Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Fakultät für Ingenieurwissenschaften, Bereich EuI

Fakultät für Ingenieurwissenschaften

Bachelor-Thesis im Fernstudiengang IT-Forensik

Sicherheitsanalyse von Android-Applikationen

Abschlussarbeit zur Erlangung des Grades

Bachelor of Engineering

Eingereicht am: 22. November 2019

von: Anika Jöschke
Erstgutachterin: Frau Prof. Dr.Ing. Antje Raab-Düsterhöft
Zweitgutachter: Herr Prof.Dr.Ing. habil. Andreas Ahrens

Aufgabenstellung

In dieser Bachelorthesis soll eine Strategie für eine Sicherheitsanalyse einer Android-Applikation erarbeitet werden. Dafür sollen auf der Basis von potentiellen Bedrohungen für Android-Applikationen, Empfehlungen und Anforderungen identifiziert werden, die einen Sicherheitsexperten bei der Planung für eine mögliches Vorgehen bei der Sicherheitsanalyse unterstützen, bevor diese im Unternehmen oder einer Behörde zum Einsatz kommt. Des Weiteren wird analysiert, welche Bedeutung die Sicherheitsanalyse von Android-Applikationen zum Schutz vor Malware bzw. Cyberangriffen hat. Die zu erstellende Strategie wird an einem ausgewählten Testfall prototypisch getestet.

Abstract

In this bachelor thesis, a strategy for a security analysis of an Android application will be developed. The purpose is to identify, on the basis of potential threats to Android applications, recommendations and requirements that assist a security expert in planning for a possible security analysis procedure before it is deployed in the enterprise or government agency. It also analyzes the importance of security analysis of Android applications to protect against malware and cyber attacks. The strategy to be created is tested prototypically on a selected test case.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen: Geschichte und Architektur des Android-Systems **7**
2.1 Entstehungsgeschichte des Android-Systems 7
2.2 Architektur des Android-Systems 9
2.2.1 Der Linux-Kernel 11
2.2.2 Die Hardwareabstraktionsschicht 13
2.2.3 Die nativen C/C++-Bibliotheken 13
2.2.4 Die Android-Laufzeitumgebung 14
2.2.5 Das Android-Applikations-Framework 15
2.2.6 Die Anwendungsebene- Android-Applikationen 16
2.3 Aufbau und Bestandteile von Android-Applikationen 17
2.3.1 Das Android-Manifest 19
2.3.2 Activities 20
2.3.3 Intents 21
2.3.4 Broadcast-Receiver 21
2.3.5 Content-Provider 22
2.3.6 Services 22
2.3.7 Shared Preferences 22

3 Sicherheitsgrundlagen und Schutzmaßnahmen von Android-Applikationen **25**
3.1 Definition und Klassifizierung von Malware 26
3.2 Klassifizierung von Angriffsvektoren für Android-Smartphones 31
3.2.1 Hardware-basierte Angriffe 32
3.2.2 Device-basierte Angriffe 32
3.2.3 Web-basierte Angriffe und Angriffe auf Infrastrukturkomponenten 32
3.2.4 App-basierte Angriffe 33
3.3 Das Sicherheitsmodell von Android 38
3.3.1 Sicherheitfunktionen des Linux-Kernels und SELinux 39
3.3.2 Sandboxing 40
3.3.3 Das Berechtigungskonzept-Permissions im Android-System . . . 41
3.3.4 Signierung der Applikation 43
3.3.5 Patchday 46

4 Sicherheitsanalyse von Android Applikationen **49**
4.1 Die Sicherheitsanalyse 50
4.1.1 Testarten 50
4.1.2 Analysemethoden 51
4.2 Die Bedrohungsmodellierung 53
4.2.1 Sicherheitsanforderungen für mobile Applikationen des BSI . . . 54
4.2.2 Schutzbedarfsfeststellung 56
4.2.3 Checkliste für die Sicherheitsanalyse 59
4.2.4 Ablaufplan einer Sicherheitsanalyse 60

5 Vorbereitung und Durchführung der Sicherheitsanalyse **65**
5.1 Aufbau der Arbeitsumgebung 65
5.1.1 Testsystem-und Umgebung 65

5.1.2 Android-Genymotion-Emulator 66
5.1.3 Testapplikationen 67
5.1.4 VBox – Oracle VM VirtualBox 67
5.1.5 Androl4b 67
5.2 Information Gathering: Automatisierte statische und dynamische Analyse 69
5.2.1 VirusTotal 70
5.2.2 Mobile-Security-Framwork-MobSF 74
5.3 Android-Sicherheitsanalyse- Testung der Anforderungen an die Datenspeicherung und Datenschutz 86

6 Zusammenfassung und Schlussbemerkungen **99**

Literaturverzeichnis **v**

Abbildungsverzeichnis **xvii**

Listings **xix**

Abkürzungsverzeichnis **xxi**

A Weitere Bilder und Tabellen **xxiii**

Selbstständigkeitserklärung **xxvii**

Index **xxvii**

1 Einleitung

„Machen wir uns nichts vor, auf unserem Handy ist unser Leben drauf“[Spr18]

Unsere Gesellschaft hat immer größeres Verlangen nach Mobilität und die Branche der Smart-Technologien boomt. In unserem multimobilen Zeitalter sind besonders Smartphones aufgrund ihrer reichhaltigen und vielseitigen Funktionalität im Alltag eines Menschen kaum mehr wegzudenken.
Allein von den 82,79 Millionen[Sta17] lebenden Menschen in Deutschland nutzen 57 Millionen[Bit18] ein Smartphone. Sie werden verwendet für die täglichen Telefonate, für die Bearbeitung der E-Mails, zum Shoppen, zum Surfen im Internet, zum Chatten mit Freunden, zum Fotografieren und für die tägliche Arbeit.[ADA17][Spr18]

Natürlich profitiert auch Deutschlands Wirtschaft von diesem Boom. Der Desktop PC ist längst in die Jahre gekommen und die Abwicklung der täglichen Geschäftsprozesse über Smartphone und Co., erhält immer mehr Einzug in den Unternehmens-und Arbeitsalltag. Durch den Einsatz von Smartphones können nicht nur Arbeitsprozesse effizienter gestaltet werden, sondern auch wichtige Geschäftsprozesse zeitnah abgeschlossen, sowie Kosten gespart werden. Der Arbeitnehmer kann unter anderem via VPN 24 Stunden an das Unternehmen angebunden sein und ist somit für den Arbeitgeber nahezu omnipräsent. Nach einer Erhebung von Eurostat im Jahr 2015 stellten 71%[Eur15] der Unternehmen in Deutschland, mit mindestens 10 Mitarbeitern, ihren Beschäftigten mobile Geräte mit Internetverbindung für geschäftliche Zwecke zur Verfügung.[Sch15]

Allerdings gestaltet sich der Einsatz von mobilen Geräten, wie z.B. Smartphones, im Gegensatz zum alten Desktop-PC wesentlich anspruchsvoller. Das mobile Gerät soll nicht nur zu jeder Zeit und von jedem Ort einsetzbar und mit möglichst allen nötigen Funktionen ausgestattet sein, sondern dem Nutzer neben geschäftlichen auch für private Zwecke dienen. Darüber hinaus ist die funktionale und technische Komplexität der Geräte, in einem enormen Tempo gestiegen. Millionen neuer Applikationen stehen dem Nutzer zum Download im World-Wide-Web zur Verfügung. Allein der Google-Play-Store bietet Android-Nutzern momentan 2,1 Millionen[App18] Applikationen für ihr Gerät zum Download (siehe hierzu Abb.1.1). Jede Woche werden zahlreiche neue Applikationen in den Stores eingeführt.

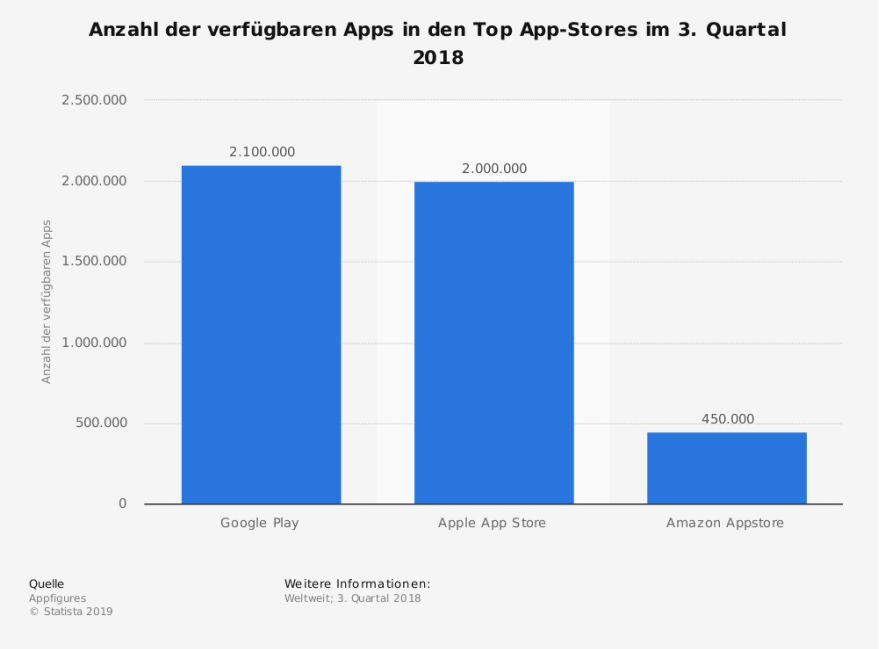


Abbildung 1.1: Anzahl der verfügbaren Apps in den Top App-Stores 2018[App18]

Sowohl privat geladene als auch von den Unternehmen bereitgestellte Applikationen werden auf das mobile Gerät installiert und die Nutzung der Geräte erfolgt über öffentliche Kanäle.
Das birgt ein enormes Gefahrenpotential, denn der große Markt für mobile Geräte ist für Cyberkriminelle nicht unbemerkt geblieben. Tausende von Cyberkriminellen überwachen den Markt und durchsuchen das Internet und die Stores nach Applikationen, die sie kompromittieren können, um z.B. Schadcode einzuschleusen und Nutzer zu schädigen. Dabei sind besonders mobile Geräte mit einem Android-Betriebssystem von dieser Bedrohung betroffen.
Smartphones mit dem Android-Betriebssystem sind derzeit eines der beliebtesten Open-Source-Betriebssysteme für mobile Geräte und gehören zu den meist verkauften Produkten nicht nur in Deutschland mit 77,9%[Kan18], sondern weltweit mit 84,2%[Gar18] und die Tendenz ist steigend. (siehe hierzu Abb.1.2 und Abb.1.3).

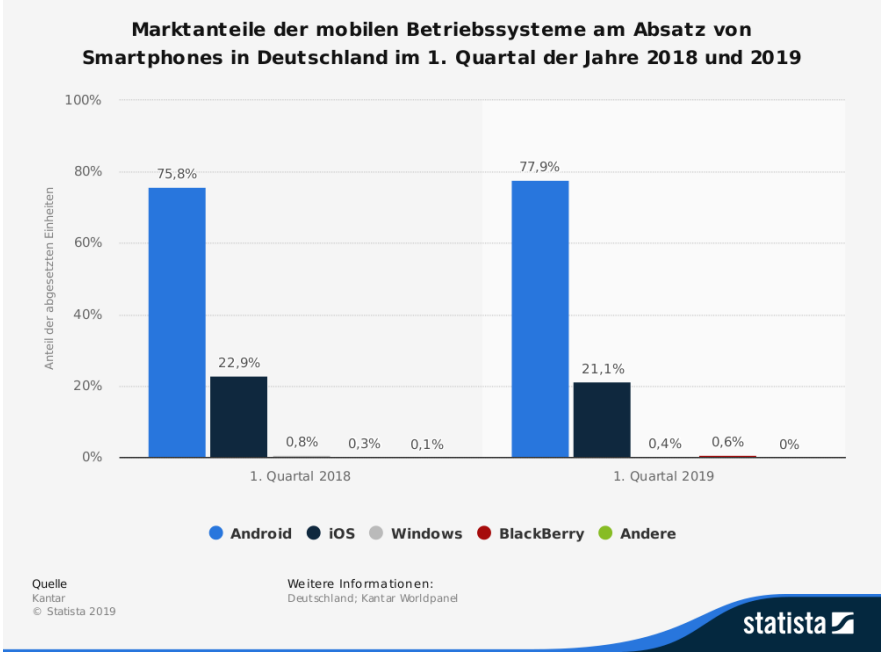


Abbildung 1.2: Marktanteil mobiler Betriebssysteme in Deutschland[Kan18]

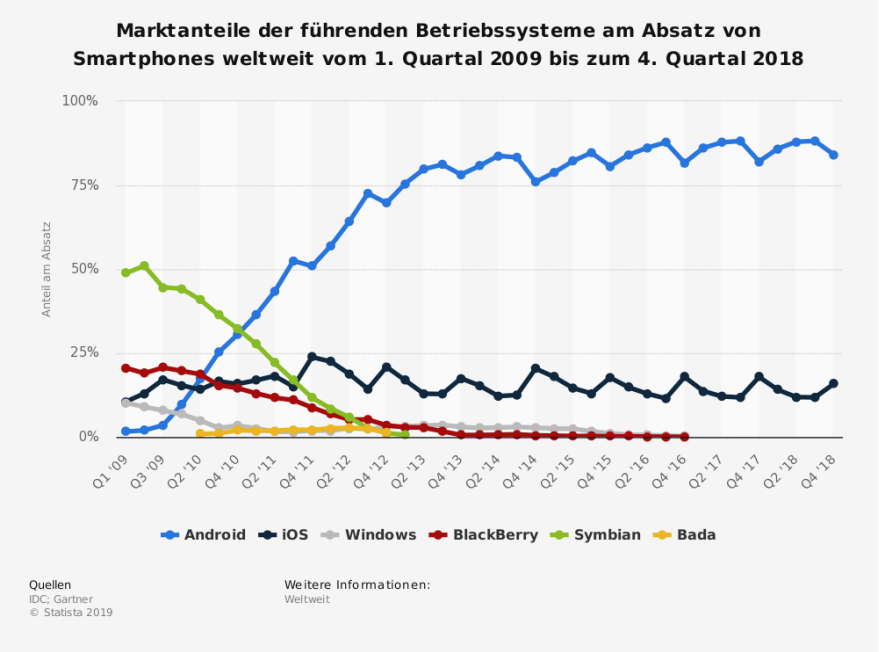


Abbildung 1.3: Marktanteil mobiler Betriebssysteme Weltweit[Gar18]

Die Beliebtheit und schnelle Verbreitung, machen das Android-Geräte, mit seinen Applikationen, die eine Schlüsselkomponente für die vollumfängliche Nutzung des Gerätes darstellen, zu einem attraktiven Ziel für Cyberkriminelle.

Nach einer Erhebung von G DATA Analysten (siehe hierzu Abb.1.4) ist die Anzahl

der neuen Android-Malware-Samples zwischen 2017 und 2018 über 40%, von 2.258.387 Millionen im 3.Quartal 2017 auf 3.189.480 Millionen im 3.Quartal 2018, gestiegen.

Abbildung 1.4: Neue Android-Schaddateien pro Jahr[G D18]

Dabei kristallisiere sich eine zunehmende Professionalisierung der Cyberkriminellen heraus mit einem Anstieg an Verkäufen von Malware für mobile Betriebssysteme. Die Malware werde immer ausgefeilter und ist, wenn vorhanden, von herkömmlichen Sicherheitssystemen, wie z.B. Virenscannern meist nicht mehr erkennbar. Sie ist sehr generisch und präsentiere sich in kürzester Zeit in einer neuen Verpackung.[G D18]

Die große Masse an Malwareattacken stellen Unternehmen und Bundesanstalten in Deutschland bei der Sicherung ihrer mobilen Infrastruktur vor enorme Herausforderungen. Gängige Sicherungsmechanismen, wie Firewalls, Intrusion Detection Systeme oder Vulnerability-und Viren-Scanner, reichen nicht mehr aus, um Informationssicherheit zu gewähren und somit die Vertraulichkeit, Integrität und Authentizität der Daten, die verarbeitet werden, zu schützen. Auch wenn die Malware erkannt werden sollte, ist die Aktualisierung und Anpassung der mobilen Geräte und Applikationen aufgrund der täglichen Masse an neuer Malware, mit großem Aufwand verbunden und gestaltet sich zu einem Wettlauf mit der Zeit.[GH17] Die Geräte und genutzten mobilen Applikationen müssen dem Nutzer u.a. ausreichend Schutz vor der Manipulation seiner Daten oder dem Abhören von Datenübertragungen bieten. Des Weiteren müssen die Unternehmenssysteme für den Mitarbeiter erreichbar sein und es muss sichergestellt sein, dass wirklich nur autorisierte Personen Zugriff auf die Systeme bzw. Infrastruktur haben.
Um dieser Bedrohung entgegen zu treten, müssen Unternehmen und Behörden Abwehrstrategien sowie Prozesse zu deren Umsetzung planen, um Sicherheit zu gewähren zu.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI), die nationale Cyber-Sicherheitsbehörde in Deutschland, ist für die Gestaltung von Informationssicherheit in der Digitalisierung durch Prävention, Detektion und Reaktion für Staat, Wirtschaft und die Gesellschaft, verantwortlich. Für die Absicherung von mobilen Applikationen und sicheren Betreibung eines Mobile-Device-Managements, stellt das BSI Richtlinien und Empfehlungen zur Verfügung.
Dabei stellen u.a. neben der Definierung eigener Sicherheitsrichtlinien und der Implementierung von verschiedenen Sicherungssystemen, die Sicherheitsanalyse bzw. Sicherheitstest von mobilen Geräten und ihren Applikationen ein wesentliches Instrument dar. Die Sicherheitstests, als Präventivmaßnahme, sollen bei der Auffindung von Schwachstellen helfen und zu einer Verringerung der Kompromittierung durch schadhafte Applikationen oder Malware beitragen.

Aber wie kann oder sollte ein mögliches Vorgehen für eine Sicherheitsanalyse einer mobilen Applikation aussehen? Was sollte der Sicherheitsexperte alles beachten, bevor die Anwendung auf der genutzten Hardware im Unternehmen zum Einsatz kommt? Zudem ist aufgrund des hohen Gefahrenpotentials ein schnelles und zielgerichtetes Vorgehen erforderlich.

Das Ziel dieser Bachelor-Thesis besteht darin, eine Strategie für eine Sicherheitsanalyse einer Android-Applikation zu erstellen. Dabei sind Empfehlungen zu erarbeiten, was ein Sicherheitsexperte beachten sollte, bevor die Applikation auf einem Smartphone im Unternehmen oder einer Behörde zum Einsatz kommt. Des Weiteren wird analysiert, welche Bedeutung die Sicherheitsanalyse von Android-Applikationen zum Schutz vor Malware bzw. Cyberangriffen hat. Die zu erstellende Strategie wird an einem ausgewählten Testfall prototypisch getestet.

Im ersten Abschnitt wird ein kurzer Überblick über die Geschichte, die Architektur und einzelnen Komponenten der Android-Applikationen gegeben. Als nächstes wird auf die aktuelle Bedrohungslage durch Malware eingegangen, Angriffsvektoren von mobilen Applikationen klassifiziert und wichtige Komponenten des Android-Sicherheitsmodells erläutert. Im vierten Abschnitt werden Grundlagen zur Sicherheitsanalysen, Testarten und Analysemethoden erläutert. Zudem werden exemplarisch die einzelnen Schritte einer Bedrohungsmodellierung dargestellt und der Aufbau der Sicherheitsanalyse, angelehnt an das OWASP Mobile Security Project, vorgestellt. Aufbauend darauf, werden Testapplikationen anhand eines ausgewählten Prüffall prototypisch einer Sicherheitsanalyse unterzogen.

2 Grundlagen: Geschichte und Architektur des Android-Systems

In diesem Kapitel sollen neben einem geschichtlichen Überblick zum Android-Betriebssystem, Grundlagen geschaffen werden, die für die spätere Durchführung der Sicherheitsanalyse und die abschließenden Bewertung der Android-Applikationen erforderlich sind. Dazu wird erläuternd auf die Architektur des Android-Betriebssystems, insbesondere auf das Applikationsframework und den Aufbau von Android-Applikationen eingegangen.

2.1 Entstehungsgeschichte des Android-Systems

Abbildung 2.1: Geschichte von Android[Tok15]

Die Entwicklung des Androids-Systems nahm ihren Anfang im Jahr 2003, mit der Gründung des Unternehmens Android Inc. in Palo Alto, Kalifornien, durch Andy Rubin, Rich Miner, Nick Sears und Chris White. Eine der Hauptambitionen für die Gründung war, ein fortschrittliches Betriebssystem für Digitalkameras zu entwickeln. Damals sollte das Android OS dazu dienen, Betriebssysteme von Digitalkameras dahingehend zu verbessern, dass sie mit einem PC über LAN oder WLAN verbunden werden können. Zudem sollte eine Cloud-Umgebung[1] für die Online-Speicherung von Fotos für die Nutzer geschaffen werden. Allerdings war der Markt für Digitalkameras rückläufig, woraufhin sich das Team beschloss, Android auf die Nutzung in Mobiltelefonen auszurichten.[Cal18]

[1] Definition „Cloud“ Als Cloud (auch Cloud-Computing[1]) bezeichnet man eine Reihe von bereitgestellten Diensten, die über ein Netzwerk (zum Beispiel dem Internet) für die Nutzer zur Verfügung gestellt werden. https://www.droidwiki.org/w/index.php?title=Cloud&oldid=6555. [Online; abgerufen am 14. Mai 2019].

Im Jahr 2005 kam es dann zu einer Übernahme von Android Inc. durch Google[2]. Google kaufte das Unternehmen aber letztendlich nicht um selber Smartphones herzustellen, wie zuerst viele glaubten, sondern mit der Idee, eine Plattform bereitzustellen, die diverse mobile Geräte von unterschiedlichen Herstellern unterstützen kann. Google sah im Android-System die Chance, seine Dienstleistungen einer möglichst breiten Masse anbieten zu können, um weiterhin konkurrenzfähig zu bleiben.
Mit der Gründung der Open Handset Alliance[Ope07] im Jahr 2007, mittlerweile ein Konglomerat aus 84 weltweit agierenden Technologie-Unternehmen mit dem Ziel der Förderung offener Standards in der Mobilfunk-Domäne, verfolgte Google sein Vorhaben weiter und Android wird offiziell zur Open-Source Plattform[3]. Google trägt gemeinsam mit der Open Handset Alliance im Rahmen des Android-Open-Source-Project, die Entwicklung. Zudem steht Android unter der Apache und GPL Lizens, wodurch die Unternehmen freien Zugriff auf die benötigte Software für ihre Geräte haben. Dadurch fallen für die Nutzung keine Lizenzgebühren an und Softwareentwickler können Anwendungen für mehrere Geräte freigeben.
Kurze Zeit später, im Oktober 2008, wird die Android-Version 1.0 mit dem Codenamen „Astro“ veröffentlicht und das erste mit Android-Astro 1.0 betriebene Smartphone T-Mobile G1-HTC Dream kommt auf den Markt. Das Android-Astro 1.0 war ein Vorreiter im Umgang mit Benachrichtigungen und das Pulldown-Benachrichtigungsfenster war eine revolutionäre Erweiterung. Weitere wesentliche Funktionen waren der Android-Market, erste Version des Google-Play-Stores[4], und die Verwendung von Startbildschirm-Widgets[5]. Die Weiterentwicklung des Android-Systems bis heute zeichnet sich deutlich in seinen unterschiedlichen Versionen ab, die ab der Version 1.5 Cupcake, Namen von US-Süßigkeiten erhielten und später, ab Version 4.4 KitKat, im Zuge einer Kooperation

[2] „Google“ Die Google LLC ist ein Unternehmen mit der US-amerikanischen Rechtsform Limited Liability Company mit Hauptsitz im kalifornischen Mountain View und Tochterunternehmen der Holding-Gesellschaft XXVI Holdings Inc. Diese gehört dem Unternehmen Alphabet Inc. Seite „Google LLC“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 10. Mai 2019, 13:28 UTC.URL:https://de.wikipedia.org/w/index.php?title=Google_LLC&oldid=188424702(Abgerufen: 15. Mai 2019, 08:45 UTC)

[3] Definition „Open-Source“ Unter freier Software, auch Open Source Software, wird Software verstanden, deren Quellcode offen gelegt und ebenfalls frei verfügbar ist. Im engeren Sinne ist damit Software gemeint, die unter einer, durch die Open Source Initiative anerkannten, freien Lizenz veröffentlicht ist (und somit frei verfügbar ist), bspw. der GNU General Public License. https://www.droidwiki.org/w/index.php?title=Freie_Software&oldid=15485. [Online; abgerufen am 14. Mai 2019]

[4] „Google-Play-Store“ Google Play (früher Android Market) ist ein App Store des US-amerikanischen Unternehmens Google, der Mobile Apps, Musik, E-Books, Hörbücher, Filme, Fernsehserien und digitale Zeitschriften anbietet. Google Play wurde am 28. August 2008 als Android Market angekündigt und am 22. Oktober 2008 für die Nutzung zur Verfügung gestellt.Seite „Google Play“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 5. Mai 2019, 20:00 UTC. URL: https://de.wikipedia.org/w/index.php?title=Google_Play&oldid=188268786. (Abgerufen: 15. Mai 2019, 10:08 UTC.)

[5] Definition „Widgets“ Ein Widget ist eine Komponente eines grafischen Fenstersystems. Das Widget besteht zum einen aus dem Fenster, einem sichtbaren Bereich, der Maus-, Touchscreen- und/oder Tastaturereignisse empfängt, und zum anderen aus dem nicht sichtbaren Objekt, das den Zustand der Komponente speichert und über bestimmte Zeichenoperationen den sichtbaren Bereich verändern kann. Seite „Widget“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 26. Mai 2018, 06:09 UTC. URL:https://de.wikipedia.org/w/index.php?title=Widget&oldid=177755590. (Abgerufen: 15. Mai 2019, 10:15 UTC).

mit Nestlé, Namen von Nestlé-Süßigkeiten. Seit 2011 erscheint jedes Jahr eine neue Android-Version und mittlerweile gibt es mit der Android-Version-9 namens Pie, die 28 Android-API. Wie in der unteren Abbildung 2.2 zu erkennen, stellt die Android-API 23, gefolgt von der Android-API 27, die meist genutzte API dar.[KC13, And19a, Spr17]

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Abbildung 2.2: The Android software stack[And19a]

2.2 Architektur des Android-Systems

Das Android-Betriebssystem ist, wie in Kapitel 2.1 auf Seite 7 erklärt wurde, ein Open-Source-System, dass auf einem Linux-Kernel basiert. Bei der Entwicklung des Android-Systems wurde auf Linux zurückgegriffen, weil die Entwickler, wie u.a. Andy Rubin, ein System schaffen wollten, das wie Linux mit möglichst vielen Geräten kompatible ist. Zudem bringt der Linux-Kernel verschiedene Sicherheitsfunktionen mit, auf die im weiteren Verlauf der Arbeit noch eingegangen wird, und übernimmt die Speicherverwaltung und das Management der Prozesse, denen er bei Bedarf die benötigten Ressourcen zuweisen kann.[Mac16]

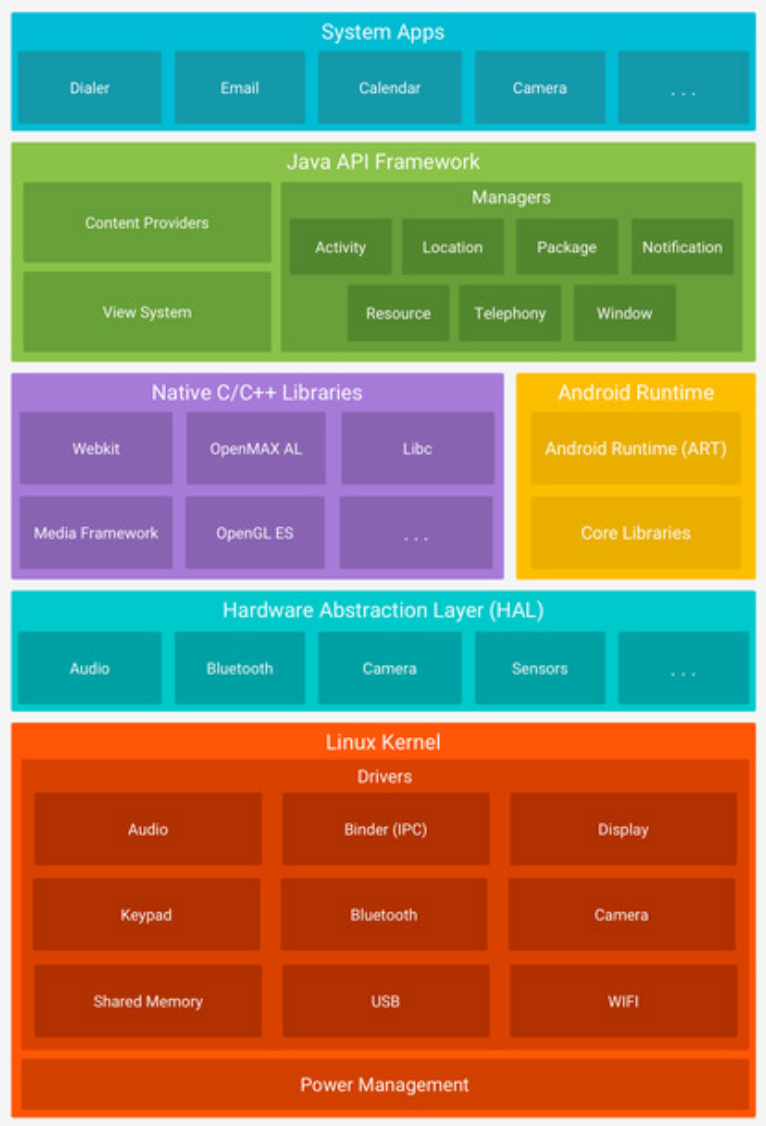


Abbildung 2.3: The Android software stack[And19c]

Die Architektur von Android versteht sich als ein mehrschichtiges-Modell für mobile Geräte, wie z.B. Smartphones oder Tablets. Dabei bildet der Android(-Linux)-Kernel, von unten betrachtet siehe Abbildung 2.3 die Basis des System, auf dem der Hardware-Abstraktions-Layer (HAL)aufsetzt, gefolgt von der Android-Laufzeitumgebung-Umgebung, die die Android-Runtime und die Java-Bibliotheken enthält. Neben der Android-Laufzeit-Umgebung befinden sich in der dritten Schicht noch die C/C++-Bibliotheken und die beiden obersten Schichten bilden das Java-Application Framework und die Anwendungsebene, die die System-und Benutzer-Apps, enthält. Jede der Schichten stellt Schnittstellen und Systemressourcen für die darüberliegende Schicht bereit, so dass die Schichten untereinander interagieren können. Des Weiteren bietet Android Möglichkeiten, Anwendungen auf verschiedenen Ebenen zu erstellen, auszuführen und zu verwalten. Zu diesem Zweck werden beim Android-System klar die Begriffe

Anwendung, Prozess, Aufgabe und Thread voneinander abgegrenzt.[Zha18]

2.2.1 Der Linux-Kernel

Bei Android bildet ein Linux-Kernel der speziell für die Hardware der Android-Geräte modifiziert wurde, die Grundlage des Betriebssystems. Zu den wichtigsten Modifikationen gehören zum einen der Binder, ein Treiber zur Unterstützung der RPC[6]/IPC[7]-Mechanismen in Android, dann Ashmen, ein Ersatz für die standardmäßige Linux-Shared-Memory[8], Wakelock, ein Mechanismus, zur Steuerung der Energieverwaltung und zur Unterbindung der Ruhefunktion des Kernels und der Logger, der eine systemweite Protokollierungsfunktion bietet und mit dem Kommandozeilen-Tool logcat von Android ausgelesen werden kann.[Bir15]
Der Linux-Kernel oder auch Android-Kernel, ist für jeden im Quellcode frei verfügbar und stellt Treiber für die Verwaltung der Prozesse, der Speicher, der Interprozesskommunikation (IPC), das Netzwerk sowie die Zugriffe auf die Dateisysteme zur Verfügung. Somit übernimmt der Kernel das Scheduling für die Hardwarezugriffe und bildet eine Schnittstelle zwischen der Hardware, der Programmsteuerung, den Bibliotheken und der Android-Laufzeit-Umgebung.[Ele14, Spr17]
Da die meisten mobilen Geräte über verschiedene Chipsätze verfügen, sind über die Jahre unterschiedliche, auf die Geräte angepasste Kernel-Versionen entwickelt wurden. So wird im Gegensatz zu Linux, bei Android kein generischer Kernel gepflegt, dessen Idee eigentlich ist, viele verschiedene Hardwarekomponenten und Funktionen, Open-Source zu unterstützen. Bei Android-Geräten kann die Kernel-Version in der Regel unter den Geräteeinstellungen eingesehen werden. Die aktuellste Longterm Stable Version[9] ist die Version 4.19 und wurde am 22.Oktober 2018 veröffentlicht.[The18, Mac16] Ein wichtiger und auch entscheidender Teil des Kernels, sind bei der Interprozesskommunikation

[6]Definition „Remote Procedure Call“ (RPC; dt. „Aufruf einer fernen Prozedur“) ist eine Technik zur Realisierung von Interprozesskommunikation. Sie ermöglicht den Aufruf von Funktionen in anderen Adressräumen.Seite „Remote Procedure Call“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 3. April 2019, 19:32 UTC. URL:https://de.wikipedia.org/w/index.php?title=Remote_Procedure_Call&oldid=187207234. (Abgerufen: 16. Mai 2019, 18:45 UTC).

[7]Definition „Interprozesskommunikation“ Der Begriff Interprozesskommunikation (englisch interprocess communication, kurz IPC) bezeichnet in der Informatik verschiedene Verfahren des Informationsaustausches zwischen den Prozessen eines Systems. Seite „Interprozesskommunikation“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 8. Februar 2019, 12:28 UTC. URL:https://de.wikipedia.org/w/index.php?title=Interprozesskommunikation&oldid=185494073. (Abgerufen: 16. Mai 2019, 18:00 UTC).

[8]Definition „Linux-Shared-Memory“ Shared Memory ist ein vom Kernel verwalteter Speicherbereich, der von mehreren Prozessen gelesen und beschrieben werden kann.

[9]„Longterm Stable Version“ Stabile Kernel-Releases basieren direkt auf den Releases von Linus Torvald. Jeder stabile Kernel-Release-Baum wird von einem einzelnen Kernel-Entwickler gepflegt, der für die Auswahl der erforderlichen Patches für das Release und die Verwaltung des Überprüfungs- / Release-Prozesses verantwortlich ist. Seite „Stabile Kernel Releases & Updates“ In: Android Source. URL:https://source.android.com/devices/architecture/kernel/releases. (Abgerufen: 15. Mai 2019, 17:11 UTC).

IPC, die Prozesse mit ihrem dazugehörigen Process Identifier PID.[10] Erst anhand der PID können die einzelnen Prozesse eindeutig den jeweiligen Anwendungen zugeordnet werden.(siehe Abbildung 2.4).[Spr17]

Example 2-7. zygote and application process relationship

```
$ ps
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     680    540   ffffffff 00000000 S /init❶
--snip--
root      181   1     858808 38280 ffffffff 00000000 S zygote❷
--snip--
radio     1139  181   926888 46512 ffffffff 00000000 S com.android.phone
nfc       1154  181   888516 36976 ffffffff 00000000 S com.android.nfc
u0_a7     1219  181   956836 48012 ffffffff 00000000 S com.google.android.gms
```

Abbildung 2.4: PID und die zugeordnete Anwendung[Ele14]

Daneben speichert der Kernel Informationen über den Prozessstatus und den Thread[11] sowie Details über verwendete Dateien. Die Prozesse werden in der -task_struct- abgelegt und verwaltet. Die -task_struct- beinhaltet alles, was an Informationen zu einem Android-Prozess gehört, siehe hierzu Ausschnitt „task_struct“ Abbildung 2.5..[Man10]

```
struct task_struct {
    ...
    int prio;                   // Dynamische Priorität
    int static_prio;            // Statische Priorität
    unsigned long sleep_avg;    // Zeit, die der Prozess schläft (inaktiv ist)
    unsigned long last_run;     // Zeitpunkt, zu dem der Prozess zuletzt lief
    unsigned long policy;       // Scheduling-Strategie: Normal (Timesharing), FIFO, RR
    ...
    unsigned int time_slice;    // verbleibendes Quantum
    ...
}
```

Abbildung 2.5: Aufbau der task-struct[Man10]

[10] Definition „Process Identifier“ Ein Process identifier (process ID, PID; deutsch Prozesskennung) ist in der Informatik ein einzigartiger Schlüssel, welcher der eindeutigen Identifikation von Prozessen dient. Der PID ändert sich während der Laufzeit des Prozesses nicht.Die meisten Betriebssysteme (z. B. Unix, Windows) verwenden dafür eine natürliche Zahl. Process IDs werden hauptsächlich für die Interprozesskommunikation benötigt. Seite „Process identifier“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 28. Oktober 2016, 08:01 UTC. URL:https://de.wikipedia.org/w/index.php?title=Process_identifier&oldid=159146052. (Abgerufen: 16. Mai 2019, 17:38 UTC).

[11] Definition „Thread“ Ein (Kernel-)Thread ist ein sequentieller Abarbeitungslauf innerhalb eines Prozesses und teilt sich mit den anderen vorhandenen Threads (multithreading) des zugehörigen Prozesses eine Reihe von Betriebsmitteln, wie z.B. das Codesegment oder das Datensegment. Seite „Thread (Informatik)“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 5. März 019, 09:41 UTC. URL:https://de.wikipedia.org/w/index.php?title=Thread_(Informatik)&oldid=186272528. (Abgerufen: 16. Mai 2019, 18:12 UTC).

