

HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

WebUSB-basierte Sicherung forensischer Spuren auf Android-Mobiltelefonen

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.) in Informatik

eingereicht von: Kara Engelhardt

Gutachter/innen: Prof. Dr. Jens-Peter Redlich
Prof. Dr. Matthias Weidlich

eingereicht am: verteidigt am:

Inhaltsverzeichnis

1	Zusammenfassung	2
2	Bestehende Ansätze	3
2.1	Externe Traffic-Analyse	4
2.2	Backups	4
2.3	Android Debug Bridge (ADB)	5
3	Anforderungen	12
3.1	Angreifer*innen-Modell	12
3.2	Nutzer*innen-Modell	15
3.3	Umgang mit den gesammelten Daten	16
4	Praktische Lösung	19
4.1	Relevante Schnittstellen	19
4.2	Eingesetzte Bibliotheken	25
4.3	Implementation	30
4.4	Aufgetretene Probleme	41
5	Auswertung	43
5.1	Anforderungen	43
5.2	Vergleich mit AndroidQF	47
6	Ausblick	50
6.1	Bessere Wartbarkeit und Integrierbarkeit	50
6.2	Weitere Datenquellen	50
6.3	ADB mit root-Rechten ausführen	50
6.4	Prüfsumme des generierten Archivs	51
6.5	Nutzer*innen-Freundlichkeit	51
6.6	Weitere Verschlüsselungsverfahren	51
6.7	Andere Kompressionsverfahren	52
6.8	Reproducible Builds	52
6.9	Reduzierung der Anzahl der eingesetzten Bibliotheken	52
6.10	Offline-Paketierung	52
	Literaturverzeichnis	i

1 Zusammenfassung

Smartphones sind ein wichtiger Teil des Lebens vieler Menschen und damit ein lohnenswertes Angriffsziel für verschiedene Formen von Spyware. Große Aufmerksamkeit erhielten in den letzten Jahren Veröffentlichungen zu sogenannter Mercenary Spyware wie das Pegasus Project[1], Catalan Gate[2] oder die Predator Files[3]. Als Mercenary Spyware bezeichnet man spezialisierte Spyware, die vor allem für den Einsatz durch staatliche Institutionen, insbesondere Strafverfolgungsbehörden und Geheimdienste, entwickelt wird.

Daneben spielt Smartphone-Spyware auch im Rahmen von Beziehungsgewalt[4] oder als sogenannte Stalkerware[5] eine Rolle. Im Jahr 2023 konnte Kaspersky etwas mehr als 31.000 Infektionen mit Stalkerware auf Geräten nachweisen, auf denen Kasperskys Produkte liefen[6]. Leaks und Hacks von Stalkerware-Anbietern konnten zehntausende bis Millionen von Kund*innen nachweisen[7].

Durch spezialisierte Programme wie „AndroidQF“[8] können bestimmte forensisch relevante Spuren auf Android-Geräten gesammelt werden. Die so gesammelten Daten können anschließend von Nutzer*innen selbst mithilfe bestehender Werkzeuge wie dem „Mobile Verification Toolkit“ (MVT)[9] ausgewertet oder anderen Personen oder Gruppen zur Auswertung übersandt werden. Allerdings sind die bestehenden Programme zur Sammlung dieser Spuren als Kommandozeilen-Tools für viele Nutzer*innen nicht intuitiv nutzbar und stellen so eine Hürde dar.

Diese Arbeit zeigt mit einem Prototypen, dass ein nutzer*innen-freundlicheres Sammeln dieser Daten über die WebUSB-Schnittstelle Chromium-basierter Browser möglich ist. Das im Rahmen dieser Arbeit entwickelte Tool läuft direkt im Browser der Nutzer*innen. Dabei wird von den Nutzer*innen dieses Tools kein technisches Verständnis über das Sammeln und Auswerten der Daten verlangt.

Zunächst wird in „Bestehende Ansätze“ ein Überblick über verschiedene Wege, Schadsoftwareinfektionen nachzuweisen, die dafür genutzten Tools und benötigten Daten gegeben. Anschließend werden in „Anforderungen“ zwei Angreifer*innen sowie ein Nutzer*innen-Modell formuliert und daraus Anforderungen an die Software zum Sammeln der Daten abgeleitet. Im Abschnitt „Praktische Lösung“ werden dann relevante Technologien für die Umsetzung als WebUSB-basiertes Browser-Tool erklärt sowie die praktische Umsetzung und die dabei aufgetretenen Probleme beschrieben. In der „Auswertung“ wird die Umsetzung mit den vorab formulierten Anforderungen abgeglichen und die Software mit AndroidQF verglichen. Schlussendlich liefert der „Ausblick“ mögliche Ansätze, um die Lösung weiter zu verbessern.

2 Bestehende Ansätze

***Hinweis:** Bei „Mercenary Spyware“ und „Stalkerware“ handelt es sich um zwei Arten von Spyware, die in dieser Arbeit betrachtet werden sollen. Diese werden im Kapitel „Anforderungen“ detaillierter eingeführt.*

Schadsoftware-Infektionen können auf vielen Wegen nachgewiesen werden:

Deutsche Ermittlungsbehörden müssen nach einem Einsatz von Quellen-TKÜ und Online-Durchsuchung gemäß § 101 StPO[10] die überwachte Person sowie relevant Mit-Betroffene informieren.

2021 wurde das „Pegasus Project“[1] veröffentlicht, welches auf einem Leak von 50.000 Telefonnummern basierte, die in Systemen des Mercenary-Spyware-Herstellers „NSO Group“ zur Überwachung ausgewählt wurden[11]. TechCrunch zählte im Juni 2025 26 Anbieter von Stalkerware, die gehackt wurden oder deren Daten in signifikanter Menge veröffentlicht wurden[7]. Einige der Unternehmen wurden sogar mehrfach getroffen. Die so erlangten Daten erlauben es, Betroffene von Spyware zu identifizieren und bieten die Möglichkeit, diese zu informieren.

Auch Apple[12] und Meta[13] haben Nutzer*innen informiert, wenn ein Einsatz von Mercenary-Spyware gegen diese erkannt wurde.

Solche externen Bestätigungen bieten hohe Sicherheit, dass eine Infektion vorlag und können dies für viele Betroffene gleichzeitig bestätigen. Allerdings ist es größtenteils ausserhalb der Kontrolle der Infizierten, ob und wann sie diese Bestätigung erhalten. Zudem ist ein Ausbleiben einer solchen Bestätigung kein Indiz für eine Nicht-Infektion.

Haben Nutzer*innen den Verdacht einer Schadsoftware-Infektion, ist eine Analyse des Geräts nötig. Hierfür werden Spuren der Infektion auf dem Gerät gesucht.

Zunächst ist anzumerken, dass sich die sammelbaren Spuren je nach Betriebssystem stark unterscheiden. Android-Geräte halten dabei deutlich weniger nützliche Daten als iOS-Geräte vor, was den Nachweis einer Infektion erschwert. In der Dokumentation des Auswertungstools MVT heißt es:

„Unfortunately Android devices provide much less observability than their iOS cousins. Android stores very little diagnostic information useful to triage potential compromises, and because of this `mvt-android` capabilities are limited as well.“[14]

Auch Amnesty International schreibt dazu in einem forensischen Bericht über das Finden von Pegasus Spyware:

„In Amnesty International’s experience there are significantly more forensic traces accessible to investigators on Apple iOS devices than on stock Android devices, therefore our methodology is focused on the former. As a result, most recent cases of confirmed Pegasus infections have involved iPhones.“[15]

Um Daten für diese Analyse zu sammeln, gibt es verschiedene Ansätze, von denen im Folgenden drei vorgestellt werden, die für die Analyse von Android-Smartphones eine besondere Rolle spielen.

2.1 Externe Traffic-Analyse

Bei der Traffic-Analyse werden die von einem Gerät übertragenen Daten mitgeschnitten und anschließend analysiert. Dies kann beispielsweise durch einen eigenen WiFi-Hotspot geschehen, wie beim *SpyGuard*-Projekt[16]. Dieser Ansatz erlaubt es, unabhängig von der Art des Endgeräts Daten zu sammeln, solange dieses mit einem WiFi-Netzwerk verbunden werden kann.

Sollen neben den Verbindungs-Metadaten und unverschlüsselt übertragenen Inhalten¹ auch TLS-verschlüsselte Inhalte analysiert werden, wäre zudem ein TLS-Proxy und damit verbunden die Installation einer eigenen Root-CA auf dem Gerät nötig. Dies könnte eine Schadsoftware durch *Certificate Pinning* erkennen.

Um auch Schadsoftware zu erkennen, die lediglich über Mobilfunk Daten überträgt, könnten eigene Mobilfunk-Basisstationen eingesetzt werden[18], was allerdings mit einem deutlich höheren Aufwand und gegebenenfalls rechtlichen Schwierigkeiten, z. B. Lizenzen für die Nutzung von Mobilfunkbändern, verbunden ist.

2.2 Backups

Eine zweite Möglichkeit, Daten von einem Gerät zu sammeln, ist die eingebaute Backup-Funktion. Google selbst bietet ein Online-Backup in die Google-Cloud an[19], auch über ADB kann ein Backup durchgeführt werden[20].

Während diese Backups auf iOS meist alle nötigen Spuren enthalten[21], ist dies auf Android nicht der Fall. Diese Backups stellen keine gute Datenquelle dar, da Apps sich durch das Setzen des `android:allowBackup`-Attributes auf `false` selbst aus Backups ausschließen können[20]. Ab Android 12 werden bei adb-backups keine App-Daten mehr gebackupt, solange dies nicht gesondert aktiviert wurde[22].

¹Stalkerware nutzt teilweise weiterhin unverschlüsselte HTTP-Verbindungen[17].

Zudem werden nur Apps und ihre Daten in das Backup einbezogen, nicht jedoch System-Logs.

2.3 Android Debug Bridge (ADB)

Ein weiterer Ansatz ist das Sammeln von Daten über die Android Debug Bridge (ADB)[23], eine eigentlich für die Android-Entwicklung gedachte Schnittstelle. Diese muss zunächst über die Systemeinstellungen des Geräts aktiviert werden. Über ADB können Programme auf dem Gerät ausgeführt, Information wie System-Logs vom Gerät gesammelt und Dateien vom sowie auf das Gerät kopiert werden.

Im Rahmen des Pegasus Projects hat das Amnesty International Security Lab das Mobile Verification Toolkit (MVT) veröffentlicht[24], eine Python-Software, die sowohl Daten für eine forensische Analyse sammeln, als auch diese mit einer Liste von *Indicators of Compromise* vergleichen kann.

Indicators of Compromise

Als *Indicators of Compromise* (IoC) bezeichnet man Spuren, die ein starkes Indiz für eine Schadsoftwareinfektion bieten[25]. Dabei kann es sich beispielsweise um verdächtige Prozessnamen, Signaturen von Software, bei der Infektion genutzte Domains, IP-Adressen von Command und Control-Servern (C2-Server) oder E-Mail-Adressen handeln[26]. Wird eine Infektion auf einem Gerät mittels manueller Analyse nachgewiesen, kann eine Liste dabei gefundener verräterischer Spuren erstellt werden. Werden diese auf einem anderen System gefunden, ist dies ein starkes Indiz dafür, dass auch dieses System mit der jeweiligen Schadsoftware befallen ist oder war. Diese IoC erlauben es, automatisiert nach bereits bekannten Spuren zu suchen und können so den Nachweis einer Infektion deutlich beschleunigen. Das Teilen von IoC erlaubt es auch anderen Organisationen oder Individuen, nach bereits bekannten Spuren zu suchen, selbst wenn kein Detailwissen über die Schadsoftware existiert.

Das Amnesty International Security Lab veröffentlicht eine Reihe IoC ihrer Ermittlungen, beispielsweise zum Befall mit der Pegasus Spyware[26]. Für Stalkerware veröffentlicht die französische NGO Echap eine Liste von IoC[27]. Zum Austausch von IoC existieren standardisierte Dateiformate, insbesondere STIX2[28], jedoch kommen in manchen Tools auch eigene Formate zum Einsatz[27].

Das MVT wird seitdem von einer Reihe von Organisationen kontinuierlich weiterentwickelt und erweitert[29]. Es bietet damit einen guten Überblick, welche Spuren

sich in der Praxis bewährt haben und wie diese ausgewertet werden können. Für Android setzt das MVT primär auf die Datensammlung via ADB[30].

Daten via ADB können entweder während der Auswertung gesammelt werden oder vorab gesammelt und später ausgewertet werden.

2.3.1 Live-Auswertung am Beispiel von MVTs `check-adb`

Bei der Live-Auswertung werden die forensisch relevanten Spuren vom Gerät gesammelt und direkt ausgewertet. Das MVT bietet diese Funktionalität über das `check-adb`-Subkommando an. Diese hat eine Reihe von Submodulen[31], welche jeweils bestimmte Daten einsammeln und mit einer Liste von IoC vergleichen.

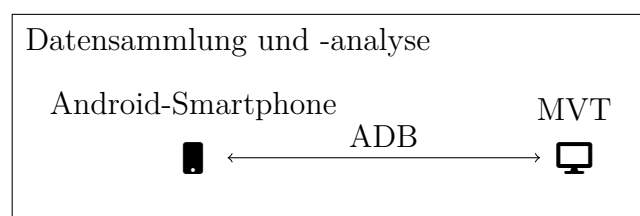


Abbildung 1: Systemdiagramm für MVTs `check-adb`²

2.3.2 Trennung von Datensammlung und Analyse am Beispiel von AndroidQF und MVTs `check-androidqf`

Da die für eine Auswertung üblicherweise relevanten Daten jedoch bekannt sind, können die Schritte der Datensammlung und -analyse auch voneinander getrennt werden. Dies ist beispielsweise mittels AndroidQF und dem MVT-Subkommando `check-androidqf` möglich.

Bei AndroidQF handelt es sich um ein in Go geschriebenes Kommandozeilen-Tool, das sich via ADB mit einem Android-Handy verbindet und dort eine Reihe forensisch relevanter Daten sammelt[8]. Diese können optional in ein Zip-Archiv gepackt und verschlüsselt werden. Das MVT bietet ein Subkommando (`check-androidqf`), welches die so gesammelten Daten anschließend analysieren und mit einer Liste von IoC vergleichen kann.

²Icons aus FontAwesome 5, Lizenz: SIL OFL 1.1

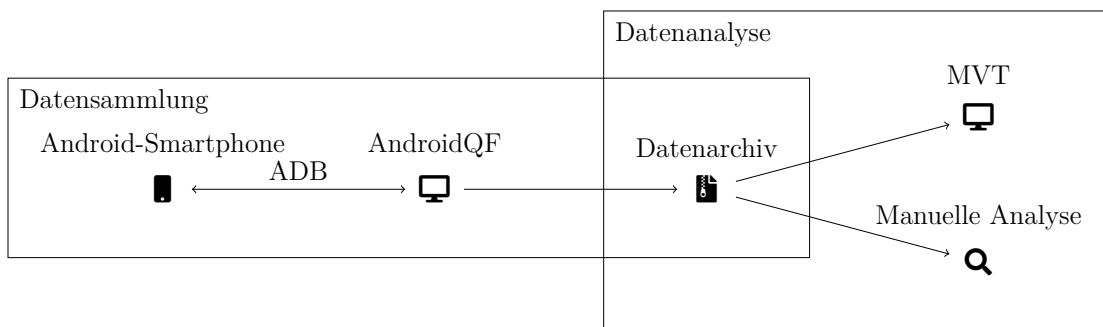


Abbildung 2: Systemdiagramm für AndroidQF und MVTs `check-androidqf`²

Diese Methode bietet einige Vorteile:

- Die Analyse kann später und von anderen Personen durchgeführt werden. So können Betroffene selbst ein Archiv zusammenstellen, welches dann an eine andere Gruppe übergeben wird, die die Analyse übernimmt. Diese Gruppe muss sich nicht am gleichen Ort befinden, was beispielsweise bei Betroffenen in repressiven Staaten von Vorteil sein kann. Zudem müssen Betroffene so nicht selbst das technische Wissen haben, um eine solche Analyse durchzuführen.
- Die Analyse kann explorativ erfolgen: Während die Live-Analyse den Fokus auf den automatischen Vergleich mit bekannten IoC legt, können die so gesammelten Daten auch manuell analysiert werden. Beispielsweise können Systemlogs auf neue, bisher unbekannte Spuren durchsucht werden.
- Werden neue IoC bekannt, können bestehende Archive erneut nach diesen durchsucht werden.

2.3.3 Datenquellen

Der Quellcode von MVT[31] und AndroidQF[32] bietet einen Überblick über Daten, die sich bisher als nützlich erwiesen haben. Im Folgenden werden zunächst die von diesen Tools unterstützten Datenquellen kurz beschrieben. Anschließend wird ein Überblick gegeben, welche ADB-Funktionen für das Sammeln dieser Daten genutzt werden müssen und welche der Tools welche Daten unterstützen.

- **Backup:** Mittels der auch über ADB ansprechbaren Backup-Funktion kann ein Backup entweder aller oder einzelner Anwendungen und ihrer Daten durchgeführt werden. Dies ist, wie bereits beschrieben, nur für Anwendungen möglich, die dies nicht deaktivieren.

- **Bug-Report:** Android erlaubt es, sogenannte „Bug-Reports“ zu erstellen. Hierbei handelt es sich um Zip-Files, die eine Reihe von Diagnose-Informationen enthalten, unter anderem Systemlogs und die Ausgabe von `dumpsys`[33].

