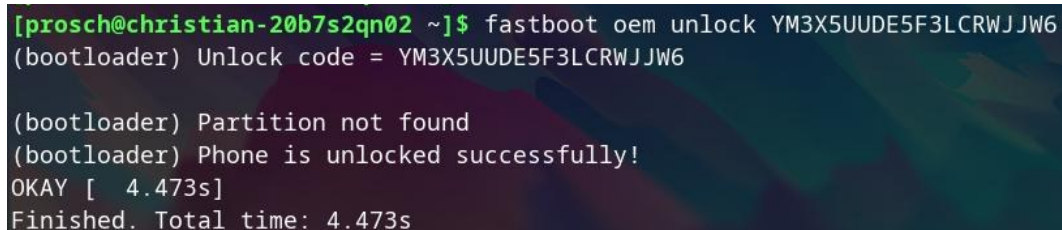
Über die Tastenkombination **{leiser}** + **{Power}** wurde das Gerät in den Fastboot-Modus gestartet. Nach der Verbindung mit dem PC konnten via fastboot-Befehl die benötigten Entsperrdaten abgefragt werden.



```
[prosch@christian-20b7s2qn02 ~]$ fastboot oem get_unlock_data
(bootloader) 3A85790806995231#00000000000000
(bootloader) 00000000585431303638000000#86EA
(bootloader) 9011F332EE563945C22D92EF3182C00
(bootloader) 7D625#AE0949080F00000000000000
(bootloader) 00000000
OKAY [ 0.139s]
Finished. Total time: 0.139s
```

**Bild 107:** Moto G - Abfrage der Unlock-Daten über Fastboot

Diese wurde an Motorola-Website [163] übermittelt. Es erfolgte eine Antwort-Mail auf die mit dem hinterlegten Google-Konto verknüpfte Mail-Adresse. Der Entsperrcode aus der erhaltenen Email wurde über fastboot an das Gerät übermittelt und der Bootloader somit entsperrt:

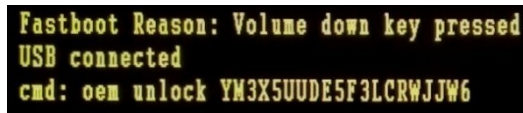


```
[prosch@christian-20b7s2qn02 ~]$ fastboot oem unlock YM3X5UUDE5F3LCRWJJW6
(bootloader) Unlock code = YM3X5UUDE5F3LCRWJJW6

(bootloader) Partition not found
(bootloader) Phone is unlocked successfully!
OKAY [ 4.473s]
Finished. Total time: 4.473s
```

**Bild 108:** Moto G - Entsperren des Bootloaders über Fastboot

Eine Bestätigung erfolgte ebenfalls auf dem Display des Gerätes:

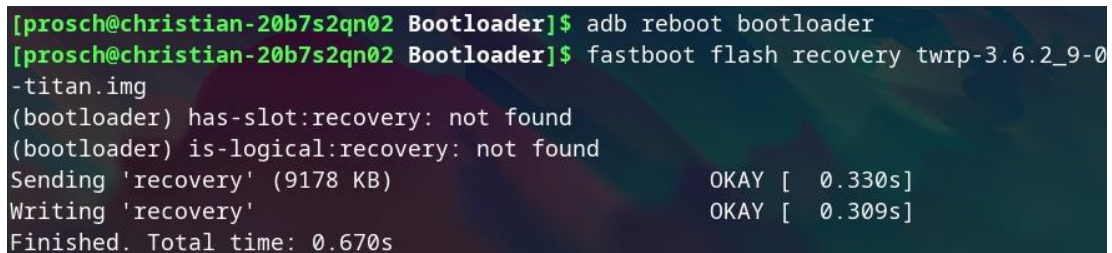


```
Fastboot Reason: Volume down key pressed
USB connected
cmd: oem unlock YM3X5UUDE5F3LCRWJJW6
```

Bild 109: Moto G - Unlock Befehl

## 2. TWRP aufspielen

Das entsprechende TWRP-Image für das Moto G (2014) konnte auf der offiziellen Homepage bezogen werden: <https://dl.twrp.me/titan/>. Hier wurde das aktuelle Image twrp-3.6.2\_9-0-titan.img gewählt. Nach der Verifikation der md5-Prüfsumme wurde TWRP via fastboot aufgespielt.



```
[prosch@christian-20b7s2qn02 Bootloader]$ adb reboot bootloader
[prosch@christian-20b7s2qn02 Bootloader]$ fastboot flash recovery twrp-3.6.2_9-0-titan.img
(bootloader) has-slot:recovery: not found
(bootloader) is-logical:recovery: not found
Sending 'recovery' (9178 KB)          OKAY [ 0.330s]
Writing 'recovery'                  OKAY [ 0.309s]
Finished. Total time: 0.670s
```

Bild 110: Moto G - Aufspielen von TWRP über Fastboot

Der Start von TWRP erfolgt über den Menüpunkt „Recovery“ aus dem Fastboot-Modus heraus. Unüblich ist hierbei, dass lediglich die **{leiser}** Taste für die Navigation verwendet wird und eine Bestätigung der Auswahl über die **{lauter}** Taste erfolgt. Üblicherweise wird eine Auswahl von Menüeinträgen über die **{Power}** Taste bestätigt.

### 3. Installation von SailfishOS

Die Installation erfolgte über TWRP entsprechend der Beschreibung im Thread „/Forums/Motorola/Moto G (2014)/G 2014 Android Development/Sailfish OS“ [164] des XDA-Forums. Dabei wurden zunächst die Partitionen System, Cache, Data und Dalvik gewiped. Anschließend erfolgte die Installation von CM 12 (cm-12.1-20151117) zur Gewährleistung des erforderlichen Firmware-Standes. Zuletzt wurde das SailfishOS-RootFS Image installiert.

```
Installing zip file '/sdcard/sailfishos-titan-
release-4.2.0.21-STABLE17.zip'
Unmounting System...
=====
Hybris Installer
=====
Device: titan
Version: 4.2.0.21
Image: sailfishos-titan-release-4.2.0.21-
STABLE17.tar.bz2
Size: 423M
```

Bild 111: Moto G - Installation von SailfishOS über TWRP

### 4. Update von SailfishOS

Ein Update auf die aktuelle Version 4.4.0.58 erfolgte entsprechend der Anleitung im XDA-Forum über die Befehle (Terminal auf dem Gerät):

```
devel-su ssu re 4.4.0.58
```

```
devel-su ssu ar adaptation-community http://repo.merproject.org »
```

```
/obs/nemo:/testing:/hw:/motorola:/titan/sailfishos_4.4.0.58/
```

```
devel-su ssu ar adaptation-community-common http://repo.merproject »
```

```
.org/obs/nemo:/testing:/hw:/common/sailfishos_4.4.0.58/
```

```
devel-su zypper clean -a
```

```
devel-su zypper ref -f
```

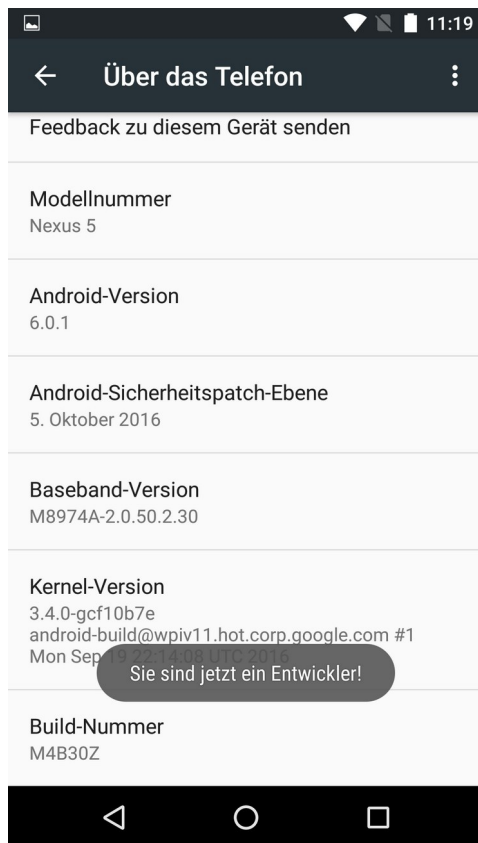
```
version --dup
```

```
reboot phone
```

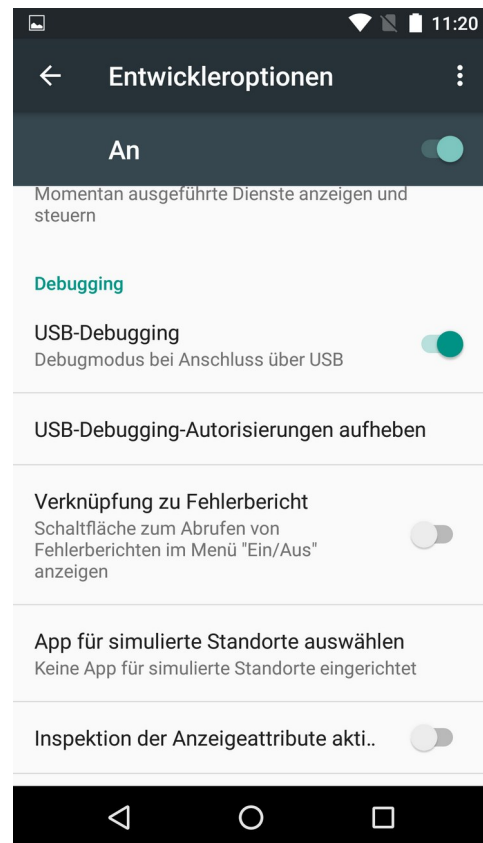
# Google Nexus 5 – Ubuntu Touch

## 1. Bootloader unlocken

Das Entsperren des Bootloaders erfordert bei diesem Gerät keine Kontaktaufnahme zum Hersteller. Im installierten Android System ist lediglich die Aktivierung des Entwicklermodus und der ADB-Verbindung erforderlich. Im Menüpunkt „Über das Telefon“ wurde der Eintrag „Build-Nummer“ siebenmal betätigt. Im neu verfügbaren Menü „Entwickleroptionen“ konnte in der Folge das USB-Debugging aktiviert werden.



**Bild 112:** Nexus 5 - Aktivieren des Entwicklermodus



**Bild 113:** Nexus 5 - USB-Debugging aktivieren

Um in den Fastboot Modus zu booten, wurde das Gerät zunächst ausgeschaltet und über die Tastenkombination **{leiser} + {Power}** neu gestartet. Es wurde eine USB-Verbindung zum PC aufgebaut und über das Terminal der Befehl:

```
fastboot oem unlock
```

abgesetzt. Eine Abfrage auf dem Gerätedisplay wurde mit „Yes – Unlock bootloader (may void warranty)“ bestätigt und das Gerät neu gestartet.

### 3. Installation von Ubuntu Touch

Das Ubuntu Touch Installationstool konnte von der offiziellen UBPorts Homepage [165](#) bezogen werden:

Es wurde die Option „Other Linux distributions (ApplImage)“ gewählt und so die Datei: `ubports-installer_0.9.5-beta_linux_x86_64.ApplImage` geladen. Da über den OEM-Unlock sämtliche Nutzerdaten gelöscht werden, musste erneut eine Ersteinrichtung des Geräts vorgenommen und USB-Debugging aktiviert werden.

Das Mobiltelefon wurde erneut per USB mit dem PC verbunden und der Ubports Installer gestartet. Dieser erkannte das verwendete Nexus 5 und führte über wenige Eingaben zu einem voll automatisierten Installationsprozess.

Nach Abschluss der Installationsroutine startete das Gerät neu und zeigte den Ubuntu Touch Installationsassistenten. Bemerkenswert ist hierbei, dass bei der Wahl einer PIN zum Entsperren des Sperrbildschirms nur 4 Zahlen akzeptiert werden.

## BQ Aquaris E5 HD – Ubuntu Touch

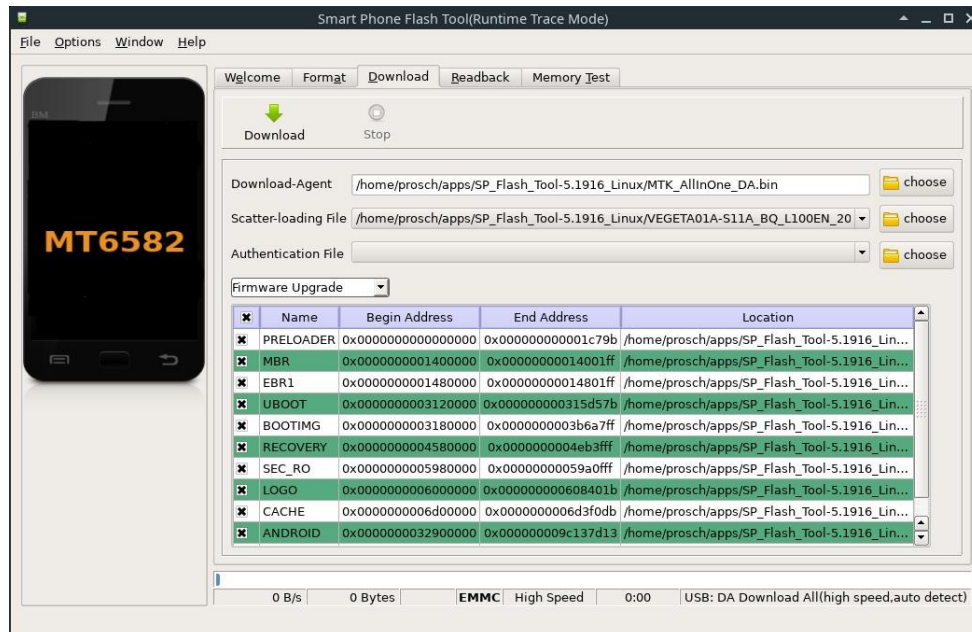
### 1. Ubuntu Rom flashen

Das Aquaris E5 ist eines der wenigen Smartphones, das in Kooperation mit Canonical als „Powered by Ubuntu“-Variante vertrieben wurde. Das genutzte Gerät wies jedoch eine werksseitige Android-Installation auf. Gemäß der Angaben des UBports Portals ist das Beschreiben des Flash Speichers über das SP Flash Tool direkt mit dem Ubuntu-Herstellerimage möglich. Über die bereitgestellten Links der Installationsseite konnte das Image: VEGETA01A-S11A\_BQ\_L100EN\_2017\_170207.zip bezogen werden. Das SP Flash Tool wurde in der derzeit aktuellen Version 5.1916 von der Entwicklerseite bezogen. Der Start des Tools erfolgte über den Befehl:

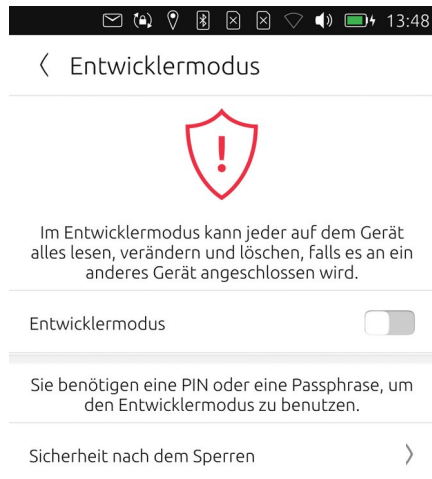
```
sudo ./flash_tool.sh
```

Kurz darauf erschien das Programmfenster des SP Flashtools. Der Anleitung des UBports Installationsleitfadens folgend, wurde die Datei „Android\_scatter.txt“ aus dem entpackten Image-Zip-Archiv über den Button „Scatter-loading File“ eingelesen und der Modus „Firmware Upgrade“ gewählt. Der Button „Download“ führt zum Beschreiben des Flash-Speichers im Preloader-Modus sobald das ausgeschaltete Smartphone per USB mit dem PC verbunden wird (Bild 114).

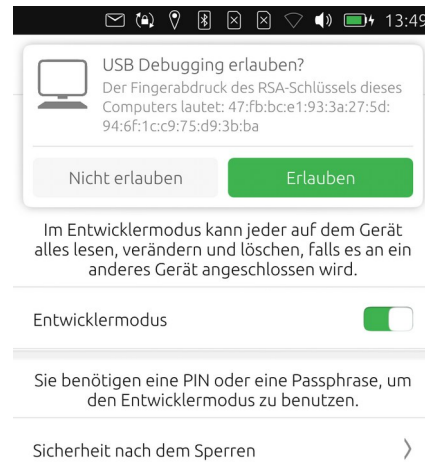
Hierbei kam es zunächst zur Ausgabe eines Verbindungsfehlers während des Schreibvorgangs. Eine erneute Betätigung des Download-Buttons und ein erneutes Verbinden des Smartphones führte zu einem erfolgreichen Beschreiben des Flash-Speichers. Nach dem Gerätestart wurde die Originalsoftware auf Basis von Ubuntu 15.04 geladen. Um die aktuelle UBports Version von Ubuntu Touch zu installieren, wurde der Entwicklermodus im Bereich der Systemeinstellungen aktiviert und die USB-Debugging-Verbindung mit dem Host-System gestattet (Bild 115 und 116).



**Bild 114:** Aquaris E5 - Beschreiben des Flash-Speichers mit dem SP Flash Tool



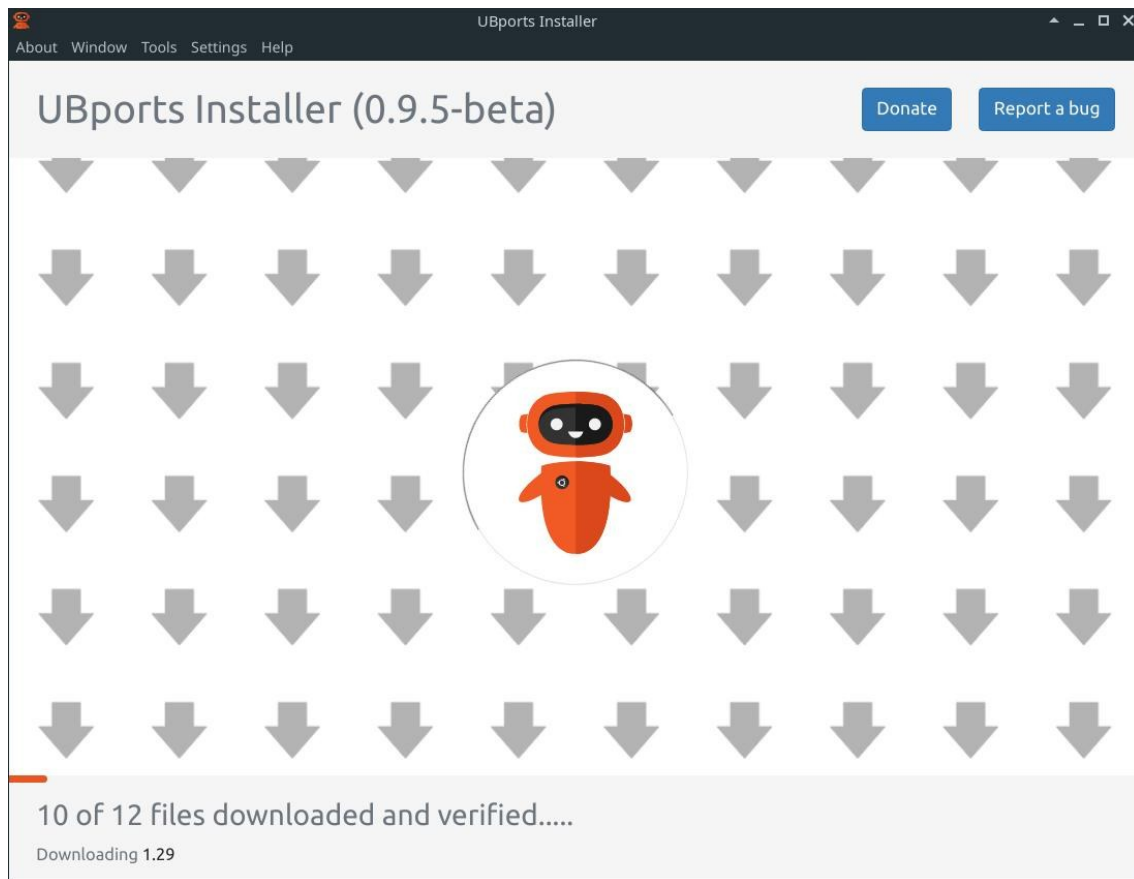
**Bild 115:** Aquaris E5 - Aktivieren des Entwicklermodus



**Bild 116:** Aquaris E5 - USB-Debugging erlauben



Daraufhin wurde das Smartphone über den UBports Installer erkannt und wie bereits das Nexus 5 über USB mit der aktuellen Ubuntu Touch Version beschrieben.



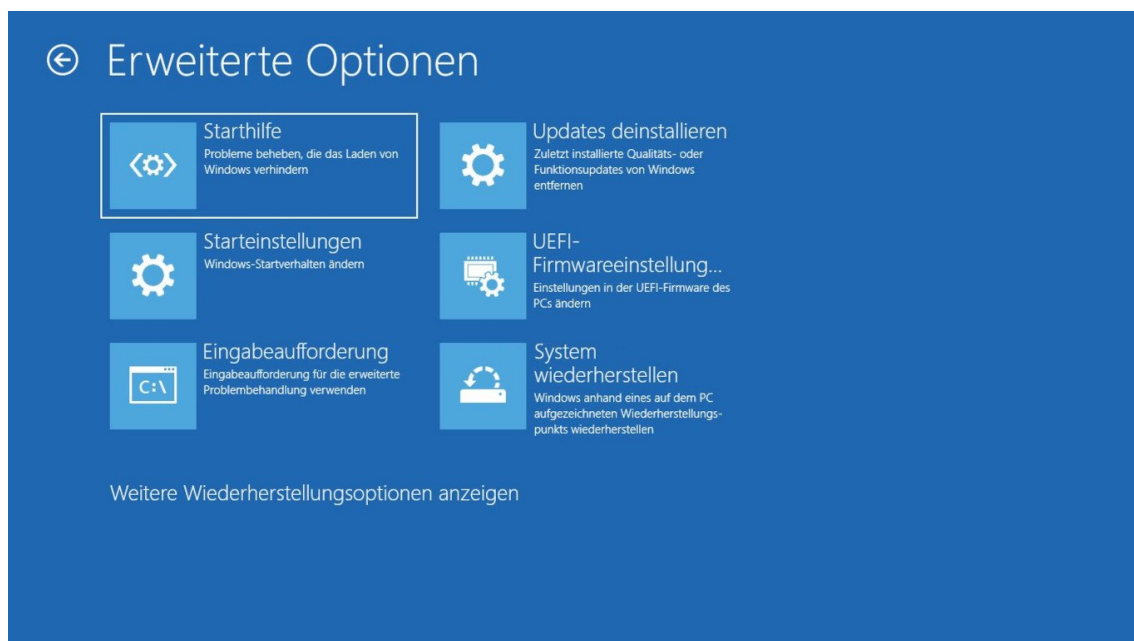
**Bild 117:** Aquaris E5 - Verwendung des Ubports Installers

Nach der erfolgreichen Installation startete das Gerät neu und zeigte den Ubuntu Touch Einrichtungsassistenten.

