Zentrale Stelle
für Informationstechnik
im Sicherheitsbereich

HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Jannik Weber

Analyse der Datenerhebung von Werbenetzwerken in Android Apps

2019

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Analyse der Datenerhebung von Werbenetzwerken in Android Apps

Autor:
Jannik Weber

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO16w5-B

Erstprüfer:
Prof. Dr. Dirk Pawlaszczyk

Zweitprüfer:
Dr. Ralf Zimmermann

Mittweida, August 2019

Bibliografische Angaben

Weber, Jannik: Analyse der Datenerhebung von Werbenetzwerken in Android Apps, 71 Seiten, 18 Abbildungen, 14 Tabellen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit, 2019

Satz: LaTeX

Referat

In dieser Arbeit werden ausgewählte Werbenetzwerke in Android-Apps im Hinblick auf deren Datenerhebung betrachtet. Dabei stellen sich folgende Forschungsfragen: Welche Daten werden an die Werbenetzwerke übermittelt? Wie decken sich die übermittelten Daten mit den Angaben seitens der Werbenetzwerke? Wie werden die Daten von dem Android-Gerät erhoben und welchen Einfluss haben die Daten auf die Privatsphäre des Benutzers?

Um die Forschungsfragen zu beantworten, ist eine Auswahl von 100 Apps aus dem *Google Play Store* getroffen und auf deren enthaltenen Werbenetzwerke analysiert worden. Anschließend wurde der Netzwerkverkehr von ausgewählten Werbenetzwerken mithilfe eines Proxy-Server betrachtet, um die übermittelten Daten abzufangen. Gleichzeitig sind die Datenschutzerklärungen der ausgewählten Werbenetzwerke im Hinblick auf deren Angaben zu erhobenen Daten betrachtet worden. Abschließend wurde der Quellcode von den Werbenetzwerken angeschaut, um herauszufinden wie die Daten aus dem Android-System erhoben werden.

Die Ergebnisse der Analyse des Datenverkehrs zeigen, dass neben Daten über das verwendete Gerät, die ausführende App, zur ungefähren Positionsbestimmung und zu Einstellungsparameter auch Statusinformationen, wie Ladezustand der Batterie und freier Speicherplatz, übertragen werden. Es wurde festgestellt, dass Werbenetzwerke erheblich mehr Daten erheben und übermitteln als in deren Datenschutzerklärungen angegeben. Durch die Analyse des Quellcodes der Werbenetzwerke konnte herausgefunden werden, dass Daten zum einen über API-Aufrufe an das Android-System ermittelt werden und zum anderen durch entsprechende Algorithmen der Werbenetzwerke.

Mithilfe der Ergebnisse konnte gezeigt werden, dass seitens der Werbenetzwerke eine bessere und umfangreichere Angabe zu den erhobenen Daten wünschenswert ist. Weiterhin konnten unter den Daten keine personenbezogenen Informationen gefunden werden, welche eine Zuordnung zu einem einzelnen Individuum erlauben. Jedoch wurden Informationen gefunden, die eine Zuordnung der Daten zu einem einzelnen Gerät erlauben.

Schlüsselwörter: Android, API-Aufrufe, Burp-Suite, Datenerhebung, Datenübermittlung, Proxy-Server, SDK, Softwarebibliothek, Werbenetzwerk

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

ADB ..... Android Debug Bridge, Seite 23
API ..... Application Programming Interface, Seite 7
APK ..... Android Package Kit, Seite 8
ART ..... Android Runtime, Seite 5
CPA ..... Cost Per Action, Seite 16
CPC ..... Cost Per Click, Seite 15
CPI ..... Cost Per Install, Seite 16
CPM ..... Cost Per Mile, Seite 15
CPU ..... Central Processing Unit, Seite 44
CPV ..... Cost Per View, Seite 16
DEX ..... Dalvik Executeable Format, Seite 8
GPS ..... Global Positioning System, Seite 5
HAL ..... Hardware Abstraction Layer, Seite 7
HTTP ..... Hypertext Transfer Protocol, Seite 17
HTTPS ..... Hypertext Transfer Protocol Secure, Seite 17
IMEI ..... International Mobile Station Equipment Identity, Seite 49
IP ..... Internet Protocol, Seite 29
JSON ..... JavaScript Object Notation, Seite 37
LTE ..... Long Term Evolution, Seite 17
MMS ..... Multimedia Messaging Service, Seite 10
SDK ..... Software Developement Kit, Seite 1
SMS ..... Short Message Service, Seite 7
SSL ..... Secure Sockets Layer, Seite 19
UID ..... User ID, Seite 9
URL ..... Uniform Resource Locator, Seite 18
WLAN ..... Wireless Local Area Network, Seite 17

1 Einleitung

Heutzutage besitzt nahezu jede Person ein Smartphone. Dabei werden aktuell 4 von 5 Smartphones mit dem Betriebssystem Android verkauft [22]. Android ist das weltweit am weitesten verbreitete mobile Betriebssystem, mit einem Marktanteil von über 80% [22]. Geräte mit diesem Betriebssystem lassen sich um eine Vielzahl an Drittanbieter-Apps ergänzen, die eine Menge an zusätzlichen Funktionen bereitstellen. Dafür stellt Google den offiziellen Markt *Google Play Store* zur Verfügung. Von Spielen über Tools, wie QR-Scanner und Thermometer bis hin zu Dating-Apps, ist dort eine Breite Masse an zusätzlichen Anwendungen für mobile Geräte zu finden.

Über diese Plattform haben Entwickler die Möglichkeit auf einfache Weise ihre Anwendungen einer großen Masse an potentiellen Kunden bereitzustellen. Dabei sind über 95% der Anwendungen im *Google Play Store* kostenlos [10]. Dies hat den Grund, dass es heutzutage für Entwickler profitabler ist ihre Anwendungen durch Werbung zu monetarisieren. Dadurch können sie Gewinn erwirtschaften ohne für die Anwendungen selbst Gebühren zu verlangen. Möglich machen dies personalisierte Werbeanzeigen, die auf Interessen oder geografischen Informationen beruhen. Dabei machen sich die meisten Benutzer von kostenlosen Anwendungen keine Gedanken darüber, was die Anwendungen im Hintergrund machen um personalisierten Werbeanzeigen bereitstellen zu können.

Denn dafür kommunizieren Apps ständig mit dem Internet und senden Daten über das Smartphone an entfernte Server. Dahinter stecken Werbenetzwerke, welche durch Auswertung und Analyse der Daten zielgerichtete Werbeanzeigen bestimmen und an die App senden. Dazu stellen die Werbenetzwerke Entwicklern sogenannte *Software Developement Kits (SDKs)* zur Verfügung. Dabei handelt es sich um fertige Java-Software-Bibliotheken, die von dem Entwickler lediglich in eine App integriert werden müssen um diese zu monetarisieren. Das spart dem Entwickler eine Menge Zeit während der Entwicklung, da er sich nicht selber um die Implementation der Funktionen von Werbeanzeigen kümmern sondern die Funktionalitäten des SDKs lediglich aufrufen muss.

Im Hinblick auf diese Situation stellen sich folgende Forschungsfragen: Welche Daten werden durch die Werbenetzwerke von einem Android-Gerät an die Server übermittelt und wie deckt sich das mit den deren Angaben? Wie werden die übermittelten Daten von dem Gerät erhoben und welchen Einfluss haben diese auf die Privatsphäre eines Benutzers?

1.1 Stand der Forschung

Bereits in der Vergangenheit haben die Werbenetzwerke in Android-Apps Aufmerksamkeit auf sich gezogen. Dabei wurden die Softwarebibliotheken bereits auf die Aspekte Sicherheit und Privatsphäre, im Bezug auf die erhobenen Daten, betrachtet.

Dazu wurde in der Vergangenheit unter anderem ein System namens AdRisk entwickelt [16] um systematisch potentielle Risiken von Werbe-SDKs zu identifizieren. Die Autoren beschäftigten sich mit der Thematik, welche Risiken Werbe-SDKs für die Privatsphäre und Sicherheit eines Smartphones darstellen. In dieser Studie wurden 100.000 Apps aus dem *Play Store* heruntergeladen und daraus die häufigsten 100 Werbe-SDKs extrahiert. Die Forscher kamen zu dem Ergebnis, dass die meisten Bibliotheken private Informationen wie Standort, Anruflisten, Telefonnummer, Browser-Lesezeichen sowie die Liste installierter Anwendungen sammeln und manche Bibliotheken sogar einen Programmcode aus dem Internet laden und diesen auf dem Smartphone ausführen, was ein Sicherheitsrisiko darstellt.

Des Weiteren wurden die Auswirkungen von Werbe-SDKs auf die Privatsphäre eines Endnutzers betrachtet [23]. Dabei verglichen die Autoren zunächst einmal die Unterschiede von Werbeanzeigen im Internetbrowser und in mobilen Anwendungen. Anschließend untersuchten sie die Auswirkungen von 13 Werbe-SDKs auf die Privatsphäre durch eine Analyse der verwendeten Berechtigungen. Die Autoren zeigten erhebliche Schwachstellen im Bezug zur Privatsphäre auf, da einige Werbe-SDKs Berechtigungen abgefragt hatten, welche nicht dokumentiert waren. Diese Berechtigungen wurden bei Verfügbarkeit ausgenutzt, um private Daten zu lesen. Darunter befanden sich Berechtigungen zum Zugriff auf die Kamerafunktionen des Smartphones, sowie auf den Kalender und die Kontakte.

Berechtigungen von Werbe-SDKs wurden ebenfalls schon im Hinblick auf die Entwicklung über mehrere Versionen untersucht [13]. Dazu wurden Werbe-SDKs aus 114.000 Apps klassifiziert und die Veröffentlichungsdaten verschiedenster SDKs geschätzt. Dies wurde durch die Betrachtung der Erscheinungsdaten von Apps durchgeführt, in denen die Software-Bibliotheken enthalten waren. Das Ergebnis dieser Forschung war, dass die Verwendung von Berechtigungen über die Jahre zugenommen hatte und mehr Bibliotheken Berechtigungen verlangten, die ein besonderes Risiko für die Privatsphäre und Sicherheit der Nutzer darstellten.

Andere Arbeiten wiederum beschäftigten sich mit der automatischen Erkennung von Softwarebibliotheken [20] oder mit der Identifizierung und Charakterisierung von Domains, die im Zusammenhang mit Werbeanzeigen und Nutzerverfolgung stehen [24].

Auch wurden Werbe-SDKs im Hinblick auf die Frage, wie sie die Benutzer werbefinanzierter Apps vor böswilliger Werbung schützen, analysiert [17]. Dabei stellten die Au-

toren fest, dass die SDKs im großen und ganzen den Nutzer vor böswilliger Werbung schützen, indem diese die Werbeanzeigen in eine geschützte Umgebung innerhalb der Anwendung laden. Die Autoren zeigten ebenfalls, wie böswillige Werbeanzeigen auf sensitive Informationen von Benutzern durch die Zugriffsberechtigung auf den Speicher schließen können.

1.2 Thema und Struktur der Arbeit

Wie im vorherigen Abschnitt angeführt, haben sich bestehende Arbeiten bereits umfassend mit den Softwarebibliotheken der Werbenetzwerke beschäftigt. Dabei wurde teilweise auf die erhobenen Daten eingegangen, jedoch nur im Bezug auf die Implementation der Softwarebibliotheken und nicht der tatsächlich übermittelten Daten an die Server. Auch sind die bestehenden Forschungen schon etwas in die Jahre gekommen. Die Zahl der kostenlosen Anwendungen ist beispielsweise in den letzten Jahren stark angestiegen. Auch hat sich das Android Betriebssystem in den letzten Jahren verändert.

Um die Thematik also nochmal aufzugreifen und auf den aktuellen Stand zu bringen, werden in dieser Arbeit verschiedene Werbenetzwerke hinsichtlich der Datenübermittlung analysiert. Dabei wird aufgezeigt, welche Daten an die Server der Werbenetzwerke gesendet und wie diese Daten aus dem Android-System angefordert werden. Außerdem findet ein Vergleich der tatsächlich erhobenen Daten mit den Angaben der Werbenetzwerke statt. Abschließend wird diskutiert, welche Bedeutung die übermittelten Daten für die Privatsphäre eines Benutzers haben.

Die folgenden Kapitel dieser Arbeit setzen sich wie folgt zusammen: In Kapitel 2 werden die nötigen Grundlagen vorgestellt, welche für das Verständnis der Thematik und der weiteren Schritte dieser Arbeit notwendig sind. In Kapitel 3 wird die Herangehensweise vorgestellt, mit der die Werbenetzwerke hinsichtlich der Forschungsfragen analysiert wurden. Die Ergebnisse der Analyse werden anschließend in Kapitel 4 aufgezeigt und das entsprechende Fazit in Kapitel 5.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vorgestellt, die für das nötige Verständnis des weiteren Verlaufs dieser Arbeit nötig sind.

2.1 Android

Android ist ein quelloffenes Betriebssystem, das hauptsächlich von mobilen Endgeräten benutzt wird. Die Android Plattform besteht aus den folgenden Hauptbausteinen [2]:

1. **Gerätehardware**: Das Android Betriebssystem kann auf einer Vielzahl verschiedenster Hardware ausgeführt werden, darunter gehören unter anderem Smartphones, Tablets und Smart-TVs. Android ist prozessorunabhängig und nutzt die Vorteile der zugrunde liegenden Hardware.
2. **Android Betriebssystem**: Die Basis des Betriebssystems bildet der Linux-Kernel. Der Zugriff auf Ressourcen einer Hardware wie Kamera, GPS, Bluetooth, Telefon und Netzwerkverbindungen erfolgt über das Betriebssystem.
3. **Android Application Runtime**: Android-Anwendungen werden üblicherweise in der Programmiersprache Java geschrieben und in der Android Runtime (ART) ausgeführt. Alle Anwendungen werden in einer Sicherheitsumgebung innerhalb der sogenannten *Application-Sandbox* ausgeführt. Die Anwendungen erhalten in dieser Sicherheitsumgebung einen bestimmten Teil des Dateisystems, auf dem sie die privaten Anwendungsdaten speichern können.

Android Anwendungen erweitern die Basis des Betriebssystems um zusätzliche Funktionen. Diese Anwendungen können in die folgenden zwei Kategorien eingeteilt werden [2]:

1. **Vorinstallierte Anwendungen**: Android enthält eine Reihe an vorinstallierter Software wie beispielsweise die Anwendungen für Telefon, E-Mail, Kalender, Webbrowser und Kontakte. Dadurch werden die Basisfunktionalitäten des Android Betriebssystems gewährleistet, auf die von anderen Anwendungen zugegriffen werden kann.
2. **Vom Benutzer installierte Anwendungen**: Android bietet eine offene Entwicklungsumgebung, die Software von Drittanbietern unterstützt. Dazu stellt Google einen offiziellen Markt für Android, den *Google Play Store*, zur Verfügung. Neben diesem Markt existieren aber auch noch Drittanbieter-Märkte wie beispielsweise F-Droid. Diese Dienste beinhalten ebenfalls eine große Anzahl an Anwendungen, die zur Installation auf den mobilen Geräten bereitstehen.

Google bietet für das Android Betriebssystem verschiedenste Dienste an. Im Folgenden sind die wichtigsten zwei aufgelistet [2]:

1. **Google Play**: Bei *Google Play* handelt es sich um eine Sammlung von Diensten, mit denen Benutzer eine Vielzahl an Anwendungen auf dem Android-Gerät im Internet suchen, umsonst installieren oder diese kaufen können. *Google Play* bietet Entwicklern eine Möglichkeit auf einfache Weise Android-Nutzer und potentielle Kunden zu erreichen. Neben den Anwendungen bietet *Google Play* auch das Feedback durch die Community, sowie die Überprüfung der Anwendungslizenz und Anwendungssicherheit.
2. **Android Updates**: Dieser Service teilt dem Nutzer eines Android Gerätes mit, ob ein Update für das Betriebssystem verfügbar ist und welche neuen Funktionen zu diesem Update hinzugekommen sind.

2.1.1 Architektur

In diesem Abschnitt wird auf die Architektur des Android Betriebssystems eingegangen [8]. Die Komponenten der Plattform sind in Abbildung 2.1 dargestellt.

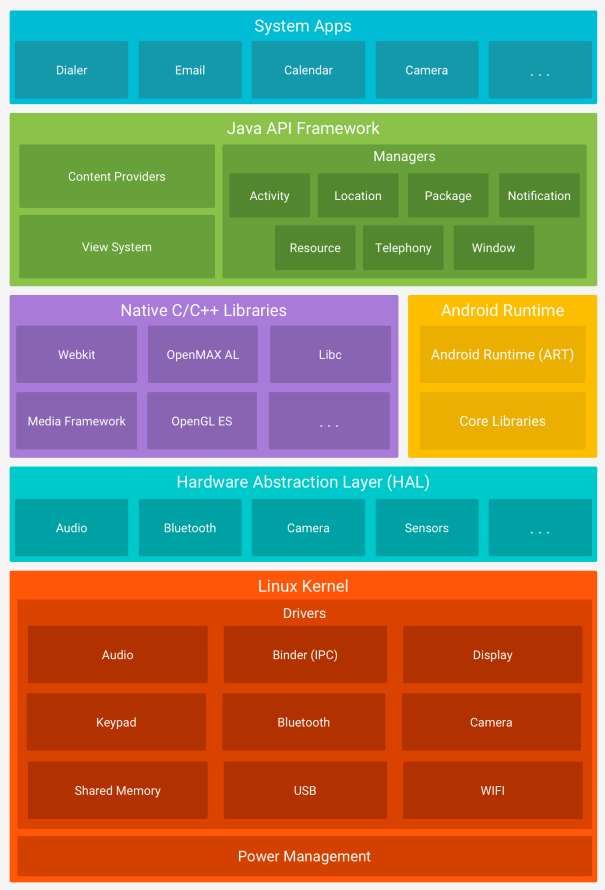


Abbildung 2.1: Android Plattform Architektur [3]

Linux Kernel

Auf der untersten Ebene der Android-Plattform-Architektur befindet sich der **Linux Kernel**. Dieser wird zur Auflösung von Systemaufrufen im Zusammenhang mit Hardwareressourcen verwendet und bildet die Basis des Android Betriebssystems.

Hardware Abstraction Layer

Über dem Kernel befindet sich der **Hardware Abstraction Layer** (HAL), welcher Standartschnittstellen zu Gerätehardwarefunktionen für das übergeordnete *Java-API-Framework* zur Verfügung stellt.

Native Bibliotheken und Android Runtime

Über dem HAL befinden sich die **Nativen C/C++ Bibliotheken** und die sogenannte **Android Runtime** (ART). Bei der ART handelt es sich um eine Laufzeitumgebung, welche für die Ausführung von Anwendungen unter Android und einigen Systemdiensten verwendet wird. Sie ist konzipiert um mehrere virtuelle Maschinen auf Geräten mit wenig Speicher auszuführen. Hierfür wird ein speziell für Android entwickelter DEX-Bytecode ausgeführt, welcher im Gegensatz zum Java Bytecode weniger Ressourcen benötigt.

Java API Framework

Das **Java API Framework** dient als Programmierschnittstelle zum Android Betriebssystem und ist in der Programmiersprache Java geschrieben. Es besteht aus mehreren Bibliotheken, wovon jedes Modul eine Schnittstelle für eine bestimmte Hardwarekomponente implementiert. Beispielsweise existieren Module, die Schnittstellen zur Kamera oder dem Bluetooth-Modul bereitstellen. Über diese Schnittstellen haben Anwendungen und Dienste Zugriff auf das gesamte Funktionsspektrum des Android Betriebssystems. Dadurch haben Entwickler die Möglichkeit, Bausteine von Android in ihre Anwendungen zu integrieren. Wenn eine Anwendung mittels eines API-Aufrufs auf eine Komponente von Android zugreift, wird das entsprechende Bibliotheksmodul für die aufgerufene Hardwarekomponente geladen.

Anwendungen

Auf der oberstem Schicht liegen alle **Anwendungen**, die auf dem System installiert sind. Darunter befinden sich sowohl vorinstallierten Apps wie beispielsweise E-Mail, SMS und Kalender als auch Anwendungen von Drittanbietern. Dabei haben die vorinstallierten Apps keinen besonderen Status und können von Drittanbieter-Apps ersetzt werden, bis auf einige Ausnahmen wie beispielsweise die vorinstallierte Anwendung für

Einstellungen des Betriebssystems. Zudem können Entwickler von Drittanbieter-Apps auf Funktionen von System-Apps zugreifen. Wenn eine App beispielsweise eine SMS Nachricht übermitteln möchte, dann muss diese Funktion von einem Entwickler nicht selbst implementiert, sondern kann über einen Aufruf der bereits installierten SMS-App durchgeführt werden.