Um einen Bug-Report zu erstellen, müssen, je nach Android-Version, die Kommandozeilentools `bugreport` oder `bugreportz` aufgerufen werden. Ersteres schreibt die Bug-Report-Daten auf seine Standardausgabe, Zweiteres erstellt ein Zip-Archiv auf dem Gerät und liefert über die Standardausgabe nur Statusinformationen, insbesondere den Pfad dieser. Dieses Archiv kann anschließend vom Gerät übertragen werden.

- **Chrome-Browserverlauf:** Hierbei wird die Datenbank des Chrome-Browsers vom Gerät kopiert und der darin enthaltene Browserverlauf ausgelesen.
- **Dumpsys:** `dumpsys` ist ein Shell-Kommando, welches Informationen zu den auf dem Gerät laufenden Diensten ausgibt[34]. Darunter finden sich Informationen dazu, welchen Anwendungen welche Berechtigungen gewährt wurden, welche Anwendungen welche Intents empfangen, welche Pakete Accessibility-Berechtigungen haben, zum Batterieverbrauch (auch einzelner Apps und Dienste), welche Anwendungen als `Device Administrator` registriert sind und vieles mehr.
- **Umgebungsvariablen:** Hierbei werden durch Aufrufen des `env`-Shell-Kommandos die in der Geräteshell gesetzten Umgebungsvariablen erfasst.
- **Dateien:** Hierbei wird für eine Reihe von Ordnern eine Liste der darin liegenden Dateien, jeweils zusammen mit einigen Metadaten, erstellt.

Dies geschieht bei MVTs `check-adb` durch einen Aufruf des `find`-Kommandos.

AndroidQF kopiert hierfür ein Binary (`collector`) auf das Gerät und führt dieses aus, um forensisch relevante Informationen zuverlässiger sammeln zu können[35]. Falls dies nicht funktioniert, wird `find` ausgeführt.

- **GetProp:** `getprop` ist ein Shell-Kommando, welches die Android System Properties und ihre jeweiligen Werte ausgibt. Android System Properties ist ein systemweiter Schlüssel-Wert-Speicher, in dem Werte wie die Zeitzone, Build-Versionen des Systems oder die IMEI gespeichert sind[36].
- **Logcat:** `logcat` ist ein Shell-Kommando, welches die über den zentralen Log-Daemon `logd` eines Android-Geräts erfassten Log-Nachrichten ausgibt[37]. Dabei handelt es sich sowohl um Systemlogs als auch Anwendungslogs.

- **Logs:** Zusätzlich zum zentralen Log-System gibt es noch eine Reihe von Log-files, welche ebenfalls vom Gerät kopiert werden können, soweit sie vorhanden sind.
- **Pakete:** Durch Aufrufe des Android Package Manager `pm` können folgende Daten gesammelt werden:
 - eine Liste der installierten Pakete (auch deaktivierte und System-Apps)
 - der Pfad ihrer Programm-Archive (`.apk`-Dateien)
 - durch welches Paket sie installiert wurden

Zudem können die Programm-Archive dieser Pakete auf dem Gerät ghasht sowie vom Gerät kopiert werden.

- **Prozesse:** Hierbei wird eine Liste der laufenden Prozesse gesammelt. Dies kann durch einen Aufruf von `ps` geschehen. AndroidQF versucht auch hierfür präferiert den Collector auf das Gerät und auszuführen und nutzt `ps` nur als Fallback.
- **Root Binaries:** Hierbei wird nach dem Vorhandensein eines der üblichen Kommandos, um `root`-Rechte zu erlangen, gesucht.
- **SELinux Status:** Hierbei wird durch Aufruf des `getenforce`-Kommandos der SELinux-Status ausgelesen.
- **Laufende Systemdienste:** Hierbei wird durch Aufruf des `service list`-Kommandos eine Liste der laufenden Systemdienst generiert.
- **Systemeinstellungen:** Hierbei werden die Systemeinstellungen extrahiert. Android speichert drei verschiedene Arten von Systemeinstellungen:
 - nutzer*innen-übergreifende Einstellungen (`global`),
 - nutzer*innen-spezifische Einstellungen, die von Applikationen nur gelesen, aber nicht geschrieben werden können (`secure`),
 - nutzer*innen-spezifische Einstellungen, die von Applikationen auf dem Gerät geschrieben werden können (`system`)[\[38\]](#).

Diese Einstellungen können über die `SettingsProvider`-Klasse gelesen werden, welche mit dem `SettingsService` auch einen Dienst für den Kommandozeilenzugriff bereitstellt. Dieser kann über das `cmd`-Kommando aufgerufen werden: `cmd settings list [namespace]`, wobei `[namespace]` die Werte `global`, `secure` oder `system` haben kann.

- **SMS:** Hierbei werden die SMS-Datenbanken³ üblicher SMS-Apps vom Gerät kopiert und analysiert. Dies ist nur mit `root`-Rechten möglich.
- **Tempfiles:** Hierbei werden die Dateien im Standard-Verzeichnis für temporäre Dateien vom Gerät kopiert und analysiert.
- **WhatsApp:** Hierbei wird die Nachrichten-Datenbank der WhatsApp-Anwendung vom dem Gerät kopiert und ausgewertet. Dies ist nur mit `root`-Rechten möglich.

Hier folgende Tabelle zeigt, welche ADB-Funktionalitäten jeweils genutzt wird. „Backup“ meint das Ansprechen des Backup-Services. „Push“ meint das Übertragen von Dateien auf das Gerät. „Pull“ meint das Übertragen von Dateien vom Gerät. „Root“ meint das Ausführen von Operationen mit `root`-Rechten. „Shell“ meint das Ausführen von Shell-Kommandos auf dem Gerät und das Lesen ihrer Ausgabe.

Tabelle 1: Benötigte Funktionalitäten

Methode	Backup	Push	Pull	Root	Shell
Backup	*				
Bug-Report			*		*
Chrome-Browserverlauf			*	*	
Dumpsys					*
Umgebungsvariablen					*
Dateien		* ⁴			*
GetProp					*
Logcat					*
Logs			*		
Pakete			*		*
Prozesse		* ⁴			*
Root Binaries					*
SELinux Status					*
Laufende Systemdienste					*
Systemeinstellungen					*
SMS			*	*	
Tempfiles			*		*
WhatsApp			*	*	

³Je nach SMS-App sind in diesen Daten auch MMS und RCS-Nachrichten enthalten.

⁴Diese Funktionalität wird nur benötigt, um den Collector auf das Gerät zu kopieren.

Die folgende Tabelle gibt einen Überblick darüber, welche Datenquellen von AndroidQF sowie MVTs `check-androidfq` und `check-adb` jeweils unterstützt werden. Die Spalte „AndroidQF“ zeigt an, ob AndroidQF diese Daten sammeln kann. Die Spalte „`check-androidqf`“ zeigt an, ob MVTs `check-androidqf` mittels AndroidQF Daten dieser Art auswerten kann. Die Spalte „`check-adb`“ zeigt an, ob MVTs `check-adb` Daten dieser Art sammeln und auswerten kann.

Tabelle 2: Unterstützung der Methoden durch AndroidQF und MVT

Methoden	AndroidQF	<code>check-androidqf</code>	<code>check-adb</code>
Backup	★ ⁵	★ ⁶	★ ⁷
Bug-Report	★	★	★
Chrome-Browserverlauf			★ ⁸
Dumpsys	★	★	★
Umgebungsvariablen	★		
Dateien	★	★	★
GetProp	★	★	★
Logcat	★	★	★
Logs	★	★ ⁹	
Pakete	★	★	★
Prozesse	★	★	★
Root Binaries	★		★
SELinux Status	★		★
Laufende Systemdienste	★		
Systemeinstellungen	★	★	★
SMS			★ ⁸
Tempfiles	★		
WhatsApp			★ ⁸

⁵Wahlweise ist ein Backup aller möglichen Anwendungen oder nur der SMS-Datenbank (`com.android.providers.telephony`) möglich.

⁶Unterstützt das Auslesen der SMS-Datenbank aus einem Backup

⁷Nutzt die Backup-Funktion via ADB als Fallback-Lösung, um die SMS-Datenbank zu lesen, falls das Gerät nicht gerootet ist[39, Z. 137].

⁸Nur auf gerooteten Geräten möglich

⁹`check-androidqf` kann lediglich die Zeitstempel der Logdateien auswerten, nicht deren Inhalt.

3 Anforderungen

Im Folgenden sollen Anforderungen an die im Rahmen dieser Arbeit entwickelte Software formuliert werden. Dazu werden zunächst zwei Angreifer*innen-Profile sowie ein Nutzer*innen-Modell beschrieben. Aus diesen werden jeweils Anforderungen an die Software abgeleitet. Anschließend werden Anforderungen an die Software definiert, die den Umgang mit den gesammelten Daten gemäß den Schutzzielen der Informationssicherheit erleichtern können.

3.1 Angreifer*innen-Modell

Für diese Arbeit soll sich auf zwei Arten von Spyware konzentriert werden: Stalkerware und State Sponsored Malware (auch Mercenary Spyware genannt).

3.1.1 Stalkerware

Gemäß Definition der „Coalition against Stalkerware“ ist Stalkerware

„Software, die für Privatpersonen frei oder käuflich verfügbar ist und mittels Remote-Steuerung eine Person in die Lage versetzt, Aktivitäten auf dem Gerät eines anderen Benutzers zu verfolgen, ohne dessen Zustimmung und ohne ausdrückliche, stete Benachrichtigung an diesen Benutzer zu senden. Daher kann Stalkerware die Überwachung von Intimpartnern, Belästigung, Missbrauch, Stalking und/oder Gewalt erleichtern.“[5]

Funktionen solcher Spyware sind beispielsweise der Zugriff auf Nachrichten, die Bildergalerie, den Gerätestandort, Bildschirminhalt sowie Kamera[40, S. 9]. Zudem enthält solche Spyware häufig Techniken, um ihre eigene Existenz gegenüber den überwachten Nutzer*innen zu verschleiern, beispielsweise unauffällige Namen und App-Icons, um sich als Systemapp zu tarnen[17, S. 4].

Bei Stalkerware handelt es sich um Massenprodukte, die kein tiefes technisches Verständnis verlangen[17]. Lediglich kurzzeitiger Zugriff auf das Gerät ist nötig, der auf verschiedenen Wegen erlangt werden kann[41, S. 5]. Nicht immer handelt es sich um Software die (nur) für den Einsatz als Stalkerware entwickelt wurde. Solche Anwendungen können auch andere Anwendungszwecke haben, beispielsweise Diebstahlschutz, das Wiederfinden eines verlorenen Geräts, das Synchronisieren von Daten zwischen Geräten oder die freiwillige Standortfreigabe an Freund*innen[4].

Während diese Apps weitreichenden Zugriff auf Daten auf dem Gerät geben, nutzen sie hierfür jedoch stets APIs des Betriebssystems, wenn auch teils anders als von den Entwickler*innen dieser APIs angedacht (insbesondere Accessibility-APIs)[17]. Es findet keine Kompromittierung von Systemdiensten statt, sodass diesen für eine Analyse des Systems weiterhin vertraut werden kann.

Daraus ergeben sich folgende Anforderungen:

1. Die Software muss Daten sammeln, aus denen diese Art von Spyware erkannt werden kann. Aus der bestehenden Literatur[17], [42] ergeben sich:
 - eine Liste der installierten Anwendungen
 - eine Liste von Anwendungen, die eine oder mehrere der folgenden Berechtigungen haben:
 - `android.permission.RECORD_AUDIO` - Mikrofon-Zugriff
 - `MediaRecorder.AudioSource.VOICE_CALL` - Anrufaufzeichnung
 - `android.permission.BIND_ACCESSIBILITY_SERVICE` - Registrierung als Barrierefreiheitsdienst
 - `android.permission.RECEIVE_SMS` - Erhalten von Nachrichten
 - `android.permission.BIND_DEVICE_ADMIN` - Registrieren als Device Admin
 - eine Liste von Anwendungen, die folgende Intents empfangen:
 - `android.intent.action.NEW_OUTGOING_CALL` - gesendet, wenn ein Anruf getätigt wird[43]; kann zum Starten einer versteckten Bedienoberfläche durch das Wählen einer bestimmten Nummer genutzt werden
 - `android.accessibilityservice.AccessibilityService` - muss empfangen werden können, damit sich eine Anwendung als Barrierefreiheitsdienst registrieren kann[44]
 - `android.provider.Telephony.SMS_RECEIVED` - wird gesendet, wenn eine neue SMS empfangen wird[45]
 - `android.intent.action.VIEW` - wird gesendet wenn eine URL geöffnet wird; kann zum Starten einer versteckten Bedienoberfläche durch das Öffnen einer bestimmten URL genutzt werden
 - `android.app.action.DEVICE_ADMIN_ENABLED` - muss empfangen werden können, damit sich eine Anwendung als Device Admin registrieren kann[46]
 - `android.intent.action.BOOT_COMPLETED` - wird gesendet, nachdem das Gerät gestartet wurde; wird von Spyware-Apps genutzt, um sich selbst nach einem Neustart des Gerätes zu starten
 - eine Liste von Anwendungen, die keine Launcher-Activities definieren

- eine Liste von Anwendungen, die als **Device Administrator** registriert sind
2. Die gesammelten Spuren müssen aus einer von der Spyware nicht manipulierbaren Quelle stammen. So genügt eine Auflistung der im App-Launcher sichtbaren Apps nicht, da Spyware sich dort verstecken kann[17]. Eine Auswertung von Daten der Systemdienste ist allerdings weiterhin möglich.
 3. Die Nutzung darf durch die Spyware nicht erkennbar sein, bzw. es muss eine andere plausible Erklärung für die von der Spyware erkennbaren Spuren geben („Plausible Deniability“).

3.1.2 State-Sponsored Malware / Mercenary Spyware

Bei sogenannter „State-Sponsored Malware“, oft auch „Mercenary Spyware“ (Söldner-Spyware) genannt, handelt es sich um spezialisierte Spyware, die vor allem für den Einsatz durch staatliche Institutionen (insbesondere Strafverfolgungsbehörden und Geheimdienste) entwickelt wird. Die Entwicklung findet durch Staaten selbst[47], durch private Unternehmen oder gemeinsam statt[48]. Diese Spyware ist lediglich wenigen Akteur*innen und in der Regel nur für Millionen-Beträge zugänglich[49].

Diese Art von Spyware stellt in einigen Aspekten einen Gegensatz zu Stalkerware da. Während die Endanwender*innen dieser Spyware nicht notwendigerweise über hohes technisches Wissen verfügen, ist dies bei den Entwickler*innen der Spyware durchaus der Fall. Bei den in den vergangenen Jahren bekannt gewordenen Fällen handelt es sich um hochkomplexe Angriffe, bei denen teils mehrere bis dahin unbekannte Sicherheitslücken (0-day exploits) zum Einsatz kommen[50]. Anders als bei Stalkerware ist nicht immer ein physischer Zugriff auf das angegriffene System nötig, sondern eine Infektion kann aus der Ferne und ohne Zutun der Angegriffenen möglich sein (sog. „zero click infections“)[51].

Die Hersteller solcher Spyware werben damit, dass ihre Produkte keinerlei Spuren auf den Geräten hinterlassen[52, S. 9]. Dafür werden einzelne Features nur aktiviert, wenn dies unauffällig möglich ist, Infektionen abgebrochen, wenn ein Risiko für eine Entdeckung erkannt wird[53] und gezielt versucht, die Spuren einer Infektion zu verwischen[50]. Dennoch wurden auf infizierten Geräten wiederholt Spuren und Hinweise auf Infektionen gefunden[2], [15], [50], [51], [53], [54], [55], [56], [57].

Die hohen technischen Fähigkeiten in Kombination mit dem tiefen Eindringen in die Systeme stellt bei der Suche nach dieser Art von Spyware eine besondere Schwierigkeit dar. Da die Anwendungen tief ins System eindringen und durch das

Ausnutzen von Sicherheitslücken Kernel- / `root`-Rechte erlangen[50], kann nicht ausgeschlossen werden, dass Systemdienste manipuliert werden, um ein Erkennen der Spyware zu erschweren. So könnte beispielsweise der ADB-Dienst derart manipuliert werden, dass er beim Extrahieren bestimmter Dateien gezielt Spuren der Spyware entfernt, die in diesen normalerweise zu finden wären.

Im Rahmen der Literaturrecherche konnte keine Veröffentlichung gefunden werden, in der gezielt Systemdienste, über die Daten für eine Spywareanalyse vom Gerät gesammelt werden, manipuliert wurden. Stattdessen setzen die Spyware-Autor*innen darauf, möglichst wenig Spuren zu hinterlassen, die hinterlassenen Spuren gut zu tarnen, beispielsweise sind bei Pegasus eine Reihe von Prozessen ähnlich benannt wie existierende Systemprozesse[15], oder die Spuren im Nachgang zu entfernen[50]. Teils wurden dabei neue Spuren hinterlassen[56, S. 6].

Dies scheint naheliegend, da das Manipulieren der Systemdienste mit signifikant höherem Aufwand verbunden wäre, als schlicht einzelne Spuren nachträglich zu verwischen oder zu entfernen. Zudem sind Daten teils über verschiedene Wege abrufbar, die alle gleichartig manipuliert werden müssten, ohne dabei selbst wieder neue Spuren zu erzeugen. Außerdem wäre eine derartige Manipulation der Systemdienste nur möglich, solange die Schadsoftware auf dem System aktiv ist. Wird diese inaktiv, beispielsweise durch einen Abbruch der Infektion, das Schließen von genutzten Sicherheitslücken oder – bei nicht-persistenter Spyware – einen Neustart des Geräts, wären die Spuren wieder auffindbar.