2.2.2 Die Hardwareabstraktionsschicht

In der Hardwareabstraktionsschicht HAL werden Standardschnittstellen, wie z.B. Kamera-Treiber[Andj], definiert und bereitgestellt, die die Hardwarefunktionen für das übergeordnete Android-Applikations-Framework verfügbar machen. Über die Schnittstellen für die Kameratreiber hinaus, werden die Zugriffe auf Audio-, Bluetooth-und Sensortreiber verwaltet. Unter Linux werden die Treiber für die Hardware eigentlich entweder in den Kernel eingebettet oder dynamisch gelinkt. Im Android-Betriebssystem wird dagegen für jeden Hardwaretyp speziell eine Anwendung, die von den oberen Schichten für die Interaktion mit dem entsprechenden Hardwaretyp verwendet wird, bereitgestellt. Die Treiber sind nicht in den Kernel eingebettet und die Mechanismen zum dynamischen Linken der Treiber ist deaktiviert.[Zha18, And19g]

```
127|dreamlte:/ $ ls -l /system/lib/hw
total 7808
-rw-r--r-- 1 root root   34460 2008-12-31 16:00 audio.a2dp.default.so
-rw-r--r-- 1 root root   22168 2008-12-31 16:00 audio.playback_record.default.so
-rw-r--r-- 1 root root   18072 2008-12-31 16:00 audio.primary.default.so
-rw-r--r-- 1 root root   18072 2008-12-31 16:00 audio.primary.goldfish.so
-rw-r--r-- 1 root root  117960 2008-12-31 16:00 audio.primary.universal8895.so
-rw-r--r-- 1 root root   34456 2008-12-31 16:00 audio.r_submix.default.so
-rw-r--r-- 1 root root   26280 2008-12-31 16:00 audio.tms.default.so
-rw-r--r-- 1 root root   30360 2008-12-31 16:00 audio.usb.default.so
-rw-r--r-- 1 root root   96016 2008-12-31 16:00 audio_policy.default.so
-rw-r--r-- 1 root root 1690592 2008-12-31 16:00 bluetooth.default.so
-rw-r--r-- 1 root root   13844 2008-12-31 16:00 camera.goldfish.jpeg.so
-rw-r--r-- 1 root root  219144 2008-12-31 16:00 camera.goldfish.so
-rw-r--r-- 1 root root   13844 2008-12-31 16:00 camera.ranchu.jpeg.so
-rw-r--r-- 1 root root  219144 2008-12-31 16:00 camera.ranchu.so
-rw-r--r-- 1 root root   42904 2008-12-31 16:00 camera.universal8895.so
-rw-r--r-- 1 root root   22220 2008-12-31 16:00 cellgeofence.default.so
-rw-r--r-- 1 root root   13976 2008-12-31 16:00 fingerprint.default.so
-rw-r--r-- 1 root root   22168 2008-12-31 16:00 fingerprint.goldfish.so
-rw-r--r-- 1 root root   22168 2008-12-31 16:00 fingerprint.ranchu.so
-rw-r--r-- 1 root root   22232 2008-12-31 16:00 gatekeeper.exynos8895.so
-rw-r--r-- 1 root root   18072 2008-12-31 16:00 gps.goldfish.so
-rw-r--r-- 1 root root   18072 2008-12-31 16:00 gps.ranchu.so
-rw-r--r-- 1 root root   22488 2008-12-31 16:00 gralloc.default.so
-rw-r--r-- 1 root root   26600 2008-12-31 16:00 gralloc.exynos5.so
-rw-r--r-- 1 root root   22168 2008-12-31 16:00 hwcomposer.exynos5.so
-rw-r--r-- 1 root root   13896 2008-12-31 16:00 keystore.default.so
```

Abbildung 2.6: Ausschnitt-HAL Libaries Galaxy S8

2.2.3 Die nativen C/C++-Bibliotheken

Über dem HAL befinden sich die nativen Bibliotheken, die C/C++-Bibliotheken, von Android. Die Bibliotheken werden zur Laufzeit dynamisch gelinkt und ermöglichen dem Gerät verschiedene, hardwarespezifischen Daten zu verarbeiten. Die Bibliotheken werden über Java-Schnittstellen abgerufen. Eine wichtige native Bibliothek ist z.B. der „Surface-Manager“ der zum Verwalten der Geräteanzeige verwendet wird oder auch die SQLite[12]-Datenbank, die in Android für die Datenspeicherung verwendet wird und für

[12]Definition „SQLite“ SQLite ist eine gemeinfreie Programmbibliothek, die ein relationales Datenbanksystem enthält. Sie unterstützt einen Großteil der im SQL-92-Standard festgelegten SQL-Sprachbefehle.

alle Anwendungen verfügbar ist. Überdies beinhalten die native Bibliotheken u.a. noch eine Browser-Engine, für die Anzeige von HTML-Inhalten, sowie das Media-Framework, dass die Wiedergabe und Aufnahme verschiedener Audio-, Video-und Bildformate, wie z.B. MPEG, JPEG oder auch AAC erlaubt.[Nar16, Gar11]

2.2.4 Die Android-Laufzeitumgebung

Die Android-Laufzeitumgebung ART setzt sich aus der Android-Runtime und den Java-Kernbibliotheken zusammen und befindet sich in der gleichen Schicht wie die C/C++-Bibliotheken. Die ART ist eine für Android-Geräte optimierte Java-Virtual-Machine JVM. In der JVM wird Java-Quellcode in Java Bytecode kompiliert, der in .class-Dateien gespeichert wird. Die .class-Dateien werden zur Laufzeit von der JVM gelesen. Jede Klasse im Java-Code wird in eine .class-Datei konvertiert. In der ART wird der Java-Quellcode zwar auch in .class-Dateien kompiliert. Aber die .class-Dateien werden nach der Generierung, mit dem Dex-Compiler, dem sogenannten „Dexer“ in eine .dex-Datei[13] konvertiert, die dann von der DVM interpretiert werden können (siehe hierzu Abbildung 2.7.).[Gar11, Nar16, RSM15]

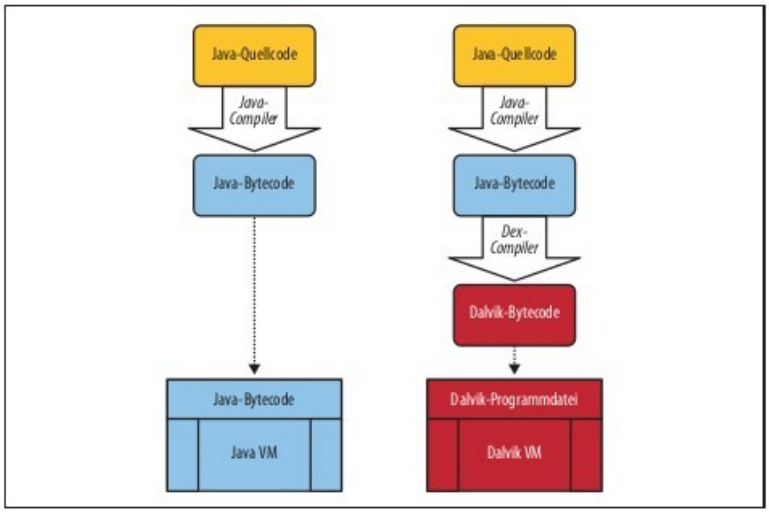


Abbildung 2.7: Die Dalvik-VM im Vergleich zur Java-VM[Gar11]

Im Android-Betriebssystem wird jede Java-Anwendung in der Android-Runtime aus-

Unter anderem implementiert SQLite Transaktionen, Unterabfragen (Subselects), Sichten (Views), Trigger und benutzerdefinierte Funktionen. Seite „SQLite“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 23. April 2019, 17:54 UTC. URL:https://de.wikipedia.org/w/index.php?title=SQLite&oldid=187842777. (Abgerufen: 16. Mai 2019, 10:42 UTC)

[13]Definition „DEX-Datei“ Dalvik Executable Format (DEX) ist der von Android verwendete Bytecode, eine Abwandlung des Java-Bytecode. Aus mehreren .CLASS-Dateien entsteht durch einen DEX-Compiler (alias "Dexer") eine .DEX-Datei. Seite “Was ist Dalvik Executable Format?“İn: dotnetframework. Bearbeitungsstand: 18.12.2017. URL:https://www.overleaf.com/project/5cbb25a003073944b64b6851. (Abgerufen: 16.Mai 2019, 18:40 UTC

geführt, für die eine eigene Instanz, eine sogenannte Sandbox[14], generiert wird. Da für jede Anwendung eine eigene virtuelle Maschine ausgeführt wird, ist es notwendig, dass die VM-Instanzen möglichst schnell gestartet werden. Für den Start und Initialisierungsprozess der ART nutzt das Android-System daher Zygote. Zygote lässt sich als ein Prozess auffassen, in dem alle Java-Kernbibliotheken verknüpft sind. Der Zygote-Prozess wird mit dem System gestartet und parallel wird eine ART initialisiert, die die Java-Kernbibliotheksklassen vorlädt und vorinitialisiert.[Zha18, Bor08]

2.2.5 Das Android-Applikations-Framework

Im Android-Applikations-Framework befinden sich alle Java-Bibliotheken, die nicht Teil der Android-Laufzeitumgebung sind. Daneben enthält das Framework die grundlegenden Komponenten bzw. Services zum Erstellen von Android-Anwendungen, wie z.B. die Basisklassen für die „Activities“ „Services“ und „Content Providers“ oder auch Klassen für die Datei-und Datenbankzugriffe. Jeder Service ist für die Bereitstellung des Zugriffs auf eine bestimmte Systemressource, wie z.B. den Windows-Manager oder auch Location-Manager, verantwortlich.[Ele14, Bac16]

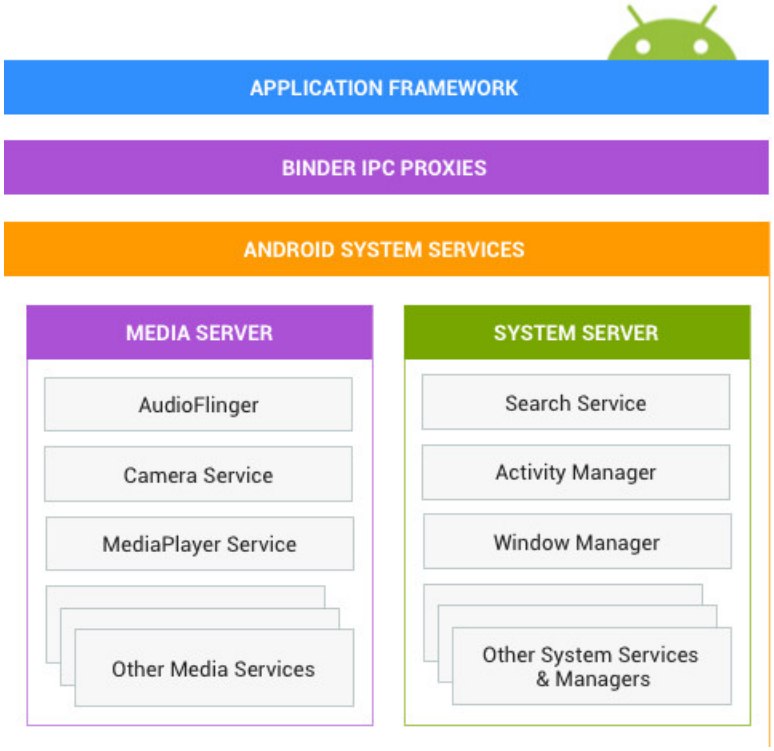


Abbildung 2.8: Aufbau Android-Applikations-Framework[Andd]

[14]Definition „Sandbox“ Mit dem Begriff Sandbox wird eine Technik bezeichnet, Software innerhalb einer speziellen - d.h. von den übrigen Systemressourcen isolierten - Laufzeitumgebung auszuführen. Die Technik kann mit dem Prinzip von in sich geschlossenen, sicheren Containern verglichen werden, in denen Software ausgeführt werden kann, ohne andere Ressourcen eines Systems zu beeinflussen. Seite „Sandbox“ In: ITWissen.info. Bearbeitungsstand: 20.02.2019. URL:https://www.itwissen.info/Sandbox-sandbox.html. (Abgerufen am 14.Mai 2019 20:34 UTC)

Die Android-Services werden über Schnittstellen aufgerufen, die von Binder IPC Proxies bereitgestellt werden und in AIDL[15] programmiert sind. Der AIDL-Compiler generiert automatisch Stub[16]-und Proxy-Klassen[17], die das schnittstellenspezifische Remote-Procedur-Call-Protokoll zum Aufrufen des Dienstes implementieren. Mit Stubs-Klassen werden die Binder-Schnittstelle implementiert und mit den Proxies werden die Services aufgerufen.[Ele14, Andd, Bac16]

2.2.6 Die Anwendungsebene- Android-Applikationen

In der obersten Schicht des Androids-System befinden sich die Android-Applikationen, mit denen die Benutzer direkt interagieren können. Die Applikationen werden in der Regel in Java programmiert und wie in Kapitel 2.2.4, schon erläutert, wird zur Laufzeit für jede Applikation eine eigene Instanz in der Android-Runtime ausgeführt.[Spr17] Alle Applikationen haben die gleiche Struktur und bauen auf dem Android-Applikations-Framework auf. Zudem wird zwischen System-Applikationen und Benutzer-Applikationen unterschieden.
Die System-Applikationen stellen entweder Kernbetriebssysteme, wie z.B. die Zertifikatsverwaltung oder Systemeinstellungen, oder vorinstallierte Benutzeranwendungen, wie Mailclients oder Browser, dar.[Wik16] Sie sind auf der Partition /system (/system/app) installiert und schreibgeschützt. Aufgrund dessen können sie vom Benutzer weder geändert noch deinstalliert werden.

[15] Definition „AIDL“ Die Android Interface Definition Language ist eine Java-ähnliche Sprache für die Definition von RPC-Schnittstellen (für Interprozesskommunikation, IPC). AIDL-Programme werden im Android-Entwicklungssystem (z. B. Android Studio) als .aidl-Dateien gespeichert, aus denen die Android SDK tools automatisch Java-Dateien generieren (ähnlich wie auch aus XML-Ressourcendateien). Über diese können entfernte Prozeduraufrufe durchgeführt werden. Seite „Schnittstellenbeschreibungssprache“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 1. August 2016, 21:54 UTC. URL:https://de.wikipedia.org/w/index.php?title=Schnittstellenbeschreibungssprache&oldid=156646016. (Abgerufen: 25. Mai 2019, 10:10 UTC).

[16] Definition „Stub“ Ein Stub (von englisch stub, Stubben, Stummel, Stumpf) oder Stutzen bezeichnet in der Softwareentwicklung einen Programmcode, der anstelle eines anderen Programmcodes steht. Dabei ist der eigentliche Programmcode, der durch den Stub ersetzt wird, noch nicht entwickelt (Top-Down-Ansatz) oder auf einem anderen Rechner oder Speicherbereich. Liegt der Code an einem anderen Ort, so ist der Stub der lokale Anknüpfungspunkt, um Softwarekomponenten einfach anzusprechen, die ansonsten nur über komplexe Protokolle erreichbar wären bzw. sind und um diese Komplexität zu verbergen. Ein Stub entspricht dann dem Entwurfsmuster eines Stellvertreters. Seite „Stub (Programmierung)“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 2. September 2018, 12:10 UTC. URL:https://de.wikipedia.org/w/index.php?title=Stub_(Programmierung)&oldid=180559257 (Abgerufen: 16. Mai 2019, 17:14 UTC)

[17] Definition „Proxy“ Der Proxy, auch Stellvertreter genannt, ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung, das zur Kategorie der Strukturmuster (englisch structural design patterns) gehört. Das Muster überträgt die Steuerung eines Objektes auf ein vorgelagertes Stellvertreterobjekt. Ein Proxy in seiner allgemeinsten Form ist eine Klasse, die als Schnittstelle zu einem so genannten Subjekt auftritt. Dieses Subjekt kann beispielsweise eine Netzwerkverbindung, ein großes Objekt im Speicher, eine Datei oder eine andere Ressource sein. Als Stellvertreter dieses Subjektes kann der Proxy die Erzeugung des Subjektes sowie den Zugriff darauf steuern. Seite „Stellvertreter (Entwurfsmuster)“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 23. Dezember 2018, 08:27 UTC. URL:https://de.wikipedia.org/w/index.php?title=Stellvertreter_(Entwurfsmuster)&oldid=183960990. (Abgerufen: 16. Mai 2019, 17:18 UTC)

Benutzer-Applikationen liegen auf der Partition /data (/data/data/app) und können beispielsweise vom Benutzer über den Google-Playstore geladen sowie vom Bentzer ohne Einschränkung installiert oder deinstalliert werden. Genauso wie die System-Applikationen werden auch für die Benutzer-Applikationen eigene Instanzen in der ART ausgeführt. Des Weiteren enthält jede Applikation ein Datenverzeichnis /data/data/[Name-Applikation], indem Daten gespeichert werden, die zur Laufzeit entstanden.[Ele14, Wik18a]

2.3 Aufbau und Bestandteile von Android-Applikationen

Wie schon in Kap.2.2.3 und Kap.2.2.6 kurz angesprochen, werden die Android-Applikationen hauptsächlich in Java geschrieben und die einzelnen Komponenten, wie der Quellcode, Ressourcen-Dateien oder auch Metadaten, der Applikation in eine Archivdatei, die .apk-Datei, kompiliert. In der folgenden Abbildung wird die Kompilierung einer Android-Anwendung dargestellt.
Wie in Abb. 2.9 zu sehen, werden zuerst aus dem Java-Code mit dem Java-Compiler .class-Dateien erstellt, die Java-Bytecode enthalten. Die .class-Dateien werden dann gemeinsam mit den jar. Bibliotheksdateien, mit dem Dex-Compiler zu einzelnen .dex-Dateien kompiliert. Im letzten Schritt werden die einzelnen Komponenten, wie die .dex-Dateien, die Ressourcen-Dateien und Metadaten sowie den nativen C/C++-Biblotheken, mit dem APK-Builder (APPT) zu einer .apk-Datei komprimiert.[Cha17, RSM15]

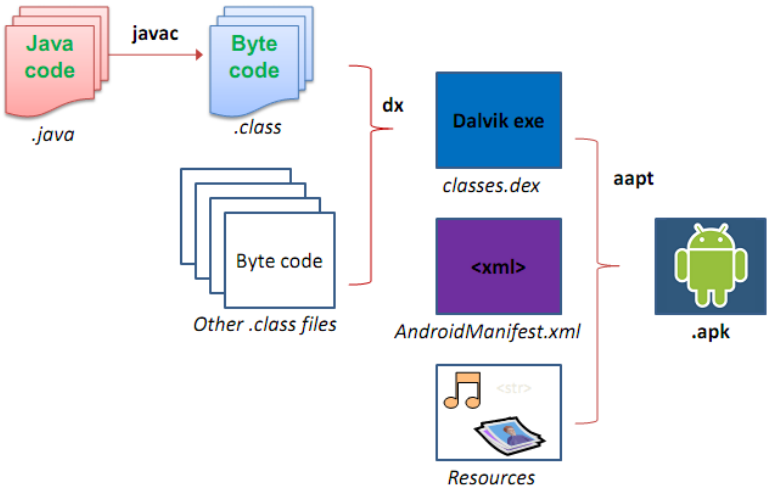


Abbildung 2.9: Der Android-Build-Prozess[Cha17]

Die Hauptkomponenten einer Android-Applikation sind im Allgemeinen, das Android-Manifest, Activities, die Services, die Broadcast-Reciever, die Content-Provider, und die Shared-Preferences. Die Kommunikation zwischen den einzelnen Komponenten wird unter Verwendung eines speziellen Datentyps, den Intents, realisiert. Zudem werden allen Komponenten, wie die Activities oder auch Services, im Android-Manifest,

das eine .xml-Datei ist, deklariert. Zusätzlich umfasst das Android-Manifest.xml u.a. von der Anwendung benötigte Benutzerberechtigungen, die sogenannten Permissions, und die Mindest-API-Stufe sowie die von der Anwendung zu verwendende Hardware- und Softwarefunktionen, wie z.B. Multitouch-Bildschirm oder Kamera. Neben dem Android-Manifest.xml beinhaltet die .apk-Datei, die classes.dex, die aus dem kompilierte Java-Code bestehen einen lib-Ordner, der den für die Prozessor-Software spezifischen kompilierten Code enthält, einen META-INF-Ordner, der das Anwendungszertifikat, eine Liste der Ressourcen und einen SHA1[18]-Digest aller Ressourcen beinhaltet und einer resources.arsc-Datei, die vorkompilierte Bibliotheken sowie einen res-Ordner, der Bibliotheken enthält, die nicht in der resources.arsc kompiliert wurden (siehe hierzu Abb.2.10).[Spr17, Ele14]

```
root@annie-Aspire-E1-572G:/home/annie/Schreibtisch/APK Analyse/Testapk# unzip -l UnCrackable-Level1.apk
Archive:  UnCrackable-Level1.apk
  Length      Date    Time    Name
---------  ---------- -----   ----
     1648  1980-00-00 00:00   AndroidManifest.xml
     1319  1980-00-00 00:00   META-INF/CERT.RSA
     1139  1980-00-00 00:00   META-INF/CERT.SF
     1077  1980-00-00 00:00   META-INF/MANIFEST.MF
     5528  1980-00-00 00:00   classes.dex
     1260  1980-00-00 00:00   res/layout/activity_main.xml
      456  1980-00-00 00:00   res/menu/menu_main.xml
     4751  1980-00-00 00:00   res/mipmap-hdpi-v4/ic_launcher.png
     2348  1980-00-00 00:00   res/mipmap-mdpi-v4/ic_launcher.png
     7275  1980-00-00 00:00   res/mipmap-xhdpi-v4/ic_launcher.png
    14398  1980-00-00 00:00   res/mipmap-xxhdpi-v4/ic_launcher.png
    22801  1980-00-00 00:00   res/mipmap-xxxhdpi-v4/ic_launcher.png
     2748  1980-00-00 00:00   resources.arsc
---------                     -------
    66748                     13 files
```

Abbildung 2.10: Inhalt einer Android-Applikation

Des Weiteren wird zwischen webbasierten-, nativen-und hybriden Applikationen unterschieden. Bei einer Web-Applikation handelt es sich um eine App, die zur Bereitstellung z.B. von Interaktions-und Navigationsfunktionen, vorrangig mit Webtechnologien, wie JavaScript oder HTML5, entwickelt wurde. Die Applikationen werden im Webbrowser ausgeführt und gerendert.
Dagegen werden native Applikationen mit plattformabhängigen Technologien entwickelt. Sie werden entsprechend dem Betriebssystem und der Hardwarekomponenten angepasst, die auf dem Gerät zum Einsatz kommen. Native Applikationen werden nicht im Browser ausgeführt sondern auf dem Gerät selbst und werden in der Regel in Java, C++ oder auch Python programmiert.

[18]Definition „SHA“ Der Begriff Secure Hash Algorithm (kurz SHA, englisch für sicherer Hash-Algorithmus) bezeichnet eine Gruppe standardisierter kryptologischer Hashfunktionen. Diese dienen zur Berechnung eines Prüfwerts für beliebige digitale Daten (Nachrichten) und sind unter anderem die Grundlage zur Erstellung einer digitalen Signatur. Der Prüfwert wird verwendet, um die Integrität einer Nachricht zu sichern. Wenn zwei Nachrichten den gleichen Prüfwert ergeben, soll die Gleichheit der Nachrichten nach normalem Ermessen garantiert sein, unbeschadet gezielter Manipulationsversuche an den Nachrichten. Seite „Secure Hash Algorithm“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 16. Mai 2019, 10:27 UTC. URL:https://de.wikipedia.org/w/index.php?title=Secure_Hash_Algorithm&oldid=188627916. (Abgerufen: 16. Mai 2019, 12:23 UTC).

Bei hybriden Applikationen handelt es sich um eine Mischung aus webbasierten und nativen Apps. Sie werden mit Webtechnologien entwickelt, aber wie native Apps auf dem Gerät ausgeführt.[HWA+17] OWASP Top 10 mobilen Risikobereiche

2.3.1 Das Android-Manifest

Eine Android-Applikation verfügt grundsätzlich über eine Android-Manifest.xml, in die Inhalte im binären XML-Format implementiert sind. Das Android-Manifest.xml befindet sich im Stammverzeichnis der Applikation (siehe hierzu Abb. 2.10) und liefert dem Android-System alle nötigen Informationen zum Ausführen der Applikation. Im Android-Manifest werden alle wichtigen App-Komponenten, wie Activities, Services, Broadcast-Receiver oder auch Content-Provider, beschrieben und die Klassen deklariert, die die Komponenten implementieren. Ferner enthält es App-Metadaten, wie das Symbol oder auch die Versionsnummer und Informationen zu kompatiblen APIs und die Art des Speichers auf dem die Applikation installiert werden kann, da ab API Version 8, die Installation der Applikationen auch auf externen Speichern, wie z.B. SD-Cards, möglich ist. Des Weiteren sind im Manifest die Bibliotheken gelistet, mit denen eine Applikation verlinkt ist und die Permissions, über die die Applikation verfügen muss, werden definiert und deklariert. Dabei ist zu bedenken, dass einige der Berechtigungen möglicherweise erst zur Laufzeit von der App angefordert und somit nicht im Manifest deklariert werden. In der unteren Abbildung 2.11 wird beispielhaft eine AndroidManifest.xml dargestellt.[Spr17, Anda]

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.android.insecurebankv2" platformBuildVersionCode="22" platfo
rmBuildVersionName="5.1.1-1819727"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="22" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.READ_PROFILE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <android:uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <android:uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" android:maxSdkVersion="18" />
    <android:uses-permission android:name="android.permission.READ_CALL_LOG" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-feature android:glEsVersion="0x00020000" android:required="true" />
    <application android:theme="@android:style/Theme.Holo.Light.DarkActionBar" android:label="@string/app_name" android:icon="@mipma
p/ic_launcher" android:debuggable="true" android:allowBackup="true">
        <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" androi
d:windowSoftInputMode="adjustNothing|stateVisible" />
        <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin" />
        <activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:expor
ted="true" />
        <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin" />
        <activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:exp
orted="true" />
        <activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" andro
id:exported="true" />
        <provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="co
m.android.insecurebankv2.TrackUserContentProvider" />
        <receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
            <intent-filter>
                <action android:name="theBroadcast" />
            </intent-filter>
        </receiver>
        <activity android:label="@string/title_activity_change_password" android:name="com.android.insecurebankv2.ChangePassword" and
roid:exported="true" />
        <activity android:theme="@android:style/Theme.Translucent" android:name="com.google.android.gms.ads.AdActivity" android:confi
gChanges="keyboard|keyboardHidden|orientation|screenLayout|screenSize|smallestScreenSize|uiMode" />
        <activity android:theme="@style/Theme.IAPTheme" android:name="com.google.android.gms.ads.purchase.InAppPurchaseActivity" />
        <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version" />
        <meta-data android:name="com.google.android.gms.wallet.api.enabled" android:value="true" />
        <receiver android:name="com.google.android.gms.wallet.EnableWalletOptimizationReceiver" android:exported="false">
            <intent-filter>
                <action android:name="com.google.android.gms.wallet.ENABLE_WALLET_OPTIMIZATION" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

Abbildung 2.11: AndroidManifest.xml

2.3.2 Activities

Activities sind dafür zuständig, auf dem Bildschirm die Daten der Applikation grafisch darzustellen und ermöglichen dem Benutzer über Touch-Events[19], die Interaktion mit

[19] Definition „Touch-Event“ Touch-Events bieten die Möglichkeit, Finger- (oder Stift-) Aktivitäten auf Touchscreens oder Trackpads zu interpretieren, um eine hochwertige Unterstützung für berührungsbasierte Benutzeroberflächen bereitzustellen. Die Schnittstellen für Touch-Events sind APIs auf relativ niedriger Ebene, die zur Unterstützung anwendungsspezifischer Multi-Touch-Interaktionen wie z. B. einer Zwei-Finger-Geste verwendet werden können. Eine Multitouch-Interaktion beginnt, wenn ein Finger (oder Stift) zum ersten Mal die Kontaktfläche berührt. Andere Finger können anschließend die Oberfläche berühren und sich optional über die Berührungsoberfläche bewegen. Die Interaktion endet, wenn die Finger von der Oberfläche entfernt werden. Während dieser Interaktion empfängt eine Anwendung Touch-Events während der Start-, Bewegungs- und Endphase. Seite . Seite „Touch events“İn: MDN webdocs. Bearbeitungsstand: 15.Mai.2019, 15:39 UTC. URL:https://developer.mozilla.org/en-US/docs/Web/API/Touch_events. (Abgerufen: 16.Mai 2019, 15:23 UTC).

der Applikation. Dabei kann eine Applikation mehrere Activities haben, zwischen denen der Benutzer navigieren kann. Zusätzlich können Activities Abhängigkeiten zu anderen Applikationen und deren Activities aufweisen. Jede Activity ist im Android-Manifest deklariert und der Lebenszyklus einer Activity wird über den Activity-Manager verwaltet. Sobald eine Applikation gestartet wird, ist der Activity-Manager nicht nur für den Initiierungsprozess der Activity verantwortlich, sondern auch für das Managen der Navigation zwischen den Activities. Wechselt der Benutzer nur kurzweilig die Applikation, werden die Activities vom Manager erst mal in den Hintergrund geschoben und der Benutzer hat so die Möglichkeit, die zuletzt genutzte Activity nochmal aufzurufen. Wird aber über eine längere Zeitraum die Activity nicht genutzt, wird sie automatisch beendet. Jede Activity einer Applikation muss im Manifest registriert sein, damit sie vom System ausgeführt werden kann.[Spr17, Gar11]

2.3.3 Intents

Intents ermöglichen den Austausch von Nachrichten im Android-System und dienen der Kopplung von zur Laufzeit anfallenden Aufgaben mit entsprechenden Activities, Services oder Broadcasts. Mit Intents können Activities oder auch Services sowohl innerhalb einer Applikation als auch zwischen Applikationen angesprochen werden. Zudem enthalten Intents Informationen zu auszuführenden Vorgängen oder bei Broadcast-Receivers zu Einzelheiten eines aufgetretenen Ereignisses. Der Datentyp setzt sich aus zwei Hauptattributen, Data und Action, sowie drei (optionalen)-Attributen, Types, Catgories und Extras, zusammen. Mit dem Attribut „Action“ werden vorab Aufgaben definiert und einer Datenquelle zugewiesen, um damit dann bestimmte Activities ansprechen zu können. In der „Activity“ im Android-Manifest, ist festgelegt, welche Actions-und Data-Typen von einer Activity verarbeitet werden können. Darüber hinaus wird zwischen expliziten und impliziten Intents differenziert. Explizite Intents dienen dabei dem Informationsaustausch innerhalb einer Applikation. Dafür muss genau die Activity oder der Service angegeben werden, der angesprochen werden soll. Dagegen werden die impliziten Intents zur Kommunikation zwischen Activities von verschiedenen Applikationen genutzt. Über sie werden Informationen zur Art der Activity die ausgeführt werden soll und auch die Datenquelle geliefert.[Spr17, Gar11, And12]

2.3.4 Broadcast-Receiver

Broadcast-Receiver stellen eine Art Aktivität von Applikationen dar, für die bestimmte System-und Anwendungsereignisse registriert bzw. abonniert werden können. Alle registrierten Receiver werden dann von der Android-Laufzeitumgebung benachrichtigt, sobald das Ereignis eintritt, für das sie registriert wurden. Der Receiver ist sozusagen, als ein ruhender Code zu verstehen, der durch das Auftreten eines Ereignisses aktiviert wird, für das er registriert wurde. Um Broadcast-Ereignisse zu übermitteln, werden Intents

verwendet (siehe hierzu Kap.2.3.3). Im Android-System werden die ganze Zeit Ereignisse gebroadcast, wie z.B. der Eingang einer SMS, die Meldung über den Akku-Stand oder auch eingehende Anrufe. Die Receiver verfügen über keine grafische Benutzeroberfläche und laufen auch nicht aktiv im Speicher. Die Receiver werden lediglich über den Broadcast-Intent ausgelöst, wenn ein entsprechendes Ereignis eintritt, und der Receiver startet dann eine Activity oder einen anderen Dienst.[Spr17, GN14]

2.3.5 Content-Provider

Unter Content-Providern sind Komponenten zu verstehen, die Schnittstellen für den Austausch von Daten zwischen Applikationen bereitstellen. Wie schon zuvor erwähnt, wird für jede Applikation eine eigene Instanz in der Android-Runtime ausgeführt, wodurch alle Daten einer Applikation von anderen Applikationen isoliert werden. Die Intents können nur kleine Datenmengen austauschen und es gibt keinerlei Speicherbereich, auf den die Applikationen gemeinsam Zugriff haben. Aufgrund dessen werden Content-Provider genutzt, um die persistente Daten auszutauschen, die von den Applikationen mit SQL in relationalen Datenbanken vorgehalten werden.[Spr17, GN14]

2.3.6 Services

Services sind Android-Komponenten ohne grafische Benutzeroberflächen, die im Hintergrund ausgeführt werden und an den Lebenszyklus einer Aktivität gebunden sind. Sie werden für sich wiederholende und lange laufende Prozesse verwendet, wie z.B. Updates oder Downloads. Ebenso werden Services mit einer höheren Priorität ausgeführt als inaktive oder nicht sichtbare Activities, weshalb sie vom Android-System auch nicht so schnell abgebrochen werden. Services können aber auch so programmiert werden, dass sie automatisch wieder starten, falls sie vom Android-System vorzeitig beendet wurden.

2.3.7 Shared Preferences

Das Android-System bietet viele Methoden zum Speichern von Daten einer Applikation. Shared Preferences sind XML-Dateien und stellen eine dieser Methoden dar. Sie werden von der Applikation zur Speicherung von kleineren Datenmengen verwendet und können wichtige Informationen, wie z.B. zur Verschlüsselung von Datenbankinhalten, Passwörter für Nutzer, Policies oder URLs und IP-Adressen der Backend-Systeme, enthalten. Die XML-Dateien können sowohl im private-Mode, was bedeutet, dass sie nur für die Applikation selbst zur Verfügung stehen, als auch im shared-Mode vorliegen. Im shared-Mode sind auch andere Applikationen zum Zugriff auf die gespeicherten Daten berechtigt. Die Shared-Preferences-xml einer Applikation kann im Verzeichnis

/data/data/packageName/shared_prefs eingesehen werden. Die folgende Abbildung zeigt die Shared-Preference-xml der com.android.browser-Applikation.[Spr17]

```
root@generic:/ # ls -l /data/data/com.android.browser/shared_prefs/
-rw-rw---- u0_a14   u0_a14        287 2017-07-11 05:07 com.android.browser_preferences.xml
ser/shared_prefs/com.android.browser_preferences.xml                          <
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="enable_hardware_accel_skia" value="false" />
    <long name="last_recovered" value="1499763973459" />
    <boolean name="debug_menu" value="false" />
    <boolean name="last_paused" value="true" />
</map>
root@generic:/ #
```

Abbildung 2.12: Shared-Preference-xml

3 Sicherheitsgrundlagen und Schutzmaßnahmen von Android-Applikationen

„(...)Bösartige Android-App mit mehr als 100 Millionen Downloads“[Sch19c]

Wie einleitend bereits erwähnt, hat die Bedrohung von Android-Smartphones und deren Applikationen durch Malware, in den letzten Jahren erheblich zugenommen (siehe hierzu 2.4). Experten von G DATA betonen, dass die Gefahr, trotz eines leichten Rückgangs von Angriffen mit Schads-Apps im ersten Quartal 2019, weiterhin sehr groß sei. Insgesamt wurden im 1.Halbjahr 2019 1,85 Millionen[G D19b] Angriffe mit schadhaften Apps von GDATA gezählt. Nach einer Statistik von Kaspersky, die die weltweite Verbreitung von Mobile Malwarearten des Jahres 2018 aufzeigt (siehe hierzu 3.1, gehörten danach RiskTool bzw. (eine Mischform zwischen Ransomware und Hardware Damaging Malware) mit 52,06%, Trojan-Dropper mit 17,21% und Adware mit 8,46% zu den TOP 3 Malwarearten, die sich die Anwender mit den Applikationen auf ihre mobilen Geräte installierten.

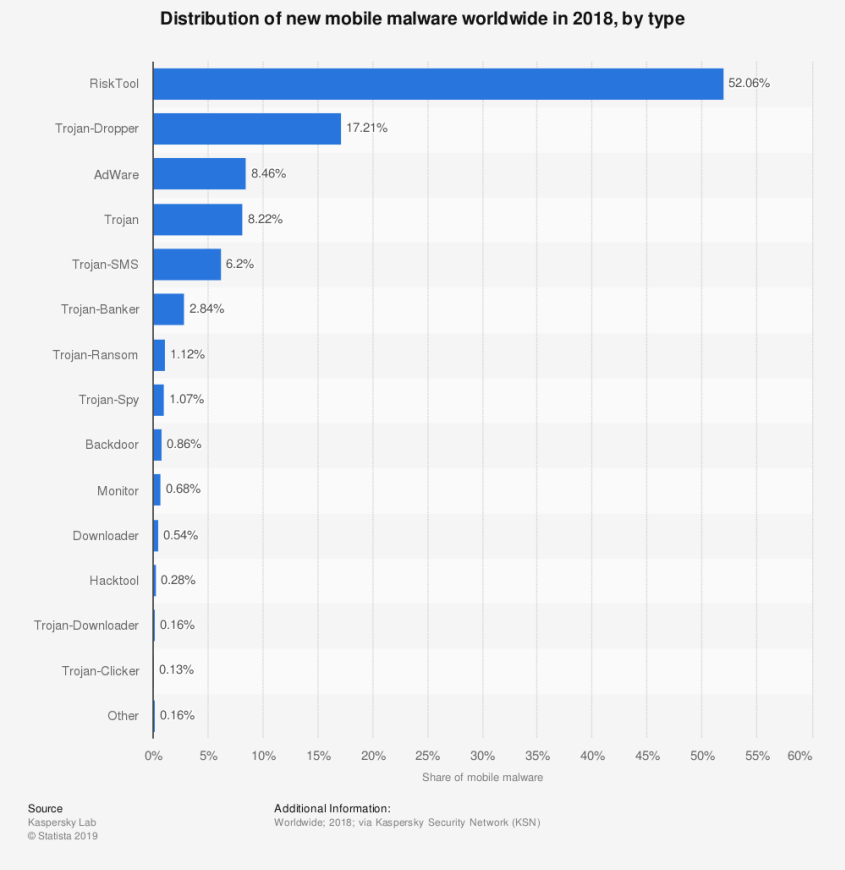


Abbildung 3.1: Weltweite Verbreitung von mobilen Malware-Typen im Jahr 2018

Für ein besseres Verständnis der einzelnen Faktoren bei Angriffen oder auch der Infiltration durch mobile Malware, wird folgend unter Einbezug der Ausführungen zur Malware-Forensik von Christian Hummerts aus dem Buch „Forensik in der digitalen Welt“, Edith Hubers Buch „Cybercrime“ und John Aycocks Buch „Computer Viruses and Malware“ kurz der Begriff Malware erläutert und eine grobe Kategorisierung der verschiedenen Malwarearten anhand charakteristischer Merkmale vorgenommen. Aufbauend darauf werden im folgenden Kapitel 3.2, die Hauptangriffsvektoren umrissen, die einen Angriff mit Malware begünstigen.
Abschließend werden wesentliche Schutzkomponenten aus dem Android-Sicherheitsmodell dargestellt.
Die Identifizierung möglicher Angriffsvektoren und die Kenntnisse zum Android-Sicherheitsmodell sollen dem Sicherheitsexperten bei der Eingrenzung und Bestimmung der zu testenden Komponenten und bei der Erarbeitung einer Vorgehensweise für die Sicherheitsanalyse unterstützen.