2.1.2 Anwendungen

Anwendungen für Android können in den Programmiersprachen Java, Kotlin und C++ geschrieben werden und sind als Dateien mit der Endung `.apk` abgespeichert. Bei diesem Dateiformat handelt es sich um eine Archivdatei, in welcher der kompilierte Quellcode zusammen mit allen Daten und den Ressourcendateien (Bilder, Videos etc.) abgelegt ist. Eine APK-Datei enthält also den gesamten Inhalt einer Android-App und ist gleichzeitig die Datei mit welcher Android-Geräte eine Anwendung installieren [4].

In der Datei ist auch die sogenannte Manifest-Datei enthalten. Diese hat die Bezeichnung `AndroidManifest.xml` und enthält alle wichtigen Konfigurationsdaten über eine Anwendung. Dazu gehören unter anderem die Komponenten einer Anwendung sowie die angefragten Berechtigungen [4].

Quellcode

Der Java-Quellcode einer Anwendung ist in Form von `.java`-Dateien in mehrere Paketen aufgeteilt. Die Pakete kann man sich hierbei wie Ordner in einem Dateisystem vorstellen, welche weitere Pakete oder die `.java`-Dateien in unterschiedlichen Ebenen enthalten. Eine `.java`-Datei enthält ein oder mehrere Klassen mit Funktionen und Variablen, welche die eigentliche Programmlogik bilden. In Abbildung 2.2 ist der Aufbau des Quellcodes in einer vereinfachten Form dargestellt. Dabei stellen die blauen Kästchen mit der Aufschrift *Code* die `.java`-Dateien dar, die den eigentlichen Quellcode enthalten.

Um aus dem Quellcode eine ausführbare App zu machen, wird dieser in das *Dalvik Executeable-Format (DEX)* überführt. Bei DEX handelt es sich um einen binären Code (Maschinensprache), den ein Computer verstehen kann. Dieser Byte-Code ist dabei in einer oder mehreren `.dex`-Dateien innerhalb der `.apk`-Datei abgespeichert und wird in der ART ausgeführt.

Application Sandbox

Jede Anwendung läuft unter Android in einer eigenen geschützten Umgebung, der sogenannten **Application Sandbox**. Hierbei handelt es sich um eine geschützte Laufzeit-

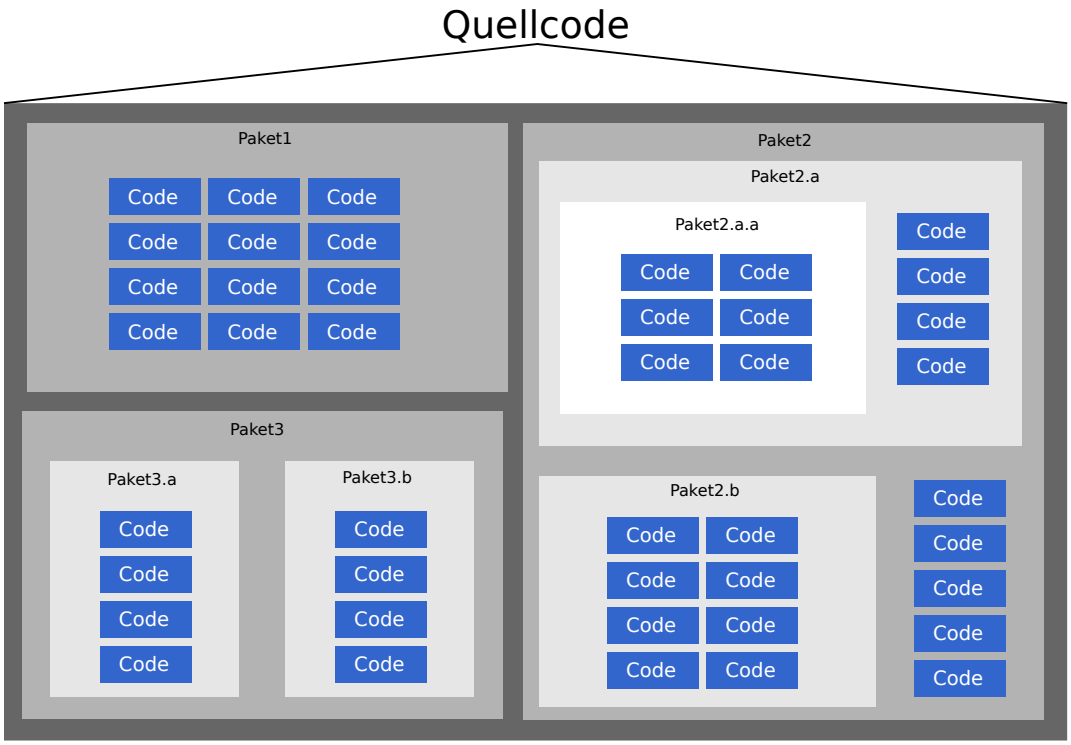


Abbildung 2.2: Schaubild der Struktur des Quellcodes einer Android App

umgebung, welche durch die folgenden Sicherheitsfunktionen von Android geschützt wird [4]:

1. Beim Android-Betriebssystem handelt es sich um ein Mehrbenutzer-Linux-System, bei dem jede Anwendung einen anderen Benutzer hat.
2. Standardmäßig wird jeder Anwendung eine eindeutige Linux-Benutzer-ID zugewiesen (UID). Dabei legt das System für alle Dateien einer Anwendung Berechtigungen fest, sodass nur die, für diese Anwendung zugewiesene Benutzer-ID, auf die Daten zugreifen kann.
3. Jeder Prozess läuft in einer eigenen virtuellen Maschine. Somit wird der Code einer Anwendung isoliert von anderen Anwendungen ausgeführt.
4. Standardmäßig läuft jede Anwendung in einem eigenen Linux-Prozess, welcher gestartet wird, wenn eine Komponente der Anwendung ausgeführt werden muss. Der Prozess wird beendet, wenn er nicht mehr benötigt wird oder das System Speicher für andere Anwendungen braucht.

Durch diese Sicherheitseigenschaften können Anwendungen unter Android standardmäßig nicht miteinander kommunizieren und haben nur beschränkten Zugriff auf das Betriebssystem. Wenn eine Anwendung A beispielsweise versucht auf Daten einer anderen Anwendung B zuzugreifen oder einen Telefonanruf zu tätigen, ohne die benötigten Berechtigungen zu haben, dann verhindert das Betriebssystem diese Aktionen [5].

Das heißt, jede App hat standardmäßig nur Zugriff auf die Komponenten, welche sie für ihre Arbeit benötigt.

Das Teilen von Daten zwischen Apps und der Zugriff auf Systemdienste von Android ist jedoch nicht vollkommen unmöglich. Android stellt hierfür spezielle Möglichkeiten zur Verfügung [4]:

1. Es ist möglich, dass zwei Apps dieselbe *UID* verwenden, womit der Zugriff auf die Dateien des jeweils anderen möglich wird. Außerdem können Anwendungen mit derselben *UID* denselben Prozess und somit dieselbe virtuelle Maschine benutzen um System-Ressourcen zu sparen. Hierfür müssen die Anwendungen jedoch mit dem gleichen Zertifikat signiert sein.
2. Eine App kann die Berechtigungen anfordern, welche nötig sind, um auf die Gerätedaten zugreifen zu können. Der Benutzer muss hierfür der Anwendung explizit die Berechtigung erteilen um beispielsweise einen Zugriff auf die Kontakte, SMS-Nachrichten, Speicher, Kamera oder Bluetooth zu gewähren.

2.1.3 Berechtigungen

Android schützt mithilfe von Berechtigungen den Zugriff von Anwendungen auf vertrauliche Benutzerdaten und Systemfunktionen. So können Anwendungen nicht ohne weiteres auf private Daten wie Kontakte und SMS zugreifen oder sogar die Kamera des Gerätes aufrufen [7].

Der Zugriff auf diese Daten und Systemfunktionen erfolgt bei Android mithilfe von API-Aufrufen über das *Java API Framework*. Einige dieser API-Aufrufe benötigen keine Berechtigungen, andere jedoch schon. Eine Berechtigung ist erforderlich, wenn beispielsweise ein Zugriff auf Standortdaten, Internetverbindungen, Telefon, SMS- und MMS-Funktionen, Kamera oder Bluetooth erfolgt. Dabei sind alle Berechtigungen, die durch eine Anwendung benötigt werden, in der Manifest-Datei definiert.

Android unterteilt Berechtigungen in mehrere Schutzlevel, welche definieren ob ein Benutzer diese Berechtigung gewähren muss oder das System diese automatisch gewährt. Die wichtigsten Schutzlevel sind die normalen und gefährlichen Berechtigungen [6]:

1. **Normale Berechtigungen**: Diese Berechtigungen betreffen Daten und Ressourcen, bei denen das Risiko für die Privatsphäre des Benutzers oder den Betrieb anderer Apps sehr gering ist. Eine normale Berechtigung bei Android ist beispielsweise `android.permission.INTERNET`, welche den Zugriff auf das Internet gewährt. Eine Liste aller normalen Berechtigungen unter Android 9 ist in Tabelle 2.1 zu finden [7]. Das Android-System gewährt diese Berechtigungen automatisch

zum Zeitpunkt der Installation einer Anwendung, wenn diese im Manifest angegeben sind. Dabei fordert das System den Nutzer nicht auf, normale Berechtigungen zu erteilen. Auch widerrufen kann ein Benutzer eine dieser Berechtigungen nicht.

2. **Gefährliche Berechtigungen**: Diese Berechtigungen betreffen Daten und Ressourcen, welche private Informationen des Benutzers enthalten. Auch fallen hierunter Daten und Ressourcen, die möglicherweise die gespeicherten Daten des Benutzers oder den Betrieb anderer Anwendungen beeinträchtigen können. Eine gefährliche Berechtigung ist beispielsweise das Lesen der Kontakte eines Benutzers `android.permission.READ_CONTACTS`. Eine vollständige Liste gefährlicher Berechtigungen kann ebenfalls aus Tabelle 2.1 entnommen werden [7]. Ein Benutzer muss eine gefährlichen Berechtigung explizit gewähren, wenn eine Anwendung solch eine Berechtigung benötigt und diese im Manifest angegeben ist. Bis der Benutzer eine solche Berechtigung genehmigt, kann eine Anwendung keine Funktionen bereitstellen, die von dieser Berechtigung abhängen.

Bei Android besitzen Anwendungen standardmäßig keine Berechtigungen, die sich nachteilig auf den Nutzer, das Betriebssystem oder andere Apps auswirken. Dies umfasst das Lesen und Schreiben von privaten Daten wie Kontakte, Chatverläufe oder der Dateien einer anderen App.

Tabelle 2.1: Berechtigungen

normale Berechtigungen:	**gefährliche Berechtigungen:**
ACCESS_LOCATION_EXTRA_COMMANDS	READ_CALENDAR
ACCESS_NETWORK_STATE	WRITE_CALENDAR
ACCESS_NOTIFICATION_POLICY	READ_CALL_LOG
ACCESS_WIFI_STATE	WRITE_CALL_LOG
BLUETOOTH	PROCESS_OUTGOING_CALLS
BLUETOOTH_ADMIN	CAMERA
BROADCAST_STICKY	READ_CONTACTS
CHANGE_NETWORK_STATE	WRITE_CONTACTS
CHANGE_WIFI_MULTICAST_STATE	GET_ACCOUNTS
CHANGE_WIFI_STATE	ACCESS_FINE_LOCATION
DISABLE_KEYGUARD	ACCESS_COARSE_LOCATION
EXPAND_STATUS_BAR	RECORD_AUDIO
FOREGROUND_SERVICE	READ_PHONE_STATE
GET_PACKAGE_SIZE	READ_PHONE_NUMBERS
INSTALL_SHORTCUT	CALL_PHONE
INTERNET	ANSWER_PHONE_CALLS
KILL_BACKGROUND_PROCESSES	ADD_VOICEMAIL
MANAGE_OWN_CALLS	USE_SIP
MODIFY_AUDIO_SETTINGS	BODY_SENSORS
NFC	SEND_SMS
READ_SYNC_SETTINGS	RECEIVE_SMS
READ_SYNC_STATS	READ_SMS
RECEIVE_BOOT_COMPLETED	RECEIVE_WAP_PUSH
REORDER_TASKS	RECEIVE_MMS
REQUEST_COMPANION_RUN_IN_BACKGROUND	READ_EXTERNAL_STORAGE
REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	WRITE_EXTERNAL_STORAGE
REQUEST_DELETE_PACKAGES	-
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	-
SET_ALARM	-
SET_WALLPAPER	-
SET_WALLPAPER_HINTS	-
TRANSMIT_IR	-
USE_FINGERPRINT	-
VIBRATE	-
WAKE_LOCK	-
WRITE_SYNC_SETTINGS	-

2.2 Monetarisierung

Um Apps kostenlos anbieten zu können, monetarisieren Entwickler diese durch integrierte Werbeanzeigen. Unter Monetarisierung versteht man den Vorgang, ein Produkt in Geld umzuwandeln. Also im Bezug auf Android-Apps die Einbindung von Werbung, um einen finanziellen Nutzen zu erzeugen [19].

Für die Integration von Werbeanzeigen in Apps veröffentlichen Werbedienstleister Softwarebibliotheken, welche auch als *Software Development Kits* (SDKs) bezeichnet werden. Bei einem SDK handelt es sich um ein Java-Archiv, welches dafür gedacht ist, in den Programmcode einer Anwendung integriert zu werden [17] und zur Laufzeit Werbeanzeigen anfordert. Da das SDK ein Teil des Programmcodes einer Android-App ist und dadurch im selben Prozess läuft, hat es Zugriff auf dieselben Berechtigungen, welche die ausführende App an das Android-System anfragt [16, 17, 23].

Im Folgenden wird auf das System hinter den Werbeanzeigen einer Android-App eingegangen sowie aufgeführt, in welchen Formen diese auftreten können. Außerdem werden die verschiedenen Geschäftsmodelle erklärt, die hinter der mobilen Werbung stecken. Zum Abschluss werden die verschiedenen Mittel vorgestellt, welche für zielgerichtete Werbung verwendet werden.

2.2.1 Werbenetzwerk

Um das System hinter den Werbeanzeigen in Apps zu verstehen, werden im folgenden die Teilnehmer vorgestellt. Auch werden die grundlegenden Schritte erläutert, welche hinter dem Darstellen einer Werbeanzeige stecken.

In Abbildung 2.3 sind die Teilnehmer hinter dem System der Werbeanzeigen dargestellt. Dazu gehören Werbetreibende, Werbedienstleister, Entwickler und Verbraucher.

1. **Werbetreibende** sind Unternehmen, welche für ihre Produkte und Dienstleistungen werben möchten und beauftragen dazu einen Werbedienstleister mit einer Werbekampagne. Dazu laden sie die Werbemotive in Form von Texten, Bildern und Videos in das Netzwerk des Werbedienstleisters hoch. In der Werbekampagne werden in der Regel das Werbebudget und die Zielanzahl der Klicks über einen bestimmten Zeitraum festgelegt [15].
2. **Werbedienstleister** sind Unternehmen, welche aus einer Werbeagentur und einem Werbenetzwerk bestehen. Die Werbeagentur kümmert sich um die Marketing-Strategien der Werbetreibenden und das Werbenetzwerk um die Verarbeitung der Werbemotive. Bestandteil des Werbenetzwerkes sind unter anderem ein oder mehrere Ad-Server, welche die Werbeanzeigen verteilen. Außerdem veröffentlichen Werbedienstleister die Softwarebibliotheken, welche für eine Integration in

die Apps, durch die Entwickler, vorgesehen sind.

3. **Entwickler** entwerfen mobile Anwendungen für Benutzer mobiler Endgeräte (Verbraucher). Sie integrieren die Softwarebibliotheken von den Werbedienstleistern in ihren Apps, um diese zu monetarisieren und somit Profit aus ihren Apps zu erzielen.
4. **Verbraucher** sind die Endverbraucher von Apps auf mobilen Endgeräten. Sie bekommen die Werbeanzeigen und werden dadurch beworben.

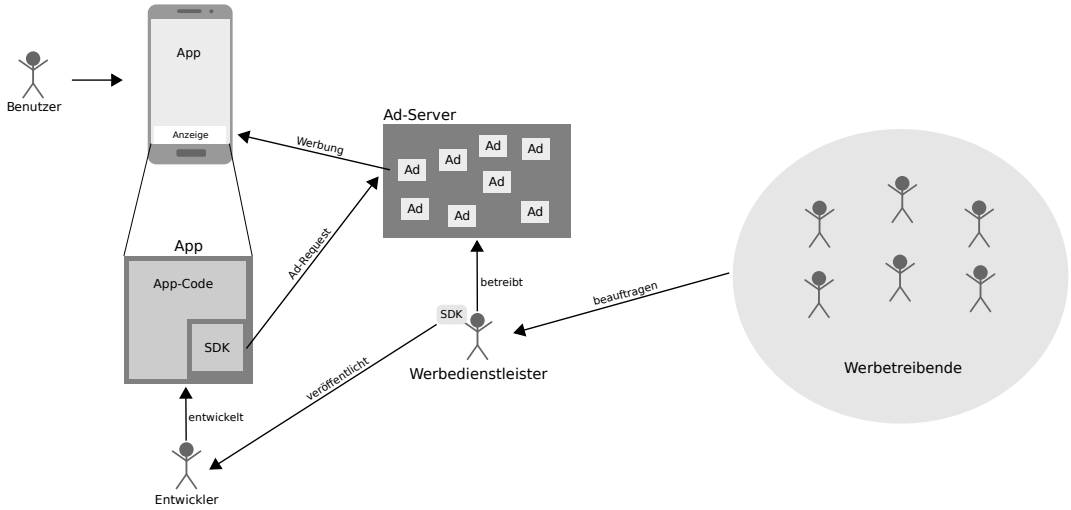


Abbildung 2.3: Teilnehmer des Systems von Werbeanzeigen

Der Entwickler einer App bekommt von einem Werbedienstleister eine Softwarebibliothek (SDK), die er in seine App integriert. Das SDK stellt eine Programmierschnittstelle zu dem Netzwerk des Werbedienstleisters dar, über diese der Entwickler Werbung anfordern kann. Eine Werbeanzeige wird durch die Programmierschnittstelle über einen sogenannten Ad-Request von einem Ad-Server aus dem Werbenetzwerk des Werbedienstleisters angefragt. Dabei werden an den Ad-Request Informationen über den Benutzer beigefügt. Über die angefügten Benutzerinformationen wählt der Ad-Server eine passende Werbeanzeige aus, die in der anfragenden App angezeigt werden soll.

Die integrierte Softwarebibliothek des Werbedienstleisters kümmert sich dabei um die nötigen Programmabläufe zum Verarbeiten der zurückgelieferten Daten des Ad-Servers und die Anzeige der Werbung [23]. Dies hat den Vorteil, dass der Entwickler diese Programmabläufe nicht selber implementieren muss und somit während der Entwicklung Zeit spart, da er die unterschiedlichen Formate der Ad-Requests verschiedener Werbedienstleister nicht verstehen muss [23].

Im weiteren Verlauf dieser Arbeit wird der Begriff Werbenetzwerk benutzt um das Zusammenspiel der Softwarebibliothek eines Werbedienstleisters mit dessen Netzwerk zu beschreiben.

2.2.2 Arten von Werbung

Werbung taucht in den verschiedensten Formen in einer Android-App auf. Tabelle 2.2 stellt die verschiedenen Arten von Werbeanzeigen vor. Dabei sind einige Arten von Werbung aufdringlicher als andere. Zum Beispiel sind native Werbeanzeigen weniger aufdringlich als Videos, da diese sich in die Oberfläche der App integrieren als wären sie ein Teil der App. Videos hingegen belegen meist den gesamten Bildschirm und dauern in der Regel zwischen 30 und 60 Sekunden an.