Daher wird im Rahmen dieser Arbeit davon ausgegangen, dass die Systemdienste, über die Daten von den Geräten gesammelt werden können (z. B. Backup, ADB), weiterhin in ihrem Gesamtbild vertrauenswürdig sind und die über die gesammelten Daten denen auf dem Gerät im Wesentlichen entsprechen (was etwaige Manipulationen an den Dateien auf dem Gerät nicht ausschließt).

Daraus ergibt sich folgende weitere Anforderung an die Software:

1. Die Software sollte solche Daten sammeln, die sich in vergangenen Analysen als hilfreich erwiesen haben, namentlich die in [Datenquellen](#) genannten Daten. Mindestens jedoch muss sie alle dafür nötigen Funktionalitäten unterstützen und leicht um weitere Datenquellen erweitert werden können.

3.2 Nutzer*innen-Modell

Sowohl bei Stalkerware als auch bei State-Sponsored-Malware lassen sich wenige Annahmen über die Betroffenen treffen – und damit auch über die potenziellen Nutzer*innen des hier entwickelten Tools. Insbesondere kann im Allgemeinen

nicht vorausgesetzt werden, dass die Nutzer*innen ein weitreichendes technisches Verständnis der Software oder große Fähigkeiten in der forensischen Analyse der durch die Software gesammelten Daten haben. Im Gegenteil berichtet eine Studie[58], dass sowohl Betroffene als auch Hilfestellen in Fällen von Gewalt innerhalb einer Beziehung ihr fehlendes technisches Wissen beklagen.

Die Software soll daher Datenpakete zusammenstellen, die anschließend anderen Gruppen zur Analyse übersandt werden können. Dabei wird der sichere Übertragungsweg zur analysierenden Gruppe als gegeben vorausgesetzt und ist nicht Teil dieser Arbeit. Dennoch soll die Software darauf achten, Übertragungen der Daten möglichst einfach zu machen und den Schutz der Daten auf dem Transportweg unterstützen.

Daraus ergeben sich folgende weiteren Anforderungen an die Software:

1. Die Nutzung muss ohne technisches Verständnis der Funktionsweise der Software möglich sein:
 - a. Dafür müssen für alle Schritte Anleitungen oder Erklärtexpte bereitgestellt werden.
 - b. Soweit innerhalb des Datensammelungsprozesses Einstellungen getätigt werden können, sollen Standardwerte für diese an die Software übergeben werden können, sodass eine Vor-Konfiguration möglich ist.
 - c. Die Nutzung muss mittels üblicher Software möglich sein. Dies sei definiert als Software, die auf der Mehrzahl der Geräte der Nutzer*innen bereits installiert ist.
2. Um eine Übertragung weiterer Daten durch die Spyware zu erschweren, muss es möglich sein, die Daten vom Gerät zu sammeln, auch wenn Mobilfunk und WiFi deaktiviert sind oder sich das Gerät im *Flugmodus* befindet.

3.3 Umgang mit den gesammelten Daten

Ebenso wie der ggf. nötige sichere Übertragungsweg zu einer Stelle, die die Analyse der gesammelten Daten durchführt, ist auch der Umgang mit den gesammelten Daten nicht Gegenstand dieser Arbeit. Dennoch soll im Folgenden eine oberflächliche Analyse dieses Umgangs unter den Schutzzielen der CIA-Triade der Informationssicherheit[59], also Vertraulichkeit (*Confidentiality*), Integrität (*Integrity*) und Verfügbarkeit (*Availability*) erfolgen, um daraus Anforderungen an die Software abzuleiten, die ein Erreichen dieser Schutzziele erleichtern kann.

3.3.1 Vertraulichkeit

DIN EN ISO/IEC 27000:2020-06[60] definiert Vertraulichkeit als die „Eigenschaft, dass Information unbefugten Personen, Entitäten oder Prozessen nicht verfügbar gemacht oder offengelegt wird“. In diesem Fall besteht das Schutzziel daraus, eine Preisgabe der gesammelten Daten an Unberechtigte zu verhindern.

Um die Gewährleistung der Vertraulichkeit der Daten zu erleichtern, kann die entwickelte Software folgende Anforderungen erfüllen:

1. Die Daten sollten nur auf dem Gerät der Nutzer*in gespeichert werden. Eine Übertragung oder Verarbeitung auf anderen Geräte, insbesondere ggf. Server auf denen die Software gehostet wird, soll nicht stattfinden.
2. Es muss die Möglichkeit bestehen, die Daten mittels eines asymmetrischen Verfahrens zu verschlüsseln.

So kann ein Schlüssel verwendet werden, dessen privater Teil den Nutzer*innen der Software nicht bekannt ist, sondern lediglich der Gruppe, die die Daten später auswerten soll. Dies verhindert eine Preisgabe der Daten durch die Nutzer*in, auch unter Druck. Aus Geschwindigkeitsgründen bieten sich hier hybride Verfahren wie **age** an.

3.3.2 Integrität

DIN EN ISO/IEC 27000:2020-06[60] definiert Integrität als die „Eigenschaft der Richtigkeit und Vollständigkeit“.

In diesem Fall besteht das Schutzziel daraus, eine unerkannte Veränderung der gesammelten Daten nach dem Sammeln zu verhindern.

Um dies zu erleichtern, kann die Software folgende Anforderungen erfüllen:

1. Die Daten sollten mit einer Prüfsumme oder ähnlichen Verfahren vor einer unbeabsichtigten Veränderung der Daten, beispielsweise durch *bit rot*, geschützt werden.
2. Es kann eine Prüfsumme der gesammelten Daten erstellt und der Nutzer*in angezeigt werden.

Diese Prüfsumme kann auf einem sicheren Kanal unabhängig von den gesammelten Daten übertragen werden, so kann eine Veränderung der Daten während der Speicherung oder Übertragung erkannt werden.

3.3.3 Verfügbarkeit

DIN EN ISO/IEC 27000:2020-06[60] definiert Verfügbarkeit als die „Eigenschaft zugänglich und nutzbar zu sein, wenn eine befugte Entität Bedarf hat“.

Die Verfügbarkeit der gesammelten Daten nach der Sammlung liegt nicht im Einflussbereich der Software, da die Daten nach der Sammlung auf dem System der Nutzer*in gespeichert werden. Jedoch kann die Software eine Sicherstellung der Verfügbarkeit der gesammelten Daten erleichtern. Dafür kann die Software die folgende Anforderung erfüllen:

1. Die Größe der gesammelten Daten sollte möglichst klein gehalten werden, indem
 - eingestellt werden kann, welche Daten gesammelt werden und
 - die Daten komprimiert werden.

4 Praktische Lösung

Die im Rahmen dieser Arbeit entwickelte Software „AndroidQF-Web“ dient als Alternative zu AndroidQF. Statt als Kommandozeilen-Tool läuft sie im Browser und spricht über WebUSB mit dem ADB-Daemon auf dem Smartphone. Als Kompressions- sowie Verschlüsselungssystem kommen ebenso wie bei AndroidQF ZIP und age zum Einsatz.

Im Folgenden werden zunächst die für die Umsetzung relevanten Schnittstellen und eingesetzten Bibliotheken aufgeführt. Anschließend wird die erfolgte Implementation vorgestellt und dabei einige dabei aufgetretene Probleme beleuchtet.

4.1 Relevante Schnittstellen

4.1.1 WebUSB

WebUSB ist eine von Google zur Standardisierung vorgeschlagene Schnittstelle[61]. WebUSB soll es Webseiten erlauben, auf sichere Art und Weise direkt auf USB-Geräte zuzugreifen. Zum Zeitpunkt dieser Arbeit ist WebUSB noch im Zustand eines „Draft Community Group Report“, wird allerdings bereits durch die aktuellen Versionen der Chromium-basierten Desktop-Browser unterstützt[62].

WebUSB stellt eine low-level Schnittstelle zu einzelnen USB-Geräten bereit. Die über USB übertragenen Protokolle müssen darauf basierend selbst implementiert werden.

In Browsern, die WebUSB unterstützen, können Webseiten über die Funktion `navigator.usb.requestDevice` Zugriff auf ein USB-Gerät anfordern. Dies muss für jedes Gerät und jede Webseite durch die Browser-Nutzer*in freigegeben werden. Nach der Freigabe ist der Zugriff bis zu ihrem Widerruf möglich.

Um einen Missbrauch zu erschweren, ist der Aufruf einer Reihe von Web-APIs, darunter auch `requestDevice`, nur im sogenannten `Transient activation`-Zustand[63] zulässig. Dieser Zustand ist gegeben, wenn kürzlich¹⁰ eins der im HTML Standard genannten „activation triggering input event[s]“ aufgetreten ist[65], die Nutzer*in also mit der Website interagiert hat.

¹⁰Zum Zeitpunkt dieser Arbeit muss diese Interaktion in Chromium-basierten Browsern innerhalb der letzten Sekunde erfolgt sein[64].

Zudem ist ein Zugriff auf die WebUSB-Schnittstelle nur aus einem Secure Context[66] möglich. Dies ist insbesondere der Fall wenn eine Seite local (von localhost, 127.0.0.1, ::1, ...) sowie via HTTPS ausgeliefert wird.

Weiterhin kann auf eine Reihe von Geräten nur mit erweiterten Zugriffsrechten zugegriffen werden, welche für normale Webseiten nicht gegeben sind. Diese erhöhte Zugriffshürde gilt für alle Geräte bestimmter geschützter Geräteklassen, wie z. B. Eingabe- und Speichergeräte, sowie für Geräte, die auf einer *Blocklist* stehen[61].

Nach der Zugriffserteilung durch die Nutzer*in kann die Website mit dem USB-Gerät interagieren. Es können verschiedene Eigenschaften des Geräts ausgelesen werden (z. B. die angebotenen Konfigurationen und ihre Interfaces über das `USBDevice.configurations`-Attribut oder der Name des Herstellers über `USBDevice.manufacturerName`).

Für diesen Anwendungsfall relevante Funktionen sind das Auswählen einer Konfiguration `USBDevice.selectConfiguration` sowie der exklusive Zugriff auf ein Interface `USBDevice.claimInterface`. Nachdem der exklusive Zugriff auf ein Interface erlangt wurde, können Daten an die Endpoints des USB-Geräts gesendet bzw. von diesem empfangen werden. Hierfür existieren Funktionen für die verschiedenen möglichen Arten des Endpoints (Control, Interrupt, Isochronous und Bulk), wobei für diese Arbeit lediglich die Übertragungs-Funktionen für Bulk-Endpoints benötigt werden (`USBDevice.transferIn/USBDevice.transferOut`).

4.1.2 Streams

Die Streams-API[67] stellt standardisierte Methoden bereit, um kontinuierliche Datenströme zu verarbeiten und wurde besonders für die Abbildung von IO-Operationen entworfen. Die Schnittstellen `ReadableStream`, `WritableStream` und `TransformStream` sind abstrahierte Interfaces für Quellen, Senken und Pipes. Diese können durch Funktionen wie `pipeTo` oder `pipeThrough` miteinander verbunden werden. Das Vorhalten noch nicht gelesener/geschriebener Daten (`Queueing`) sowie die Steuerung der Strom-Geschwindigkeit (`Backpressure`) geschieht automatisch durch den Browser, kann jedoch auch durch die Website beeinflusst werden.

Die Streams-API erlaubt es, Eingabedaten aus einem `ReadableStream` direkt mittels einem oder mehreren `TransformStreams` zu verarbeiten, beispielsweise zu komprimieren oder verschlüsseln, und in einen `WritableStream` zu schreiben, ohne dass diese Daten vollständig zwischengespeichert werden müssen. Dies erlaubt eine effizientere Nutzung der Systemressourcen, da eine Weiterverarbeitung bereits parallel zum Lesen der Daten möglich ist. Zudem können so Datenmengen verarbeitet werden, die die Speicherkapazitäten überschreiten würden.

4.1.3 File System Access

Die `File System Access-API`[68] ist ein von Google vorgeschlagene Erweiterung des „File System“-Standards[69]. Diese Schnittstelle erlaubt es Webseiten, auf das Dateisystem der Nutzer*in zuzugreifen. Hierzu kann die Website Zugriff auf eine Datei¹¹ anfragen, entweder nur-lesend oder lesend-schreibend. Daraufhin präsentiert der Browser der Nutzer*in einen Dateiauswahldialog. Wird hier eine Datei ausgewählt, erhält die Website ein `FileSystemFileHandle`-Objekt zurück. Ggf. müssen nun noch weitere Zugriffsrechte angefordert werden, um die Datei auch schreiben zu können. Ist dies passiert, kann über die `FileSystemFileHandle.createWritable()`-Methode ein `WritableStream`-Objekt erlangt werden, durch welches dann in die Datei geschrieben werden kann.

Ebenso wie bei WebUSB ist eine Nutzung dieser API nur aus einem Secure Context und mit Transient Activation möglich. Zudem können Browser den Zugriff auf bestimmte Dateien, die als zu sensibel oder gefährlich angesehen werden, gänzlich verbieten.

Zum Zeitpunkt dieser Arbeit ist die *File System Access-API* noch im Zustand eines „Draft Community Group Report“, wird allerdings bereits durch die aktuellen Versionen der Chromium-basierten Desktop-Browser unterstützt[70].

4.1.4 Android Debug Bridge

Im Folgenden werden die Funktionsweise der bereits eingeführten Android Debug Bridge (ADB) und die für diese Arbeit relevanten Dienste und ihre Protokolle näher erläutert. ADB ist eine ursprünglich zur Android-Entwicklung gedachte Schnittstelle, die einen Zugriff auf das Android-Gerät von einem anderen Gerät (Host) aus erlaubt. Sie besteht aus zwei Komponenten: Einem ADB-Daemon (dem `adb`) auf dem Smartphone sowie einem ADB-Client auf dem Host-Computer. Die Verbindung zwischen Daemon und Client¹² ist über verschiedene Transportprotokolle, bspw.

¹¹Im Rahmen dieser Arbeit werden nur die Schnittstellen zum Speichern einzelner Dateien genutzt, daher werden diese beschrieben. Es existiert analoge Funktionalität für Ordner.

¹²In ADB-Entwicklungs-Setup besteht üblicherweise keine direkte Verbindung zwischen Client und Daemon. Stattdessen wird auf dem Host-Computer ein ADB-Server-Prozess gestartet. Dieser erkennt verbundene Geräte, z. B. via USB oder TCP, und kümmert sich um die direkte Kommunikation mit diesen. Die ADB-Clients wiederum verbinden sich mit diesem Server, der die Kommunikation mit dem Daemon weiterleitet[71]. So können mehrere Client-Prozesse gleichzeitig mit einem Gerät kommunizieren. Die im Rahmen dieser Arbeit entstandene Software kommuniziert als Client direkt mit dem Daemon auf dem Gerät, ohne einen zwischengeschalteten Server. Da der ADB-Server für diese Arbeit keine Rolle spielt, wird vereinfachend davon ausgegangen, dass Client und Daemon stets direkt miteinander kommunizieren.

via USB oder über die Netzwerkschnittstellen des Geräts, möglich. Seit Android 11 (API-Level 30) ist zudem ein verschlüsselter Netzwerkanal möglich[72].

Über das gewählte Transportprotokoll sind wiederum weitere Protokollschichten implementiert[71]. Die unterste Protokollschicht bildet dabei das ADB-Protokoll, welches Handshaking und Multiplexing übernimmt. Im Rahmen des Handshakes muss die Nutzer*in dem Host Zugriff auf das Gerät erlauben. Damit dies nicht bei jeder Verbindung nötig ist, überträgt der Client dabei seinen öffentlichen Schlüssel¹³, dem von der Nutzer*in einmaliges oder dauerhaftes Vertrauen ausgesprochen werden kann. Bei dauerhaftem Vertrauen speichert das Gerät den Schlüssel und bei der nächsten Verbindung kann sich Clients gegenüber dem Daemon mittels eines *Challenge-Response-Verfahrens* authentifizieren. Ist dies erfolgreich, so ist keine Bestätigung durch die Nutzer*in mehr nötig.

Das Multiplexing innerhalb des ADB-Protokolls ermöglicht es, über eine Transport-Verbindung mehrere parallele bi-direktionale Datenströme (**Streams**) zu übertragen[73]. Um einen neuen *Stream* zu öffnen, sendet der Client eine entsprechende Anfrage an den Daemon. Diese enthält neben dem ID des Streams auf dem Client das gewünschte Ziel des Datenstroms. Das Ziel kann ein TCP- oder UDP-Port, ein Socket auf dem Gerät etc. sein, aber auch ein über ADB zur Verfügung gestellter Dienst (**Service**). [74]

Wird ein solcher Dienst als Ziel angegeben, öffnet der ADB-Daemon ein neues Socket-Paar und startet einen neuen Thread, der über dieses Socket-Paar mit dem Client kommuniziert und die Funktionalität des Dienstes bereitstellt. Die Dienste implementieren teils eigene Protokolle. Im Rahmen dieser Arbeit sind die folgenden Services relevant:

- **shell:**

Der **shell**-Service erlaubt es, wahlweise ein Shell-Kommando auf dem Gerät auszuführen oder eine interaktive Shell zu öffnen. Vom Client über den Stream gesendete Daten werden vom Daemon in die Standardeingabe des Prozesses geschrieben. Vom Prozess in die Standardausgabe sowie die Standardfehlerausgabe geschriebene Daten werden vom Daemon über den Stream zum Client übertragen. Der Rückgabewert des Prozesses wird nicht übertragen[74].