3.1 Definition und Klassifizierung von Malware

Mit dem Begriff Malware, der eine Wortkomposition der beiden englischen Wörter „malicious= bösartig/boshaft“ und „Software= Computerprogramm“ darstellt, werden im Allgemeinen Schadprogramme oder auch Werkzeuge bezeichnet, die zum Angriff, der Infiltration oder auch Missbrauch von Computersystemen-bzw. Programmen, verwendet werden. Malware kann in vielfältiger Form angefangen mit Skripten, Programmcode über ausführbare Dateien oder auch als Software auftreten.[Hub03, LSLS17]
Über Malware wollen Angreifer Zugang zum Smartphone und den darauf laufenden Applikationen erhalten, um z.B. persönliche und vertrauliche Daten, wie Bankdaten oder auch Unternehmens-oder Staatsgeheimnisse ausspähen und stehlen.
Üblicherweise wird Malware je nach Art ihrer Verbreitung, ihres Payloads oder auch entsprechend bestimmter Eigenschaften, wie Selbstreplikation oder parasitärem Verhalten, in eine Reihe von Klassen unterteilt. Allerdings besteht bezüglich der Kategorisierung von Malware derzeit kein Konsens in der gängigen Literatur, was sich u.a. auf die Masse und vor allem Vielfältigkeit der Malwarearten und auch ihre Hybridisierung heutzutage zurückführen lässt.[LSLS17, Hub03]

Folgend werden einzelne Arten von Malware und charakteristische Merkmale vorgestellt und anhand einiger Beispielen veranschaulicht.

1) **Virus**: Der *Virus* bezeichnet ein Computerprogramm,das sich durch das Infizieren fremder Programme oder Dateien reproduziert. Dabei kann der Virus sein Wirtsprogramm z.B. ändern, schädigen, mit eigenem Code überschreiben oder auch mit Sprungbefehlen vom Wirtsprogramm zum Viruscode wechseln. Viren können als Datei-, Link-,Boot-oder Makroviren auftreten.
   Ein gutes Beispiel für ein Virus insbesondere bei mobilen Applikationen, stellen

sogenannte Fake-Apps dar. Nachdem sie unwissentlich vom Anwender geladen wurden, sind sie in der Lage, eine andere Applikation einfach durch die Fake-Version zu ersetzen.[Ayc06, LSLS17, Hub03]

2) **Wurm**: Ein *Wurm* ist ein Programm, das sich aktiv über Computernetzwerke oder Wechselmedien mit oder ohne menschliche Interaktion ausbreitet. Dabei versucht der Wurm Schwachstellen, Sicherheitslücken oder Programmierfehler auszunutzen. Im Gegensatz zum Virus, benötigt der Wurm keine Wirtsdatei und verbreitet sich nach einmaliger Aktivierung völlig selbstständig. Weitere Arten von Würmern sind u.a. E-Mail-, Peer-to-Peer-oder auch IRC-Würmer.
   Ein bekannter Smartphone-Wurm war z.B. der Android ADB.Miner. Dieser Wurm nutzte eine Sicherheitslücke in der Android-Debug-Bridge[1] als Schnittstelle aus, um sich auf dem Smartphone zu verbreiten. Sobald sich der Wurm auf dem Smartphone befand, nahm dieser ein Verbindung zum Internet auf und suchte nach offenen TCP-Ports 5555, um weitere Geräte zu kompromittieren.[G D19a, Ayc06, Hub03]

3) **Trojaner**: Der *Trojaner* oder auch Trojanisches Pferd[2] versteht sich als ein Programm mit versteckter Schadfunktion. Die scheinbar nützliche Anwendung entfaltet meist bei der Aktivierung durch den Anwender z.B. ihren Schadcode oder öffnet ein weiteres schädliches Programm. In der Regel dienen Trojaner zum Ausspähen, Protokollieren von Informationen oder auch der Fernsteuerung des infiltrierten Systems. Im Gegensatz zu Viren und Würmer, kann sich ein Trojaner nicht selbstständig verbreiten. Ins System gelangt die unerwünschte Malware sehr oft z.B. über schadhafte E-Mail-Anhänge oder Links.
   Ein derzeit sehr bekannter Smartphone Trojaner, stellt die Trojaner-App CamScanner, ein Phone PDF Creator, dar. Die Applikation dient als Downloader für den Schadcode Trojan-Dropper.AndroidOS.Necro.n.. Ist die Malware aktiviert, können die Angreifer u.a. persönliche Daten ausspähen und kostenpflichtige Abos auslösen.[Sch19a, LSLS17, Hub03]

[1] Die Android-Debug-Bridge (oft mit adb abgekürzt) ist eine Software-Schnittstelle für das Android-System, die genutzt werden kann, um von einem Computer per USB-Kabel auf ein Android-Smartphone zugreifen und Befehle ausführen zu können.Android-Debug-Bridge. (13. Oktober 2018). Android Wiki, . Abgerufen am 30. August 2019, 12:59 von https://www.droidwiki.org/w/index.php?title=Android_Debug_Bridge&oldid=17559.

[2] „Trojanisches Pferd“ Der Name ist metaphorisch vom Trojanischen Pferd der Mythologie abgeleitet. Der Legende nach konnte die unbezwingbare Stadt Troja nur durch einen Trick eingenommen werden: Die Angreifer präsentierten den Bewohnern ein riesiges Holzpferd als Friedensangebot. Im Inneren verbargen sich feindliche Soldaten, die so Zugang zum Stadtinneren erlangten. Seither ist der Begriff „Trojanisches Pferd“ gleichbedeutend mit „etwas Vortäuschen“. Analog dazu beabsichtigt ein Trojaner, als etwas Nützliches getarnt, durch den Angegriffenen selbst in den geschützten Bereich des Systems gebracht zu werden. Seite „Trojanisches Pferd (Computerprogramm)“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 16. August 2019, 20:21 UTC. URL:https://de.wikipedia.org/w/index.php?title=Trojanisches_Pferd_(Computerprogramm)&oldid=191405048, (Abgerufen: 30. August 2019, 14:01 UTC)

4) **Backdoor/Rootkits**: Eine *Backdoor* „dt. Hintertür“ stellt, wie der deutsche Name schon verrät, einen zusätzlichen bzw. alternativen, dem Anwender verborgenen, Zugang zum System oder Programmen dar. Mit der Backdoor können Angreifer Schutz-und Sicherheitsmechanismen, wie Virenscanner oder Firewalls umgehen und sich unautorisiert Zugang zu einem System verschaffen. Ähnlich wie Trojaner, können Backdoors u.a. über manipulierte E-Mail-Anhänge oder auch Links in das System gelangen. Über eine Backdoor hat der Angreifer unter Umständen über einen ungewissen Zeitraum, unentdeckt Zugriff zu sicher geglaubten Systemen.
Zu einer der bekanntesten Backdoors bei Android-Systemen, zählte Android-Hummingbad. Mit Hummingbad wurde eine Backdoor mit Root-Rechten auf den Smartphones der Anwender installiert, wodurch die Angreifer frei Applikationen installieren und deinstallieren konnten. Des Weiteren wurde eine Applikation installiert, die mittels Klickbetrug[3] vom Anwender Geld erschlich.[LSLS17, Hub03]

Rootkits stellen ein Spezialfall von Backdoors dar und sind als eine Art Werkzeugkiste zu verstehen, um sich unerkannt auf einem System Root-Rechte zu verschaffen. Dazu kann unter anderem Programmcode und Skripte gezählt werden, mit denen unerwünschte Prozesse vor Anwendern versteckt werden sollen oder auch andere Formen von Malware. Hauptziel besteht für den Angreifer auf jeden Fall darin, Root-Rechte zu erhalten. Rootkits können unterteilt werden in Applikation-, Kernel-, Userland-oder auch Speicherrootkits. [LSLS17, Hub03]
Im Jahr 2016 machte ein Android-Updater einer chinesischen Firma Schlagzeilen, da dieser vollen Root-Zugang besaß und die Ausführung der Applikation auf dem Gerät dem Nutzer verborgen war. Zudem ließ der Updater, da er die Daten der Nutzer unverschlüsselt übertrug, Man-in-the-Middle-Angriffe[4] zu, wodurch ungehindert persönliche Daten von Nutzern, wie die IMEI[5] oder Telefonnum-

[3] Als „Klickbetrug“ bezeichnet man eine Art des Internetbetrugs, die vorrangig auf pro Klick vergütete Werbebannereinblendungen (vergleiche Pay per Click) abzielt. Dabei werden kommerzielle Werbeflächen geklickt oder solche Klicks simuliert, um dahinterliegende Abrechnungssysteme gezielt zu manipulieren.Seite „Klickbetrug“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 27. November 2017, 15:34 UTC. URL:https://de.wikipedia.org/w/index.php?title=Klickbetrug&oldid=171442854, (Abgerufen: 30. August 2019, 16:03 UTC)

[4] Ein Man-in-the-Middle-Angriff (MITM-Angriff), auch Janusangriff (nach dem doppelgesichtigen Janus der römischen Mythologie) genannt, ist eine Angriffsform, die in Rechnernetzen ihre Anwendung findet. Der Angreifer steht dabei entweder physisch oder – heute meist – logisch zwischen den beiden Kommunikationspartnern, hat dabei mit seinem System vollständige Kontrolle über den Datenverkehr zwischen zwei oder mehreren Netzwerkteilnehmern und kann die Informationen nach Belieben einsehen und sogar manipulieren. Die Janusköpfigkeit des Angreifers besteht darin, dass er den Kommunikationspartnern vortäuscht, das jeweilige Gegenüber zu sein. Seite „Man-in-the-Middle-Angriff“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 18. Juli 2019, 05:34 UTC. URL:https://de.wikipedia.org/w/index.php?title=Man-in-the-Middle-Angriff&oldid=190523140, (Abgerufen: 31. August 2019, 15:23 UTC)

[5] Die International Mobile Station Equipment Identity (IMEI) ist eine eindeutige 15-stellige Seriennummer, anhand derer jedes GSM- oder UMTS-Endgerät weltweit eindeutig identifiziert werden kann.Dual-SIM-Handys besitzen zwei IMEI-Nummern. Seite „International Mobile Equipment Identity“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 23. Juni 2019, 08:02 UTC. URL:https://de.wikipedia.org/w/index.php?title=International_Mobile_Equipm

mern abgegriffen werden konnten. Die Update-App wurde daraufhin als Rootkit eingestuft.[Eik16]

5) **Spyware/Adware**: *Spyware* versteht sich als Spionagesoftware, deren Ziel es ist, vom Anwender heimlich Informationen zu sammeln. Bei den Informationen kann es sich u.a. um persönliche Daten, wie Aufenthaltsort des Anwenders, Bankdaten oder auch Daten zum Surfverhalten handeln. Spyware gelangt meist über andere vom Anwender geladene Applikationen auf das Smartphone und leitet die gesammelten Informationen über eine hergestellte Internetverbindung an Dritte weiter.
Ein Beispiel stellt hier die von GDATA in einer Kurzstudie untersuchte Android-Spyware „Busham.A.“ dar. Busham.A. ist u.a. neben Zugriffe auf Kamera- und Mikrofonfunktionen, dazu fähig, die komplette Whatsapp-Datenbank, das Adressbuch, den Browserverlauf oder die Lesezeichen auszulesen. Die gesammelten Informationen, werden dann über eine von der App hergestellten Internetverbindung, an Dritte gesendet.[LSLS17, Wik19g, Tre18]

   Adware, als verwandte Malwareform von Spyware, sammelt Adware wie Spyware auch Informationen über den Anwender, hat aber die zusätzliche Funktion, entsprechend der über den Nutzer gesammelten Informationen, auf dem Smartphone Werbung anzuzeigen. Adware gelangt meist mit freier Software oder z.B. Spiele-Apps auf das Smartphone und zeigt die Werbung in Pop-Up-Fenstern oder kurze Werbeeinblendungen an. In der Regel ist sich der Anwender nicht darüber bewusst, dass die Adware nicht nur personalisierte Werbung anzeigt, sondern im Hintergrund eine Verbindung mit dem Internet aufnimmt und Informationen über den Anwender an Dritte versendet.[LSLS17, Wik19a]
   Aktuelle Beispiele für Adware stellen bspw. die Applikationen „Super Selfie“ oder „Cos Camera“ dar, über die nach einer Analyse des Unternehmens Trend Micro, Adware auf die Smartphones der Nutzer geschleust wurde. Die eingeschleuste Adware zeigte den Nutzer aufdringliche Werbung an, blockierte die Anrufunktion auf dem Smartphone und sammelte Nutzerinformationen.

6) **Botnetz**: Ein *Botnetz* setzt sich aus mehreren Schadprogrammen den Bots, eine Abkürzung des Wortes „Robot“ zusammen, deren Aufgaben es ist, die Ressourcen und Daten von gekaperten Systemen in einem Netzwerk freizugeben und abzugreifen. Die infiltrierten Systeme werden z.B. zur Durchführung von DDos-Attacken, zum Bitcoin-Mining, zum Massen-Spam-Versenden oder auch als Proxys verwendet. Die Verbreitung der Bots erfolgt meist über anderen Arten von Malware, wie Trojanern oder auch Exploits und werden über einen Command-and-Control-Server kontrolliert.[LSLS17, Wik19b]

ent_Identity&oldid=189789404,(Abgerufen: 30. August 2019, 16:22 UTC)

Eines der weltweit größten und noch aktive Botnetzen, ist das Botnetz Mirai. Mirai ist eine Linux-Schadsoftware und nutzt Sicherheitslücken in IoT-Geräten[6], wie Waschmaschinen, Toastern oder Staubsaugern, um ihren Schadcode zu verbreiten.[Wik19d]

7) **Hardware Damaging Malware/Riskware**: Bei dieser Form von Malware, handelt es sich explizit um Schadprogramme-oder Schadcode die Hardwarekomponenten, wie Festplatten beschädigen, wichtige Systemkomponenten, wie das BIOS löschen oder auch die Firmware flashen und so das Computersystem in seinem Betrieb stark einschränken oder gar zerstören.[LSLS17]
Zu dieser Kategorie von Malware zählt z.B. der Trojaner Loapi, der neben dem Einschleusen von Werbung und der Durchführung von DDos-Attacken, in hohem Maße Cryto-Mining auf den befallenen Smartphones betrieben hat und dadurch die Akkus der Geräte überhitzte und zerstörte.[Rix17]

8) **Ransomware**: *Ransomware* stellt ein Art von Malware dar, die sensible Daten verschlüsselt oder stiehlt und eine Zahlung verlangt, um sie zu entschlüsseln oder zurückzugeben. Mobile Ransomware begann mit einem „Sperrbildschirm“, auf dem das Opfer oft beschuldigt wurde, illegale Inhalte angesehen zu haben und Geld für die Entsperrung des Geräts zu verlangen.[Wik19f] Es hat sich zur Malware entwickelt, die alle Dateien auf dem mobilen Geräten verschlüsselt und eine Zahlung verlangt, um den Schlüssel zum Entschlüsseln für den Zugriff auf die Dateien bereitzustellen.[LSLS17]
Eine der aktuellsten Ransomwares, die insbesondere Android-Smartphones befallen, zählt Android/Filecoder.C. Die Ransomware nutzt die Kontaktlisten der Anwender und verbreitet sich über SMS auf weitere Smartphones. Sobald die Schad-SMS versendet wurde, verschlüsselt sie die Daten der Anwender und verlangt ein Lösegeld.[Ste19b]

9) **Exploit**: Als *Exploit* wird ein schadhafter Programmcode oder Befehlsfolge verstanden, mit dem Angreifer systematisch Sicherheitslücken oder Fehlfunktionen an einem Computersystem ausnutzen können. So wollen Angreifer unentdeckt Zugriff zum System bekommen, um Anwender auszuspähen, Daten zu manipulieren oder Datendiebstahl zu begehen. Je nach Angriffsart können Exploits nochmal unterteilt werden, in Remote-, Lokal-, Denial-of-Service-, SQL-Injection-, Command-Execution-oder Zero-Day-Exploits.[Hub03, LS17] Ein bekannter Exploit für Android-Systeme, ist z.B. StageFright, benannt nach der Medienwieder-

[6]Definition:„Internet of Things“ Das Internet der Dinge (IdD) (auch: „Allesnetz“;[1] englisch Internet of Things, Kurzform: IoT) ist ein Sammelbegriff für Technologien einer globalen Infrastruktur der Informationsgesellschaften, die es ermöglicht, physische und virtuelle Gegenstände miteinander zu vernetzen und sie durch Informations-und Kommunikationstechniken zusammenarbeiten zu lassen. Seite „Internet der Dinge“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 26. August 2019, 09:27 UTC. URL:https://de.wikipedia.org/w/index.php?title=Internet_der_Dinge&oldid=191692324, (Abgerufen: 31. August 2019, 19:29 UTC)

gabebibliothek, die eine kritische Sicherheitslücke aufweist. So können Angreifer mittels MMS-Nachrichten z.B. das Smartphone eines Anwenders infiltrieren und persönliche Daten stehlen. [Inf19]

10) **Phishing/Pharming**: Über *Phishing* (Wortzusammensetzung aus den englischen Wörtern „Password“ und „Fishing“) wollen Angreifer z.B. über gefakte E-Mails und Websites, Anwender dazu verführen, ihre Daten, wie Passwörter, PINs oder andere persönliche Daten preiszugeben.[Hub03]
    Im Gegensatz zum Pishing, das Anwender dazu verleiten möchte, gefälschte Websites zu besuchen, leitet *Pharming*, das eine Weiterentwicklung des Phishings darstellt, die Anwender durch die Manipulation der DNS-Abfrage des Webbrowsers auf die gefälschte Webseite um.[Wik19e]
    Erst im Juni dieses Jahres fanden Entwickler von ESET-Research[7] Phishing und Pharming Apps, die z.B. die SMS-basierte Zwei-Faktor-Authentifizierung umgehen und Einmalpasswörter zur Authentifizierung des Dienstes abfangen. Eine andere Fake-App ahmt eine türkische Kryptowährungsbörse nach und stiehlt, sobald die Nutzer sich anmelden, die Login-Daten.[Ste19a]

11) **Dropper**: Bei *Dropper* handelt es sich um Malware, die erst wenn sie sich auf dem System befindet, Verbindung zum Internet aufnimmt und dann den Schadcode nachlädt mit dem das Gerät infiziert wird. Dropper können je nach den Eigenschaften, die sie mitbringen, für ein Zielsystem abgestimmte Malware installieren.[LSLS17]
    Als Beispiel für einen Dropper wird an dieser Stellen nochmal auf den Trojan-Dropper.AndroidOS.Necro.n verwiesen, der sich in der App-CamScanner verbarg und eine Mischung aus Trojaner und Dropper darstellt (siehe hierzu Punkt 3) Trojaner).

3.2 Klassifizierung von Angriffsvektoren für Android-Smartphones

Wie in den vorangegangenen Kapiteln ansatzweise schon aufgezeigt wurde, kann Android-Malware mobile Geräte auf verschiedene Weise infizieren, z.B. durch Klicken auf schädliche Links in E-Mails, durch SMS-Nachrichten, durch das Aufrufen manipulierter Websites oder auch durch das Herunterladen einer infizierten Anwendung aus dem Google Play Store oder dem App Store eines Drittanbieters.
Folgend wird in Anlehnung an Becher und Spreitzenbarth versucht, bestehende Angriffsvektoren zu klassifizieren und zu erläutern, wobei besonderer Fokus auf App-basierte

[7] ESET ist ein Unternehmen für Sicherheitssoftware, das 1992 in Bratislava (Slowakei, damals Tschechoslowakei) gegründet wurde. Es befindet sich in Privatbesitz und hat Niederlassungen in mehreren Ländern. Die exklusive Distribution von ESET-Produkten in Deutschland wird von der ESET Deutschland GmbH in Jena wahrgenommen. Seite „ESET“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 9. Mai 2019, 09:46 UTC. URL:https://de.wikipedia.org/w/index.php?title=ESET&oldid=188386607, (Abgerufen: 1. September 2019, 18:00 UTC).

Angriffe, unter Einbezug der OWASP Top 10 Sicherheitsrisiken für mobile Geräte und ihre Applikationen, gelegt wird.[Spr17, BFH+11]

3.2.1 Hardware-basierte Angriffe

Unter diese Kategorie fallen Angriffe auf Hardwarekomponenten des Smartphones, wie z.B. der Speicher, der Akku oder auch der Prozessor. Angriffe auf die Hardwarekomponenten eines Smartphones können entweder über den direkten physischen Zugriff oder u.a. über Hardware Damaging Malware und Trojaner, stattfinden.[Spr17, BFH+11]

3.2.2 Device-basierte Angriffe

Das Android-Betriebssystem enthält Schwachstellen, die mit Malware u.a. lokal oder remote ausgenutzt werden können. Die Ausnutzung solcher Sicherheitslücken kann zu Privilege Escalations[8] z.B. auf Kernel-Ebene führen. Sobald das Smartphone gerootet ist, hat der Angreifer vollen Zugriff auf das Gerät und kann beliebig Anwendungen laden oder auch löschen sowie das Betriebssystem manipulieren.[Spr17, BFH+11]

3.2.3 Web-basierte Angriffe und Angriffe auf Infrastrukturkomponenten

Diese Art von Angriffen richten sich auf die Netzwerkumgebung, in die das mobile Gerät ggf. eingebunden ist. Unter anderem kommt Malware zum Überwachen eines Back-End-Systems zum Einsatz in Form von MITM-Angriffen oder es werden Angriffe auf die Proxy-Server durchgeführt.[Spr17, BFH+11]
Mit Rogue-Access-Points sind Angreifer unter Umständen in der Lage, Sicherheitslücken in der SSL-Konfiguration zu durchbrechen und mit gefälschten Root-CA[9], Zugriff auf den sonst verschlüsselten Datenverkehr zu erhalten.[Eil18]

[8] Als Rechteausweitung, auch Rechteerhöhung, Privilegienerweiterung oder Privilegien-Eskalation genannt, bezeichnet man die Ausnutzung eines Computerbugs bzw. eines Konstruktions- oder Konfigurationsfehlers einer Software mit dem Ziel, einem Benutzer oder einer Anwendung Zugang zu Ressourcen zu verschaffen, deren Nutzung mit eingeschränkten Rechten nicht möglich ist.Seite „Rechteausweitung“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 9. Juni 2019, 21:55 UTC. URL:https://de.wikipedia.org/w/index.php?title=Rechteausweitung&oldid=189406279, (Abgerufen: 2. September 2019, 11:52 UTC).

[9] Definition: „Root-CA“ In cryptography and computer security, a root certificate is a public key certificate that identifies a root certificate authority (CA).[1] Root certificates are self-signed and form the basis of an X.509-based public key infrastructure (PKI). Either it has matched Authority Key Identifier with Subject Key Identifier, in some cases there is no Authority Key identifier, then Issuer string should match with Subject string (RFC5280). For instance, the PKIs supporting HTTPS[2] for secure web browsing and electronic signature schemes depend on a set of root certificates. Wikipedia contributors, Root certificate,"Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Root_certificate&oldid=911976951, (accessed September 2, 2019).

3.2.4 App-basierte Angriffe

Angriffe mit Malware über manipulierte Android-Apps bzw. Fake-Apps oder die Ausnutzung von Sicherheitslücken in Android-Apps, stellen mitunter den beliebtesten Angriffsvektor für Cyberkriminelle dar. Dabei kann die Malware über vielfältige Quellen ohne Wissens des Nutzers auf das Smartphone gelangen.
Die meisten Applikationen verfügen über eine Client-Server-Architektur[10], was bedeutet, dass angefangen beim Client über das API-Backend bis hin zum Server und einer dahinter liegenden Datenbank, viele Sicherheitslücken und damit auch Einstiegspunkte, bestehen können.[Spr17, HWA+17]

1. **Angriffsvektor Client**: Unter anderem speichern immer mehr Applikationen enorme Mengen an Daten auf dem Gerät. Dabei kann es sich um persönliche oder gar vertrauliche Daten handeln, die z.B. im lokalen Speicher, in den Shared-Preferences oder den SQLite-Datenbanken, unverschlüsselt hinterlegt werden. Erhält ein Angreifer über eine schadhafte Applikation Zugriff auf das Smartphone, ist es ihm möglich, auf die Speicher zuzugreifen und die Daten auszulesen, zu manipulieren oder zu löschen.
   Des Weiteren tauscht eine Applikation viele Daten mit dem Backend-Server aus. Auch hier können Angreifer, wenn der Datenverkehr nicht hinreichend gesichert und verschlüsselt ist, vertrauliche Daten, wie Benutzernamen, Kennwörter, PINs oder Authentifizierungstoken, sniffen. Auch Codierungsfehler in einer Applikation können schwerwiegende Folgen haben. So könnte sich ein Angreifer über Client-side-Injection[11] Zugriff zur laufenden Applikation erlangen, um an die Daten eines Anwenders zu gelangen. Über einen Zugriff auf die Binärdateien könnte er auch die Applikation dekompilieren und sich Zugriff auf ggf. fest codierte Anmeldeinformationen erlangen.[HWA+17]

2. **Angriffsvektor Backend**: Der hinter einer der Applikation stehende Webdienst

[10] Das Client-Server-Modell (auch Client-Server-Konzept, -Architektur, -System oder -Prinzip genannt) beschreibt eine Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Die Aufgaben werden von Programmen erledigt, die in Clients und Server unterteilt werden. Der Client kann auf Wunsch einen Dienst vom Server anfordern (z. B. ein Betriebsmittel). Der Server, der sich auf demselben oder einem anderen Rechner im Netzwerk befindet, beantwortet die Anforderung (das heißt, er stellt im Beispiel das Betriebsmittel bereit); üblicherweise kann ein Server gleichzeitig für mehrere Clients arbeiten. Seite „Client-Server-Modell“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 16. Juli 2019, 13:14 UTC. URL:https://de.wikipedia.org/w/index.php?title=Client-Server-Modell&oldid=190479595, (Abgerufen: 3. September 2019, 09:47 UTC).

[11] Definition: „Code injection“ is the exploitation of a computer bug that is caused by processing invalid data. Injection is used by an attacker to introduce (or ïnject") code into a vulnerable computer program and change the course of execution. The result of successful code injection can be disastrous, for example by allowing computer worms to propagate. Wikipedia contributors. Code injection. Wikipedia, The Free Encyclopedia. July 7, 2019, 22:58 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Code_injection&oldid=905253640., Accessed September 3, 2019.

kann genauso wie Webanwendungen diverse Sicherheitslücken aufweisen. Dabei treten u.a. häufig Schwachstellen bei der Authentifizierung und Autorisierung, dem Sessionmanagement, der Eingabevalidierung oder aufgrund unzureichender Verschlüsslungsmechanismen auf.

- *Unsichere Authentifizierung*: Die Authentifizierung stellt einen wesentlichen Bestandteil der Architektur mobiler Applikationen dar und kann bei Fehlern in der Implementierung dazu führen, dass der Nutzer nicht hinreichend authentifiziert wird. Wenn z.B. bei der Authentifizierung mit dem Backend leicht zu fälschende Kennungen oder IDs, wie die IMEI oder auch GPS-Daten verwendet werden, können Angreifer sich leicht Zugriff verschaffen und die Daten der Applikation auslesen.[HWA+17, Spr17]
- *Schlechtes Sessionmanagement*: Generell wird bei Sessionmanagement ein Authentifizierungstoken generiert, der eindeutig zuordenbar ist und für den Rest der Sitzung verwendet wird. Meist werden diese Tokens nach Beendigung einer Sitzung aber nicht ordnungsgemäß gelöscht, was einem Angreifer ermöglicht, Tokens zu stehlen und sich als authentifizierter Nutzer gegenüber dem Backend auszugeben.[HWA+17, Spr17]
- *Unzureichende Eingabevalidierung*: Die Eingabevalidierung z.B. von Benutzerdaten wird durchgeführt, um sicherzustellen, dass nur ordnungsgemäß formatierte Daten und vor allem keine schadhaften Daten in ein System gelangen. Wenn Steuerelemente, die die Eingabevalidierung regeln, nicht richtig implementiert sind, haben Angreifer die Möglichkeit, sich über SQL-Injection oder auch Cross-Site-Scripting, Zugang zu verschaffen.[HWA+17, Spr17]
- *Unzureichende Kryptographie*: Die Verwendung unzureichender bzw. nicht sicherer kryptografischer Verfahren zum Verschlüsseln der Daten im Backend und des Datenverkehrs zwischen Backend und Client, stellen einen weiteren Einstiegspunkt für Angreifer dar. Angreifer können ggf. einfach Daten, die ausgetauscht werden entschlüsseln oder sich Zugriff zum Backend-Server verschaffen.[HWA+17, Spr17]

Weitere Angriffsvektoren bzw. eine gute und aktuelle Zusammenfassung dieser, können den Top 10 Mobile Risks der OWASP[12]entnommen werden. Auf der Basis von aktuellen Schwachstellen und der Bedrohungslandschaft für mobile Applikationen ermittelt die

[12]Das Open Web Application Security Project (OWASP) ist eine Non-Profit-Organisation mit dem Ziel, die Sicherheit von Anwendungen und Diensten im World Wide Web zu verbessern. Durch Schaffung von Transparenz sollen Endanwender und Organisationen fundierte Entscheidungen über wirkliche Sicherheitsrisiken in Software treffen können. Seite „Open Web Application Security Project“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 22. März 2019, 12:51 UTC. URL:https://de.wikipedia.org/w/index.php?title=Open_Web_Application_Security_Project&oldid=186815331. (Abgerufen: 31. August 2019, 11:40 UTC)

OWASP in 2-3 Jahres Abständen Sicherheitsrisiken mit dem Schwerpunkt auf mobile Anwendungen, die im folgenden kurz dargestellt werden sollen.[Spr17, Car16]

Abbildung 3.2: OWASP Top 10 Mobile Risks

M1 Unsachgemäße Plattformnutzung: Die unsachgemäße Nutzung der Plattform deckt hauptsächlich den Missbrauch von Plattformfunktionen oder die Nichtverwendung von Plattformsicherheitskontrollen ab, die von der Plattform bereitgestellt und dokumentiert werden. Hiervon können verschiedene Android-Systemkomponenten betroffen sein, wie z.B. Intents oder Plattform-Permissions. Ein Angreifer könnte sich über den Browser mit Cross-Site-Scripting (XSS) Zugang zum Smartphone verschaffen. Zur Abwendung solcher Angriffe sollten sichere Codierung-und Konfigurationsmethoden angewendet werden. Des Weiteren treten bei Android-Apps häufig Fehler bei der Vergabe von Datei-Permissions auf. Unter anderem können die Permissions „mode_world_readable“ und „mode_world_writeable“vollen Dateizugriff gewähren.[Car16, cod17]

M2 Unsicherer Datenspeicher und unbeabsichtigter Datenverlust: Die Kategorie „Unsicherer Datenspeicher“ deckt Schwachstellen oder Risiken ab, die sich erstens auf vertrauliche Informationen beziehen, die von der Applikation gespeichert werden, und zweitens auf die Art und Weise, wie diese Informationen auf dem Smartphone gesichert wurden. Dabei liegt der Fokus auf der Speicherung vertraulicher Informationen in den Anwendungsprotokollen, im Anwendungsspeicher oder im dekompilierten Code oder auch die Speicherung vertraulicher Informationen auf Medien wie SD-Cards, Anwendungsdateien oder lokale SQLite-Datenbanken. Ebenso umfasst diese Kategorie jede Art von ungewollten Datenverlust, wie z.B. das Cashing von URLs.[Car16, Spr17]

M3 Unsichere Kommunikation: Diese Schwachstelle beschäftigt sich mit schlechtem Handshaking, falschen oder veralteten SSL-Versionen und der mangelnden Verschlüsselung der zu übertragenden Daten. Die Sicherung des Datenaustausches ist für die Sicherheit der privaten bzw. vertraulichen Daten von größter Bedeutung. Daher sollten Zertifikate mit möglichst langen Schlüsseln, ab 2048 Bit, verwendet werden und die Zertifikate sollten von vertrauenswürdigen Zertifizierungsstellen stammen. Zudem sollten sichere Protokolle z.B. TLSv3 und starke Cipher-Suites aus der SHA2-Familie bevorzugt verwendet werden, um eine sichere Kommunikation zu gewährleisten.[Car16, Chr18]

M4 Unsichere Authentifizierung: Durch unsichere Authentifizierung können anfällige Authentifizierungsschemata ausgenutzt werden, indem die Authentifizierung gefälscht oder umgangen wird. Eine gängige Methode bei Angreifer ist die Umgehung der Authentifizierung unter Nutzung von Malware oder Botnetzen. So kann der Angreifer vertrauliche und verschlüsselte Kommunikation aufdecken und die Kommunikationspartner abhören.
Die Nutzung schwacher Authentifizierungsschemata kann zu Fehlfunktionen der Applikation führen. So kann es passieren, dass die Applikation den Nutzer nicht mehr richtig identifiziert oder die Benutzeraktivitäten nicht mehr vollumfänglich überwacht und protokolliert. Hinweise auf die Nutzung unsicherer Authentifizierung, sind die anonyme Ausführung von API-Serviceanforderungen, ohne Bereitstellung eines Zugriffstokens, die Verwendung von hardcodierten Passwortdaten, die Verwendung schwacher Kennwortrichtlinien oder auch die Verwendung von Touch-ID.[Car16, Spr17]

M5 Unzureichende Kryptographie: Unzureichende bzw. unsichere Verwendung von Kryptographie ist eine häufige Schwachstelle bei mobilen Applikationen. Aufgrund der schwachen Verschlüsselungsalgorithmen oder Fehlern im Verschlüsselungsprozess kann der verschlüsselte Code oder die vertraulichen Daten sehr leicht entschlüsselt und ausgelesen werden. Ein weiteres Problem stellen hardcodierte Passwörter oder auch die Verwendung von unsicheren und alten Verschlüsselungsalgorithmen und Protokollen, wie z.B. RC2, MD5 oder SHA1, dar.
Ein Angriff auf die Kryptographie kann erhebliche Auswirkungen auf geschäftliche Prozesse haben, da sie zu Verstößen gegen die Privatsphäre, Diebstahl vertraulicher Informationen oder auch Diebstahl von geistigem Eigentum führen kann etc..[Spr17, Car16]

M6 Unsichere Autorisierung: Diese Kategorie erfasst Fehler in der Autorisierungslogik. Es ist üblich, dass eine Offline-Applikation über ein lokales Authentifizierungsschemata oder eine serverseitige Authentifizierung für Online-Daten Zugriff auf vertrauliche Daten bietet. Durch das Fehlen entsprechender Sicherheitskontrollen besteht die Möglichkeit, die Autorisierung vollständig zu umgehen und auf

ansonsten eingeschränkte Bereiche der Applikation zuzugreifen. Angreifer könnten aufgrund dieser Schwachstelle, möglicherweise CRUD-Vorgänge ausführen oder Dienste aufrufen und empfangen, zu denen sie aufgrund ihrer Anmeldeinformationen keinen Anspruch hätten.
Als Gegenmaßnahme werden an dieser Stellen, die Verstärkung der Autorisierungskontrollen unter Zurhilfenahme von lokalen Integritätsprüfungen empfohlen, umso nicht autorisierte Veränderungen am Code rechtzeitig erkennen zu können.[Car16, Spr17]

M7 Client Code Qualität: Diese Kategorie umfasst Ereignisse und Themen, wie Pufferüberläufe, Sicherheitslücken bei Format-Strings, unsichere IPC-Mechanismen für clientseitige Injections und verschiedene andere Fehler auf Code-Ebene, bei denen Code, der auf dem Smartphone ausgeführt wird, neu geschrieben werden muss. Ein Pufferüberlauf in C oder ein DOM-basiertes XSS in einer Browser-App, kann Probleme mit der Codequalität verursachen. So können Angreifer die auf dem Smartphone erzwungenen Sicherheitskontrollen umgehen und es besteht ein hohes Risiko, dass vertrauliche Daten abgeschöpft werden.[Car16]

M8 Code Manipulation: Diese Schwachstelle behandelt die Änderung des Verhaltens einer Applikation zur Laufzeit, durch z.B. das Einschleusen eines Agenten, um die Applikation zu instrumentalisieren.
Sobald der Agent eingeschleust wurde, hat der Angreifer viele Möglichkeiten zur Manipulation der Applikation. So hat er vollen Zugriff auf Code, Speicherinhalte oder auch die von der Applikation verwendeten System-APIs. Als Gegenmaßnahme wird u.a. die Verwendung sicherer Codierungstechniken empfohlen.[Car16]

M9 Reverse Engineering: Bei dieser Kategorie geht es um Risiken, die durch das Reverse-Engineering von Applikationen entstehen können. Die Analyse der Kernbinärdatei, des Quellcodes oder auch die Bestimmung der verwendeten Bibliotheken einer Applikation, können einem Angreifer, einen guten Einblick in deren Funktionsweise geben. Der Angreifer erhält Informationen zu Schwachstellen, zu genutzten Back-End-Servern, kryptografischen Konstanten oder auch Chiffren. Zudem kann er Sicherheitskontrollen umgehen, den Code manipulieren und somit die ganze Applikation manipulieren. Die manipulierte Applikation kann dann beispielsweise zum Phishing von Anmeldeinformationen oder anderen vertraulichen Daten führen.[Car16]

M10 Fremdfunktionalität: Diese Schwachstelle tritt auf, wenn Entwickler vergessen, zusätzliche Funktionen, Backdoor-Funktionen, Administrator-Konten oder andere interne Sicherheitskontrollen, die während des Entwicklungsprozesses erstellt wurden, um die Applikationen besser testen zu können, zu entfernen. Der Angreifer kann sich leicht unter Ausnutzung dieser Schwachstelle Zugang zum Smartphone und den darauf befindlichen Applikationen verschaffen.

Die Entwickler sollten möglichst darauf achten, dass sich auf der Applikation kein Testcode mehr befindet, bevor diese veröffentlicht wird.