## Asus Transformer Book T100TA – Mobian

### 1. Secure Boot deaktivieren

Bei dem Transformer Book handelt es sich um ein x86 Tablet/Convertible, welches mit Windows 10 (32 Bit) ausgeliefert wird. Die Installation alternativer Betriebssysteme erfordert das Deaktivieren von Secure Boot in den UEFI Settings. Aus dem laufenden Windows 10 heraus sind diese über die Wiederherstellungsoptionen und die Auswahl „erweiterter Start“ erreichbar (siehe Bild 118).



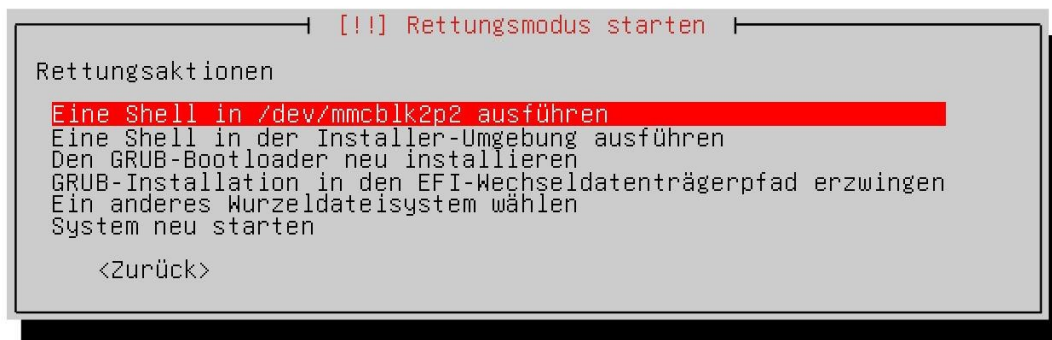
**Bild 118:** T100TA - Aufruf des erweiterten Windows-Starts

Das Secure-Boot Menü findet sich anschließend unter dem Reiter „Security“

## 2. Installation von Mobian

Es wurde der Versuch unternommen, das Mobian x86 UEFI Image von der offiziellen Projektseite (<https://wiki.mobian-project.org/doku.php?id=install-x86>) über einen USB-Stick am Gerät zu laden. Dies gelang nicht. Ursächlich ist hier das vorhandene 32Bit UEFI, welches den Start eines 64Bit EFI Files nicht gestattet. Eine 32Bit Version von Mobian wird nicht bereit gestellt. Da die CPU des T100TA jedoch in der Lage ist 64Bit Systeme auszuführen, wurde eine aktuelles „netinst multiarch Image“ von der offiziellen Debian Downloadseite bezogen [166]. Um die Unterstützung des verwendeten WLAN-Chipsatzes zu gewährleisten, wurde ein „non-free“ Image gewählt. Diese Installation gelang. Der Start von Debian über Grub endete jedoch bereits mit der Meldung „Loading initial ramdisk“.

Daher wurde das Gerät erneut über das Installationsmedium gestartet und der Rettungsmodus gewählt. Nach der Wahl der korrekten Systempartition mmcblk2p2 konnte in der Umgebung des Systems eine Shell geöffnet werden.



**Bild 119:** T100TA - Reparatur des Systems über den Rettungsmodus

Der Befehl:

update-grub

stellte die Startfähigkeit des Systems wieder her. Gemäß der Angaben im offiziellen Mobian-Blog [167], setzt Mobian aktuell auf dem Debian Release „Bookworm“ auf. Daher wurden die Einträge in der „sources.list“ Datei entsprechend von „bullseye“ auf „bookworm“ geändert.

```

GNU nano 6.3 /etc/apt/sources.list
#deb cdrom:[Debian GNU/Linux 11.3.0 _Bullseye_ - Unofficial Multi-architecture amd64/i386 NET]
deb http://deb.debian.org/debian/ bookworm main non-free contrib
deb-src http://deb.debian.org/debian/ bookworm main non-free contrib

deb http://security.debian.org/debian-security bookworm-security main contrib non-free
deb-src http://security.debian.org/debian-security bookworm-security main contrib non-free

# bullseye-updates, to get updates before a point release is made;
# see https://www.debian.org/doc/manuals/debian-reference/ch02.en.html#_updates_and_backports
deb http://deb.debian.org/debian/ bookworm-updates main contrib non-free
deb-src http://deb.debian.org/debian/ bookworm-updates main contrib non-free

```

Bild 120: T100TA - Ändern der Release-Version auf Bookworm (testing)

Um die Mobian-Pakete in die Installation zu laden, wurde die Mobian-Repository in der neuen Datei „mobian.list“ hinterlegt.

```

GNU nano 6.3 /etc/apt/sources.list.d/mobian.list
deb http://repo.mobian-project.org/ bookworm main non-free

```

Bild 121: T100TA - Anlegen der Mobian Paketquellen

Weiter war es erforderlich, den GPG Schlüssel der Paketquelle zu beziehen:

```
wget -O - - https://repo.mobian-project.org/mobian.gpg.key |
```

```
>>
```

```
sudo apt-key add -
```

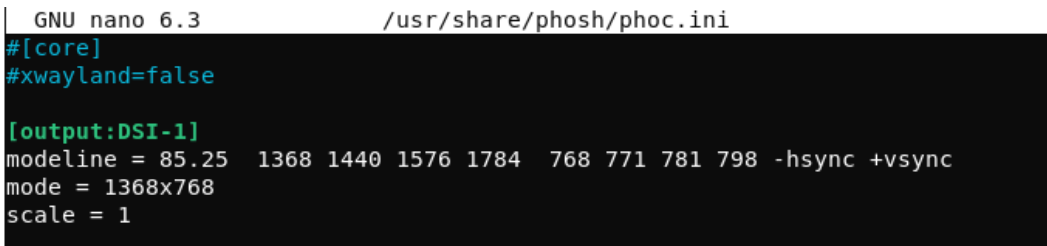
Die Installation der Mobian Pakete erfolgte mittels des apt-Paketmanagers. Dabei wurden die Pakete:

calamares-extensions, mobian-plymouth-theme, chrony, mobian-phosh-base, mobian-phosh-extras, ibegl-mesa0 und phosh-tablet installiert.

Im Anschluss konnte die vorinstallierte Gnome-Shell deaktiviert und Phosh aktiviert werden:

```
sudo systemctl disable gdm3 && sudo systemctl enable phosh
```

Aufgrund eines stark ausgeprägten Overscans war eine Bedienung des Systems nach dem ersten Neustart erheblich erschwert. Dies ließ sich über eine CVT-konform abgewandelte Eintragung der Bildschirmauflösung in der Datei „/usr/share/phosh/phoc.ini“ beheben:



```
GNU nano 6.3 /usr/share/phosh/phoc.ini
#[core]
#xwayland=false

[output:DSI-1]
modeline = 85.25 1368 1440 1576 1784 768 771 781 798 -hsync +vsync
mode = 1368x768
scale = 1
```

**Bild 122:** T100TA - Hinterlegen der korrekten Auflösung für Phosh

Zuletzt wurde der angelegte Nutzer „mobian“ per „visudo“ in die „/etc/sudoers“ Datei eingepflegt, da der Standardnutzer gemäß der FAQ des Mobian-Projekts per „sudo“ zur Ausführung von Befehlen mit Root-Berechtigung fähig ist.

Dabei wurde unter dem Punkt „sudoers“ der Eintrag:

```
mobian    ALL=(ALL:ALL) ALL
```

ergänzt.

## Lenovo A6000 – PostmarketOS

### 1. Flash des aktuellen Systemimages

Da das Testgerät eine fehlerhafte Boot-Partition aufwies, wurde zunächst das aktuelle Systemimage (Lenovo\_A6000\_S013\_Q102113\_20161230) über das Kommandozeilentool „qdl“ im EDL-Modus aufgespielt.

Dazu wurde der Befehl:

```
qdl prog_emmc_firehose_8916.mbn rawprogram_unsparse.xml »
```

```
patch0.xml
```

ausgeführt und das ausgeschaltete Gerät mit gedrückten **{lauter}** und **{leiser}** Tasten per USB mit dem PC verbunden. Im Anschluss an den Flashvorgang war ein Zugriff auf den Fastboot- und Recoverymodus möglich.

### 2. Installation von lk2nd und PostmarketOS

Entsprechend der Handlungsanweisungen auf der offiziellen PostmarketOS Homepage [168] konnte der Fastboot-Modus über die Tastenkombination **{Power}** und **{leiser}** aufgerufen werden. Abweichend von dieser Anleitung war auch hier das Entsperren des Bootloaders über den Befehl:

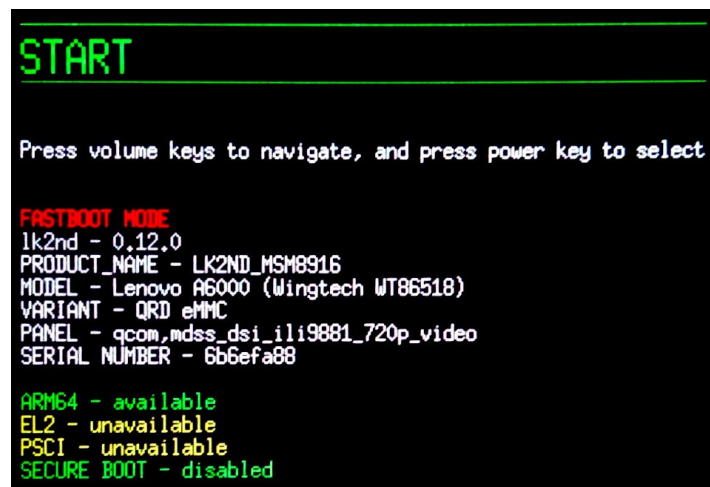
```
fastboot oem unlock
```

erforderlich. In der Folge konnte der zweite Bootloader „lk2nd“ [169] über Fastboot installiert werden:

```
[prosch@christian-20b7s2qn02 Bootloader]$ fastboot flash boot lk2nd-msm8916.img
Sending 'boot' (396 KB)                                OKAY [ 0.016s]
Writing 'boot'                                          OKAY [ 0.027s]
Finished. Total time: 0.054s
```

Bild 123: Lenovo A6000 - Installation von lk2nd

Da über die Kombination aus {Power} und {leiser} der erste Bootloader geladen wird, erfolgt der Aufruf von lk2nd während des Gerätestarts über die Betätigung der {leiser} Taste unmittelbar nach der ersten Vibration des Gerätes.



```
START

Press volume keys to navigate, and press power key to select

FASTBOOT MODE
lk2nd - 0.12.0
PRODUCT_NAME - LK2ND_MSM8916
MODEL - Lenovo A6000 (Wingtech WT86518)
VARIANT - QRD eMMC
PANEL - qcom,mdss_dsi_ili9881_720p_video
SERIAL NUMBER - 6b6efa98

ARM64 - available
EL2 - unavailable
PSCI - unavailable
SECURE BOOT - disabled
```

Bild 124: Lenovo A6000 - Aufruf von lk2nd

In der Ausgabe (Bild 124) ist zu erkennen, dass ein Panel des Typs ili9881 verbaut wurde. Dieses ist mit der angebotenen Version von PostmarketOS für das A6000 kompatibel. Daher wurde von der offiziellen Projektseite das aktuelle Image (22.06) in der Phosh-Version geladen.

Die Installation des Images erfolgte über den Fastboot-Modus in lk2nd:

```
fastboot flash userdata 20220704-0918-postmarketOS- »
v22.06-phosh-18-lenovo-a6000.img
```

Ein Neustart des Telefons zeigte nach Eingabe der Standard-Pin (147147) den PostmarketOS Willkommensbildschirm.

## Nokia N9 – MeeGo

### 1. Flash des aktuellen Systemimages

Das Nokia N9 wurde bereits werksseitig mit MeeGo ausgeliefert.

Um einen neuwertigen Zustand ohne Reste von Nutzerdaten oder Veränderungen an Systemeinstellungen zu gewährleisten, wurde das aktuelle Image „DFL61\_HARMATTAN\_40.2012.21-3\_PR\_LEGACY\_009-OEM1-958\_ARM.bin“ über die Software „flasher“ auf das Gerät geschrieben und anschließend das EMMC-Nutzerimage „E7FEB593\_DFL61HARMATTAN\_40.2012.13-.UKIRELAND\_EMMC\_UKIRELAND.bin“ geladen. Diese Images wurden dazu zunächst in „main.bin“ und „emmc.bin“ umbenannt. Anschließend wurden die folgenden Befehle abgesetzt und das ausgeschaltete Gerät per USB mit dem Host-PC verbunden.

```
flasher -f -F main.bin
```

```
flasher -f -F main.bin -F emmc.bin --flash-only=mmc
```

Zunächst wurde bei dem Flashvorgang des Images „main.bin“ ein Fehler ausgegeben. Ein erneutes Beschreiben mittels „flasher“ wurde jedoch fehlerfrei ausgeführt:

Battery level 21 %, continuing.

image	[state	progress	transfer		flash speed]
-----					
[x] cert-sw	[finished	100 %	1 /	1 kB	NA ]
[x] cmt-2nd	[finished	100 %	95 /	95 kB	NA ]
[x] cmt-algo	[finished	100 %	789 /	789 kB	NA ]
[x] cmt-mcusw	[finished	100 %	6050 /	6050 kB	3153 kB/s]
[x] xloader	[finished	100 %	23 /	23 kB	NA ]
[x] secondary	[finished	100 %	94 /	94 kB	NA ]
[x] kernel	[finished	100 %	2714 /	2714 kB	1900 kB/s]
[x] rootfs	[finished	100 %	1170206 /	1170206 kB	10860 kB/s]
Updating SW release					
Success					

**Bild 125:** Nokia N9 - Aufspielen des Systemimages mittels flasher



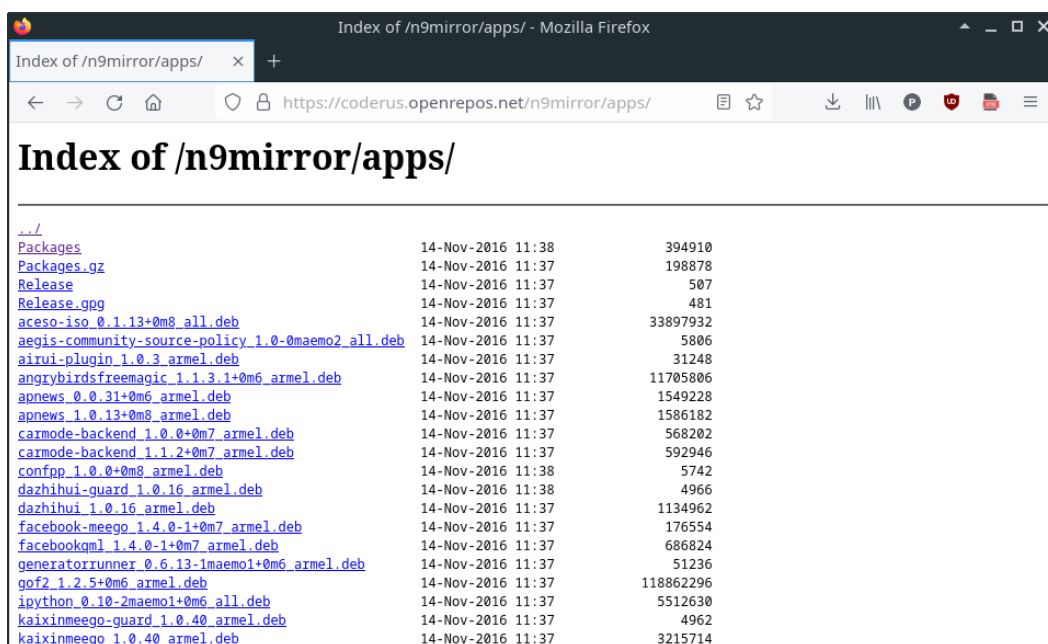
## 2. Installation des N9 RepoMirrors

Da Nokia sein offizielles Repository Anfang des Jahres 2014 einstellte [170], ist ein produktiver Einsatz des Nokia N9 im Jahr 2022 ohne Weiteres kaum möglich.

Seitens der MeeGo Community wurde jedoch eine Kopie des Nokia Repositories erstellt, welche als deb-Paket auf dem Gerät installiert werden kann. [171]

Zur Installation des Pakets war es zunächst notwendig, unter [Einstellungen] → [Programme] → [Installationen] die Installation von Fremdquellen zu gestatten.

Das heruntergeladene File: n9repomirror\_0.7.2\_armel.deb wurde anschließend via Bluetooth an das Gerät übertragen und dort fehlerfrei installiert. Erst nach dieser Installation ist das Aktivieren des 'Entwicklermodus' am Gerät möglich, wodurch in der App-Übersicht eine Terminal-Applikation verfügbar wird, welche für die Installation weiterer Programme und Abhängigkeiten genutzt werden kann.



File Name	Date	Size
../		
<a href="#">Packages</a>	14-Nov-2016 11:38	394910
<a href="#">Packages.gz</a>	14-Nov-2016 11:37	198878
<a href="#">Release</a>	14-Nov-2016 11:37	507
<a href="#">Release.gpg</a>	14-Nov-2016 11:37	481
<a href="#">aceso-iso_0.1.13+0m8_all.deb</a>	14-Nov-2016 11:37	33897932
<a href="#">aegis-community-source-policy_1.0-0maemo2_all.deb</a>	14-Nov-2016 11:37	5806
<a href="#">airui-plugin_1.0.3_armel.deb</a>	14-Nov-2016 11:37	31248
<a href="#">angrybirdsfreemagic_1.1.3.1+0m6_armel.deb</a>	14-Nov-2016 11:37	11705806
<a href="#">apnews_0.0.31+0m6_armel.deb</a>	14-Nov-2016 11:37	1549228
<a href="#">apnews_1.0.13+0m8_armel.deb</a>	14-Nov-2016 11:37	1586182
<a href="#">carmode-backend_1.0.0+0m7_armel.deb</a>	14-Nov-2016 11:37	568202
<a href="#">carmode-backend_1.1.2+0m7_armel.deb</a>	14-Nov-2016 11:37	592946
<a href="#">confpp_1.0.0+0m8_armel.deb</a>	14-Nov-2016 11:38	5742
<a href="#">dazhuhui-guard_1.0.16_armel.deb</a>	14-Nov-2016 11:38	4966
<a href="#">dazhuhui_1.0.16_armel.deb</a>	14-Nov-2016 11:37	1134962
<a href="#">facebook-meeGo_1.4.0-1+0m7_armel.deb</a>	14-Nov-2016 11:37	176554
<a href="#">facebookqml_1.4.0-1+0m7_armel.deb</a>	14-Nov-2016 11:37	686824
<a href="#">generatorrunner_0.6.13-1maemo1+0m6_armel.deb</a>	14-Nov-2016 11:37	51236
<a href="#">go2_1.2.5+0m6_armel.deb</a>	14-Nov-2016 11:37	118862296
<a href="#">ipython_0.10.2maemo1+0m6_all.deb</a>	14-Nov-2016 11:37	5512630
<a href="#">kaixinmeeGo-guard_1.0.40_armel.deb</a>	14-Nov-2016 11:37	4962
<a href="#">kaixinmeeGo_1.0.40_armel.deb</a>	14-Nov-2016 11:37	3215714

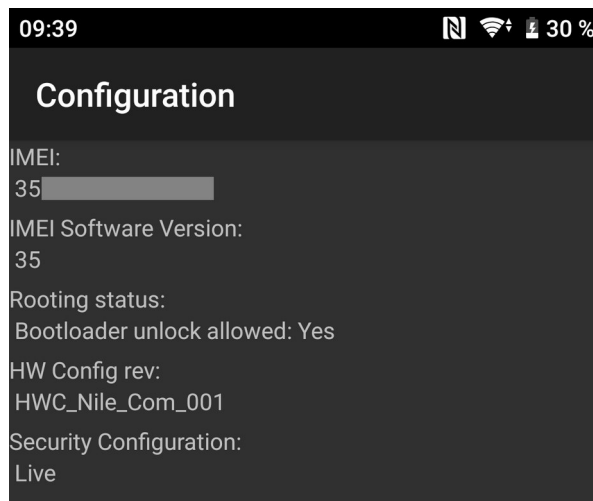
Bild 126: Nokia N9 - gespiegeltes Repository

# Sony Xperia XA2 – SailfishOS X

## 1. Bootloader unlocken

Bei dem Sony Xperia XA2 handelt es sich um ein von Jolla im Rahmen des SailfishOS X Projekts offiziell unterstütztes Gerät. Entsprechend wird eine Anleitung zum Entsperren des Bootloaders und zum Flashen von SailfishOS auf der offiziellen Homepage [172] bereitgestellt.

Zunächst sollte über die Eingabe: `*##7378423##` in der Dialer-App verifiziert werden, ob sich das Gerät für die Bootloader-Entsperrung eignet:



**Bild 127:** Xperia XA2 - Das Unlocken des Bootloaders ist gestattet

Das Entsperren des Bootloaders erfordert die Freischaltung der Entwickleroptionen über das siebenfache Betätigen der Buildnummer in den Geräteeigenschaften. Im neu verfügbaren Menü „Entwickleroptionen“ sind nun die Punkte „USB debugging“ und „OEM unlocking“ zu aktivieren.

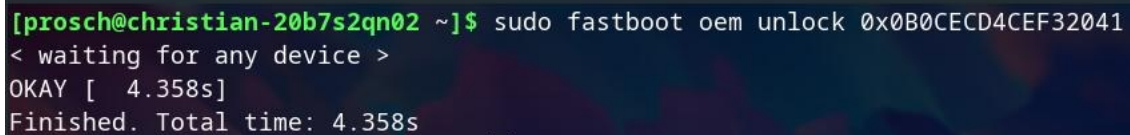
Die zuvor ermittelte IMEI-Nummer wird zur Bereitstellung des Entsperrcodes an die Sony Open Devices Plattform [173] übermittelt. Im konkreten Fall wurde der Entsperrcode: 0B0CECD4CEF32041 generiert.

Das Gerät wurde anschließend ausgeschaltet und mit gedrückter **{lauter}** Taste an den Host-PC angeschlossen. Die **{lauter}** Taste war dabei so lange gedrückt

zu halten, bis die LED im oberen rechten Bereich des Geräts ihre Farbe von rot auf blau änderte. Über die Kommandozeile konnte nun der Befehl:

```
sudo fastboot oem unlock 0x0B0CECD4CEF32041
```

abgesetzt werden, welcher erfolgreich ausgeführt wurde.



```
[prosch@christian-20b7s2qn02 ~]$ sudo fastboot oem unlock 0x0B0CECD4CEF32041
< waiting for any device >
OKAY [ 4.358s]
Finished. Total time: 4.358s
```

**Bild 128:** Xperia XA2 - Unlocken des Bootloaders via Fastboot

Eine Verbindung zum Telefon war hier nur über einen USB2-Anschluss möglich. Versuche, das Gerät über USB3 zu verbinden, führten regelmäßig zu einem Neustart des Mobiltelefons in den Lademodus. Nach der Bootloader-Installation war (abweichend von der Installationsanleitung von Jolla) ein Neustart des Geräts erforderlich. Erst nach dem Neustart gelang die Ausführung des Installationsskripts.