- **shell,v2:**

Der **shell,v2**-Service ist eine Variante des **shell**-Dienstes, der ab Android-Version 7.0 (API-Level 24) verfügbar ist. Auch dieser erlaubt es, Shell-Kommandos auf dem Gerät auszuführen, implementiert allerdings ein eigenes

¹³ADB verwendet hier RSA-Schlüsselpaare mit 2048 Bit Länge.

Protokoll, das sog. **Shell Protokoll**. Dieses Protokoll erlaubt es, Standard-Ausgabe und Standard-Fehler-Ausgabe zu trennen und den Rückgabewert des ausgeführten Kommandos zu erhalten[74].

Dieses Protokoll besteht aus über den bestehenden Stream gesendeten Paketen im Type-Length-Value-Schema (TLV). Diese bestehen jeweils aus einem Byte für den Typ des Pakets, vier Bytes für die Länge der enthaltenen Daten sowie die Daten selbst:

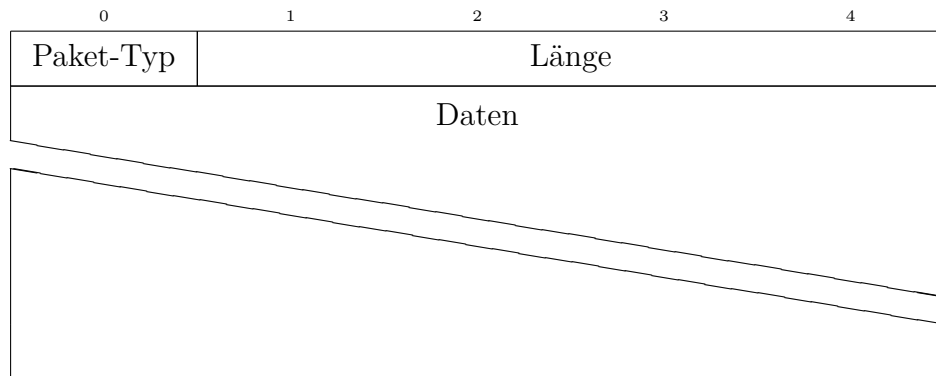


Abbildung 3: Paket des Shell-Protokolls

Die Bedeutung der Daten ergibt sich aus dem Paket-Typ:

Paket-Typ	Bedeutung
0	Daten der Standardeingabe
1	Daten der Standardausgabe
2	Daten der Standardfehlerausgabe
3	Exit-Code des Kommandos

Pakete des Typs 0 werden nur vom Client gesendet, Pakete der Typen 1, 2 oder 3 nur vom Daemon.

- **backup:**

Über diesen Service kann mit dem Android Backup Manager interagiert werden. So können Backups einzelner oder aller Apps auf dem System angelegt werden – hierbei gelten jedoch die in [Backups](#) dargelegten Einschränkungen.

Intern ruft der Backup-Service das `bu`-Kommando auf dem System auf[75], welches dann den Android Backup Manager anspricht[76]. Beim Öffnen des Streams werden dem `backup`-Service die gewünschten Optionen übergeben,

die direkt an `bu` weitergegeben werden. Hier relevant sind insbesondere die Möglichkeit, entweder einzelne Paketnamen anzugeben, um nur diese zu backupen, oder mittels `-all` alle backupbaren Pakete zu backupen[77].

Das Backup muss zunächst durch die Nutzer*in mittels eines auf dem Gerät angezeigten Bestätigungs-Screens zugelassen werden. Der Service nutzt kein eigenes Protokoll für die Übertragung des Backup-Archivs, sondern überträgt das Backup in einem eigenen, tar-basierten Format direkt über den bestehenden Stream.

- **sync**

Der Sync-Service stellt eine Reihe von Dateioperationen bereit und nutzt hierfür ein eigenes Protokoll. Über diesen Dienst ist es unter anderem möglich, Dateien vom Gerät zu lesen, auf das Gerät zu schreiben, Dateieigenschaften und Verzeichnisinhalte abzurufen[78].

Zum Zeitpunkt dieser Arbeit unterstützt die aktuelle Version der `adb-Daemon`-Implementation von Google zehn mögliche Operationen innerhalb des Sync-Protokolls[79, Z. 827]. Da die in dieser Arbeit entwickelte Software nur die `RECV`-Operation (Übertragung des Inhalts einer Datei zum Client) verwendet, wird der Fokus auf diese Operation gelegt.

Das Sync-Protokoll verwendet Pakete im folgenden Schema:

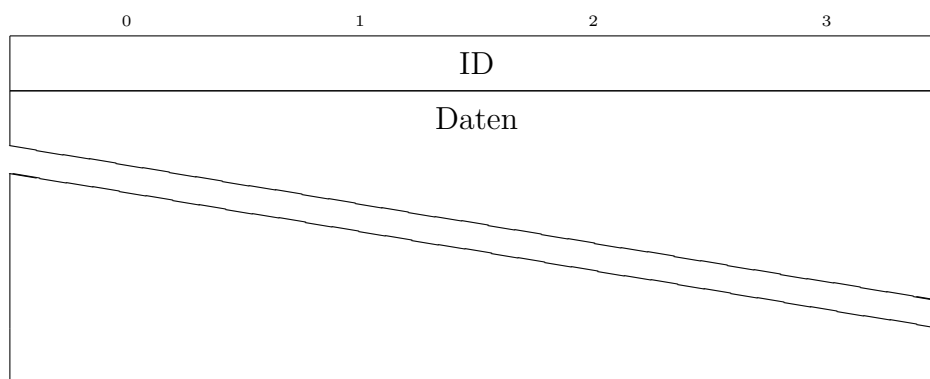


Abbildung 4: Paket des Sync-Protokolls

Um eine Dateioperation durchzuführen, sendet der Client einen sogenannten `SyncRequest`, ein Paket, bei dem das `ID`-Feld auf die gewünschte Operation gesetzt ist und die Daten aus einem String-Argument für diese Operation mit vorangestellter Länge des Strings als 32-Bit-Integer bestehen. Für das Übertragen einer Datei zum Client wird im `SyncRequest` das `ID`-Feld auf

RECV gesetzt und das String-Argument enthält den Pfad der zu übertragenden Datei.

Der Daemon antwortet darauf mit einem oder mehreren Paketen, sogenannten `SyncResponses`. Der Inhalt der Daten ist hierbei abhängig von der Operation.

Bei der RECV-Operation sendet der Daemon 0 oder mehr `SyncResponses` mit ID `DATA`, gefolgt von einer `SyncResponse` mit ID `FAIL` oder `DONE`:

- `DATA-SyncResponses` enthalten jeweils einen Teil der zu übertragenden Datei. Im Daten-Feld enthält 32-Bit-Integer mit der Länge der darauf folgenden Datei, gefolgt von eben diesen Daten.
- Eine `FAIL-SyncResponse` wird gesendet, wenn ein Fehler aufgetreten ist, beispielsweise wenn die Datei nicht geöffnet werden konnte oder es einen Lese-Fehler gab. Das Datenfeld enthält in diesem Fall einen String mit dem Fehlergrund, mit vorangestelltem 32-Bit-Integer mit der Länge des Fehlergrunds sowie den Fehlergrund selbst.
- Eine `DONE-SyncResponse` markiert das erfolgreiche Ende der Operation. Das Datenfeld enthält in diesem Fall vier Null-Bytes.

Nach einem `FAIL`- oder `DONE`-Paket werden keine Pakete mehr im Rahmen dieser RECV-Operation übertragen.

4.2 Eingesetzte Bibliotheken

Zur Entwicklung der Software wurde eine Reihe von Bibliotheken eingesetzt, die im Folgenden vorgestellt werden. Teilweise wurden diese Bibliotheken um nötige Funktionalität erweitert.

4.2.1 wadb

Das Google-Chrome-Team hat im Rahmen der „Google Chrome Labs“ eine Implementation von ADB über WebUSB in TypeScript veröffentlicht[80]. Die Bibliothek enthält die nötigen Funktionen für das Finden von und Verbinden mit angeschlossenen Android-Geräten sowie eine Implementation des ADB-Protokolls sowie der Protokolle einiger Services. Hierfür stellt `wadb` die `AdbClient`-Klasse bereit, die eine ADB-Verbindung abbildet.

Die Bibliothek wurde im Rahmen dieser Arbeit um folgende Funktionalität erweitert:

- der `shell,v2`-Service

`wadb` unterstützte bereits den `shell`-Service, welcher allerdings den Rückgabewert des ausgeführten Befehls nicht an den Client überträgt. Zudem werden der Inhalt der Standardausgabe und der Standardfehlerausgabe gemeinsam übertragen, sodass nicht unterschieden werden kann, welche Daten durch den Befehl auf welcher Ausgabe ausgegeben wurden.

Im Rahmen dieser Arbeit wurde `wadb` um Support für den `shell,v2`-Service erweitert. Hierfür wurde eine neue `ShellV2`-Klasse zu `wadb` hinzugefügt, welche einen `ADB-Stream` übergeben bekommt^[81]. Die Instanzen der Klasse stellen über die Attribute `stdout`, `stderr` jeweils einen `ReadableStream` bereit, um die Standard-(Fehler-)Ausgaben des ausgeführten Kommandos zu lesen. Das Attribut `exitCode` enthält einen `Promise`, der zum Rückgabewert des ausgeführten Kommandos auflöst.

Aus dem übergebenen `ADB-Stream` liest diese Klasse kontinuierlich Pakete und dekodiert die darin enthaltenen Shell-Pakete¹⁴. Die Daten der beiden Ausgaben werden in die Queue des entsprechenden `ReadableStream` geschrieben, der Rückgabewert wird zum Resolven des Promises genutzt. Wird der unterliegende `ADB-Stream` geschlossen, so werden auch die beiden `ReadableStreams` der beiden Ausgaben geschlossen.

- der `backup`-Service

Die erfolgte Erweiterung von `wadb` besteht aus einer Funktion mit einem `String`-Argument, welches dem `Backup-Service` übergeben wird. Hierüber kann gesteuert werden, welche Anwendungen und welche Daten dieser Anwendungen gebackupt werden sollen. Die Funktion öffnet einen neuen `backup-Stream` mit den entsprechenden Argumenten und gibt einen `ReadableStream` zurück. Die vom Daemon zurückerhaltenen Daten werden kontinuierlich gelesen und über den `ReadableStream` zur Verfügung gestellt.

```

1  async backup(args: string): Promise<ReadableStream> {
2      const stream = await Stream.open(this, `backup:${args}`,
3      this.options);
4      return new ReadableStream({
5          async start(controller): Promise<void> {
6              while (true) {
7                  const cmd = await stream.read();
8                  if (cmd.header.cmd == 'CLSE') {

```

¹⁴Ein über das `ADB`-Protokoll gesendetes Paket kann mehrere Pakete des `Shell`-Protokolls enthalten.

```

8         break
9     } else if (cmd.header.cmd == 'WRTE') {
10        if (cmd?.data?.buffer) {
11            controller.enqueue(new
12                Uint8Array(cmd.data.buffer));
13        }
14        await stream.write('OKAY');
15    }
16    await stream.close();
17    controller.close()
18 }
19 })
20 }

```

- Erweiterung der Sync-Implementation

Die bestehende Implementation der RECV-Operation des Sync-Protokolls, also des Übertragens einer Datei vom Gerät an den Client, speicherte zunächst den gesamten Inhalt der Datei zwischen und gab diesen als Blob zurück. Im Rahmen dieser Arbeit wurde die Implementation um die Nutzung von `ReadableStreams` erweitert. Hierzu wurde die Funktion `pullAsStream` hinzugefügt, die einen `Promise` zurückgibt. Wurde der `SyncRequest` erfolgreich übertragen, resolved dieser zu einem `ReadableStream`, der kontinuierlich `SyncResponses` vom Daemon liest und den darin enthaltenen Datei-Inhalt bereitstellt. So kann der Dateiinhalt ohne Zwischenspeichern gelesen werden.

Die im Rahmen dieser Arbeit entwickelten Erweiterungen von `wadb` wurden den `wadb-Maintainer`*innen zur Verfügung gestellt [82].

4.2.2 zip.js

`zip.js` ist eine Bibliothek zum Lesen und Schreiben des Zip-Dateiformats in JavaScript [83].

Um die gesammelten Daten als eine Datei bereitzustellen, speichert die im Rahmen dieser Arbeit entstandene Software – ebenso wie `AndroidQF` – diese in einem Zip-Archiv.

In der JavaScript-Welt gibt es eine Reihe von Bibliotheken für das Erstellen von Zip-Archiven. Unter diesen wurde `zip.js` ausgewählt, da diese Bibliothek

1. gute Unterstützung für die Streams-API hat,

2. Archive bei der Erstellung nicht vollständig im Speicher halten muss, sondern kontinuierlich schreiben kann und
3. selbst keine weiteren Laufzeitabhängigkeiten benötigt.

4.2.3 age / typage

`age` ist ein seit 2019 von *Fillipo Valsorda et al.* entwickeltes und 2021 in Version 1.0.0[84] veröffentlichtes Datei-Verschlüsselungsformat. `age` nutzt eine hybride Verschlüsselung aus einer blockweisen Verschlüsselung mit ChaCha20-Poly1305[85] mit jeweils einem Schlüssel pro Datei, welcher mit den öffentlichen X25519-Schlüsseln der Empfänger*innen oder Passphrasen¹⁵ gewrappt und als Header der verschlüsselten Datei vorangestellt wird.

`typage`[86] ist eine `age`-Implementation in TypeScript. Falls möglich, verwendet `typage` die native WebCrypto-API des Browsers[87]. Für die zugrundeliegenden kryptographischen Primitiven wird `noble`[88] verwendet, eine Sammlung von TypeScript-Bibliotheken für kryptographische Primitiven.

Seit Version 0.2.4 unterstützt `typage` die Streams-API und kann Daten verschlüsseln, ohne diese zunächst vollständig zwischenzuspeichern[89].

4.2.4 React

React ist ein von Meta entwickeltes open-source¹⁶ JavaScript-Framework zur Frontend-Entwicklung[94]. Die Website-Entwicklung mit React basiert auf dem Erstellen sogenannter *Components*, wiederverwendbarer interaktiver UI-Blöcke. Hierfür kommt eine Erweiterung von JavaScript namens JSX zum Einsatz. Diese *Components* können dann zu immer komplexeren Elementen bis hin zu ganzen Webseiten zusammengesetzt werden.

4.2.5 Tailwind

Tailwind ist ein open-source¹⁶ CSS-Framework, welches auf sog. *Utility Classes* basiert[95]. Dabei handelt es sich um kleine CSS-Klassen, die jeweils nur eine Handvoll CSS-Attribute setzen. Beispielsweise gibt es Klassen, um die Hintergrundfarbe oder Schriftgröße eines Elements zu setzen. Zudem gibt es die Möglichkeit, Attribute

¹⁵Über Plugins unterstützt `age` auch andere Schlüssel-Typen, die allerdings für diese Arbeit keine Rolle spielen.

¹⁶React, Tailwind, Redux und Redux Toolkit stehen alle unter MIT-Lizenz[90], [91], [92], [93].

nur zu setzen, wenn ein Element beispielsweise fokussiert ist oder der Dark-Mode des Browsers aktiv ist. Diese kleinen *Utility Classes* können dann zur gewünschten Gestaltung der Seite kombiniert werden.

In Kombination mit React erlaubt es Tailwind, schnell einheitlich gestaltete Komponenten zu erstellen.

4.2.6 Redux & Redux Toolkit

Redux ist eine open-source¹⁶ JavaScript-/TypeScript-Bibliothek zur anwendungsweiten Zustandsverwaltung[96]. Sie erlaubt es, einen globalen State zu definieren, auf den aus verschiedenen Komponenten zugegriffen werden kann.

Redux Toolkit ist eine ebenfalls von den Redux-Entwickler*innen entwickelte open-source¹⁶ Bibliothek, die übliche Muster beim Einsatz von Redux vereinfacht[97]. Redux Toolkit bringt unter anderem Funktionen mit, um den Anwendungszustand in sogenannte „*Slices*“ aufzuteilen, die jeweils nur den nötigen Zustand einer Teilfunktion der Anwendung enthalten[98].

4.2.7 TypeScript

TypeScript ist eine von Microsoft entwickelte Programmiersprache, die JavaScript um statische Typisierung erweitert[99]. TypeScript wird nach JavaScript kompiliert. JavaScript-Code ist stets auch gültiger TypeScript-Code, sodass bestehende JavaScript-Bibliotheken und -Schnittstellen auch in TypeScript-Code verwendet werden können.

React, Tailwind, Redux und TypeScript wurden aufgrund bestehender Erfahrung und ihrer großen Verbreitung eingesetzt.

4.3 Implementation



Basierend auf den vorgestellten Bibliotheken wurde eine TypeScript-Anwendung unter Verwendung des React-Frameworks namens „AndroidQF-Web“ entwickelt, welche unter <https://github.com/pajowu/androidqf-web> abrufbar ist. Die in dieser Arbeit beschriebene Version von AndroidQF-Web ist der Code-Stand des Commits [2715fb11c0cd0e0015c50966f6cc738eaba7ec9c](https://github.com/pajowu/androidqf-web/commit/2715fb11c0cd0e0015c50966f6cc738eaba7ec9c).

AndroidQF-Web besteht aus drei Teilen, die im Folgenden näher beschrieben werden: Module, Datenspeicherung und UI.