Wie in diesem Kapitel gezeigt werden konnte, kann ein mobiles Gerät und seine Applikationen eine Unmenge an Schwachstellen aufweisen, wodurch sich für Cyberkriminelle und ihre Malware wiederum eine Fülle von Angriffsvektoren bietet. Nach einer Analyse des Unternehmens NowSesure[13], die insgesamt 45.000 Applikationen aus den offiziellen App-Stores von Apple, Google und Drittanbietern, in Bezug auf die Top 10 Mobile Risks von OWASP testeten, fielen 85% der getesteten Applikationen in mindestens einen der OWASP Top 10 mobilen Risikobereiche. Dabei fiel auf, dass besonders viele Applikationen eine unsichere Datenspeicherung aufwiesen (50%), eine unsichere Kommunikation (48%), Fremdfunktionalitäten (47%) und Reverse Engineering sowie Client Code Quality mit jeweils 32% (siehe hierzu Abb.3.2).[Sch18]

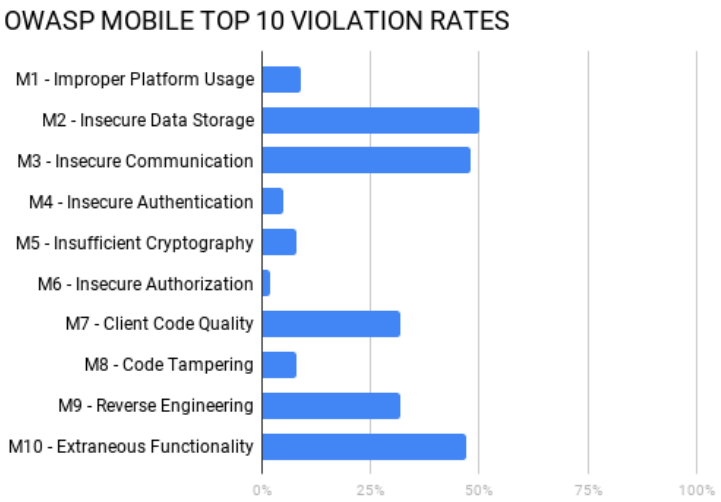


Abbildung 3.3: OWASP Top 10 Violation Rates[Sch18]

3.3 Das Sicherheitsmodell von Android

Das Android-Sicherheitsmodell präsentiert sich sowohl auf Plattform- als auch auf Applikationsebene, als eine flexibles und mehrschichtiges Modell .Auf Plattformebene bedient sich Android u.a. den Sicherheitskomponenten des Linux-Kernels, um z.B. Prozesse zu isolieren. Auf Applikationsebene stützt sich das Android OS mit den Permissions auf ein umfangreiches Berechtigungskonzept, um Anwendungen oder Komponenten die Interaktion mit anderen Applikationen, Komponenten oder kritischen Ressourcen zu ermöglichen oder zu verbieten. Zudem wird jede Android-App in einer separaten ART,

[13]NowSecure (Formerly viaForensics) is a Chicago-based mobile security company that publishes mobile app and device security software. Wikipedia contributors. NowSecure. Wikipedia, The Free Encyclopedia. June 27, 2019, 02:06 UTC. Available at: https://en.wikipedia.org/w/index.php?title=NowSecure&oldid=903656038., Accessed September 2, 2019.

einer sogenannten Sandbox, ausgeführt, die eine zentrale Rolle bei der Sicherung des Android OS spielt, da von ihr fast die gesamte Ausführung des Codes über dem Kernel eingeschränkt wird.
Neben den Sicherheitsfunktionen des Linux-Kernels und dem Berechtigungskonzept, werden im Weiteren das Sandboxing, der Ablauf bei der APK-Signierung, Mechanismen zur Kryptographie und das Patching detaillierter betrachtet.[Lev14, Andi]

3.3.1 Sicherheitfunktionen des Linux-Kernels und SELinux

Wie schon erwähnt, bedient sich das Android OS den Sicherheitsfunktionen des Linuxkernels. Dieser kann sowohl Prozesse als auch Benutzerressourcen voneinander isolieren. So erhält ein Benutzer keine Zugriffsberechtigungen auf die Dateien eines anderen Benutzers, außer es wurde explizit eine Zugriffsberechtigung erteilt. Zudem wird jeder Prozess mit der Benutzer-ID (UID) des Nutzers eindeutig gekennzeichnet.
Das Android-System bedient sich dieser Funktionen des Linux-Kernels u.a. zur eindeutigen Differenzierung und Isolation der Applikationsprozesse, die die Basis für das Sandboxing (vgl. hierzu 2.2.4 Die Android-Laufzeitumgebung) bilden. Das Dateisystem **core / include / private / android /_ filesystem /_ config.h** enthält eine Liste aller vordefinierten Benutzer-und Gruppen-IDs, denen Prozesse zugewiesen sind. Mit dem Kommandozeilenbefehl „ps | grep Browser“ kann ein einzelner laufender Prozess mit seiner zugehörigen UID ausgelesen werden. (vgl. Abb.3.4). Die Abbildung zeigt einen laufenden Prozess der Browser Applikation mit der UID „u0a14“. [Andf, Lev14]

```
root@generic:/system/app # ps | grep browser
u0_a14    1714  53    391588 64024 ffffffff b6f1c5cc S com.android.browser
root@generic:/system/app #
```

Abbildung 3.4: Benutzer-ID

Eine wesentliche Erweiterung, auch hinsichtlich der Isolierung von Prozessen, stellt Security Enchanced Linux-SELinux dar. SELinux, ursprünglich von der NSA[14] entwickelt, stellt eine Mandatory Access Control (MAC) bereit, mit der die Isolierung und Zugriffskontrollen für Prozesse und Ressourcen über eine vordefinierte Policy „der SEPolicy“ geregelt werden. Bei der Regelung der Zugriffskontrollen wird das Least Privilege Prinzip verfolgt, wodurch Prozesse nur das Minimum an benötigten Rechten erhalten. Prozesse oder Aktionen, die gegen die Policy verstoßen, werden geblockt.[Andf, Lev14]
Prozesse die geblockt werden, werden vom Kernel geloggt und können und mit den Kommandozeilenbefehlen „dmesg“ oder „logcat“ ausgelesen werden.

[14] Die National Security Agency (deutsch Nationale Sicherheitsbehörde), offizielle Abkürzung NSA, ist der größte Auslandsgeheimdienst der Vereinigten Staaten. Seite „National Security Agency“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 18. August 2019, 08:09 UTC. URL:https://de.wikipedia.org/w/index.php?title=National_Security_Agency&oldid=191453509, (Abgerufen: 4. September 2019, 12:15 UTC).

Die Vergabe der Zugriffsberechtigungen wird durch das Labeln der Ressourcen umgesetzt. Über ein Label wird einer Ressource eine Sicherheitsdomäne für einen Prozess zugewiesen. Dadurch kann SElinux erzwingen, dass nur Prozesse in derselben Domäne auf die Ressourcen zugreifen können und jeder Service erhält seine eigenen Berechtigungen und Sicherheitsprofile. Die Datei „seapp_context“ zeigt die Zuordnung der Applikationen anhand ihrer UIDs zu den einzelnen Domänen (vgl. hierzu Abb.3.5). Genauso wie den Applikationen, werden auch Dateien, einem Sicherheitskontext zugewiesen.[Andf, Lev14]

```
root@generic:/ # ps -Z | grep platform_app
u:r:platform_app:s0            u0_a7     430   52    com.android.systemui
u:r:platform_app:s0            u0_a6     666   52    com.android.externalstorage
root@generic:/ #
```

Abbildung 3.5: Zuordnung der Applikation zur Domäne

3.3.2 Sandboxing

Für jede Applikation im Android-System wird von der Android-Laufzeitumgebung (ART) eine eigene Instanz bzw. eine eigene Sandbox erzeugt. Die Sandbox regelt die Zugriffe des Programms auf die Ressourcen und Bestandteile des Android-Systems, wie z.B. das Netzwerk, die Telefonfunktionen oder auch den Speicher. Sie schränkt somit die Umgebung ein, in der Code ausgeführt werden kann. Bei dieser Funktion wird die Sandbox von SELinux (vgl. Kapitel 3.3.1 Sicherheitfunktionen des Linux-Kernels und SELinux) unterstützt. Ferner erhält jede Applikation bei der Erzeugung der Sandbox, eine eigene UID. Jede Applikations-Datei ist für den jeweiligen generierten Benutzer privat und andere Anwendungen können nicht auf diese Dateien zugreifen. Die UIDs beginnen bei 10000 und die UID 10078, würde bspw. dem Benutzernamen u0_a78 entsprechen (siehe hierzu Abb.3.6). Alle Dateien, die sich im Dateiverzeichnis der Browser-Applikation befinden, sind der UID u0_a14 zugeordnet.

```
1|root@generic:/ # ls -l /data/data/com.android.browser
drwxrwx--x u0_a14   u0_a14           2017-07-10 12:57 app_appcache
drwxrwx--x u0_a14   u0_a14           2017-07-10 12:57 app_databases
drwxrwx--x u0_a14   u0_a14           2017-07-10 12:57 app_geolocation
drwxrwx--x u0_a14   u0_a14           2017-07-10 12:57 app_icons
drwxrwx--x u0_a14   u0_a14           2017-07-11 05:06 app_webview
drwxrwx--x u0_a14   u0_a14           2017-07-11 05:07 cache
drwxrwx--x u0_a14   u0_a14           2017-07-10 12:57 databases
lrwxrwxrwx install  install          2017-07-10 08:27 lib -> /data/app-lib/com.android.browser
drwxrwx--x u0_a14   u0_a14           2017-07-11 05:07 shared_prefs
root@generic:/ #
```

Abbildung 3.6: Benutzer-ID

Verwaltet werden die UIDs gemeinsam mit anderen Paket-Metadaten sowohl in der /data/system/packages.xml als auch in der /data/system/packages.list. Wenn dann für eine Applikation Permissions (vgl. hierzu Kap.3.3.3 erteilt werden, wird noch eine sogenannte Gruppen-ID (GID) zum Prozess der Applikation hinzugefügt. Details

zu den einzelnen Beziehungen zwischen der GID und der jeweiligen Permission, über einen Aufruf der Datei **frameworks/base/data/etc/platform.xml** ermittelt werden. Neben den UIDs und den GIDs, gibt es noch die „Shared"UIDs , die es ermöglichen, mehrere Applikationen in einer Sandbox zu betreiben und diesen Zugriff auf dieselben Ressourcen zu gewähren, als ob es sich letztendlich nur um eine Applikation handeln würde. Die Freigabe der Ressourcen kann mithilfe von Services oder Content Providern vorgenommen werden.
Ziel des Sandboxing ist es, über die Isolierung der Applikation, die Sicherheit zu verbessern. Mit der Isolierung und der eindeutigen Zuordnung der Prozesse der Applikation zu einem Identifier, soll das Eindringen von Malware oder anderer nicht berechtigter Prozesse verhindert werden. [BP10, Ele14]

3.3.3 Das Berechtigungskonzept-Permissions im Android-System

Aus Sicherheitsgründen können Android-Applikationen nur auf Systemfunktionen und Ressourcen außerhalb der Sandbox zugreifen, wenn in der Applikation eine bestimmte Berechtigung (Permission) eindeutig definiert wurde. Beispielsweise ist es für das Versenden einer SMS-Nachricht erforderlich, dass vorher die Permission „Send_SMS" angefordert wird. Sollte nach dem Aufruf der SMS-Applikation keine Permission angefordert werden, wird zur Laufzeit vom Android-System eine Sicherheitsausnahme ausgelöst und die Applikation sofort beendet. Die Permissions werden im Android-Manifest gepflegt und erteilt, damit sie zum einen bei der Installation bekannt sind und zum anderen von anderen Applikationen genutzt werden können. Vor der Android-API 23 wurden alle im Android-Manifest verwalteten Permissions sofort nach dem Start der Applikation erteilt. Mit der Version 23 hat sich dieses Konzept geändert. Nun können die Permissions zur Laufzeit erteilt werden.
Darüber hinaus werden die Permissions in vier Schutzstufen eingeteilt:

1) **Normal**: Eine „normale" Permission, ist eine Permission mit einem geringen Risiko und wird einer Applikation automatisch vom System erteilt, ohne das der Nutzer zustimmen muss.

2) **Gefährlich**: Permissions fallen in die Kategorie „gefährlich" wenn die Applikation mit ihr Zugriff auf private Benutzerdaten oder die Kontrolle über Gerätefunktionen erhält. Permissions mit der Einstufung „gefährlich"können erhebliche auswirken auf die Sicherheit der Applikation und auch des Geräts haben.

3) **Signaturen**: Hier drunter fallen Permissions, die das System nur erteilt, wenn die anfordernde Applikation mit demselben kryptografischen Schlüssel signiert ist wie die Applikation, die die Permission deklariert hat. Wenn die Schlüssel übereinstimmen, erteilt das System automatisch die Permission, ohne den Benutzer zu benachrichtigen oder um dessen ausdrückliche Zustimmung zu bitten.

4) **SignatureOrSystem**: Die vierte Stufe umfasst Permissions, die das System nur für Applikation gewährt, die sich im Android-Systemimage befinden oder die mit demselben Zertifikat signiert sind, wie die Applikation, die die Permission deklariert hat.

Mit dem Befehl **$adb shell pm list permissions -f** werden alle auf einen Android-System befindlichen Permissions angezeigt, mit einer Beschreibung der Permission und der jeweiligen Schutzstufe (siehe hierzu Abb.3.7 list permissions).

```
All Permissions:

+ permission:android.permission.INTERNAL_SYSTEM_WINDOW
  package:android
  label:display unauthorized windows
  description:Allows the app to create windows that are intended to be used by the internal system user
 interface. Not for use by normal apps.
  protectionLevel:signature
+ permission:android.permission.MOVE_PACKAGE
  package:android
  label:move app resources
  description:Allows the app to move app resources from internal to external media and vice versa.
  protectionLevel:signature|system
+ permission:android.permission.CONTROL_WIFI_DISPLAY
  package:android
  label:control Wifi displays
  description:Allows the app to control low-level features of Wifi displays.
  protectionLevel:signature
+ permission:android.permission.READ_INPUT_STATE
  package:android
  label:record what you type and actions you take
:
```

Abbildung 3.7: list permissions

Um eine genauere Beschreibung der auf dem System verwendeten Permissions zu erhalten, kann der Befehl **$adb shell pm list permissions -s** verwendet werden (siehe hierzu Abb.3.8 List permissionsDescription).

```
All Permissions:

Development tools: access external storage of all users, send Linux signals to apps, read sensitive log
 data, force background apps to close, modify secure system settings, limit number of running processes
, change system display settings, retrieve system internal state, enable app debugging

Calendar:

Affects Battery: allow Wi-Fi Multicast reception, prevent phone from sleeping, control flashlight, tran
smit infrared, control vibration

Status Bar: expand/collapse status bar

Other Application UI: draw over other apps

Camera: disable transmit indicator LED when camera is in use, take pictures and videos

Services that cost you money:

Network communication: connect and disconnect from WiMAX, view network connections, change network conn
ectivity, control Near Field Communication, connect and disconnect from Wi-Fi, view Wi-Fi connections,
:
```

Abbildung 3.8: list permissions description

Bei den meisten Permissions handelt es sich um Konstanten oder Strings, die der Applikations-UID zugeordnet sind. Geprüft wird eine Zuordnung, indem die Zuordnung

nach einem bestimmten Paar vom Permission-UIDs durchsucht wird. Dabei ist zu beachten, dass eine bestimmte Anzahl an Permissions dem Linux-Kernel des Android-Systems zugeordnet wird, was im Umkehrschluss bedeutet, dass die UID einer Applikation, die über ein solche Permission verfügt, auch Mitglied in der GID-Gruppe ist. Dies trifft in der Regel auf alle Permissions zu, die mit privilegierten Aktionen verknüpft sind, die nicht von einem Systemdienst bereitgestellt werden, jedoch direkt mit dem Linux-Kernel interagieren. Beispiele für solche Aktionen sind Dateisystemvorgänge, wie der Zugriff auf SD-Cards, das Öffnen eines Netzwerksockets oder auch der Zugriff auf bestimmte Sysfs-Einträge.[And19b, Kof18]

3.3.4 Signierung der Applikation

Android erfordert, dass alle Applikationen bevor sie veröffentlicht und installiert werden, mit einem Zertifikat digital signiert werden. Mit der Signatur soll zum einen der Entwickler der Applikation eindeutig identifizierbar gemacht und sichergestellt werden, dass diese nach dem ersten Release nicht manipuliert wurde. Nicht signierte Anwendungen werden z.B. von Google Play oder dem Paketinstallationsprogramm des Android-Geräts abgelehnt. Selbst Debugbuilds werden mit einem Debugerzertifikat signiert. Informationen zu den .apk-Dateien und zum Entwickler enthält der Ordner META-INF. Mit den Befehlen **$openssl pkcs7 -in /Pfad zur entpackten .apk/META-INF/CERT.RSA -inform DER -print** oder **$jarsigner -verify -certs -verbose NameApk** können nähere Informationen zum Entwickler und zur Signierung der Applikation ermittelt werden (Siehe hierzu Abb.3.8 und Abb.3.9). [Ele14, HWA+17]

```
root@anaken-Aspire-E1-572G:/home/anaken/Downloads/apk# openssl pkcs7 -in META-INF/CERT.RSA -inform DER -print
PKCS7:
  type: pkcs7-signedData (1.2.840.113549.1.7.2)
  d.sign:
    version: 1
    md_algs:
        algorithm: sha256 (2.16.840.1.101.3.4.2.1)
        parameter: NULL
    contents:
      type: pkcs7-data (1.2.840.113549.1.7.1)
      d.data: <ABSENT>
    cert:
        cert_info:
          version: 2
          serialNumber: 1326412680
          signature:
            algorithm: sha1WithRSAEncryption (1.2.840.113549.1.1.5)
            parameter: NULL
          issuer: C=US, ST=California, L=San Francisco, O=Wikimedia Foundation, OU=Mobile, CN=Wikimedia Foundation
          validity:
            notBefore: Jan 12 23:58:00 2012 GMT
            notAfter: May 30 23:58:00 2039 GMT
          subject: C=US, ST=California, L=San Francisco, O=Wikimedia Foundation, OU=Mobile, CN=Wikimedia Foundation
          key:
            algor:
              algorithm: rsaEncryption (1.2.840.113549.1.1.1)
              parameter: NULL
```

Abbildung 3.9: Ausschnitt-CERT.RSA Wikipedia-App

Abbildung 3.10: Ausschnitt-Signierung APK

Während des Installationsprozesses wird durch den Paketmanager geprüft, ob die .apk-Datei mit dem in der .apk-Datei enthaltenen Zertifikat signiert wurde. Bei der Codesignierung werden Public-Key-Verschlüsselungsverfahren[15] und X.509-Zertifikate[16] verwendet.

Für den Signierungsprozess unterstützt Android drei verschiedenen Signaturschemata. Das eine basiert auf Java Archive, das (JAR)-Signatur Scheme v1 oder auch v1-Schema und das zweite auf APK Signature Scheme v2, dass ab Android Version 7.0 eingeführt wurde und das dritte auf APK Signatur Scheme v3. In der Regel wird die Anwendung aller Signaturschemata empfohlen, da das v1-Schema einige Teile der apk.Datei nicht vollumfänglich schützt. Der apk-Prüfer muss viele nicht vertrauenswürdige Datenstrukturen verarbeiten und dann diejenigen verwerfen, die im Signierungsprozess nicht komplett abgedeckt sind, was ein hohes Sicherheitsrisiko in sich birgt. Darüber hinaus muss der apk-Prüfer alle komprimierten Einträge dekomprimieren, wodurch viel Zeit und Speicherplatz beansprucht wird. Daher wurde das v2-Schema eingeführt. Beim v2-Schema wird während des Signierungsprozesses der Inhalt der .apk-Datei gehasht und signiert. Anschließend wird der resultierende apk-Signaturblock in die .apk-Datei

[15] Definition „Public-Key-Verschlüsselungsverfahren“ Ein Public-Key-Verschlüsselungsverfahren ist ein Verfahren, um mit einem öffentlichen Schlüssel einen Klartext in einen Geheimtext umzuwandeln, aus dem der Klartext mit einem privaten Schlüssel wieder gewonnen werden kann. Seite „Asymmetrisches Kryptosystem“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 7. Mai 2019, 10:45 UTC. URL:https://de.wikipedia.org/w/index.php?title=Asymmetrisches_Kryptosystem&oldid=188322140. (Abgerufen: 17. Mai 2019, 16:54 UTC)

[16] Definition „X.509-Zertifikate“ X.509 ist ein ITU-T-Standard für eine Public-Key-Infrastruktur zum Erstellen digitaler Zertifikate. Der Standard ist auch als ISO/IEC 9594-8 zuletzt im Oktober 2016 aktualisiert worden. Der Standard spezifiziert die folgenden Datentypen: Public-Key-Zertifikat, Attributzertifikat, Certificate Revocation List (CRL) und Attribute Certificate Revocation List (ACRL). In der elektronischen Kommunikation finden X.509-Zertifikate Anwendung bei den TLS-Versionen diverser Übertragungsprotokolle, wie z. B. beim Abruf von Web-Seiten mit dem HTTPS-Protokoll, oder zum Unterschreiben und Verschlüsseln von E-Mails nach dem S/MIME-Standard. Seite „X.509“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 19. März 2019, 10:45 UTC. URL:https://de.wikipedia.org/w/index.php?title=X.509&oldid=186736124. (Abgerufen: 17. Mai 2019, 17:01 UTC)

eingefügt. Während der Validierung wird die .apk-Datei dann als ein sogenannter Blob[17] behandelt und die Signaturprüfung wird für die gesamte Datei durchgeführt. Sobald Änderungen an der .apk-Datei festgestellt werden sollten, wird die Signatur ungültig (siehe hierzu Abb. 3.11).

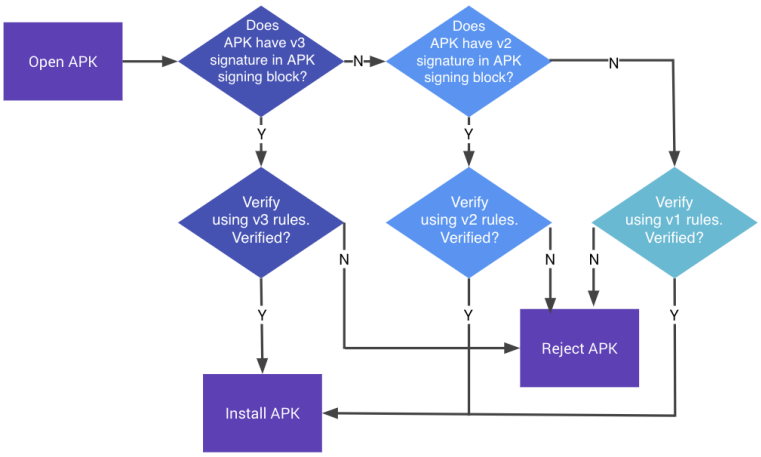


Abbildung 3.11: APK-Signaturüberprüfungsprozess[Ande]

Während des Signierungsprozesses hängt ein Signiertool, der apksigner von Android, das Zertifikat an die Applikation, wodurch die .apk-Datei einem privaten Schlüssel (i.d.R. der private Schlüssel des Entwicklers, der die Applikation signierte und geheim ist) zugeordnet wird. Mit dieser Vorgehensweise wird sichergestellt, das durchgeführte Aktualisierungen authentisch sind und auch vom ursprünglichen Entwickler stammen. Der verwendete Schlüssel wird als App-Signaturschlüssel bezeichnet und setzt sich aus einem privaten-und öffentlichen Schlüssel zusammen. Eine Applikation behält während ihrer gesamten Lebensdauer dasselbe Zertifikat. Zudem wird von der Applikation ein Hash mit MD5[18], SHA1 oder auch SHA-256 erstellt, den der Entwickler mit seinem privaten Schlüssel verschlüsselt. Damit können Dritte die Authentizität der .apk-Datei überprüfen. Im Weiteren dienen die gebildeten Hashwerte zur Registrierung der Dienste der Applikation (siehe hierzu Abb. 3.11[Ele14, And19e, Ande]

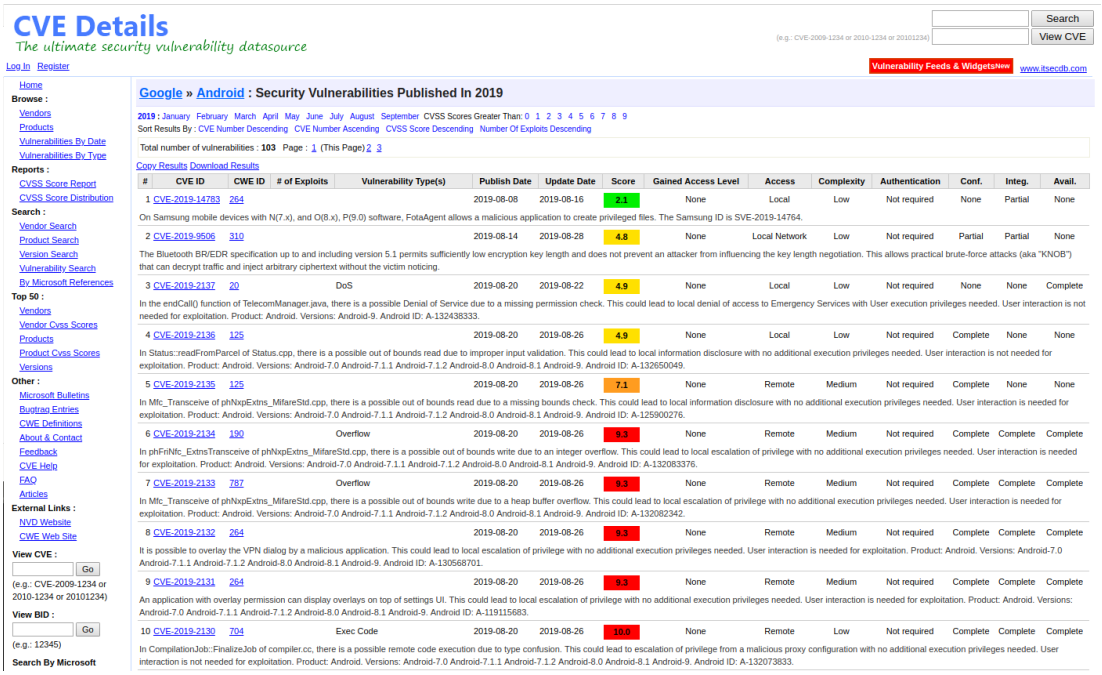
[17] Definition „Blob“ Binary Large Objects (BLOBs) sind große binäre Datenobjekte wie z. B. Bild- oder Audiodateien. Der Begriff wird häufig im Zusammenhang mit Datenbanken und Open-Source-Projekten benutzt. Seite „Binary Large Object“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 23. Oktober 2018, 17:01 UTC. URL:https://de.wikipedia.org/w/index.php?title=Binary_Large_Object&oldid=182075109. (Abgerufen: 30. Mai 2019, 21:11 UTC)

[18] Definition „MD5“Message-Digest Algorithm 5 (MD5) ist eine weit verbreitete kryptographische Hashfunktion, die aus einer beliebigen Nachricht einen 128-Bit-Hashwert erzeugt. Sie ist ein Vertreter aus einer Reihe von kryptographischen Hashfunktionen, die 1991 von Ronald L. Rivest am Massachusetts Institute of Technology entwickelt wurde, als Analysen ergaben, dass sein Vorgänger MD4 wahrscheinlich unsicher ist. Seite „Message-Digest Algorithm 5“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 22. Mai 2019, 11:12 UTC. URL:https://de.wikipedia.org/w/index.php?title=Message-Digest_Algorithm_5&oldid=188835034 (Abgerufen: 17. Mai 2019, 17:41 UTC)

3.3.5 Patchday

Unentdeckte und nicht gepatchte Schwachstellen, können für die Hard-und Software von Smartphones ein großes Sicherheitsrisiko darstellen. Oft entstehen die Schwachstellen schon während der Entwicklung und können diverse Ursachen haben, wie z.B. zunehmende Komplexität der Software, unsicherer Code oder auch die fehlerhafte Interaktion zwischen Soft-und Hardware.
Beim Android-Betriebssystem stellt der Patch-Prozess einen komplexen und zeitaufwendigen Vorgang dar, denn im Gegensatz zum iOS-oder Windows Betriebssystem, ist Android ein heterogenes Betriebssystem. So bestehen mittlerweile viele Android-Versionen, die entsprechend der einzelnen Hardwarekomponenten von den jeweiligen Herstellern angepasst wurden. Zudem besteht das Android-System aus sehr vielen Komponenten, die alle von Schwachstellen betroffen sein können und für die dann entsprechende Patches bereitgestellt werden müssen, das bedeutet, wenn eine Schwachstelle entdeckt wurde, entwickelt und veröffentlicht Google eine Patch. Der Patch wird dann mit dem Quellcode des Android-Systems zusammengeführt und anschließend den diversen Herstellern zur Verfügung gestellt. Der Seite „https://source.android.com/setup/contribute/life-of-a-patch“ können weitere Informationen zum Workflow eines Patches entnommen werden.
Informationen zu Patchdays bzw. Sicherheitsupdates für Android, werden zum einen auf den Seiten der Hersteller veröffentlicht und können zum anderen über die Seite „https://source.android.com/security/bulletin/2019-04-01.html“ abgerufen werden.[Kof18]
Des Weiteren werden Informationen zu Schwachstellen von MITRE[19] veröffentlicht. Die gemeldeten Schwachstellen werden von Mitre nach den Common Vulerabilities and Exposures (CVE) katalogisiert und bewertet. Zur Veröffentlichung wird der Schwachstelle eine eindeutige CVE-ID, wie z.B. CVE-2017-0637, zugwiesen. Zudem enthält der CVE-Eintrag eine Beschreibung der Schwachstelle, eine Risikoeinstufung und ggf. Verweise zu relevanten Schwachstellenberichten (siehe hierzu Abb.3.12).

[19] Die MITRE Corporation ist eine Organisation zum Betrieb von Forschungsinstituten im Auftrag der Vereinigten Staaten, die durch Abspaltung vom Massachusetts Institute of Technology (MIT) entstanden ist. Seite „Mitre Corporation“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 2. Juni 2019, 07:54 UTC. URL:https://de.wikipedia.org/w/index.php?title=Mitre_Corporation&oldid=189173045, (Abgerufen: 1.September 2019, 17:09 UTC)



CVE Details
The ultimate security vulnerability datasource

Search | View CVE

Log In | Register

Vulnerability Feeds & Widgets | www.itsecdb.com

Home
Browse :
Vendors
Products
Vulnerabilities By Date
Vulnerabilities By Type
Reports :
CVSS Score Report
CVSS Score Distribution
Search :
Vendor Search
Product Search
Version Search
Vulnerability Search
By Microsoft References
Top 50 :
Vendors
Vendor Cvss Scores
Products
Product Cvss Scores
Versions
Other :
Microsoft Bulletins
Bugtraq Entries
CWE Definitions
About & Contact
Feedback
CVE Help
FAQ
Articles
External Links :
NVD Website
CWE Web Site
View CVE :
Go
(e.g.: CVE-2009-1234 or 2010-1234 or 20101234)
View BID :
Go
(e.g.: 12345)
Search By Microsoft

Google » Android : Security Vulnerabilities Published In 2019

2019 : January February March April May June July August September CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

Total number of vulnerabilities : 103 Page : 1 (This Page) 2 3

Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2019-14783	264			2019-08-08	2019-08-16	2.1	None	Local	Low	Not required	None	Partial	None
	On Samsung mobile devices with N(7.x), and O(8.x), P(9.0) software, FotaAgent allows a malicious application to create privileged files. The Samsung ID is SVE-2019-14764.													
2	CVE-2019-9506	310			2019-08-14	2019-08-28	4.8	None	Local Network	Low	Not required	Partial	Partial	None
	The Bluetooth BR/EDR specification up to and including version 5.1 permits sufficiently low encryption key length and does not prevent an attacker from influencing the key length negotiation. This allows practical brute-force attacks (aka "KNOB") that can decrypt traffic and inject arbitrary ciphertext without the victim noticing.													
3	CVE-2019-2137	20		DoS	2019-08-20	2019-08-22	4.9	None	Local	Low	Not required	None	None	Complete
	In the endCall() function of TelecomManager.java, there is a possible Denial of Service due to a missing permission check. This could lead to local denial of access to Emergency Services with User execution privileges needed. User interaction is not needed for exploitation. Product: Android. Versions: Android-9. Android ID: A-132438333.													
4	CVE-2019-2136	125			2019-08-20	2019-08-26	4.9	None	Local	Low	Not required	Complete	None	None
	In Status::readFromParcel of Status.cpp, there is a possible out of bounds read due to improper input validation. This could lead to local information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-132650049.													
5	CVE-2019-2135	125			2019-08-20	2019-08-26	7.1	None	Remote	Medium	Not required	Complete	None	None
	In Mfc_Transceive of phNxpExtns_MifareStd.cpp, there is a possible out of bounds read due to a missing bounds check. This could lead to local information disclosure with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-125900276.													
6	CVE-2019-2134	190		Overflow	2019-08-20	2019-08-26	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
	In phFriNfc_ExtnsTransceive of phNxpExtns_MifareStd.cpp, there is a possible out of bounds write due to an integer overflow. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-132083376.													
7	CVE-2019-2133	787		Overflow	2019-08-20	2019-08-26	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
	In Mfc_Transceive of phNxpExtns_MifareStd.cpp, there is a possible out of bounds write due to a heap buffer overflow. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-132082342.													
8	CVE-2019-2132	264			2019-08-20	2019-08-26	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
	It is possible to overlay the VPN dialog by a malicious application. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-130568701.													
9	CVE-2019-2131	264			2019-08-20	2019-08-26	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
	An application with overlay permission can display overlays on top of settings UI. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-119115683.													
10	CVE-2019-2130	704		Exec Code	2019-08-20	2019-08-26	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	In CompilationJob::FinalizeJob of compiler.cc, there is a possible remote code execution due to type confusion. This could lead to escalation of privilege from a malicious proxy configuration with no additional execution privileges needed. User interaction is not needed for exploitation. Product: Android. Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9. Android ID: A-132073833.													

Abbildung 3.12: Ausschnitt-Liste CVE 2019 (Stand: 30. August 2019[CVEc]

Im Jahr 2017 führte das Android-System mit 843 vorhandenen Schwachstellen, die „Top 50 Products By Total Number Of "Distinct"Vulnerabilities in 2017“ an. 2018 sinkt die Anzahl der Schwachstellen stark, auf 611, und 2019 führte MITRE 103 (Stand: 30.August 2019) gemeldete Schwachstellen für Android-Systeme (vgl. hierzu Abb. 3.13).[CVEa, CVEc]

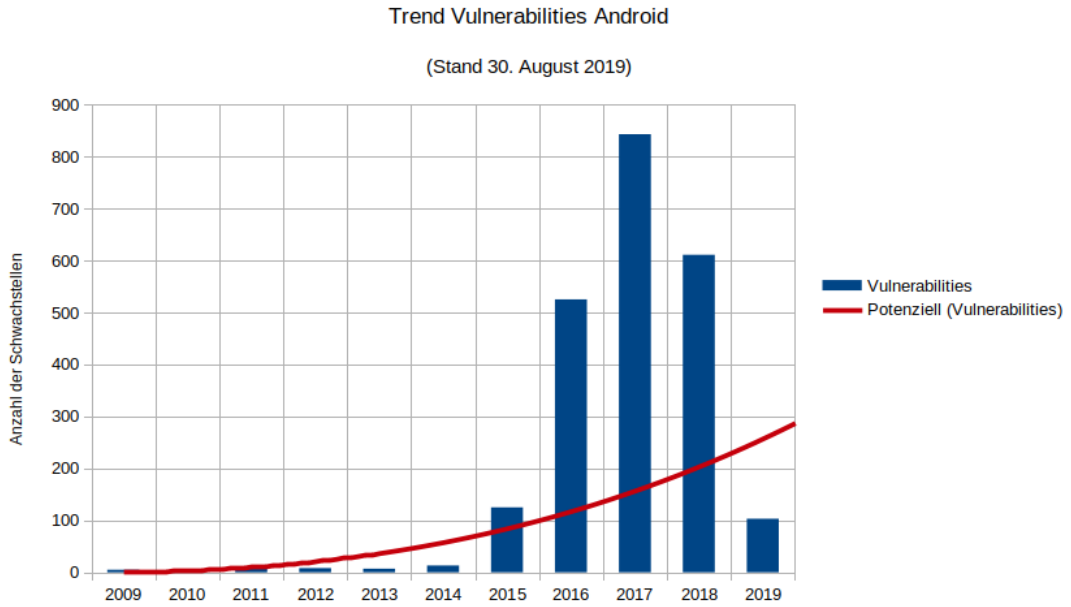


Abbildung 3.13: Trend Android-Vulnerabilities (Stand: 30. August 2019[CVEc]

Allerdings sollte an dieser Stelle angemerkt werden, dass keine Pflicht zur Meldung der Schwachstellen besteht und die Prozesse rein optional sind. Zudem können die Informationen zu den gemeldeten Schwachstellen auch stark voneinander abweichen. Des Weiteren lässt die Anzahl und Art der Schwachstelle keine direkten Rückschlüsse auf das Sicherheitsniveau von Android zu, da wie bereits dargestellt Android vielen Cyberattacken ausgesetzt ist und dadurch wesentlich mehr Schwachstellen als bei anderen Systemen aufgedeckt werden.[Kof18]

4 Sicherheitsanalyse von Android Applikationen

Wie im dritten Abschnitt aufgezeigt werden konnte, haben heutige mobile Apps eine umfangreiche Client-Server-Architektur und bieten dadurch eine breite Angriffsfläche. Selbst das mehrschichtige Sicherheitsmodell von Android kann keinen hinreichenden Schutz bieten. Sicherheitslücken, die zu einem ungewollten Datenabfluss führen, können aus unterschiedlichsten Gründen entstehen. Aufgrund dessen ist es sehr wichtig, dass Prozesse und Verfahren für den Schadensfall bestehen.
Für Unternehmen und vor allem für Bundesanstalten, bei denen ein erhöhter Schutzbedarf z.B. aufgrund der Verarbeitung sensibler und vertraulicher Daten besteht, sollten nicht nur, sondern müssen weitere Schutzmechanismen etabliert werden. Richtlinien und Empfehlungen zur Absicherung von mobilen Applikationen und sicheren Betreibung eines Mobile-Device-Managements, gibt u.a. auf nationaler Ebene das Bundesamt für Informationssicherheit (BSI) im IT-Grundschutz-Kompendium[BSI19]. Mit den bereitgestellten Richtlinien und Empfehlungen soll ganzheitlich Informationssicherheit basierend auf den Schutzzielen Verfügbarkeit, Vertraulichkeit und Integrität von Daten gewährt und umgesetzt werden.
Die Durchführung von Sicherheitstest-bzw. Analysen stellen dabei wichtige Instrumente dar, um die Umsetzung der Anforderungen zu überprüfen und letztendlich ganzheitlich Informationssicherheit gewähren zu können. Die Sicherheitsanalysen sollen Schwachstellen oder auch Funktionsfehler, die ggf. zu einer Sicherheitslücke führen könnten, identifizieren. Zudem sollen die Richtlinien und Anforderungen als Hilfestellung bei der Sicherung der Systeme und Applikationen dienen, um der Bedrohung durch Cyberangriffe entgegenwirken zu können.

Zu Beginn dieses Abschnitts wird auf allgemeine Grundsätze von Sicherheitsanalysen eingegangen und Testarten sowie Analysemethoden besprochen. Darauf aufbauend werden wesentliche Schritte für die Planung und Durchführung einer Sicherheitsanalyse erläutert. Unter Einbezug der BSI-Anforderungen aus dem IT-Grundschutzkompendium, der OWASP Top 10 mobilen Risikobereiche und des OWASP Mobile Security Guides[Sch19b] und ermittelter Schutzbedarfskategorien, wird eine Bedrohungsmodellierung vorgenommen. Ergebnis der Bedrohungsmodellierung soll eine Checkliste für die spätere Durchführung der Sicherheitsanalyse und ein Ablaufplan sein.
Da die Anforderungen des BSI sehr umfangreich sind, werden diese aufgrund des gegebenen Umfangs dieser Arbeit in tabellarischer Form zusammengefasst dargestellt. Auch die vorgenommene Bedrohungsmodellierung ist rein exemplarisch und soll als ein möglicher Rahmen verstanden werden. Zudem bestehen sehr viele Ansätze und Modelle zur Durchführung von Bedrohungsmodellierungen, die in der Regel während des Secure Software Development Life Cycles SSDLC vorgenommen, werden. Der SSDLC ist ein Framework,

das in detaillierten Schritten den Prozess zum Erstellen einer Software definiert. Während dieses Prozesses, werden mehrere Sicherheitstest durchgeführt, um ganzheitlich Sicherheit gewährleisten zu können. Auf den SSDLC wird aber auch nicht weiter eingegangen, da sich diese Arbeit auf die Durchführung von Sicherheitsanalysen von mobilen Android-Apps vor dem „Go-Live“ konzentriert. Weitere Informationen zur Bedrohungsmodellierung oder auch zum SSDLC können diese Seiten entnommen werden (https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Webanwendungen/Webanw_Auftragnehmer.pdf?__blob=publicationFile&v=2).