## 2. Installation von SailfishOS

Die Installation erfolgte hier über das von Jolla bereitgestellte Installationsskript, welches nach einer Registrierung auf der Website bezogen werden kann. Es wurde vom Kauf einer Lizenz für Sailfish X abgesehen und lediglich die kostenfreie Testversion geladen. Diese unterscheidet sich unter anderem in der fehlenden Kompatibilität zu Android-Apps und der Microsoft-Exchange Unterstützung. Weiterhin wurden für die Installation die „Software binaries for AOSP Oreo (Android 8.1) – Kernel 4.4 – Nile (v16)“ von Sony [174] geladen und in das Verzeichnis des Jolla-Installationsscripts extrahiert.

Nach dem beschriebenen Prinzip konnte erneut eine Verbindung über den Fastboot-Modus zum Gerät aufgebaut und das Installationsscript ausgeführt werden:

```
[prosch@christian-20b7s2qn02 XA2_Sailfish]$ sudo bash ./flash.sh
Flash utility v1.2
Detected Linux
Searching device to flash..
Found H3113, serial:CQ3000CFHJ, baseband:1311-2845_50.2.A.3.77, bootloader:1310-0301_X_Boot_SDM630_LA3.0_P_38
Found matching device with serial CQ3000CFHJ
Fastboot command: fastboot -s CQ3000CFHJ
>> fastboot -s CQ3000CFHJ getvar secure
<< secure: no
>> fastboot -s CQ3000CFHJ flash:raw boot_a hybris-boot.img
Sending 'boot_a' (18680 KB) OKAY [ 0.548s]
Writing 'boot_a' OKAY [ 0.211s]
Finished. Total time: 0.808s
>> fastboot -s CQ3000CFHJ flash:raw boot_b hybris-boot.img
Sending 'boot_b' (18680 KB) OKAY [ 0.543s]
Writing 'boot_b' OKAY [ 0.172s]
Finished. Total time: 0.734s
>> fastboot -s CQ3000CFHJ flash userdata sailfish.img001
Sending sparse 'userdata' 1/3 (517110 KB) OKAY [ 14.471s]
Writing 'userdata' OKAY [ 0.000s]
Sending sparse 'userdata' 2/3 (524285 KB) OKAY [ 15.690s]
Writing 'userdata' OKAY [ 0.000s]
Sending sparse 'userdata' 3/3 (438989 KB) OKAY [ 13.700s]
Writing 'userdata' OKAY [ 0.000s]
Finished. Total time: 43.934s
>> fastboot -s CQ3000CFHJ flash system_b fimage.img001
Sending sparse 'system_b' 1/2 (524284 KB) OKAY [ 15.540s]
Writing 'system_b' OKAY [ 0.000s]
Sending sparse 'system_b' 2/2 (122688 KB) OKAY [ 17.680s]
Writing 'system_b' OKAY [ 0.000s]
Finished. Total time: 93.725s
>> fastboot -s CQ3000CFHJ flash vendor_a vendor.img001
Sending 'vendor_a' (316 KB) OKAY [ 0.015s]
Writing 'vendor_a' OKAY [ 0.000s]
Finished. Total time: 3.369s
>> fastboot -s CQ3000CFHJ flash vendor_b vendor.img001
Sending 'vendor_b' (316 KB) OKAY [ 0.014s]
Writing 'vendor_b' OKAY [ 0.000s]
Finished. Total time: 142.832s
>> fastboot -s CQ3000CFHJ flash oem_a ./SW_binaries_for_Xperia_Android_8.1.6.4_r1_v16_nile.img
Sending 'oem_a' (211516 KB) OKAY [ 6.370s]
Writing 'oem_a' OKAY [ 1.645s]
Finished. Total time: 150.891s
```

**Bild 129:** Xperia XA2 - Ausführung des SailfishOS-Installationskripts

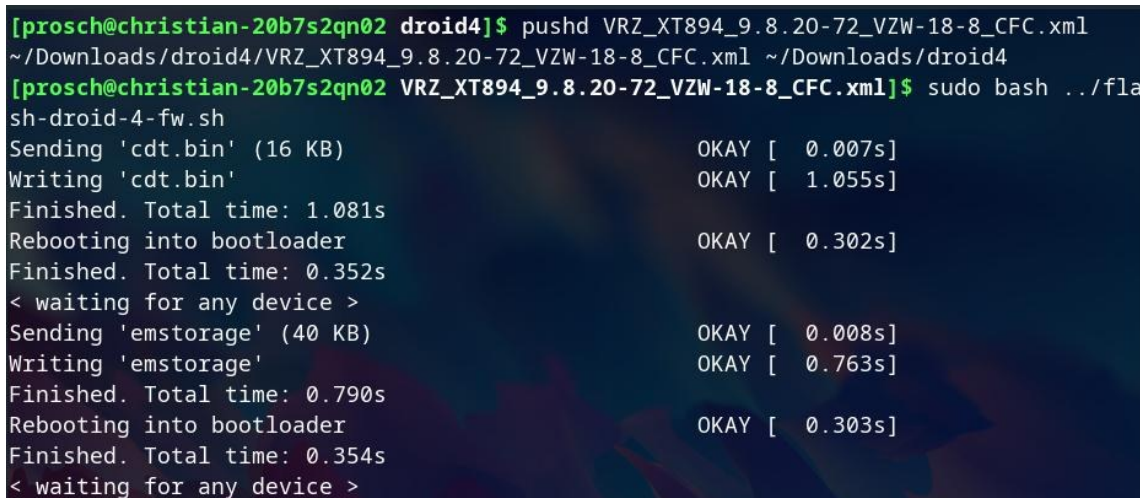
Die Ausführung des Skripts führte zum Absetzen diverser fastboot-Befehle zur Erstellung eines neuen Partitionsschemas und dem Beschreiben der neuen Partitionen. Nach einem Neustart des Geräts wurde der Ersteinrichtungsassistent von SailfishOS angezeigt. Im Rahmen der Ersteinrichtung erfolgte die Verschlüsselung des Gerätespeichers.

## Motorola Droid 4 – Maemo Leste

### 1. Aktualisierung von Android

Um die aktuelle Firmware bereitzustellen, macht sich die Installation der zuletzt erschienenen Android-Version erforderlich. Diese ist zusammen mit einem passenden Installationsscript und der aktuellen Version von Maemo Leste für das Droid 4 von der Projektseite beziehbar [175].

Für die Installation wird das Gerät im ausgeschalteten Zustand über die **{leiser}** und **{Power}** Tasten in den Fastboot-Modus versetzt, die Datei „cdma\_maserati\_9.8.20-72\_VZW-18-8\_cfc.xml“ per [pushd] auf das Gerät übertragen und im Anschluss das Installationsscript „flash-droid-4-fw.sh“ ausgeführt. Die zuvor übertragene xml-Datei beinhaltet die Hashwerte der einzelnen Partitionsfiles und dient der Verifikation der erfolgreichen Übertragung.



```
[prosch@christian-20b7s2qn02 droid4]$ pushd VRZ_XT894_9.8.20-72_VZW-18-8_CFC.xml
~/Downloads/droid4/VRZ_XT894_9.8.20-72_VZW-18-8_CFC.xml ~/Downloads/droid4
[prosch@christian-20b7s2qn02 VRZ_XT894_9.8.20-72_VZW-18-8_CFC.xml]$ sudo bash ../flash-droid-4-fw.sh
Sending 'cdt.bin' (16 KB)                                OKAY [ 0.007s]
Writing 'cdt.bin'                                       OKAY [ 1.055s]
Finished. Total time: 1.081s
Rebooting into bootloader                              OKAY [ 0.302s]
Finished. Total time: 0.352s
< waiting for any device >
Sending 'emstorage' (40 KB)                             OKAY [ 0.008s]
Writing 'emstorage'                                    OKAY [ 0.763s]
Finished. Total time: 0.790s
Rebooting into bootloader                              OKAY [ 0.303s]
Finished. Total time: 0.354s
< waiting for any device >
```

**Bild 130:** Droid 4 - Aktualisierung von Android

Während der Installation startet das Gerät mehrfach neu. Ein weiterer Neustart vor der Ausführung weiterer Fastboot-Befehle ist nicht erforderlich.



## 2. Installation von kexecboot

Im Rahmen der Gerätevorbereitung wird Android hier nicht durch Maemo ersetzt. Es wird lediglich ein Bootloader namens kexecboot installiert, welcher den Start eines Betriebssystems auf einer eingesetzten MicroSD-Karte erlaubt. Dieser wird auf der Partition 13 installiert und der Gerätebootloader derart modifiziert, dass Partition 13 beim Gerätestart geladen wird. Die benötigten Dateien können vom Github-Repository „droid4-kexecboot.git“ bezogen werden [176].

Auch diese Installation wird im Fastboot-Modus durchgeführt:

```
[prosch@christian-20b7s2qn02 droid4]$ sudo fastboot flash mbm VRZ_XT894_9.8.20-72_VZ
W-18-8_CFC.xml/allow-mbmloader-flashing-mbm.bin
[sudo] Passwort für prosch:
Sending 'mbm' (512 KB)                                OKAY [ 0.032s]
Writing 'mbm'                                          OKAY [ 0.525s]
Finished. Total time: 0.578s
[prosch@christian-20b7s2qn02 droid4]$ sudo fastboot reboot-bootloader
Rebooting into bootloader                             OKAY [ 0.006s]
Finished. Total time: 0.056s
[prosch@christian-20b7s2qn02 droid4]$ sudo fastboot flash bpsw droid4-kexecboot.img
Sending 'bpsw' (199 KB)                                OKAY [ 0.017s]
Writing 'bpsw'                                         OKAY [ 0.246s]
[prosch@christian-20b7s2qn02 droid4]$ sudo fastboot flash utags utags-mmcb1k1p13.bin
Sending 'utags' (199 KB)                                OKAY [ 0.017s]
Writing 'utags'                                         OKAY [ 0.333s]
Finished. Total time: 0.369s
[prosch@christian-20b7s2qn02 droid4]$ sudo fastboot reboot
Rebooting                                              OKAY [ 0.006s]
Finished. Total time: 1.562s
```

Bild 131: Droid 4 - Installation von kexecboot

## 3. Installation von Maemo

Die Image-Datei (maemo-leste-1.0-armhf-droid4-20220612.img.xz) wurde mittels Gnome Disks auf eine MicroSD Karte (SanDisk 32GB) übertragen und diese in das Gerät eingesetzt. Über das Kexecboot-Menü kann Maemo nun geladen werden.

## Pine64 Pinephone – Manjaro ARM KDE

### 1. Erstellen einer Jumpdrive-MicroSD-Karte

Zwar wurde das vom Hersteller bezogene Pinephone werksseitig mit Manjaro KDE ausgeliefert; aufgrund veralteter Pakete war jedoch das Updaten des Systems zu Beginn nicht möglich. Gemäß der Beschreibung des Herstellers kann ein neues Image über Jumpdrive geladen werden [152]. Dazu wurde das Pinephone-Jumpdrive-Image vom entsprechenden Github-Repository [151] bezogen und per Gnome Disks auf eine MicroSD-Karte geladen (8GB Sandisk).

### 2. Installation von Manjaro auf den internen eMMC Speicher

Das Pinephone konnte nun mit der erstellten Jumpdrive-MicroSD-Karte gestartet werden. Nach dem Anschluss an das Hostsystem wurde der interne Speicher als USB-Laufwerk ausgegeben und konnte mittels Gnome Disks mit dem aktuellen Manjaro KDE Image (Manjaro-ARM-plasma-mobile-pinephone-beta12.img.xz) beschrieben werden.



Bild 132: Pinephone - Beschreiben des eMMC über Jumpdrive

# Asus Transformer Pad TF300T – PostmarketOS

## 1. Unlocken des Bootloaders

Für ein früheres Android-Projekt wurde der Bootloader des Geräts bereits entsperrt. Dazu wurde eine Apk-Installation verwendet (UnLock\_Device\_App\_V8.apk). Die erfolgten Schritte können im zugehörigen Blog-Eintrag nachvollzogen werden [177].

## 2. Kompilieren des Linux-Kernels

Gemäß der Installationsanleitung der Projektseite [135] ist das Kompilieren eines kompatiblen Kernels erforderlich. Wegen seiner Aktualität wurde der Grate-Driver-Fork des Github Users Clamor-S gewählt [178].

Zunächst wurde ein Cross-Compiler auf dem Hostsystem installiert. Es wurde das Paket: „arm-linux-gnueabi-hf-gcc75-linaro-bin 7.5-0“ gewählt und über die Arch User Repository bezogen. Dieser Compiler wurde anschließend der ARM-Architektur zugewiesen:

```
export ARCH="arm" CROSS_COMPILE=/bin/arm-linux-gnueabi-hf-
```

Über eine Default-Konfiguration konnte der Kernel nun kompiliert werden:

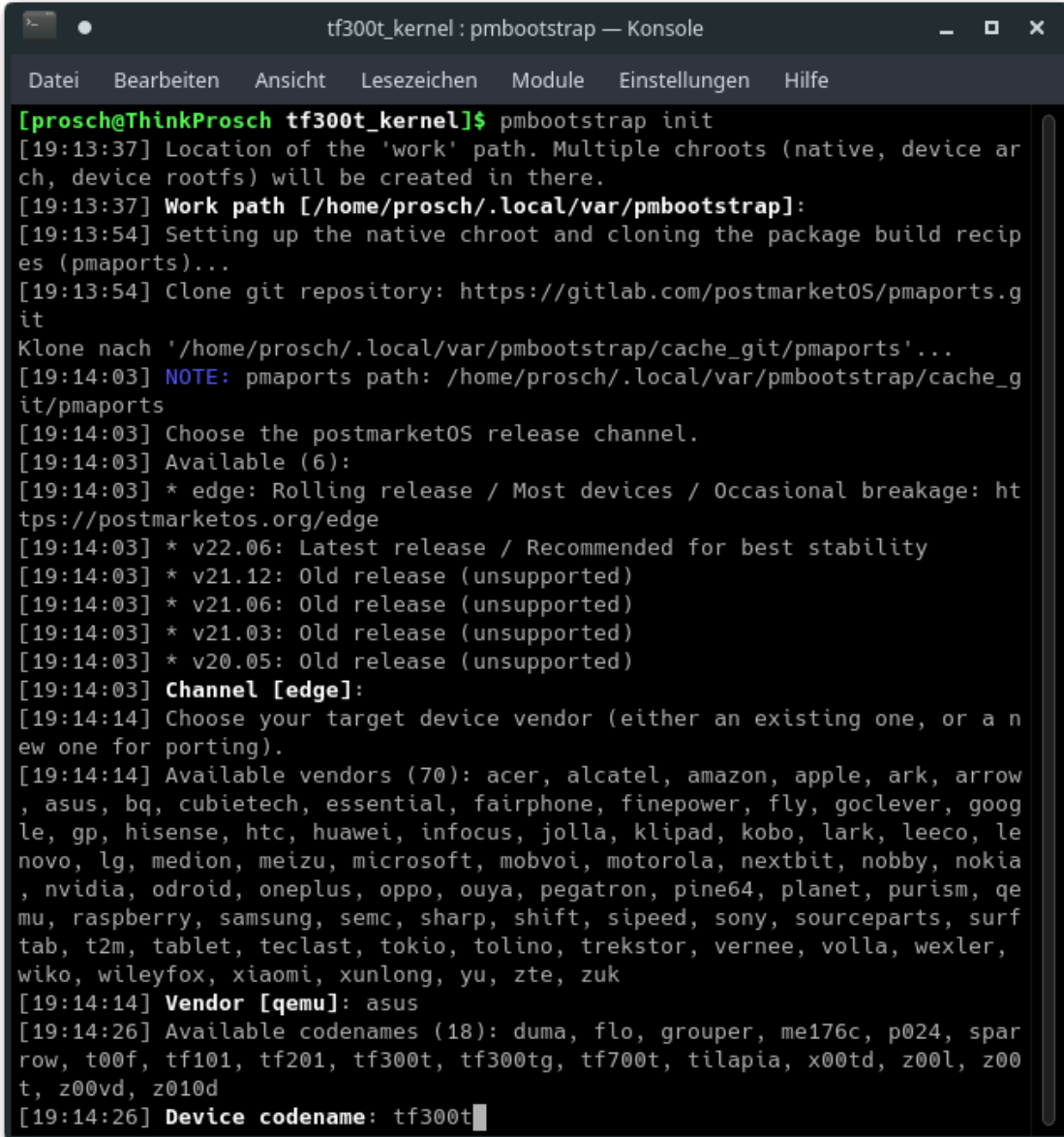
```
make transformer_defconfig
```

## 3. Installation von PostmarketOS (pmbootstrap)

Pmbootstrap konnte wie schon zuvor der ARM-Compiler über die AUR bezogen werden. Der Aufruf von `pmbootstrap init` führte zu einem Konfigurator zum Erstellen des gewünschten Images. Dabei wurde der Release Channel „edge“,



der Vendor „asus“ und das Modell „tf300t“ gewählt. Gemäß der Empfehlung auf der Projektseite wurde die Desktopumgebung „Mate“ installiert, da mobile Oberflächen mangels GPU Unterstützung keine flüssige Bedienung zulassen.



```
tf300t_kernel : pmbootstrap — Konsole
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe
[prosch@ThinkProsch tf300t_kernel]$ pmbootstrap init
[19:13:37] Location of the 'work' path. Multiple chroots (native, device arch, device rootfs) will be created in there.
[19:13:37] Work path [/home/prosch/.local/var/pmbootstrap]:
[19:13:54] Setting up the native chroot and cloning the package build recipes (pmaports)...
[19:13:54] Clone git repository: https://gitlab.com/postmarketOS/pmaports.git
Klone nach '/home/prosch/.local/var/pmbootstrap/cache_git/pmaports'...
[19:14:03] NOTE: pmaports path: /home/prosch/.local/var/pmbootstrap/cache_git/pmaports
[19:14:03] Choose the postmarketOS release channel.
[19:14:03] Available (6):
[19:14:03] * edge: Rolling release / Most devices / Occasional breakage: https://postmarketos.org/edge
[19:14:03] * v22.06: Latest release / Recommended for best stability
[19:14:03] * v21.12: Old release (unsupported)
[19:14:03] * v21.06: Old release (unsupported)
[19:14:03] * v21.03: Old release (unsupported)
[19:14:03] * v20.05: Old release (unsupported)
[19:14:03] Channel [edge]:
[19:14:14] Choose your target device vendor (either an existing one, or a new one for porting).
[19:14:14] Available vendors (70): acer, alcatel, amazon, apple, ark, arrow, asus, bq, cubietech, essential, fairphone, finepower, fly, goclever, google, gp, hisense, htc, huawei, infocus, jolla, klipad, kobo, lark, leeco, lenovo, lg, medion, meizu, microsoft, mobvoi, motorola, nextbit, nobby, nokia, nvidia, odroid, oneplus, oppo, ouya, pegatron, pine64, planet, purism, qemu, raspberry, samsung, semc, sharp, shift, sipeed, sony, sourceparts, surf tab, t2m, tablet, teclast, tokyo, tolino, trekstor, vernee, volla, wexler, wiko, wileyfox, xiaomi, xunlong, yu, zte, zuk
[19:14:14] Vendor [qemu]: asus
[19:14:26] Available codenames (18): duma, flo, grouper, me176c, p024, sparrow, t00f, tf101, tf201, tf300t, tf300tg, tf700t, tilapia, x00td, z00l, z00t, z00vd, z010d
[19:14:26] Device codename: tf300t
```

**Bild 133:** TF300T - Erstellen des Images über pmbootstrap

Nach der Erstellung des Images konnte eine MicroSD-Karte mit dem Image beschrieben und der Kernel via Fastboot an das Tablet gesendet werden:

```

tf300t_kernel : pmbootstrap — Konsole
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe

[prosch@ThinkProsch tf300t_kernel]$ pmbootstrap install --sdcard=/dev/mmcblk0
[19:30:07] *** (1/4) PREPARE NATIVE CHROOT ***
[19:30:10] (native) install cryptsetup util-linux parted
[sudo] Passwort für prosch:
[19:30:16] *** (2/4) CREATE DEVICE ROOTFS ("asus-tf300t") ***
[19:30:20] (rootfs_asus-tf300t) install postmarketos-base device-asus-tf300t postmarketos-ui-mate
device-asus-tf300t-nonfree-firmware blueman gvfs mate-tweak network-manager-applet onboard pavucon
trol pulseaudio-utils unclutter-xfixes lang musl-locales postmarketos-base-nofde
[19:30:25] (rootfs_asus-tf300t) install device-asus-tf300t
[19:30:31] (rootfs_asus-tf300t) install postmarketos-mkinitfs
[19:30:36] (rootfs_asus-tf300t) mkinitfs postmarketos-grate
[19:30:39] *** SET LOGIN PASSWORD FOR: 'prosch' ***
New password:
Retype new password:
passwd: password updated successfully
[19:30:53] NOTE: No valid keymap specified for device
[19:31:00] *** (3/4) PREPARE INSTALL BLOCKDEVICE ***
[19:31:00] (native) mount /dev/install (host: /dev/mmcblk0)
[19:31:00] EVERYTHING ON /dev/mmcblk0 WILL BE ERASED! CONTINUE? (y/n) [n]: y
[19:31:07] (native) partition /dev/install (boot: 256M, reserved: 0M, root: the rest)
[19:31:07] (native) install e2fsprogs
[19:31:08] (native) format /dev/installp2 (root, ext4)
[19:31:27] (native) mount /dev/installp2 to /mnt/install
[19:31:28] (native) install e2fsprogs
[19:31:29] (native) format /dev/installp1 (boot, ext2), mount to mountpoint
[19:31:29] *** (4/4) FILL INSTALL BLOCKDEVICE ***
[19:31:29] (native) copy rootfs_asus-tf300t to /mnt/install/
[19:31:33] Unmounting SD card (this may take a while to sync, please wait)

```

Bild 134: TF300T - Beschreiben der MicroSD Karte mit PostmarketOS

```

tf300t_kernel : bash — Konsole
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe

[prosch@ThinkProsch tf300t_kernel]$ pmbootstrap flasher flash_kernel
[19:22:30] (rootfs_asus-tf300t) install device-asus-tf300t
[19:22:36] (rootfs_asus-tf300t) install postmarketos-mkinitfs
[19:22:42] (rootfs_asus-tf300t) mkinitfs postmarketos-grate
[19:22:46] (native) flash kernel postmarketos-grate
[19:22:46] (native) install android-tools avbtool
Sending 'boot' (6864 KB)                                OKAY [ 2.632s]
Writing 'boot'                                           OKAY [ 7.614s]
Finished. Total time: 10.303s
[19:23:02] You will get an IP automatically assigned to your USB interface shortly.
[19:23:02] Then you can connect to your device using ssh after pmOS has booted:
[19:23:02] ssh prosch@172.16.42.1
[19:23:02] NOTE: If you enabled full disk encryption, you should make sure that osk-sdl has been p
roperly configured for your device
[19:23:02] NOTE: chroot is still active (use 'pmbootstrap shutdown' as necessary)
[19:23:02] DONE!
[prosch@ThinkProsch tf300t_kernel]$

```

Bild 135: TF300T - Beschreiben der Boot-Partition mit dem Linux-Kernel

Nach dem Einsetzen der MicroSD-Karte konnte das Tablet in den Mate-Desktop gestartet werden. Der Fastboot-Modus des Tablets lässt sich über **{leiser}** + **{Power}** aufrufen.