Tabelle 2.2: Verschiedene Arten von Werbung

Werbung Typ	**Beschreibung**
Banner	Hierbei handelt es sich um ein statisches oder dynamisches Bild, welches direkt auf der Oberfläche einer App angezeigt wird. Meist befindet sich diese Art von Werbung am unteren oder oberen Bildschirmrand.
Native	Dieses Format integriert sich in die App, als wäre es ein Teil der Oberfläche. Diese Art von Werbung ist weniger aufdringlich.
Interstitial	Eine Vollbildanzeige, welche die komplette Oberfläche der Anwendung abdeckt. Diese Art von Werbung wird häufig beim Wechseln verschiedener App-Bildschirme angezeigt
Video	Eine Videoanzeige, welche in der Regel aus einem bis zu 60 Sekunden andauernden Videoclip besteht.
OfferWall	Hierbei handelt es sich um einen Anzeigenblock in der App, der Endbenutzern meist Angebote zu Spielen anderer Entwickler oder desselben Entwicklers anzeigt

2.2.3 Geschäftsmodelle

Die verschiedenen Werbedienstleister bieten den Nutzern ihrer Dienste verschiedene Arten von Geschäftsmodellen an, mit denen Nutzer Werbekampagnen durchführen können. Haupsächlich gibt es 5 verschiedene Geschäftsmodelle - CPM, CPC, CPI, CPA und CPV [11]:

1. Das **CPM**-Modell (Cost-Per-Mile) basiert darauf, wie oft eine Anzeige geschaltet wird. Hierbei wird einem Werbetreibenden jeweils eine Gebühr berechnet, wenn seine Werbeanzeigen 1000-Mal geschaltet werden. Bei diesem Geschäftsmodell verdient der Benutzer mit jeder geschalteten Werbeanzeige Geld und kann außerdem seine Einnahmen prognostizieren.
2. Bei dem **CPC**-Modell (Cost-Per-Click) wird einem Werbetreibenden für jeden Klick auf seine Werbeanzeigen eine Gebühr berechnet. Bei diesem Modell werden die Werbetreibenden nur belastet, wenn Interesse an seinem Produkt besteht und auf die Werbeanzeige geklickt wurde.

3. Das **CPI**-Modell (Cost-Per-Install) belastet Werbetreibende nur dann, wenn ein Klick auf eine ihrer Werbeanzeigen zur Installation einer App führt. Bei diesem Geschäftsmodell handelt es sich um einen speziellen Fall des CPC-Modells.
4. Bei dem **CPA**-Modell (Cost-Per-Action) handelt es sich um eine fortgeschrittene Variante des CPI-Modells. Hierbei werden Werbetreibende belastet, wenn der Benutzer einer App bestimmte Aktionen wie In-App-Käufe, Abonnements, Anmeldungen usw. tätigt.
5. Das verbleibende Geschäftsmodell ist das **CPV**-Modell (Cost-Per-View) und bezieht sich auf Videoanzeigen. Die Kosten für Werbetreibende richten sich danach, wie oft eine Videoanzeige angezeigt wird. Für jedes gezeigte Video eine Gebühr berechnet.

2.2.4 Targeting

Werbetreibende möchten mit ihren Werbekampagnen meiste eine bestimmte Zielgruppe an Personen erreichen. Diese Art der zielgerichteten Werbung wird als *Targeting* bezeichnet. Hierfür gibt es bei mobiler Werbung eine Reihe von Parametern, mit denen sich die Reichweite von Werbekampagnen auf eine bestimmte Zielgruppe eingrenzen lässt [11]. Die wichtigsten Parameter sind in Tabelle 2.3 aufgeführt.

Werbetreibende können über die Parameter beispielsweise für Produkte oder Dienstleistungen werben, die nur für eine bestimmte Region relevant sind. Außerdem besteht die Möglichkeit, in Abhängigkeit des Geschlechtes, Werbekampagnen speziell für Frauen oder Männer durchzuführen. Auch können Werbeanzeigen in Abhängigkeit der Netzwerkverbindung geschaltet werden, in welchem sich die Zielperson befindet. Dies wird unter anderem von den Werbeanbietern für die Entscheidung, ob eine Videoanzeige oder Bildanzeige geschaltet werden soll, genutzt und somit der mobile Datenverbrauch einer Zielperson geschont.

Tabelle 2.3: Parameter für zielgerichtete Werbung

Targeting Typ	**Beschreibung**
Geografisch	Erlaubt eine Werbekampagne auf ein bestimmtes Land oder eine Region zu beschränken.
Gerät	Ermöglicht, dass Anzeigen nur auf bestimmten Geräten angezeigt werden (Smartphones, Tablets etc.). Mit dieser Art des Targetings können Werbetreibende Anzeigen auf Geräten, mit bestimmten Bildschirmgrößen und Hardwarekomponenten, schalten.
Mobilfunkanbieter	Mit dieser Option lassen sich Werbeanzeigen auf Nutzer eines bestimmten Mobilfunkanbieters eingrenzen.
Betriebssystem	Erlaubt das Schalten von Werbeanzeigen in Abhängigkeit des Betriebssystems und dessen Version.
Verbindungstyp	Durch diesen Parameter können Werbeanzeigen je nach Art der Verbindung mit dem Netzwerk geschaltet werden (3G, 4G, LTE, WLAN).
Interessen	Werbeanbieter sammeln über ihre Werbenetzwerke Daten mobiler Nutzer und erstellen daraus Profile. Hiermit können Werbeanzeigen auf eine Zielgruppe mit bestimmten Interessen beschränkt werden.
Geschlecht	Erlaubt die Eingrenzung einer Werbekampagne auf ein bestimmtes Geschlecht.

2.3 Abhören von Netzwerkverkehr

Wie im vorherigen Kapitel angeführt, kommuniziert das SDK eines Werbedienstleisters mit sogenannten Ad-Servern, um Werbeanzeigen für eine bestimmte Zielgruppe an Personen zu schalten.

Die Daten, welche für die zielgerichtete Werbung von den Ad-Servern verwendet werden, sollen im Rahmen dieser Arbeit sichtbar gemacht und anschließend analysiert werden. Dies ist mithilfe eines Proxy Servers möglich, über welchen man den gesamten Netzwerkverkehr leitet und diesen anschließend entschlüsselt.

Daher werden im Folgenden zunächst die nötigen Grundlagen der Datenübertragung in einem Netzwerk erläutert und anschließend das Prinzip und die Funktionsweise eines Proxy-Servers erklärt.

2.3.1 Datenübertragung

Daten werden in einem Netzwerk über Protokolle übertragen. Die relevanten Protokolle für diese Arbeit sind dabei *HTTP* und *HTTPS*, welche im weiteren Verlauf vorgestellt werden.

Grundlegend können über beide Protokolle Daten übertragen werden, jedoch sind diese bei *HTTPS* verschlüsselt. Für eine verschlüsselte Datenübertragung ist zudem ein Zertifikat nötig, dessen Prinzip ebenfalls im Folgenden erläutert wird.

Hypertext Transfer Protocol

Das *Hypertext Transfer Protocol* (HTTP) ist ein Kommunikationsprotokoll, über welches Daten in einem Netzwerk übertragen werden können [21]. Die übertragenen Daten bei HTTP sind standardmäßig nicht verschlüsselt. Diese werden erst unter Verwendung des *Hypertext Transfer Protocol Secure* (HTTPS) verschlüsselt, welches auf HTTP aufbaut.

Kommunikation bei HTTP findet nach dem Client-Server-Prinzip über HTTP-Nachrichten statt [21]. Dabei sendet ein Client eine Anfrage (*HTTP-Request*) an einen Server, welcher daraufhin mit einem *HTTP-Response* antwortet. Dieses Prinzip kann in Abbildung 2.4 betrachtet werden. Dabei besteht jede HTTP-Nachricht aus zwei Bestandteilen, dem Nachrichtenkopf (*HTTP-Header*) und dem Nachrichtenrumpf (*HTTP-Body*).

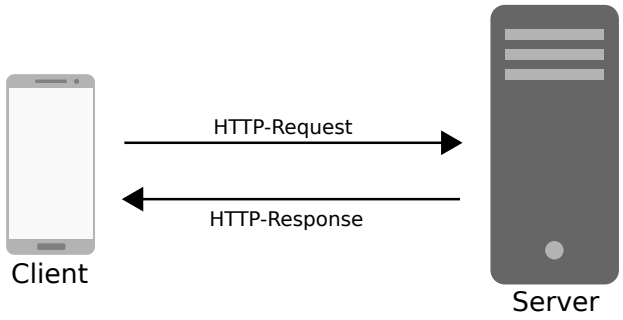


Abbildung 2.4: HTTP-Kommunikation

Beim **HTTP-Request** [21] handelt es sich um eine Anfrage eines Clients an einen Server. Der Nachrichtenkopf besteht unter anderem aus der verwendeten Methode und der aufgerufenen URL[1] sowie der HTTP-Version. Die Methode gibt dem Server an, was dieser mit einer Anfrage machen soll. Dabei sind die am häufigsten verwendeten Methoden *GET* und *POST*.

- Mit der **GET-Methode** werden Ressourcen (Dateien, Bilder etc.) von einem Server angefordert. Die Ressource wird dabei durch die angegebene URL adressiert, in welcher sich Daten als Argumente übermitteln lassen. Diese werden in einer codierten Form an die URL angefügt und sind durch ein Fragezeichen (?) von der eigentlichen URL getrennt. Die Daten selbst sind untereinander durch das &-Zeichen voneinander getrennt und werden in der Form `NAME=WERT` übermittelt.

[1] Uniform Resource Locator, siehe: RFC 3986

- Die **POST-Methode** ist der *GET-Methode* ähnlich. Diese Methode ist für die Übermittlung von Daten an einen Server gedacht. Dabei werden die Daten nicht wie bei der *GET-Methode* an die URL angefügt, sondern im Nachrichtenrumpf der Anfrage übertragen. Damit können größere Datenmengen, wie beispielsweise Textdateien, übermittelt werden.

Die Antwort eines Servers auf eine Anfrage wird als **HTTP-Response** [21] bezeichnet. Diese Antwort besteht unter anderem aus der verwendeten HTTP-Version, einem Status-Code, sowie der Klartextmeldung des zurückgegebenen Status-Codes. Bei dem Code handelt es sich um eine dreistellige Nummer, welche den Client eine Rückmeldung über die Verfügbarkeit der angeforderten Daten gibt.

SSL-Zertifikate

Um eine verschlüsselte Datenverbindung (HTTPS) zu einem Server führen zu können, wird ein SSL-Zertifikat benötigt. Das Zertifikat besteht unter anderem aus dem öffentlichen Schlüssel (Public Key) und der Angabe des Eigentümers. SSL-Zertifikate werden hauptsächlich nach dem X.509[2] Standard erstellt.

Zertifikate werden beim Sender in einem Zertifikatsspeicher abgelegt. Dieser Speicher beinhaltet vorinstallierte Zertifikate von vertrauenswürdigen Zertifizierungsstellen [12]. Man kann auch eigene Zertifikate erstellen, diese werden jedoch nicht als vertrauenswürdig eingestuft.

Eine HTTPS-Verbindung wird über den sogenannten SSL-Handshake aufgebaut, dabei wird auf Basis eines asymmetrischen Verschlüsselungsverfahrens die Identifikation sowie Authentifizierung des Kommunikationspartners durchgeführt [12]. Bei diesem Verbindungsaufbau wird das Zertifikat durch den Server ausgeliefert. Der Client prüft daraufhin in seinem Zertifikatsspeicher nach, ob in diesem ein Zertifikat der Zertifizierungsstelle enthalten ist [12]. Ist das Zertifikat in dem Speicher enthalten, dann ist die Identität des Serverbetreibers bewiesen, da dieser das Zertifikat von der Zertifizierungsstelle bekommen haben muss. Dabei wird der Eigentümer durch den enthaltenen öffentlichen Schlüssel verifiziert. Nach diesem Vorgang wird der weitere Datenverkehr symmetrisch verschlüsselt.

2.3.2 Abhören von Netzwerkverkehr mit der Burp-Suite

Eine Möglichkeit den Netzwerkverkehr abzuhören bietet, die Burp-Suite. Sie ist eine auf Java basierende Plattform zum Testen der Sicherheit von Webanwendungen [25] und enthält den Burp-Proxy, mit dem HTTP und HTTPS Verkehr abgefangen und modifiziert werden kann.

[2] siehe: RFC 5280

Bei diesem Tool handelt es sich um einen Proxy-Server, welcher als Vermittler in einem Netzwerk arbeitet. Dabei nimmt er Anfragen entgegen und leitet sie anschließend stellvertretend für den Anfragenden weiter. Mit einem Proxy Server lässt sich die Kommunikation zwischen einem lokalen Client und dem Internet abhören und auch manipulieren. In Abbildung 2.5 ist das Prinzip eines Proxy-Servers dargestellt.

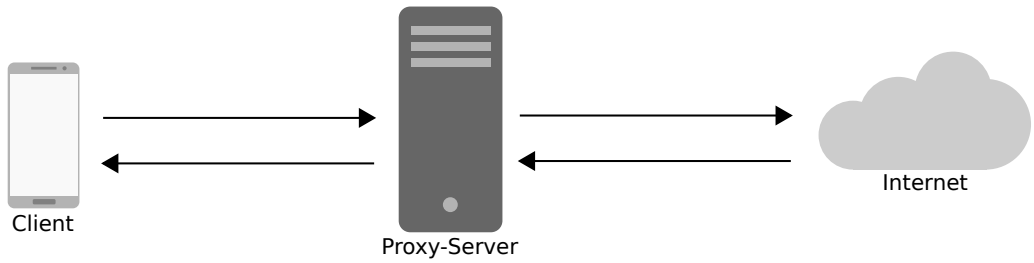


Abbildung 2.5: Prinzip eines Proxy-Servers

Der Burp-Proxy wird vor allem von Penetration-Testern verwendet, um den Datenverkehr zwischen einem Webbrowser und einer Webanwendung zu Analyse- und Manipulationszwecken zu erfassen [25]. Beispielweise kann man einen HTTP-Request pausieren und dessen Parameter verändern, bevor dieser an den Webserver gesendet wird. Mit Burp kann man sowohl HTTP-Requests als auch HTTP-Responses abfangen und somit das Verhalten von Anwendungen, die über das Internet kommunizieren, beobachten.

In Kapitel 3.1.3 dieser Arbeit wird darauf eingegangen, wie man den Burp-Proxy einrichtet, damit dieser den Verkehr zwischen einem Smartphone und dem Internet abfängt. Außerdem wird gezeigt, wie man mithilfe von Burp HTTPS-Verkehr entschlüsseln und die übertragenen Daten anzeigen kann.

3 Methodik

In diesem Kapitel werden die Techniken und Vorgehensweisen vorgestellt, welche zur Analyse der Werbenetzwerke verwendet wurden.

Für diesen Zweck wurde ein gerootetes Android-Smartphone, von der *Zentralen Stelle für Informationstechnik im Sicherheitsbereich*, zur Verfügung gestellt. Dabei handelt es sich um ein *Xiaomi Mi A2*, welches Android Pie (9) als Betriebssystem verwendet und zuvor mit dem *Magisk Manager* ausgestattet wurde.

3.1 Vorbereitungen

In diesem Abschnitt wird auf die Vorbereitungen eingegangen, welche für die spätere Analyse, der Datenerhebung von Werbenetzwerken, notwendig waren. Dazu wurden zunächst Anwendungen aus dem *Google Play Store* beschafft, eine Software zur statischen Analyse von Android-Apps installiert und Vorbereitungen zur Umleitung des Datenverkehrs von dem Smartphone auf den Computer getroffen.

3.1.1 Beschaffen von Anwendungen

Zur späteren Analyse wurden Apps benötigt, die aktuell unter Nutzern im Umlauf sind. Hierzu wurde eine Auswahl an Anwendungen aus dem *Google Play Store* getroffen. Anschließend wurden die ausgewählten Apps heruntergeladen und auf dem Smartphone installiert.

Auswahl der Anwendungen

Es wurde entschieden die Auswahl auf die 100 beliebtesten kostenlosen Anwendungen zu beschränken. Dadurch sollten jene Werbenetzwerke identifiziert werden, die momentan den größten Einfluss bei der Schaltung von Werbeanzeigen bei Endnutzern haben. Google stellt für die beliebtesten kostenlosen Apps im *Play Store* eine eigene Kollektion zur Verfügung. Die Anordnung dieser Anwendungen spiegeln den aktuellen Beliebtheitsgrad, durch die Anzahl ihrer Downloads in der letzten Zeit, wieder. Diese Apps werden momentan am meisten auf mobilen Endgeräten installiert.

Die Top 100 Apps aus dieser Kollektion wurden mit dem Stand vom 27.06.2019 anhand ihrer Anordnung festgehalten. Eine Liste von diesen Apps ist dabei im Anhang A dieser Arbeit zu finden.

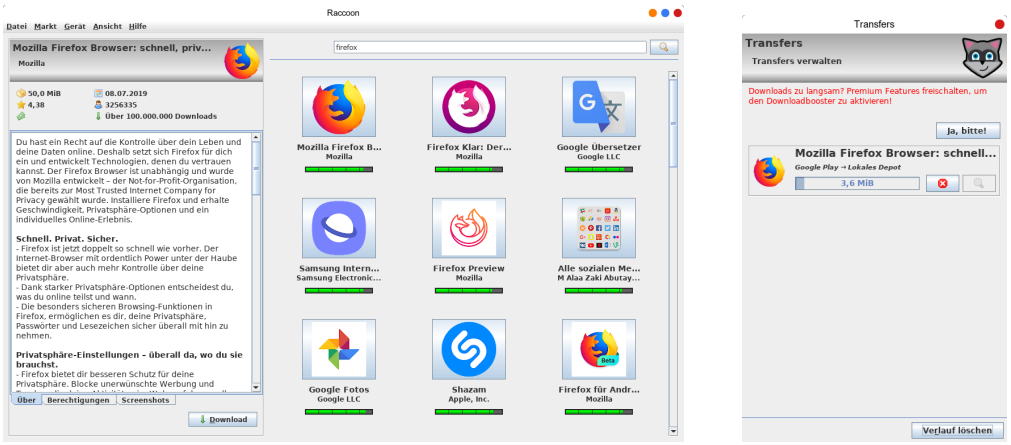
Download der Anwendungen

Nach der Auswahl sollten die Anwendungen nun aus dem *Play Store* heruntergeladen und für Analysezwecke auf einem Linux-Rechner gespeichert werden. Der Download von Apps ist jedoch grundsätzlich nur über ein Android-Gerät möglich. Daher wurde nach einer Möglichkeit gesucht, die lokale Speicherung von Installationspaketen der Apps auf einem Linux-System zu realisieren.

Eine Möglichkeit besteht darin, die Installationspakte in Form der `.apk`-Dateien direkt von dem Smartphone auf den Computer zu kopieren. Diese sind nach der Installation einer App in dem Ordner `/data/app` zu finden, auf den man jedoch ohne einen Root des Gerätes keine Zugriffsrechte hat. Die Installationspakete werden dabei innerhalb von Ordnern, die mit dem Paketnamen der App versehen sind, abgespeichert und tragen die Bezeichnung `base.apk`.

Alle Anwendungen auf dem Smartphone herunterzuladen und nach der Installation die Installationspakete auf einen Computer zu kopieren gestaltet sich jedoch etwas mühselig. Außerdem ist es hilfreich, bei einer solchen Anzahl von Anwendungen, die Installationspakete auf dem Computer mit einer Software verwalten zu können.

Aus diesen Gründen wurde entschieden, das Programm **Raccoon**[3] für den Download der Apps zu verwenden. *Raccoon* ist ein Desktop-Client, mit dem sich die Installationspakete von Android-Anwendungen direkt aus dem *Google Play Store* auf den Computer herunterladen lassen. Das Programm ist in der Programmiersprache *Java* geschrieben, quelloffen und für Linux, MacOS sowie Windows verfügbar.



(a) (b)

Abbildung 3.1: Suche und Download in Raccoon

[3] https://raccoon.onyxbits.de/de/apk-downloader/

Nach dem Download von *Raccoon* wird das Programm unter Linux mit dem Befehl

```
$ java -jar <PROGRAMM>
```

ausgeführt, da es sich um ein ausführbares Java-Archiv handelt. Zudem kann man über die Option `-Draccoon.home=PFAD` angeben, welchen Pfad das Programm zum abspeichern der Apps verwenden soll. Letztendlich wurde der Befehl

```
$ java -Draccoon.home=/media/jannik/DATA/Raccoon
-jar raccoon-4.8.0.jar
```

zum Ausführen von *Raccoon* in der Version 4.8.0 verwendet.

Um *Raccoon* nutzen zu können, benötigt man kein Android-Gerät sondern lediglich ein gültiges Google-Konto, welches man in dem Programm angeben muss. Anschließend wird durch das Programm ein Pseudo-Gerät angelegt, über welches der Download ermöglicht wird. Nach der Einrichtung kann man gezielt nach Apps suchen und diese herunterladen. Abbildung 3.1a zeigt beispielhaft die Suche nach der Anwendung *Firefox* und Abbildung 3.1b den anschließenden Download.