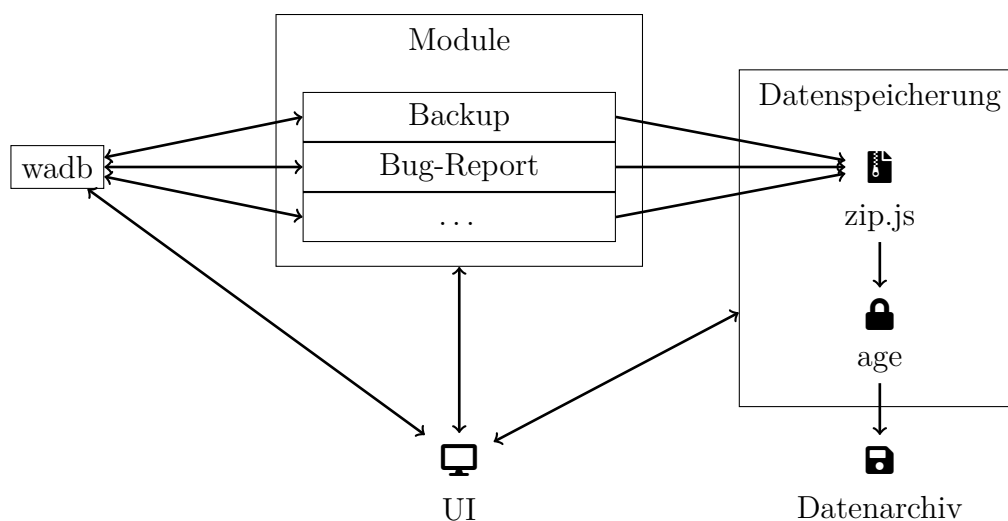


Abbildung 5: Systemarchitektur von AndroidQF-Web¹⁷

4.3.1 Module

Die Module bilden das Herzstück der Datensammlung. Sie entsprechen jeweils einer Datenquelle und enthalten die nötige Logik für Konfiguration und Datensammlung.

Module implementieren das `Module`-Interface. Dieses besteht aus

- `render`: einer Funktion, die *Components* für die Konfiguration des Moduls zurückgibt,
- `run`: einer Funktion, die das eigentliche Sammeln der Daten ausführt und

¹⁷Icons aus FontAwesome 5, Lizenz: SIL OFL 1.1

- **name**: einem String mit dem Namen des Moduls, der Nutzer*innen angezeigt wird.

```

1 export interface Module {
2   render: () => JSX.Element;
3   run: ModuleRunFunction;
4   name: string;
5 }

```

Zudem können Module einen Zustands-*Slice* definieren, der Teil des globalen Anwendungszustands wird. So kann die Konfiguration des Moduls gespeichert werden.

Die **run**-Funktion bekommt jeweils eine Instanz der **Acquisition**, **AdbClient** und **RootState**-Klasse sowie eine **errorCallback**-Funktion übergeben. **Acquisition** ist die im Abschnitt „**Datenspeicherung**“ beschriebene Klasse, **AdbClient** die Klasse der wadb-Bibliothek und **RootState** der Redux-State der Anwendung. Der **RootState** beinhaltet den globalen Anwendungszustand, aus welchem das Modul die Modulkonfiguration auslesen kann. Das **errorCallback** kann aufgerufen werden, um bei der Datensammlung entstandene Fehler zurückzugeben, damit diese der Nutzer*in angezeigt werden können.

Daraus ergibt sich folgende Signatur:

```

1 export type ModuleRunFunction = (
2   storage: Acquisition,
3   client: AdbClient,
4   state: RootState,
5   errorCallback: (e: string) => void
6 ) => Promise<void>;

```

Im Folgenden werden exemplarisch drei Module und ihre Funktionsweisen näher erläutert sowie ein Überblick über die weiteren implementierten Module gegeben.

Backup Das Backup-Modul unterstützt, ebenso wie AndroidQF, zwei Modi: die Sicherung aller unterstützten Anwendungen sowie lediglich der SMS-Datenbank.

Diese Auswahl wird als *Slice* im globalen Anwendungszustand gespeichert. Ausgangszustand ist hierbei, lediglich die SMS-Datenbank zu sichern, soweit über den Fragment-Teil der URL nichts anderes übergeben wird. Über die **setMode**-Funktion kann der ausgewählte Modus gesetzt werden.

```

1  enum Mode {
2    OnlySMS = "Only SMS",
3    Everything = "Everything",
4  }
5  export type BackupSlice = { mode: Mode };
6
7  function getInitialState(): BackupSlice {
8    const hashData = getHashData();
9    return {
10     mode:
11       "backup.mode" in hashData && hashData["backup.mode"] ==
12         "everything"
13         ? Mode.Everything
14         : Mode.OnlySMS,
15   };
16 }
17 export const backupSlice = createSlice({
18   name: "backup",
19   initialState: getInitialState,
20   reducers: {
21     setMode: (slice, action: PayloadAction<Mode>) => {
22       slice.mode = action.payload;
23     },
24   });
25
26 const { setMode } = backupSlice.actions;

```

Die render-Funktion zeigt der Nutzer*in eben diese Modus-Auswahl unter Nutzung der Select-Komponente an und setzt den gespeicherten Modus entsprechend:

```

1  export const backupModule: Module = {
2    render: () => {
3      const mode = useAppSelector((state) => state.backup.mode);
4      const dispatch = useAppDispatch();
5
6      return (
7        <>
8          <Select
9            label="Backup Option: "
10           value={mode}

```

```

11         onChange={(e) => dispatch(setMode(e.target.value as
12         Mode))}
13     >
14     {Object.values(Mode).map((x) => (
15         <option key={x} value={x}>
16             {x}
17         </option>
18     ))}
19 </Select>
20 </>
21 );

```

Die `run`-Funktion ruft die im Rahmen dieser Arbeit entwickelte `backup`-Funktion des `AdbClient`s auf. Hierbei übergibt sie je nach ausgewähltem Modus die entsprechenden Argumente für den Backup-Service. Der von dieser zurückgegebene `ReadableStream` wird dem Archiv, durch Aufruf der entsprechenden Funktion der `Acquisition`-Klasse, hinzugefügt.

```

1  run: async (acq: Acquisition, client: AdbClient, state:
2  RootState) => {
3      let service = '';
4      switch (state.backup.mode) {
5          case Mode.OnlySMS:
6              service = 'com.android.providers.telephony';
7              break;
8          case Mode.Everything:
9              service = '-all';
10             break;
11     }
12     const backup = await client.backup(service);
13     await acq.addFileFromReadableStream('backup.ab', backup);

```

Als Name wird `Backup` gesetzt:

```

1  name: 'Backup',
2  };

```

Env Das `Env`-Modul speichert eine Kopie der in der Geräteshell gesetzten Umgebungsvariablen. Da keine Konfiguration nötig ist, definiert dieses Modul keinen `Slice` und die `render`-Funktion gibt nur ein leeres Element zurück:

```

1 export const envModule: Module = {
2   render: () => {
3     return <></>;
4   },

```

Die run-Funktion nutzt die Hilfs-Funktion runShellAndAddToAcquisition, die ein Shell-Kommando auf dem Gerät ausführt und seine Ausgaben und den Rückgabewert in das Datenarchiv schreibt:

```

1   run: async (acq: Acquisition, client: AdbClient, _state:
2     RootState) => {
3     await runShellAndAddToAcquisition(acq, client, 'env', 'env');
4   },

```

Zuletzt wird noch der Name des Moduls gesetzt:

```

1   name: 'Environment',
2 };

```

Logs Das Logs-Modul sucht an einer Reihe von Orten nach Log-Dateien und speichert diese, falls gefunden und möglich, in das Archiv. Dazu wird zunächst eine Liste einzelner Log-Dateien sowie von Log-Verzeichnissen definiert. Diese Listen sind aus dem Code von AndroidQF übernommen[100].

```

1 // taken from https://github.com/mvt-project/androidqf/blob/facb7_
2 a02e30e81dcf5f35d3597c8403b93b47418/modules/logs.go
3 const LOGFILES = [
4   "/data/system/uiderrors.txt",
5   "/proc/kmsg",
6   "/proc/last_kmsg",
7   "/sys/fs/pstore/console-ramoops",
8 ];
9 const LOGDIRS = ["/data/anr/", "/data/log/", "/sdcard/log/"];

```

Anschließend werden zwei Hilfsfunktionen definiert. listFiles sucht auf dem Gerät innerhalb des übergebenen Pfades mittels des find-Kommandos nach Dateien. Fehler werden dabei ignoriert. Die gefundenen Dateien werden zurückgegeben.

addFiles bekommt eine Liste von Dateien übergeben, kopiert diese vom Gerät und schreibt die in das Archiv. Dabei wird die im Rahmen dieser Arbeit entwickelte pullAsStream-Funktion genutzt, um den Dateiinhalt direkt als ReadableStream weiterzuschreiben, ohne ihn vorher vollständig zwischenspeichern.

```

1  async function listFiles(
2    client: AdbClient,
3    path: string
4  ): Promise<Array<string>> {
5    const fileList = await client.shell(`find ${path} -type f 2>
6      /dev/null`);
7    return fileList.split("\n").filter((x) => x !== "");
8  }
9
10 async function addFiles(
11   acq: Acquisition,
12   client: AdbClient,
13   files: string[],
14   errorCallback: (e: string) => void
15 ) {
16   for (const file of files) {
17     try {
18       await acq.addFileFromReadableStream(
19         `logs/${file}`,
20         await client.pullAsStream(file)
21       );
22     } catch (e) {
23       const err = e as Error;
24       errorCallback(err.message);
25     }
26   }
27 }

```

Da keine Konfiguration nötig ist, definiert auch dieses Modul keinen *Slice* und die `render`-Funktion gibt nur ein leeres Element zurück:

```

1  export const logsModule: Module = {
2    render: () => {
3      return <></>;
4    },

```

Die `run`-Funktion speichert zuerst mithilfe der `addFiles`-Hilfsfunktion die bekannten Log-Dateien im Archiv. Anschließend werden die in den Log-Verzeichnissen enthaltenen Dateien aufgelistet und in das Archiv gespeichert.

```

1  run: async (acq: Acquisition, client: AdbClient, _state,
2    errorCallback: (e: string) => void) => {

```

```

2     await addFiles(acq, client, LOGFILES, errorCallback);
3     for (const logdir of LOGDIRS) {
4         const files = await listFiles(client, logdir);
5         await addFiles(acq, client, files, errorCallback);
6     }
7 },

```

Zuletzt wird der Name des Moduls gesetzt.

```

1     name: 'Logs',
2 };

```

Weitere Module Zudem wurden folgende Module implementiert:

Name	Beschreibung	Konfiguration
Dumpsys	Aufruf des <code>dumpsys</code> -Kommandos auf dem Gerät und Archivieren seiner Ausgabe	keine
Device Properties	Aufruf des <code>getprop</code> -Kommandos auf dem Gerät und Archivieren seiner Ausgabe	keine
Logcat	Aufruf des <code>logcat</code> -Kommandos auf dem Gerät und Archivieren seiner Ausgabe	keine
Packages	Aufruf von <code>pm</code> mit entsprechenden Optionen; Parsen des Outputs, um Pakete und Metadaten zu extrahieren; Hashen der Programm-Archive und Speichern dieser Hashes; je nach Konfiguration Herunterladen und Archivieren der Programm-Archive; Generieren einer Liste aller Launcher-Activities und Speichern dieser	Auswahl, welche Programme Archive vom Gerät kopiert werden sollen
Processes	Aufruf des <code>ps</code> -Kommandos auf dem Gerät und Archivieren seiner Ausgabe	keine
Settings	Aufruf des <code>cmd settings list [NAMESPACE]</code> -Kommandos auf dem Gerät für die drei möglichen Namespaces und Archivieren der Ausgaben	keine

4.3.2 Datenspeicherung

Die Datenspeicherung findet mit Hilfe der `Acquisition`-Klasse statt. Diese bietet eine dünne Abstraktion über die Details der `zip.js`- und `age`-Integration.

Der Konstruktor der `Acquisition`-Klasse bekommt den `WritableStream` übergeben, in den das Archiv geschrieben werden soll. Falls das Archiv verschlüsselt werden soll, bekommt der Konstruktor zudem den dafür zu verwendenden öffentlichen `age`-Schlüssel übergeben, sonst `null`.

Basierend auf diesen Argumenten initialisiert der Konstruktor eine Instanz von `zip.js`' `ZipWriter`-Klasse, sowie ggf. `ages` `Encrypter`-Klasse für die Verschlüsselung.

```
1 constructor(output: WritableStream, ageRecipient: string | null) {
2   let writer: WritableStream;
3   if (ageRecipient !== null) {
4     const identityStream = new TransformStream();
5     const encryptor = new Encrypter();
6     encryptor.addRecipient(ageRecipient);
7     encryptor.encrypt(identityStream.readable).then(x =>
8       x.pipeTo(output));
9     writer = identityStream.writable;
10  } else {
11    writer = output;
12  }
13  this.zipWriter = new ZipWriter(writer);
}
```

Zudem stellt sie drei Funktionen bereit, um Daten aus einem `ReadableStream` oder einem String dem Archiv hinzuzufügen sowie die Datensammlung zu beenden und den `ZipWriter` zu schließen.

Neben der `Acquisition`-Klasse existiert noch eine Hilfsfunktion, um ein Shell-Kommando auf dem Gerät auszuführen und das Ergebnis in das Archiv zu speichern. Diese versucht, dem im Rahmen dieser Arbeit zu `wadb` hinzugefügten `shell, v2`-Support zu nutzen. Sollte dies nicht möglich sein¹⁸, fällt sie auf den `shell`-Service zurück.

¹⁸Siehe „[Verfügbarkeit des shell, v2-Services](#)“

4.3.3 UI

Zunächst wurde eine Reihe von Basis-Komponenten gestaltet, auf denen die Oberfläche der Anwendung basiert. Beispielsweise ein Container-Element mit maximaler Breite und Abstand; einfache Eingabe-Elemente wie Checkboxes, Dropdowns und Textfelder; ein Stepper-Element, der mehrere Dialog-Schritte nacheinander anzeigen kann sowie typografische Element wie Überschriften und Absätze.

Aus diesen wurden dann drei Dialoge konstruiert, die die Nutzer*in durch den Prozess der Archivgenerierung führen.

Dialog: Geräteauswahl Im ersten Dialog wird die Nutzer*in angeleitet, ADB zu aktivieren und AndroidQF-Web Zugriff auf das Gerät zu gestatten. Um ADB zu aktivieren, sind einige Schritte auf dem Smartphone nötig, die sich teils zwischen verschiedenen Android-Versionen und Herstellern unterscheiden. Da eine Abbildung aller mögliche Weg, ADB zu aktivieren, zu aufwändig wäre, wird lediglich auf den entsprechenden Eintrag in der offiziellen Android-Dokumentation verlinkt.

Zudem ist es nötig, der Website Zugriff auf das USB-Gerät zu gestatten. Dafür muss, nach einem Klick auf „Add New Device“, das Gerät in einem Browser-Popup ausgewählt und der Zugriff gestattet werden. Hierbei werden lediglich Geräte angezeigt, die ADB via USB bereitstellen.

Android QF

Welcome to AndroidQF. The following steps will guide you through everything necessary to acquire an archive of forensically relevant data from your android device.

Select Device

Please select the device from which data shall be gathered.

Make sure that you have enabled USB debugging. [You can find instructions on how to do so using this link.](#)

If you do not see your device in the list below, you might need to grant AndroidQF-Web permission to access it first. You can do so by clicking "Add New Device" and allowing access to your device in the browser popup.

Add New Device

Next Step

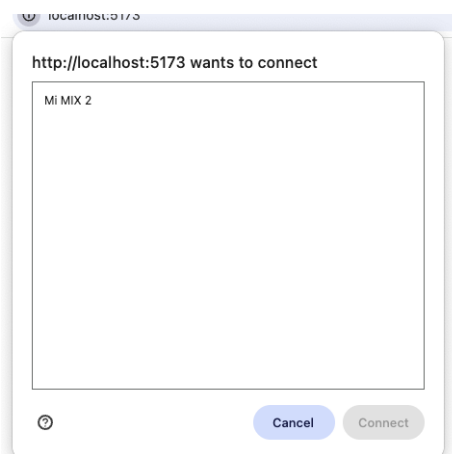


Abbildung 6: Startseite der Anwendung mit Geräteauswahl

Abbildung 7: Browserdialog, um den WebUSB-Zugriff zuzulassen

Nachdem der Zugriff auf das Gerät im Browser gestattet wurde, verbindet sich AndroidQF-Web als Client mit dem Android-Daemon auf dem Gerät. Hierfür

wird die `AdbClient`-Klasse aus `wadb` genutzt. Da bei allen modernen Android-Versionen diese Verbindung auf dem Gerät zugelassen werden muss, wird auch hierfür entsprechende Hilfestellung angezeigt. Die `AdbClient`-Instanz wird im globalen Anwendungszustand gespeichert. Anschließend kann die Nutzer*in durch den Button „*Next Step*“ zum nächsten Dialog übergehen.

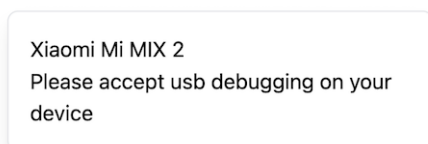


Abbildung 8: Aufforderung in AndroidQF-Web, USB-Debugging auf dem Gerät zuzulassen

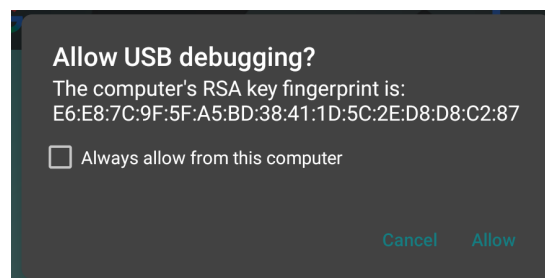


Abbildung 9: Dialog auf dem Gerät, um USB-Debugging zuzulassen

Dialog: Einstellungen Auf der nächsten Seite können Einstellungen zum erstellten Archiv getätigt werden. Momentan ist dies lediglich die Einstellung, ob und mit welchem Schlüssel verschlüsselt werden soll. Hierbei wird überprüft, ob der eingegebene Schlüssel valide¹⁹ ist. Hier wären bei einer Weiterentwicklung von AndroidQF-Web weitere Einstellungen möglich.