# Gigaset GS290 / Volla Phone – Ubuntu Touch

## 1. Flash des Hersteller-Images

Das vorliegende Gerät (Gigaset) wurde mit Android 9 betrieben. Da es sich bei dem Volla Phone um ein baugleiches Gerät handelt [179], welches sich nur in der installierten Software unterscheidet, konnte das Hersteller Image der Firma Hallo Welt verwendet werden. Gemäß einer Reparaturanleitung von Rúben Carneiro [180] konnte das Ubuntu Touch Herstellerimage im Jahr 2021 noch von der Herstellerseite ([www.volla.tech](http://www.volla.tech)) bezogen werden. Diese Möglichkeit stand zum Zeitpunkt der Anfertigung dieser Arbeit nicht mehr zur Verfügung. Eine Kopie des Images wurde über den Telegram-Channel „Volla User Community“ bezogen.

Dieses Image ließ sich über das SP Flash Tool nach Einladen der Datei „MT6763\_Android\_scatter.txt“ auf das Gerät schreiben. Dazu wurde das Mobiltelefon ausgeschaltet und nach Betätigung des „Download“ Buttons mit gedrückten **{lauter}** und **{leiser}** Tasten per USB mit dem Hostsystem verbunden.

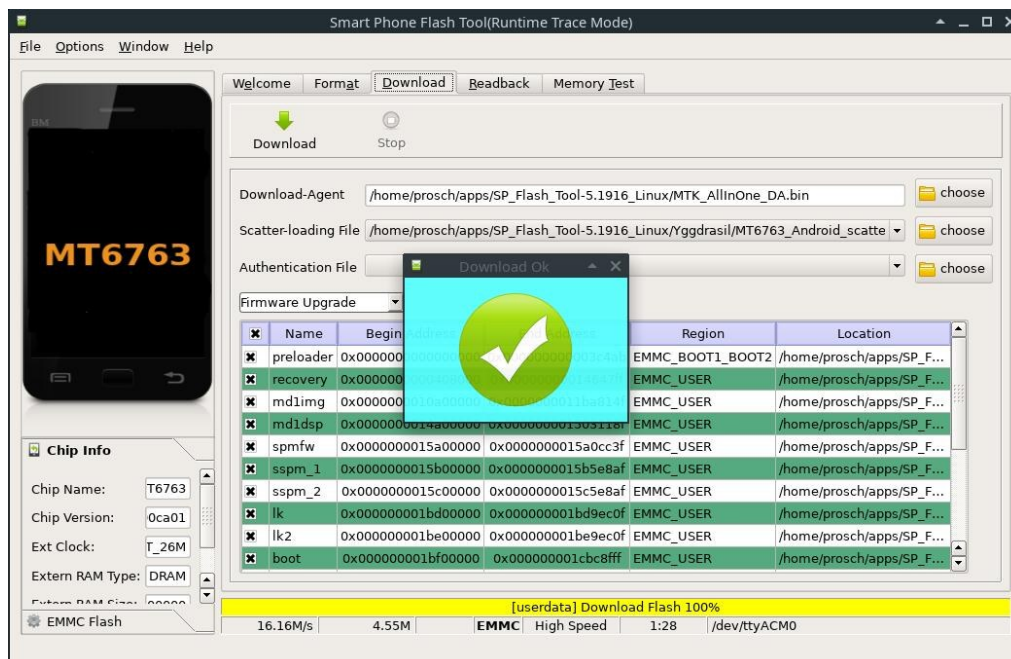


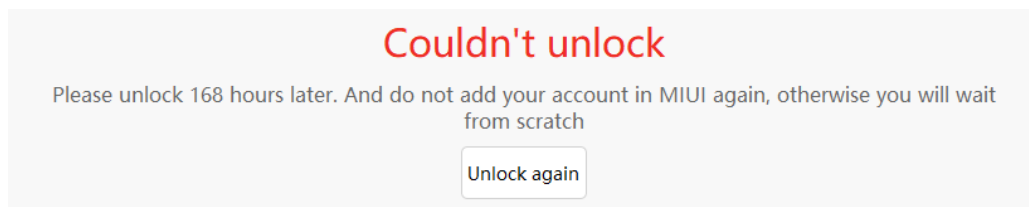
Bild 136: Volla Phone - Beschreiben des Gerätes mit dem Herstellerimage

## Xiaomi Redmi Note 8T – Ubuntu Touch

### 1. Entsperren des Bootloaders

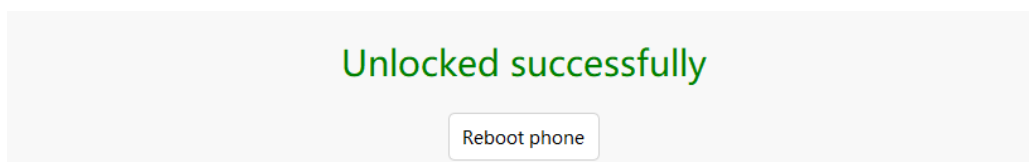
Das Entsperren von Xiaomi-Geräten erfordert meist die Verwendung des offiziellen Mi-Unlock-Tools. Zunächst wird auf dem vorinstallierten Android System (MIUI) der Entwicklermodus durch sieben Betätigungen des MIUI-Version-Buttons aktiviert.

Im neu erschienenen Entwicklermodus-Menü können die Einstellungen „USB-Debugging“ und „MI Unlock Status“ betätigt werden. Der Aufruf des Unlocks erfordert, dass ein MI-Konto mit dem Gerät verknüpft und eine SIM-Karte eingelegt wird. Anschließend kann das Gerät über die Tastenkombination **{leiser}** und **{Power}** in den Fastboot-Modus versetzt werden. Über das Mi Unlock Tool lässt sich der Entsperrvorgang anstoßen. Dies wird erst gestattet, wenn das verwendete Profil bereits vor mindestens 168 Stunden mit dem Gerät verknüpft wurde.



**Bild 137:** Redmi Note 8T - Frist zum Entsperren des Bootloaders

Nachdem diese Frist verstrichen war, konnte der Unlock durchgeführt werden.



**Bild 138:** Redmi Note 8T - Entsperren des Bootloaders

## 2. Installation von Ubuntu Touch und notwendiger Pakete

Einem Beitrag des Ubports Forums [181] folgend, wurde zunächst via Fastboot das original MIUI 11 Image (Android 9) Version 11.0.11.0 auf das Gerät geschrieben.

```
[prosch@christian-20b7s2qn02 willow_eea_global_images_V11.0.11.0.PCXEXUM_20200819.0000.00_9.0_eea]$ ./flash_all.sh
< waiting for any device >
Sending 'xbl' (3444 KB)                OKAY [ 0.087s]
Writing 'xbl'                        OKAY [ 0.022s]
Finished. Total time: 0.115s
Sending 'xblbak' (3444 KB)            OKAY [ 0.108s]
Writing 'xblbak'                     OKAY [ 0.022s]
Finished. Total time: 0.143s
Sending 'xbl_config' (48 KB)          OKAY [ 0.010s]
Writing 'xbl_config'                 OKAY [ 0.003s]
Finished. Total time: 0.019s
Sending 'xbl_configbak' (48 KB)       OKAY [ 0.007s]
Writing 'xbl_configbak'              OKAY [ 0.003s]
Finished. Total time: 0.017s
Sending 'imagefv' (20 KB)             OKAY [ 0.011s]
Writing 'imagefv'                   OKAY [ 0.002s]
Finished. Total time: 0.020s
Sending 'imagefvbak' (20 KB)          OKAY [ 0.010s]
Writing 'imagefvbak'                OKAY [ 0.002s]
Finished. Total time: 0.019s
```

**Bild 139:** Redmi Note 8T - Aufspielen des Android 9 Herstellerimages

Im Anschluss konnte die passende TWRP Recovery (twrp\_ginkgo-willow\_3.4A9\_23-12-2020\_BL.img) geflasht werden:

```
fastboot erase recovery
```

```
fastboot flash recovery twrp_ginkgo-willow_3.4A9_23-12- »
```

```
2020_BL.img
```

Ein Neustart über den Fastboot Modus führte zunächst in die original Mi Recovery. Es war erforderlich, nach dem Flashen von TWRP über die Kombination **{Lauter}** und **{Power}** direkt nach der Zwangsabschaltung die Recovery aufzurufen.

Aus der Recovery heraus wurde das Löschen der Data-Partition veranlasst und der adb-Sideload-Modus aufgerufen.

Über ADB konnten dann die im Forums-Beitrag verlinkten Pakete installiert werden:

```
adb sideload halium-ramdisk.zip
```

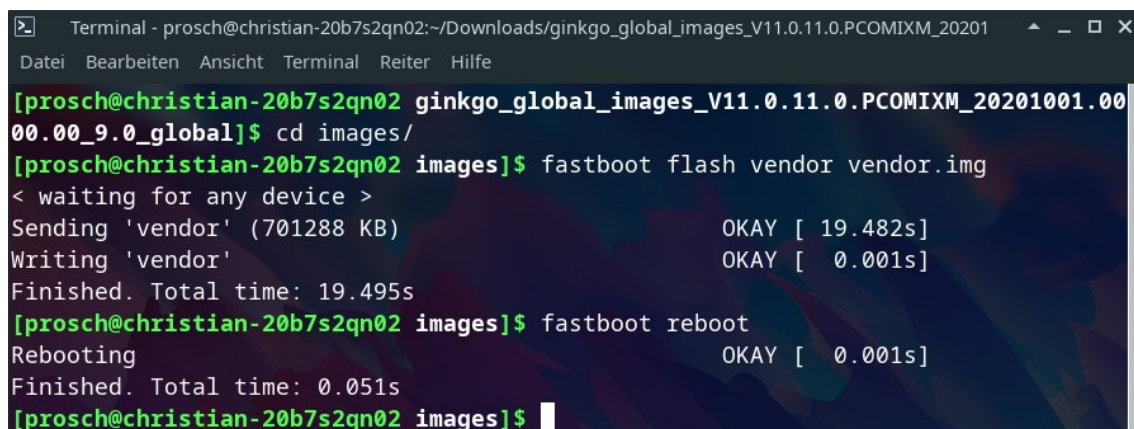
```
adb sideload ginkgo_Kernel_4.14-20201015-0455.zip
```

```
adb sideload ubports_GSI_installer_v10.zip
```

```
adb sideload ginkgo-postinstaller.zip
```

Ein Neustart des Gerätes führte zum Start von Ubuntu Touch. Im Rahmen der Ersteinrichtung konnte bislang jedoch keine WLAN-Verbindung hergestellt werden.

Dieses Fehlerbild wurde durch das Flashen der Vendor-Partition aus dem Herstellerimage des nahezu baugleichen Redmi Note 8 (Ginko) behoben.



```
Terminal - prosch@christian-20b7s2qn02:~/Downloads/ginkgo_global_images_V11.0.11.0.PCOMIXM_20201
Datei Bearbeiten Ansicht Terminal Reiter Hilfe

[prosch@christian-20b7s2qn02 ginkgo_global_images_V11.0.11.0.PCOMIXM_20201001.00
00.00_9.0_global]$ cd images/
[prosch@christian-20b7s2qn02 images]$ fastboot flash vendor vendor.img
< waiting for any device >
Sending 'vendor' (701288 KB)          OKAY [ 19.482s]
Writing 'vendor'                     OKAY [  0.001s]
Finished. Total time: 19.495s
[prosch@christian-20b7s2qn02 images]$ fastboot reboot
Rebooting                          OKAY [  0.001s]
Finished. Total time: 0.051s
[prosch@christian-20b7s2qn02 images]$
```

**Bild 140:** Redmi Note 8T - Flashen der Vendor-Partition des Redmi Note 8



# Samsung Galaxy S5 – PostmarketOS

## 1. Installation von PostmarketOS

Das Entsperren des Bootloaders war hier nicht notwendig. Über das Tool pmbootstrap wurde zunächst ein Image generiert:

```
pmbootstrap init
```

```
pmbootstrap install
```

Das Gerät wurde dann über die Tastenkombination {leiser} + {Home} + {Power} in den Download-Modus versetzt und der Befehl:

```
pmbootstrap flasher --method=heimdall-bootimg
```

```
flash_rootfs
```

ausgeführt. Gegen Ende des Vorgangs (Übertragung bei 100%) wurden am Gerät die Tasten {leiser} und {Home} gedrückt gehalten, um erneut den Download-Modus nach dem Neustart aufzurufen. Es wurde der Befehl:

```
pmbootstrap flasher --method=heimdall-bootimg
```

»

```
flash_kernel --no-install
```

abgesetzt.

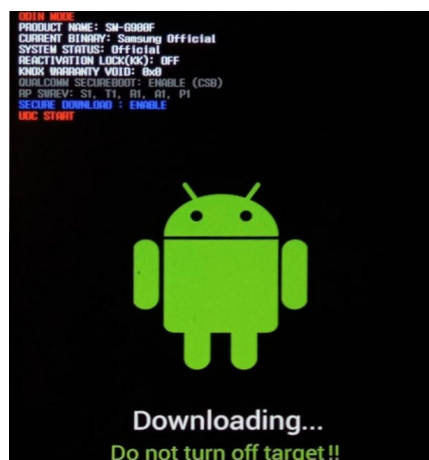


Bild 141: Galaxy S5 - Download Modus

## Xiaomi Mi A2 lite – Droidian

### 1. Installation des Android 9 Images

Das Gerät wurde mit bereits entsperrtem Bootloader erworben. Es konnten die Schritte einer Installationsanleitung aus dem XDA-Forum nachvollzogen werden [182]. Auch die benötigten Installationsdaten konnten aus diesem Thread bzw. entsprechenden Verlinkungen bezogen werden. Das gewählte Halium-Image basiert auf Halium 9. Entsprechend wurde die Firmware: daisy\_global\_images\_V10.0.13.0.PDLMIXM\_9.0 per fastboot (Tasten {leiser} und {Power}) über das Installationsskript: „flash-all.sh“ installiert.

### 2. Installation von Droidian

Das Gerät wurde erneut in den Fastboot-Modus versetzt und die alternative Recovery „OrangeFox-daisy-stable@R11.1.zip“ über den Befehl:

```
fastboot boot recovery.img
```

geladen. Hier konnten sämtliche Partitionen des Geräts gewählt und gewiped werden. Anschließend erfolgte eine Formatierung der Data-Partition.

Die Installation der alternativen Recovery und das Deaktivieren der Geräteverschlüsselung erfolgten im „Sideload“ Modus (am Gerät aktiviert) über die Aufrufe:

```
adb sideload OrangeFox-daisy-stable@R11.1.zip
```

```
adb sideload ForcedEncryptionDisablerDaisy.zip
```

Anschließend wurde wieder in die Recovery gebootet und das Partitionsfile „halium-boot-anbox.img“ per „adb push“ auf das Gerät übertragen:



```
adb push halium-boot-anbox.img /storage/
```

Über das Gerätemenü konnte das Image auf die Boot-Partition geschrieben werden. Das Droidian Image und die nötigen Patches konnten ebenfalls über Sideload installiert werden:

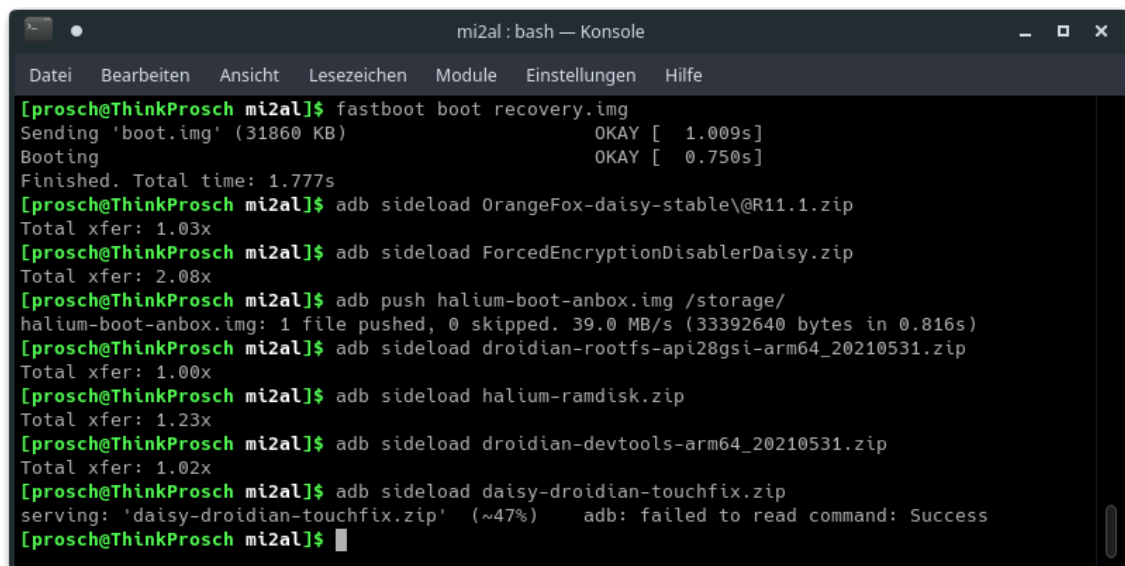
```
adb sideload droidian-rootfs-api28gsi-arm64_20210531.zip
```

```
adb sideload halium-ramdisk.zip
```

```
adb sideload droidian-devtools-arm64_20210531.zip
```

```
adb sideload daisy-droidian-touchfix.zip
```

Nach einem Neustart konnte der Droidian-Einrichtungsassistent genutzt werden.



```
mi2al : bash — Konsole
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe
[prosch@ThinkProsch mi2al]$ fastboot boot recovery.img
Sending 'boot.img' (31860 KB)          OKAY [  1.009s]
Booting                             OKAY [  0.750s]
Finished. Total time: 1.777s
[prosch@ThinkProsch mi2al]$ adb sideload OrangeFox-daisy-stable\@R11.1.zip
Total xfer: 1.03x
[prosch@ThinkProsch mi2al]$ adb sideload ForcedEncryptionDisablerDaisy.zip
Total xfer: 2.08x
[prosch@ThinkProsch mi2al]$ adb push halium-boot-anbox.img /storage/
halium-boot-anbox.img: 1 file pushed, 0 skipped. 39.0 MB/s (33392640 bytes in 0.816s)
[prosch@ThinkProsch mi2al]$ adb sideload droidian-rootfs-api28gsi-arm64_20210531.zip
Total xfer: 1.00x
[prosch@ThinkProsch mi2al]$ adb sideload halium-ramdisk.zip
Total xfer: 1.23x
[prosch@ThinkProsch mi2al]$ adb sideload droidian-devtools-arm64_20210531.zip
Total xfer: 1.02x
[prosch@ThinkProsch mi2al]$ adb sideload daisy-droidian-touchfix.zip
serving: 'daisy-droidian-touchfix.zip' (~47%)  adb: failed to read command: Success
[prosch@ThinkProsch mi2al]$
```

Bild 142: Mi A2 lite - Installation von Droidian

## Oneplus 3T – Droidian

### 1. Entsperren des Bootloaders

Zum Entsperren des Bootloaders genügt es, die OEM-Entsperrung in den Entwickleroptionen zu aktivieren. Diese werden auch bei diesem Gerät durch sieben Betätigungen der Buildnummer aktiviert. Anschließend wurde das Gerät über {leiser} und {Power} in den Fastboot-Modus versetzt und der Befehl:

```
fastboot oem unlock
```

abgesetzt. Am Gerät musste der Unlock lediglich bestätigt werden.