4.1 Die Sicherheitsanalyse

Die Sicherheitsanalyse von mobilen Applikationen versteht sich als ein wesentliches Instrument zur Prüfung der Korrektheit, Unumgänglichkeit und Wirksamkeit von Sicherheitsanforderungen, die die Client-Server-Architektur und die von der mobilen Applikation verwendeten serverseitigen APIs umfasst. Im allgemeinen wird bei Sicherheitsanalysen zwischen zwei Arten unterschieden. Zum einen können Sicherheitsanalyse von Beginn an in einen SDLC eingebunden werden und zum anderen können Sicherheitsanalysen vor dem Go-Live einer Applikation durchgeführt werden. Bei den eingesetzten Testverfahren wird zwischen manuellen und automatisierten Testverfahren unterschieden. Eine Sicherheitsanalyse bzw. Sicherheitstest kann grob in 4 Phasen aufgeteilt werden.
In der Vorbereitungsphase werden der Umfang des Tests, die geltenden Sicherheitsanforderungen , die Testziele, die Testart und einzusetzende Tools bzw. die Testumgebung, festgelegt. In der zweiten Phase, dem „Information Gathering“ sollen Informationen zur Applikation gesammelt werden, um ein allgemeines Verständnis über die gesamte Architektur zu erhalten. Die nächste Phase beinhaltet die Detektion und Analyse von Schwachstellen. In der Regel folgt die Analyse fest definierten Testfällen. Gegebenenfalls wird auch versucht gefundene Schwachstelle zu penetrieren, um diese dann eindeutig verifizieren zu können. In der letzten Phase erfolgt die Berichterstellung. Im Bericht werden alle Schwachstellen dokumentiert und bewertet. Zudem wird ausführlich beschrieben auf welche Art und Weise die Schwachstelle kompromittiert werden konnte und Empfehlungen zur Behebung der Schwachstelle gegeben.[Vol17, Fra14, Sch19b]

4.1.1 Testarten

- **Blackbox-Test:** Bei einem Blackbox-Test werden dem Tester keine Informationen zur Architektur oder dem Quellcode der Applikation zur Verfügung gestellt. Ziel eines Blackbox-Test ist es, die fehlerfreie und vollständige Umsetzung einer Applikation zu überprüfen. In der Regel dienen Blackbox-Test zur Ermittlung von Schwachstellen am Backend-Server oder anderen Webdiensten mit denen eine

Applikation verbunden ist. Der Tester versetzt sich sozusagen in die Rolle eines Hackers und versucht Sicherheitslücken aufzufinden und auszunutzen. Zur Durchführung von Blackbox-Test wird auf verschiedene Blackbox-Verfahren zurückgegriffen, mit denen Testfälle erstellt werden. Neben den gängigen Usability-Tests, werden im Rahmen von Blackbox-Test oft sogenannte „Monkey-Tests“durchgeführt, die einen Stress-und Robustheitstest darstellen. Dabei werden wilde Aktionen ausgeführt, so als wenn ein Affe die Applikation bedienen würde, um das System bzw. die Applikation z.B. zum Absturz zu bringen.[Vol17, Fra14]

- **Whitebox-Test:** Bei einem Whitebox-Test oder auch Full-Knowledge Testing werden alle Informationen zur Applikation, wie Client-Server-Architektur, Strukturen, Klassen und Funktionen der Applikation, offengelegt. Ziel eines Whitebox-Test ist es, alle Komponenten einer Applikation auf Korrektheit und Vollständigkeit zu prüfen und den Programmcode zu analysieren.[Vol17, Fra14]

- **Greybox-Test:** Der Greybox-Test versteht sich als eine Kombination aus White- und Blackboxtest. Abhängig vom Test-Scope, werden im Rahmen eines Greybox-Tests nur Teilinformationen der zu testenden Applikation offengelegt. Dabei ist Ziel des Tests, die richtige Umsetzung der Sicherheitsanforderungen und den korrekten Ablauf sowie den Aufbau der Applikation, zu testen, wobei der Fokus auf der Testung der Schnittstellen in der Client-Server-Architektur der Applikation, liegt.[Vol17, Fra14]

4.1.2 Analysemethoden

Statische Analyse

Bei der statischen Analyse wird je nach eingesetztem Verfahren entweder der Quellcode oder auch der kompilierte Bytecode oder Binärcode auf mögliche Schwachstellen hin analysiert. „Statisch“ bedeutet in diesem Zusammenhang, dass die Applikation während der Analyse nicht ausgeführt wird. Um den Quellcode auf potenzielle Schwachstellen zu prüfen, kommen bei der statischen Analyse verschiedene Verfahren zum Einsatz, wie z.B. die lexikalische-oder Datenflussanalyse.
Statische Analysen können manuell oder automatisiert durchgeführt werden. Bei einer manuellen Überprüfung, wird der Quellcode entweder zeilenweise oder nach bestimmten Schlüsselwörtern, wie spezielle Funktions-oder Methodenaufrufe, hin untersucht. Bei der automatisierte statischen Analyse wird unter Zuhilfenahme bestimmter Tools automatisiert nach vordefinierten Regel der Quellcode auf Sicherheitslücken überprüft.[Spr17, Sch19b]

Dynamische Analyse

Mit der dynamische Analyse wird die Applikation zur Laufzeit überprüft, wobei das Hauptziel darin besteht, Sicherheitslücken und Schwachstellen im laufenden Programm zu finden. Dies umfasst u.a. die Analyse des lokalen Dateisystems, des Netzwerkverkehrs zwischen der Anwendung und dem Server sowie die Analyse der IPC-Mechanismen der Applikation. Die dynamische Analyse dient hauptsächlich zur Offenlegung von Schwachstellen in der Kryptographie, SSL-Verschlüsselung, bei Authentifizierungsproblemen oder auch Serverkonfigurationsfehlern. Genau wie die statische Analyse, kann auch eine dynamische Analyse manuell oder automatisiert erfolgen.[Spr17, Sch19b]

4.2 Die Bedrohungsmodellierung

Die Erstellung eines Bedrohungsmodells versteht sich als ein systematischer und iterativer Prozess. Ziel dieses Prozesses ist es, die Bereiche bzw. Komponenten der Applikation zu identifizieren und zu bewerten, die Schwachstellen aufweisen, um entsprechend Schutzmechanismen zu implementieren und Gegenmaßnahmen bestimmen zu können. Das fertige Bedrohungsmodell soll dann in Form einer Checkliste entweder schon während des Entwicklungsprozesses zur Sicherung einer App oder spätestens bei der Durchführung von Sicherheitsanalysen vor dem Go-Live einer App, dienen. Die Abb.4.1 Ablauf einer Bedrohungsmodellierung, zeigt exemplarisch den Ablauf einer Bedrohungsmodellierung. In der ersten Phase der Bedrohungsmodellierung wird die Definition der Schutzziele bzw. Assets vorgenommen. In den folgenden Phasen werden dann die definierten Schutzziele den identifizierten Angriffsvektoren zugeordnet und unter Zuhilfenahme verschiedener Bewertungsmodelle, wie z.B. STRIDE[1] oder anderer Schutzbedarfskategorien, klassifiziert und bewertet. Die Identifizierung und Bewertung der Angriffsvektoren baut dabei auf ein sicheres und tiefes Verständnis der gesamten Infrastruktur der Applikation.[Tae07, SFSP10]

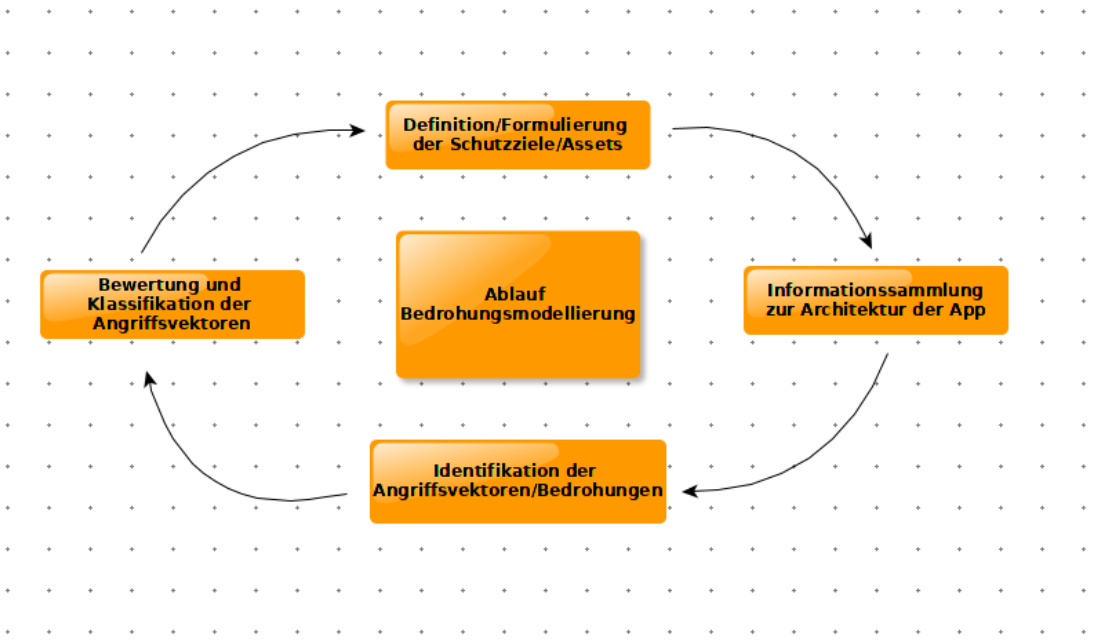


Abbildung 4.1: Ablauf einer Bedrohungsmodellierung

[1]STRIDE is a model of threats developed by Praerit Garg and Loren Kohnfelder at Microsoft for identifying computer security threats. It provides a mnemonic for security threats in six categories. The threats are: 1)Spoofing of user identity, 2) Tampering, 3) Repudiation, 4) Information disclosure (privacy breach or data leak), 4) Denial of service (D.o.S), 5) Elevation of privilege. The STRIDE was initially created as part of the process of threat modelling. STRIDE is a model of threats, used to help reason and find threats to a system. It is used in conjunction with a model of the target system that can be constructed in parallel. This includes a full breakdown of processes, data stores, data flows and trust boundaries. Wikipedia contributors. STRIDE (security). Wikipedia, The Free Encyclopedia. July 29, 2019, 15:11 UTC. Available at: https://en.wikipedia.org/w/index.php?title=STRIDE_(security)&oldid=908411919. Accessed September 20, 2019.

Folgend werden die einzelnen Komponenten erläutert die später für die Definition der Schutzziele und die Identifikation und Klassifikation der Angriffsvektoren herangezogen werden.

4.2.1 Sicherheitsanforderungen für mobile Applikationen des BSI

Das IT-Grundschutz-Kompendium des BSI kategorisiert sich in mehrere Bausteine. Der Baustein „APP.1.4 Mobile Anwendungen“ regelt den Schutz von Informationen, die auf mobilen Geräten mit Applikationen verarbeitet werden. Ebenso wird die Einbindung von Applikationen in EMM bzw. MDM-Lösungen betrachtet und Anforderungen zur App-Auswahl und deren Betreibung definiert. Dabei werden alle mobilen Applikationen betrachtet, die auf mobilen Betriebssystemen, wie z.B. Android oder iOS, und entsprechenden mobilen Geräten installiert sind.
Anforderungen an die jeweiligen Betriebssysteme oder an die Entwicklung und Verwaltung von Applikationen über MDMs, werden gesondert in den Bausteinen „SYS.3.2.4 Android“und „ SYS.3.2.2 Mobile Device Management (MDM)“ geregelt. Da, wie schon bereits erläutert, eine Applikation in ihrer gesamten Client-Server-Struktur, betrachtet werden soll, um ganzheitlich Informationssicherheit zu gewähren, fließen in die Formulierung der Anforderungen auch anwendungsspezifischen Aspekte mit ein, die in den Bausteinen „APP.3.1 Webanwendungen, APP.3.5 Webservices, APP.4.3 Relationale Datenbanksysteme“ u.a. geregelt werden. Zudem sollten die Bausteine, die die Anforderungen bezüglich der Testung von Applikationen „OPS.1.1.6 Software-Tests und -Freigaben“ formulieren, berücksichtigt werden.
Wie bereits erwähnt, haben die Anforderungen des BSIs einen enormen Umfang, aufgrund dessen nur der Baustein APP.1.4 Mobile Anwendungen kurz näher betrachtet wird. Alle weiteren relevanten Bausteine, die auch für die Bedrohungsmodellierung und für die Gestaltung und den Umfang der Sicherheitsanalyse von Bedeutung sein könnten, werden mit einem Verweis auf das BSI IT-Grundschutzkompendium, nur benannt.
Die Anforderungen zu den mobilen Applikationen „APP.1.4 Mobile Anwendungen“ wurden entsprechend spezifischer Angriffsvektoren für mobile Anwendungen vom BSI definiert. Die einzelnen Anforderungen werden folgend kurz erläutert und mit ihren zugehörigen Gefährdungen bzw. Angriffsvektoren tabellarisch dargestellt (siehe hierzu Abb.4.2) BSI Referenztabelle.[BSI19]

Schutzbedarf	BSI Referenztabelle APP.1.4 Mobile Apps	Gefährdungen / Anforderungen	G0.9 Ausfall oder Störung von Kommunikationsnetzen	G 0.14 Ausspähen von Informationen (Spionage)	Verlust/Verlust von Verlust von Datenträgern oder Dokumenten	G0.18 Fehlplanung oder fehlende Anpassung	G0.19 Offenlegung schützenswerter Informationen	G0.20 Informationen oder Produkte aus unzuverlässiger Quelle	G0.22/0.23 Manipulation/ Unbefugtes Eindringen von	G0.25/0.26 Ausfall/Defekt von Geräten oder Systemen	G0.28 Software-Schwachstellen oder -Fehler	G0.29/0.32 Verstoß gegen Gesetz Mißbrauch von Berechtigungen oder der Regelungen	G0.30/0.38 Diebstahl/ Missbrauch personenbezogener Daten	G0.39 Schadprogramme	G0.42 Social Engineering
Basisanforderungen	APP.1.4.A1 Anforderungsanalyse f. d. Nutzung v. Apps	- Welche Geschäftsprozesse werden unterstützt? - An welche IT-Komponenten wird d. App angebunden? - Festlegung von Sicherheitsanforderungen - Betrachtung d. Schutzbedarfs	X	X	X	X	X			X		X	X		
	APP.1.4.A2 Regelungen f. d. Verwendung v. mobilen Endgeräten und Apps	- Erstellung von Nutzerrichtlinien: > Welche Daten dürfen verarbeitet werden > Ist die private Nutzung gestattet? > Wie muss d. Nutzer sich in öffentl. Netzen verhalten > Verhalten bei Geräteverlust		X	X	X	X		X			X			X
	APP.1.4.A3 Verwendung Sicherer Quellen für Apps	-Bezug von Apps aus sicheren u. vertrauenswürdigen Quellen - interne Apps u. Apps d. schutzbedürftige Informationen verarbeiten, sollten über gesonderten Store bezogen werden		X		X	X	X	X			X		X	
	APP.1.4.A4 Test und Freigabe von Apps	- App muss Anforderungen d. Fachlichen Bereiche , der Informationssicherheit und d. Datenschutzes erfüllen - App wird vor Freigabe getestet - Definition von Freigabekriterien - Tests sollten gesamte MDM umfassen				X				X	X	X	X	X	X
	APP.1.4.A5 Minimierung und Kontrolle von App-Berechtigungen	- Kontrolle d. App-Permission-> Erhält die App nur die minimal nötigen Permissions? - Fixierung der Permissions bei sicherheitsrelevanten Einstellungen - regelmäßigen Überprüfung der Permissions		X			X		X			X	X	X	X
	APP.1.4.A6 Patchmanagement für Apps	- zeitnahe Einspielung von Updates - zentralisierte Updatesteuerung über vorhandene MDM-Lösung - Formulierung von Ausnahmebedingung, wenn Schwachstelle nicht zeitnah geschlossen werden kann								X	X				
	APP.1.4.A7 Sichere Speicherung Lokaler App-Daten	- wenn App auf interne Sharepoints zugreift = Sicherstellung der lokalen Datenhaltung - Daten sollten hinreichend verschlüsselt sein - keine Speicherung von vertraulichen Daten auf externen Speichermedien (SD-Card)		X	X		X								
	APP.1.4.A8 Verhinderung von Datenabfluss	- Sicherstellung der Verschlüsselung d. Kommunikation von sensiblen u. vertraulichen Daten - Testung der Kryptographie während d. Sicherheitsanalyse/Pentest - Testung der App auf ungewollte Protokollierungs-und Hilfsdateien		X			X						X		
Standardanforderungen	APP.1.4.A9 Sichere Anbindung An Backend-Systeme	- hinreichende kryptographische Absicherung d. Datenverkehrs zw. Backend u. anderen Systemen	X	X			X		X						
	APP.1.4.A10 Sichere Authentisierung von Apps	- Sicherstellung der Authentisierungsmechanismen - Werden die Daten mit einem sicheren Protokoll übertragen?	X	X			X		X						
	APP.1.4.A11 Zentrales Management von Apps	- Es SOLLTE eine MDM-Lösung etabliert werden - Es SOLLTE über Prüfmechanismen und Whitelists sichergestellt werden, dass nur geprüfte und freigegebene Apps verwendet werden können				X						X			
	APP.1.4.A12 Sichere Deinstallation von Apps	Wenn Apps deinstalliert werden, SOLLTEN auch alle von der App generierten Dateien gelöscht werden. Dabei SOLLTEN ebenso vom Betriebssystem zwischengespeicherte Daten mit Bezug zur App gelöscht werden, etwa Caches. Außerdem SOLLTEN auch Daten gelöscht werden, die auf externen Systemen, beispielsweise beim App-Anbieter, gespeichert wurden.		X		X	X					X	X		
Erhöhter Schutzbedarf	APP.1.4.A13 Entwicklung von Fallback-Lösungen für Apps	- Bereitstellung redundanter Systeme	X			X			X	X					
	APP.1.4.A14 Unterstützung zusätzlicher Authentisierungsmerkmale bei Apps	- Nutzung einer Zwei-Faktor Authentifizierung - Überprüfung der verwendeten Mechanismen der Authentisierung auf Sicherheit		X	X	X	X		X			X	X		
	APP.1.4.A15 Durchführung Von Penetrationstests für Apps	Durchführung von Penetrationtests/Sicherheitsanalysen vor dem Go-Live einer App. Dabei soll die gesamte Infrastruktur auf Sicherheitslücken untersucht werden. Die Test sollen in regelmäßigen Abständen wiederholt werden.							X	X	X				

Abbildung 4.2: BSI Referenztabelle

Informationen zu allen anderen Bausteinen, die für den Schutz von mobilen Applikationen relevant sind, können der Tabelle „BSI Anforderungen zum Schutz von mobilen Apps“ entnommen werden.[BSI19]

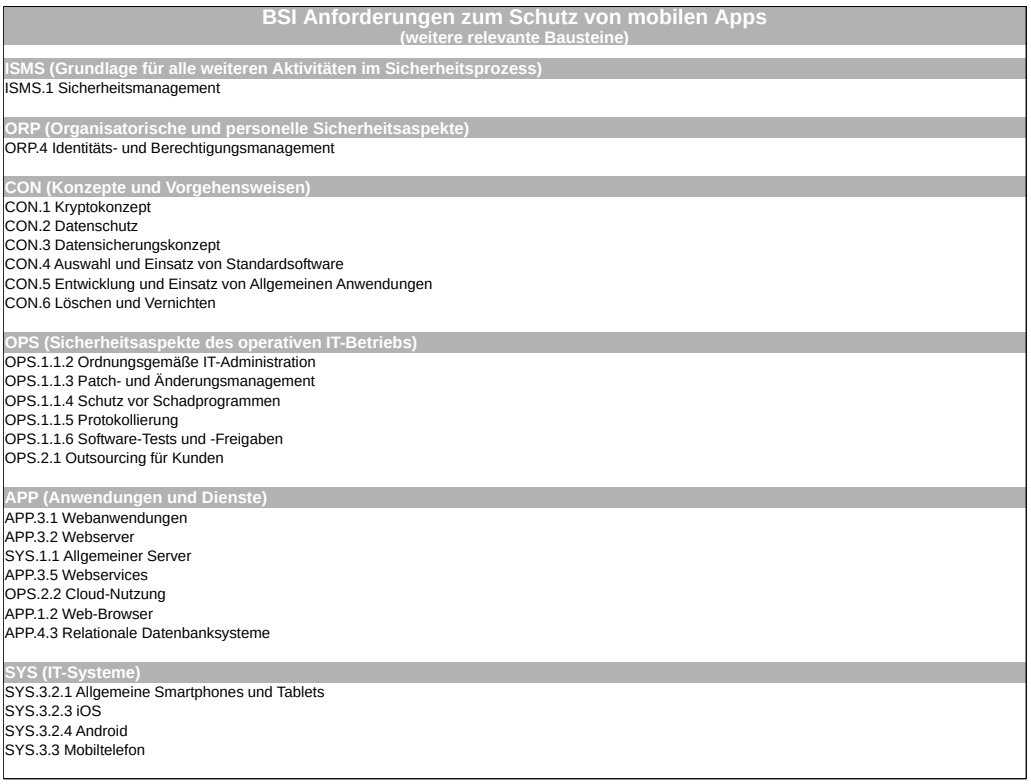


Abbildung 4.3: BSI Anforderungen zum Schutz von mobilen Apps

4.2.2 Schutzbedarfsfeststellung

Eine Schutzbedarfsfeststellung richtet sich an den Hauptschutzzielen Vertraulichkeit, Integrität und Verfügbarkeit, zur Gewährung der Informationssicherheit aus. Ziel der Schutzbedarfsfeststellung ist eine Ermittlung der Schadenshöhe im Falle einer Verletzung der Informationssicherheit. Sowohl für die Informationen die erzeugt und verarbeitet werden, um sie je nach ihrer Wertigkeit und Kritikalität einordnen zu können, als auch zur Unterscheidung des Schutzbedarfs, werden Kategorien definiert.
Zu den Informationskategorien zählen: Unternehmensinformationen, Verschlusssachen und personenbezogene Daten.Schutz2019

1 **Unternehmensinformationen**: Unternehmensinformationen können in die Kategorien, Frei verwendbar, Intern, Vertraulich und Streng vertraulich, unterteilt werden.

2 **Personenbezogene Daten**: Personenbezogene Daten sind gemäß EU-Datenschutzgrundverordnung (DSGVO) und Bundesdatenschutzgesetz (BDSG) alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person beziehen. Persönliche Daten werden in folgenden Schutzklassen kategorisiert:

- SK1: Zu dieser Schutzklasse gehören alle besonderen Kategorien personenbezogener Daten, die sich auf eine natürliche Person beziehen oder zumindest beziehbar sind, wie z.B. Private-und Dienstliche Kontaktdaten, Geburtsdatum, Berufsbezeichnung, physische Merkmale, Besitzmerkmale, Benutzerkennungen, IP-Adressen.

- SK2: Zu dieser Schutzklasse gehören alle besonderen Kategorien personenbezogener Daten, die sich auf eine natürliche Person beziehen oder zumindest beziehbar sind. Darüber hinaus gehören zu dieser Schutzklasse allgemeine Kategorien personenbezogener Daten die aufgrund des Verwendungszusammenhangs oder der Anzahl einen besonderen Schutzbedarf aufweisen. Darunter fallen z.B. Abmahnungen und Kündigungen, ethnische Herkunft, politische Meinungen, religiöse oder philosophische Überzeugungen, Gesundheit, Biometrische Daten.

- SK3: Die Schutzklasse 3 entspricht den besonderen Kategorien von besonders schützenswerten personenbezogener Daten gemäß SK2. Darüber hinaus fallen die Daten zusätzlich unter das Berufsgeheimnis gemäß §203 StGB. Hieraus ergeben sich besondere Verpflichtungen für den „Geheimnisträger“ in Bezug auf die interne Verarbeitung und bei Beauftragung von Dienstleistern. Die Verarbeitung und damit jeder Zugriff oder Zutritt zu diesen Daten ist nur zulässig, wenn eine besondere Verpflichtung der Zugriffs- und Zutrittsberechtigten seitens des Geheimnisträgers vor der ersten Verarbeitung stattgefunden hat. Beispiele für Daten die in diese Kategorie fallen, sind: Diagnosedaten, Arztbriefe, Daten aus psychologischen Gutachten, Mandantenakten von Rechtsanwälten und Notaren, Akten von Wirtschafts- und Buchprüfern/Steuerberatern.

2 Verschlusssachen: VS-Informationen sind VS-NfD (Verschlussachen nur für den Dienstgebrauch), wenn die Kenntnisnahme durch Unbefugte für die Interessen der Bundesrepublik Deutschland oder eines ihrer Länder nachteilig sein kann. Verschlusssachen (auch kurz als "VS"bezeichnet) sind nach §4 Abs. 1 SÜG im öffentlichen Interesse geheimhaltungsbedürftige Tatsachen, Gegenstände oder Erkenntnisse, unabhängig von ihrer Darstellungsform (zum Beispiel Schriftstücke, Zeichnungen, Karten, Fotokopien, Lichtbildmaterial, elektronische Datenträger, elektrische Signale, Geräte, technische Einrichtungen, das gesprochene Wort etc.). Verschlusssachen werden entsprechend ihrer Qualität von einer amtlichen Stelle oder auf deren Veranlassung eingestuft. Die detaillierte Behandlung von Verschlusssachen (VS) wird im Handbuch für den Geheimschutz in der Wirtschaft (Geheimschutzhandbuch (GHB) des Bundesministeriums für Wirtschaft und Energie (BMWi)) beschrieben. Die Behandlung von VS-NUR FÜR DEN DIENSTGEBRAUCH (VS-NfD) ist in der Anlage 4 des GHB beschrieben.

Der Schutzbedarf wird in der Regel in die drei Kategorien

1 **Normal**: Die Schadensauswirkungen sind begrenzt und überschaubar.

2 **Hoch**: Die Schadensauswirkungen können beträchtlich sein.

3 **Sehr Hoch**: Die Schadensauswirkungen können ein existentiell bedrohliches, katastrophales Ausmaß erreichen.

eingeteilt.[BSI, Roe, BfD]

Daraus lassen sich folgende Schutzbedarfskategorien bilden:

Schutzbedarfsfestellung	
Informationskategorie/Schutzklasse	**Schutzbedarf**
Frei verwendbar	kein Schutzbedarf
Intern/SK1	normaler Schutzbedarf
Vertraulich/SK2	hoher Schutzbedarf
Streng Vertraulich/SK3	Sehr hoher Schutzbedarf

Abbildung 4.4: Ermittlung des Schutzbedarfs

4.2.3 Checkliste für die Sicherheitsanalyse

Die Checkliste für die Sicherheitsanalyse bildet eine wichtige Grundlage für die Gestaltung und den Ablauf des gesamten Prüfprozesses und stellt das Ergebnis der vorgenommenen Bedrohungsmodellierung dar (vgl. hierzu Abb.4.5 Checkliste für die Sicherheitsanalyse).

Checkliste Sicherheitsanalyse Mobile Apps

ID	Anforderungen nach Schutzbedarf	SK1	SK2	SK3	Status	Bemerkungen
V1	**Anforderungen an Architektur, Design und Bedrohungsanalysen/ M10 Fremdfunktionalitäten**					
1.1	Alle Komponenten der mobilen App sind identifiziert und für den Betrieb der App erforderlich.	✓	✓	✓	Pass	
1.2	Die App stammt aus einer sicheren Quelle.	✓	✓	✓		
1.3	Sicherheitsfunktionen sind niemals nur auf Client-Seite implementiert sondern immer auch im entsprechenden entfernten API-Endpunkt.	✓	✓	✓	Fail	
1.4	Es existiert eine Architekturübersicht über die mobile App und alle verbundenen API-Endpunkte und Security wurde in der Gesamtarchitektur berücksichtigt.		✓	✓	N/A	
1.5	Alle sensiblen Daten im Kontext der mobilen App wurden klar identifiziert.		✓	✓		
1.6	Für jede Komponente der App ist der angebotene fachliche Funktionsumfang und/oder Sicherheitsfunktionen/-mechanismen klar definiert.		✓	✓		
1.7	Für die mobile App und die genutzten API-Endpunkte wurde eine Bedrohungsanalyse durchgeführt und potentielle Bedrohungen und Gegenmaßnahmen identifiziert.		✓	✓		
1.8	Alle Sicherheitsfunktionen wurden in Form von zentralen Komponenten implementiert.		✓	✓		
1.9	Eine dedizierte Richtlinie zum Management von kryptographischen Schlüsseln (falls in der App genutzt) beschreibt den sicheren Umgang mit Schlüsseln über den gesamten Lebenszyklus, idealerweise basierend auf Standards (siehe hierzu BSI-IT-Grundschutz)		✓	✓		
1.10	Über Prüflisten und Whitelists sollte sichergestellt werden, dass nur geprüfte Apps verwendet werden.	✓	✓	✓		
1.11	Es gibt ein Mobile Device Management, über das Apps zentral gemanagt und aktualisiert werden. Updates können, falls notwendig erzwungen werden.	✓	✓	✓		
1.12	Es besteht im MDM eine klare und wirksame Mandantentrennung.	✓	✓	✓		
1.13	Es besteht eine zentrale Schnittstellen Verwaltung, die über das MDM administrierbar ist. Darunter fallen insbesondere Schnittstelle, wie Bluetooth, Infrarot, WLAN, SMS, MMS, GPS und USB.	✓	✓	✓		
V2	**Anforderungen an Datenspeicherung und Datenschutz/ M2 Unsicherer Datenspeicher**					
2.1	Die App speichert sensible Daten wie personenbezogene Daten, Anmeldedaten oder kryptographische Schlüssel unter Nutzung der vom jeweiligen Betriebssystem angebotenen sicheren Speichermechanismen.	✓	✓	✓		
2.2	Es werden keine sensiblen Daten außerhalb des App-Containers oder außerhalb des vom jeweiligen Betriebssystem angebotenen sicheren Speichermechanismus abgelegt.	✓	✓	✓		
2.3	Es werden keine sensiblen Daten in die Logfiles der App geschrieben.	✓	✓	✓		
2.4	Es werden keine sensiblen Daten mit Dritten geteilt - es sei denn dies wurde in der App-Architektur definiert und ist zur Erfüllung des Zwecks der App erforderlich.	✓	✓	✓		
2.5	Der Tastatur-Cache ist für alle sensiblen Texteingaben deaktiviert.	✓	✓	✓		
2.6	Es werden keine sensiblen Daten über IPC-Mechanismen zur Verfügung gestellt.	✓	✓	✓		
2.7	Es werden keine sensiblen Daten wie Passwörter oder Pins über die Benutzeroberfläche exponiert.	✓	✓	✓		
2.8	Es ist sichergestellt, dass betriebssystemgesteuerte Backups keine sensiblen App-Daten enthalten.	✓	✓	✓		
2.9	Die App entfernt sensible Daten aus der aktuellen Ansicht wenn der Hintergrundmodus aktiviert wird.	✓	✓	✓		
2.10	Die App hält sensible Daten nur solange wie nötig im Speicher und betroffene Speicherbereiche werden nach Nutzung explizit gelöscht.		✓	✓		
2.11	Die App erzwingt ein Minimum an Geräteschutz-Richtlinien wie das Definieren eines Gerätepassworts.		✓	✓		
2.12	Die App klärt den Nutzer über die Art und Weise der verarbeiteten personenbezogenen Daten auf und gibt dem Nutzer Security-Best-Practice-Empfehlungen zum Umgang mit der App.			✓		
V3	**Anforderungen an Kryptographie/ M5 Unzureichende Kryptographie**					
3.1	Verschlüsselung in der App basiert nicht nur auf symmetrischer Kryptographie mit hart-codierten Schlüsseln.	✓	✓	✓		
3.2	Die App nutzt bewährte Implementierungen zur Umsetzung kryptographischer Primitive.	✓	✓	✓		
3.3	Die App nutzt kryptographische Primitive passend zum spezifischen Anwendungsfall, konfiguriert nach Best-Practice Vorgaben dem Stand der Technik entsprechend.	✓	✓	✓		
3.4	Die App nutzt keine kryptographischen Protokolle oder Algorithmen die allgemein als veraltet und unsicher gelten.	✓	✓	✓		
3.5	Die App verwendet einen kryptographischen Schlüssel für genau einen Zweck und nicht für mehrere Zwecke.	✓	✓	✓		
3.6	Alle Zufallswerte werden über einen ausreichend sicheren kryptographischen Zufallszahlengenerator erzeugt.		✓	✓		
V4	**Anforderungen an Authentifizierung und Session Management/ M4 Unsichere Authentifizierung, M6 Unsichere Autorisierung**					
4.1	Falls die App Nutzern Zugriff auf entfernte Service APIs bietet wird am API-Endpunkt eine Authentifizierung z.B. mit Nutzername/Passwort durchgeführt.	✓	✓	✓		
4.2	Kommt Session-Management am API-Endpunkt zum Einsatz, so werden zufällig generierte Session-IDs erzeugt um Client-Anfragen zu authentifizieren und keine Nutzer-Anmeldedaten versandt.	✓	✓	✓		
4.3	Kommt statuslose Token-basierte Authentifizierung zum Einsatz, so werden die Token am Server mit einem sicheren Algorithmus signiert.	✓	✓	✓		
4.4	Der API-Endpunkt beendet die existierende Nutzersitzung sobald sich der Nutzer abmeldet.	✓	✓	✓		
4.5	Es existiert eine Passwort-Richtlinie die am entfernten API-Endpunkt erzwungen wird.	✓	✓	✓		
4.6	Der entfernte API-Endpunkt implementiert einen Mechanismus um sich gegen eine exzessive Anzahl von Login-Versuchen zu schützen.	✓	✓	✓		
4.7	Nach einer definierten Inaktivitätsdauer werden Nutzersitzungen am entfernten API-Endpunkt beendet und Zugriffs-Tokens werden nach Ablauf der Gültigkeitsdauer abgelehnt.	✓	✓	✓		
4.8	Biometrische Authentifizierung basiert auf dem Betriebssystem-basierten Entsperren des Keystores und nicht auf einer z.B. event-basierten API die einfach "true" oder "false" zurück liefert.		✓	✓		
4.9	Es gibt einen 2. Authentifizierungsfaktor und er wird am entfernten API-Endpunkt konsistent erzwungen.		✓	✓		
4.10	Sensible Transaktionen erfordern eine zusätzliche Authentifizierung (durch einen weiteren Authentifizierungsfaktor).			✓		
4.11	Die App informiert den Nutzer über alle Anmelde-Vorgänge am Nutzerkonto. Nutzer können eine Liste aller mit dem Konto verbundenen Geräte sehen und ausgewählte Geräte blockieren.			✓		
V5	**Anforderungen an Netzwerkkommunikation/M3 Unsichere Kommunikation**					
5.1	Daten werden zur Netzwerkkommunikation innerhalb der App durchgängig mit TLS verschlüsselt.	✓	✓	✓		
5.2	Die TLS-Einstellungen entsprechen aktuellen Best-Practices, oder erfüllen die Best-Practices so gut wie möglich falls das mobile Betriebssystem die Empfehlung nicht unterstützt.	✓	✓	✓		
5.3	Die App überprüft das X.509-Zertifikat des Servers beim TLS-Handshake. Die App akzeptiert nur Zertifikate die von einer vertrauenswürdigen Zertifizierungsstelle signiert wurden.	✓	✓	✓		
5.4	Die App nutzt entweder ihren eigenen Zertifikatspeicher oder nutzt Public Key-/ Zertifikats-Pinning und akzeptiert deshalb keine Verbindungen zu Endpunkten die andere Zertifikate/Schlüssel präsentieren - selbst wenn die Zertifikate von einer vertrauenswürdigen Zertifizierungsstelle signiert sind.		✓	✓		
5.5	Die App nutzt keine unsicheren Kommunikationskanäle wie Email oder SMS für kritische Operationen wie Konten-Registrierung oder Konten-Reaktivierung.		✓	✓		
5.6	Die App nutzt aktuelle Bibliotheken zum Aufbau von sicheren Kommunikationsverbindungen.		✓	✓		
V6	**Anforderungen zur Plattform-Interaktion/ M1 Unsachgemäße Plattformnutzung**					
6.1	Die App fordert vom Nutzer nur die unbedingt erforderlichen App-Berechtigungen.	✓	✓	✓		
6.2	Alle Eingaben von externen Quellen und dem Nutzer werden validiert und falls erforderlich bereinigt. Dazu gehören Daten aus der GUI, IPC Mechanismen wie Intents oder spezifische URL-Schemas und Netzwerk-Daten.	✓	✓	✓		
6.3	Die App bietet keine sensible Funktionalität über App-eigene URL-Schemas an - wenn doch, ist der Mechanismus angemessen geschützt.	✓	✓	✓		
6.4	Die App bietet keine sensible Funktionalität über Interprozesskommunikation (IPC) an - wenn doch, ist der Mechanismus angemessen geschützt.	✓	✓	✓		
6.5	JavaScript ist in WebViews deaktiviert wenn es nicht unbedingt erforderlich ist.	✓	✓	✓		
6.6	WebViews sind so konfiguriert, dass nur die minimal nötigen Protokoll-Handler erlaubt sind (Idealerweise nur HTTPS). Potentiell gefährliche Handler wie file, tel und app-id sind deaktiviert.	✓	✓	✓		
V7	**Anforderungen zu Code Qualität und Build-Einstellungen/M7 Schlechte Code-Qualität**					
7.1	Die App ist signiert und mit einem gültigen Zertifikat provisioniert, dessen privater Schlüssel angemessen geschützt ist.	✓	✓	✓		
7.2	Bibliotheken und Frameworks von Drittanbietern die die App nutzt wurden auf Schwachstellen geprüft.	✓	✓	✓		
7.5	Die App führt eine sichere Fehlerbehandlung durch, indem sie Exceptions abfängt und kontrolliert behandelt.	✓	✓	✓		
7.4	Treten in Sicherheitsfunktionen Fehler auf lehnt die App-Fehlerbehandlung die Zugriffe standardmäßig ab.	✓	✓	✓		
7.5	Das Speichermanagement (Allokation und Freigabe von Speicher) erfolgt in unmanaged code auf sichere Weise.	✓	✓	✓		
7.6	Angebotene Sicherheitsfunktionen der Entwicklungsumgebung wie Byte-Code Minimierung, Stack-Protection, PIE-Support und automatisches Reference-Counting sind aktiviert.	✓	✓	✓		
V8	**Anforderungen an Manipulationssicherheit/Resilienz/ M8 Code Manipulation/ M9 Reverse Engineering**					
8.1	Die App erkennt und reagiert auf ein gerootetes Gerät oder Geräte mit jailbreak indem der Nutzer gewarnt oder die App beendet wird.		✓	✓		
8.2	Die App verhindert Debugging und/oder erkennt und reagiert auf einen Debugger. Alle verfügbaren Debugging-Protokolle müssen abgedeckt werden.		✓	✓		
8.3	Die App erkennt und reagiert auf Manipulation von Code und Daten im eigenen Arbeitsspeicherbereich.		✓	✓		
8.4	Programmatische Abwehrmaßnahmen werden durch Obfuskierung geschützt, was wiederum De-Obfuskierung mittels dynamischer Analyse verhindert.			✓		
8.5	Die App implementiert einen Mechanismus zur Gerätebindung bei dem der Geräte-Fingerabdruck aus mehreren einzigartigen Geräteeigenschaften ermittelt wird.			✓		
8.6	Alle ausführbaren Dateien und Bibliotheken der App sind entweder auf Dateiebene verschlüsselt und/oder wichtige Code- und Datenabsegmente in ausführbaren Dateien sind verschlüsselt oder durch Packing obfuskiert. Triviale statische Analyse offenbart keinen wichtigen Code oder Daten.			✓		
8.7	Wenn das Ziel der Obfuskierung der Schutz sensibler Logik wie Algorithmen oder Berechnungen ist, so wird ein angemessener Obfuskierungsmechanismus, dem Stand der Technik entsprechend, genutzt der resilient gegen manuelle und automatisierte De-Obfuskierungsangriffe ist. Die Wirksamkeit der Obfuskierungsmethode muss durch manuelle Tests überprüft werden. Es ist zu beachten, dass hardware-basierte Isolations-Mechanismen softwarebasierter Obfuskierung vorzuziehen sind.			✓		

Legende	
Symbole	Definition
Pass	Die App erfüllt die genannte Anforderung.
Fail	Die App kann die genannte Anforderung nicht erfüllen.
N/A	Anforderung konnte nicht getestet werden

Abbildung 4.5: Checkliste für die Sicherheitsanalyse mobiler Apps

In der Checkliste wurden anhand der formulierten Schutzziele und identifizierten Gefährdungen und Anforderungen des BSI und der OWASP (siehe hierzu OWASP Top10 Risikobereiche Kap.3.2.4 „App-basierte Angriffe“ und OWASP MSTG-Checkliste https://github.com/OWASP/owasp-mstg/blob/master/Checklists/Mobile_App_Security_Checklist-English_1.1.2.xlsx), acht Testfälle definiert. Der Schutzbedarf (siehe hierzu Kap.4.2.2) entscheidet dabei wesentlich über die tiefe der vorzunehmenden Analyse. (Die komplette Checkliste befindet sich nochmal im Anhang und liegt als Datei bei.)