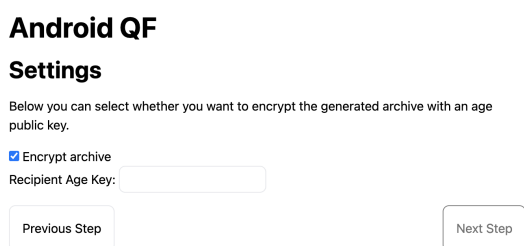


Abbildung 10: Seite "Settings" mit Eingabeoption für age-Schlüssel

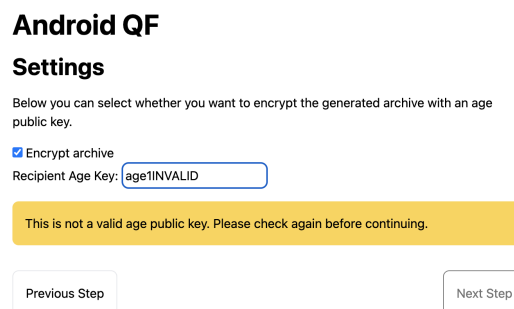


Abbildung 11: Fehlermeldung bei ungültigem age-Schlüssel

¹⁹Ein Schlüssel wird dann als valide angenommen, wenn er von der `X25519Recipient`-Klasse der `typeage`-Library akzeptiert wird^[101].

Dialog: Modulauswahl Auf der letzten Dialogseite kann die Nutzer*in auswählen, welche Daten gesammelt werden sollen. Dafür kann aus der Modulliste ausgewählt werden, welche Module ausgeführt werden sollen sowie deren Konfiguration angepasst werden.

Anschließend kann der Datensammlungsprozess durch den „Run“-Button gestartet werden. Hier muss die Nutzer*in zunächst eine lokale Datei auswählen, in die die Daten geschrieben werden sollen. Anschließend werden nacheinander alle Module ausgeführt und die Daten in das Archiv geschrieben und verschlüsselt.

Android QF
Select Modules

Below you can select which data should be collected. Once you selected the data you want to collect, press the "Run" button to start the data collection process. This may take some minutes and you need to confirm the collection on your phone.

- Dumpsys
- Environment
- Backup
- Logcat
- Device Properties
- Logs
- Packages
- Processes
- Settings

Run

Previous Step

Abbildung 12: Modulauswahl

Vor-Konfiguration über URL-Fragment Um eine Vorkonfiguration von AndroidQF-Web zu ermöglichen, liest die Software die Standard-Werte für alle möglichen Konfigurationsparameter aus dem Fragment-Teil der URL. Dieser Teil wird auch als „Hash“ bezeichnet, da er vom Rest der URL durch ein vorangestelltes #-Zeichen getrennt wird. Hierbei werden die Daten als **Schlüssel=Wert**-Paare enkodiert, welche vom &-Zeichen getrennt werden. Dieser Weg der Konfiguration wurde gewählt, da der Fragment-Teil der URL nicht an den Server übertragen wird und diese so keine Kenntnis über die Konfiguration erlangt[102].

Hierdurch kann der **age**-Schlüssel übergeben werden, an den verschlüsselt werden soll, die Liste der aktivierten Module sowie – soweit vorhanden – deren Konfiguration. Alle Werte können durch die Nutzer*in überschrieben werden.

4.4 Aufgetretene Probleme

4.4.1 Memory Limits

Web-Browser beschränken die Menge an Speicher, den eine Website allozieren kann. Je nach Objekt-Typ und Browser wird diese Beschränkung unterschiedlich berechnet. Beispielsweise berechnet Chrome die Beschränkung für `Blob`-Objekte abhängig vom insgesamt verfügbaren Arbeitsspeicher und der Systemarchitektur[103]. Für andere Objekte gibt es einen gemeinsamen Heap, auf dem bis zu einer gewissen Gesamt-Größe Objekte alloziert werden können[104].

Während der Entwicklung führte ein Erreichen dieses Limits nicht nur zu Fehlermeldungen und einem Stop der JavaScript-Ausführung, sondern auch zu länger anhaltenden Problemen, die teils erst durch einen Neustart des Browsers verschwanden. Beispielsweise konnten keine `WebAssembly`-Binaries mehr in den Browser geladen werden, dies führte stets zu einer Fehlermeldung.

Initial wurden die Daten pro Modul zunächst als JavaScript-Objekt, häufig als `Blob`, gesammelt und danach in das Zip-Archiv übertragen. Am Ende des Sammlungsprozesses wurde dieses Archiv ggf. verschlüsselt und zum Download angeboten. Gerade bei größeren Datenmengen (z. B. ein vollständiges Backup) wurden hier schnell die Beschränkungen des Browsers erreicht.

Daher wurde in einer zweiten Iteration auf die weitgehende Nutzung der `Streams`- und `File System Access`-API umgestellt.

4.4.2 Verfügbarkeit des `shell, v2`-Services

Einige der in [Android Debug Bridge](#) beschriebenen Protokolle sind nur auf Geräten mit einer ausreichend neuen Version des `adb`-Daemons verfügbar. Insbesondere der für den Zugriff auf die Shell des Geräts, und damit für die meisten Datenquellen, benötigte `shell, v2`-Service ist nur auf Geräten mit einer Android-Version ab 7.0 (API-Level 24) verfügbar[74].

Falls der `shell, v2`-Service nicht verfügbar ist, fällt der Code nun auf den älteren `shell`-Service zurück.

4.4.3 Unterschiedliche Android-Bedienoberflächen

Je nach Android-Version und Geräte-Hersteller unterscheiden sich die Bedienoberflächen stark. Dadurch ist es nicht möglich, eine Screenshot-Serie zu generieren, die das Aktivieren von ADB erklärt und für alle Nutzer*innen nachvollziehbar ist. Es wäre nötig, eine große Anzahl von Screenshot-Serien zu erstellen, was den Umfang dieser Arbeit sprengen würde. Daher wird auf die offizielle Android-Dokumentation verwiesen, die die nötigen Schritte allgemein erklärt.

5 Auswertung

Im Folgenden wird die erfolgte Umsetzung von AndroidQF-Web mit den formulierten Anforderungen sowie AndroidQF verglichen und es werden die Vor- und Nachteile gegenüber AndroidQF diskutiert.

5.1 Anforderungen

Dieser Abschnitt betrachtet die im Kapitel „[Anforderungen](#)“ formulierten Anforderungen erneut und bewertet jeweils, ob diese erfüllt wurden.

5.1.1 Anforderungen aus dem Angreifer*innen-Model

Stalkerware

1. Anforderung: „Die Software muss Daten sammeln, aus denen diese Art von Spyware erkannt werden kann. [...]“

Diese Anforderung wurde erfüllt, da alle nötigen Daten durch in AndroidQF-Web enthalte Module gesammelt werden:

Geforderte Daten	Gesammelt durch Modul
Liste der installierten Anwendungen	Packages – Generiert eine Liste aller installierten Pakete
Liste von Anwendungen, die eine oder mehrere der folgenden Berechtigungen haben[...]	Dumpsys – Ausgabe enthält eine Liste aller gewährten Berechtigungen
Liste von Anwendungen, die folgende Intents empfangen[...]	Dumpsys – Ausgabe enthält eine Liste Intent-Empfänger
Liste von Anwendungen, die keine Launcher-Activities definieren	Packages – Generiert eine Liste aller Pakete sowie aller Launcher-Activities, so können Pakete ohne Launcher-Activity herausgefiltert werden
Liste von Anwendungen, die als Device Administrator registriert sind	Dumpsys – Ausgabe enthält Liste der Anwendungen, die als Device Administrator registriert sind

2. Anforderung: „Die gesammelten Spuren müssen aus einer von der Spyware nicht manipulierbaren Quelle stammen. So genügt eine Auflistung der im App-Launcher sichtbaren Apps nicht, da Spyware sich dort verstecken kann[17]. Eine Auswertung von Daten der Systemdienste ist allerdings weiterhin möglich.“

Diese Anforderung ist erfüllt, da die Daten aus Systemdiensten gesammelt werden.

3. Anforderung: „Die Nutzung darf durch die Spyware nicht erkennbar sein, bzw. es muss eine andere plausible Erklärung für die von der Spyware erkennbaren Spuren geben (“Plausible Deniability”).“

Diese Anforderung ist nicht erfüllt. Für die meisten Nutzer*innen gibt es keine plausible Erklärung für das Aktivieren von ADB, dieses kann aber durch die Spyware erkannt werden[105].

Dieser Einschränkung unterliegt jede Software, die ADB zur Datensammlung nutzt, also auch AndroidQF und das MVT. Stattdessen müsste auf andere Datenquellen, wie auf anderen Wegen erstellte Bug-Report-Files[106] oder eine externe Analyse des Traffics, zurückgegriffen werden. Diese enthalten jedoch weniger Daten, als via ADB gesammelt werden können.

Das Risiko, dass die ADB-Nutzung erkannt wird, sollte daher vor der Nutzung solch einer Software bedacht werden. Gerade in Fällen, in denen direkte Gefahr, etwa durch Gewalt im Rahmen einer Beziehung oder repressive Staaten droht, kann es daher sinnvoll sein auf andere Datensammlungsansätze zu setzen.

State-Sponsored Malware / Mercenary Spyware

1. Anforderung: „Die Software sollte solche Daten sammeln, die sich in vergangenen Analysen als hilfreich erwiesen haben, namentlich die in [Datenquellen](#) genannten Daten. Mindestens jedoch muss sie alle dafür nötigen Funktionalitäten unterstützen und leicht um weitere Datenquellen erweitert werden können.“

Diese Anforderung ist teilweise erfüllt. AndroidQF-Web unterstützt einige, aber nicht alle Datenquellen, die von AndroidQF sowie MVT unterstützt werden.

Jedoch sind alle Funktionalitäten die zum Implementieren der Datenquellen zwingend nötig sind unterstützt. Lediglich die *Push*-Funktionalität ist nicht unterstützt, diese wird allerdings nur zum Übertragen des *Collectors* auf das Gerät benötigt. Eine Sammlung aller Daten ist jedoch auch ohne *Collector*

möglich. Die *Root*-Funktionalität wird von keinem Modul genutzt, ist jedoch in MVTs `check-adb` durch Ausführen eines Shell-Kommandos implementiert, was von AndroidQF-Web unterstützt wird.

Durch die modulare Struktur ist es zudem leicht möglich, weitere Datenquellen zu implementieren.

5.1.2 Nutzer*innen-Modell

1. Anforderung: „Die Nutzung muss ohne technisches Verständnis der Funktionsweise der Software möglich sein:[...]“

Diese Anforderung ist erfüllt, da alle Unterpunkte erfüllt sind:

1. „Dafür müssen für alle Schritte Anleitungen oder Erklärtexte bereitgestellt werden.“

Erfüllt, da, wie unter [UI](#) beschrieben, jeder Schritt mit Erklärtexten sowie teilweise mit Verweisen auf Anleitungen ausgestattet ist.

2. „Soweit innerhalb des Datensammlungsprozesses Einstellungen getätigt werden können, sollen Standardwerte für diese an die Software übergeben werden können, sodass eine Vor-Konfiguration möglich ist.“

Erfüllt, da eine [Vor-Konfiguration über das URL-Fragment](#) möglich ist.

3. „Die Nutzung muss mittels üblicher Software möglich sein. Dies sei definiert als Software, die auf der Mehrzahl der Geräte der Nutzer*innen bereits installiert ist.“

Erfüllt, da alle benötigten Standards in modernen Chrome-Browsern unterstützt werden. Ein solcher Browser ist mit „Edge“ auf den meisten Windows-Systemen vorinstalliert und für alle gängigen Desktop-Betriebssysteme verfügbar. Die Seite „Can I Use“ bietet eine Übersicht der von verschiedenen Browser-Versionen unterstützten Features[107]. Zum Zeitpunkt der Arbeit listet „Can I Use“ für die benötigten Features folgende Verbreitungsraten unter Desktop-Browsern („*tracked desktop*“). „Gesamt“ gibt die Verbreitungsrate der Überschneidung aller Standards an.

Standard	Verbreitungsrate
WebUSB	82.81 % [62]
Streams	88.03 % [108]

Standard	Verbreitungsrate
File System Access	80.63 % [70]
Gesamt	80.63 % [109]

2. Anforderung: „Um eine Übertragung weiterer Daten durch die Spyware zu erschweren, muss es möglich sein, die Daten vom Gerät zu sammeln, auch wenn Mobilfunk und WiFi deaktiviert sind oder sich das Gerät im *Flugmodus* befindet.“

Diese Anforderung ist erfüllt, da die Datensammlung über eine USB-Verbindung erfolgt. Eine Netzwerkverbindung des Geräts ist nicht erforderlich. Da die Datensammlung innerhalb von AndroidQF-Web lediglich durch client-seitiges JavaScript erfolgt, ist zudem nach dem Aufrufen des Webapp keine Internetverbindung des Geräts, auf dem die Daten gesammelt werden, mehr nötig.

5.1.3 Umgang mit den gesammelten Daten

Vertraulichkeit

1. Anforderung: „Die Daten sollten nur auf dem Gerät der Nutzer*in gespeichert werden. Eine Übertragung oder Verarbeitung auf anderen Geräte, insbesondere ggf. Server auf denen die Software gehostet wird, soll nicht stattfinden.“

Diese Anforderung wurde erfüllt. Die Sammlung, Verarbeitung und Speicherung der Daten findet lediglich auf den Geräten der Nutzer*in statt. Eine Übertragung der Daten auf andere Geräte findet nicht statt.

2. Anforderung: „Es muss die Möglichkeit bestehen, die Daten mittels eines asymmetrischen Verfahrens zu verschlüsseln.“

Diese Anforderung wurde erfüllt. Eine Verschlüsselung mit dem asymmetrischen *age*-Verfahren ist möglich.

Integrität

1. Anforderung: „Die Daten sollten mit einer Prüfsumme oder ähnlichen Verfahren vor einer unbeabsichtigten Veränderung der Daten, beispielsweise durch *bit rot*, geschützt werden.“

Diese Anforderung wurde erfüllt, da das `age`-Verfahren eine Prüfsumme für den Header generiert und das genutzte `ChaCha20-Poly1305`-Verfahren als AEAD-Verfahren ebenfalls eine Prüfsumme („`tag`“) für die generierten Ciphertext-Blöcke enthält.

2. Anforderung: „Es kann eine Prüfsumme der gesammelten Daten erstellt und der Nutzer*in angezeigt werden.“

Diese Anforderung wurde nicht erfüllt, wäre aber eine denkbare Erweiterung.

5.1.4 Verfügbarkeit

1. Anforderung: „Die Größe der gesammelten Daten sollte möglichst klein gehalten werden[...]“.

Diese Anforderung ist erfüllt, da ihre Unterpunkte erfüllt sind:

- Es kann eingestellt werden, welche Daten gesammelt werden.

Einzelne Module können deaktiviert werden, zudem kann für manche Module konfiguriert werden, welche Daten gesammelt werden.

- Daten werden komprimiert.

Hierbei kommen die Standard-Einstellungen von `zip.js` zum Einsatz, welche ein Kompressionslevel von 5 vorsehen[110].

5.2 Vergleich mit AndroidQF

AndroidQF diente nicht nur für den Namen, sondern auch bei weiteren Aspekten als Inspiration für AndroidQF-Web. AndroidQF-Web ist der Versuch einer Web-Version von AndroidQF unter den Benutzungserwartungen an Webapps. Daher ähnelt der Aufbau von AndroidQF-Web AndroidQF: Die Daten werden über ADB gesammelt, eine Reihe von Modulen übernimmt jeweils einen Teil der Datensammlung, Eine `Acquisition`-Klasse bündelt die Funktionalität zum Speichern der Daten und ebenso wie AndroidQF können die Daten als Zip-Archiv gespeichert und optional mit `age` verschlüsselt werden.

Die Logik der implementierten Module wurde größtenteils von AndroidQF übernommen, lediglich das `Packages`-Modul sammelt weitere Daten: Hier wird zusätzlich eine Liste der Launcher-Aktivitäten aller Apps generiert und im Datenarchiv gespeichert. So ist es möglich, Anwendungen auf dem Gerät zu erkennen, die nicht aus

dem System-Launcher gestartet werden können, eine häufig von Spyware genutzte Technik.

Im Gegensatz zu AndroidQF nutzt AndroidQF-Web kein *Collector*-Binary, welches auf das Gerät kopiert und ausgeführt wird. Stattdessen greift AndroidQF-Web auf die System-Tools zurück. Zudem wurden nicht alle in AndroidQF enthaltenen Module auch für AndroidQF-Web implementiert. Eine vollständige Implementation der in AndroidQF und MVT enthaltenen Module wäre jedoch möglich.

Der größte Unterschied zwischen AndroidQF und AndroidQF-Web ist die Bedienung: AndroidQF ist ein Kommandozeilen-Tool, welches die Installation des offiziellen ADB-Clients und dafür benötigter Treiber voraussetzt. Es ist nur möglich, entweder ein einzelnes oder alle Module auszuführen, das feingranulare (De-)Aktivieren einzelner Module ist nicht möglich. Bei zwei Modulen ist es möglich, genauer zu konfigurieren, welche Daten gesammelt werden (Backup, Packages).