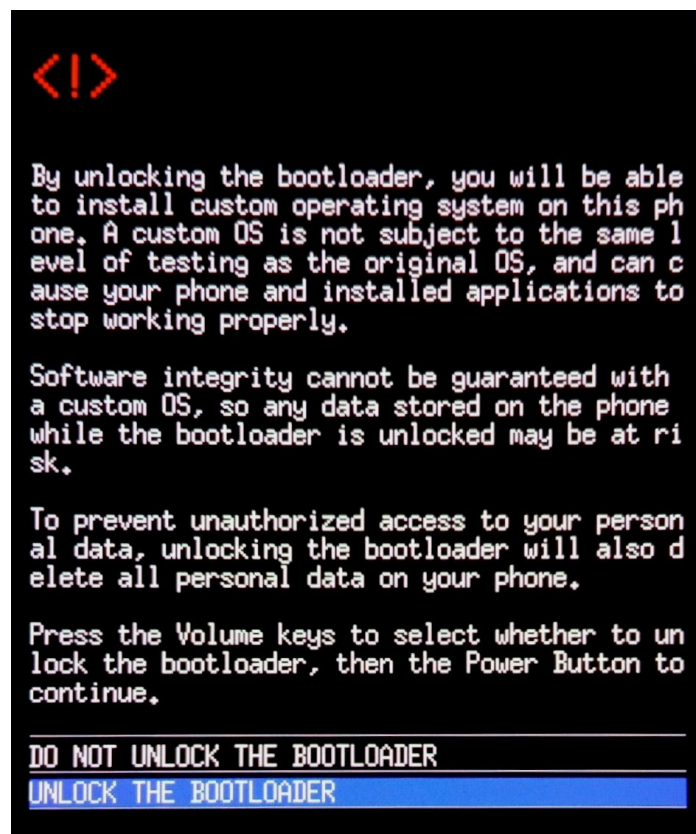


Bild 143: Oneplus 3T - Entsperren des Bootloaders

## 2. „Treblizing“ des Geräts und Installation von Droidian

Die Vorbereitung des Geräts und der eigentliche Installationsprozess können in einem XDA-Forenbeitrag [183] nachvollzogen werden. Hier werden auch benötigte Dateien bereitgestellt bzw. verlinkt. Zunächst wurde das Gerät auf die aktuelle Version von OxygenOS (9.0.6) aktualisiert. Über Fastboot (nach Entsperren des Bootloaders erreichbar über die Betätigung einer Lautstärketaste während des Warnhinweises nach dem Gerätestart) konnte eine Treble-kompatible Version der TWRP-Recovery installiert werden:

```
fastboot flash recovery twrp-op3treble-3.3.1-1.img
```

Im Fastboot-Bildschirm konnte dann über die Lautstärketasten der Recovery-Modus gewählt und unter dem Menüpunkt „Erweitert“ / „Advanced“ ein Terminal aufgerufen werden. Der Befehl:

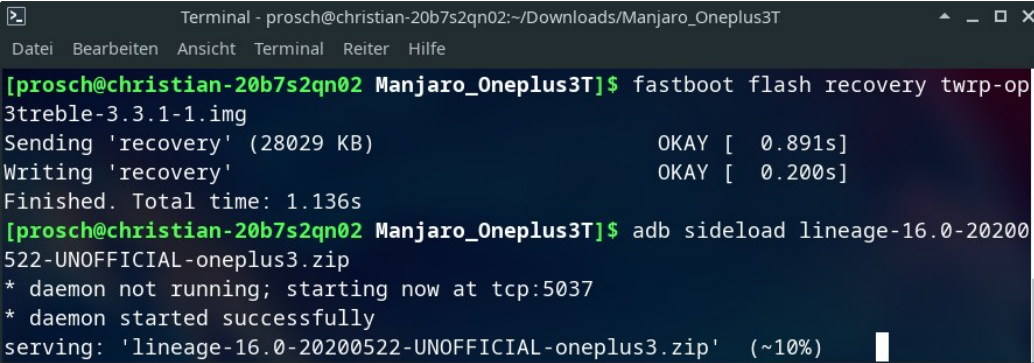
```
treblize
```

führte ein Script aus, über welches eine Vendor-Partition auf dem Gerät angelegt wurde. Nach einem Neustart des Geräts in die Recovery wurde eine Treble-kompatible Version der Custom Rom LineageOS 16 installiert:

```
adb sideload lineage-16.0-20190826-UNOFFICIAL - »
```

```
oneplus3.zip
```

und das Telefon in das installierte System neu gestartet. Hier wurden die Funktionen des Telefons kurz überprüft und anschließend erneut der Recovery-Modus aufgerufen.



```
Terminal - prosch@christian-20b7s2qn02:~/Downloads/Manjaro_Oneplus3T
Datei Bearbeiten Ansicht Terminal Reiter Hilfe

[prosch@christian-20b7s2qn02 Manjaro_Oneplus3T]$ fastboot flash recovery twrp-op
3treble-3.3.1-1.img
Sending 'recovery' (28029 KB)          OKAY [ 0.891s]
Writing 'recovery'                    OKAY [ 0.200s]
Finished. Total time: 1.136s
[prosch@christian-20b7s2qn02 Manjaro_Oneplus3T]$ adb sideload lineage-16.0-20200
522-UNOFFICIAL-oneplus3.zip
* daemon not running; starting now at tcp:5037
* daemon started successfully
serving: 'lineage-16.0-20200522-UNOFFICIAL-oneplus3.zip' (~10%)
```

**Bild 144:** OnePlus 3T - Installation von TWRP und LineageOS

Die Installation der Rom war zum Befüllen der Vendor-Partition (Gerätetreiber / Blobs etc.) erforderlich.

Über die Recovery wurden anschließend sämtliche Gerätepartitionen mit Ausnahme von „vendor“ über „Erweitertes Löschen“ gewiped. Abweichend von der XDA-Anleitung war es ebenfalls erforderlich, über „Format Data / Factory Reset“ die Datenpartition zu löschen. Hierbei wird die Geräteverschlüsselung aufgehoben. Nach einem weiteren Neustart in die Recovery konnten die benötigten Pakete per „adb push“ auf das Gerät übertragen und über TWRP installiert werden:

```
adb push droidian-OFFICIAL-phosh-phone-rootfs-api28- »
```

```
arm64-24_20220804.zip /tmp/
```

```
adb push halium-boot-img /tmp/
```

```
adb push op3-gsi-fix-droidian.zip /tmp/
```

Nach dem Start des Geräts wurde die Terminal-Anwendung geöffnet und:

```
move-home
```

abgesetzt. So wurde ein Skript aufgerufen, das den kompletten Speicher des Geräts für den Nutzer zugänglich macht.

# Samsung Galaxy S9 – Ubuntu Touch

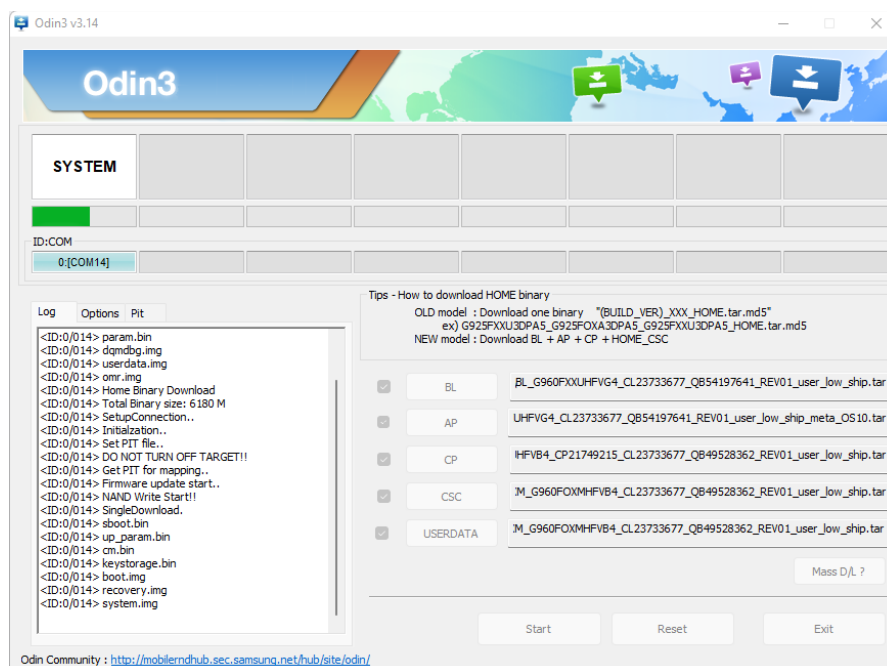
## 1. Entsperren des Bootloaders

Das Entsperren des Bootloaders konnte über den Menüpunkt: „OEM-Entsperrung“ in den Entwickleroptionen aktiviert werden. Diese werden auch bei diesem Gerät durch sieben Betätigungen der Buildnummer aktiviert. Erst nachdem eine korrekte Systemzeit eingestellt wurde, erschien der Button zum Entsperren des Bootloaders, welcher auch einen Werksreset auslöste.

Zusätzlich war der Menüpunkt „USB-Debugging“ zu aktivieren, da das Gerät sonst das Flashen unsignierter Images verweigert.

## 2. Flashen der erforderlichen Software-Version

Der Halium-Port für dieses Gerät setzt auf Halium 10 auf, so dass auch ein Android 10 Image für den Betrieb benötigt wird. Es wurde die Version G960FXXUHFVG4 gewählt und von [www.galaxyfirmware.com](http://www.galaxyfirmware.com) bezogen.



**Bild 145:** Galaxy S9 - Flashen von Android 10 mittels Odin

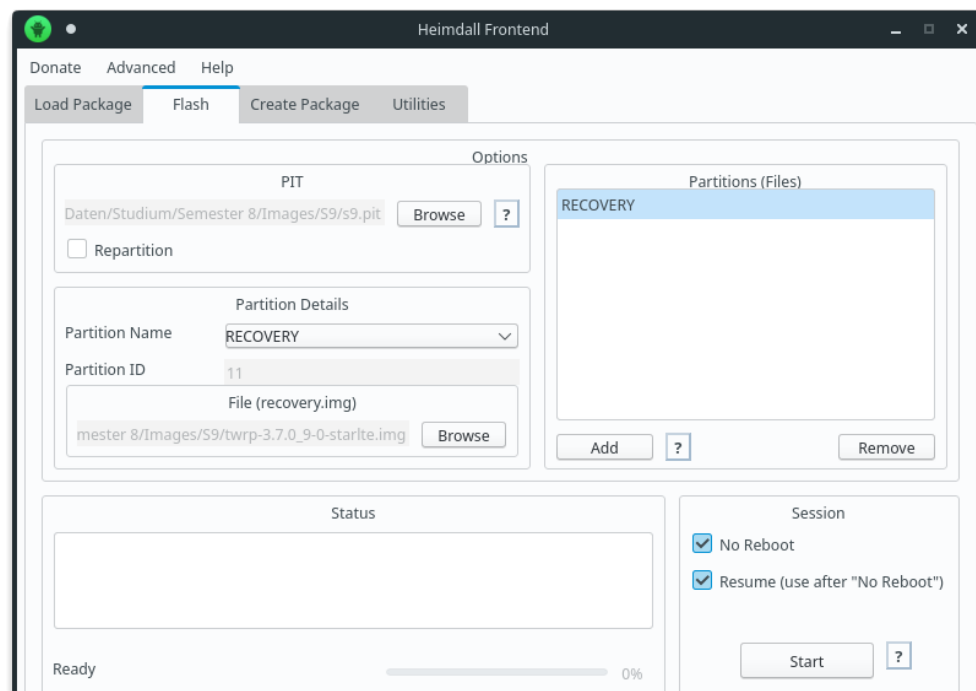
Zum Beschreiben des Gerätes mit der gewählten Firmware wurde dieses über die Tastenkombination **{leiser}**, **{Bixby}** und **{Power}** in den Download-Modus versetzt. Wegen des hohen Automatisierungsgrades der Software wurden die einzelnen Images über die Software „Odin 3.14“ unter Windows 11 aufgespielt (Bild 145).

Nach der Installation des Images wurde erneut der Menüpunkt „USB-Debugging“ aktiviert.

### 3. Installation von Ubuntu Touch

Die Installationsanleitung für Ubuntu Touch auf dem Galaxy S9 konnte im Telegram-Kanal „Ubuntu Touch & Linux for Samsung Galaxy S9/S9+/N9“ vorgefunden werden.

Zunächst erfolgte die Installation von TWRP 3.7.0. Das Image „twrp-3.7.0\_9-0-starlte.img“ konnte von der offiziellen TWRP-Seite (twrp.me) bezogen werden.



**Bild 146:** Galaxy S9 - Flashen von TWRP über Heimdall

Erneut wurde das Telefon über `{leiser}`, `{Bixby}` und `{Power}` in den Download-Modus versetzt und nun über die Software Heimdall das TWRP-Image auf die Recovery-Partition geschrieben (Bild [146]). Hierbei war das Auswahlfeld „No Reboot“ zu wählen, da im Rahmen des Neustarts über eine Systemreparatur TWRP mit der original Samsung-Recovery überschrieben werden würde.

Nach dem Flashen von TWRP wurde mit der Tastenkombination `{leiser}` und `{Power}` ein Reset des Geräts erzwungen und unmittelbar auf die Kombination `{lauter}`, `{Bixby}` und `{Power}` gewechselt, um in die TWRP-Recovery zu starten. Nach dem ersten Start wird das Überschreiben von TWRP unterbunden.

Die benötigten Images wurden über das Ubports Gitlab-Repository [184] als regelmäßig generierte Job-Artifacts bezogen. Es wurde die Datei „boot.img“ per Push-Befehl auf das Gerät übertragen:

```
adb push boot.img /tmp/
```

und über das TWRP-Menü auf die Boot-Partition geschrieben. Anschließend wurde im Löschmodenü der Menüpunkt „Format Data“ gewählt und mit der Eingabe „yes“ bestätigt. Zuletzt wurde das Image Ubuntu.img per Push auf /data übertragen und das Gerät in die Ubuntu Touch Oberfläche neu gestartet.

```
adb push ubuntu.img /data
```

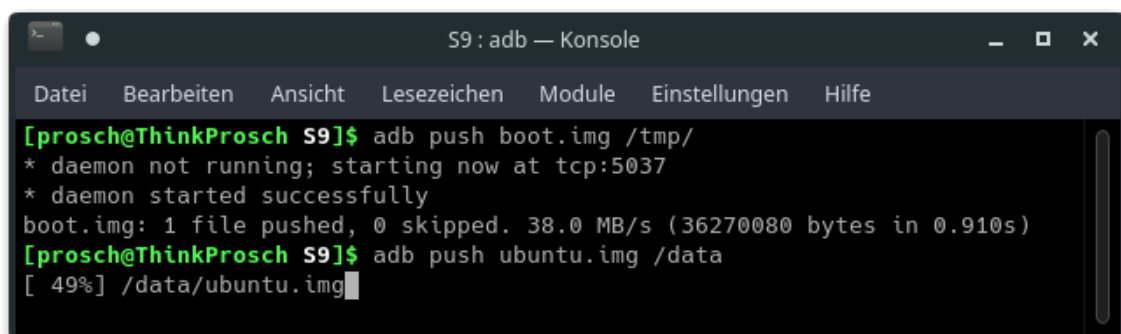


Bild 147: Galaxy S9 - Beschreiben mit Ubuntu Touch über TWRP und ADB

## HTC One M7 – Ubuntu Touch

### 1. Entsperren des Bootloaders

Die Website „htcdev.com“ bietet nach einer Registrierung kostenfrei die Möglichkeit, den Bootloader bestimmter HTC-Geräte zu entsperren. Am Gerät wurde dazu der Entwicklermodus und dort der Punkt „USB Debugging“ aktiviert. Über den Befehl:

```
adb reboot bootloader
```

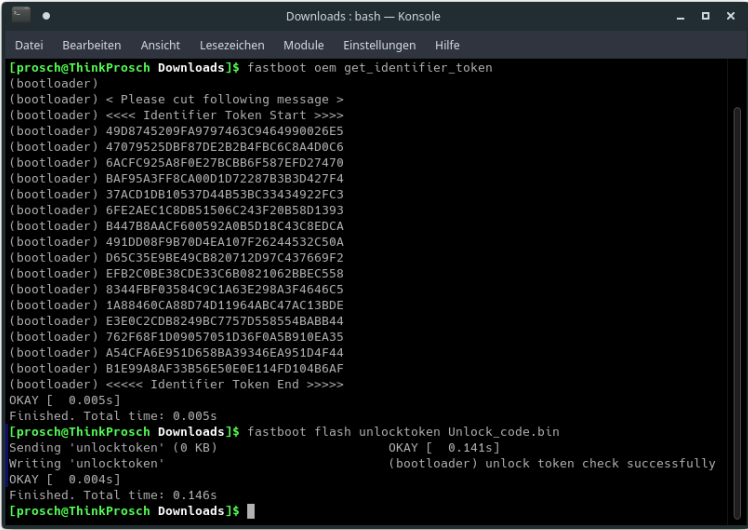
wurde das Gerät in den Fastboot Modus versetzt. Über die Eingabe:

```
fastboot oem get_identifier_token
```

wurde ein Token ausgegeben, welcher anschließend an die Seite htcdev.com übermittelt wurde. Der zugehörige Unlockschlüssel wurde anschließend per Mail zugestellt und konnte über den Befehl:

```
fastboot flash unlocktoken Unlock_code.bin
```

an das Telefon übertragen werden. Der Unlock wurde zuletzt auf dem Gerätedisplay bestätigt und somit ein Werksreset ausgelöst.



```
Downloads: bash — Konsole
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe
[prosche@ThinkProsch Downloads]$ fastboot oem get_identifier_token
(bootloader) < Please cut following message >
(bootloader) <<<< Identifier Token Start >>>>
(bootloader) 49D8745209FA9797463C9464990026E5
(bootloader) 470795250BF87DE2B2B4F8C6C8A4D0C6
(bootloader) 6ACFC925A8F0E278C8B6F587EFD27470
(bootloader) BAF95A3FF8CA00D1D728783B3D427F4
(bootloader) 37ACD1D810537D44B53BC33434922FC3
(bootloader) 6FE2AEC1C80B51506C243F20B5801393
(bootloader) B447B8AACF600592A0B5018C43C8EDCA
(bootloader) 491DD08F9B70D4EA107F26244532C50A
(bootloader) D65C35E9BE49C820712097C437669F2
(bootloader) EFB2C0BE38CE33C6B0821062BBEC558
(bootloader) 8344FBF03584C9C1A63E298A3F4646C5
(bootloader) 1A88460CA88074D11964ABC47AC13BDE
(bootloader) E3E0C2C0B82498C7757D5585548ABB44
(bootloader) 762F68F1D09057051D36F0A58910EA35
(bootloader) A54CFA6E951D658BA39346EA951D4F44
(bootloader) B1E99A8AF33B56E50E0E114F0104B6AF
(bootloader) <<<< Identifier Token End >>>>
OKAY [ 0.005s]
Finished. Total time: 0.005s
[prosche@ThinkProsch Downloads]$ fastboot flash unlocktoken Unlock_code.bin
Sending 'unlocktoken' (0 KB) OKAY [ 0.141s]
Writing 'unlocktoken' (bootloader) unlock token check successfully
OKAY [ 0.004s]
Finished. Total time: 0.146s
[prosche@ThinkProsch Downloads]$
```

Bild 148: HTC One - Entsperren des Bootloaders



## 2. Installation von Ubuntu Touch

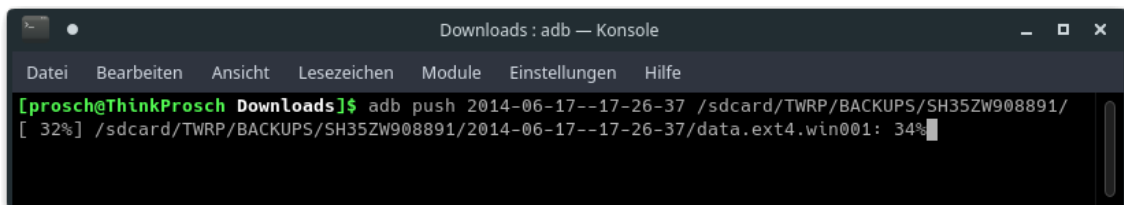
Das HTC M7 wird im Ubuntu Wiki noch immer als kompatibles Gerät geführt [185]. Installierbare Images konnten hier jedoch nicht bezogen werden. Über weiterführende Links konnte der XDA-Foreneintrag „Ubuntu Touch (latest)“ des Nutzers „ModestMouse1312“ gefunden werden. Dieser stellt über seinen Post ein TWRP-Backup des installierten Ubuntu Systems bereit, welches auf dem Testgerät wiederhergestellt werden konnte.

Voraussetzung war hier die Installation von TWRP (bezogen von twrp.me):

```
fastboot flash recovery twrp-3.0.0-2-m7wls.img
```

Aus dem Fastboot-Modus heraus konnte TWRP über den Menüpunkt „Recovery“ aufgerufen werden.

Per „adb push“ ließ sich das Backup auf das Gerät übertragen:



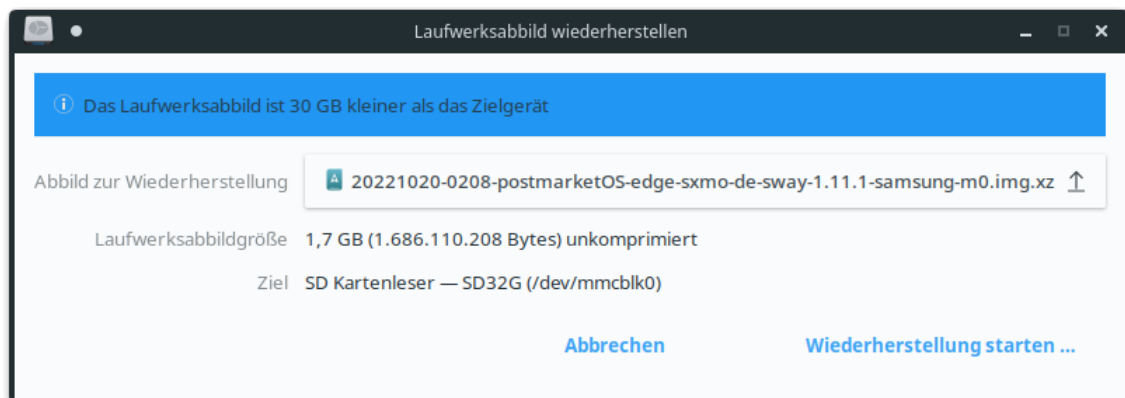
**Bild 149:** HTC One - Einspielen des Ubuntu Touch Backup Images

Zuletzt konnte das Backup über die TWRP-Funktion „Restore“ ausgewählt und auf den Speicher geschrieben werden.

# Samsung Galaxy S III – PostmarketOS

## 1. Beschreiben einer MicroSD-Karte

Gemäß der Anleitung der PostmarketOS-Projektseite [186] wird das rootfs-Image auf eine MicroSD-Karte gespielt. Dazu wurde eine Sandisk 32 GB MicroSD-Karte und das Tool Gnome Disks verwendet.



**Bild 150:** Samsung S III - Beschreiben der MicroSD-Karte

## 2. Beschreiben der Boot-Partition

Das erforderliche Boot-Image (zusammen mit dem SD-Karten-Image von der Projektseite bezogen) wurde mittels Heimdall auf das Mobiltelefon geschrieben. Zuvor wurde dieses über die Tastenkombination **{leiser} + {Home} + {Power}** in den Download-Modus versetzt. Anschließend konnte der Befehl:

```
heimdall flash --BOOT 20221020-0208-postmarketOS-edge-  »
sxmo-de-sway-1.11.1-samsung-m0-boot.img --verbose
```

ausgeführt werden. Nach dem Start des Systems konnte das Dateisystem über das Tool cfdisk auf die volle Größe erweitert werden. Eine funktionierende WLAN-Verbindung konnte durch das Einspielen der Firmware des Modells i9005 erreicht werden.

# Xiaomi Redmi 2 Pro – PostmarketOS

## 1. Flash der aktuellen Firmware

Die aktuelle Android 5.1 Firmware konnte über das Portal [xiaomifirmwareupdater.com](http://xiaomifirmwareupdater.com) bezogen und anschließend per Fastboot (Tasten **{leiser}** + **{Power}** am Gerät) über das enthaltene Skript „flash\_all.sh“ auf das Gerät gespielt werden.