Die zuvor notierten Apps wurden schließlich mit *Raccoon* heruntergeladen und auf dem Linux-Rechner abgespeichert.

Installation der Anwendungen

Nach dem Download mittels *Raccoon*, wurden die Apps zur späteren Analyse auf einem Android-Smartphone installiert. Zu diesem Zweck wurde die *Android-Debug-Bridge (ADB)* verwendet. Bei *ADB* handelt es sich um eine Software-Schnittstelle zum Android-System, die eine Kommunikation zwischen Computer und Android-Gerät ermöglicht [1]. Unter anderem kann man über diese Schnittstelle Anwendungen auf einem Android-Gerät installieren.

Auf dem Computer gespeicherte Installationspakete von Android-Apps werden über das Kommando

```
$ adb install APK_DATEI
```

installiert.

3.1.2 Installation von JADX

Im Bezug auf eine spätere Analyse der Installationspakete von Android-Apps war es nötig, die internen Strukturen einer App, inklusive des Quellcodes, sichtbar zu machen.

Hierzu wurde die Software **JADX**[4] verwendet. *JADX* ist eine quelloffene Software zur statischen Analyse von Android-Apps. Mithilfe dieser Software lassen sich durch Übergabe einer `.apk`-Datei sämtliche Paketstrukturen und die darin enthaltenen Klassen mit dem Java-Quellcode wiederherstellen. Um die Software zum Laufen zu bekommen, sind jedoch einige Schritte[5] notwendig, die im Folgenden erläutert werden.

Zuerst ist es nötig, den Quellcode der Software auf den Computer zu laden. Dazu führt man das Kommando

```
$ git clone https://github.com/skylot/jadx.git
&& cd jadx
```

über das Linux-Terminal aus. Dabei wird ein Ordner **jadx** im aktuellen Pfad des Dateisystems erstellt, in den alle nötigen Dateien und Ordner geladen werden. Ist der Vorgang abgeschlossen, wird in den erstellten Ordner gewechselt. Um die Installation abzuschließen, wird das Kommando

```
$ ./gradlew dist
```

ausgeführt. Dieser Vorgang kann einige Minuten dauern, da weitere Dateien aus dem Internet heruntergeladen werden und anschließend der zuvor heruntergeladene Quellcode kompiliert wird. Bei diesem Prozess werden weitere Dateien und Ordner in dem aktuellen Pfad erstellt.

Ist der Vorgang abgeschlossen, wird in den Ordner **build** gewechselt.

```
$ cd build
```

In diesem Ordner befindet sich ein weiterer Ordner **jadx**, welcher von weiterer Bedeutung ist. Diesen Ordner sollte man aus Gründen der Übersicht an eine andere Stelle im Dateisystem kopieren. Beispielsweise legt man hierzu einen Ordner **Programme** in das *HOME*-Verzeichnis des Linux-Systems an, in welchen man diesen Ordner kopiert. Dies kann beispielsweise über

```
$ mkdir /home/<BENUTZERNAME>/Programme
&& cp -r jadx /home/<BENUTZERNAME>/Programme/
&& cd /home/<BENUTZERNAME>/Programme/jadx
```

erreicht werden. Nach diesem Kommando befindet man sich in dem kopierten Ordner **jadx**.

Starten kann man das Programm nun, indem man das Kommando

[4] https://github.com/skylot/jadx
[5] https://github.com/skylot/jadx#build-from-source

```
$ bash /home/<BENUTZERNAME>/Programme/
jadx/bin/jadx-gui
```

ausführt. Dabei öffnet sich ein Fenster (Open file), in welchem man eine `.apk`-Datei auswählen kann. Nach der Auswahl einer Datei stellt das Programm automatisch die Komponenten einer Android-App inklusive des Quellcodes her.

3.1.3 Abhören des Netzwerkverkehrs

Um den Netzwerkverkehr zwischen Smartphone und Internet abhören zu können, müssen ein paar Vorkehrungen getroffen werden. Darunter gehört unter anderem die Einrichtung des Burp-Proxys auf dem Linux-Rechner, sowie die Umleitung der Netzwerkkommunikation des Smartphones über diesen Proxy. Ein wichtiges Kriterium war es, den verschlüsselten Datenverkehr (HTTPS) entschlüsseln und somit alle übermittelten Daten lesen zu können.

Die Umsetzung dieser Vorkehrungen wird im Folgenden vorgestellt.

Generierung eines Zertifikates

Wie bereits besprochen, ist es nötig, den verschlüsselten Datenverkehr lesen zu können. Hierfür wurde ein eigenes Zertifikat auf dem Linux-Rechner erstellt, welches der Burp-Proxy und das Smartphone anschließend für den HTTPS-Datenverkehr verwenden und mit dem die Entschlüsselung des Datenverkehrs ermöglicht wird. Um ein geeignetes Zertifikat zu erstellen, sind verschiedene Schritte notwendig [18].

Zunächst einmal wurde für die Erstellung des Zertifikates ein eigener Ordner erstellt und in diesen gewechselt:

```
$ mkdir /home/<BENUTZERNAME>/Zertifikate &&
cd /home/<BENUTZERNAME>/Zertifikate
```

Um Zertifikate erstellen und verwalten zu können, ist es notwendig, die Software *OpenSSL* zu besitzen. Falls *OpenSSL* nicht installiert ist, kann man es mittels

```
$ sudo apt install openssl
```

installieren. Für die weitere Vorgehensweise ist es notwendig, die Datei `openssl.cnf` in den zuvor erstellten Ordner zu kopieren. Der Speicherort dieser Datei kann je nach Linux-System variieren. In diesem Fall handelte es sich um ein Ubuntu-Linux und die Datei konnte unter dem Pfad `/usr/lib/ssl/openssl.cnf` gefunden werden.

Die Datei kann mittels des Kommandos

```
$ cp /usr/lib/ssl/openssl.cnf ./
```

in den erstellten Ordner kopiert werden. Nun kann man das eigentliche Zertifikat erstellen. Dazu führt man das Kommando

```
$ openssl req -x509 -days 730 -nodes -newkey
rsa:2048 -outform der -keyout server.key
-out ca.der -extensions v3_ca -config openssl.cnf
```

aus. Nun wird man nach Informationen gefragt, die in das Zertifikat geschrieben werden sollen. Darunter befinden sich unter anderem die eigene Email-Adresse und der Name. Diese Informationen füllt man nach Belieben aus. Bei Abschluss des Vorgangs wird ein privater Schlüssel in Form der Datei `server.key` und das eigentliche Zertifikat `ca.der` erstellt. Der private Schlüssel wird für die Entschlüsselung des verschlüsselten Datenverkehrs benötigt.

Bei der Datei `ca.der` handelt es sich um ein binäres Zertifikat-Format. Für die spätere Installation auf dem Smartphone muss das Zertifikat mit der Dateiendung `.cer`

```
$ cp ca.der ./ca.cer
```

versehen werden.

Der private Schlüssel muss nun noch in ein anderes Format konvertiert werden, damit dieser in den Burp-Proxy eingebunden werden kann. Dafür konvertiert man den Schlüssel zunächst in das *DER*-Format

```
$ openssl rsa -in server.key -inform pem
-out server.key.der -outform der
```

und anschließend in das *pkcs8*-Format.

```
$ openssl pkcs8 -topk8 -in server.key.der
-inform der -out server.key.pkcs8.der
-outform der -nocrypt
```

Einrichtung der Burp-Suite

Die *Burp-Suite* lässt sich in ihrer kostenlosen Variante (Community-Edition) von der Website des Entwicklers *Portswigger* herunterladen[6]. Bei der heruntergeladenen Datei handelt es sich um ein ausführbares Shell-Skript. Zunächst einmal muss sichergestellt

[6] https://portswigger.net/burp/communitydownload

werden, dass dieses die nötigen Rechte zur Ausführung besitzt. Dafür wendet man das Kommando

```
$ sudo chmod 755 <DATEI>
```

auf die heruntergeladene Datei an. Anschließend startet man die Installation, indem man das Shell-Skript mittels:

```
$ ./<DATEI>
```

ausführt. Nach erfolgreicher Installation führt man *Burp-Suite-Community-Edition* aus und legt ein temporäres Projekt an. Anschließend wählt man noch die Standardeinstellungen aus und kann daraufhin die Software starten. Die Proxy-Funktion von *Burp* ist über den Reiter **Proxy** zu erreichen.

Damit der Proxy den Datenverkehr des Smartphones abfangen kann, muss der sogenannte *Proxy-Listener* konfiguriert werden. Dieser ist unter dem Reiter **Options** des Proxy zu finden und wird in Abbildung 3.2 dargestellt. Zuerst legt man einen neuen

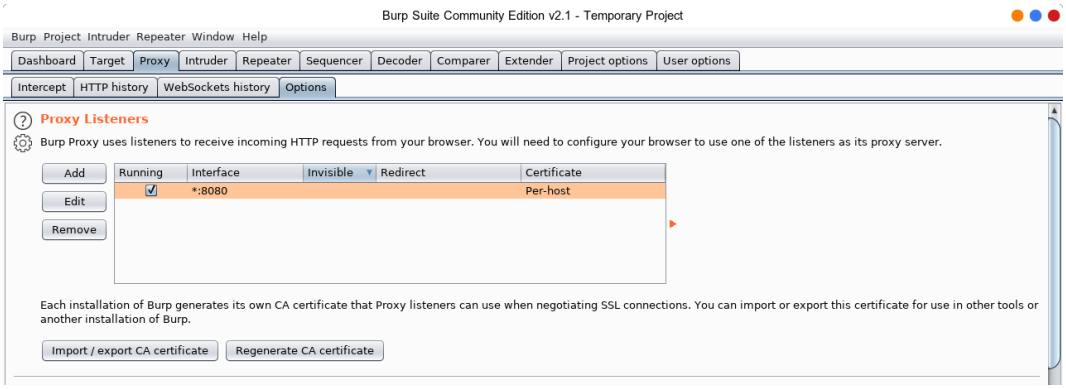


Abbildung 3.2: Proxy Listener

Proxy-Listener über den Button **Add** an und lässt diesen beispielsweise an Port 8080 auf eintreffende Anfragen lauschen. Dazu trägt man den gewünschten Port bei **Bind to port** ein. Weiterhin wählt man bei **Bind to address** die Option *All interfaces* aus. Der *Proxy-Listener* ist nun für das Abfangen des Netzwerkverkehrs aller Geräte konfiguriert, die den Netzwerkverkehr über diesen umleiten wollen.

Als nächstes importiert man das zuvor erstellte Zertifikat in den *Proxy-Listener*. Dies erreicht man über **Import** / **export CA certificate** und wählt unter **Import** *Certificate and private key in DER format* aus. Nun wird man aufgefordert, die Dateien, welche das Zertifikat und den privaten Schlüssel beinhalten, anzugeben. Dazu wählt man für das Zertifikat die Datei `ca.der` und für den privaten Schlüssel die Datei `server.key.pkcs8.der` aus, wie in Abbildung 3.3 zu erkennen. Wenn anschließend keine Fehlermeldung auftritt, kann der Burp Proxy nun sämtlichen HTTPS-Verkehr entschlüsseln, welcher mit dem angegebenen Zertifikat signiert und verschlüsselt wurde.

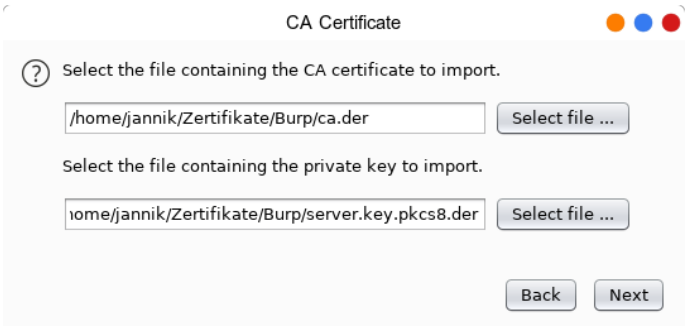


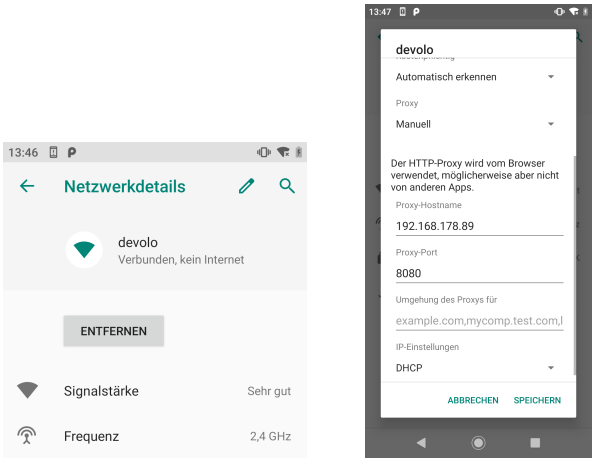
Abbildung 3.3: Burp - Auswahl des Zertifikates und des privaten Schlüssels

Wie man dem Smartphone mitteilt, den HTTPS-Netzwerkverkehr mit einem selbst erstellten Zertifikat zu verschlüsseln, wird im Folgenden Abschnitt vorgestellt.

Umleitung von Datenverkehr des Smartphones

Nachdem der *Burp-Proxy* eingerichtet wurde, muss das Smartphone noch für die Umleitung des Netzwerkverkehrs über einen Proxy konfiguriert werden.

Dazu kann man bei Android in den Einstellungen des Gerätes einen Proxy angeben, über den der Netzwerkverkehr geleitet werden soll. Dafür wählt man die Einstellungen für *WLAN* unter **Netzwerk und Internet** aus. Anschließend wählt man das Netzwerk aus, in welchem sich der Computer mit dem *Burp-Proxy* befindet. Wenn man alles richtig gemacht hat, befindet man sich nun in der Einstellungsebene **Netzwerkdetails** des jeweiligen Netzwerkes. Die Einstellungsebene ist beispielhaft in Abbildung 3.4a dargestellt, kann jedoch je nach Android-Version variieren.



(a) (b)

Abbildung 3.4: Konfiguration eines Proxy Servers bei Android

Hier wählt man die Bearbeitung aus und gelangt zu den Einstellungsoptionen des ge-

wählten Netzwerkes. Anschließend wählt man unter **Erweiterte Optionen** für die Proxy-Optionen *Manuell* aus. Dabei tauchen weitere Optionen für einen HTTP-Proxy auf. In diesen Optionen trägt man unter **Proxy-Hostname** die IP-Adresse des Computers ein und unter **Proxy-Port** den zuvor gewählten Port des Proxy-Listeners. Die fertige Konfiguration kann in Abbildung 3.4b betrachtet werden.

Installation des Zertifikates auf dem Smartphone

Ist die Konfiguration des HTTP-Proxy abgeschlossen, muss noch das zuvor erstellte Zertifikat auf dem Smartphone installiert werden. Mithilfe dieses Zertifikates wird dann der ausgehende HTTPS-Netzwerkverkehr auf dem Smartphone verschlüsselt.

Zertifikate können unter Android direkt aus dem Dateisystem heraus installiert werden. Dazu kopiert man die zuvor erstellte Datei `ca.cer` von dem Computer auf das Smartphone. Dies ist auch mittels ADB möglich, dazu führt man den Befehl

```
$ adb push ca.cer /sdcard
```

im Speicherort der Datei `ca.cer` aus. Nach diesem Vorgang befindet sich das Zertifikat auf dem Smartphone.

Installieren kann man das Zertifikat nun, indem man in den Einstellungen unter **Netzwerk und Internet** *WLAN* auswählt und anschließend die **WLAN-Einstellungen** aufruft. In den WLAN-Einstellungen kann man Zertifikate nun über die Option **Zertifikate installieren** auf dem Smartphone installieren. Wählt man die Option aus, wird der Dateimanager des Smartphones geöffnet, in dem man das zuvor kopierte Zertifikat suchen und auswählen muss. Daraufhin muss man aus Sicherheitsgründen einen Code oder ein Muster eingeben um fortfahren zu können. Falls kein Code oder Muster für das Gerät festgelegt wurde, ist zuerst eine Einrichtung dieser Sicherheitsmaßnahme erforderlich. Nach der Eingabe des Codes oder Musters kann man dem Zertifikat noch einen Namen zuweisen und daraufhin installieren.

Vertrauen des installierten Zertifikates

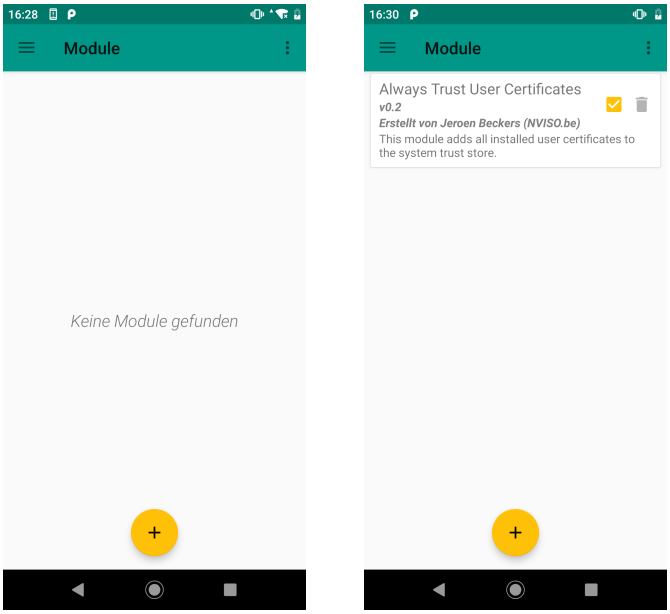
Damit Anwendungen unter Android einem selbst erstellten Zertifikat vertrauen, müssen Modifikationen an dem Android Betriebssystem vorgenommen werden. Für diesen Vorgang ist ein Root des Gerätes und die installierte Anwendung *Magisk Manager* notwendig.

Android teilt den Zertifikatsspeicher in System- und Benutzerzertifikate auf. Dabei benötigen Anwendungen zum Aufbau einer verschlüsselten Verbindung ein vertrauenswürdiges Zertifikat aus dem System-Zertifikatsspeicher. Damit Anwendungen den Datenverkehr mit dem zuvor erstellten Zertifikat verschlüsseln und man somit diesen über

den *Burp-Proxy* wieder entschlüsseln kann, muss man das Zertifikat in den System-Zertifikatsspeicher importieren.

Hierzu hat *NVISIO LABS* das Zusatzmodul *Magisk Trust User Certs*[7] veröffentlicht, welches alle installierten Benutzerzertifikate automatisch als Systemzertifikate festlegt. Das Modul lädt man in Form eines `.zip`-Archivs von der *Releases*-Seite[8] des Github Repositorys herunter und kopiert das Archiv, wie zuvor das Zertifikat, auf das Smartphone. Anschließend startet man den *Magisk Manager* und wechselt im Navigationsmenü auf die Seite *Module*, wie in Abbildung 3.5a zu sehen. Hier lassen sich Module über den gelben Button (+) hinzufügen. Man fügt das Modul hinzu, indem man es im Dateibrowser auswählt und dieses im Anschluss installiert. Wenn die Installation erfolgreich war, muss man das Gerät neu starten, damit die Installation abgeschlossen werden kann.

Nach dem Neustart sind alle Benutzerzertifikate als Systemzertifikate festgelegt.



(a) (b)

Abbildung 3.5: Magisk Module

[7] https://github.com/NVISO-BE/MagiskTrustUserCerts
[8] https://github.com/NVISO-BE/MagiskTrustUserCerts/releases

3.2 Analyse der Apps

Nach den Vorbereitungsmaßnahmen wurden die auf dem Smartphone installierten Anwendungen hinsichtlich ihrer enthaltenen Werbenetzwerke analysiert.

3.2.1 Identifizierung von Werbenetzwerken

Wie in Kapitel 2.2.1 beschrieben, geben Werbenetzwerke Softwarebibliotheken für die Einbindung ihrer Werbedienste an die Entwickler von Apps heraus. Nutzt eine App die Dienste eines Werbedienstleisters, dann ist in dieser die zugehörige entsprechende Softwarebibliothek des Werbenetzwerkes enthalten.

Verantwortliche Werbenetzwerke und deren Softwarebibliotheken mussten zunächst einmal identifiziert werden. Die dazu angewandten Methoden werden im folgenden vorgestellt.

Appbrain Ad Detector

Zu Anfang war wenig über die unterschiedlichen Werbenetzwerke bekannt, daher wurde nach einer Methode gesucht um vorhandene Software von Werbenetzwerken in Apps zu identifizieren. Die einzige Methode, welche ein zufriedenstellendes Ergebnis lieferte, war die Software *Appbrain Ad Detector*[9].