4.2.4 Ablaufplan einer Sicherheitsanalyse

In den folgenden Abbildungen 4.6 und 4.7 wird ein möglicher Ablaufplan für die Sicherheitsanalyse einer mobile App dargestellt und tabellarisch eine Beschreibung vorgenommen. Ausführliche Inhalte zu den einzelnen Prüffällen können der in Kapitel 4.2.3 vorgestellten Checkliste entnommen werden.

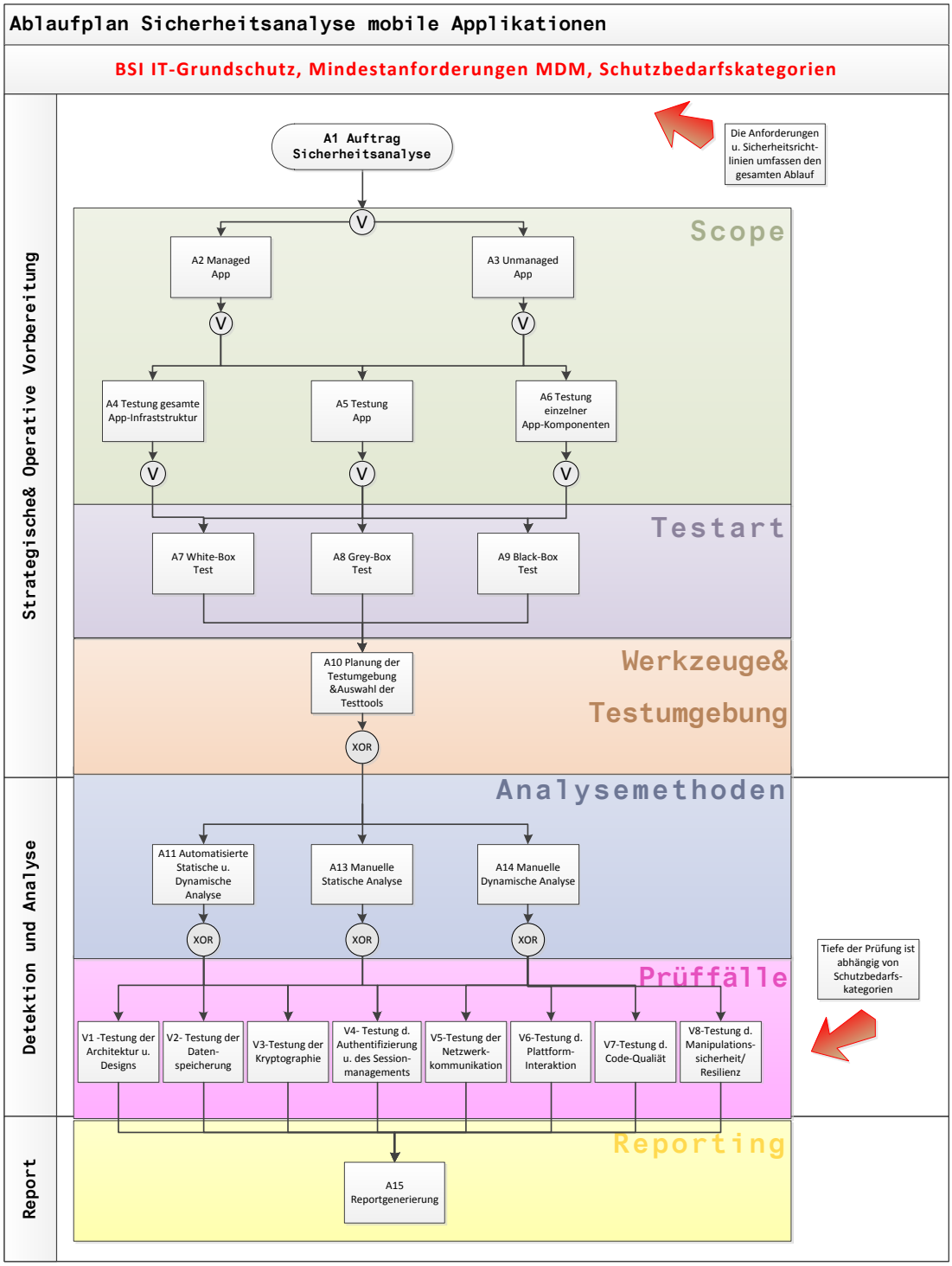


Abbildung 4.6: Ablaufplan für die Sicherheitsanalyse einer mobilen App

Ablauf			Beschreibung
Strategische& Operative Vorbereitung	Scope	**A1**	Von einem Kunden geht der Auftrag zur Durchführung einer Sicherheitsanalyse von einer mobilen Applikation ein.
		A2/A3	A2: Der Auftrag umfasst die Sicherheitsanalyse einer „Managed App“: - die App wurde selber durch das Unternehmen entwickelt - es wurden für das Unternehmen Modifizierungen an der App vorgenommen - die App hat Zugriff auf das Backend des Unternehmens - die App verarbeitet Daten der Schutzkategorien 1,2,3 - die App verarbeitet Informationen der Kategorien: intern, vertraulich, streng vertraulich - die App verarbeitet Verschlusssachen **Oder** A3: Der Auftrag umfasst die Sicherheitsanalyse einer „Unmanaged App“: - die App hat keine Verbindung zur Infrastruktur des Unternehmens - die App verarbeitet für die Öffentlichkeit frei zugängliche Daten - die App verarbeitet nur Daten der Schutzkategorie 1 - z.B. DB-App, Internetauftritt des Unternehmens
		A4-A6	A4: Die Sicherheitsanalyse soll die gesamte App-Infrastruktur umfassen. Hierbei wird die gesamte Client-Server-Architektur, das MDM von der die App gemanagt wird und auch alle Komponenten der Applikation einer Analyse unterzogen. Oder A5: Die Sicherheitsanalyse umfasst nur die Testung der Applikation. Oder A6: Die Sicherheitsanalyse umfasst die Testung einzelner Komponenten der App, wie z.B. die Verwaltung der Permissions, Prüfung der IPC-Mechanismen oder die Verschlüsselung
	Testart	**A7-A9**	Es wird die Testart für die Sicherheitsanalyse festgelegt: A7: Die Sicherheitsanalyse wird als Whitebox-Test durchgeführt **Oder** A8: Die Sicherheitsanalyse wird als Greybox-Test durchgeführt **Oder:** A9: Die Sicherheitsanalyse wird als Blackbox-Test durchgeführt
	Werkzeuge	**A10**	In diesem Schritt werden Bedingungen und Aufbau für die Testumgebung geklärt. - Werden für die Analyse ggf. noch mobile Devices benötigt und Zugangsdaten für das Backend? - Sollen die Tests in einer speziellen Integrationsumgebung/Testumgebung durchgeführt werden? - Steht die zu testende Applikation zur Verfügung etc. Des Weiteren wird geklärt, welche Tools für die Analyse zum Einsatz kommen bzw. kommen dürfen (→ Clean-Source-Ansatz)
Detektion und Analyse	Analysemethoden	**A11-A14**	A11: Es sollen automatisierte statische und dynamische Analysen durchgeführt werden. **und/oder** A12: Es soll eine manuelle statische Analyse erfolgen. **und/oder** A13: Es soll eine manuelle dynamische Analyse erfolgen.
	Prüffälle	**V1-V8**	Folgende Prüffälle soll die Analyse umfassen: V1: Testung der Architektur und des Designs und/oder: V2: Testung der Datenspeicherung und/oder: V3: Testung der Kryptographie und/oder V4: Testung der Authentifizierung und des Sessionmanagements und/oder V5: Testung der Netzwerkkommunikation und/oder: V6: Testung der Plattformkommunikation und/oder: V7: Testung der Code-Qualität und/oder: V8: Testung der Manipulationssicherheit/Resilienz
Report	Reporting	**A15**	Erstellung des Reports. Der Report beinhaltet eine detaillierte Auflistung aller gefundenen Schwachstellen und eine Beschreibung über die Art und Weise, wie diese kompromittiert werden konnten. Ggf. werden Empfehlungen zur Behebung der Schwachstellen gegeben

Seite 1

Abbildung 4.7: Tabellarische Beschreibung des Ablaufs der Sicherheitsanalyse

Die in der Bedrohungsmodellierung definierten Sicherheitsanforderungen, identifizierten Angriffsvektoren und letztendliche Bewertung dieser, nehmen Einfluss auf den gesamten Ablauf der Sicherheitsanalyse. Mit ihnen wird geregelt, wie mit „Managed-Apps“ oder „Unmanaged Apps“ verfahren wird, was sich wiederum auf den Testumfang niederschlagen kann und auch über die Testart entscheidet.
Selbst die Auswahl der Testtools und Aufbau bzw. Gegebenheiten der Testumgebung können entscheidend vom Schutzbedarf sowie gegebener Sicherheitsrichtlinien abhängen. Ebenso die Tiefe der Analyse ist abhängig vom Schutzbedarf.

5 Vorbereitung und Durchführung der Sicherheitsanalyse

In diesem Abschnitt der Arbeit wird anhand des Prüffalls „Anforderungen an Datenspeicherung und Datenschutz“aus der in Kap.4.2.3 erstellten Checkliste, prototypisch eine Sicherheitsanalyse durchgeführt. Es werden Tipps zum Aufbau der Testumgebung und zu gängigen Tools für die Durchführung von Analysen von Android-Apps gegeben. Für die Analyse von Android-Applikationen gibt es mittlerweile viele bewährte Open-Source-Tools mit denen sich automatisiert und manuell sowohl statische als auch dynamische Analysen durchführen lassen. Dabei hat jedes Tool seine Vor-und Nachteile. Eine umfangreiche Sammlung mit Erklärungen und Links können zum einen der Seite „Awesome Hacking“ entnommen werden. Die Liste mit Tools wird ständig aktualisiert und es werden Literatur und Beiträge zu Standards für Analysen und Pentest bereitgestellt (`https://github.com/ashishb/android-security-awesome`). Zum anderen bietet die OWASP umfangreiche Informationen für die Durchführung von Sicherheitsanalysen, die u.a. Online zur Verfügung gestellt werden und gerade im Bereich der Durchführung von Sicherheitsanalysen und Pentest einen Standard bilden. Es werden nicht nur Checklisten zur Orientierung bereitgestellt, sondern auch viele Erläuterungen zur Durchführung einer Analyse mit den entsprechenden Tools gegeben (`https://github.com/OWASP/owasp-mstg`).

5.1 Aufbau der Arbeitsumgebung

5.1.1 Testsystem-und Umgebung

Als Testsystem für die Analyse dient u.a. ein Notebook, Acer-Aspire E1-572G 64-Bit, mit Ubuntu-Mate-Bionic 18.04.03 LTS als Betriebssystem. Folgende aktuelle Kernel und Compiler Version befinden sich auf dem System: Linux version 5.0.0-29-generic (builddlgw01-amd64-039) (gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1 18.04.1)) #31 18.04.1-Ubuntu.

Für die Testumgebung wird ein System mit mindestens 8 GiB Arbeitsspeicher empfohlen und genügend freier Speicherplatz auf der Festplatte, um eine einwandfreie Funktion der virtuellen Analyseumgebung und des Android-Emulators von Genymotion (vgl. hierzu Kap.5.2.2) garantieren zu können. Zudem sollte die aktuellste Python-Version installiert sein, da einige für die Analyse verwendete Tools, wie z.B. das Mobile-Security-Framework, auf Python basieren. Die aktuell vom System verwendete Python-Version kann im Terminal mit dem Befehl **$ python -V**.

Für die spätere dynamische Analyse dient u.a. ein gerootestes Samsung Galaxy S8 mit der Android-Version 7.0 (Custom-Rom[1]). Ausführliche Anleitungen zum Rooten eines Android-Gerätes können der Seite https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05b-Basic-Security_Testing.md. Es wird darauf hingewiesen, dass mit dem Rooten des Smartphones eventuell die Hersteller-Garantie verloren gehen kann oder bei nicht ordnungsgemäßer Durchführung das Gerät nicht mehr funktioniert. Falls kein Android-Gerät für die Analyse zur Verfügung steht, können die Emulatoren im AndroL4b oder MobSF genutzt werden.

5.1.2 Android-Genymotion-Emulator

Der Genymotion-Emulator ist ein plattformübergreifender Android-Emulator. Genymotion kann virtuelle Android-Geräte auf gängigen Betriebssystemen, wie Linux, iOS oder Windows ausführen und verfügt über vordefinierte virtuelle Geräte für verschiedene Android-Telefone und -Tablets. Der Android-Emulator kann auch mit Android Studio verwendet werden.[Gen]
Der Genymotion ist ein wichtiger Bestandteil des Mobile-Security-Frameworks, das in Kap.5.2.2, vorgestellt wird. Darüber hinaus dient er als virtuelles Testdevice.
Für die Ausführung des Android-Emulators verwendet Genymotion VirtualBox. Daher muss auf dem Testsystem VirtualBox installiert sein und die Hardwarevirtualisierung (VT-d / VT-x / AMD-v) im BIOS aktiviert werden. Da Genymotion nicht im Paket-Repository von Ubuntu 18.04 vorhanden ist, muss es von der Seite „https://www.genymotion.com/fun-zone/“ geladen werden. Um die Installationsdatei zu erhalten, verlangt der Anbieter, dass ein Account angelegt wird.
Sobald der Account erstellt wurde, wird die Installationsdatei zum Download freigegeben. Im nächsten Schritt muss die Datei für die Installation ausführbar gemacht werden. Dazu dient der Befehl **$ chmod +x ~/Downloads/genymotion-3.0.2-linux_x64.bin**. Anschließend wird mit dem Befehl **$ sudo ~/Downloads/./genymotion-3.0.2-linux_x64.bin** die Installation gestartet. Nach Abschluss der Installation, wird das Programm über einen erstellten Icon im Anwendungsmenü von Ubuntu gestartet. Für weitere Hilfestellungen zur Auswahl einer passenden API oder auch Gerätes, wird die Anleitung auf folgender Seite empfohlen: https://linuxhint.com/install_genymotion_android_emuator_ubuntu/.
In dieser Arbeit wird für die Sicherheitstests ein Android-Phone Galaxy S8 mit der Android API 7 gewählt. Nach Start des Android-Emulator, muss parallel die VirtualBox gestartet werden. Das Device wird in der Regel automatisch in VirtualBox eingebunden.

[1] https://de.wikipedia.org/wiki/Liste_von_Android-Custom-ROMs

5.1.3 Testapplikationen

Für die Sicherheitsanalyse werden folgende Testapplikationen verwendet:

- Diva-Android: DIVA wurde speziell für Testzwecke entwickelt und soll Sicherheitsexperten vermitteln, welche Mängel bei Android-Applikationen aufgrund schlechter Codierungspraktiken auftreten können. Mit DIVA können Schwachstellen bei der Protokollierung, der Datenspeicherung, der Eingabevalidierung und der Zugriffssteuerung aufgezeigt werden. Die .apk-Datei ist auf folgender Seite erhältlich $\rightarrow https://github.com/payatu/diva-android$

- InsecureBankv2: InsecureBankv2 ist eine mit Sicherheitslücken entwickelte Banking-App. Mit ihr sollen u.a. folgende Schwachstellen aufgezeigt werden: Fehlerhafte IPC-Mechanismen, unsichere Authentifizierung-und Autorisierung, unzureichende Kryptographie, unsichere Datenspeicherung und unsichere Netzwerkkommunikation. Die .apk-Datei ist auf folgender Seite erhältlich $\rightarrow https://github.com/dineshshetty/Android-InsecureBankv2$

5.1.4 VBox – Oracle VM VirtualBox

Die Vbox[2] von Oracle dient zur Virtualisierung von Betriebssystemen und ist kompatibel mit diversen Hostsystemen. Sie erlaubt nicht nur das gleichzeitige Ausführen mehrerer Betriebssysteme, sondern auch den Austausch von Daten zwischen dem Gast-und Hostsystem. In diesem Projekt soll die VBox zur Einbindung des virtuellen Images „AndroL4b“ und des Android-Genymotion-Emulator dienen. Wenn als Hostsystem für die Vbox ein Linux-Ubuntu genutzt wird, sollte für die Installation von Vbox nicht das in Ubuntu mitgelieferte Paket verwendet werden, da dies unter Umständen zu Schnittstellenproblemen führen kann, wie z.B. keine Erkennung von USB-Sticks oder anderer Medien. Aktuelle Versionen von Vbox für das jeweils verwendete Betriebssystem können direkt von Oracle geladen werden https://www.virtualbox.org/wiki/Downloads. „genutzte Version: Oracle VM VirtualBox 6.0.10 r132072 (Qt5.9.5)“

5.1.5 Androl4b

AndroL4b ist eine virtuelle Maschine die auf Ubuntu-Mate basiert (AndroL4b v.3 aktuellste Version basiert auf Ubuntu-Mate 17.04) und eine umfangreiche Sammlung an Tools, Frameworks, Tutorials und einen Android-Emulator für die Durchführung u.a. von Sicherheitsanalysen von Android Applikationen enthält. Folgend wird kurz auf

[2]Seite „VirtualBox“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 13. September 2019, 09:11 UTC. URL:https://de.wikipedia.org/w/index.php?title=VirtualBox&oldid=192217153, (Abgerufen: 27. September 2019, 06:12 UTC).

Tools eingegangen, die in der Sicherheitsanalyse zum Einsatz kommen. Eine ausführliche Liste zu den Inhalten und auch zur Installation von AndroL4b können dieser Seite https://github.com/sh4hin/Androl4b entnommen werden.

- Drozer: Drozer ist ein Open-Source-Framework, dass speziell für die Sicherheitsanalyse von Android-Applikationen entwickelt wurde. Es dient u.a. zur Auffindung von Schwachstellen im Quellcode, Überprüfung der IPC-Mechanismen und der Überprüfung der Android-Funktionen, wie Activities, Services und Permissions. Zudem bietet es eine Exploit-Engine mit der gefundene Schwachstellen, direkt penetriert werden können. → *https* : *//github.com/FSecureLABS/drozer*

- APKtool: Mit dem APKtool können Android-Applikationen dekompiliert und analysiert werden. Zudem können mit dem APKtool Android-Applikationen manipuliert werden, da es Funktionen zum Packen und erneuten zertifizieren der Applikation bietet, damit diese lauffähig ist. → *https* : *//github.com/iBotPeaches/Apktool*

- AndroidStudio IDE: Android Studio ist die offizielle Entwicklungsumgebung von Android-Apps. Mit Android-Studio können die Applikationen statisch analysiert und manipuliert werden. Zudem bietet es einige Tools, wie z.B. Codeinspect, zur Malwareanalyse und einen Android-Emulator.
  → *https* : *//developer.android.com/studio/intro*

- Classyshark: ClassyShark ist ein Tool zur Dekompilierung des Quellcodes von Android-Applikationen und bietet Funktionen zum Durchsuchen der in der Applikation enthaltenen Dateien. ClassyShark unterstützt verschiedene Formate, darunter die Bibliotheken (.dex, .aar, .so), ausführbare Dateien (.apk, .jar, .class) und alle binären Android-XML-Dateien: AndroidManifest, Ressourcen, Layouts.
  → *https* : *//github.com/google/android − classyshark*

- Burp Suite: Die Burp-Suite ist ein Tool zur Netzwerkanalyse und der Testung von Webanwendungen. Sie enthält u.a. einen Proxy, mit dem HTTP/HTTPS-Verkehr abgefangen und HTTP-Header modifiziert werden können, bevor die Daten wieder an den Server geschickt werden. → *https* : *//portswigger.net/burp*

- Wireshark: Wireshark dient zur Netzwerkanalyse. Es ist eine frei erhältliche Software und in der Lage, Datenverkehr auf unterschiedlichen Schnittstellen, wie WLAN-oder Ethernet, zu sniffen. Zudem kann der Datenverkehr aufgenommen und analysiert werden.→ *https* : *//www.wireshark.org*

- MARA-Mobile Applikation Reverse Engineering und Analyse Framework-Framework: MARA ist ein Framework, dass umfangreiche Funktionen

zur statischen und dynamischen Analyse von mobilen Applikationen bietet. Mit MARA kann der Quellcode von Android-Applikationen dekompiliert und analysiert werden. Zudem führt MARA eine Schwachstellenanalyse durch und ordnet diese den OWASP Top10 mobile Risks zu. $\rightarrow$ *https://github.com/xtiankisutsa/MARA_Framework*

- Frida: Frida ist ein Open-Source-Framework, dass verschiedene Tools für die dynamische Analyse von Android-Applikationen bietet. Mit Frida können laufende Prozesse gedebuggt werden und es bietet eine Vielzahl von Methoden zur Code-Injection. $\rightarrow$ *https://www.frida.re/*

- Quark: Quark ist ein Open-Source-Tool, das auf Python basiert und der Auffindung von Schwachstellen in gepackten Applikationen und im Quellcode dient. $\rightarrow$ *https://github.com/linkedin/qark*

- Android PackageManager: Der Android-PackageManager ist Teil einer jeden Android-Kommandozeile, mit dem Informationen zu installierten Applikationen ausgelesen werden können. Mit dem Befehl „**$ adb shell pm**“ wird eine Übersicht zu allen Funktionen des PackageManagers ausgegeben.[Spr17]

- Android-Debug-Bridge-ADB: Die ADB ist ein Kommandozeilenprogramm und dient als Hauptschnittstelle zwischen dem Desktop-PC und dem mobilen Gerät. Mit der ADB kann das mobile Gerät vom Rechner aus gesteuert werden. Mit dem Befehl „**$ adb**“ wird eine Übersicht zu allen Funktionen der ADB ausgegeben.

5.2 Information Gathering: Automatisierte statische und dynamische Analyse

Um zu Beginn einer Sicherheitsanalyse einen groben Überblick über die Funktionen und eventuelle Schwachstellen einer Applikation zu erhalten, werden Tools empfohlen die automatisiert statische und dynamische Analysen durchführen. Manuelle Analysen des Quellcodes können unter Umstände je nach Applikation und Backend-Struktur sehr zeitaufwendig werden und erfordern fiel Erfahrung seitens der Experten.

Für die Durchführung von automatisierten statischen und dynamischen Analysen bieten sich die Tools VirusTotal und das Mobile-Security-Framework-MobSF an, die folgend kurz vorgestellt werden.

5.2.1 VirusTotal

VirusTotal[3] ist ein Online-Dienst, der unterschiedliche Dateiformate, URLs oder auch IP-Adressen mit verschiedenen Antiviren-und Malwarescannern auf schädliche Inhalte überprüft.[Vira]

Für die Analyse einer Android-App wird über folgenden Link https://www.virustotal.com/ die Weboberfläche von VirusTotal aufgerufen. Wie in Abb.5.1 zu sehen, können über die Weboberfläche die zu testende Dateien entweder mit einem Klick in das Menü auf „Choose file“ oder per „Drag&Drop-Funktion“ automatisch zur Analyse hochgeladen werden.

Abbildung 5.1: Startseite VirusTotal

Sobald die App geladen wurde, startet VirusTotal mit der Überprüfung. Nach Abschluss des Scans, wird auf der Weboberfläche ein Bericht angezeigt (siehe hierzu Abb.5.2). Im oberen Abschnitt des Berichts werden u.a. Informationen zum Befall durch möglichen Schadcode, der Paketname und die gebildete Hashsumme der App angezeigt. Weiter unten befindet sich eine Auflistung der VirusTotal-Partner, durch die eine Überprüfung der App stattgefunden hat.

[3] VirusTotal ist ein heute vom Unternehmen Google Inc. betriebener kostenloser Online-Dienst, um einzelne Dateien hochzuladen und online durch über 70 verschiedene Antivirenprogramme und Malwarescanner in derzeit 28 Sprachen analysieren zu lassen, wobei nicht jedes Programm jeden Dateityp scannen kann oder für jede Suchoption zur Verfügung steht. Seite „VirusTotal“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 23. August 2019, 11:06 UTC. URL: https://de.wikipedia.org/w/index.php?title=VirusTotal&oldid=191602230, (Abgerufen: 27. September 2019, 10:55 UTC)

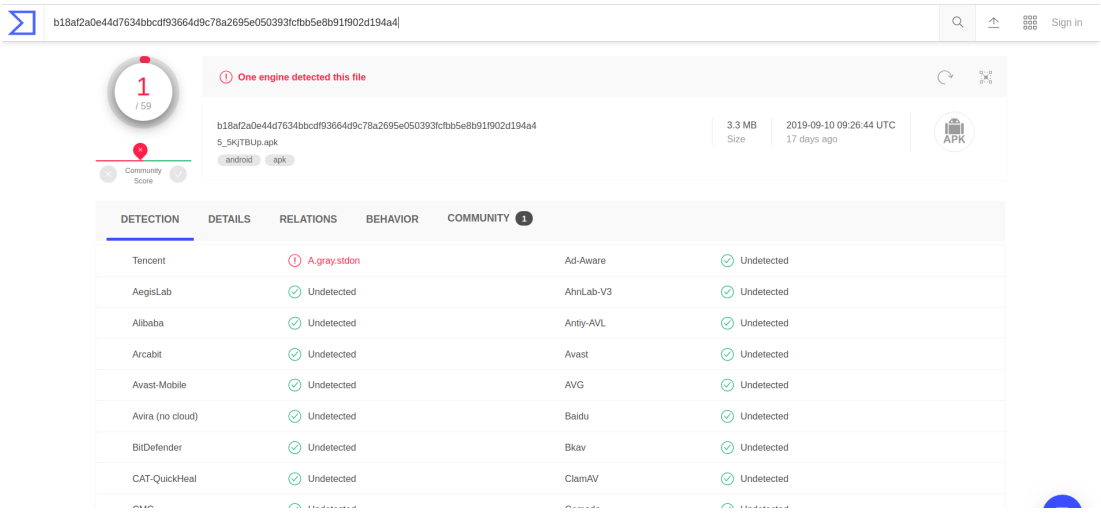


Abbildung 5.2: Ausschnitt: Detection InsecureBankv2-App

Auf der Seite „Details“ stellt VirusTotal weitere Informationen, wie z.B. die Dateiart- und Größe, das Zertifikat und Informationen zum Entwickler zur Verfügung. Zudem werden Hauptkomponenten der Applikation auf Schwachstellen geprüft. Die Abb.5.3 zeigt einen Ausschnitt der Details, wo von VirusTotal in der App vorhandenen Permissions aufgelistet werden, die ggf. ein Risiko darstellen und z.B. zu einem Abfluss von sensiblen Daten führen könnten.

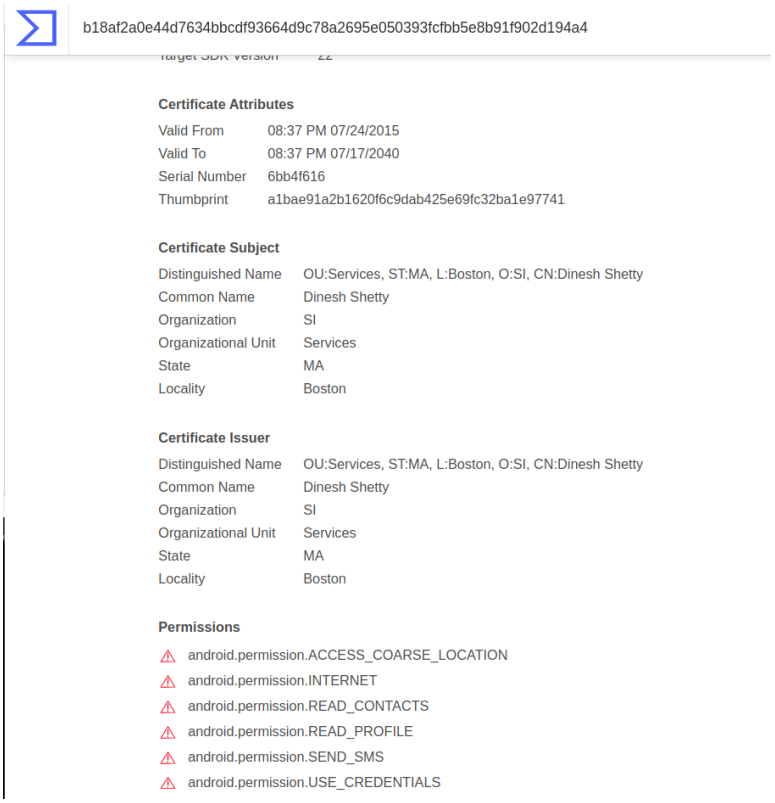


Abbildung 5.3: Ausschnitt: Details InsecureBankv2-App

Des Weiteren analysiert VirusTotal während des Scans die Beziehungen der Applikation und überprüft, inwiefern ein Datenaustausch stattgefunden hat. Darunter fallen z.B. Beziehungen zu anderen Applikationen, URLs, Domains oder IP-Adressen. In der Abb.5.4 ist zu erkennen, dass die InsecureBankv2-App eine Verbindung zu der URL „http://10.0.2.2:8888/login“ aufgenommen hat.

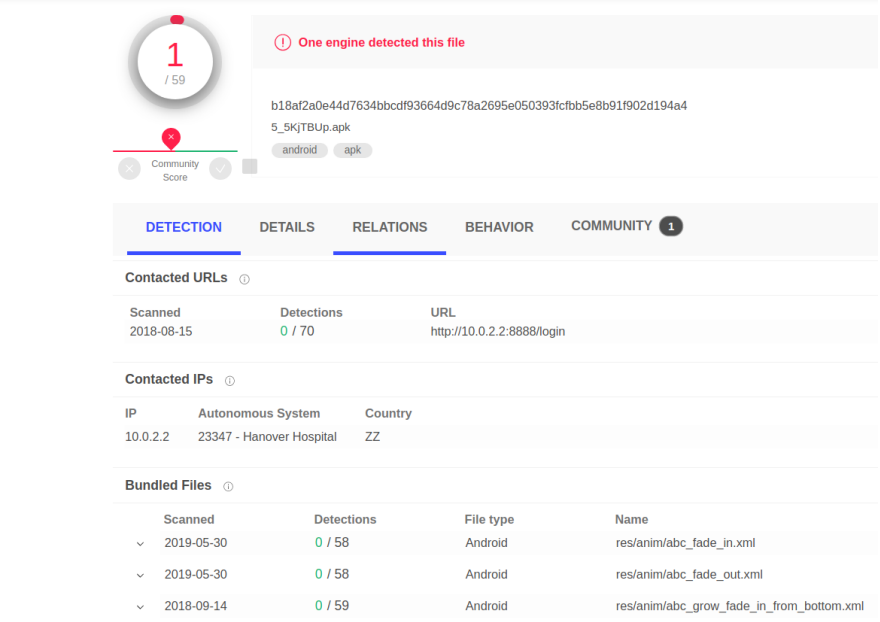


Abbildung 5.4: Ausschnitt: Relations InsecureBankv2-App

Im Bereich „Behavior“ Abb.5.5, werden die Ergebnisse aus der Überprüfung der Netzwerkkommunikation und der Hintergrundprozesse zur Verfügung gestellt. Für die Analyse des Datenverkehrs nutzt VirusTotal eine eigene Sandbox, in der u.a. eine dynamische Analyse der Prozesse und Activities, der App vorgenommen wird. Ebenso werden Dateizugriffe-und Veränderungen, wie in der Abb.5.5 zu sehen, erfasst.

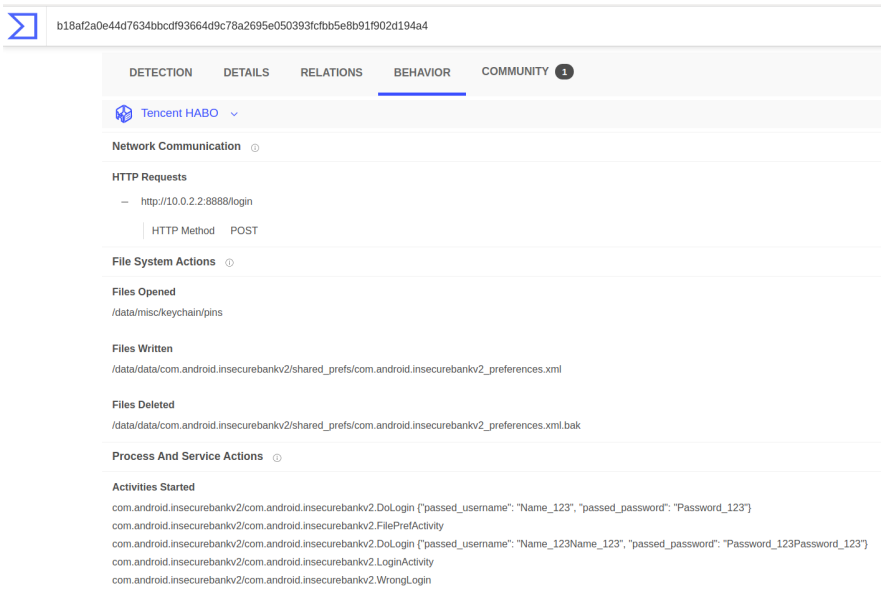


Abbildung 5.5: Ausschnitt: Behavior InsecureBankv2-App

VirusTotal bietet viele weitere Funktionen, wie z.B. eine graphische Darstellung der Beziehungen oder auch einen Kommentar-Bereich, in dem Informationen zu den gescannten Dateien ausgetauscht werden können. Im Weiteren besteht die Möglichkeit z.B. den gefundenen URLs oder IPs zu denen eine Applikation Verbindung aufgenommen hat zu folgen und diese weitergehend zu analysieren etc.. Wie in Abb.5.4 an der URL zu erkennen, handelt es sich um eine Login-Seite zu der die Banking-App versucht, eine Verbindung herzustellen. An dieser Stelle könnte überprüft werden, inwieweit Benutzerdaten mit übertragen werden und ob diese hinreichend bei der Übertragung verschlüsselt werden.
Weitere Informationen zu Funktionen von VirusTotal können der Dokumention auf der Seite: https://support.virustotal.com/hc/en-us/categories/360000162878-Documentation, entnommen werden.
Bei der Nutzung von VirusTotal sollte dringend beachtet werden, dass die Applikation ins Internet geladen wird und alle Informationen öffentlich zugänglich sind und gespeichert werden. Um trotz alledem nicht auf die Funktionen von VirusTotal verzichten zu müssen, besteht die Möglichkeit, von der Applikationen mit dem Befehl **$ sha256sum <AppPackage>**eine Hashsumme (vgl. hierzu Abb.5.6 und Abb.5.7) zu bilden und diese dann bei VirusTotal hochzuladen. VirusTotal nimmt dann einen Vergleich mit der in der Datenbank gespeicherten Berichten vor. So kann verhindert werden, dass sensible oder vertrauliche Daten bzw. firmeninterne Entwicklungen, offengelegt werden.[Virb]

```
root@anaken-Aspire-E1-572G:/home/anaken/Schreibtisch/Android-InsecureBankv2-master# sha256sum InsecureBankv2.apk
b18af2a0e44d7634bbcdf93664d9c78a2695e050393fcfbb5e8b91f902d194a4  InsecureBankv2.apk
root@anaken-Aspire-E1-572G:/home/anaken/Schreibtisch/Android-InsecureBankv2-master#
```

Abbildung 5.6: Bildung Hash-Summe

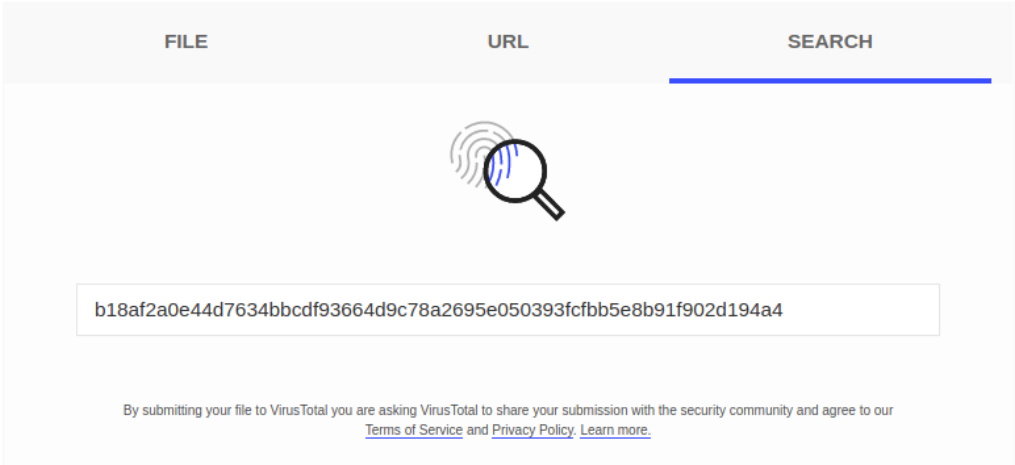


Abbildung 5.7: VirusTotal Hash-Suche

5.2.2 Mobile-Security-Framwork-MobSF

Das MobSF ist ein in Python entwickeltes Open-Source-Testframework zur automatisierten Analyse von mobilen Applikationen. Es soll u.a. zur Unterstützung bei Sicherheitsanalysen oder Pentests dienen. Mit MobSF können automatisiert sowohl statische als auch dynamische Analysen durchgeführt werden. Zudem verfügt es über einen integrierten Emulator, eine API mit der die Analyse automatisch ausgeführt werden kann und eine Web-API-Fuzzing Funktion, das HTTPTool, das u.a. die Analyse von Headern und die Erkennung von , SSRF (Server-Side Request Forgery) oder Path Traversal, unterstützt.
MobSF kann lokal auf dem Rechner installiert werden. Da es über keine Zugriffsverwaltung verfügt und somit über das Dashboard leicht zugänglich ist, sollte es möglichst nur in einem lokalen Netzwerk mit gut gesichertem Internetzugang installiert werden. Ausführliche Informationen zur Installation und die Installationsdatei können von der Seite: `https://github.com/MobSF/Mobile-Security-Framework-MobSF`, abgerufen werden. Das Tool wird regelmäßig aktualisiert.[DA16]

Wie in Abb.5.8 zu erkennen ist, können über die Startseite automatisiert die Apps hochgeladen werden und das MobSF startet mit der statischen Analyse der Applikation.