## 2. Flash von TRWP, LineageOS, lk2nd und PostmarketOS

Der Installationsanleitung der offiziellen pmOS-Geräteseite [187] folgend, wurde zunächst das aktuelle TWRP-Image von der offiziellen Website [twrp.me](http://twrp.me) bezogen und im Fastboot Modus per:

```
fastboot flash recovery twrp-3.7.0_9-0-wt88047.img
```

übertragen. Über die Tastenkombination **{leiser}**, **{lauter}** und **{Power}** konnte TWRP aufgerufen werden. Hier wurde über den die Partitionen „Data“, „Cache“ und „System“ gewiped und LineageOS über Sideload installiert.

```
adb sideload lineage-14.1-20170101-UNOFFICIAL-wt88047.zip
```

Ein Neustart in den Bootloader-Modus erlaubte die Installation von lk2nd:

```
fastboot flash boot lk2nd-msm8916.img
```

Ein weiterer Neustart in lk2nd (**{leiser}** gedrückt halten nach dem Start) erlaubte die Installation von PostmarketOS:

```
fastboot flash userdata 20221019-1422-postmarketOS- »
```

```
v22.06-plasma-mobile-3.8-xiaomi-wt88047.img
```

```
fastboot erase system
```

# Xiaomi Pocophone F1 – Mobian

## 1. Entsperren des Bootloaders

Der Entsperrvorgang entspricht jenem des Redmi Note 8T (Seite AA).

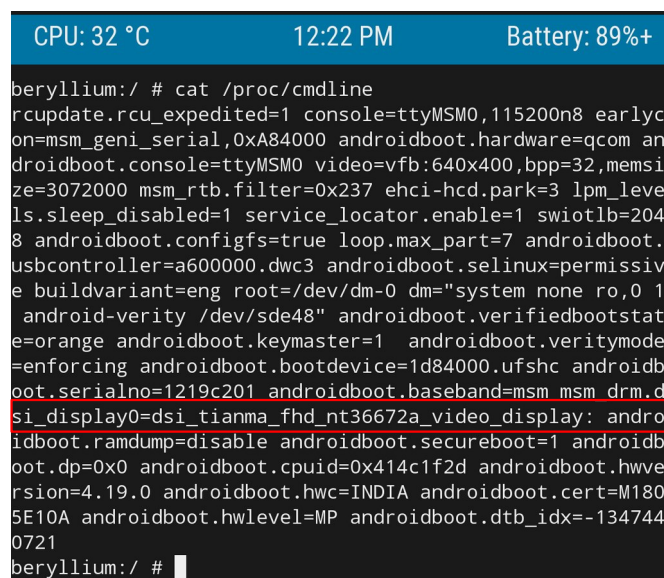
## 2. Bestimmung des Paneltyps und Installation von Mobian

Da im Pocophone F1 zwei unterschiedliche Kombinationen aus Panel und Digitizer verbaut wurden, welche Einfluss auf die zu verwendenden Treiber haben, wurde zunächst der Paneltyp ermittelt. Dazu wurde ein TWRP-Image für das Pocophone F1 von der offiziellen Projektseite [twrp.me](https://twrp.me) geladen und im Fastboot-Modus (Tasten **{leiser}** + **{Power}** am Gerät) übertragen:

```
fastboot flash recovery twrp-3.7.0_9-0beryllium.img
```

```
fastboot reboot recovery
```

Über TWRP konnte eine Konsole aufgerufen und die Datei „/proc/cmdline“ eingesehen werden:



```

CPU: 32 °C          12:22 PM          Battery: 89%+
beryllium:/ # cat /proc/cmdline
rcupdate.rcu_expedited=1 console=ttyMSM0,115200n8 earlycon=msm_geni_serial,0xA84000 androidboot.hardware=qcom androidboot.console=ttyMSM0 video=vfb:640x400,bpp=32,memsize=3072000 msm_rtb.filter=0x237 ehci-hcd.park=3 lpm_levels.sleep_disabled=1 service_locator.enable=1 swiotlb=2048 androidboot.configfs=true loop.max_part=7 androidboot.usbcontroller=a600000.dwc3 androidboot.selinux=permissive buildvariant=eng root=/dev/dm-0 dm="system none ro,0 1 android-verity /dev/sde48" androidboot.verifiedbootstate=orange androidboot.keymaster=1 androidboot.veritymode=enforcing androidboot.bootdevice=1d84000.ufshc androidboot.serialno=1219c201 androidboot.baseband=msm_msm_drm.dsi_display0=dsi_tianma_fhd_nt36672a_video_display: androidboot.ramdump=disable androidboot.secureboot=1 androidboot.dp=0x0 androidboot.cpuid=0x414c1f2d androidboot.hwversion=4.19.0 androidboot.hwc=INDIA androidboot.cert=M1805E10A androidboot.hwlevel=MP androidboot.dtb_idx=-1347440721
beryllium:/ #
  
```

**Bild 151:** Poco F1 - Ermittlung des Paneltyps via TWRP

Es ist erkennbar, dass im Testgerät ein Tianma-Panel verbaut wurde.

Entsprechend der Installationsanleitung des Mobian-Projekts [188] wurde zunächst das derzeit aktuelle Mobian-Image „mobian-pocof1-phosh-20221113.tar.gz“ geladen, entpackt und per Fastboot auf das Gerät geschrieben:

```
fastboot flash boot mobian-pocof1-phosh-20221113.boot-  »
```

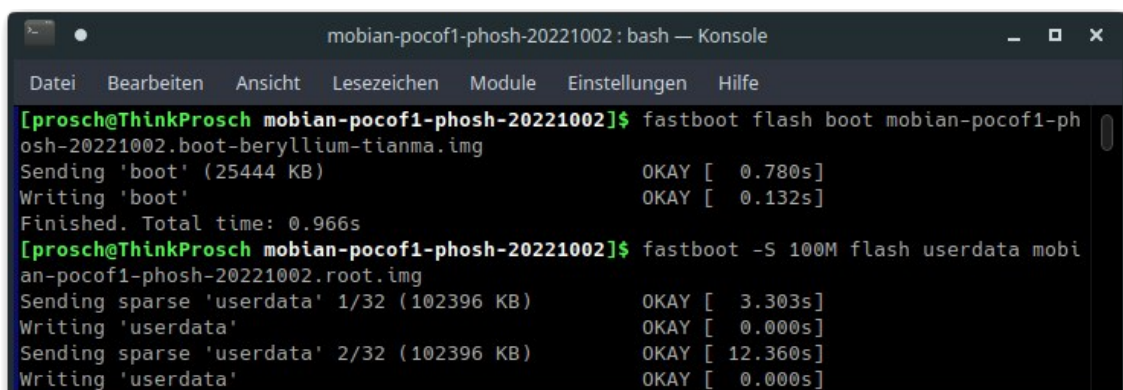
```
beryllium-tianma.img
```

```
fastboot -S 100 flash userdata mobian-pocof1-phosh-  »
```

```
20221113.root.img
```

Nach dem Gerätestart konnte jedoch kein Netzwerkinterface gefunden und somit keine WLAN-Verbindung hergestellt werden. Über eine Mitteilung an das Projekt wurde das Image als „broken“ markiert.

Es wurde eine erneute Installation mit dem Image: „mobian-pocof1-phosh-20221002.tar.gz“ durchgeführt. In Folge dieser Installation war eine Netzwerkverbindung möglich und es konnten über „apt-get“ die aktuellen Pakete installiert werden.



```
mobian-pocof1-phosh-20221002 : bash — Konsole
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe
[prosch@ThinkProsch mobian-pocof1-phosh-20221002]$ fastboot flash boot mobian-pocof1-ph
osh-20221002.boot-beryllium-tianma.img
Sending 'boot' (25444 KB)          OKAY [ 0.780s]
Writing 'boot'                    OKAY [ 0.132s]
Finished. Total time: 0.966s
[prosch@ThinkProsch mobian-pocof1-phosh-20221002]$ fastboot -S 100M flash userdata mobi
an-pocof1-phosh-20221002.root.img
Sending sparse 'userdata' 1/32 (102396 KB) OKAY [ 3.303s]
Writing 'userdata'                  OKAY [ 0.000s]
Sending sparse 'userdata' 2/32 (102396 KB) OKAY [ 12.360s]
Writing 'userdata'                  OKAY [ 0.000s]
```

**Bild 152:** Poco F1 - Installation von Mobian über Fastboot

# Xiaomi Redmi 4X – SailfishOS

## 1. Entsperren des Bootloaders

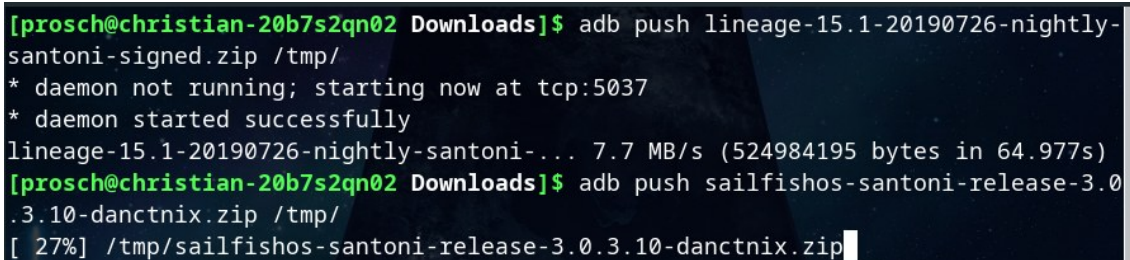
Der Entsperrvorgang entspricht jenem des Redmi Note 8T (Seite AA).

## 2. Flash von TWRP, LineageOS und Sailfish

Das ausgeschaltete Gerät wurde mit gedrückter **{leiser}** Taste an den Host-PC angeschlossen und das passende TWRP-Image (bezogen über twrp.me) per Fastboot auf das Gerät geschrieben:

```
fastboot flash recovery twrp-3.2.2-0-santoni.img
```

Der Installationsanleitung der Github Projektseite [189] des Ports folgend, wurde das Smartphone über die Tastenkombination **{lauter}** + **{Power}** in die Recovery gebootet. Im folgenden Menü konnte über den Aufruf „recovery 模式“ TWRP aufgerufen werden. Die Partitionen Dalvik, Data, Cache und System wurden über das erweiterte Löschmenü formatiert und die Lineage- und Sailfish-Images per „adb push“ auf das Gerät übertragen:



```
[prosch@christian-20b7s2qn02 Downloads]$ adb push lineage-15.1-20190726-nightly-santoni-signed.zip /tmp/
* daemon not running; starting now at tcp:5037
* daemon started successfully
lineage-15.1-20190726-nightly-santoni-... 7.7 MB/s (524984195 bytes in 64.977s)
[prosch@christian-20b7s2qn02 Downloads]$ adb push sailfishos-santoni-release-3.0.3.10-danctnix.zip /tmp/
[ 27%] /tmp/sailfishos-santoni-release-3.0.3.10-danctnix.zip
```

**Bild 153:** Redmi 4x - Übertragen der Image-Dateien

Die übertragenen Images konnten über TWRP installiert werden. Dabei erfolgte zunächst die Installation von LineageOS 15 und im Anschluss die Installation von SailfishOS 3.0.3.



## Xiaomi Redmi 9C – Droidian

### 1. Entsperren des Bootloaders

Der Entsperrvorgang entspricht jenem des Redmi Note 8T (Seite AA).

### 2. Installation von Droidian

Es wurde der Anleitung des Users „FakeShell“ aus dem XDA-Forum [191] gefolgt. Hierzu wurden die hinterlegten Dateien heruntergeladen und per Fastboot (Tasten **{leiser}** + **{Power}** am Gerät) auf das Gerät geschrieben:

```
fastboot flash boot boot-dandelion.img
```

```
fastboot flash dtbo dtbo-dandelion.img
```

```
fastboot --disable-verity --disable-verification flash »  
vbmeta vbmeta-dandelion.img
```

```
fastboot flash recovery OrangeFox-R11-garden-droidian.img
```

```
fastboot format:ext4 userdata
```

Die installierte Recovery konnte über:

```
fastboot reboot recovery
```

aufgerufen werden. Hier wurde über den ADB-Sideload-Modus Droidian installiert:

```
adb sideload droidian-OFFICIAL-phosh-phone-rootfs-api29-armhf-»  
nightly_20221229.zip
```

```
adb sideload adaptation-droidian-garden.zip
```



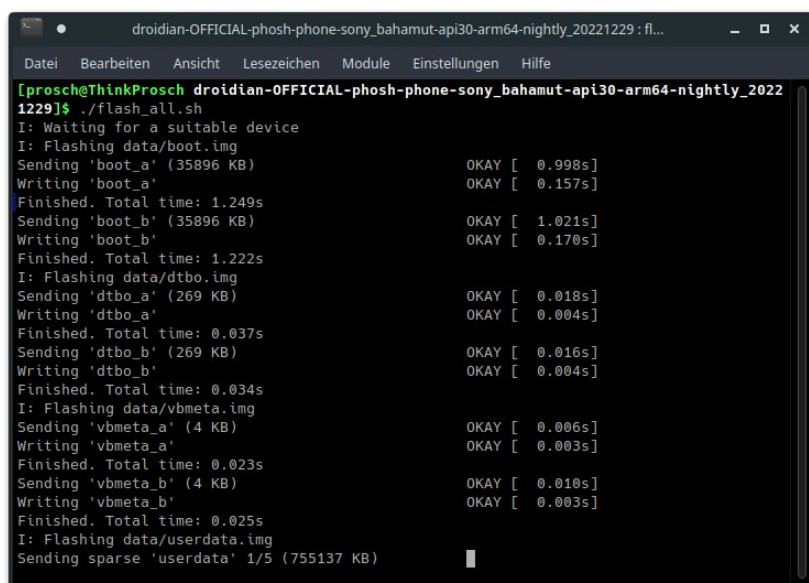
## Sony Xperia 5 – Droidian

### 1. Bootloader unlocken

Die Unlockprozedur entspricht der des Xperia XA2 (Seite: Q). Hierbei wurde selbstverständlich ein anderer Unlockcode generiert.

### 2. Installation von Droidian

Das offizielle „nightly“ Droidian-Image konnte von der Github-Projektseite [56] bezogen werden. Das Image droidian-OFFICIAL-phosh-phone-sony\_bahamut-api30-arm64-nightly\_20230101.zip wurde entpackt und das enthaltene Skript: flash\_all.sh ausgeführt. Anschließend wurde das ausgeschaltete Gerät mit gedrückter **{lauter}** Taste über USB an den PC angeschlossen. Dabei wurde die Taste so lange gedrückt gehalten, bis die LED des Geräts von rot auf blau wechselte. Nach Beendigung des Installationsskripts startete das Smartphone neu und zeigte den Droidian-Sperrbildschirm. Über die Verknüpfung „Device encryption“ konnte die Geräteverschlüsselung aktiviert werden. Hierbei wurde das Passwort „753159“ gewählt.



```
droidian-OFFICIAL-phosh-phone-sony_bahamut-api30-arm64-nightly_20221229 : fl...
Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe
[prosche@ThinkProsch droidian-OFFICIAL-phosh-phone-sony_bahamut-api30-arm64-nightly_20221229]$ ./flash_all.sh
I: Waiting for a suitable device
I: Flashing data/boot.img
Sending 'boot_a' (35896 KB) OKAY [ 0.998s]
Writing 'boot_a' OKAY [ 0.157s]
Finished. Total time: 1.249s
Sending 'boot_b' (35896 KB) OKAY [ 1.021s]
Writing 'boot_b' OKAY [ 0.170s]
Finished. Total time: 1.222s
I: Flashing data/dtbo.img
Sending 'dtbo_a' (269 KB) OKAY [ 0.018s]
Writing 'dtbo_a' OKAY [ 0.004s]
Finished. Total time: 0.037s
Sending 'dtbo_b' (269 KB) OKAY [ 0.016s]
Writing 'dtbo_b' OKAY [ 0.004s]
Finished. Total time: 0.034s
I: Flashing data/vbmeta.img
Sending 'vbmeta_a' (4 KB) OKAY [ 0.006s]
Writing 'vbmeta_a' OKAY [ 0.003s]
Finished. Total time: 0.023s
Sending 'vbmeta_b' (4 KB) OKAY [ 0.010s]
Writing 'vbmeta_b' OKAY [ 0.003s]
Finished. Total time: 0.025s
I: Flashing data/userdata.img
Sending sparse 'userdata' 1/5 (755137 KB)
```

Bild 155: Xperia 5 - Installation von Droidian über Fastboot

## Microsoft Surface 2 – Ubuntu Mate 22.04

### 1. Secure Boot deaktivieren

Das Deaktivieren von Secure Boot wurde seitens Microsoft durch diverse Patches unterbunden. Daher wurde das Recovery-Image: Surface2\_BMR\_20.2.19.0.zip von der Microsoft-Seite (<https://support.microsoft.com/en-us/surface-recovery-image>) bezogen und auf einen USB-Stick entpackt. Das Surface wurde im Anschluss unter Verwendung der Tastenkombination **{leiser}** + **{Power}** vom USB-Stick gestartet und Windows RT neu installiert. Nach der Installation wurde der Anleitung des Portals „Open RT Community Gitbook“ [136] gefolgt, um zunächst den „Golden Keys“ Jailbreak anzuwenden. Dazu wurde das Paket „SecureBoot.zip“ auf dem Gerätespeicher des Surface Tablets entpackt und das Script: InstallPolicy.cmd als Administrator ausgeführt. Während des folgenden Neustarts wurde die Installation der Debug Policies bestätigt. Der Test Modus konnte nun über die Befehle:

```
bcdedit /set {bootmgr} testsigning on
```

und

```
bcdedit /set {default} testsigning on
```

aktiviert werden.

Um nun Secure Boot zu deaktivieren, wurde die ebenfalls von der Open RT Projektseite bezogene Datei „yahahallo.zip“ auf einen USB-Stick entpackt und das Surface erneut über **{leiser}** + **{Power}** vom USB-Stick gestartet. Nach der Ausführung der Unlock-Befehle wurde das Gerät nach Aufforderung neu gestartet.

Das Windows-Boardtool „msinfo32.exe“ vermeldete nun, dass Secure Boot deaktiviert wurde (Bild 156).

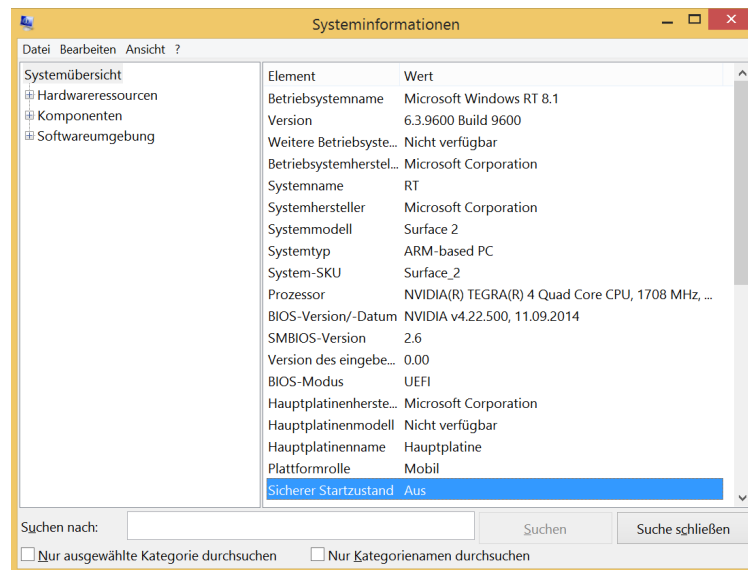


Bild 156: Surface 2 - Secure Boot wurde deaktiviert

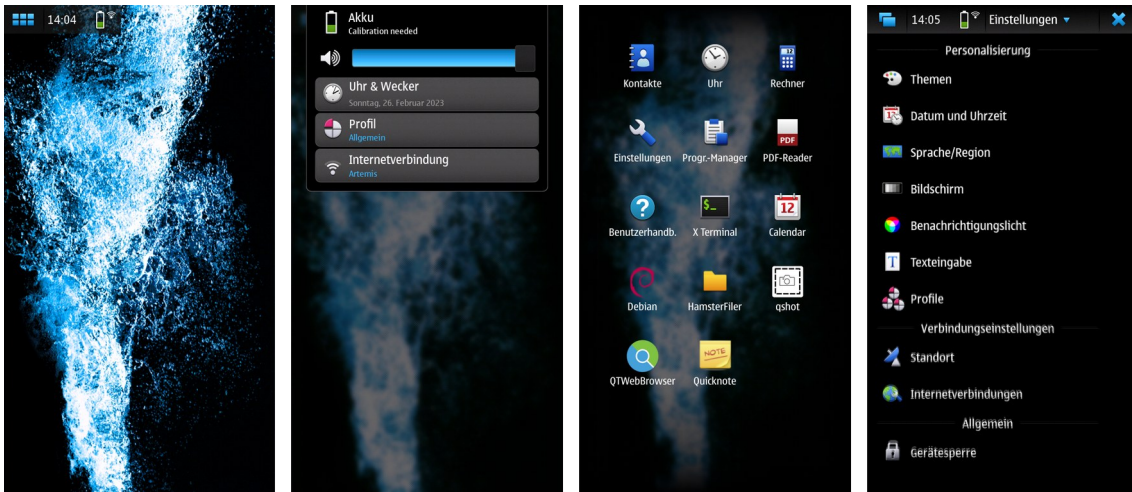
## 2. Installation von Ubuntu

Das Ubuntu Mate 22.04 armhf-Image (für einen Raspberry Pi) wurde von der offiziellen Website [192] bezogen und mittels Balena Etcher auf eine MicroSD-Karte (SanDisk 64GB) geschrieben. Die nötigen Kernel-Module und ein vorkompilierter Kernel konnten aus der Open RT Discord-Gruppe [137] bezogen werden. Diese wurden auf einen FAT32-formatierten USB-Stick kopiert. Das enthaltene „lib“ Verzeichnis wurde zudem in der EFI-Partition der erstellten MicroSD-Karten-Installation abgelegt. Die MicroSD-Karte und der Bootstick wurden in das Surface eingesetzt und das Gerät erneut über **{leiser}** + **{Power}** vom USB-Stick gestartet. Nach der erfolgten Ersteinrichtung wurden die benötigten Treibermodule über den Befehl:

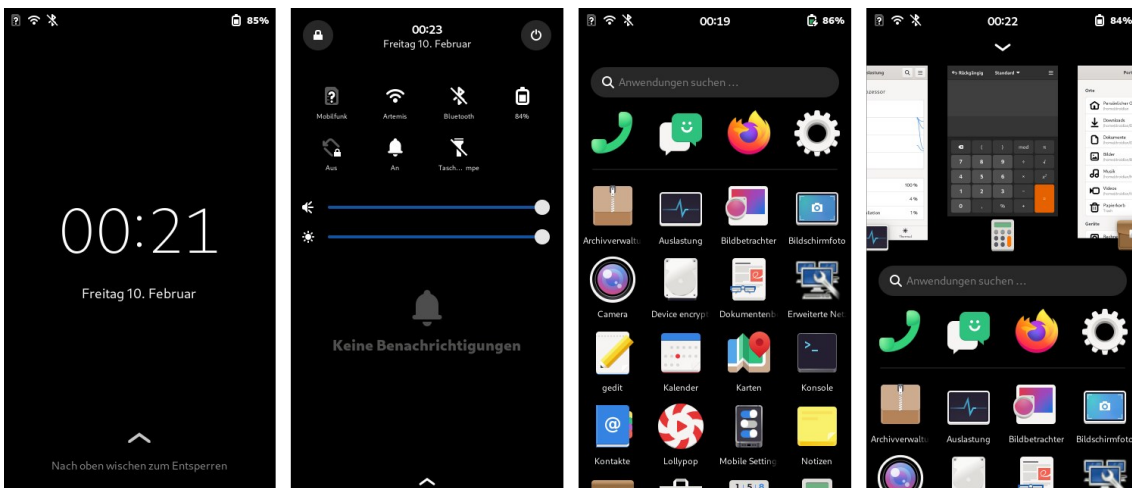
```
cp /boot/firmware/lib/modules/* -rv /lib/modules
```

in das System kopiert. Mittels „dd“ wurden die Partitionen des USB-Sticks und der MicroSD-Karte auf den eMMC des Tablets übertragen und die cmdline.txt in der EFI-Boot-Partition entsprechend angepasst, um auf die rootfs-Partition zu verweisen (Hier mmcblk0p4).