Dabei handelt es sich um eine App, die über den *Google Play Store* auf dem Smartphone installiert[10] werden kann. Innerhalb der Anwendung hat man die Möglichkeit, für jede installierte App auf dem Gerät nachzuschauen, welche Werbenetzwerke und Entwicklerwerkzeuge enthalten sind. Dazu analysiert die Anwendung den Code aller auf einem Gerät installierten Anwendungen auf die enthaltenen Softwarebibliotheken der Werbenetzwerke [9].

Mithilfe des *Appbrain Ad Detector* wurden daraufhin die in den Apps enthaltenen Werbenetzwerke identifiziert und notiert. In Abbildung 3.6 ist unter **Werbenetzwerk** beispielhaft das Analyseergebnis der enthaltenen Werbenetzwerke in der App *mobile.de* zu sehen. Anschließend wurde aus den so erhaltenen Ergebnissen eine Liste aller identifizierten Werbenetzwerke erstellt und deren Vorkommen unter den 100 Apps abgezählt.

[9] https://www.appbrain.com/app/appbrain-ad-detector/com.appspot.swisscodemonkeys.detector
[10] https://play.google.com/store/apps/details?id=com.appspot.swisscodemonkeys.detector&hl=de

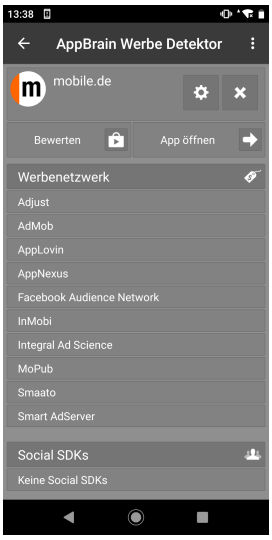


Abbildung 3.6: Werbesoftware in *mobile.de*

Identifizierung der Softwarebibliotheken

Mithilfe der im vorherigen Schritt erhaltenen Listen von Werbenetzwerken sollten nun die verantwortlichen Softwarebibliotheken identifiziert werden. Dazu wurde die zuvor erstellte Liste an Werbenetzwerken durchgearbeitet. Für jeden der erfassten Werbenetzwerke wurde geschaut, in welchen Apps deren Softwarebibliotheken enthalten waren.

Die Anwendungen mit der Softwarebibliothek eines Werbenetzwerkes wurden daraufhin in der Software *JADX* betrachtet. Hierzu wurde nach Stichwörtern wie *ads*, *sdk* in Verbindung mit dem Namen des Werbenetzwerkes gesucht. Dadurch konnten die Pakete identifiziert werden, in denen die Softwarebibliotheken der Werbenetzwerke enthalten sind. Als Beispiel ist in Abbildung 3.7 die dekompilierte App *mobile.de* in *JADX* dargestellt. Das Paket `facebook.ads` ist hierbei hervorgehoben. Bei diesem Paket handelt es sich beispielsweise um die Softwarebibliothek des Werbenetzwerkes *Facebook Audience Network*. Der vollständige Name des Paketes lautet `com.facebook.ads`.

Die dadurch erhaltenen Paketnamen wurden in die Liste der Werbedienstleister ergänzt.

3.2.2 Analyse der Werbenetzwerke

Nach der Identifizierung von Werbenetzwerke und deren Softwarebibliotheken sollte als nächstes der Datenaustausch zwischen dem Smartphone und den Ad-Servern betrachtet werden. Dazu wurde zunächst eine Auswahl an Werbenetzwerken getroffen, deren Datenübermittlung analysiert werden sollte. Die ausgewählten Kandidaten wurden dar-

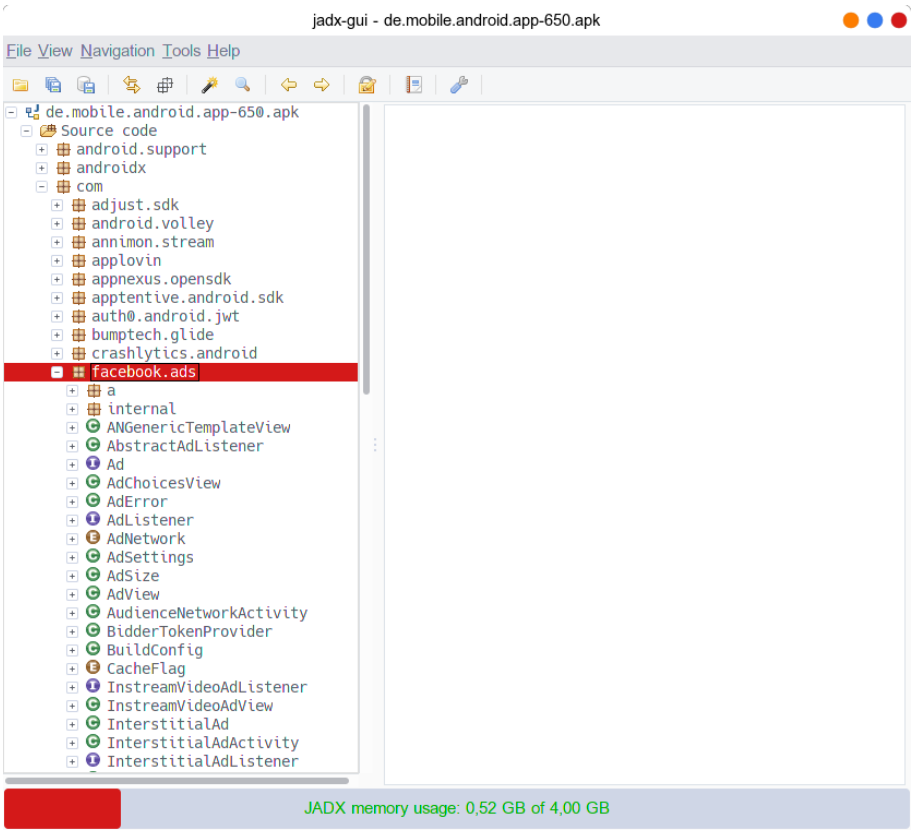


Abbildung 3.7: Softwarebibliothek des *Facebook Audience Network*

aufhin dahingehend untersucht, welche Angaben diese zu der Datenerhebung machen und welche Daten sie letztendlich wirklich erheben und übermitteln. Hierzu wurden die einzelnen Datenschutzerklärungen der Werbenetzwerke und anschließend der Netzwerkverkehr des Smartphones mithilfe der in 3.1.3 vorgestellten Methode betrachtet

Auswahl der zu analysierenden Werbesoftware

Für die Analyse der Datenermittlung wurde eine Auswahl von bestimmten Werbenetzwerken getroffen. Darin wurden die Analyseergebnisse aus 3.2.1 sowie globale Statistiken berücksichtigt.

Die Entwickler der in 3.2.1 vorgestellten Anwendung *Appbrain Ad Detector*, betreiben zusätzliche eine Internetplattform[11]. Auf dieser Internetplattform veröffentlichen sie Statistiken über das Vorkommen von Werbenetzwerken in Android-Apps. Im Bezug auf diese Arbeit war dazu die Statistik über die auf Endgeräten am meisten verbreiteten Werbenetzwerke [9] von Bedeutung.

Mithilfe dieser Statistik wurden Werbenetzwerke bevorzugt in die Auswahl übernommen,

[11] https://www.appbrain.com/

wenn diese das gleiche Vorkommen wie anderer Werbenetzwerke unter den 100 analysierten Anwendungen hatten, aber in der Statistik ein größeres Vorkommen vorwiesen. Somit wurde versucht, eine ausgewogene Auswahl an Werbenetzwerken zu erzeugen, besonders im Hinblick auf tägliche Änderungen unter den 100 beliebtesten Apps im *Google Play Store*.

Abfangen der übermittelten Daten

Um herauszufinden, welche Daten die Softwarebibliotheken der Werbenetzwerke in den Apps an die Ad-Server übertragen, wurde der Netzwerkverkehr des Smartphones, mit dem in 3.1.3 eingerichteten Burp-Proxy-Server, abgefangen.

Dazu wurde *Burp-Suite-Community-Edition* zunächst einmal gestartet und der Proxy ausgewählt. Anschließend wurde sichergestellt, dass alle Vorkehrungen getroffen wurden, die zum erfolgreichen Mitlesen des Netzwerkverkehrs zwischen Smartphone und dem HTTP-Proxy nötig sind.

Anschließend wurde mithilfe der Ergebnisse aus 3.2.1 nachgeschaut, welche Anwendungen die Softwarebibliothek eines ausgewählten Werbenetzwerkes beinhaltet. Mithilfe dieser Informationen wurde die Datenübermittlung jedes auserwählten Werbenetzwerkes betrachtet. Dazu wurden die Apps, welche die entsprechende Softwarebibliothek enthalten, systematisch ausgeführt. Wenn also die Datenübermittlung eines Werbenetzwerkes **W** betrachtet werden sollte, dann wurde eine Anwendung **A** ausgewählt, welche die Softwarebibliothek von **W** enthält.

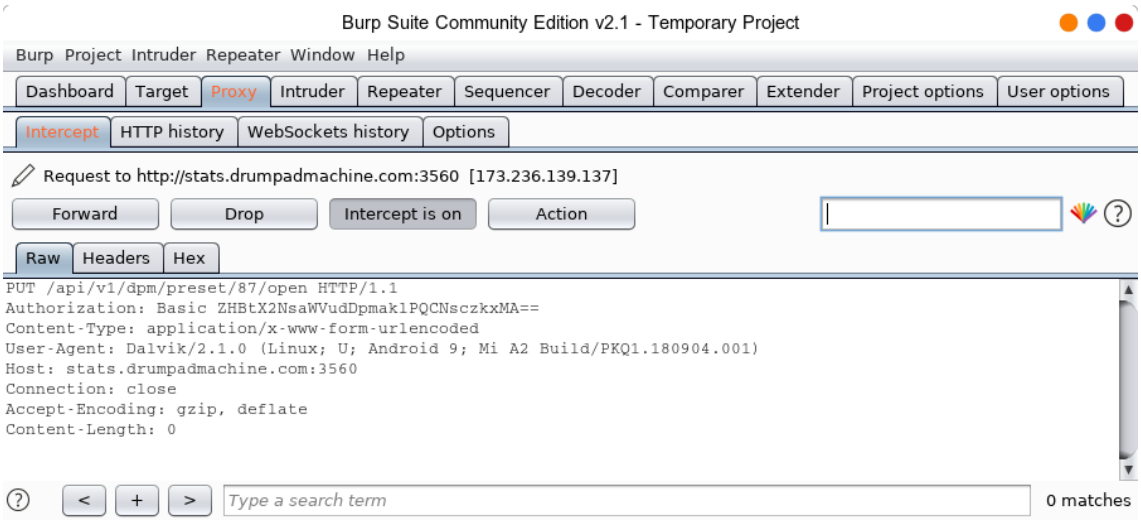


Abbildung 3.8: Burp-Proxy - Zustand nach dem Ausführen einer App

Die Anwendungen wurden ausgeführt und gleichzeitig im *Burp-Proxy* betrachtet. Dazu musste innerhalb des *Burp-Proxy*, nach dem Starten einer Anwendung, das Abfangen der Nachrichten unter dem Tab **Intercept** ausgeschaltet werden. Ausschalten kann man

diese Funktion, indem man den Button **Intercept is on** auswählt, welcher in Abbildung 3.8 zu sehen ist. Dadurch leitet der *Burp-Proxy* nun den Netzwerkverkehr weiter und agiert als **Man In The Middle**.

Anschließend kann der Netzwerkverkehr über den Tab **HTTP history** betrachtet werden. Hier befindet sich eine tabelle (*History table*), in welcher der Netzwerkverkehr in Form der HTTP-Nachrichten aufgelistet wird. Diese Tabelle kann in Abbildung 3.10 betrachtet werden.

Standardmäßig werden jedoch nicht alle HTTP-Nachrichten angezeigt. Dafür verantwortlich ist ein Filter. Um diesen auszuschalten und somit alle Nachrichten anzuzeigen, muss die Option **Show all** in den Filtereinstellungen aktiviert werden, welche in Abbildung 3.9 betrachtet werden kann. Zu den Filtereinstellungen gelangt man, indem zwischen dem Tab **HTTP history** und der Tabelle in Abbildung 3.10 das Feld **Filter** auswählt.

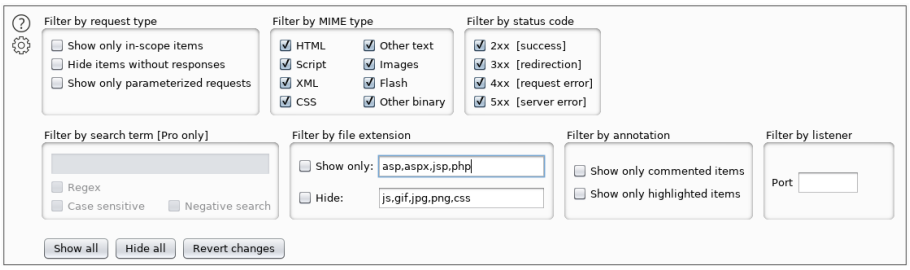


Abbildung 3.9: Burp-Proxy - Filtereinstellungen

Somit konnte der ausgehende HTTP-Verkehr zusammen mit den übermittelten Daten aufgezeichnet und gleichzeitig auch entschlüsselt werden, da hierfür in Kapitel 3.1.3 die nötigen Vorkehrungen getroffen wurden.

Extrahieren der übermittelten Daten

Im nächsten Schritt wurden die übermittelten Daten aus den übertragenen HTTP-Nachrichten extrahiert. Dazu wurden die Nachrichten des Werbenetzwerkes über den aufgerufenen Host identifiziert. Eine große Hilfe war hierbei die alphabetische Sortierung der Host-Namen, welche durch Auswählen der Spalte *Host* in der **History table** (Abbildung 3.10) möglich ist.

Für jedes Werbenetzwerk wurde somit der zugehörige HTTP-Verkehr über die HTTP-Methoden **POST** und **GET** betrachtet. Wie in Kapitel 2.3.1 beschrieben, können Daten über diese HTTP-Methoden übermittelt werden. Dabei wurden die Daten bei *POST* aus dem Nachrichtenrumpf kopiert und abgespeichert.

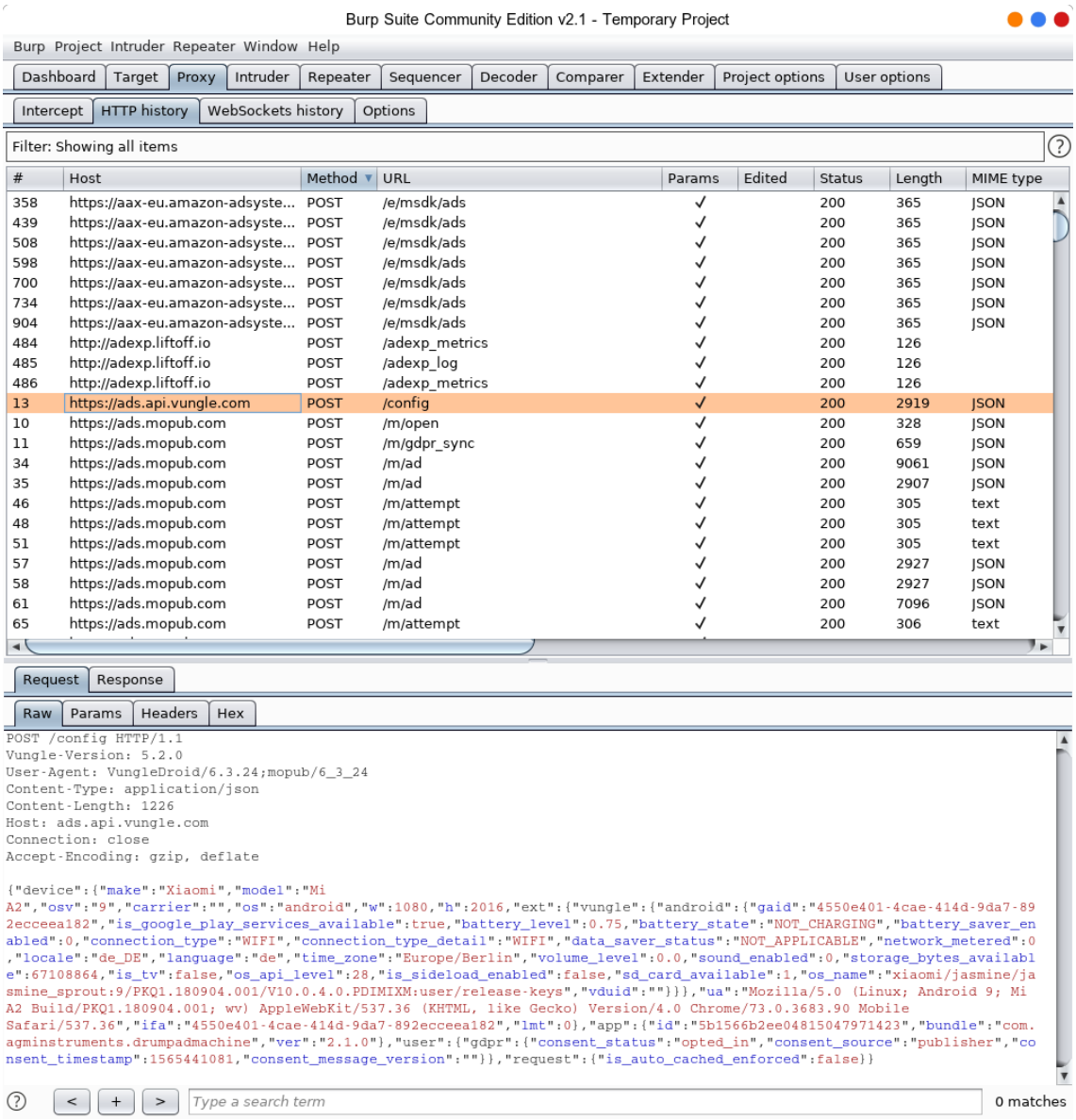


Abbildung 3.10: Burp-Proxy - Netzwerkverkehr

Als Beispiel lässt sich die HTTP-POST Nachricht an den Host des Werbenetzwerkes *Vungle* `https://ads.api.vungle.com` betrachten, welche in Abbildung 3.10 hervorgehoben ist. Diese Nachricht beinhaltet eine **JSON**-Datei im Nachrichtenrumpf, deren Inhalt in der unteren Hälfte von Abbildung 3.10 in geschwungenen Klammern ({...}) zu sehen ist. Die Daten wurden in diesem Fall in dem ursprünglichen Dateiformat auf dem Computer abgespeichert.

Daten, welche über die HTTP-Methode *GET* übertragen wurden, konnten aus den aufgerufenen URLs extrahiert und anschließend ebenfalls auf dem Computer abgespeichert werden.

Angaben der Werbenetzwerke zur Datenerhebung

Im Bezug auf die Angaben, welche die ausgewählten Werbenetzwerke über die Datenerhebung ihrer Softwarebibliotheken machen, wurden die einzelnen Datenschutzerklärungen durchgelesen. Dabei wurde auf Angaben seitens der Werbenetzwerke zu erhoben Daten und zur Datenweiterverarbeitung geachtet.

Erstellen einer Übersichtstabelle

Mithilfe der gewonnenen Informationen aus den extrahierten Daten des Netzwerkverkehrs und den Datenschutzerklärungen der Werbenetzwerke wurde eine Übersichtstabelle erstellt. In dieser wurden die Erkenntnisse der Analysen für jedes Werbenetzwerk zusammengeführt.

Dabei wurde jedes der Werbenetzwerke dahingehend überprüft, welche Daten in der Datenschutzerklärung für die Erhebung angegeben und welche Daten in Wirklichkeit an den Host des Werbenetzwerkes von dem Smartphone übertragen wurden.

Die Erkenntnisse, welche bei dem Erstellen der Übersichtstabelle entstanden sind, werden in Kapitel 4 vorgestellt. Die Übersichtstabelle selbst befindet sich im Anhang B dieser Arbeit.

3.2.3 Statische App Analyse

Im Anschluss an die Analyse der Datenerhebung von den Werbenetzwerken wurden die Softwarebibliotheken innerhalb der Apps analysiert. Bei der Analyse sollten die für die Datenerhebung verantwortlichen Programmteile und die darin getätigten API-Aufrufe an das Android-System aufgedeckt werden. Mithilfe der folgenden Schritte wurde ein Verständnis entwickelt, wie die Softwarebibliothek eines Werbenetzwerkes die Daten von dem Smartphone erhebt.