AndroidQF-Web hingegen benötigt auf den meisten Systemen keine zusätzliche Software und kann direkt in einem Chromium-basiertem Browser ausgeführt werden. Eine Installation weiterer Treiber ist nicht nötig. AndroidQF-Web ermöglicht zudem eine feingranularere Konfiguration, welche Module ausgeführt werden. Die gleichen beiden Module wie bei AndroidQF erlauben darüber hinausgehend eine Konfiguration innerhalb des Moduls.

5.2.1 Vor- und Nachteile einer Nutzung

Aus der großen Ähnlichkeit von AndroidQF und AndroidQF-Web ergeben sich ähnliche Überlegungen, wann eine Nutzung der Tools nützlich sein kann. Dennoch gibt es auch Unterschiede, auf die nun eingegangen werden soll.

Vorteile Der größte Vorteil von AndroidQF-Web ist seine leichtere Bedienbarkeit, insbesondere für technisch weniger erfahrene Nutzer*innen. Die Nutzung von Web-Browsern und Web-Apps dürfte den meisten Nutzer*innen, im Gegensatz zur Nutzung von Kommandozeilentools, geläufig sein. Ebenso ist keine Treiber-Installation notwendig, was eine Nutzung abermals vereinfacht.

Durch die feingranulare Steuerung der Module können Spuren zielgerichtet und fallbezogen gesichert werden.

Zudem ist wäre eine Integration in andere Websites möglich (siehe „[Bessere Wartbarkeit und Integrierbarkeit](#)“).

Nachteile Dies stellt gleichzeitig einen Nachteil dar: Während das Herunterladen und Ausführen von Kommandozeilenprogrammen zumindest bei sicherheitsbewussten Nutzer*innen eine deutliche Hürde darstellt und zum Nachdenken anregt, ist dies bei der Website-Nutzung meist nicht der Fall.

Dennoch ist auch hier Vorsicht geboten, denn der Website muss vertraut werden, beispielsweise das tatsächlich der gewünschte Code ausgeliefert wird und die Daten nur auf den Geräten der Nutzer*in gespeichert und verarbeitet werden. Sicherzustellen, dass dies passiert, ist nur mit einer Analyse des ausgelieferten JavaScript-Codes möglich, welche jedoch gerade Nutzer*innen ohne großes technisches Wissen nicht trivial möglich ist. Ein möglicher Ansatz dafür wären Signaturen oder „*Reproducible Builds*“. Während Windows und macOS bereits großflächig Software-Signaturen einsetzen, ist die für Websites bisher unüblich.

Ein weiterer Nachteil der aktuellen Form von AndroidQF-Web ist, dass nicht alle Module unterstützt werden. Zudem werden die Daten aktuell nicht mittels des zuverlässigeren *Collectors* gesammelt, sondern es wird auf System-Tools zurückgegriffen.

Als Nachteil kann es auch betrachtet werden, dass die Nutzung von AndroidQF-Web Nutzer*innen daran gewöhnt, Websites via WebUSB Zugriff auf USB-Geräte zu geben. Dieser Zugriff bietet hohe Risiken, falls er missbräuchlich eingesetzt wird, wie bereits in der WebUSB-Spezifikation diskutiert wird[61].

6 Ausblick

Der im Rahmen dieser Arbeit entwickelte Prototyp zeigt, dass eine nutzer*innenfreundlichere Sammlung forensischer Daten von Android-Smartphones vom Browser aus möglich ist. Im Folgenden wird eine Reihe von Möglichkeiten vorgestellt, diesen Prototypen weiterzuentwickeln.

6.1 Bessere Wartbarkeit und Integrierbarkeit

Momentan sind der Code für die Datensicherung und der des User-Interfaces eng miteinander verwoben. Dies ermöglichte eine schnellere Entwicklung, erschwert jedoch beispielsweise eine mögliche Weiterentwicklung sowie Code-Reviews. Zudem können Teile des Codes so schwerer in andere Projekte integriert werden.

Die Komponenten zur Datensicherung könnten aus der Anwendung in eine eigenständige Bibliothek ausgegliedert werden. So könnten Entwickler*innen die Datensicherungsfähigkeiten nutzen, um andere Anwendungen zu entwickeln. Beispielsweise könnte eine analysierende Gruppe die Datensicherung in eine Website integrieren, die die gesammelten Daten gleich zur Analyse auf die Server der Gruppe hochlädt.

6.2 Weitere Datenquellen

Die Auswahl der Datenquellen bildet nur einen Teil der möglichen Quellen ab und orientiert sich stark am Entwicklungsstand von AndroidQF aus dem März 2023. Seitdem wurden weitere Datenquellen und der *Collector* hinzugefügt[111].

AndroidQF-Web könnte um diese Quellen erweitert werden. Zudem könnten die Datenquellen, die das MVT in seinem `check-adb`-Modul sammelt, AndroidQF allerdings nicht, ergänzt werden.

6.3 ADB mit root-Rechten ausführen

Je nach Systemversion kann der ADB-Daemon mit `root`-Rechten gestartet werden. Hierfür muss das *Android System Property* `ro.debuggable` auf `wahr` gesetzt sein, was es üblicherweise nur in Entwicklungsversionen von Android ist[112]. Ist dies der Fall, so startet sich der ADB-Daemon selbst mit `root`-Rechten neu, sobald der `root-Service` angesprochen wird. Bei Ansprechen des `unroot-Service`s startet

sich der ADB-Daemon ohne `root`-Rechte neu[75]. Dies erlaubt es, Kommandos als `root` auszuführen, auch wenn kein `su`-Binary auf dem Gerät vorhanden ist über das `root`-Rechte erlangt werden könnten.

6.4 Prüfsumme des generierten Archivs

Wie bereits im Abschnitt zu Integrität in „Anforderungen“ vorgeschlagen, könnte die Software eine Prüfsumme des generierten Archivs erstellen und der Nutzer*in anzeigen. Diese könnte auf einem sicheren Kanal unabhängig übertragen werden, auch wenn dies für das gesamte Archiv aufgrund seiner Größe vielleicht nicht möglich wäre. So könnte das Archiv auf einem weniger sicheren Weg übertragen werden und Manipulationen trotzdem erkannt werden.

6.5 Nutzer*innen-Freundlichkeit

Auch im Bereich der Nutzer*innen-Freundlichkeit liegen einige Verbesserungen nahe. Viele Schritte haben aktuell keine oder nur sehr spärliche Erklärungen, diese könnten ergänzt bzw. ausgebaut werden. Zudem sind diese Erklärungen, ebenso wie der Rest der Anwendung, auf Englisch gehalten. Hier könnte über eine Übersetzung in weitere Sprachen nachgedacht werden.

Manche der Module benötigen einige Zeit, um die Daten zu sammeln. Hier könnte – soweit möglich – eine Fortschrittsanzeige angezeigt werden, um der Nutzer*in zu signalisieren, dass weiterhin Daten gesammelt werden und ein Gefühl dafür zu geben, wie lange dies vermutlich ungefähr noch dauern wird.

Mittels Usability-Testing sowie basierend auf dem Feedback von Nutzer*innen könnten weitere Nutzungshürden identifiziert und durch Änderungen in der Software beseitigt werden.

6.6 Weitere Verschlüsselungsverfahren

Die von der Anwendung erstellten Archive können momentan nur mit dem `age`-Verschlüsselungssystem verschlüsselt werden. Hier wäre die Einführung von anderen Verfahren, insbesondere Post-Quanten-Verschlüsselungsverfahren eine Möglichkeit, möglicherweise zusätzlich zu einer Verschlüsselung mit `age`.

6.7 Andere Kompressionsverfahren

Aktuell nutzt die Anwendung Zip-Archive, hier könnte eine Umstellung auf platzsparende Verfahren wie *zstd* in Erwägung gezogen werden.

6.8 Reproducible Builds

Ein Softwarekompilationsprozess (*Build*) wird als *reproducible* bezeichnet, wenn bei gleichem Quellcode, gleicher Build-Umgebung und den gleichen Build-Instruktionen stets ein identisches Resultat erzeugt wird[113]. Dies erleichtert es zu verifizieren, dass die ausgelieferte Software tatsächlich dem gewünschten Quellcode entspricht, da ein unabhängiger *Build* des gleichen Quellcodes zu einem bit-identischen Ergebnis führen müsste. Dies könnte mit der ausgelieferten Version verglichen werden, bspw. durch Prüfsummen. Eine solche Verifikation wäre mit etwas Anleitung auch technisch unerfahrenen Nutzer*innen möglich.

6.9 Reduzierung der Anzahl der eingesetzten Bibliotheken

AndroidQF-Web setzt auf eine Reihe von externen Bibliotheken. Während einige davon für Kernfunktionalität benötigt werden und nur schwer ersetzt werden könnten, beispielsweise `wadb` oder `typeage`, ist dies für andere leichter möglich. Beispielsweise könnten die Frontend-Bibliotheken (React, Redux, Redux-Toolkit, Tailwind) durch kleinere Bibliotheken oder sogar JavaScript/TypeScript ohne weitere Abhängigkeiten ersetzt werden. Dies würde das Risiko von Supply-Chain-Attacks deutlich reduzieren.

Im Juli 2025 traf eine solche Attacke eine Reihe beliebter Pakete, von denen eins auch von AndroidQF-Web eingesetzt wird, jedoch in einer nicht kompromittierten Version[114].

Zudem enthalten diese Bibliotheken viel Code, der von AndroidQF nicht genutzt wird. Durch das Entfernen oder Ersetzen dieser Bibliotheken könnte daher sowohl die Größe der erzeugten Software als auch die Angriffsfläche reduziert werden.

6.10 Offline-Paketierung

Da AndroidQF-Web die Daten nur lokal verarbeitet und nach dem initialen Laden keine weitere Internetverbindung braucht, wäre es möglich, die gesamte Anwendung als kleine Anzahl von Dateien zu bündeln, die lokal im Web-Browser geöffnet

werden könnten. Diese Datei könnte auf einem bestehenden sicheren Weg zur Nutzer*in übertragen werden und nur lokal ausgeführt werden. Dies kann – gerade in Verbindung mit reproducible Builds – eine erhöhte Sicherheit herstellen, dass tatsächlich (nur) die gewünschte Software ausgeführt wird. Zudem wäre so eine Ausführung auch auf air-gapped Geräten einfach möglich.

Literaturverzeichnis

- [1] Forbidden Stories, „About the Pegasus Project“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://forbiddenstories.org/about-the-pegasus-project/>
- [2] J. Scott-Railton u. a., „CatalanGate: Extensive Mercenary Spyware Operation against Catalans Using Pegasus and Candiru“, Apr. 2022, Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <http://hdl.handle.net/1807/119418>
- [3] „Predator Files: Technical Deep-Dive into Intellexa Alliance’s Surveillance Products“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://securitylab.amnesty.org/latest/2023/10/technical-deep-dive-into-intellexa-alliance-surveillance-products/>
- [4] R. Chatterjee u. a., „The Spyware Used in Intimate Partner Violence“, in *2018 IEEE Symposium on Security and Privacy (SP)*, Mai 2018, S. 441–458. doi: [10.1109/SP.2018.00061](https://doi.org/10.1109/SP.2018.00061).
- [5] Koalition gegen Stalkerware, „Informationen für Medien | Coalition Against Stalkerware (DE)“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://stopstalkerware.org/de/informationen-fur-medien/>
- [6] „The State of Stalkerware in 2023“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2024/03/07160820/The-State-of-Stalkerware-in-2023.pdf>
- [7] L. Franceschi-Bicchierai, „Hacked, Leaked, Exposed: Why You Should Never Use Stalkerware Apps“. Zugegriffen: 4. Juli 2025. [Online]. Verfügbar unter: <https://techcrunch.com/2025/07/02/hacked-leaked-exposed-why-you-should-stop-using-stalkerware-apps/>
- [8] „Mvt-Project/Androidqf“. MVT. Zugegriffen: 8. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/androidqf>
- [9] „Mobile Verification Toolkit“. Zugegriffen: 9. Juli 2025. [Online]. Verfügbar unter: <https://docs.mvt.re/en/latest/>
- [10] „§ 101 StPO - Einzelnorm“. Zugegriffen: 4. Juli 2025. [Online]. Verfügbar unter: https://www.gesetze-im-internet.de/stpo/___101.html
- [11] S. Kirchgaessner u. a., „Revealed: Leak Uncovers Global Abuse of Cyber-Surveillance Weapon“, *The Guardian: World news*, 18. Juli 2021. Zugegriffen: 4. Juli 2025. [Online]. Verfügbar unter: <https://www.theguardian.com/world/2021/jul/18/revealed-leak-uncovers-global-abuse-of-cyber-surveillance-weapon-nso-group-pegasus>
- [12] „About Apple Threat Notifications and Protecting against Mercenary Spyware“. Zugegriffen: 5. Juli 2025. [Online]. Verfügbar unter: <https://support.apple.com/en-us/102174>

- [13] L. Franceschi-Bicchierai, „WhatsApp Says It Disrupted a Hacking Campaign Targeting Journalists with Paragon Spyware“. Zugegriffen: 5. Juli 2025. [Online]. Verfügbar unter: <https://techcrunch.com/2025/01/31/whatsapp-says-it-disrupted-a-hacking-campaign-targeting-journalists-with-spyware/>
- [14] „Android Forensic Methodology - Mobile Verification Toolkit“. Zugegriffen: 9. Juli 2025. [Online]. Verfügbar unter: <https://docs.mvt.re/en/latest/android/methodology/>
- [15] „Forensic Methodology Report: How to Catch NSO Group’s Pegasus“. Zugegriffen: 20. Juli 2025. [Online]. Verfügbar unter: <https://www.amnesty.org/en/latest/research/2021/07/forensic-methodology-report-how-to-catch-nso-groups-pegasus/>
- [16] „SpyGuard“. 17. Juli 2025. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://github.com/SpyGuard/SpyGuard>
- [17] E. Liu *u. a.*, „No Privacy Among Spies: Assessing the Functionality and Insecurity of Consumer Android Spyware Apps“, *PoPETs*, Bd. 2023, Nr. 1, S. 207–224, Jan. 2023, doi: [10.56553/popets-2023-0013](https://doi.org/10.56553/popets-2023-0013).
- [18] S. Lee, „Quickstart“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://open5gs.org/open5gs/docs/guide/01-quickstart/>
- [19] „Daten auf einem Android-Gerät sichern oder wiederherstellen - Android-Hilfe“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://support.google.com/android/answer/2819582?hl=de>
- [20] „<application> | App Architecture“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/guide/topics/manifest/application-element>
- [21] „iOS Forensic Methodology - Mobile Verification Toolkit“. Zugegriffen: 5. Juli 2025. [Online]. Verfügbar unter: <https://docs.mvt.re/en/latest/ios/methodology/>
- [22] „Behavior Changes: Apps Targeting Android 12“. Zugegriffen: 9. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/about/versions/12/behavior-changes-12>
- [23] „Android Debug Bridge (Adb) | Android Studio“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/tools/adb>
- [24] „Mvt-Project/Mvt“. MVT. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/mvt>
- [25] N. C. S. Centrum, „Factsheet Indicators of Compromise - Factsheet - National Cyber Security Centre“. Zugegriffen: 4. Juli 2025. [Online]. Verfügbar unter: <https://english.ncsc.nl/publications/factsheets/2019/juni/01/factsheet-indicators-of-compromise>

- [26] „Investigations/2021-07-18_nso at Master · AmnestyTech/Investigations“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: https://github.com/AmnestyTech/investigations/tree/master/2021-07-18_nso
- [27] „AssoEchap/Stalkerware-Indicators“. Echap, 8. Juli 2025. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://github.com/AssoEchap/stalkerware-indicators>
- [28] „STIX Version 2.1“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html>
- [29] „Mvt/README.Md at 1df61b5bbf54cb44d6ff156da229924deb836b0b · Mvt-Project/Mvt“. Zugegriffen: 9. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/mvt/blob/1df61b5bbf54cb44d6ff156da229924deb836b0b/README.md>
- [30] „Mvt/Src/Mvt/Android at 595a2f6536d9fc1dff85605995631eff4146994 · Mvt-Project/Mvt“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/mvt/tree/595a2f6536d9fc1dff85605995631eff4146994/src/mvt/android>
- [31] „Mvt/Src/Mvt/Android/Modules at 1df61b5bbf54cb44d6ff156da229924deb836b0b · Mvt-Project/Mvt“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/mvt/tree/1df61b5bbf54cb44d6ff156da229924deb836b0b/src/mvt/android/modules>
- [32] „Androidqf/Modules at 25e1999bc38ce65b65f56961956448f8697d37d1 · Mvt-Project/Androidqf“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/androidqf/tree/25e1999bc38ce65b65f56961956448f8697d37d1/modules>
- [33] „Capture and Read Bug Reports | Android Studio“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/studio/debug/bug-report>
- [34] „Dumpsys | Android Studio“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: <https://developer.android.com/tools/dumpsys>
- [35] „Androidqf/README.Md at 25e1999bc38ce65b65f56961956448f8697d37d1 · Mvt-Project/Androidqf“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/androidqf/blob/25e1999bc38ce65b65f56961956448f8697d37d1/README.md>
- [36] „Add System Properties | Android Open Source Project“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: <https://source.android.com/docs/core/architecture/configuration/add-system-properties>
- [37] „Logcat Command-Line Tool | Android Studio“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: <https://developer.android.com/tools/logcat>
- [38] „Settings | API Reference“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: <https://developer.android.com/reference/android/provider/Settings>