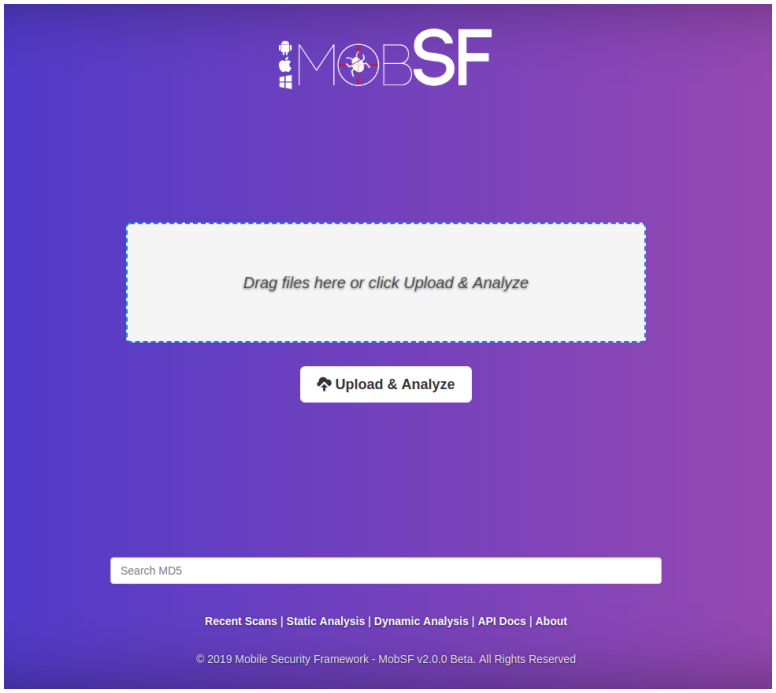


Abbildung 5.8: Startseite MobSF

Folgende Funktionen und Komponenten werden u.a bei der statischen Analyse automatisiert geprüft(vgl. hierzu Abb.5.9 und Abb.5.10):

- Ermittlung und Analyse der Activities, Services, Receiver und Providers
- Dekompilierung der Binärdatei in Java-und Smali-Quellcode
- Analyse der Manifest-Datei
- Zertifikat-und Signaturanalyse
- Analyse und Klassifizierung der Permissions
- Analyse der nativen Bibliotheken (z.B. Fehler im Code)
- Überprüfung möglicher SDK-Manipulationen
- Erkennung von Codefehlern (z. B. Protokolle mit vertraulichen Informationen, nicht gelöschte TMP-Dateien)
- Überprüfung auf Schadcode mit VirusTotal und APKiD
- Erkennung der externen API-Kommunikation (URLs, E-Mails)
- Erkennung sensibler Informationen im Code
- Erkennung von Trackern

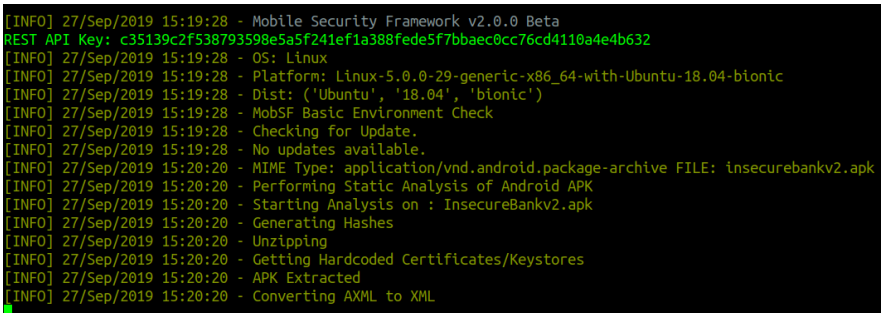


Abbildung 5.9: Ablauf statische Analyse MobSF InsecureBankv2-App

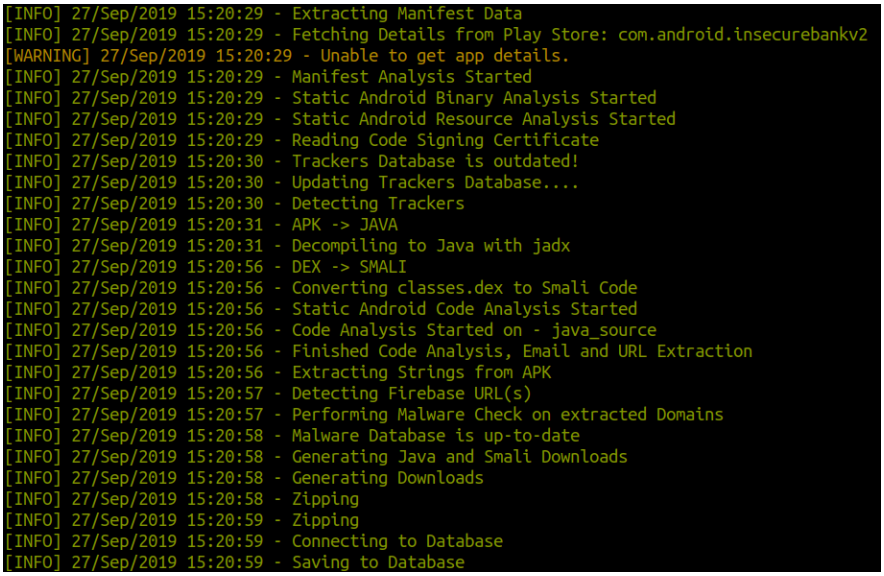


Abbildung 5.10: Ablauf statische Analyse MobSF InsecureBankv2-App II

Nach Abschluss der Analyse generiert das MobSF auf der Weboberfläche einen ausführlichen Bericht mit den Ergebnissen. Der Bericht enthält alle Informationen zur analysierten App, die in Kategorien unterteilt sind. Die Kategorien basieren auf den Bereichen, in denen eine Analyse stattgefunden hat. Über das Menüband auf der linken Seite (siehe hierzu Abb.5.11), kann durch den Bericht navigiert werden.

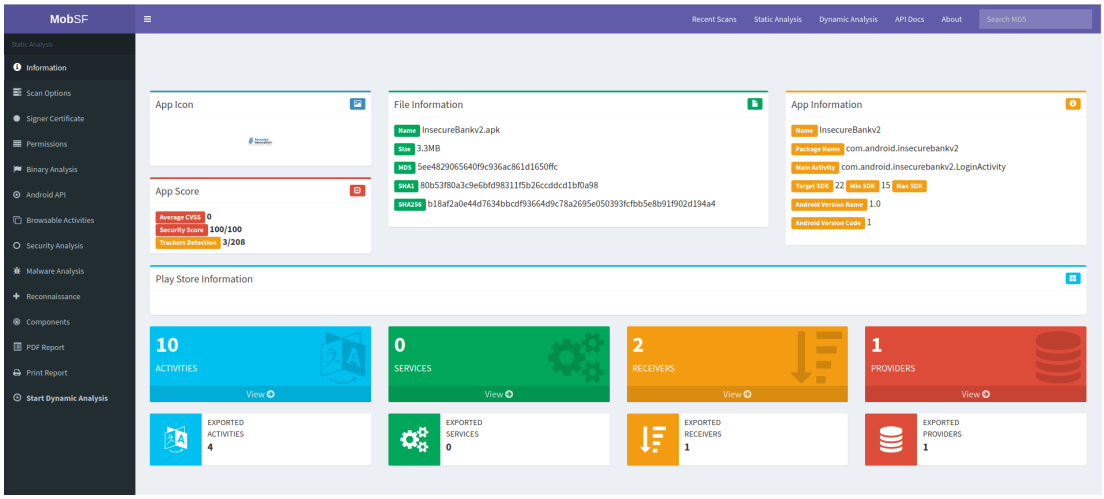


Abbildung 5.11: Weboberfläche MobSF

- **1.Informationen**: Im Bereich „Informationen“ werden, wie in Abb.5.12 zu sehen, grundlegende Daten zur App angezeigt. Neben einem App-Score, der einen Überblick zu gefundenen Schwachstellen gibt, werden Datei-Informationen, wie das App-Icon, der App-Name, die Hash-Summe, die Datei-Größe und die unterstützte Android-Betriebsversionen der App dargestellt. Die InsecureBankv2-App hat eine Score von 100, was bedeutet, dass die App sehr unsicher ist und der Einsatz ein sehr hohes Sicherheitsrisiko darstellt. Des Weiteren wurden drei Tracker identifiziert. Unter den Informationen zur App werden die einzelnen Komponenten aufgelistet, die bei der App erkannt und auf Schwachstellen überprüft wurden.

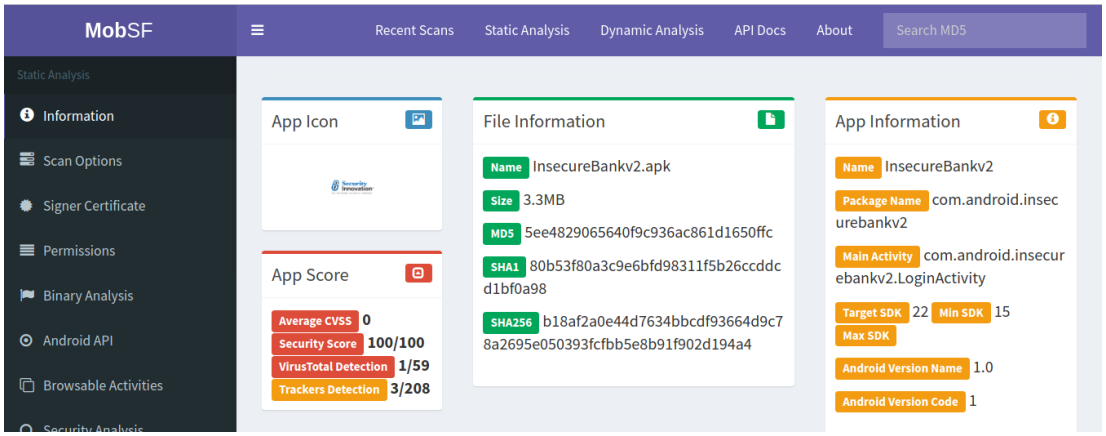


Abbildung 5.12: Bericht App-Informationen InsecureBankv2-App

- **2.Scan-Options**: Über den Bereich Scan-Options können sowohl die Manifest-Datei als auch der dekompillierte Java-und Smali-Code eingesehen und abgerufen werden (vgl. hierzu 5.13). Zudem besteht die Möglichkeit die App noch einmal zu scannen oder automatisiert die dynamische Analyse zu starten.

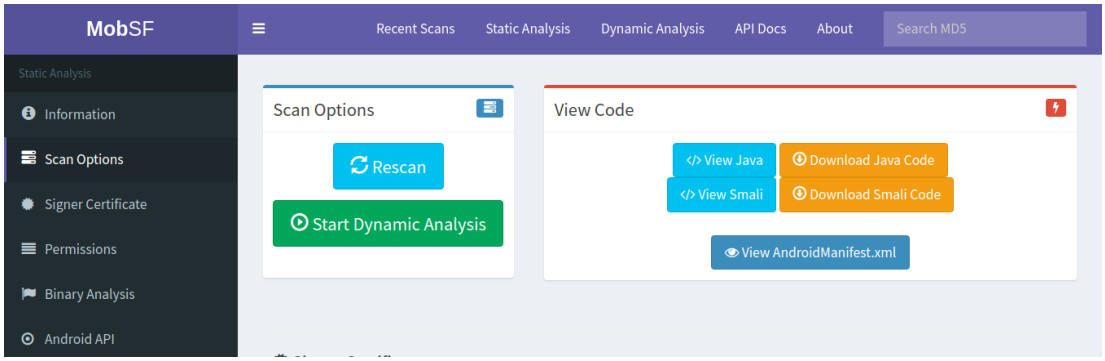


Abbildung 5.13: Bericht App-Scan Options InsecureBankv2-App

- **3.Signer Certificate**: Bei der statischen Analyse werden automatisiert die Signaturen der App geprüft und die Quelle der App verifiziert. Wenn eine App mit einem Zertifikat aus der SHA2-Familie signiert wurde, wird es von MobSF als sicher eingestuft. In der Abb.5.12 ist zu sehen, dass bei der Analyse des Zertifikats der InsecureBankv2-App keine Schwachstellen gefunden werden konnten und die App mit einem sicheren Hash, SHA256, signiert wurde. Android erfordert (vgl. hierzu Kap.3.3.4 Signierung der Applikation), dass alle .apk-Dateien mit einem Zertifikat digital signiert werden. Wenn eine APK signiert wird, wird ein Public-Key-Zertifikat angehängt. Dieses Zertifikat verknüpft die .apk-Datei eindeutig mit dem Entwickler und dessen privaten Schlüssel.
  Darüber hinaus wird die Gültigkeitsdauer des Zertifikats angezeigt und es wurde überprüft mit welchem Signaturschema die App signiert wurde. Die InsecureBankv2-App wurde mit dem v1-Schema (Jar-Signierung) signiert.

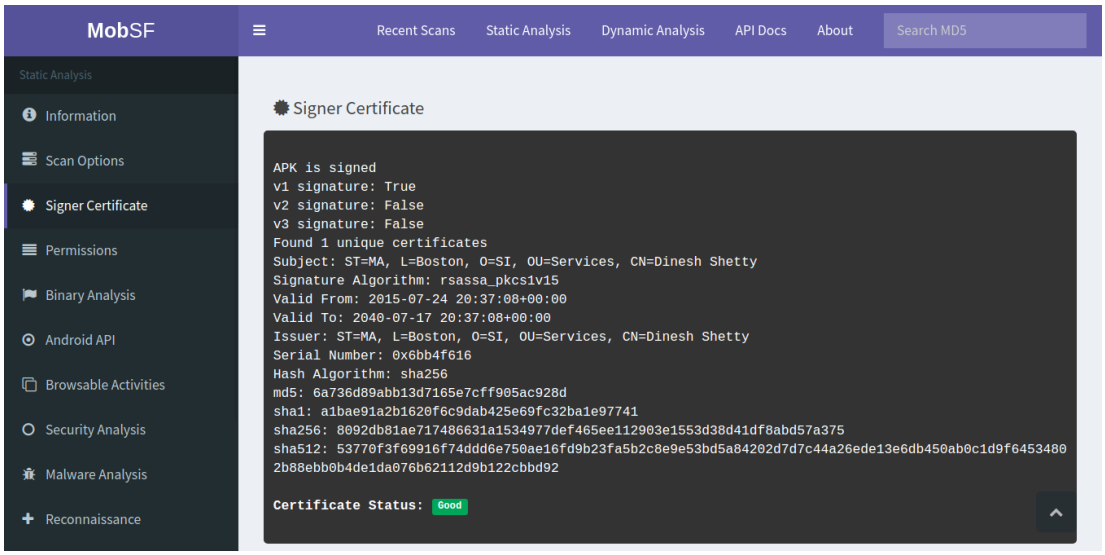


Abbildung 5.14: Bericht App-Signer Certificate InsecureBankv2-App

- **3.Permissions**: Das MobSF analysiert die Permissions der App und ermittelt den Status entsprechend der Kritikalität, mit einer kurzen Erklärung. Dabei ist zu

beachten, dass es eine Reihe von Permissions gibt, die grundsätzlich als kritisch eingestuft werden. Wie z.B. in Abb.5.15 zu erkennen, enthält die InsecureBankv2-App die Permission „android.permission.INTERNET“ die mit der Kritikalitätsstufe „dangerous“ eingestuft wurde, da sie der App vollen Zugang zum Internet gewährt. Das eine Banking-App Zugang zum Internet benötigt, damit der Nutzer seine Bankgeschäfte abwickeln kann, ist erstmal nichts ungewöhnliches. An dieser Stelle wird ein gutes Verständnis der Architektur einer Applikation und auch des Betriebssystems benötigt, um die Kritikalität dieser Permission in ihrem Gesamtkontext beurteilen zu können. Überdies sollte die Konfiguration der Permission in der Manifest-Datei überprüft werden.

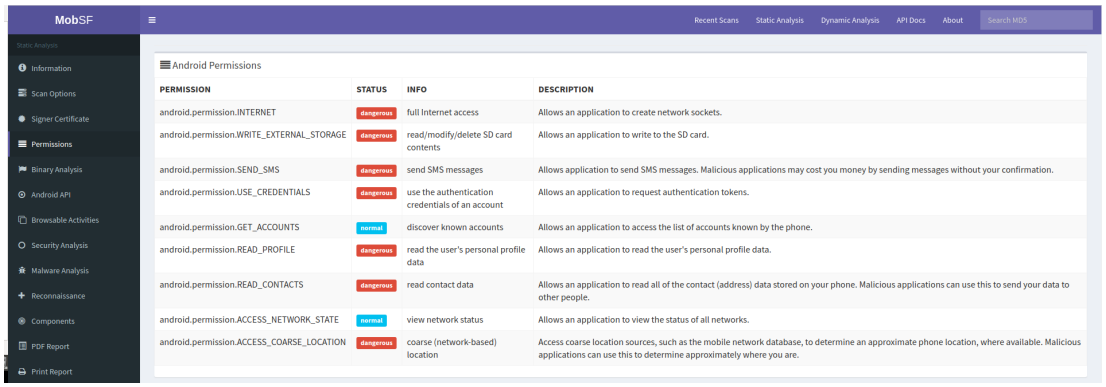


Abbildung 5.15: Bericht App-Signer Certificate InsecureBankv2-App

- **4.Security & Malware Analysis**: Im Bereich Security Analysis werden die Ergebnisse der Manifest-Analyse sowie der Code-und Fileanalyse dargestellt.
  Bei der Überprüfung des Manifest werden alle deklarierten Komponenten, Funktionen und Methoden einer Schwachstellenanalyse unterzogen. Diese Komponenten können bei falscher Konfiguration zu Sicherheitsproblemen führen.
  MobSF hat im Manifest der App verschiedene Schwachstellen erkannt, dann bewertet und Empfehlungen zu Behebung dieser bereitgestellt.
  Wie in Abb.5.16 zu erkennen, wurde die Activity „com.android.insecurebankv2.PostLogin)“ aufgrund der Konfiguration „android:exported=true[4]“ als hohes Risiko eingestuft. Mit der Funktion„android:exported“ kann gesteuert werden, ob der Broadcast Receiver Nachrichten von Quellen außerhalb der App empfangen kann. Wenn der Wert „true“ gesetzt ist, kann der Broadcast Receiver Nachrichten außerhalb der App empfangen. Steht der Wert auf „False“ können vom Broadcast Receiver nur Nachrichten empfangen werden, die von Komponenten derselben App oder von Apps mit derselben Benutzer-ID gesendet wurden. Somit könnten andere Apps Zugriff auf die Daten von der InsecureBankv2-App haben.

[4] https://developer.android.com/guide/topics/manifest/service-element

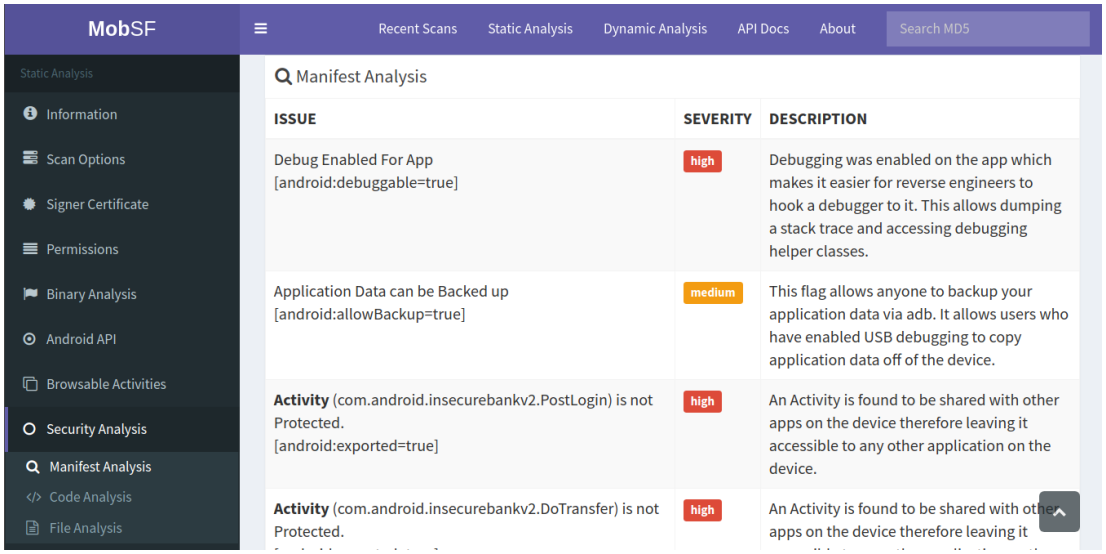


Abbildung 5.16: Bericht Manifest Analysis InsecureBankv2-App

Neben der Analyse des Manifests, hat das MobSF den Java-Code und die Dateien der App auf Schwachstellen untersucht. Sowohl der Java-Code als auch die Dateien können sensible Informationen, wie Benutzerdaten, Kennwörter oder Schlüssel enthalten.

Bei der Analyse wurden zuerst potentielle Schwachstellen identifiziert und dann deren Schweregrad und die Dateien bestimmt, in denen diese Art von Schwachstelle gefunden wurde. Die Bewertung der Schwachstelle basiert auf einem allgemein gebräuchlichen Schwachstellenbewertungssystem, dem CVSS-Common Vulnerability Scoring System[5]. Dabei wird der Schwachstelle eine CVSS-Basisbewertung zwischen 0.0 und 10.0 zugewiesen. Über die CWE-Common Weakness Enumeration[6] können mehr Informationen zu der angezeigten Schwachstelle ermittelt werden.

Da bei der Schwachstellenanalyse auch Fehler auftreten können, sogenannte „False Positives“, was bedeutet, dass fälschlicherweise eine Schwachstellen identifiziert

[5]Das Common Vulnerability Scoring System (wörtlich übersetzt: „Allgemeines Verwundbarkeitsbewertungssystem“), abgekürzt CVSS, ist ein Industriestandard zur Bewertung des Schweregrades von möglichen oder tatsächlichen Sicherheitslücken in Computer-Systemen. Im CVSS werden Sicherheitslücken nach verschiedenen Kriterien, sogenannten Metrics, bewertet und miteinander verglichen, so dass eine Prioritätenliste für Gegenmaßnahmen erstellt werden kann. CVSS ist selbst kein System zur Warnung vor Sicherheitslücken, sondern ein Standard, um verschiedene Beschreibungs- und Messsysteme miteinander kompatibel und allgemein verständlich zu machen. Seite „CVSS“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 28. Dezember 2018, 13:25 UTC. URL:https://de.wikipedia.org/w/index.php?title=CVSS&oldid=184123027, (Abgerufen: 28. September 2019, 23:45 UTC)

[6]Die Common Weakness Enumeration (CWE) ist ein Kategoriesystem für Softwareschwächen und Schwachstellen. Es wird von einem Community-Projekt unterstützt, dessen Ziel es ist, Softwarefehler zu verstehen und automatisierte Tools zu erstellen, mit denen diese Fehler identifiziert, behoben und verhindert werden können. Wikipedia contributors. Common Weakness Enumeration [Internet]. Wikipedia, The Free Encyclopedia; 2019 Jan 7, 02:23 UTC [cited 2019 Sep 28]. Available from: https://en.wikipedia.org/w/index.php?title=Common_Weakness_Enumeration&oldid=877178659.

wurde, müssen die Angaben nochmal manuell im Code geprüft werden.[DA16]

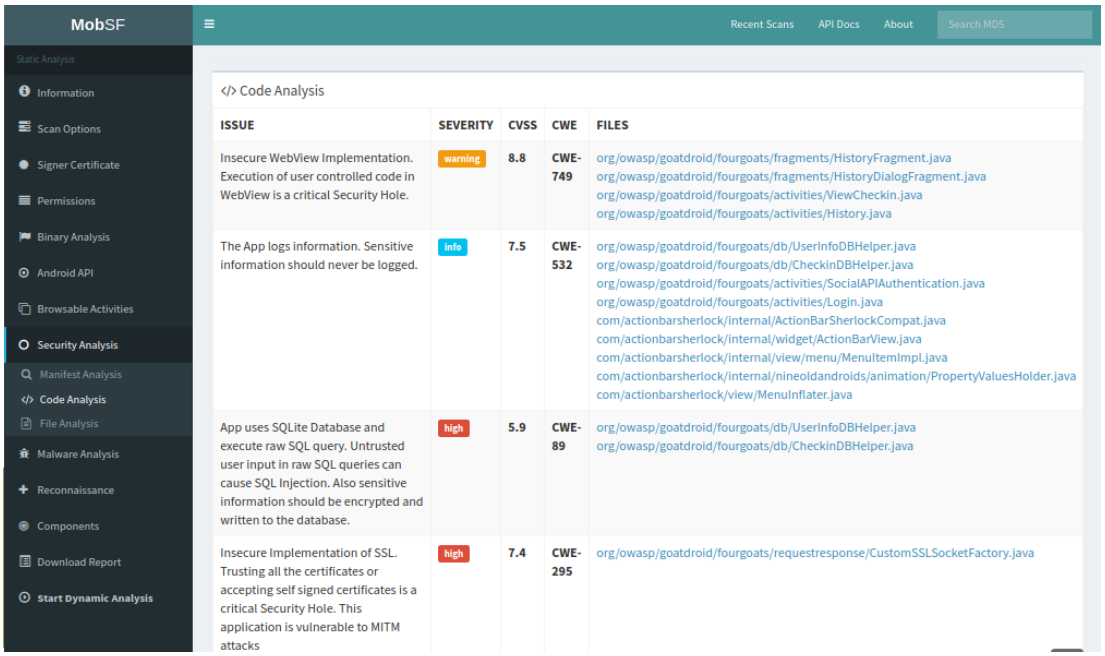


Abbildung 5.17: Bericht Code Analysis InsecureBankv2-App

Des Weiteren werden App-Dateien die Zusatzfunktionen APKiD[7] und VirusTotal (vgl. hierzu Kap.5.2.1) einer Malware-Analyse unterzogen. Mit dem APKiD kann u.a. ermittelt werden, wie eine App erstellt wurde und es ermittelt Compiler, Packer und Ofuscator.

- **5.Reconnaissance**: Im Bereich Reconnaissance werden zum einen durch die Analyse ermittelte URLs, IP-und Emailadressen angezeigt sowie die Dateien und Dateipfade, in denen sie gespeichert sind oder über die sie aufgerufen werden können. Zum anderen werden mögliche Firebase DBs analysiert und die App auf Tracker überprüft.
  Bei der String Analyse werden die Text-Dateien analysiert, die sich im Verzeichnis „res“ befinden. Die Dateien können sensible Informationen enthalten.

[7] https://github.com/rednaga/APKiD

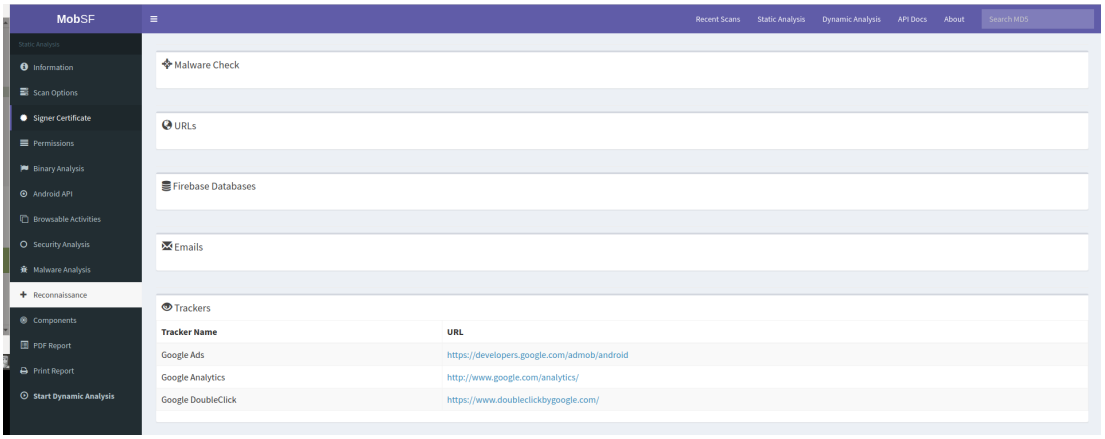


Abbildung 5.18: Bericht Reconnaissance InsecureBankv2-App

- **Components**: Unter Components wird eine vollständige Liste der Komponenten der App , wie Activities, Services, Content Provider, Receiver und importierten Bibliotheken sowie Dateien angezeigt.

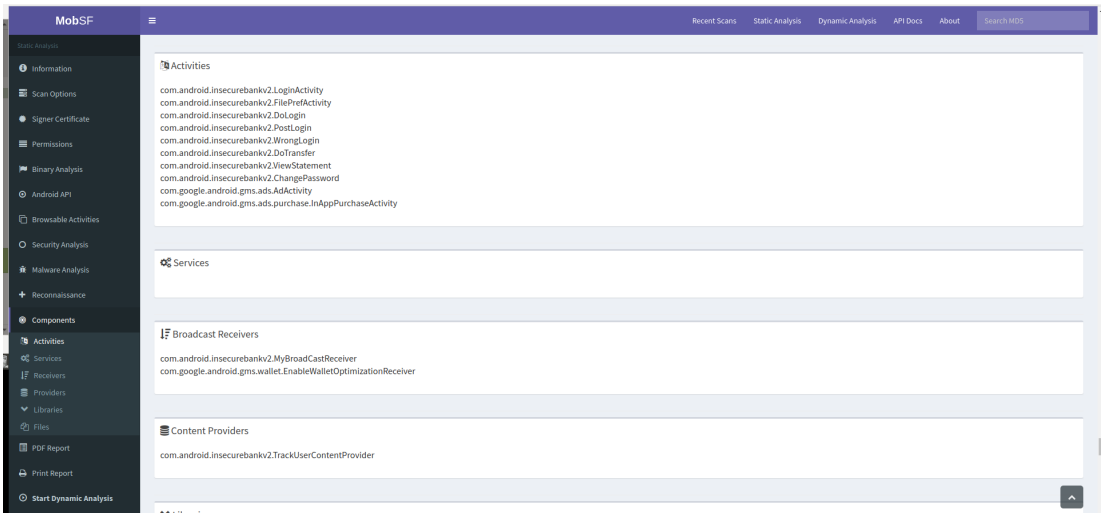


Abbildung 5.19: Bericht Components InsecureBankv2-App

Wenn die statische Analyse der App abgeschlossen wurde, kann über die Weboberfläche die dynamische Analyse gestartet werden.
Mit dem Start der dynamischen Analyse wird gleichzeitig eine Verbindung zum Android-Emulator erstellt(siehe hierzu auch Kap.5.1.2 „Android-Genymotion-Emulator“) und die App automatisch auf dem virtuellen Test-Gerät installiert und ausgeführt. Parallel dazu startet der Dynamic Analyzer einen HTTPS-Intercepting-Proxy, um den Web-Verkehr zu erfassen und zu analysieren sowie den MobSF Agenten, der Dienste und Services die im Hintergrund der App laufen, erfasst. (vgl. hierzu Abb.5.20).

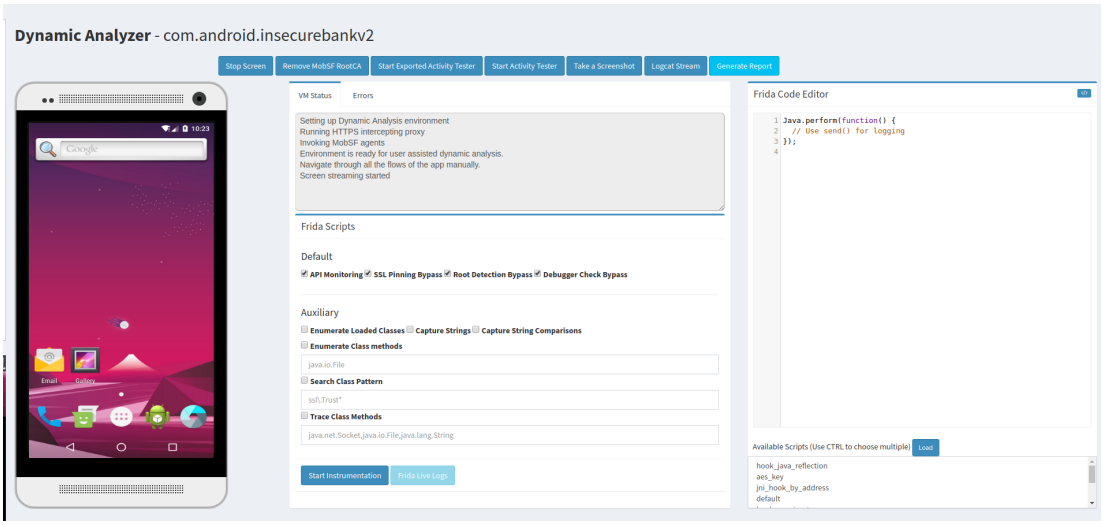


Abbildung 5.20: Dynamic Analysis InsecureBankv2-App

Des Weiteren können mit dem Exported Activity- und Activity-Tester zur Laufzeit die Activities auf Schwachstellen überprüft werden. Wie schon im Zuge der statischen Analyse erläutert können Activities,je nachdem wie sie konfiguriert wurden, applikationsübergreifend miteinander kommunizieren oder durch eine anderen Anwendung aufgerufen werden. Zudem können übertragenen auch in den vorhandenen Datenbanken zwischengespeichert werden. Das kann ggf. zu einem ungewollten Datenabfluss führen, aufgrund dessen die Activities auch zur Laufzeit analysiert werden sollten, um entsprechend Modifizierungen zur Sicherung vornehmen zu können. Weitere Informationen zur Funktion des Activity-Testers, können in der Test_Common.py im Ordner DynamicAnalyzer/views/android, gefunden werden.

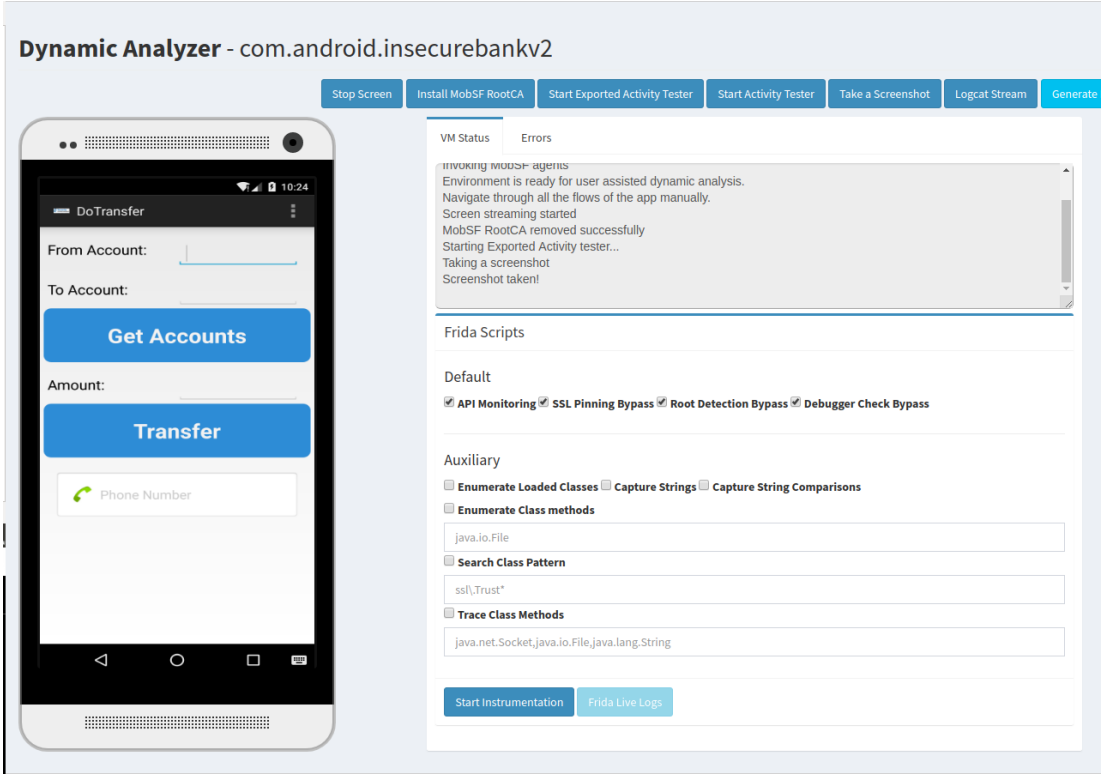


Abbildung 5.21: Dynamic Analysis Exported Activity Tester

Über die Funktion Logcat Stream werden von MobSF App- und Systemprozesse zur Laufzeit geloggt.

```
09-28 22:19:04.095 267 272 D FFMPEG : android source close\n'
09-28 22:19:04.095 267 272 I FFmpegExtractor: sniff through BetterSniffFFMPEG success\n'
09-28 22:19:04.095 267 272 D FFmpegExtractor: ffmpeg detected media content as 'audio/vorbis' with confidence 0.08\n"
09-28 22:19:04.107 268 268 I MediaPlayerService: MediaPlayerService::getOMX\n'
09-28 22:19:04.110 577 898 I OMXClient: MuxOMX ctor\n'
09-28 22:19:04.112 664 897 E WVMExtractor: Failed to open libwvm.so: dlopen failed: library "libwvm.so" not found\n'
09-28 22:19:04.117 265 743 I OMXMaster: makeComponentInstance(OMX.google.vorbis.decoder) in mediacodec process\n'
09-28 22:19:04.137 655 878 I LatinIME:LogUtils: Dictionary info: dictionary = userunigram.en_US ; version = 1569723543 ; date = ?\n'
09-28 22:19:04.138 655 878 E ActivityThread: Failed to find provider info for user_dictionary\n'
09-28 22:19:04.138 655 878 I LatinIME:LogUtils: Dictionary info: dictionary = UserHistoryDictionary.en_US ; version = 1569723544 ; date = ?\n'
09-28 22:19:04.151 267 267 I FFmpegExtractor: android-source:0xe8e5c000\n'
09-28 22:19:04.151 267 267 D FFMPEG : android source begin open\n'
09-28 22:19:04.151 267 267 D FFMPEG : android open, url: android-source:0xe8e5c000\n'
09-28 22:19:04.151 267 267 D FFMPEG : ffmpeg open android data source success, source ptr: 0xe8e5c000\n'
09-28 22:19:04.151 267 267 D FFMPEG : android source open success\n'
09-28 22:19:04.187 267 267 I FFMPEG : Input #0, ogg, from 'android-source:0xe8e5c000':\n"
09-28 22:19:04.187 267 267 I FFMPEG : Duration: 00:00:00.23, start: 0.000000, bitrate: 293 kb/s\n'
09-28 22:19:04.187 267 267 I FFMPEG : Stream #0:0: Audio: vorbis, 44100 Hz, stereo, fltp, 160 kb/s\n'
09-28 22:19:04.187 267 267 I FFmpegExtractor: FFmpegExtrator, url: android-source:0xe8e5c000, format_name: ogg, format_long_name: Ogg\n'
09-28 22:19:04.187 267 267 I FFmpegExtractor: list the formats suppoted by ffmpeg: \n'
09-28 22:19:04.187 267 267 I FFmpegExtractor: ========================================\n'
09-28 22:19:04.187 267 267 I FFmpegExtractor: ========================================\n'
09-28 22:19:04.187 267 267 I FFmpegExtractor: adjust mime(application/ogg -> audio/vorbis)\n'
09-28 22:19:04.187 267 267 D FFmpegExtractor: suppoted codec(vorbis) by official Stagefright\n'
09-28 22:19:04.187 267 267 D FFMPEG : android source close\n'
09-28 22:19:04.187 267 267 I FFmpegExtractor: sniff through BetterSniffFFMPEG success\n'
09-28 22:19:04.187 267 267 D FFmpegExtractor: ffmpeg detected media content as 'audio/vorbis' with confidence 0.08\n"
09-28 22:19:04.220 268 268 I MediaPlayerService: MediaPlayerService::getOMX\n'
09-28 22:19:04.240 664 906 I OMXClient: MuxOMX ctor\n'
09-28 22:19:04.247 655 878 I LatinIME:LogUtils: Dictionary info: dictionary = main:en ; version = 54 ; date = 1414726273\n'
09-28 22:19:04.251 577 593 D Sensors : batch handle 1, but there is no batch sensor in genymotion\n'
09-28 22:19:04.251 577 593 W KeyguardServiceDelegate: onScreenTurningOn(): no keyguard service!\n'
09-28 22:19:04.252 577 600 I ActivityManager: Displayed com.android.settings/.FallbackHome: +2s258ms\n'
09-28 22:19:04.254 265 743 I OMXMaster: makeComponentInstance(OMX.google.vorbis.decoder) in mediacodec process\n'
09-28 22:19:04.337 267 838 I FFmpegExtractor: android-source:0xe763b000\n'
09-28 22:19:04.337 267 838 D FFMPEG : android source begin open\n'
09-28 22:19:04.337 267 838 D FFMPEG : android open, url: android-source:0xe763b000\n'
09-28 22:19:04.337 267 838 D FFMPEG : ffmpeg open android data source success, source ptr: 0xe763b000\n'
09-28 22:19:04.337 267 838 D FFMPEG : android source open success\n'
```

Abbildung 5.22: Dynamic Analysis Logcat Stream

Genau wie bei der statischen Analyse, bietet auch die Weboberfläche der dynamischen

Analyse eine Option zur Berichtsgenerierung. Auch hier ist der Bericht, entsprechend der analysierten Bereiche, in verschiedene Kategorie unterteilt.