## Anhang 2: mobile Oberflächen



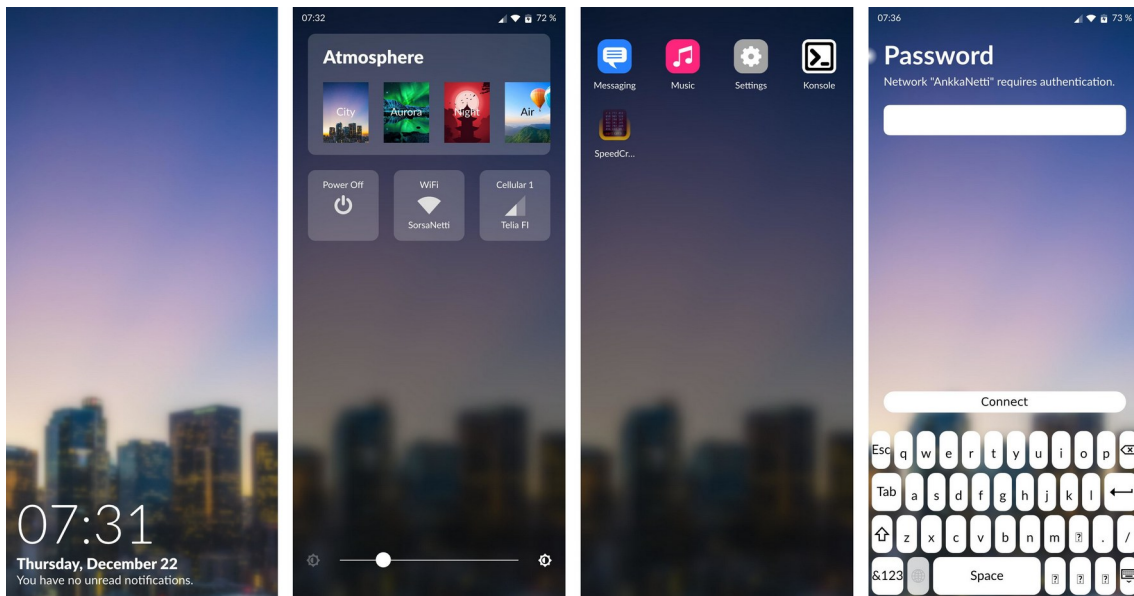
**Bild 157:** Hildon-Oberfläche unter Maemo Leste



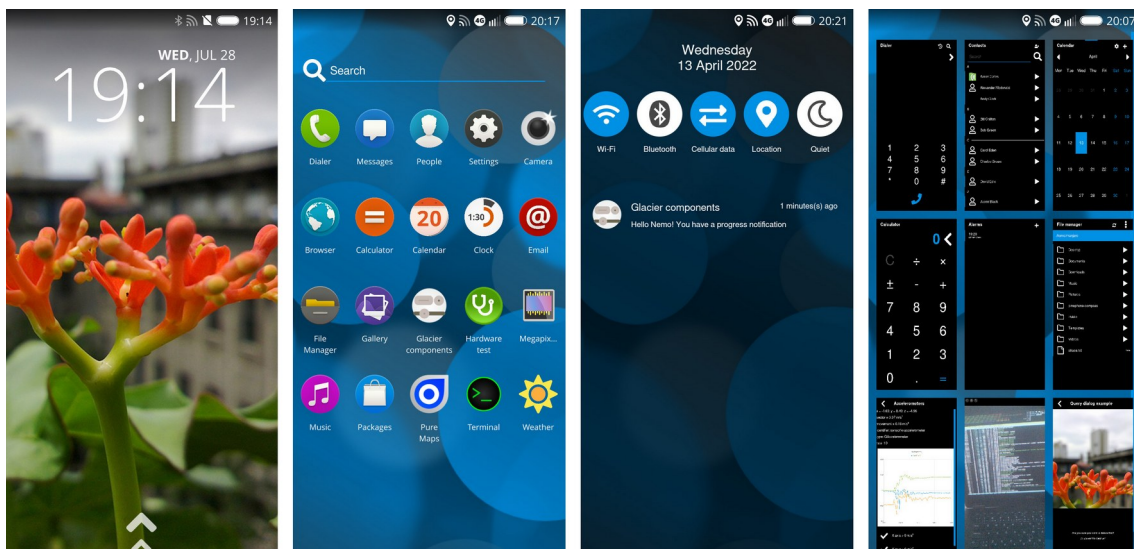
**Bild 158:** Phosh-Oberfläche unter Droidian (Phosh Version 0.24)



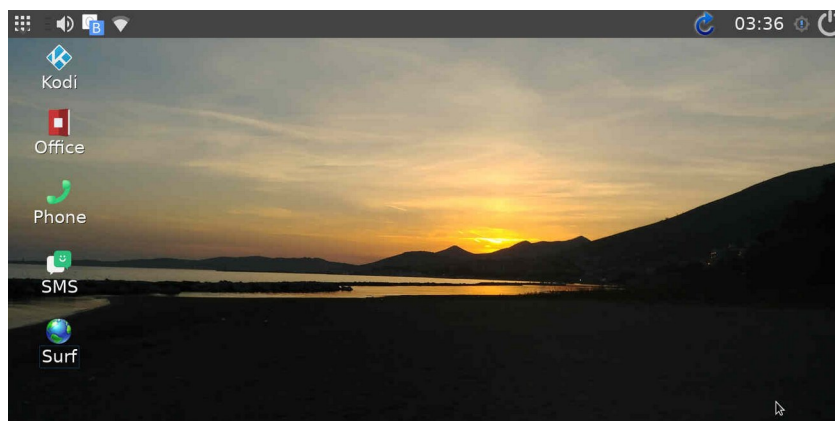
**Bild 159:** Gnome Mobile unter PostmarketOS



**Bild 160:** Cutie-Shell (Bildquelle: <https://cutie-shell.org/screenshots>)

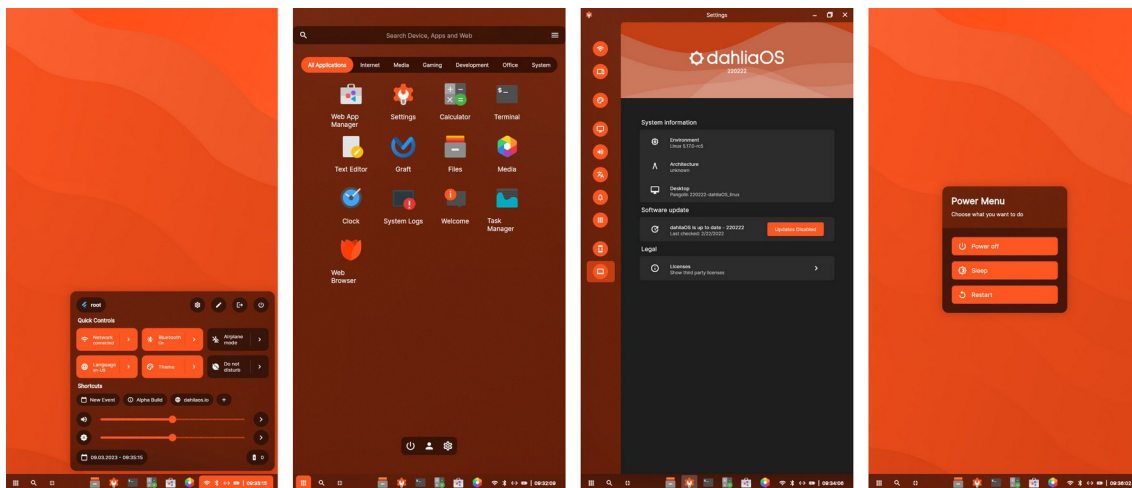


**Bild 161:** Glacier-Oberfläche (Bildquelle: <https://nemomobile.net/glacier-home/>)



**Bild 162:** angepasster Mate-Desktop (Bildquelle: <https://archivista.ch>)

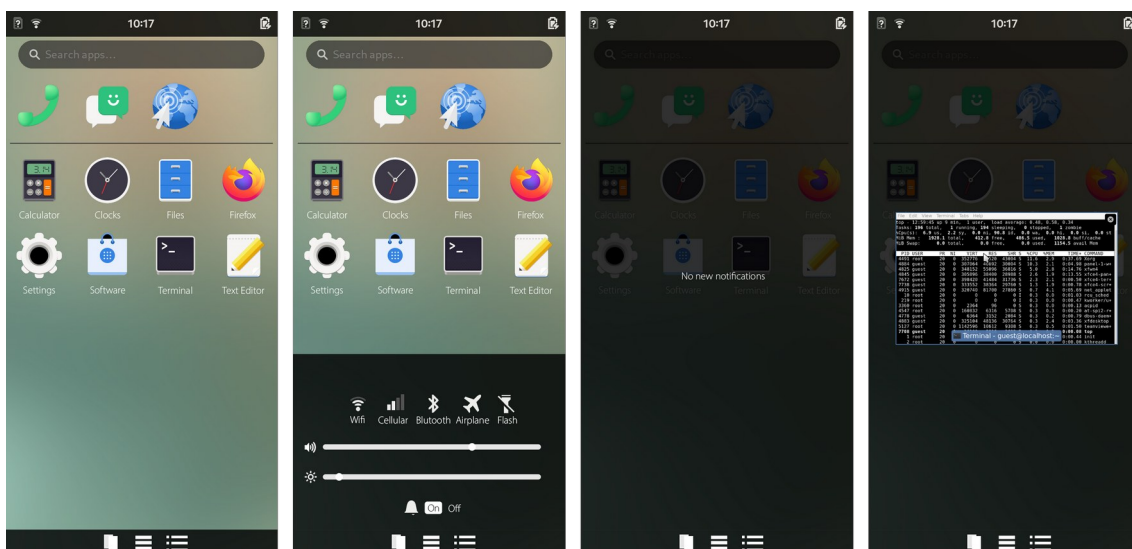




**Bild 163:** Pangolin UI unter dahliaOS (VM)



**Bild 164:** Maui Shell (Bildquelle: <https://github.com/Nitrux/maui-shell>)



**Bild 165:** tuna (Bildquelle: <https://github.com/Gusaroot/tuna/blob/main/screenshots/>)

## Anhang 3: Übersicht der Sicherungen

### TCP-Live-Sicherungen

Gerät	System	Protokoll	Transfer	Speicher	Backup	Dauer
Redmi 2	PostmarketOS	SSH	USB	14 GiB	2,4 GiB	22min 29s
Surface 2	Ubuntu Mate	Netcat	WiFi	29 GiB	5 GiB	2h 58min 9s
AuroraVM	AuroraOS	SSH	localhost	8 GiB	368 MiB	5m 42s
Pinephone	Manjaro KDE	SSH	WiFi	14 GiB	2,8 GiB	1h 2min 40s
Nokia N9	MeeGo	SSH	USB	14 GiB	2,6 GiB	33min 16s
Nexus 5	Ubuntu Touch	SSH	WiFi	14 GiB	1,9 GiB	46min 11s
Aquaris E5	Ubuntu Touch	SSH	WiFi	14 GiB	1007 MiB	49min 10s
Xperia XA2	SailfishOS	SSH	USB	29 GiB	3,7 GiB	19min 39s
Moto G	SailfishOS	SSH	USB	7,2 GiB	3 GiB	13min 40s
A6000	PostmarketOS	SSH	USB	7,2 GiB	3 GiB	11min 40s
Droid 4	Maemo Leste (µSD)	SSH	WiFi	29 GiB	1,2 GiB	3h 23min 50s
TF300T	PostmarketOS (µSD)	SSH	USB	29 GiB	992 MiB	50min 11s
T100TA	Mobian / Debian	SSH	WiFi	58 GiB	8,7 GiB	6h 16min 14s
GS290	Ubuntu Touch	SSH	WiFi	58 GiB	2,4 GiB	5h 57min 37s
Redmi N 8T	Ubuntu Touch	SSH	WiFi	58 GiB	4,3 GiB	1h 55min 51s
Galaxy S5	PostmarketOS	SSH	WiFi	14 GiB	1,3 GiB	20min 55s
Mi A2 Lite	Droidian	SSH	WiFi	29 GiB	3,3 GiB	2h 16min 31s
Oneplus 3T	Droidian	N/A	-	58 GiB	-	-
Galaxy S9	Ubuntu Touch	SSH	WiFi	59 GiB	4,6 GiB	2h 22min 30s
HTC One	Ubuntu Touch	SSH	WiFi	29 GiB	2,5 GiB	50min 4s
Galaxy S3	PostmarketOS (µSD)	SSH	USB	29 GiB	731 MiB	42min 56s
Poco F1	Mobian	N/A	-	58 GiB	-	-
Redmi 4X	SailfishOS	SSH	USB	29 GiB	1,5 GiB	24min 43s
Venue 8 Pro	SailfishOS	SSH	WiFi	29 GiB	7 GiB	9h 14min 66s
Redmi 9C	Droidian	SSH	USB	29 GiB	4,5 GiB	31min 26s
Xperia 5	Droidian	SSH	USB	119 GiB	11,7 GiB	1h 44min 11s

### TCP-Live Sicherung (unverschlüsselt)

Gerät	Pfad	Speicher	Backup	Dauer
Xperia XA2	/dev/mapper/luks-5da0b92b-46d0-4445-9ef7-5b89c10bd104	17 GiB	17 GiB	22min 29s
Xperia 5	/dev/mapper/droidian_encrypted	107 GiB	103 GiB	1h 39min 27s

## ADB-Sicherungen (Ubuntu Touch)

Gerät	System	Protokoll	Transfer	Speicher	Backup	Dauer
Nexus 5	Ubuntu Touch	ADB	USB	14 GiB	1,9 GiB	54min 46s
Aquaris E5	Ubuntu Touch	ADB	USB	14 GiB	976,1 MiB	51min 3s
GS290	Ubuntu Touch	ADB	USB	58 GiB	2,3 GiB	1h 30min 25s
Redmi N 8T	Ubuntu Touch	N/A	-	58 GiB	-	-
Galaxy S9	Ubuntu Touch	N/A	-	59 GiB	-	-
HTC One	Ubuntu Touch	ADB	USB	29 GiB	Fehler	-

## Recovery-Sicherungen

Gerät	System	Installation	Recovery	Speicher	Backup	Dauer
Redmi 2	PostmarketOS	vorinstalliert	TWRP	14 GiB	2,4 GiB	15min 57s
Surface 2	Ubuntu Mate	N/A	-	29 GiB	-	-
Pinephone	Manjaro KDE	N/A	-	14 GiB	-	-
Nokia N9	MeeGo	N/A	-	14 GiB	-	-
Nexus 5	Ubuntu Touch	vorinstalliert	UBPorts	14 GiB	1,9 GiB	42min 40s
Aquaris E5	Ubuntu Touch	live	TWRP	14 GiB	976,2 MiB	46min 22s
Xperia XA2	SailfishOS	live	TWRP	29 GiB	3,7 GiB	20min 33s
Moto G	SailfishOS	vorinstalliert	TWRP	7,2 GiB	3 GiB	1h 22min 43s
A6000	PostmarketOS	live	TWRP	7,2 GiB	Fehler	-
Droid 4	Maemo Leste (µSD)	N/A	-	29 GiB	-	-
TF300T	PostmarketOS (µSD)	vorinstalliert	TWRP	29 GiB	995,8 MiB	2h 14min 52s
T100TA	Mobian / Debian	N/A	-	58 GiB	-	-
GS290	Ubuntu Touch	vorinstalliert	UBPorts	58 GiB	2,3 GiB	1h 44min 44s
Redmi N 8T	Ubuntu Touch	vorinstalliert	TWRP	58 GiB	4,3 GiB	1h 13min 41s
Galaxy S5	PostmarketOS	Installiert	TWRP	14 GiB	1,4 GiB	55min 20s
Mi A2 Lite	Droidian	live	TWRP	29 GiB	3,3 GiB	46min 20s
Oneplus 3T	Droidian	vorinstalliert	TWRP	58 GiB	5,6 GiB	51min 42s
Galaxy S9	Ubuntu Touch	vorinstalliert	TWRP	59 GiB	4,6 GiB	41min 57s
HTC One	Ubuntu Touch	vorinstalliert	TWRP	29 GiB	2,5 GiB	2h 28min 0s
Galaxy S3	PostmarketOS (µSD)	installiert	TWRP	29 GiB	726,4 MiB	51min 50s
Poco F1	Mobian	vorinstalliert	TWRP	59 GiB	13,2 GiB	56min 12s
Redmi 4X	SailfishOS	live	TWRP	29 GiB	1,5 GiB	19min 44s
Venue 8 Pro	SailfishOS	N/A	-	29 GiB	-	-
Redmi 9C	Droidian	vorinstalliert	OrangeFox	29 GiB	4,5 GiB	39min 35s
Xperia 5	Droidian	Installiert	Lineage	119 GiB	12,2 GiB	1h 4min 27s



## Bootloader-Sicherungen

Gerät	SoC	Modus	Aufruf	Speicher	Backup	Dauer
Redmi 2	Qualcomm	EDL	laut + leise + USBin	14 GiB	2,4 GiB	18min 57s
Surface 2	Nvidia	APX	laut halten + power	29 GiB	5,4 GiB	53min 52s
Pinephone	Allwinner	FEL	Testpoint	14 GiB	2,6 GiB	28min 24s
Nokia N9	Texas Instr.	-	-	14 GiB	-	-
Nexus 5	Qualcomm	EDL	laut + leise + power	14 GiB	Fehler	-
Aquaris E5	MediaTek	BROM	laut + leise + USBin	14 GiB	972,6 MiB	2h 36m 57s
Xperia XA2	Qualcomm	-	-	29 GiB	-	-
Moto G	Qualcomm	-	-	7,2 GiB	-	-
A6000	Qualcomm	EDL	laut + leise + USBin	7,3 GiB	3,0 GiB	8min 28s
Droid 4	Texas Instr.	-	-	29 GiB	-	-
TF300T	Nvidia	APX	Laut + USBin	29 GiB	995,8 MiB	2h 14min 36s
T100TA	Intel	-	-	58 GiB	-	-
GS290	MediaTek	BROM	laut + leise + USBin	58 GiB	2,3 GiB	1h 34min 53s
Redmi N 8T	Qualcomm	EDL	Testpoint	58 GiB	4,3 GiB	35min 41s
Galaxy S5	Qualcomm	-	-	14 GiB	-	-
Mi A2 Lite	Qualcomm	EDL	Testpoint	29 GiB	Fehler	-
Oneplus 3T	Qualcomm	EDL	laut + leise + USBin	58 GiB	5,6 GiB	1h 31min 36s
Galaxy S9	Exynos	-	-	59 GiB	-	-
HTC One	Qualcomm	-	-	29 GiB	-	-
Galaxy S3	Exynos	-	-	29 GiB	-	-
Poco F1	Qualcomm	EDL	Fastboot	59 GiB	Fehler	-
Redmi 4X	Qualcomm	EDL	Testpoint	29 GiB	1,5 GiB	37min 44s
Venue 8 Pro	Intel	-	-	29 GiB	-	-
Redmi 9C	MediaTek	BROM	laut + leise + USBin	29 GiB	4,6 GiB	47m 42s
Xperia 5	Qualcomm	-	-	119 GiB	-	-

## Anhang 4: Testpunkte

### Xiaomi Redmi 4X (EDL)

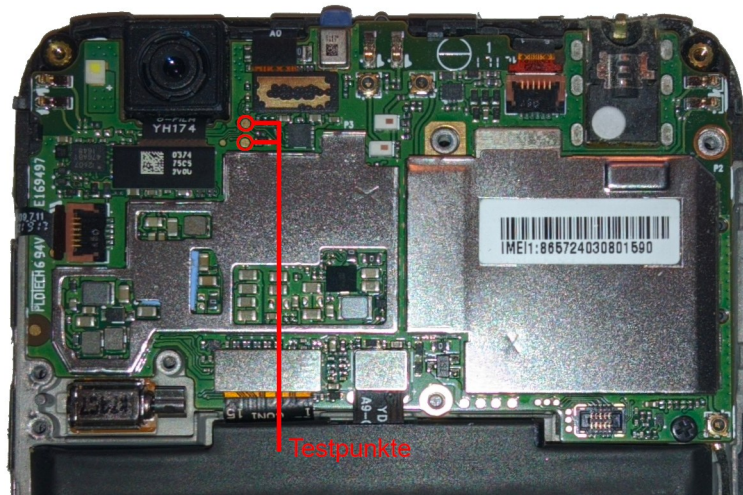


Bild 166: EDL-Testpunkte des Redmi 4X

### Xiaomi Mi A2 Lite (EDL)



Bild 167: EDL-Testpunkte des Mi A2 Lite

## Anhang 5: digitale Anlagen

Der Arbeit ist ein Datenträger beigelegt. Der Inhalt des Datenträgers wird nachfolgend dargestellt.