Reverse Enginerring

Mithilfe der in 3.1.2 vorgestellten Software *JADX* wurde der Quellcode und die interne Struktur einzelner Apps untersucht. In den Softwarebibliotheken, welche in 3.2.1 identifiziert wurden, wurde nach Programmteilen gesucht, die Informationen über das Gerät aufrufen. Dabei wurden Methoden des Reverse Engineering angewandt, welche im Praxismodul bei der *Zentralen Stelle für Informationstechnik im Sicherheitsbereich* erlernt wurden. Durch Analyse des Quellcodes wurden die Programmteile innerhalb einer Softwarebibliothek identifiziert, die Informationen über API-Aufrufe aus dem Android-Betriebssystem anfordern. Die getätigten API-Aufrufe wurden dabei notiert.

Ein Beispiel für einen solchen Programmteil einer Softwarebibliothek ist in Abbildung 3.11 zu sehen. Hierbei handelt es sich um einen Programmausschnitt der Softwarebibliothek des Werbenetzwerkes *Unity Ads*. Dieser Ausschnitt stellt einen Teil der Klasse `com.unity3d.ads.device.Device` dar, in welcher über API-Aufrufe Informationen aus dem Android-System angefordert werden.

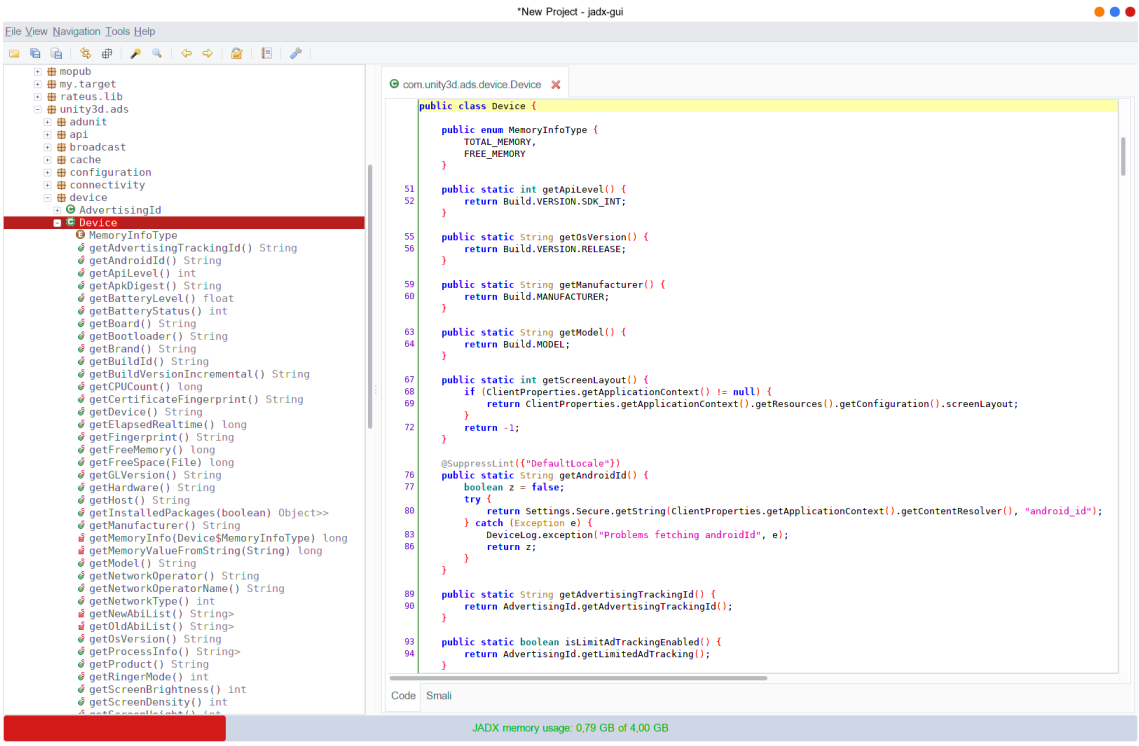


Abbildung 3.11: JADX - Ausschnitt der Klasse `Device` von *Unity Ads*

Zuordnen der API-Aufrufe

Für jeden der somit ermittelten API-Aufrufe wurde anschließend in der offiziellen Dokumentation von Android[12] nachgeschaut, welche Information dieser aus dem Android-

[12] https://developer.android.com/reference

System anfordert. Soweit es möglich war, wurden die Aufrufe in der zuvor erstellten Übersicht (3.2.2) den erhobenen Daten zugeordnet. Dadurch sollte ein Überblick darüber geschaffen werden, mithilfe welcher Mittel die Werbenetzwerke Informationen aus dem Android System beschaffen.

4 Evaluation

In diesem Kapitel werden die Ergebnisse aus den in Kapitel 3 durchgeführten Analysen präsentiert.

Dabei werden zunächst die Werbenetzwerke vorgestellt, welche durch die Analyse der Apps identifiziert werden konnten. Anschließend werden die Ergebnisse aus der Analyse von übertragenen Daten präsentiert sowie aus der statischen Analyse von Installationspaketen.

4.1 Werbenetzwerke in Android Apps

Eine Analyse von enthaltenen Werbenetzwerken wurde, wie in 3.2.1 beschrieben, mit den installierten Android Apps durchgeführt. Dazu werden im Folgenden die Ergebnisse der Werbenetzwerke mit dem größten Vorkommen sowie die Statistik zu der globalen Verbreitung von Werbenetzwerken vorgestellt. Zum Abschluss werden die Werbenetzwerke genannt, deren Auswahl aus den Ergebnissen der Analyse und der Statistik getroffen wurde.

4.1.1 Verbreitung in den Top 100

Bei der Analyse der Top 100 kostenlosen Apps aus dem *Google Play Store* konnte festgestellt werden, dass einige Werbenetzwerke in vielen Anwendungen enthalten waren. Andere hingegen waren in nur sehr wenigen Anwendungen enthalten. Mit diesem Ergebnis konnten die führenden Werbenetzwerke unter den analysierten Apps identifiziert werden. Eine Übersicht über die Werbenetzwerke mit dem größten Vorkommen kann in Abbildung 4.1 betrachtet werden.

Spitzenreiter ist dabei das von Google bereitgestellte Werbenetzwerk *AdMob*, welches in 43 von den analysierten 100 Anwendungen zu finden war. Auf dem zweiten Platz befindet sich der Dienst *Adjust*, dicht gefolgt von dem Werbenetzwerk *Facebook Audience Network* des Social Media Unternehmens Facebook auf dem dritten Platz. Außerdem waren die Werbenetzwerke *MoPub*, *Appsflyer* und *Amazon Mobile Ads* in mindestens 10 der analysierten Anwendungen zu finden. Eine Auflistung aller identifizierten Werbenetzwerke und deren Vorkommen ist im Anhang A dieser Arbeit zu finden.

Weiterhin wurde bei der Analyse der Apps festgestellt, dass in der überwiegenden Anzahl gleich mehrere Werbenetzwerke enthalten waren. Spitzenreiter war dabei die App *Drum Pad Machine*, in welcher gleich 14 Werbenetzwerke zu finden waren.

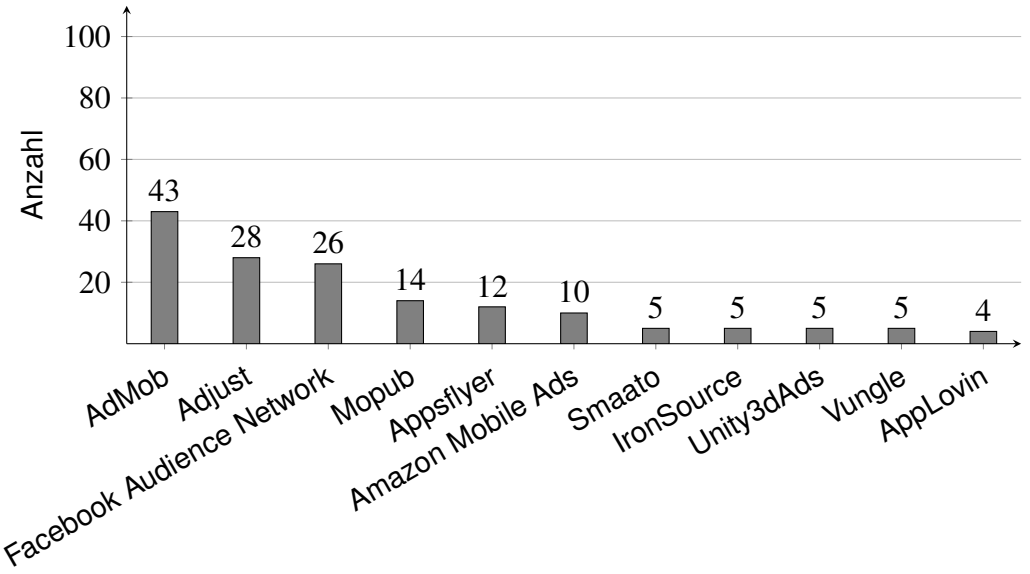


Abbildung 4.1: Werbenetzwerke in den Top 100 Apps

4.1.2 Globale Verbreitung der Werbenetzwerke

Neben den Ergebnissen aus 4.1.1 wurde eine globale Statistik [9] über die Verbreitung von Werbenetzwerken betrachtet. In dieser Statistik wird die globale Verbreitung der Werbenetzwerke, unter Berücksichtigung von Installationen auf Endgeräten, aufgelistet. Eine Übersicht über die Top 10 dieser Statistik kann in Abbildung 4.2 betrachtet werden

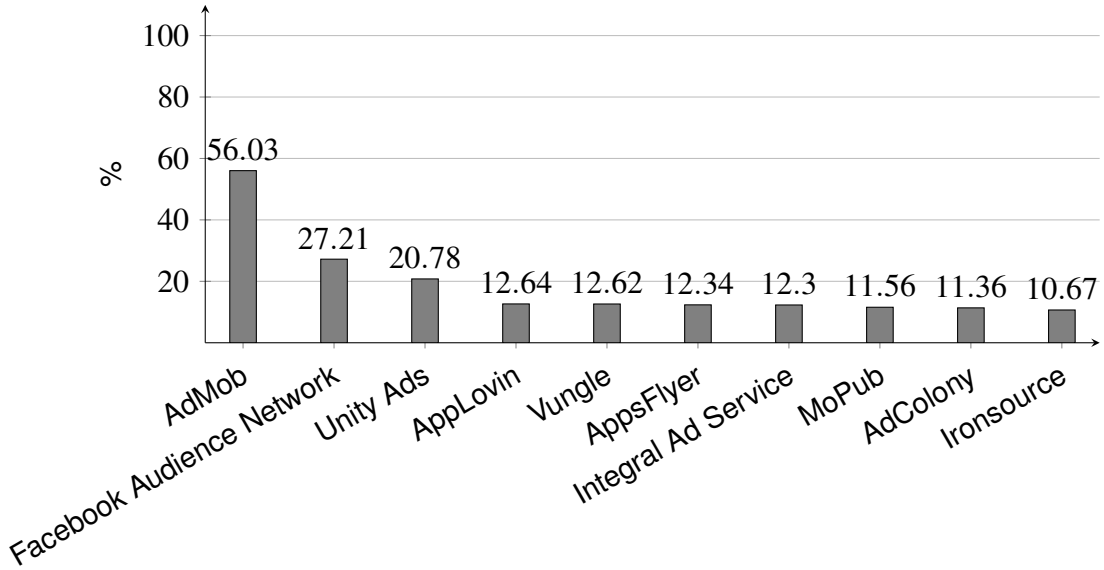


Abbildung 4.2: Verbreitung Top 10 Werbenetzwerke global

4.1.3 Auswahl

Anhand der Ergebnisse aus 4.1.1 und der globalen Verbreitung in 4.1.2 wurde, wie in Kapitel 3.2.2 beschrieben, eine Auswahl an Werbenetzwerken getroffen. Hierzu wurden die Werbenetzwerke *Admob*, *Adjust*, *AppsFlyer*, *AppLovin*, *Facebook Audience Network*, *Mopub*, *Ironsource*, *Smaato*, *Unity3dAds* und *Vungle* ausgewählt und in den weiteren Analyseschritten betrachtet.

4.2 Analyse der Werbenetzwerke

Wie in Kapitel 3.2.2 beschrieben, wurden die ausgewählten Werbenetzwerke hinsichtlich der erhobenen Daten des verwendeten Smartphones analysiert. Die Ergebnisse dieser Analyse werden nun im folgenden vorgestellt.

4.2.1 Übermittelte Daten

Zunächst einmal wurde dazu der Netzwerkverkehr von den Werbenetzwerken abgefangen, wie in Kapitel 3.2.2 beschrieben. Im Anschluss daran wurden die Daten extrahiert, welche zu den Servern der Werbenetzwerke übermittelt wurden.

Die Daten wurden in Form von Datenpaketen zum einen als **JSON-Dateien** und zum anderen **URL-Codiert** übertragen. *HTTP-POST* Nachrichten enthielten dabei Daten URL-Codiert und als JSON-Dateien, *HTTP-GET* Nachrichten nur URL-Codierte Daten.

Dabei wurde festgestellt, dass Informationen über das Gerät, die ungefähre Position des Gerätes und die ausführende Anwendung übermittelt wurden. Unter diesen Daten war jeweils ein eindeutiger Identifikator enthalten, mit dem übertragene Daten einem Gerät zugeordnet werden können. Außerdem wurden in den Datenpaketen einiger Werbenetzwerke Zeitstempel gefunden.

Insgesamt konnten die Daten den Kategorien **Identifizierung**, **Positionsinformationen**, **Geräteinformationen** sowie **Anwendungsinformationen** zugeordnet werden. Einige dieser Daten werden im Folgenden vorgestellt. Eine Vollständige Aufzählung aller übertragenen Daten, die bei der Netzwerkanalyse festgestellt werden konnten, ist im Anhang B dieser Arbeit zu finden.

Identifizierung

Bei der Analyse aller extrahierten Datenbündel wurde festgestellt, dass die Zeichenkette

```
4550e401-4cae-414d-9da7-892ecceea182
```

enthalten war. Dabei handelt es sich um die sogenannte **Android Advertising ID**, welche zur Schaltung von personalisierten Werbeanzeigen, durch die *Google Play*-Dienste, bereitgestellt wird [14]. Diese ID ist ein eindeutiger Identifikator, mit dem Daten einem Gerät eindeutig zugeordnet werden können und besteht immer aus 32 Zeichen. Es ist jedoch Möglich diese ID unter den Einstellungen des Gerätes zurückzusetzen.

Mit dieser Erkenntnis wurde nach anderen eindeutigen Identifikatoren gesucht, die neben der *Android Advertising ID* einen ähnlichen Nutzen haben. Dabei wurde festgestellt, dass das Werbenetzwerk *Appsflyer* zusätzlich die sogenannte **Android Device ID** überträgt. Bei diesem Identifikator handelt es sich um eine ID, die bei der Installation des Android-Betriebssystems generiert wird und nur durch eine Neuinstallation geändert werden kann. Selbst wenn man also die *Android Advertising ID* zurücksetzt, könnte dieses Werbenetzwerk ein Gerät anhand der *Android Device ID* erkennen.

Positionsinformationen

Weiter konnten unter den Daten Informationen beobachtet werden, die eine Eingrenzung des Aufenthaltsortes erlauben. Zu diesen Informationen gehört unter anderem die IP-Adresse, welche durch die Übermittlung eines Datenpaketes an den Server des Werbenetzwerkes übertragen wird. Weiterhin wurden Informationen zu Land und der eingestellten Zeitzone entdeckt. Bei dem Werbenetzwerk *MoPub* konnte zudem beobachtet werden, dass innerhalb des Datenpaketes die nächstgelegene Stadt angegeben war. Durch diese Informationen wird einem Werbenetzwerk ermöglicht, geografisch orientierte Werbung zu Schalten.

Geräteinformationen

Ein Großteil der übertragenen Daten waren Informationen über das Gerät. Darunter gerätespezifische Informationen wie verwendetes Gerätemodell (z.B. MiA2), der Gerätehersteller (z.B. Xiaomi, Samsung etc.), das Betriebssystem und Version des Betriebssystems, Systemsprache sowie Bildschirmmaße. Die Übertragung von CPU-Architektur und CPU-Anzahl konnten ebenfalls beobachtet werden.

Neben gerätespezifischen Informationen konnte die Übertragung von Statusinformationen festgestellt werden. Darunter eingestellte Lautstärke, Batteriestatus (Aufladen,

Entladen), Batteriestand, verfügbarer Gerätespeicher sowie verfügbare Gerätesensoren samt der Bewegungsparameter.

Unter den übertragenen Daten zu den Servern des Werbenetzwerkes *Vungle* wurden zudem Daten gefunden, welche auf die Konfiguration des Android Systems, verfügbarer Hardware sowie Eigenschaften der Mobilfunkverbindung schließen lassen.

Anwendungsinformationen

Weiterhin wurden Informationen über die App übertragen, welche auf dem Smartphone ausgeführt wurde. Dazu gehören unter anderem Name oder Prozess-Name der App und die Version. Außerdem konnten speziell unter den übertragenen Daten an die Server der Werbenetzwerke *AppLovin* und *Appsflyer* Zeitstempel für den Zeitpunkt der App-Installation entdeckt werden, bei *Appsflyer* zudem ein Zeitstempel über den Zeitpunkt der ersten Ausführung.

4.2.2 Angaben der Werbenetzwerke

In einem weiteren Schritt wurden die Werbenetzwerke dahingehend analysiert, welche Daten sie nach eigenen Angaben erheben, an die Server Übertragen und wie diese weiterverarbeitet werden. Dazu wurden die entsprechenden Datenschutzerklärungen, wie in 3.2.2 erklärt, betrachtet.

Datenerhebung

Bei dem Dienst *AdMob* von Google wurde festgestellt, dass es bezüglich erhobener Daten aus der bereitgestellten Softwarebibliothek keine generelle Datenschutzerklärung gibt. Google verweist hierbei auf die Entwickler, dass diese bei Einbindung der Softwarebibliothek eine eigene Datenschutzerklärung zu erstellen haben.

Bei allen anderen Werbenetzwerken konnte eine entsprechende Datenschutzerklärung gefunden werden, welche die Datenerhebung der bereitgestellten Softwarebibliotheken beschreibt. Dabei führen einige Werbedienstleister die Datenerhebung genauer und andere weniger genau an.

Die Werbenetzwerke *Adjust* und *Vungle* machen nur wenige Angaben über deren Datenerhebung. Darunter erwähnen sie lediglich die Erhebung der IP-Adresse und *Adjust* zusätzlich das Übertragen des Zeitpunktes der Installation und des ersten Ausführens einer App. *Adjust* gibt weiterhin an, die **Android Advertising ID** zu nutzen. *Vungle* gibt lediglich die Nutzung einiger eindeutiger Identifikatoren an, sowie Informationen über das Gerät zu erheben. Beide Dienstleister machen jedoch keinerlei Angaben darüber, welche Daten sie genau von dem Gerät erheben.

Andere Werbenetzwerke machen dabei teils genauere Angaben darüber, welche Informationen sie über ein Endgerät erheben und welche Identifikatoren sie zur Zuordnung der Daten verwenden. Darüber hinaus geben alle Werbenetzwerke an, die IP-Adresse zur ungefähren Positionsbestimmung zu verwenden und einige darüber Hinaus das Land sowie die Zeitzone zu ermitteln. Die Werbenetzwerke *Facebook Audience Network*, *MoPub* und *Smaato* gaben sogar die Erhebung und Übertragung von GPS-Koordinaten des Gerätes an.

Datenweiterverarbeitung

Aus den Datenschutzerklärungen konnten nicht nur Informationen dazu gewonnen werden, welche Angaben die Werbenetzwerke bezüglich der Datenerhebung machen, es werden auch Angaben zur Verarbeitung der Daten gemacht.

Die Daten werden laut den Werbenetzwerken hauptsächlich genutzt um die Interessen eines Nutzers zu Ermitteln und Identitäten über mehrere Geräte hinweg aufzulösen. Über die somit gewonnenen Informationen können die Werbenetzwerke personalisierte Werbung schalten. Dazu werden die Daten teilweise an Werbepartner und andere Werbenetzwerke weitergegeben. Außerdem wird in die Schaltung von Werbeanzeigen der ungefähre Standort eines Benutzers mit einbezogen, um Geotargeting zu ermöglichen.

Weiterhin verwenden die Werbenetzwerke erhobene Daten zur Analyse von Marketingkampagnen. Mithilfe der Daten werden Berichte erstellt in denen angegeben ist, wann und wie Werbeanzeigen geschaltet und ob diese angeklickt wurden. Zu diesem Zweck geben die Werbenetzwerke die Daten teilweise an Datenpartner weiter.