- [39] „Mvt/Src/Mvt/Android/Modules/Adb/Sms.Py at 1df61b5bbf54cb44d6ff156da229924deb836b0b · Mvt-Project/Mvt“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/mvt/blob/1df61b5bbf54cb44d6ff156da229924deb836b0b/src/mvt/android/modules/adb/sms.py>
- [40] Kaspersky, „The State of Stalkerware in 2022“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://securelist.com/the-state-of-stalkerware-in-2022/108985/>
- [41] D. Freed, J. Palmer, D. Minchala, K. Levy, T. Ristenpart, und N. Dell, „A Stalker’s Paradise: How Intimate Partner Abusers Exploit Technology“, in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, in CHI ’18. New York, NY, USA: Association for Computing Machinery, Apr. 2018, S. 1–13. doi: [10.1145/3173574.3174241](https://doi.org/10.1145/3173574.3174241).
- [42] Z. Shan, R. Samuel, und I. Neamtiu, „Device Administrator Use and Abuse in Android: Detection and Characterization“, gehalten auf der 25th Annual International Conference on Mobile Computing and Networking, MobiCom 2019, 2019. doi: [10.1145/3300061.3345452](https://doi.org/10.1145/3300061.3345452).
- [43] „Intent | API Reference | Android Developers“. Zugegriffen: 10. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/reference/android/content/Intent>
- [44] „Create Your Own Accessibility Service“. Zugegriffen: 10. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/guide/topics/ui/accessibility/service>
- [45] „Telephony.Sms.Intents | API Reference“. Zugegriffen: 10. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/reference/android/provider/Telephony.Sms.Intents>
- [46] „Device Administration Overview | Android Enterprise“. Zugegriffen: 10. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/work/device-admin>
- [47] A. Meister, „Kritik vom Bundesrechnungshof: Das Bundeskriminalamt will gleich zwei Staatstrojaner einsetzen“, *netzpolitik.org*, 15. August 2016. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://netzpolitik.org/2016/kritik-vom-bundesrechnungshof-das-bundeskriminalamt-will-gleich-zwei-staatstrojaner-einsetzen/>
- [48] F. Flade, G. Mascolo, und R. Pinkert, „Umstrittene Spionage-Software: BKA soll Seehofer nicht informiert haben“, *tagesschau.de*, 7. September 2021. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://web.archive.org/web/20210907105909/https://www.tagesschau.de/investigativ/ndr-wdr/spionagesoftware-nso-bka-103.html>

- [49] K. Biermann und H. Stark, „Überwachungsaffäre: Die Superwaffe und die Deutschen“, *Die Zeit*, Hamburg, 19. Juli 2021. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://www.zeit.de/politik/ausland/2021-07/ueberwachungsaffaere-spionage-software-pegasus-einsatz-deutschland-bundeskriminalamt-handydaten-rechtsstaat/komplettansicht>
- [50] B. Larin, „Operation Triangulation: The Last (Hardware) Mystery“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://securelist.com/operation-triangulation-the-last-hardware-mystery/111669/>
- [51] B. Marczak *u. a.*, „From Pearl to Pegasus: Bahraini Government Hacks Activists with NSO Group Zero-Click iPhone Exploits“, University of Toronto, Citizen Lab Research Report No. 141, Aug. 2021. Zugegriffen: 11. Juli 2025. [Online]. Verfügbar unter: <https://citizenlab.ca/2021/08/bahrain-hacks-activists-with-nso-group-zero-click-iphone-exploits/>
- [52] „NSO Pegasus - DocumentCloud“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://embed.documentcloud.org/documents/4599753-NSO-Pegasus/>
- [53] B. Marczak und J. Scott-Railton, „The Million Dollar Dissident: NSO Group’s iPhone Zero-Days Used against a UAE Human Rights Defender“, University of Toronto, Citizen Lab Research Report No. 78, Aug. 2016. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://citizenlab.ca/2016/08/million-dollar-dissident-iphone-zero-day-nso-group-uae/>
- [54] „NSO Group / Q Cyber Technologies: Over One Hundred New Abuse Cases“, Citizen Lab, University of Toronto, Okt. 2019. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://citizenlab.ca/2019/10/nso-q-cyber-technologies-100-new-abuse-cases/>
- [55] B. Marczak, J. Scott-Railton, N. Aljizawi, S. Anstis, und R. Deibert, „The Great iPwn: Journalists Hacked with Suspected NSO Group iMessage ‘Zero-Click’ Exploit“, University of Toronto, Citizen Lab Research Report No. 135, Dez. 2020. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://citizenlab.ca/2020/12/the-great-ipwn-journalists-hacked-with-suspected-nso-group-imessage-zero-click-exploit/>
- [56] B. Marczak *u. a.*, „FORCEDENTRY: NSO Group iMessage Zero-Click Exploit Captured in the Wild“, University of Toronto, Citizen Lab Research Report No. 143, Sep. 2021. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://citizenlab.ca/2021/09/forcedentry-nso-group-imessage-zero-click-exploit-captured-in-the-wild/>

- [57] B. Marczak, J. Scott-Railton, B. A. Razzak, und R. Deibert, „Triple Threat: NSO Group’s Pegasus Spyware Returns in 2022 with a Trio of iOS 15 and iOS 16 Zero-Click Exploit Chains“, Citizen Lab, University of Toronto, Apr. 2023. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://citizenlab.ca/2023/04/nso-groups-pegasus-spyware-returns-in-2022/>
- [58] D. Freed, J. Palmer, D. E. Minchala, K. Levy, T. Ristenpart, und N. Dell, „Digital Technologies and Intimate Partner Violence: A Qualitative Analysis with Multiple Stakeholders“, *Proc. ACM Hum.-Comput. Interact.*, Bd. 1, S. 1–22, Dez. 2017, doi: [10.1145/3134681](https://doi.org/10.1145/3134681).
- [59] J. Cawthra, M. Ekstrom, L. Lusty, J. Sexton, und J. Sweetnam, „Data Integrity: Detecting and Responding to Ransomware and Other Destructive Events“, National Institute of Standards and Technology, Gaithersburg, MD, Dez. 2020. doi: [10.6028/nist.sp.1800-26](https://doi.org/10.6028/nist.sp.1800-26).
- [60] *DIN EN ISO/IEC 27000:2020-06, Informationstechnik Sicherheitsverfahren Informationssicherheitsmanagementsysteme – Überblick Und Terminologie (ISO/IEC 27000:2018); Deutsche Fassung EN ISO/IEC 27000:2020*. Berlin: DIN Media GmbH. doi: [10.31030/3144079](https://doi.org/10.31030/3144079).
- [61] „WebUSB API“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://wicg.github.io/webusb/>
- [62] „WebUSB | Can I Use... Support Tables for HTML5, CSS3, Etc“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://caniuse.com/webusb>
- [63] „HTML Standard“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://html.spec.whatwg.org/multipage/interaction.html#transient-activation>
- [64] „UserGestureIndicator.Cpp - Chromium Code Search“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: https://source.chromium.org/chromium/chromium/src/+/_main:third_party/WebKit/Source/core/dom/UserGestureIndicator.cpp;l=29;drc=9171a2ab9637d50fb574ebc354a859776efa9dd0
- [65] „HTML Standard“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://html.spec.whatwg.org/multipage/interaction.html#activation-triggering-input-event>
- [66] „HTML Standard“. Zugegriffen: 8. Juli 2025. [Online]. Verfügbar unter: <https://html.spec.whatwg.org/multipage/webappapis.html#secure-context>
- [67] „Streams Standard“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://streams.spec.whatwg.org/>
- [68] „File System Access“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://wicg.github.io/file-system-access/>
- [69] „File System Standard“. Zugegriffen: 8. Juli 2025. [Online]. Verfügbar unter: <https://fs.spec.whatwg.org/>

- [70] „File System Access API | Can I Use... Support Tables for HTML5, CSS3, Etc“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://caniuse.com/native-file-system-api>
- [71] „Adb/OVERVIEW.TXT - Platform/System/Core - Git at Google“. Zugegriffen: 5. Juli 2025. [Online]. Verfügbar unter: https://android.googlesource.com/platform/system/core/+refs/tags/android-11.0.0_r20/adb/OVERVIEW.TXT
- [72] „Architecture of ADB Wifi“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: https://android.googlesource.com/platform/packages/modules/adb/+65cd52911acd865e41886d35c3bf1d1f4f591c82/docs/dev/adb_wifi.md
- [73] „Docs/Dev/Protocol.Md“. Zugegriffen: 5. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/packages/modules/adb/+65cd52911acd865e41886d35c3bf1d1f4f591c82/docs/dev/protocol.md>
- [74] „Docs/Dev/Services.Md“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/packages/modules/adb/+65cd52911acd865e41886d35c3bf1d1f4f591c82/docs/dev/services.md>
- [75] „Daemon/Services.Cpp - Platform/Packages/Modules/Adb - Git at Google“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/packages/modules/adb/+a9b3987d2a42a40de0d67fceb50c9716639ef03/daemon/services.cpp>
- [76] „Cmds/Bu - Platform/Frameworks/Base - Git at Google“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/frameworks/base/+325fb3e2aac104d7090dd4978356920875c61ee0/cmds/bu>
- [77] „Cmds/Bu/Src/Com/Android/Commands/Bu/Backup.Java - Platform/Frameworks/Base - Git at Google“. Zugegriffen: 13. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/frameworks/base/+a2fe6c5bb914bb3374a13d7c718614dd98698b71/cmds/bu/src/com/android/commands/bu/Backup.java>
- [78] „Docs/Dev/Sync.Md“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/packages/modules/adb/+65cd52911acd865e41886d35c3bf1d1f4f591c82/docs/dev/sync.md>
- [79] „Daemon/File_sync_service.Cpp - Platform/Packages/Modules/Adb - Git at Google“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: https://android.googlesource.com/platform/packages/modules/adb/+a10779a5cc1ad1b9821989bf731ebd0afbb28458/daemon/file_sync_service.cpp

- [80] „GoogleChromeLabs/Wadb: A TypeScript Implementation of the Android Debug Bridge(ADB) Protocol over WebUSB“. GoogleChromeLabs, 13. Juli 2025. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://github.com/GoogleChromeLabs/wadb>
- [81] „Wadb/Src/Lib/ShellV2.Ts at 656a2f89358da4738543709e1d952bb20b898966 · Pajowu/Wadb“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://github.com/pajowu/wadb/blob/656a2f89358da4738543709e1d952bb20b898966/src/lib/ShellV2.ts>
- [82] pajowu, „Switch Pull to Use Streams, Implement Backup & Shellv2 by Pajowu · Pull Request #49 · GoogleChromeLabs/Wadb“. Zugegriffen: 22. Juli 2025. [Online]. Verfügbar unter: <https://github.com/GoogleChromeLabs/wadb/pull/49>
- [83] „Zip.Js - JavaScript Library to Zip and Unzip Files“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://gildas-lormeau.github.io/zip.js/>
- [84] „Release Age v1.0.0 · FiloSottile/Age“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://github.com/FiloSottile/age/releases/tag/v1.0.0>
- [85] Y. Nir und A. Langley, „ChaCha20 and Poly1305 for IETF Protocols“, Internet Engineering Task Force, Request for Comments RFC 7539, Mai 2015. doi: [10.17487/RFC7539](https://doi.org/10.17487/RFC7539).
- [86] „FiloSottile/Typage: A TypeScript Implementation of the Age File Encryption Format, Available as an Npm Package or as a Bundled .Js File.“ Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/FiloSottile/typage>
- [87] „Typage/README.Md at 31e8d9aee8ca5fc1d12d1121abdc120ea3dcc43a · FiloSottile/Typage“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/FiloSottile/typage/blob/31e8d9aee8ca5fc1d12d1121abdc120ea3dcc43a/README.md>
- [88] P. Miller, „Noble Cryptography“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://paulmillr.com/noble/>
- [89] „Release v0.2.4 with Streaming APIs · FiloSottile/Typage“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://github.com/FiloSottile/typage/releases/tag/v0.2.4>
- [90] „React/LICENSE at Dffacc7b8094576c19790fe8341996f743ba4a89 · Facebook/React“. Zugegriffen: 19. Juli 2025. [Online]. Verfügbar unter: <https://github.com/facebook/react/blob/dffacc7b8094576c19790fe8341996f743ba4a89/LICENSE>
- [91] „Tailwindcss/LICENSE at 939fda6f4e101ff3d378e0d51c20a8baa229aff · Tailwindlabs/Tailwindcss“. Zugegriffen: 19. Juli 2025. [Online]. Verfügbar unter: <https://github.com/tailwindlabs/tailwindcss/blob/939fda6f4e101ff3d378e0d51c20a8baa229aff/LICENSE>

- [92] „Redux/LICENSE.Md at 9a3692deb2cfa35708f5bf81791f4d723534a4bb · Reduxjs/Redux“. Zugegriffen: 19. Juli 2025. [Online]. Verfügbar unter: <https://github.com/reduxjs/redux/blob/9a3692deb2cfa35708f5bf81791f4d723534a4bb/LICENSE.md>
- [93] „Redux-Toolkit/LICENSE at Af3e75bb9e6a51e13603d2a5684ef3ba1da07e3b · Reduxjs/Redux-Toolkit“. Zugegriffen: 19. Juli 2025. [Online]. Verfügbar unter: <https://github.com/reduxjs/redux-toolkit/blob/af3e75bb9e6a51e13603d2a5684ef3ba1da07e3b/LICENSE>
- [94] „React“. Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://react.dev/>
- [95] „Tailwind CSS - Rapidly Build Modern Websites without Ever Leaving Your HTML.“ Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://tailwindcss.com/>
- [96] „Redux - A JS Library for Predictable and Maintainable Global State Management | Redux“. Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://redux.js.org/>
- [97] „Redux Toolkit | Redux Toolkit“. Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://redux-toolkit.js.org/>
- [98] „Redux Essentials, Part 3: Basic Redux Data Flow | Redux“. Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://redux.js.org/tutorials/essentials/part-3-data-flow>
- [99] „JavaScript With Syntax For Types.“ Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://www.typescriptlang.org/>
- [100] „Androidqf/Modules/Logs.Go at Facb7a02e30e81dcf5f35d3597c8403b93b47418 · Mvt-Project/Androidqf“. Zugegriffen: 16. Juli 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/androidqf/blob/facb7a02e30e81dcf5f35d3597c8403b93b47418/modules/logs.go>
- [101] „Typage/Lib/Recipients.Ts at 440d132ce052600f307a7e00a0d7d663cb75937d · FiloSottile/Typage“. Zugegriffen: 19. Juli 2025. [Online]. Verfügbar unter: <https://github.com/FiloSottile/typage/blob/440d132ce052600f307a7e00a0d7d663cb75937d/lib/recipients.ts>
- [102] T. Berners-Lee, R. T. Fielding, und L. M. Masinter, „Uniform Resource Identifier (URI): Generic Syntax“, Internet Engineering Task Force, Request for Comments RFC 3986, Jan. 2005. doi: [10.17487/RFC3986](https://doi.org/10.17487/RFC3986).
- [103] „Blob_memory_controller.Cc - Chromium Code Search“. Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: https://source.chromium.org/chromium/chromium/src/+/main:storage/browser/blob/blob_memory_controller.cc;l=79;drc=70971f14e03d68888137fd54da8f8b82c9857ed1

- [104] „Heap.Cc - Chromium Code Search“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: [https://source.chromium.org/chromium/chromium/src/+main:v8/src/heap/heap.cc;drc=aaba5e93c0f99976ccb15d4e0b18ea8a9da96571](https://source.chromium.org/chromium/chromium/src/+/main:v8/src/heap/heap.cc;drc=aaba5e93c0f99976ccb15d4e0b18ea8a9da96571)
- [105] „Settings.Global | API Reference“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://developer.android.com/reference/android/provider/Settings.Global>
- [106] „Read Bug Reports“. Zugegriffen: 14. Juli 2025. [Online]. Verfügbar unter: <https://source.android.com/docs/core/tests/debug/read-bug-reports>
- [107] „Can I Use... Support Tables for HTML5, CSS3, Etc“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://caniuse.com/ciu/about>
- [108] „Streams | Can I Use... Support Tables for HTML5, CSS3, Etc“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://caniuse.com/streams>
- [109] „WebUSB | Can I Use... Support Tables for HTML5, CSS3, Etc“. Zugegriffen: 20. Juli 2025. [Online]. Verfügbar unter: <https://caniuse.com/webusb,native-file-system-api,streams>
- [110] „ZipWriterConstructorOptions | @zip.js/Zip.js“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://gildas-lormeau.github.io/zip.js/api/interfaces/ZipWriterConstructorOptions.html>
- [111] „Adds Android-Collector Code · Mvt-Project/Androidqf@83d529a“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: <https://github.com/mvt-project/androidqf/commit/83d529aac81c627173fef64a402bd499e155eff2>
- [112] „Core/Java/Android/Os/Build.Java - Platform/Frameworks/Base - Git at Google“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://android.googlesource.com/platform/frameworks/base/+fe011e06b5f1caf35e87c43ce5306234914d5c8c/core/java/android/os/Build.java>
- [113] „Definitions - reproducible-builds.org“. Zugegriffen: 17. Juli 2025. [Online]. Verfügbar unter: <https://reproducible-builds.org/docs/definition/>
- [114] „Active Supply Chain Attack: Npm Phishing Campaign Leads to P...“ Zugegriffen: 23. Juli 2025. [Online]. Verfügbar unter: <https://socket.dev/blog/npm-phishing-campaign-leads-to-prettier-tooling-packages-compromise>

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 24.07.2025

.....