Ordner	Inhalt
Mail-Verkehr	Im Rahmen der Bearbeitung geführte Mail-Korrespondenz
Output	Binaries, die für Bootloader-Sicherungen erzeugt wurden
Sicherungen (Logs)	Logfiles und Screenshots zu den erstellten Sicherungen
Test-Points	Bilder zu den Test-Points der verwendeten Geräten
Quellen (Web)	PDF-Kopien der zitierten Web-Quellen

### Dateien:

```
.
├── BT_Christian_Peter_2023.odt
├── BT_Christian_Peter_2023.pdf
├── Inhalt.txt
├── Mail-Verkehr
│   ├── Canonical
│   │   ├── 83857579-7e1ce580-a71b-11ea-8483-76edf01b8051.txt
│   │   └── L-09298.pdf
│   └── UBPorts
│       ├── [__EXTERNAL__] UT-Geräte stats - Florian Leeber _florian@ubports.com_ - 2023-02-25 2251.eml
│       ├── [EXTERNAL] UT-Geräte stats.pdf
│       └── Geräteverteilung Ubuntu Touch.ods
├── Output
│   ├── ASUS TF300T Blob
│   │   └── tf300.blob
│   └── Surface 2 BCT und U-Boot
│       ├── surface-2.bct
│       └── u-boot-dtb-tegra.bin
├── Quellen (Web)
│   ├── [100] usb_moded-doc.txt
│   ├── [101] Status_USB Peripheral - Maemo Leste Wiki.pdf
│   ├── [102] Was ist Secure Shell (SSH _ SSH-1 _ SSH-2 _ OpenSSH) .pdf
│   ├── [104] ewfacquirestream(1) - Linux man page.pdf
│   ├── [105] Shell access via SSH - UBports Dokumentation.pdf
│   ├── [106] apt - How can I access my Ubuntu phone over ssh - Ask Ubuntu.pdf
│   ├── [108] Device Mapper Targets – Thomas-Krenn-Wiki.pdf
│   ├── [109] howto networking [Mobian Wiki].pdf
│   ├── [110] Jolla devices.pdf
│   ├── [110] Shell access via ADB - UBports Dokumentation.pdf
│   └── [112] Compelson.pdf
```

- | — [113] TWRP - About.pdf
- | — [114] A\_B (Seamless) System Updates Android Open Source Project.pdf
- | — [115] Google mandates virtual A\_B system updates for devices launching with Android 13.pdf
- | — [116] Galaxy S23 Samsung ignoriert wichtiges Android-Feature schon wieder NETZWELT.pdf
- | — [117] [ROM][UNOFFICIAL] LineageOS 18.1 for Xperia 5 [EOL] XDA Forums.pdf
- | — [118] What Is UFS Storage How Is It Better Than eMMC - Fossbytes.pdf
- | — [119] Android\_ So schaltet ihr den Recovery- und Download-Modus ein \_ NETZWELT.pdf
- | — [11] Jolla devices.pdf
- | — [120] What Is Samsung's Odin Firmware Flashing Software .pdf
- | — [121] Motorola Droid 4 (motorola-maserati) - postmarketOS.pdf
- | — [122] Exploiting Qualcomm EDL Programmers (1)\_ Gaining Access & PBL Internals.pdf
- | — [123] A technical look at Phone Extraction FINAL.pdf
- | — [124] Redmi Note 8T EDL Mode Test Point.pdf
- | — [126] Support of MediaTek devices in Oxygen Forensic® Detective - Forensic Focus.pdf
- | — [127] [PRELOADER][IMG][STOCK] PRELOADER Partition Binary for DANDELION (Redmi 9A \_ Redmi 9AT) \_ XDA Forums.pdf
- | — [128] APX mode - XDA-Developers.pdf
- | — [129] Fusée Gelée - Open Surface RT.pdf
- | — [12] PinePhone community poll results PINE64.pdf
- | — [130] Fusée Gelée Nintendo Switch gehackt - PC-WELT.pdf
- | — [131] [GUIDE] How to enter APX mode or Bootloader menu XDA Forums.pdf
- | — [132] Tegra3 Guide nvflash NEXUS 7 (and Transformer Jellybeans) Readynests.com.pdf
- | — [133] [TUTORIAL] Generate blobs for a bricked Nexus 7 without another N7 or Tegra30 device \_ XDA Forums.pdf
- | — [134] GitHub - GeorgeMato4\_nvcryptools at forN7.pdf
- | — [135] ASUS Transformer Pad (asus-tf300t) - postmarketOS.pdf
- | — [136] Open Surface RT Home - Open Surface RT.pdf
- | — [137] Discord-Surface.png
- | — [138] GitHub - Open-Surface-RT\_fusee-tools Useful tools for Fusee Gelee on T20 \_ T30 \_ T114.pdf
- | — [139] GitHub - Open-Surface-RT\_u-boot at microsoft-surface-2.pdf
- | — [13] Purism Librem 5 Crowdfunding Ends With Over \$2 million Raised - OMG! Ubuntu.pdf
- | — [141] Linux mainlining effort - linux-sunxi.org.pdf
- | — [142] Allwinner SoC based boards — Das U-Boot unknown version documentation.pdf
- | — [143] linux-sunxi\_sunxi-tools A collection of command line tools for ARM devices with Allwinner SoCs..pdf
- | — [144] FEL - linux-sunxi.org.pdf
- | — [145] PinePhone - linux-sunxi.org.pdf
- | — [146] what are these pins above modem chip.pdf
- | — [147] GitHub - ayufan-pine64\_boot-tools\_Pine A64 mass storage using FEL and u-boot.pdf
- | — [149] Tsurugi Acquire.pdf
- | — [14] Russia's Quest for Digital Sovereignty DGAP.pdf
- | — [151] dreemurrs-embedded\_Jumpdrive Flash\_Rescue SD Card image for PinePhone and PineTab.pdf
- | — [152] PinePhone Installation Instructions - PINE64.pdf
- | — [153] Access a locked Nokia N9 MeeGo phone.pdf
- | — [154] [Release notes] Struven ketju 4.5.0.16 - Announcements - Sailfish OS Forum.pdf
- | — [155] GitHub - droidian\_droidian-encryption-service.pdf
- | — [156] cryptsetup \_ cryptsetup · GitLab.pdf
- | — [157] dm-crypt\_Luks – Begriffe, Funktionsweise und die Rolle des Hash-Verfahrens – I Linux-Blog – Dr. Mönchmeyer \_ anracon – Augsburg.pdf
- | — [158] Festplattenverschlüsselung mit LUKS » ADMIN-Magazin.pdf
- | — [159] Bruteforcing Linux Full Disk Encryption (LUKS) With Hashcat - Forensic Focus.pdf
- | — [15] projekt\_atom-tablet-linux\_start [LeineLab].pdf
- | — [160] Cracking Linux Disk Encryption LUKS - Penguin-Systems.pdf
- | — [161] libvslvm\_Logical Volume Manager (LVM) format.asciidoc at main · libyal\_libvslvm · GitHub.pdf
- | — [162] glv2\_bruteforce-luks Try to find the password of a LUKS encrypted volume..pdf
- | — [163] Unlocking the Bootloader MOTOROLA Android Phones Motorola Mobility LLC.pdf
- | — [164] Sailfish OS XDA Forums.pdf

- | — [165] Install Ubuntu Touch - UBports documentation.pdf
- | — [166] Index of \_cdimage\_unofficial\_non-free\_cd-including-firmware.pdf
- | — [167] Squashing a Book Full of Worms \_\_ Mobian's Blog.pdf
- | — [168] Lenovo A6000 (lenovo-a6000) - postmarketOS.pdf
- | — [169] msm8916-mainline\_lk2nd Custom bootloader for Qualcomm MSM8916\_MSM8226\_MSM8974\_... devices.pdf
- | — [16] Maemo - Linux-Plattform für mobile Endgeräte von Nokia - Golem.de.pdf
- | — [170] Nokia Support für Symbian und MeeGo endet 2013 - TecChannel Workshop.pdf
- | — [171] N9 RepoMirror OpenRepos.net — Community Repository System.pdf
- | — [172] Installing Sailfish X on XA2 using Linux - Jolla.pdf
- | — [173] Unlock Bootloader - Open Devices - Sony Developer World.pdf
- | — [174] Software binaries for AOSP Oreo (Android 8.1) – Kernel 4.4 – Nile (v16) - Sony Developer World.pdf
- | — [175] Directory Index - Maemo Leste.pdf
- | — [176] tmlind\_droid4-kexecboot.pdf
- | — [177] Android 7.1 auf dem Asus Transformer Pad TF300T – IT-DAD.pdf
- | — [178] GitHub - clamor-s\_linux Linux kernel with experimental patches for Tegra.pdf
- | — [179] Das Volla Phone beweist Ein Smartphone ganz ohne Google oder Apple geht.pdf
- | — [17] Nokia N9 im Test Das MeeGo-Experiment NETZWELT.pdf
- | — [180] Volla Phone Repair Rüben Carneiro.pdf
- | — [181] Flash halium vendor on redmi note 8t (willow) UBports Forum.pdf
- | — [182] [ROM] [Mi A2 Lite] Droidian GSI XDA Forums.pdf
- | — [183] [Alpha][oneplus3\_t][Unofficial] Droidian (Debian Bookworm) for OnePlus 3\_T XDA Forums.pdf
- | — [184] UBports \_ Porting \_ Community Ports \_ android10 \_ Samsung Galaxy S9 - Note 9 \_ samsung-exynos9810 · GitLab.pdf
- | — [185] Touch\_Devices\_M7 - Ubuntu Wiki.pdf
- | — [186] Samsung Galaxy S III (samsung-m0) - postmarketOS.pdf
- | — [187] Xiaomi Redmi 2 (xiaomi-wt88047) - postmarketOS.pdf
- | — [188] install-android [Mobian Wiki].pdf
- | — [189] sailfish-santoni\_projectmanagement Project Management for SailfishOS Redmi 4X..pdf
- | — [18] Windows Phone 7\_ Nokia und Microsoft verbünden sich \_ ZEIT ONLINE.pdf
- | — [190] Release Sailfish x86 amd64 v0.3 (Sailfish 4.0.1.48) · sailfish-x86\_rootfs · GitHub.pdf
- | — [191] [ROM][LINUX][ANGELICA][ANGELICAN] Droidian Bookworm XDA Forums.pdf
- | — [192] Raspberry Pi Download Options Ubuntu MATE.pdf
- | — [19] Ex-Nokia-Leute planen N9-Nachfolger\_ Meego lebt als Jolla weiter - n-tv.de.pdf
- | — [1] Wie deutsche Ermittler beschlagnahmte Smartphones knacken.pdf
- | — [20] Aus den Ruinen von Maemo und Meego entstanden Jolla und Sailfish OS.pdf
- | — [21] Cases - Sailfish OS.pdf
- | — [22] Aurora OS is based on Sailfish but owned by OMP Nokiamob.pdf
- | — [23] Ubuntu Phone OS Unveiled by Canonical - OMG! Ubuntu.pdf
- | — [24] Erste Ubuntu-Smartphones von BQ und Meizu.pdf
- | — [25] Unity 8, Konvergenz und Ubuntu Phone eingestampft - Ubuntu 18.04 geht zurück zu GNOME.pdf
- | — [26] Ubuntu UBports will Ubuntu Touch am Leben erhalten - ComputerBase.pdf
- | — [27] Librem 5 und PinePhone - LinuxCommunity.pdf
- | — [28] Phosh Overview – Purism.pdf
- | — [29] PinePhone Software Releases - PINE64.pdf
- | — [2] Glossar - OpenOS.pdf
- | — [30] postmarketOS \_\_ Aiming for a 10 year life-cycle for smartphones.pdf
- | — [31] Devices - postmarketOS.pdf
- | — [33] hybris-mobian Is Now Droidian \_ Droidian.pdf
- | — [34] Halium Project.pdf
- | — [34] Status — Project Sandcastle.pdf
- | — [35] Fedora 38 Cleared To Produce \_Mobility Phosh\_ Spins - Phoronix.pdf
- | — [36] Planning for snapd support on Ubuntu Touch - snapd - snapcraft.io.pdf
- | — [37] Ubuntu Touch safety architecture \_ UBports.pdf
- | — [38] GitHub - filipkarc\_PoC-ubuntutouch-pin-privesc\_CVE-2022-40297 - Proof of Concept\_ Privilege escalation in Ubuntu Touch 16.04 - by PIN[...]pdf



```

| | | |─ log.txt
| | | |─ nokia_n9_ssh
| | |─ 03_Pine64_Pinephone
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ pinephone_ssh
| | |─ 04_Google_Nexus_5
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ nexus_5_ssh
| | |─ 05_BQ_Aquaris_E5
| | | |─ aquaris_e5_ssh
| | | |─ Aufruf.png
| | | |─ log.txt
| | |─ 06_Sony_Xperia_XA2
| | | |─ Aufruf2.png
| | | |─ Aufruf3.png
| | | |─ Aufruf.png
| | | |─ log2.txt
| | | |─ log3.txt
| | | |─ log.txt
| | | |─ xperia_xa2_ssh
| | | |─ xperia_xa2_unenc_home_ssh
| | | |─ xperia_xa2_unenc_root_ssh
| | |─ 07_Moto_G
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ moto_g_ssh
| | |─ 08_Lenovo_A6000
| | | |─ Aufruf.png
| | | |─ lenovo_a6000_ssh
| | | |─ log.txt
| | |─ 09_Motorola_Droid_4
| | | |─ Aufruf.png
| | | |─ droid_4_mmcbk0_ssh
| | | |─ droid_4_mmcbk1_ssh
| | | |─ log2.txt
| | | |─ log.txt
| | |─ 10_ASUS_Transformer_Pad_TF300T
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ transformer_tf300t_ssh
| | |─ 11_ASUS_Transformer_Book_T100TA
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ transformer_book_ssh
| | |─ 12_Volla_Phone
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ volla_phone_ssh
| | |─ 13_Redmi_Note_8T
| | | |─ Aufruf.png
| | | |─ log.txt

```

```

| | | └─ redmi_note_8t_ssh
| | └─ 14_Samsung_Galaxy_S5
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ samsung_galaxy_s5_ssh
| | └─ 15_Xiaomi_Mi_A2_Lite
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ mi_a2_lite_ssh
| | └─ 16_Samsung_Galaxy_S9
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ samsung_galaxy_s9_ssh
| | └─ 17_HTC_One_M7
| | | └─ Aufruf.png
| | | └─ htc_one_ssh
| | | └─ log.txt
| | └─ 18_Samsung_Galaxy_SIII
| | | └─ Aufruf.png
| | | └─ log2.txt
| | | └─ log.txt
| | | └─ samsung_m0_android_ssh
| | | └─ samsung_m0_ssh
| | └─ 19_Xiaomi_Redmi_2
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ redmi_2_ssh
| | └─ 20_Pocophone_F1
| | | └─ keine_Sicherung
| | └─ 21_Xiaomi_Redmi_4X
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ redmi_4x_ssh
| | └─ 22_Dell_Venue_8_Pro
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ venue_8_ssh
| | └─ 23_Xiaomi_Redmi_9C
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ redmi_9c_ssh
| | └─ 24_Sony_Xperia_5
| | | └─ Aufruf2.png
| | | └─ Aufruf.png
| | | └─ log2.txt
| | | └─ log.txt
| | | └─ xperia_5_ssh
| | | └─ xperia_5_unenc_ssh
| | └─ 25_Surface_2
| | | └─ Aufruf.png
| | | └─ log.txt
| | | └─ surface_2_nc
| └─ 26_Oneplus_3T

```



```

| |   └─ keine_Sicherung
| └─ 02_ADB_Live
| |   └─ 01_BQ_Aquaris_E5
| | |   └─ aquaris_e5_adb
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | └─ 02_Google_Nexus_5
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ nexus_5_adb
| | └─ 03_HTC_One_M7
| | |   └─ Aufruf.png
| | |   └─ htc_one_adb
| | |   └─ log.txt
| └─ 04_Xiaomi_Redmi_Note_8T
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ redmi_note_8t_adb
| └─ 05_Samsung_Galaxy_S9
| | |   └─ keine_Sicherung
| | └─ 06_Volla_Phone
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ volla_phone_adb
| └─ 03_Recovery
| | └─ 01_Xiaomi_Redmi_2
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ redmi_2_rec
| | └─ 02_Google_Nexus_5
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ nexus_5_rec
| | └─ 03_HTC_One_M7
| | |   └─ Aufruf.png
| | |   └─ htc_one_rec
| | |   └─ log.txt
| └─ 04_Xiaomi_Redmi_4X
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ redmi_4x_rec
| | └─ 05_BQ_Aquaris_E5
| | |   └─ aquaris_e5_rec
| | |   └─ aquaris_e5_rec.E01
| | |   └─ Aufruf.png
| | └─ 06_Xiaomi_Mi_A2_Lite
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ mi_a2_lite_rec
| └─ 07_Sony_Xperia_XA2
| | |   └─ Aufruf.png
| | |   └─ log.txt
| | |   └─ xperia_xa2_rec

```

- | | |— 08\_Lenovo\_A6000
  - | | | |— a6000\_rec
  - | | | |— Aufruf.png
  - | | | |— keine\_Sicherung
  - | | | |— log.txt
- | | |— 09\_LG\_Moto\_G
  - | | | |— Aufruf.png
  - | | | |— log.txt
  - | | | |— moto\_g\_rec
- | | |— 10\_Oneplus\_3T
  - | | | |— Aufruf.png
  - | | | |— log2.txt
  - | | | |— log3.txt
  - | | | |— log4.txt
  - | | | |— log5.txt
  - | | | |— log.txt
  - | | | |— oneplus\_3t\_sda\_rec
  - | | | |— oneplus\_3t\_sdb\_rec
  - | | | |— oneplus\_3t\_sdc\_rec
  - | | | |— oneplus\_3t\_sde\_rec
  - | | | |— oneplus\_3t\_sdf\_rec
- | | |— 11\_Volla\_Phone
  - | | | |— Aufruf.png
  - | | | |— log.txt
  - | | | |— volla\_phone\_rec
- | | |— 12\_Xiaomi\_Redmi\_9C
  - | | | |— Aufruf.png
  - | | | |— log.txt
  - | | | |— redmi\_9c\_rec
- | | |— 13\_Samsung\_Galaxy\_S9
  - | | | |— Aufruf.png
  - | | | |— galaxy\_s9\_rec
  - | | | |— galaxy\_s9\_sdb\_rec
  - | | | |— galaxy\_s9\_sdc\_rec
  - | | | |— galaxy\_s9\_sdd\_rec
  - | | | |— galaxy\_s9\_sdf\_rec
  - | | | |— log2.txt
  - | | | |— log3.txt
  - | | | |— log.txt
- | | |— 14\_Xiaomi\_Redmi\_8T
  - | | | |— Aufruf.png
  - | | | |— log.txt
  - | | | |— redmi\_8t\_rec
- | | |— 15\_Sony\_Xperia\_5
  - | | | |— Aufruf.png
  - | | | |— log.txt
  - | | | |— xperia\_5\_rec
- | | |— 16\_Pocophone\_F1
  - | | | |— Aufruf.png
  - | | | |— log.txt
  - | | | |— poco\_f1\_sda\_rec
  - | | | |— poco\_f1\_sdb\_rec
  - | | | |— poco\_f1\_sdc\_rec

```

| | | |─ poco_f1_sde_rec
| | | |─ poco_f1_sdf_rec
| | |─ 17_Samsung_Galaxy_SIII
| | | |─ Aufruf2.png
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ samsung_s3_extern_rec
| | | |─ samsung_s3_intern_rec
| | |─ 18_Samsung_Galaxy_S5
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ samsung_s5_rec
| | |─ 19_ASUS_Transformer_Book_TF300T
| | | |─ Aufruf2.png
| | | |─ Aufruf.png
| | | |─ log.txt
| | | |─ tf300t_extern_rec
| | | |─ tf300t_rec
| |─ 04_Bootloader
| | |─ 01_EDL
| | | |─ 01_Xiaomi_Redmi_2
| | | | |─ Aufruf.png
| | | | |─ Bootloader.png
| | | | |─ log.txt
| | | | |─ redmi_2_bl
| | | |─ 02_Google_Nexus_5
| | | | |─ keine_Sicherung
| | | | |─ log.txt
| | | |─ 03_HTC_One_M7
| | | | |─ keine_Sicherung
| | | | |─ log.txt
| | | |─ 04_Xiaomi_Redmi_4X
| | | | |─ Aufruf.png
| | | | |─ Bootlaoder.png
| | | | |─ log.txt
| | | | |─ redmi_4X_bl
| | | |─ 05_Lenovo_A6000
| | | | |─ a6000_bl
| | | | |─ Aufruf.png
| | | | |─ log.txt
| | | |─ 06_Sony_Xperia_XA2
| | | | |─ keine_Sicherung
| | | |─ 07_LG_Moto_G
| | | | |─ keine_Sicherung
| | | |─ 08_Oneplus_3T
| | | | |─ Aufruf.png
| | | | |─ Bootloader.png
| | | | |─ log.txt
| | | | |─ oneplus_3t_bl
| | | |─ 09_Pocophone_F1
| | | | |─ Aufruf.png
| | | | |─ Bootloader.png
| | | | |─ log.txt

```

```

| | | | └─ 10_Xiaomi_Mi_A2_Lite
| | | |   └─ Bootloader.png
| | | |     └─ log.txt
| | | └─ 11_Xiaomi_Redmi_Note_8T
| | |   └─ Aufruf.png
| | |   └─ Bootloader.png
| | |   └─ log.txt
| | |     └─ redmi_note_8t_bl
| | └─ 02_MTK
| |   └─ 01_BQ_Aquaris_E5
| |     └─ aquaris_e5_bl
| |       └─ Aufruf.png
| |       └─ Bootloader.png
| |       └─ log.txt
| | └─ 02_Volla_Phone
| |   └─ Aufruf.png
| |   └─ Bootloader.png
| |   └─ log.txt
| |     └─ volla_phone_bl
| | └─ 03_Volla_Phone
| |   └─ Aufruf.png
| |   └─ Bootloader.png
| |   └─ log.txt
| |     └─ redmi_9c_bl
| | └─ 03_APX
| |   └─ 01_ASUS_Transformer_Pad_TF300T
| |     └─ Aufruf.png
| |     └─ log.txt
| |       └─ tf300t_bl
| | └─ 02_Microsoft_Surface_2
| |   └─ Aufruf.png
| |   └─ log.txt
| |     └─ surface_2_bl
| └─ 04_FEL
|   └─ 01_Pine64_Pinephone
|     └─ Aufruf.png
|     └─ Bootloader.png
|     └─ log.txt
|       └─ pinephone_fel
| └─ 05_Live_Linux
|   └─ 01_ASUS_Transformer_Book_T100TA
|     └─ Aufruf.png
|     └─ t100ta_tsurugi.info
| └─ 02_Dell_Venue_8_Pro
|   └─ Aufruf.png
|   └─ log.txt
|     └─ venue_8_tsurugi
| └─ 06_Jumpdrive
|   └─ 01_Pine64_Pinephone
|     └─ Aufruf.png
|     └─ log.txt
|       └─ pinephone_jumpdrive
| └─ 02_Pocophone_F1

```

- | | |— Aufruf.png
- | | |— log.txt
- | | |— poco\_f1\_jumpdrive
- | |— 07\_Extern
- | |— 01\_Motorola\_Droid\_4
- | | |— Aufruf.png
- | | |— droid\_4\_extern
- | | |— log.txt
- | |— 02\_Samsung\_Galaxy\_SIII
- | | |— Aufruf.png
- | | |— log.txt
- | | |— samsung\_s3\_extern
- | |— 03\_ASUS\_Transformer\_Book\_TF300T
- | | |— asus\_tf300t\_extern
- | | |— Aufruf.png
- | | |— log.txt
- |— Test-Points
- | |— EDL\_Mi\_A2\_Lite.jpg
- | |— EDL\_Redmi\_4x.jpg
- | |— EDL\_Redmi\_Note\_8t.jpeg
- | |— FEL\_Pinephone.jpeg