Laut den Angaben der Werbenetzwerke wird mithilfe der Daten zudem *Frequency Capping* betrieben, womit eine zu häufige Schaltung der selben Anzeige oder des selben Anzeigentyps verhindert wird. Außerdem benutzen sie die Daten um Betrug durch ungültige Klicks auf Werbeanzeigen erkennen und vorbeugen zu können. Dazu werden die Daten an Partner für Betrugsprävention weitergegeben.

Neben der Betrugserkennung optimieren, schützen und pflegen die Werbenetzwerke ihre Dienste durch eine Prüfung und wissenschaftliche Untersuchung der Daten. Somit können die Werbenetzwerke ihre Dienstleistungen verbessern und vermarkten. Außerdem können Daten über einen Nutzer im Falle eines Ermittlungsverfahrens an staatliche Behörden weitergegeben werden.

4.2.3 Auswertung zur Datenerhebung

Um die Datenerhebung der Werbenetzwerke beurteilen zu können, wurden deren Angaben mithilfe der Ergebnisse aus 4.2.1 überprüft. Dies war mithilfe der erstellten Übersichtstabelle aus 3.2.2 möglich, durch welche die Ergebnisse strukturiert und übersichtlich zusammengefasst werden konnten.

Grundsätzlich konnte dadurch herausgefunden werden, dass die Angaben der Werbenetzwerke in den Datenschutzbestimmungen nur teilweise mit den wirklich erhobenen Daten übereinstimmen. Zum einen wurden Daten zu den Servern der Werbenetzwerke übermittelt, welche nicht in den Datenschutzbestimmungen angegeben waren. Zum anderen konnte die Übermittlung einiger Daten nicht festgestellt werden, obwohl diese in den Datenschutzbestimmungen angegeben sind.

Nicht angegebene aber übermittelte Daten

Bezüglich der Werbenetzwerke *Adjust* und *Vungle* konnte festgestellt werden, dass diese eine große Anzahl an Daten erheben und dies nicht in den Datenschutzbestimmungen angeben. *Vungle* gibt lediglich die Erhebung einiger Informationen von dem Gerät an, ohne nähere Angaben zu den erhobenen Daten zu machen. Die Informationen, welche ohne eine Angabe seitens der Werbenetzwerke übertragen wurden, sind in Tabelle 4.1 aufgeführt. Darunter sind neben Daten über das Gerät, wie Gerätemodell und Betriebssystemversion, auch Informationen über den aktuellen Status der Batterie (Aufladen, Entladen) und des aktuellen Batteriestandes enthalten.

Neben den Geräteinformationen in Tabelle 4.1 wurde bei dem Werbenetzwerk *Vungle* festgestellt, dass zusätzlich Einstellungsinformationen des Gerätes übertragen werden. Darunter Informationen ob der Energiesparmodus des Gerätes eingeschaltet ist, ob die Installation von Anwendungen aus unbekannten Quellen aktiviert ist, ob der Sound eingeschaltet ist, ob eine SD-Karte verfügbar und sogar ob die Datenverbindung des Gerätes gedrosselt ist.

Neben *Adjust* und *Vungle* wurde bei allen anderen Werbenetzwerken ähnliche Unstimmigkeiten mit den Datenschutzerklärungen gefunden. Beispielsweise übermittelten die Werbenetzwerke *Ironsource*, *Facebook Audience Network* und *Unity Ads* ebenfalls den Batteriestand des Gerätes, sowie Daten des Bildschirms ohne dies anzugeben. Das Werbenetzwerk *Appsflyer* gibt in den Datenschutzbestimmungen, an Gerätebewegungsparameter zu erfassen. Dies konnte auch festgestellt werden, jedoch wurden darüber hinaus die verfügbaren Gerätesensoren übertragen ohne dies explizit anzugeben.

Eine vollständige Übersicht von nicht angegebenen aber übermittelten Daten ist im Anhang B dieser Arbeit zu finden.

Tabelle 4.1: Übermittlung nicht angegebener Daten des Gerätes

Geräteinformationen	Adjust	Vungle
Hersteller	×	×
Modell	×	×
Gerätetyp	×	×
Build-Nummer	×	×
Gerät Codename	-	×
Betriebssystem	×	×
Betriebssystemversion	×	×
API-Level	×	×
Bildschirmdaten	×	×
Lautstärke-Stufe	-	×
Batteriestatus	-	×
Batteriestand	-	×
Gerätespeicher	-	×
CPU Architektur	×	-
Netzwerkverbindungstyp	×	×
Mobilfunkanbieter	-	×
User Agent	×	×
Sprache	×	×
Land	×	×
Zeitzone	-	×

Angegebene aber nicht übermittelte Daten

Weiterhin wurde die Erhebung von Daten angegeben, obwohl deren Übermittlung nicht durch die getätigte Analyse festgestellt werden konnte.

Dies kann in einigen Fällen auf Ungenauigkeiten bei der Angabe der erhobenen Informationen seitens der Werbenetzwerke zurückgeführt werden. Dabei werden teilweise Angaben zu der Erhebung von Daten gemacht, aber letztendlich nicht übermittelt. Es werden Daten übertragen, welche nicht angegeben sind aber aus denen diese Informationen abgeleitet werden können.

Ein Beispiel hierfür ist das Betriebssystem und dessen Version, welches aus dem API-Level abgeleitet werden kann. Das API-Level 28 steht dabei beispielsweise für das Betriebssystem Android in der Version 9. Ein anderes Beispiel ist die Sprache, welche aus der *Locale* abgeleitet werden kann. Bei der *Locale* handelt es sich um Einstellungsparameter für Computerprogramme, in denen die Sprache und das Gebiet enthalten ist, wie beispielsweise `de_DE` für Deutschland.

In anderen Fällen kann dieser Umstand darauf zurückgeführt werden, dass die ausgeführten Anwendungen, in denen die Softwarebibliothek eines Werbenetzwerkes enthalten war, nicht die nötigen Berechtigungen hatten, um bestimmte Daten aus dem

Android-System zu bekommen. Wenn eine solche Berechtigung also nicht durch die App angefragt wird, kann auch die Softwarebibliothek nicht darauf zurückgreifen.

Ein Beispiel hierfür sind GPS-Koordinaten, welche das Android-System nur über die Berechtigung `android.permission.ACCESS_FINE_LOCATION` freigibt. Ein anderes Beispiel hierfür ist die Angabe die IMEI des Gerätes zu erheben, wofür jedoch die Berechtigung `android.permission.READ_PHONE_STATE` nötig ist. Über die IMEI kann ein Endgerät eindeutig identifiziert werden.

Ein Sonderfall war die Angabe den Mobilfunkanbieter zu erheben, wofür ebenfalls die Berechtigung `android.permission.READ_PHONE_STATE` notwendig ist. Hierfür war zwar in den Datenpaketen einiger Werbenetzwerke das entsprechende Feld enthalten, dies war jedoch leer. Dadurch konnte nachgewiesen werden, dass diese Information übermittelt wird, falls die Berechtigung der ausführenden App erteilt wurde.

4.3 Statische APK Analyse

Im letzten Schritt der Analyse wurden die Werbenetzwerke dahingehend untersucht, mithilfe welcher Methoden die Informationen aus dem Android-System angefordert werden. Dazu wurden die Softwarebibliotheken zunächst einmal identifiziert (3.2.1) und anschließend mithilfe von Methoden des Reverse Engineering (3.2.3) betrachtet. Dadurch konnten unter anderem API-Aufrufe identifiziert werden, welche für die Datenerhebung innerhalb des Programmcodes verantwortlich sind. Die Ergebnisse dieser Analyse werden im folgenden vorgestellt.

4.3.1 Paketnamen der Softwarebibliotheken

Bei Analyse der internen Struktur von Android-Apps konnte festgestellt werden, dass die Softwarebibliotheken der Werbenetzwerke in dem Paket `com` integriert sind und durch deren Namen eindeutig identifiziert werden können. Dabei ist beispielsweise die Bibliothek *Unity Ads* in dem Paket `com.unity3d.ads` zu finden. Eine Auflistung aller Paketnamen der analysierten Softwarebibliotheken kann in Tabelle 4.2 betrachtet werden.

Tabelle 4.2: Paketnamen der analysierten Softwarebibliotheken

Werbenetzwerk	**Paket**
Admob	`com.google.android.gms.ads`
Adjust	`com.adjust.sdk`
AppLovin	`com.applovin`
Appsflyer	`com.appsflyer`
Facebook Audience Network	`com.facebook.ads`
Mopub	`com.mopub`
Ironsource	`com.ironsource`
Smaato	`com.smaato.soma`
Unity Ads	`com.unity3d.ads`
Vungle	`com.vungle.warren`

4.3.2 Analyse der Softwarebibliotheken

Innerhalb der Softwarebibliotheken wurde nach dem Programmcode gesucht, welcher die Informationen aus dem Android System anfordert. Dabei wurde festgestellt, dass einige Werbenetzwerke den Programmcode verschleiern und andere nicht. In Tabelle 4.3 ist eine Übersicht dieser Ergebnisse dargestellt.

Verschleierter Programmcode

Der für die Datenerhebung relevante Programmcode wurde von den Werbenetzwerken *Admob*, *Applovin*, *Facebook Audience Network* und *Ironsource* vollständig verschlei-

Tabelle 4.3: Übersicht der Verschleierung des Programmcodes

Werbenetzwerk	Ja	Nein	Teilweise
Admob	×	-	-
Adjust	-	-	-
AppLovin	×	-	-
Appsflyer	-	-	×
Facebook Audience Network	×	-	-
Mopub	-	-	×
Ironsource	×	-	-
Smaato	×	×	-
Unity Ads	-	×	-
Vungle	-	×	-

ert. Dabei wurden die Namen der Pakete, Klassen, Funktionen und Variablen durch Buchstaben des Alphabetes ausgetauscht, wodurch eine Analyse erschwert wurde. Die Softwarebibliothek des Werbenetzwerkes *Smaato* konnte in verschleierter und unverschleierter Form vorgefunden werden.

Teilweise verschleierter Programmcode

Bei den Werbenetzwerken *Appsflyer* und *Mopub* konnte innerhalb des relevanten Programmcodes der Softwarebibliotheken eine Teilverschleierung festgestellt werden. Dabei waren einige Funktionsnamen und Variablennamen durch Buchstaben aus dem Alphabet ersetzt worden und andere nicht. Die Software *jadx* scheiterte zudem bei dem Werbenetzwerk *Appsflyer* daran, große Teile des Programmcodes sichtbar zu machen. Einige Teile des Maschinencodes konnten hier offensichtlich nicht dekompiliert werden. Dadurch wurde angenommen, dass hier zusätzliche Mittel zur Teilverschleierung, seitens des Werbenetzwerkes, getroffen wurden.

Nicht verschleierter Programmcode

Bei den Softwarebibliotheken der Werbenetzwerke *Unity Ads* und *Vungle* waren die relevanten Programmausschnitte nicht verschleiert. Ebenfalls konnte, wie bereits erwähnt, eine nicht verschleierte Implementation von *Smaato* gefunden werden. Dadurch war es bei diesen Werbenetzwerken möglich die Pfade zu den Klassen wiederherzustellen, welche für die Datenerhebung verantwortlich sind. Bei *Mopub* war dies durch die Teilverschleierung ebenfalls möglich. Die Pfade zu diesen Klassen sind in Tabelle 4.4 aufgeführt.

Mithilfe dieser nicht verschleierten Klassen konnte der Programmcode gut nachvollzogen werden. Dadurch war es möglich, API-Aufrufe und Funktionen zu identifizieren, die Informationen aus dem Android-System anfordern. Weiterhin war es dadurch mög-

Tabelle 4.4: Verantwortliche Klassen für die Datenerhebung auf dem Gerät

Werbenetzwerk	Klasse
Mopub	`com.mopub.common.ClientMetadata`
Smaato	`com.smaato.soma.internal.requests.settings.DeviceDataCollector`
Unity Ads	`com.unity3d.services.core.device.Device`
Vungle	`com.vungle.warren.network.VungleApiClient`

lich, ähnliche Implementationen in den verschleierten Softwarebibliotheken der anderen Werbenetzwerke zu finden.

4.3.3 Informationsbeschaffung aus dem Android-System

Durch die zuvor getätigte Analyse des Programmcodes der Softwarebibliotheken konnten Aufrufe an das Android-System aufgedeckt werden, welche für die Datenerhebung verantwortlich sind. Dabei wurde festgestellt, dass Informationen zum einen direkt über API-Aufrufe angefordert werden und zum anderen für bestimmte Daten die Berechnung aus mehreren Informationsquellen erfolgt.

API-Aufrufe

Für eine Reihe der übermittelten Daten konnten innerhalb der Softwarebibliotheken zugehörige API-Aufrufe identifiziert werden, welche in Tabelle 4.5 aufgelistet sind.

Auf die Android Advertising ID kann zugegriffen werden, wenn der Entwickler eine entsprechende Softwarebibliothek von Google in die App implementiert hat. Ist diese in der App enthalten, kann der Identifikator über eine Funktion `getId()` ermittelt werden. Der komplette API-Aufruf kann in Tabelle 4.5 gefunden werden. Ein weiterer Identifikator kann, wie bereits besprochen, die Android Device ID sein. Diese ID wird letztendlich über das Feld `ANDROID_ID` der Klasse `android.provider.Settings.Secure` angesteuert.

Allgemeine Informationen zu dem jeweiligen Android Gerät werden aus der Klasse `android.os.Build` aufgerufen. In dieser Klasse werden Eigenschaften des Gerätes aus Systemdateien in Feldern abgespeichert. Der Name des Herstellers steht dabei beispielsweise in dem Feld `MANUFACTURER`. Informationen über die Version von Android befinden sich in der Unterklasse `VERSION`, welche in `android.os.Build` enthalten ist. Die Version selbst steht dabei in dem Feld `RELEASE`.

Informationen über den Bildschirm des Geräte werden aus der Klasse `Displaymetrics` (`android.util.Displaymetrics`) ermittelt. Hier kann über entsprechende API-Aufrufe

aus Tabelle 4.5 auf Bildschirmhöhe, Bildschirmbreite und Pixeldichte zugegriffen werden.

Das Land sowie die Sprache wird über die Klasse `java.util.Locale`, durch den Aufruf der Funktionen `getCountry()` und `getLanguage()`, ermittelt. Die Zeitzone wiederum über die Funktion `getDisplayName()` der Klasse `java.util.Timezone`.

Der Prozessname der aktuell ausgeführten App wird über den Aufruf der Funktion `getPackageName()` aus der Klasse `android.content.Context` ermittelt. Diese Klasse stellt eine Schnittstelle zu globalen Informationen einer Anwendungsumgebung bei Android dar. Die Version sowie der Installationszeitpunkt einer App werden beispielsweise über die Klasse `android.content.pm.PackageInfo` abgerufen.

Tabelle 4.5: API-Aufrufe

Information	API-Aufruf
Android Advertising ID	`com.google.android.gms.ads.identifier.AdvertisingIdClient.Info.getId()`
Android Device ID	`android.provider.Settings.Secure.ANDROID_ID`
Gerätehersteller	`android.os.Build.MANUFACTURER`
Gerätemodell	`android.os.Build.MODEL`
Hardwarename	`android.os.Build.DISPLAY`
Prozessoranzahl	`java.lang.Runtime.availableProcessors()`
Android-Version	`android.os.Build.VERSION.RELEASE`
API-Level	`android.os.Build.VERSION.SDK_INT`
Bildschirmbreite	`android.util.DisplayMetrics.widthPixels`
Bildschirmhöhe	`android.util.DisplayMetrics.heightPixels`
Pixeldichte	`android.util.DisplayMetrics.densityDpi`
Lautstärke	`android.media.AudioManager.getStreamVolume()`
Netzwerkverbindungstyp	`android.telephony.TelephonyManager.getNetworkType()`
Netzwerkverbindung limitiert?	`android.net.ConnectivityManager.isActiveNetworkMetered()`
Mobilfunkanbieter	`android.telephony.TelephonyManager.getNetworkOperator()`
Systemsprache	`java.util.Locale.getLanguage()`
Land	`java.util.Locale.getCountry()`
Zeitzone	`java.util.TimeZone.getDisplayName()`
App-Paket	`android.content.Context.getPackageName()`
App-Version	`android.content.pm.PackageInfo.versionName`
Installationszeitpunkt	`android.content.pm.PackageInfo.firstInstallTime`

Einbezug mehrerer Informationsquellen

Bei der Analyse von relevantem Quellcode in den Softwarebibliotheken konnte festgestellt werden, dass nicht für alle der übertragenen Informationen entsprechende API-Aufrufe existieren.

So kann beispielsweise der Batteriestand und Batteriestatus nicht direkt durch einen API-Aufruf ermittelt werden, sondern über den Broadcast Reciever mit der Bezeichnung `ACTION_BATTERY_CHANGED`. Auch kann ein vorhandener Root nicht über einen API-Aufruf überprüft werden. Dazu ist wird ein bestimmter Teil des Dateisystems von Android auf die Zeichenkette `su` durchsucht. Für diese Fälle wurden bestimmte Algorithmen seitens der Werbenetzwerke in den Softwarebibliotheken implementiert.

5 Fazit und Ausblick

Durch die im letzten Kapitel vorgestellten Ergebnisse der Analysen konnte geklärt werden, welche Daten die Werbenetzwerke an ihre Server übermitteln. Weiterhin wurde gezeigt, wie die Softwarebibliotheken Daten von dem Smartphone erheben. Jedoch wurde noch nicht geklärt, welchen Einfluss die erhobenen Daten auf die Privatsphäre eines Benutzers haben.

Diesbezüglich kann man sagen, dass die Übermittlung von personenbezogenen Daten, im Gegensatz zu früheren Forschungsergebnissen, nicht festgestellt werden konnte. Die bereits existierenden Forschungsergebnisse liegen schon ein paar Jahre zurück und zeigten auf, dass private Informationen wie Standortkoordinaten, Anruflisten, Telefonnummer, Browser-Lesezeichen und eine Liste installierter Anwendungen übertragen wurden. Jedoch konnte neben den vorgestellten Ergebnissen beobachtet werden, dass auch in der heutigen Zeit noch Programmcode über die Anfragen der Softwarebibliotheken aus dem Internet geladen wird. Dies stellt nach wie vor ein Sicherheitsrisiko dar und ist für weitere Forschungen zu diesem Thema von Interesse.

Im Hinblick auf diese Forschung konnte lediglich festgestellt werden, dass Identifikatoren zusammen mit Daten über das Gerät, den ungefähren Standort und Informationen zur ausgeführten Anwendung übertragen wurden. Aus diesen Informationen kann nicht direkt eine genaue Identität abgeleitet werden, jedoch kann ein Gerät und dessen übermittelte Daten, über mehrere Anwendungen hinweg, eindeutig identifiziert werden. Weiterhin kann festgehalten werden, dass nahezu alle diese Daten ohne oder über normale Berechtigungen, welche keine Zustimmung erfordern, zugänglich sind.

Mithilfe der Ergebnisse konnte ein Überblick zur Datenerhebung von Werbenetzwerken geschaffen und gezeigt werden, welche Vorgänge hinter dem Anzeigen von Werbung stecken. Dazu wurde die Meinung gebildet, dass Verbesserungsbedarf seitens der Werbenetzwerke, im Hinblick auf die Angabe übermittelter Daten, besteht. Hierzu wurde gezeigt, dass Werbenetzwerke mehr Daten übermitteln als sie angeben und sich deren Angaben grundsätzlich nicht mit den übermittelten Daten decken. Zu den erhobenen Daten sollten genauere Angaben in den Datenschutzerklärungen gemacht werden, um diesbezüglich mehr Transparenz zu schaffen.

Neben den beliebten Apps aus dem *Google Play Store*, deren enthaltene Werbenetzwerke in dieser Arbeit betrachtet wurden, könnten in einer weiteren Forschung explizit schlecht bewertete Apps in dieser Hinsicht betrachtet werden. Im Bezug auf diese Apps wäre es ebenfalls von Interesse, den Datenverkehr enthaltener Werbenetzwerke zu betrachten, da hier meist mehr Berechtigungen von dem Android-System abgefragt werden als nötig. Werbenetzwerke können von diesen Berechtigungen Gebrauch machen und sensible Daten übermitteln.

Die Ergebnisse dieser Arbeit können darüber hinaus für zukünftige Arbeiten verwendet werden, um beispielsweise selbst ein System zur Klassifizierung von Werbenetzwerken in Android Apps zu entwickeln. Hierzu wäre die Entwicklung eines Kommandozeilentools sinnvoll, welches automatisch die enthaltenen Werbenetzwerke in einer größeren Anzahl an Apps klassifiziert. Das Nutzen der Anwendung *Appbrain Ad Detector* hat sich für diesen Vorgang etwas umständlich gestaltet, da jede App einzeln betrachtet werden musste.

Weiterhin könnte, mithilfe der gewonnenen Kenntnisse über die Informationsbeschaffung der Werbenetzwerke aus dem Android-System, eine Software zur dynamischen Analyse von Apps entwickelt werden. Mittels dynamischer Instrumentierung von Prozessen könnten hier die relevanten Funktionsaufrufe im Android-System überwacht werden. Hierzu wurden bereits geeignete Methoden mittels *FRIDA*, in dem Bericht über das Praktikum bei der *Zentralen Stelle für Informationstechnik im Sicherheitsbereich*, vorgestellt.

Literaturverzeichnis

[1] Android: *Android Debug Bridge (ADB)*. URL: https://developer.android.com/studio/command-line/adb. Zugriff am 27. Jul. 2019.

[2] Android: *Android Security Overview*. URL: https://source.android.com/security. Zugriff am 14. Jul. 2019.

[3] Android: *Android-Stack*. URL: https://developer.android.com/guide/platform/images/android-stack_2x.png. Zugriff am 15. Jul. 2019.

[4] Android: *Application Fundamentals*. URL: https://developer.android.com/guide/components/fundamentals. Zugriff am 15. Jul. 2019.

[5] Android: *Application Sandbox*. URL: https://source.android.com/security/app-sandbox. Zugriff am 15. Jul. 2019.

[6] Android: *Permission Protection Levels*. URL: https://developer.android.com/guide/topics/permissions/overview#normal-dangerous. Zugriff am 15. Jul. 2019.

[7] Android: *Permissions Overview*. URL: https://developer.android.com/guide/topics/permissions/overview. Zugriff am 15. Jul. 2019.

[8] Android: *Platform Architecture*. URL: https://developer.android.com/guide/platform. Zugriff am 15. Jul. 2019.

[9] AppBrain: *Android Ad Network statistics and market share*. URL: https://www.appbrain.com/stats/libraries/ad-networks. Zugriff am 9. Aug. 2019.

[10] AppBrain: *Free vs. paid Android Apps*. URL: https://www.appbrain.com/stats/free-and-paid-android-applications. Zugriff am 15. Jul. 2019.

[11] Business of Apps: *Top Mobile Ad Networks 2018*. Hrsg. von Artyom Dogtiev. URL: https://www.businessofapps.com/guide/top-mobile-ad-networks/. Zugriff am 9. Jul. 2019.

[12] A. Becker und M. Pant: *Programmieren für Smartphones und Tablets*. dpunkt.verlag, 2015. ISBN: 9783864916625.

[13] Theodore Book, Adam Pridgen und Dan S. Wallach: *Longitudinal Analysis of Android Ad Library Permissions*. In: *CoRR* abs/1303.0857 (2013). URL: http://arxiv.org/abs/1303.0857.

[14] Google: *Werbe-ID*. url: https://support.google.com/googleplay/android-developer/answer/6048248?hl=de. Zugriff am 13. Aug. 2019.

[15] Jayaprakash Govindaraj, Robin Verma und Gaurav Gupta: *Analyzing Mobile Device Ads to Identify Users*. In: *12th IFIP International Conference on Digital Forensics (DF)*. Hrsg. von Gilbert Peterson und Sujeet Shenoi. Bd. AICT-484. Advances in Digital Forensics XII. Part 2: MOBILE DE- VICE FORENSICS. New Delhi, India: Springer International Publishing. Jan. 2016, S. 107–126. URL: https://hal.inria.fr/hal-01758680.

[16] Michael C. Grace, Wu Zhou, Xuxian Jiang und Ahmad-Reza Sadeghi: *Unsafe Exposure Analysis of Mobile In-app Advertisements*. In: *Proceedingsof the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WISEC '12. Tucson, Arizona, USA: ACM, 2012, S. 101–112. ISBN: 978-1-4503-1265-3. URL: http://doi.acm.org/10.1145/2185448.

[17] Daehyeok Kim, Sooel Son und Vitaly Shmatikov: *What Mobile Ads Know About Mobile Users*. In: NDSS 2016. 2016.

[18] NVISIO LABS: *Using a custom root CA with Burp for inspecting Android N traffic*. Hrsg. von Jeroen Beckers. 31. Jan. 2018. URL: https://blog.nviso.be/2018/01/31/using-a-custom-root-ca-with-burp-for-inspecting-android-n-traffic/. Zugriff am 28. Jul. 2019.

[19] Online Marketing: *Monetarisierung*. 2019. URL: https://onlinemarketing.de/lexikon/definition-monetarisierung.

[20] Annamalai Narayanan, Lihui Chen und Chee Keong Chan: *AdDetect: Automated detection of Android ad libraries using semantic analysis*. In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, April 21-24, 2014*. 2014, S. 1–6. URL: https://doi.org/10.1109/ISSNIP.2014.6827639.

[21] P. Schnabel: *Netzwerktechnik-Fibel: Grundlagen, Übertragungssysteme, TCP/IP, Dienste, Sicherheit*. Elektronik-Kompendium, 2016. ISBN: 9783981530247.

[22] Statistica: *Marktanteile von Android und iOS am Absatz von Smartphones in Deutschland von Januar 2012 bis März 2019*. Hrsg. von F. Tenzer. 6. Mai 2019. URL: https://de.statista.com/statistik/daten/studie/256790/umfrage/marktanteile-von-android-und-ios-am-smartphone-absatz-in-deutschland/. Zugriff am 14. Jul. 2019.

[23] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Lee Erickson und Hao Chen: *Investigating User Privacy in Android Ad Libraries*. 2012.

[24] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Abbas Razaghpanah, Rishab Nithyanand, Mark Allman, Christian Kreibich und Phillipa Gill: *Tracking the Trackers: Towards Understanding the Mobile Advertising and Tracking Ecosystem*. In: *CoRR abs/1609.07190 (2016)*. URL: http://arxiv.org/abs/1609.07190.

[25] Sunny Wear: *Burp Suite Cookbook: Practical Recipes to Help You Master Web Penetration Testing with Burp Suite*. Packt Publishing, 2018. ISBN: 1938581164, 9781789531732.

Anhang A: Anwendungen und enthaltene Werbenetzwerke

Die im Rahmen dieser Arbeit analysierten Android Apps sind in Tabelle A.1 und A.2, anhand ihrer Platzierung im Google Play Store vom 27 Juni 2019, aufgelistet. Identifizierte Werbenetzwerke einer App sind dabei mit dem Zeichen (✓) gekennzeichnet. In Tabelle A.3 sind die Werbenetzwerke anhand ihres Vorkommens aufgelistet.

Tabelle A.1: Top 100 Apps mit enthaltenen Werbenetzwerken (1/2)

Platz:	App:	Admob	Adjust	AerServ	Amazon Mobile Ads	AppLovin	AppNext	AppNexus	Appsflyer	DU Ads	Chartboost	Facebook Audience Network	Flurry Ads	InMobi	Inneractive	Ironsource	Millenial Media	MobVista	Mopub	My Target	PubNative	Smaato	SmartAdServer	Unity Ads	Vungle	Yandex Mobile Ads
1	Joyn	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	TikTok	✓	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	TIER	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	Amazon Shopping	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	eBay Kleinanzeigen	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	Spotify	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	Youtube Music	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	Wish	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	Snapchat	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	Messenger	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	Instagram	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	Netflix	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	MP3 Music Downloader	✓	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	✓	-
14	Enlight Pixaloop	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15	S-pushTAN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16	DB Navigator	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17	OTTO - Shopping	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
18	DHL Paket	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19	Lime	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20	H&M	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
21	Deezer	✓	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
22	PayPal	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
23	McDonalds	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
24	Vova	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25	Einfaches Thermometer	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26	Google Übersetzer	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
27	wetter.com	✓	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-
28	Pinterest	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29	Sparkasse	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30	Circ	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
31	Whatsapp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
32	Google Play Spiele	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
33	Ecosia	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
34	Postbank BestSign	✓	-	-	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
35	WetterOnline	✓	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
36	Malen nach Zahlen	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	✓	-
37	Sticker Maker	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
38	FlixBus	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
39	SHEIN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
40	Google Earth	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
41	Mozilla Firefox	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
42	DuckDuckGo	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
43	Amazon Prime Video	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
44	ebay Online Shopping	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
45	Zalando	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
46	PicsArt Photo Studio	✓	-	-	✓	✓	-	-	✓	-	✓	-	-	✓	-	-	-	-	✓	-	-	-	✓	-	-	-	-
47	Wetter.de	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
48	Telegram	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
49	Booking.com	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
50	Immowelt	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabelle A.2: Top 100 Apps mit enthaltenen Werbenetzwerken (2/2)

Platz:	App:	Admob	Adjust	AerServ	Amazon Mobile Ads	AppLovin	AppNext	AppNexus	Appsflyer	DU Ads	Chartboost	Facebook Audience Network	Flurry Ads	InMobi	Inneractive	Ironsource	Millenial Media	MobVista	Mopub	My Target	PubNative	Smaato	SmartAdServer	Unity Ads	Vungle	Yandex Mobile Ads
51	Lieferando.de	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
52	mobile.de	✓	✓	-	✓	✓	-	✓	-	-	✓	✓	-	✓	✓	-	-	-	✓	-	-	✓	✓	-	-	-
53	Amazon Music	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
54	Shazam	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
55	Tinder	✓	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
56	CHECK24	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
57	AT Player	✓	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
58	Thermometer	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
59	QR Scanner & Barcode Reader	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
60	YAZIO	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
61	Samsung Smart Switch	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
62	Thermometer++	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
63	Lidl Plus	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
64	Potter.ly 3D	✓	✓	-	-	-	-	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	✓	✓	-
65	Thermometer	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
66	ImmobilienScout24	✓	✓	-	✓	-	-	-	-	-	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
67	AutoScout24	✓	✓	-	✓	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
68	Komoot	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
69	Wayfair	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
70	LOVOO	✓	✓	-	✓	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	✓	-	-	-	-
71	PAYBACK	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
72	Rossmann	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
73	QR Code Reader	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
74	Google One	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
75	AppLock	✓	-	-	-	-	-	-	✓	-	✓	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-
76	Wettervorhersage	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
77	Lidl – Prospekte	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
78	ARCore	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
79	S-ID-Check	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
80	Google Pay	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
81	Adobe Acrobat Reader	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
82	Photo Editor	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-
83	HiddenMe	✓	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
84	Discord	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
85	Burger King	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
86	Android Auto	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
87	Startpage Datenschutz	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
88	Sixpack in 30 Tagen	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
89	QR & Barcode Scanner	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
90	Adobe Lightroom	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
91	ALDI TALK	-	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	✓
92	Opera mit gratis VPN	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
93	Mero Music	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
94	Drum Pad	✓	✓	-	✓	✓	-	-	-	-	✓	✓	-	-	-	✓	✓	-	✓	-	✓	✓	-	✓	✓	✓
95	Microsoft Outlook	-	✓	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
96	Badoo	✓	-	-	-	-	-	-	✓	-	✓	-	✓	-	-	✓	-	-	✓	-	-	-	-	-	✓	-
97	Breitbandmessung	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
98	Tellonym	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
99	YouCut	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-
100	Flora Incognita	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabelle A.3: Anzahl der Werbenetzwerke in Apps

Werbenetzwerk:	**Anzahl:**
Admob	43
Adjust	28
Facebook Audience Network	26
Mopub	14
Appsflyer	13
Amazon Mobile Ads	10
Ironsource	5
Smaato	5
Unity Ads	5
Vungle	5
AppLovin	4
InMobi	3
My Target	3
DU Ads	2
SmartAdServer	2
Yandex Mobile Ads	2
AerServ	1
AppNext	1
AppNexus	1
Chartboost	1
Flurry Ads	1
Inneractive	1
MobVista	1
PubNative	1

Anhang B: Datenerhebung der Werbenetzwerke

Im Bezug auf die Analyseergebnisse, von übermittelten Daten und Angaben der Werbenetzwerke, wurde eine Übersichtstabelle erstellt. Diese Tabelle ist in Form der Anhänge B.1 und B.2 zu finden. Dabei sind die übertragenen und durch die Werbenetzwerke angegebenen Daten mit einem (✓) gekennzeichnet. Wurden Daten angegeben, deren Übertragung nicht festgestellt werden konnte, sind diese mit einem (?) gekennzeichnet. Daten die letztendlich ohne eine Angabe der Werbenetzwerke übertragen wurden sind durch ein (✕) markiert.

Tabelle B.1: Analyseergebnisse der Datenerhebung (1/2)

	Information:	Adjust	AppLovin	Appsflyer	Facebook Audience Network	Ironsource	Mopub	Smaato	Unity Ads	Vungle
Identifizierung	Android Advertising ID	✓	✓	✓	✓	✓	✓	✓	✓	×
	Android Device ID	-	-	✓	-	-	-	-	-	-
	MAC-Adresse	-	-	-	-	-	-	-	?	-
	IMEI	-	-	-	-	-	-	-	?	-
	IMSI	-	-	-	-	-	-	?	-	-
Gerät	Hersteller	×	✓	×	×	✓	✓	×	✓	×
	Modell	×	✓	✓	×	×	✓	×	✓	×
	Gerätetyp	×	-	✓	-	-	?	✓	-	×
	Hardwarename (Build-Nummer)	×	-	×	-	-	-	-	-	×
	Benutzername	-	×	×	-	-	×	-	-	×
	Betriebssystem	×	✓	?	✓	✓	✓	✓	✓	×
	Betriebssystemversion	×	×	?	✓	✓	×	×	✓	×
	API-Level	×	×	×	-	×	-	-	×	×
	Bildschirmhöhe	×	-	×	×	×	×	×	×	×
	Bildschirmbreite	×	-	×	×	×	×	×	×	×
	Bildschirmgröße	×	?	×	-	×	?	-	×	-
	Bildschirmauflösung	×	-	-	-	-	-	-	✓	-
	Pixeldichte	-	×	×	-	-	-	-	×	-
	Bildschirmausrichtung	×	✓	-	-	×	-	×	×	-
	Bildschirmformat	×		-	-	-	-	-	-	-
	Lautstärke-Stufe	-	✓	-	-	×	-	-	×	×
	Batteriestatus	-	?	×	✓	?	-	-	×	×
	Batteriestand	-		-	×	×	-	-	×	×
	Verfügbarer Gerätespeicher	-	?	?	✓	✓	-	-	×	×
	RAM Gesamt	-	-	-	×	-	-	-	-	-
	RAM verfügbar		-	-	X	-	-	-	-	-
	CPU Architektur	×	-	✓	-	-	-	-	?	-
	CPU Anzahl	-	-	-	-	-	-	-	?	-
	Verfügbare Gerätesensoren	-	-	×	-	-	-	-	-	-
	Bewegungsparameter	-	-	✓	×	-	-	-	-	-
	Installierte Anwendungen	-	-	-	-	-	?	?	?	-
	Netzwerkverbindungstyp	×	?	✓	×	✓	?	×	×	×
	Mobilfunkanbieter	-	✓	✓	✓	?	?	?	×	×
	User Agent	×	×	✓	×	?	×	✓	✓	×
	Sprache	×	?	✓	?	✓	-	-	?	×

Tabelle B.2: Analyseergebnisse der Datenerhebung (2/2)

	Information:	Adjust	AppLovin	Appsflyer	Facebook Audience Network	Ironsource	Mopub	Smaato	Unity Ads	Vungle
Standort	IP-Adresse	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Land	✗	✓	✗	-	✗	✗	-	-	-
	Zeitzone	-	?	?	?	?	✓	-	✗	✗
	Locale (z.B. de_DE)	-	✗	-	✗	-	-	✗	✗	✗
	Stadt	-	-	-	-	-	✗	-	-	-
	GPS-Koordinaten	-	-	-	?	-	?	?	-	-
Anwendung	Name	-	✓	?	✓	?	✓	-	?	-
	Paket	✗	✗	✗	✗	✓	✗	✗	✗	✗
	Version	✗	✓	✗	✗	✗	✗	-	✗	✗
	Installationszeitpunkt	?	✗	✗	-	-	-	-	-	-
	Zeitpunkt der ersten Ausführung	?	-	✗	-	-	-	-	-	-
	Benutzungsdauer	✗	-	?	-	-	?	-	-	-
	Ausführung im Emulator?	-	-	-	✗	-	-	-	-	✗
	APK-Größe	-	-	-	✗	-	-	-	-	-
	Werbe-SDK-Version	-	✗	-	✗	✓	-	-	-	-
	Typ der Werbeanzeige	-	-	?	✗		?	-	-	-
Sonstiges	Unbekannte Quellen aktiviert?	-	-	-	-	-	-	-	-	✗
	Personalisierte Werbung aktiviert?	✗	-	✗	-	✗	✗	✗	✗	✗
	Google Play-Dienste verfügbar?	-	-	-	-	-	-	-	-	✗
	Energiesparmodus eingeschaltet?	-	-	-	-	-	-	-	-	✗
	SIM-Karte verfügbar?	-	✗	-	-	-	-	-	-	-
	Headset angeschlossen?	-	-	-	-	-	-	-	✗	-
	Sound eingeschaltet?	-	-	-	-	-	-	-	-	✗
	SD-Karte verfügbar?	-	-	-	-	-	-	-	-	✗
	Root vorhanden?	-	-	-	✗	✗	-	-	-	-
	Datenverbindung gedrosselt?	-	-	-	-	-	-	-	✗	✗
	Zeitstempel	✗	-	✓	✗	✓	-	?	-	✗

Anhang C: Datenübertragung im Netzwerkverkehr

Durch die Analyse des Netzwerkverkehrs konnten URLs der Werbenetzwerke identifiziert werden, zu denen die Datenübertragung stattgefunden hat. Für jedes der betrachteten Werbenetzwerke sind in C.1 die zugehörigen URLs aufgelistet. Außerdem ist für jede der URLs angegeben, über welche HTTP-Methode der Datenaustausch stattgefunden hat und ob die Daten URL-Kodiert oder im JSON-Format übertragen wurden.

Tabelle C.1: Datenübermittlung im Netzwerkverkehr

	URL(s):	HTTP-Methode	Daten
Admob	https://googleads.g.doubleclick.net/mads/gma	GET	URL-Kodiert
	https://pubads.g.doubleclick.net/gamepad/ads	GET	URL-Kodiert
	https://googleads.g.doubleclick.net/pagead/ads	GET	URL-Kodiert
Adjust	https://app.adjust.com/session	POST	URL-Kodiert
	https://app.adjust.com/sdk_click	POST	URL-Kodiert
	https://app.adjust.com/event	POST	URL-Kodiert
	https://app.adjust.com/attribution	GET	URL-Kodiert
AppLovin	https://d.applovin.com/device	POST	JSON
	https://rt.applovin.com/pix	POST	URL-Kodiert
	https://a.applovin.com/i	GET	URL-Kodiert
	https://a.applovin.com/3.0/ad	GET	URL-Kodiert
Appsflyer	https://register.appsflyer.com/api/v4/androidevent	POST	JSON
	https://events.appsflyer.com/api/v4/androidevent	POST	JSON
	https://t.appsflyer.com/api/v4/androidevent	POST	JSON
Facebook Audience Network	https://graph.facebook.com/network_ads_common	POST	URL-Kodiert
Ironsource	https://is-gateway.supersonicads.com/gateway/sdk/request/	GET	URL-Kodiert
Mopub	https://ads.mopub.com/m/open	POST	JSON
	https://ads.mopub.com/m/gdpr_sync	POST	JSON
	https://ads.mopub.com/m/ad	POST	JSON
	https://ads.mopub.com/m/attempt	POST	JSON
	https://ads.mopub.com/m/imp	POST	JSON
	https://cb.mopub.com/load	GET	URL-Kodiert
	https://mpx.mopub.com/imp	GET	URL-Kodiert
Smaato	https://sdk-android.ad.smaato.net/oapi/v6/ad	GET	URL-Kodiert
Unity Ads	https://auction.unityads.unity3d.com/v5/games/2605085/requests	POST	JSON
	https://config.uca.cloud.unity3d.com/	POST	JSON
	https://config.unityads.unity3d.com/webview/3.0.0/release/config.json	GET	URL-Kodiert
	https://publisher-config.unityads.unity3d.com/games/2605085/configuration	GET	URL-Kodiert
Vungle	https://ads.api.vungle.com/config	POST	JSON
	https://api.vungle.com/api/v5/ads	POST	JSON

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Harbach, 26. August 2019