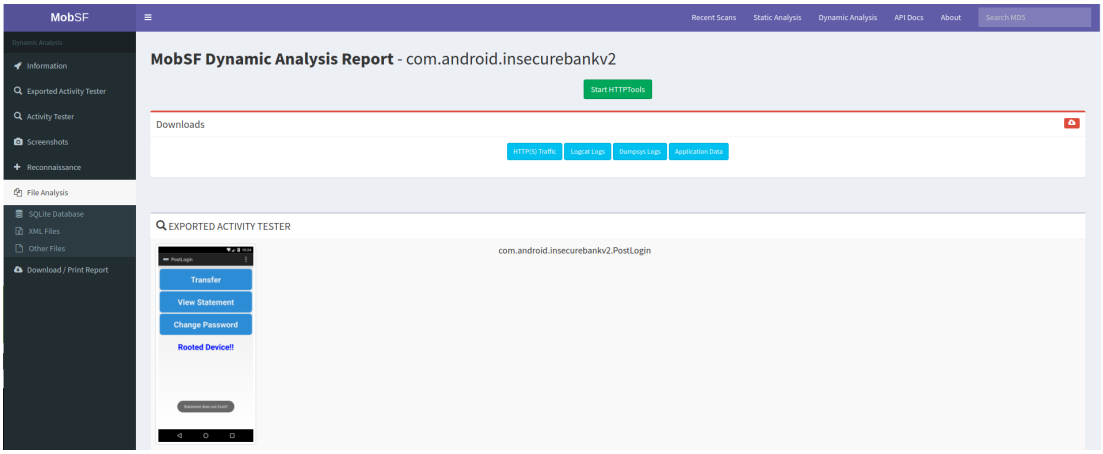


Abbildung 5.23: Dynamic Analysis Bericht

Im Bereich Download können der HTTPS-Traffic, die Logcat-und Dumpsys-Logdaten sowie die Applikations Daten eingesehen, wie die databases, shared_prefs oder der cache, und für weitere Analysen heruntergeladen werden. In den Dumpsys-Logdaten werden z.B. Informationen zu verschiedenen Systemdiensten bereitgestellt. Ebenso können die aus der File-Analyse gewonnen Daten, wie .xml-oder.db-Daten eingesehen werden, die unter Umständen sensible Daten, wie Benutzername und Passwort oder andere persönliche Daten, enthalten. Gefundene URLs und EMails werden angezeigt und einer Malware-Analyse unterzogen und unter Clipboarddump werden zur Laufzeit gesammelte Daten aus dem Zwischenspeicher dargestellt.

Das MobSF bietet viele Funktionen für die statische und dynamische Analyse, die direkt im Quellcode z.B. im dynamic_analyser.py oder enviroment.py, direkt einsehbar sind. Die können im Ordner DynamicAnalyzer des MobSF eingesehen werden. Der Code wird ausführlich dokumentiert.
Aufgrund des begrenzten Umfangs dieser Arbeit konnten leider nicht alle Funktionen in voller Gänze dargestellt werden. Weitere Informationen zum MobSF und vor allem zur dynamischen Analyse, können der Seite: https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/1.-documentation, entnommen werden.

5.3 Android-Sicherheitsanalyse- Testung der Anforderungen an die Datenspeicherung und Datenschutz

Android-Apps speichern viele Daten im Dateisystem. Jeder, der physischen Zugriff auf das Gerät hat, kann auf einfache Weise viele vertrauliche Informationen wie Fotos, Passwörter, GPS-Standorte, den Browserverlauf oder Unternehmensdaten einsehen, wenn diese nicht hinreichend gesichert bzw. verschlüsselt wurden. Zudem können Daten aufgrund falscher Konfigurationen der IPC-Mechanismen, anderen Apps auf dem Smartphone ungewollt zur Verfügung gestellt werden.
Der in diesem Kapitel simulierte Prüffall entspricht u.a.den Anforderung des BSI zur sicheren Speicherung lokaler App Daten (siehe hierzu Referenztablle BSI-APP.1.4.A7) und dem OWASP TOP 10 Mobile Risk zur unsicheren Datenspeicherung.

Bevor mit dem Analyseprozess gestartet wird, wird kurz das Android-Dateisystem erklärt. Android verwendet Unix-ähnliche Dateisysteme, um die Daten lokal zu speichern. Mit dem Befehl **$ cat /proc/filesystem** kann das Dateisystem eines Androidsystem eingesehen werden und mit dem Befehl **$ cat /proc/mtd** können die Partitionen abgefragt werden (siehe hierzu Abb. 5.24 und Abb. 5.25). Wie in Abb.5.24 zu erkennen, sind in einem Android-System verschiedene Dateisysteme im Einsatz, wie z.B. ext oder yaffs.[OWA, HWA+17]

```
root@l4b:/home/andro# adb shell cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   tmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   sockfs
nodev   pipefs
nodev   anon_inodefs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext2
        ext4
nodev   ramfs
        vfat
        msdos
nodev   nfsd
        fuseblk
nodev   fuse
nodev   fusectl
        yaffs
        yaffs2
nodev   selinuxfs
nodev   mtd_inodefs
root@l4b:/home/andro#
```

Abbildung 5.24: Übersicht Filesystem Emulator AndroL4b

```
root@l4b:/home/andro# adb shell cat /proc/mtd
dev:    size   erasesize  name
mtd0: 22600000 00000000 "system"
mtd1: 32000000 00000000 "userdata"
mtd2: 04200000 00000000 "cache"
root@l4b:/home/andro#
```

Abbildung 5.25: Übersicht Partitionen Emulator AndroL4b

In der folgenden Analyse werden von Android häufig verwendete Datenspeicher manuell statisch und dynamisch analysiert:

- **Shared-Preferences**: Speichert Einstellungen, Benutzerdaten etc. im .xml-Format.
- **Datenbanken**: Enthalten die SQLite-Daten.
- **Interner/Externer Speicher**: Dienen der Speicherung der allgemeinen App-Daten.

2.1 Die App speichert sensible Daten wie personenbezogene Daten, Anmeldedaten oder kryptographische Schlüssel unter Nutzung der vom jeweiligen Betriebssystem angebotenen sicheren Speichermechanismen

2.2 Es werden keine sensiblen Daten außerhalb des App-Containers oder außerhalb des vom jeweiligen Betriebssystem angebotenen sicheren Speichermechanismus abgelegt.

In den Shared Preferences (vgl. hierzu Kap.2.3.7„Shared Preferences“) werden je nach Einstellung sensible und private Daten, wie z.B. Benutzernamen und Passwörter im Klartext gespeichert, auf die u.a. andere Dienste oder Apps lesend oder schreibend Zugriff haben.
Um zu überprüfen, welche Daten in der Shared Preferences gespeichert werden, wird über die ADB auf das mobile Gerät zugegriffen und im Dateiverzeichnis der Applikation nach vorhandenen shard_prefs/<name.xmls gesucht (vgl. hierzu Abb.5.26).
Mit dem Befehl **$ ls -al /data/data/jakhar.aseem.diva/shard_prefs** kann der Ordner eingesehen werden.

```
root@generic:/ # ls -al data/data/jakhar.aseem.diva/shared_prefs/
-rw-rw---- u0_a53   u0_a53        151 2019-09-16 05:51 jakhar.aseem.diva_preferences.xml
root@generic:/ #
```

Abbildung 5.26: Beispiel Zugriff auf die Shared-Preferences

Die im Ordner befindlichen xml-Dateien können dann über die Kommandozeile mit dem Befehl **$cat /data/data/jakhar.aseem.diva/shard_prefs/<name.xml>** ausgelesen werden (vgl. hierzu Abb.5.26).

```
root@generic:/ # ls -al data/data/jakhar.aseem.diva/shared_prefs/
-rw-rw---- u0_a53   u0_a53        151 2019-09-16 05:51 jakhar.aseem.diva_preferences.xml
har.aseem.diva_preferences.xml                                                  <
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="user">admin</string>
    <string name="password">1234</string>
</map>
root@generic:/ #
```

Abbildung 5.27: Prüfung der Shared-Preferences der DIVA-App

In der Abbildung5.27 ist zu erkennen, dass die DIVA-App in der preferences.xml einen Benutzernamen „admin“ und ein Passwort „1234“ im Klartext speichert.
In einem weiteren Schritt muss geprüft werden, welche Dienste oder Apps ggf. lesend oder schreibend Zugriff auf die preferences.xml haben. Dafür wird überprüft, mit welchem Modi die Funktion getSharedPreferences() im Datei-Code konfiguriert wurde. Wenn auf dem mobilen Gerät ältere Versionen von Anroid (unter Android API 7.0) betrieben werden, sollte der Datei-Code auf die Modi MODE_WORLD_Readable oder MODE_WORLD_Writeable untersucht werden, da diese allen Diensten und Applikationen lesend und schreibend Zugriff gewähren. Ab API 7.0 werden diese Modi nicht mehr unterstützt. Hier wird überprüft, ob der Modi „Modus_Private“ verwendet wird, so dass nur die App Zugriff auf die Daten in der preferences.xml hat. Der Modi „Modus_Private“ stellt einen Standardmodus dar. Durch die Nutzung kann nur die aufrufende App oder Apps mit derselben UID auf die .xml zugreifen.[Andc]

Um den Java-Code zu überprüfen kann z.B. auf die Daten aus der automatisierten Analyse des MobSF (vgl. hierzu Kap.5.2.2) zurückgegriffen werden. Sollte keine automatisierte Analyse vorab durchgeführt worden sein, wird zur Analyse entweder ClassyShark oder Android Studio empfohlen. Beide Tools bieten eine Suchfunktion mit der nach einzelnen Stichworten der gesamte Code der App durchsucht werden kann. Die folgende Abbildung 5.28 zeigt ein Beispiel mit Android-Studio.
Um Zugriff auf den Code der Daten zu erhalten, wird die App InsecureBankv2 als Projekt in Android Studio geladen. Android Studio ist in der Testumgebung AndroL4b vorinstalliert und kann über das Anwendungsmenü aufgerufen werden. Im Suchfeld wird nach getSharedPreferences() oder direkt nach den Modi gesucht (vgl. hierzu die Abb.5.28 und Abb.5.29).

Abbildung 5.28: Prüfung des Java-Codes InsecureBankv2-App

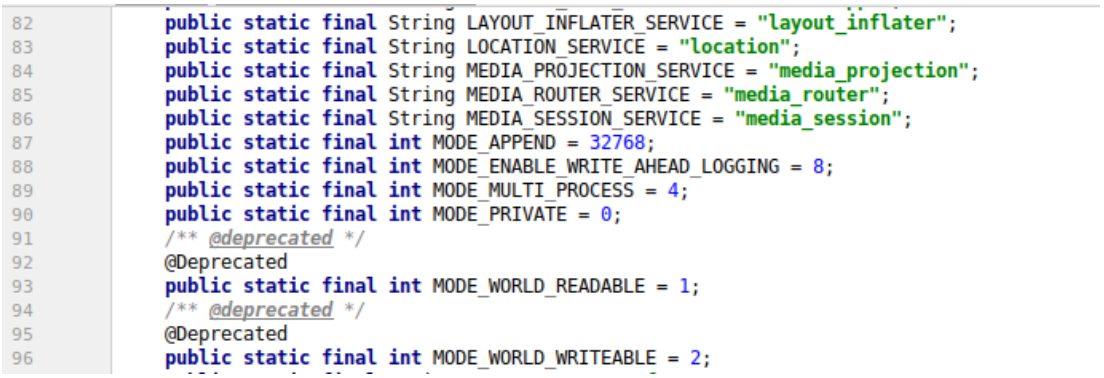


Abbildung 5.29: Prüfung der Modi InsecureBankv2-App

Es wird der erste Eintrag getSharedPreferences(in...content.Context) gewählt (Abb.5.28). Beim Scrollen durch den Code wird erkannt, dass der Content Provider (vgl. hierzu Kap.2.3.5) der App mit den veralteten Modi konfiguriert wurde. Der Content-Provider stellt einen IPC-Mechanismus der App dar. Wenn die Modi MODE_WORLD_Readable oder MODE_WORLD_Writeable verwendet werden, haben alle Dienste und Apps auf dem mobilen Gerät Zugriff auf die Daten. (Da hier eine Testapplikation verwendet wird, werden zu Testzwecken alle Modi veranschaulicht, siehe Abb.5.29)

Nachdem die SharedPreferences auf mögliche Schwachstellen überprüft wurden, werden folgend verwendete SQLite DBs überprüft. SQLite-Datenbanken werden häufig in mobilen Umgebungen verwendet. Zudem wird das SQLite-Framework von Android unterstützt, sodass viele Apps diese zur Datenspeicherung verwenden. In der Regel sind die gespeicherten SQLite-Daten aufgrund der von Android auferlegten Sicherheitsbeschränkungen standardmäßig nicht für andere Dienste und Apps auf dem Gerät

zugänglich.

Genau wie bei den SharedPreferences kann mit einem Blick ins Dateisystem der App ermittelt werden, ob Datenbanken vorhanden sind. Mit dem Befehl **$ ls -al data/data/data/jakhar.aseem.diva/databases/** können in der Kommandozeile, wie in Abb.5.30 zu erkennen, über die ADB die Datenbanken aufgerufen werden.

```
root@generic:/data/data # ls -al jakhar.aseem.diva/databases
-rw-rw---- u0_a53   u0_a53      20480 2017-07-10 18:55 divanotes.db
-rw------- u0_a53   u0_a53       8720 2017-07-10 18:55 divanotes.db-journal
-rw-rw---- u0_a53   u0_a53      16384 2019-09-16 05:50 ids2
-rw------- u0_a53   u0_a53       8720 2019-09-16 05:50 ids2-journal
root@generic:/data/data #
```

Abbildung 5.30: Beispiel 1: Prüfung der Datenbanken

Um die Inhalte der aufgelisteten DBs abrufen zu können, bieten sich zum einen ein schneller Blick mit dem Befehl **$ cat data/data/data/jakhar.aseem.diva/databases/divanotes.db** an. Damit werden alle Datenbankinhalte angezeigt (siehe hierzu Abb.5.31). Eine weitere Möglichkeit besteht darin, mit dem Befehl **$ adb pull** die Datenbank-Datei vom mobilen Gerät auf das Testsystem zu laden und dann im SQLBrowser auszuführen, um Zugriff auf die Inhalte zu erhalten. Diese Option bietet sich besonders bei größeren Datenmengen an. Wie in der Abb.5.31 zu erkennen, enthält die divanotes.db Termineinträge, z.B. „Lunch with CEO“

```
root@generic:/data/data # cat jakhar.aseem.diva/databases/divanotes.db
SQLite format 3 tablesqlite_sequencesqlite_sequenceCREATE TABLE sqlite_sequence(name,seq)z Stablenotesnotes CREATE TABLE
 notes ( id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT NOT NULL, note TEXT NOT NULL)W -tableandroid_metadataandroi
X4 Weekendb333333333333r" 9ExerciseAlternate days running) ExpenseSpent too much on home theater" holiday
her C notes root@generic:/data/data # baby, Order dinner. office10 Meetings. 5 Calls. Lunch with CEO
```

Abbildung 5.31: Beispiel 2: Auslesen der Datenbanken

Des Weiteren zeigt ein Blick in den Java-Code (siehe hierzu Abb.5.32-Auszug Bericht MobSF) das Benutzerdaten, wie User, Password und Creditcardnumber, unverschlüsselt in der Datenbank gespeichert werden. Um den Java-Code mit Android-Studio zu überprüfen, wird nach der Funktion „SQLiteDatabase . openOrCreateDatabase“ gesucht. Falls verschlüsselte Datenbanken verwendet werden, sollte überprüft werden, ob das Kennwort ggf. hardcodiert im Code steht. Dafür sollten auch nochmal die SharedPreferences überprüft werden.

```
public class SQLInjectionActivity
extends AppCompatActivity {
    private SQLiteDatabase mDB;

    /*
     * Enabled aggressive block sorting
     * Enabled unnecessary exception pruning
     * Enabled aggressive exception aggregation
     */
    @Override
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        try {
            this.mDB = this.openOrCreateDatabase("sqli", 0, null);
            this.mDB.execSQL("DROP TABLE IF EXISTS sqliuser;");
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS sqliuser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('admin', 'passwd123', '1234567812345678');");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('diva', 'p@ssword', '1111222233334444');");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('john', 'password123', '5555666677778888');");
        }
        catch (Exception exception) {
            Log.d((String)"Diva-sqli", (String)("Error occurred while creating database for SQLI: " + exception.getMessage()));
        }
        this.setContentView(2130968617);
    }
```

Abbildung 5.32: Beispiel: Prüfung der Datenbanken

Nachdem sie SQLite-Datenbanken auf mögliche Schwachstellen analysiert wurden, wird nun geprüft, welche Daten die App im internen Speicher hält.
Der interne Speicher stellt die Festplatte des mobilen Gerätes dar. Jede App hat ihren reservierten Speicherbereich, auf den nur sie Zugriff haben sollte. Bei Deinstallation der App, werden die Daten im internen Speicher gelöscht. Um zu Ermitteln, wie die Daten im internen Speicher gesichert werden, muss die Konfiguration der Datei-Modi MODE_PRIVATE, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE, im Java-Code geprüft werden. Zur Überprüfung kann Android Studio verwendet werden. Über die Suchfunktion wird nach der Funktion FileInputStream, openFileOutput oder den Datei-Modi gesucht, um die Konfiguration überprüfen zu können.

Abschließend wird der externe Speicher der App überprüft.
Auf den externen Speicher, wie z.B. eine SD-Card, haben grundsätzlich alle Apps Zugriff. Aus diesen Gründen sollten im externen Speicher möglichst keine sensiblen oder vertrauliche Dateien ohne zusätzliche Verschlüsselungsmechanismen gespeichert werden. Der Zugriff auf den externen Speicher wird über Permissions, mit dem Modus WRITE_EXTERNAL_STORAGE, geregelt. Die Abb.5.33 zeigt eine Überprüfung des AndroidManifest.xml mit Classyshark, indem die Permissions der App deklariert werden. Genau wie bei Android Studio, bietet ClassyShark eine Suchfunktionen, um nach bestimmten Strings im Code zu suchen.

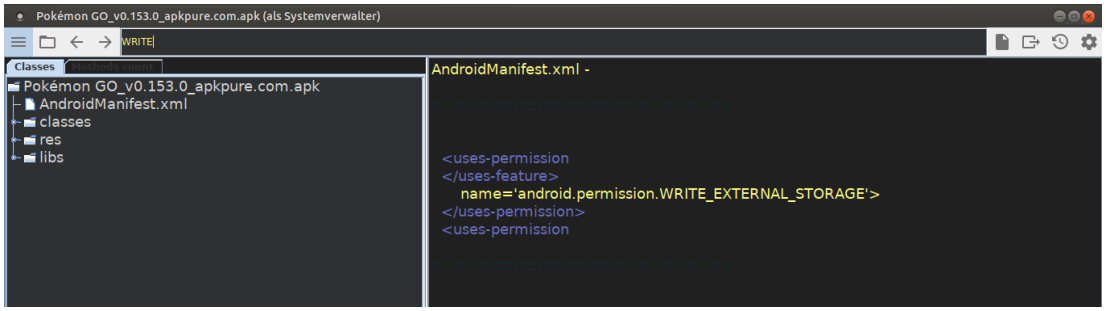


Abbildung 5.33: ClassyShark: Überprüfung Permissions External Storage

2.3 Es werden keine sensiblen Daten in die Logfiles der App geschrieben

Bei dieser Anforderung soll zum einem geprüft werden, welche Protokollierungsfunktionen von der App genutzt werden und inwieweit diese ggf. vertrauliche Daten preisgeben.

Um herauszufinden, ob die App über Protokolldaten sensible Daten offenlegt, bietet sich der Kommandozeilenbefehl logcat an. Mit dem Befehl **adb logcat | grep „$ (adb shell ps | grep com.android.browser | awk '{print $ 2}')“** werden laufende Systemprozesse geloggt.(vgl. Abb.5.34).

```
root@l4b:/home/andro# adb logcat | grep "$(adb shell ps | grep com.android.browser | awk '{print $2}')"
I/ActivityManager(  380): Start proc com.android.browser for activity com.android.browser/.BrowserActivity: pid=1896 uid=1
0014 gids={50014, 3003, 1028, 1015}
E/ActivityThread( 1896): Failed to find provider info for com.google.settings
E/ActivityThread( 1896): Failed to find provider info for com.google.settings
V/WebViewChromium( 1896): Binding Chromium to the main looper Looper (main, tid 1) {add66110}
I/chromium( 1896): [INFO:library_loader_hooks.cc(112)] Chromium logging enabled: level = 0, default verbosity = 0
I/BrowserProcessMain( 1896): Initializing chromium process, renderers=0
D/dalvikvm( 1896): GC_CONCURRENT freed 275K, 12% free 2973K/3376K, paused 29ms+60ms, total 128ms
```

Abbildung 5.34: Dynamische Analyse der Protokolldaten mit Logcat

2.4 Es werden keine sensiblen Daten mit Dritten geteilt - es sei denn dies wurde in der App-Architektur definiert und ist zur Erfüllung des Zwecks der App erforderlich.

Um herauszufinden, ob die App Informationen mit Dritten teilt, besteht ein erster Schritt in der Inspektion der Permissions, die in der AndroidManifest.xml deklariert werden. Hier kann entweder ClassyShark verwendet oder auf den generierten Bericht des MobSF zurückgegriffen werden, um die AndroidManifest.xml einsehen zu können (vgl. Abb.5.35).
Die DIVA-App hat Schreib-und Lesenberechtigungen für den externen Speicher und eine Berechtigung zum Zugriff auf das Internet. Für eine tiefergehende Analyse der Permission kann an dieser Stelle zusätzlich Drozer verwendet werden. Der Drozer-Befehl **run app.package.info -a com.android.insecurebankv2** ermittelt alle Permissions.

Allerdings werden die Permissions, genau wie auch schon in der Manifest.xml zu erkennen war, nicht weiter definiert.[Spr17]

```
dz> run app.package.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
  Application Label: InsecureBankv2
  Process Name: com.android.insecurebankv2
  Version: 1.0
  Data Directory: /data/data/com.android.insecurebankv2
  APK Path: /data/app/com.android.insecurebankv2-1.apk
  UID: 10052
  GID: [3003, 1028, 1015]
  Shared Libraries: null
  Shared User ID: null
  Uses Permissions:
  - android.permission.INTERNET
  - android.permission.WRITE_EXTERNAL_STORAGE
  - android.permission.SEND_SMS
  - android.permission.USE_CREDENTIALS
  - android.permission.GET_ACCOUNTS
  - android.permission.READ_PROFILE
  - android.permission.READ_CONTACTS
  - android.permission.READ_PHONE_STATE
  - android.permission.READ_CALL_LOG
  - android.permission.ACCESS_NETWORK_STATE
  - android.permission.ACCESS_COARSE_LOCATION
  - android.permission.READ_EXTERNAL_STORAGE
  Defines Permissions:
  - None
```

Abbildung 5.35: Drozer: Überprüfung Permissions der InsecureBank-App

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="jakhar.aseem.diva" platformBuildVersionCode="23" platformBuildVersionName="6.0-2166767"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="23" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
```

Abbildung 5.36: MobSF Bericht- Überprüfung der Permissions

Des Weiteren kann mit der dynamische Analyse des MobSF überprüft werden, ob die App Informationen mit Dritten teilt (siehe hierzu Abb.5.37).

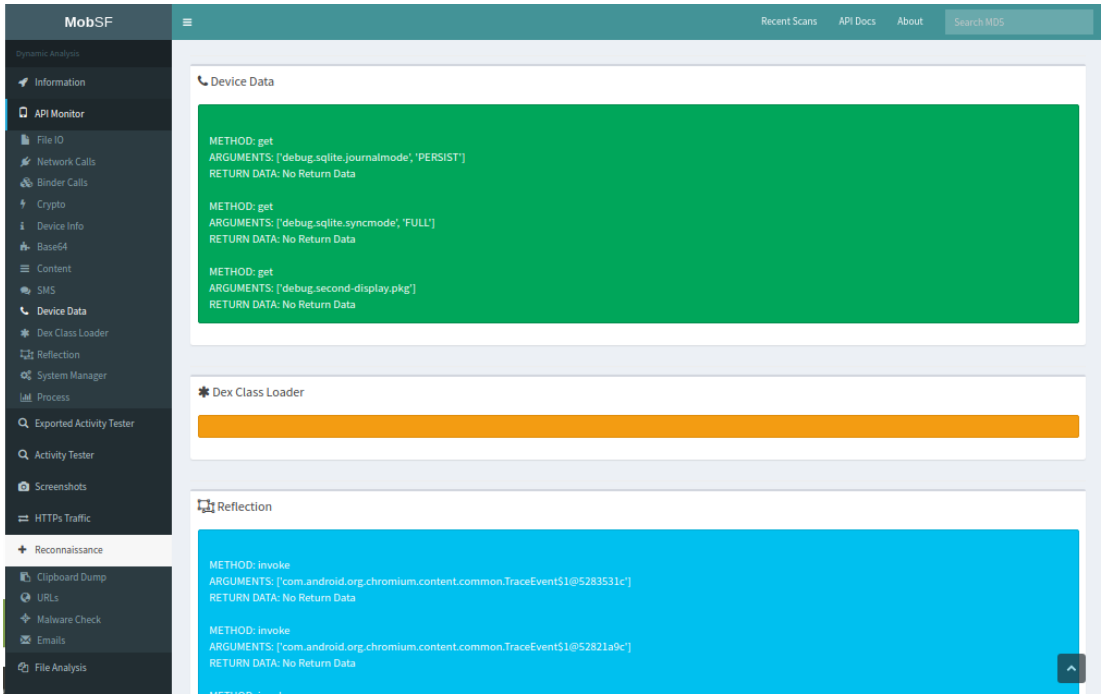


Abbildung 5.37: Analyse des Datenverkehrs der App

Darüber hinaus können ggf. ein Wireshark Mitschnitt oder die Überprüfung des Datenverkehrs mit der BurpSuite, weiteren Informationen liefern.

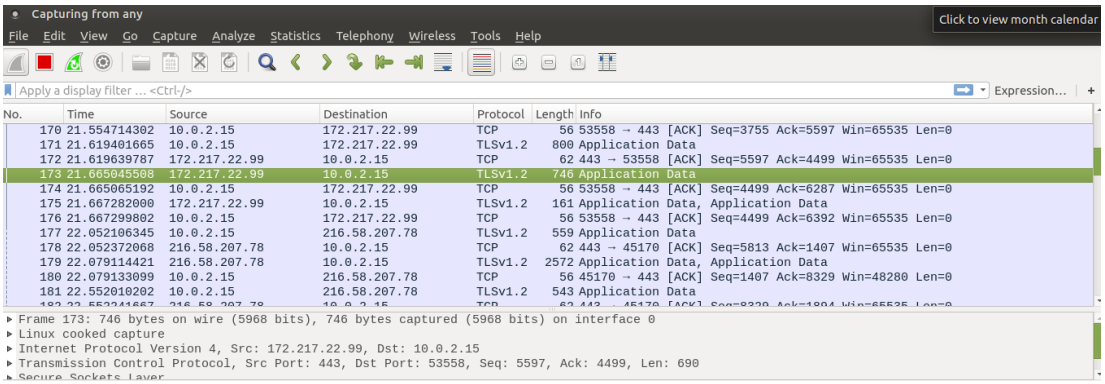


Abbildung 5.38: Mittschnitt Wireshark

2.6 Es werden keine sensiblen Daten über Interprozesskommunikation zur Verfügung gestellt.

Wie zuvor erklärt, ermöglichen Intent-Provider je nach Konfiguration, lesenden- und schreibenden Zugriff von Apps auf die Daten von anderen Apps. Bei Fehlkonfigurationen der IPC-Mechanismen, können ggf. vertrauliche Daten abfließen.
Bei einer statischen Analyse wird die AndroidManifest.mxl nach genutzten Intent-

Provider durchsucht. Die Intent-Provider können an dem Element „provider“erkannt werden (siehe hierz Abb.5.39).

```
        <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin" />
        <activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:exp
orted="true" />
        <activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" andro
id:exported="true" />
        <provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="co
m.android.insecurebankv2.TrackUserContentProvider" />
        <receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
            <intent-filter>
                <action android:name="theBroadcast" />
            </intent-filter>
        </receiver>
```

Abbildung 5.39: Analyse der Content-Provider

Im AndroidManifest der InsecureBank-App ist ein Intent-Provider deklariert. Das Tag „androidandroid:exported“ ist auf „true“ gesetzt, wodurch andere Apps Zugriff auf die Daten der App erhalten. Des Weiteren sollte geprüft werden, ob ein Zugriff auf die Daten vielleicht noch durch bestimmte Permissions geregelt wird. Aber eine weitere Analyse des Manifests ergibt, dass dahingehend keine Permissions definiert wurden.

Im Weiteren wird der gefundene Content-Provider der InsecureBank-App mit dem Tool Drozer, dynamisch auf mögliche Angriffsflächen analysiert. Dafür wird im AndroL4b der Emulator gestartet und über die Konsole mit dem Befehl **$ drozer console connect**, der Drozer.
Drozer ist im AndroL4b schon installiert, wodurch einer schnelle Analyse des Content-Providers der InsecureBank-App durchgeführt werden kann.
Mit **dz> run app.provider.info -a com.android.insecurebankv2** können die Content-Provider der App und weitere Informationen zu diesen ermittelt werden (vgl. hierzu Abb.5.40).

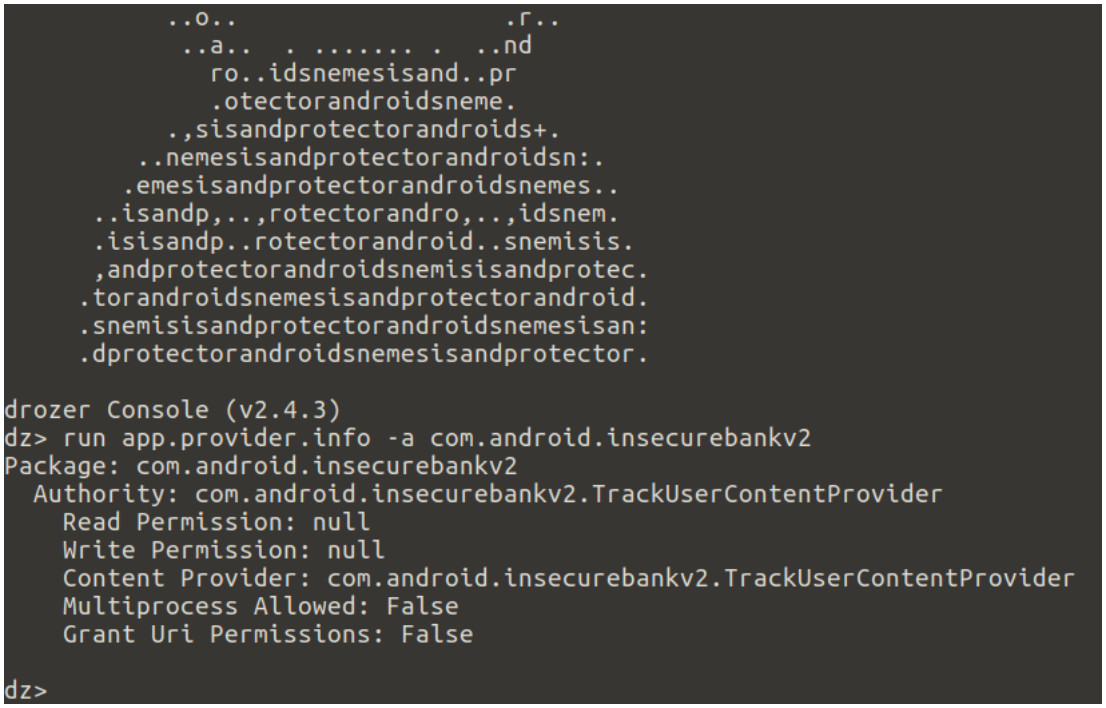


Abbildung 5.40: Drozer Analyse Content-Provider 1

Wie der Abb.5.40 entnommen werden kann, gibt es einen Content-Provider, der nicht durch Permissions geschützt wird (Read Permission: null; Write Permission:null). So kann jede App auf dem mobilen Gerät lesend und schreibend auf die Daten des Content-Providers zugreifen. Die Daten können in einer Datenbank, in Dateien oder sogar über ein Netzwerk vom Content-Provider gespeichert werden. Wenn SQLite-Datenbanken verwendet werden, könnte der Content-Provider für SQL-Injection anfällig sein.

In einem nächsten Schritt soll ermittelt werden, mit welchen URIs der Content-Provider arbeitet. Die URIs dienen der eindeutigen Identifizierung eines Pfads zu einem bestimmten Datensatz in einem der Speicher, die vom Content-Provider verwendet werden. (vgl. Abb.5.41). Mit dem Drozer-Befehl **dz> run app.provider.finduri com.android.insecurebankv2** werden die URIs angezeigt und den Befehl **run app.provider.query content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers** können die Daten ausgelesen werden.[Spr17]

```
dz> run app.provider.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
  Authority: com.android.insecurebankv2.TrackUserContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.android.insecurebankv2.TrackUserContentProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False


dz> run scanner.provider.finduris -a com.android.insecurebankv2
Scanning com.android.insecurebankv2...
Unable to Query  content://com.android.insecurebankv2.TrackUserContentProvider/
Unable to Query  content://com.google.android.gms.games
Unable to Query  content://com.android.insecurebankv2.TrackUserContentProvider
Able to Query
content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
Able to Query
content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers/
Unable to Query  content://com.google.android.gms.games/

Accessible content URIs:
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers/

dz> run app.provider.query
content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
| id | name |
| 1  | Mick |
```

Abbildung 5.41: Drozer Analyse Content-Provider 1

6 Zusammenfassung und Schlussbemerkungen

In dieser Bachelorarbeit sollten neben der Erarbeitung von Strategie zur Durchführung einer Sicherheitsanalyse, Empfehlungen und Anforderungen ermittelt werden, die einen Sicherheitsexperten bei der Planung einer Sicherheitsanalyse unterstützen können. Ferner sollte die Bedeutung von Sicherheitsanalysen für den Schutz von Android-Applikationen vor Malware und Cyberattacken erläutert werden.

Zu Beginn dieser Arbeit wurde ausführlich die Architektur des Android-System und Komponenten von Android-Applikationen beschrieben und erläutert. Sowohl für die Planung und Durchführung der Sicherheitsanalysen als auch für die spätere Ermittlung etwaiger Anforderungen für Sicherheitsanalysen ist ein gutes Verständnis der gesamten Android-Architektur unerlässlich.
Folgend wurden unter Einbezug einiger Beispiele der Versuch einer Klassifizierung der vielzähligen Malwarearten vorgenommen. Allerdings ist dabei aufgefallen, dass sich diese mittlerweile stark vermischen und so eine Klassifizierung der Malware zunehmend erschwert. Ein Beispiel für diese Vermischung, stellte u.a. der Android-Trojan-Dropper dar.
Des Weiteren wurden Angriffsvektoren unter besonderem Fokus auf die von mobilen Apps klassifiziert. Dabei zeigte sich, dass der Übergang zwischen schadhaften Apps und legitimen Applikationen fließend ist, da Bedrohungen nicht nur von extern eingeschleuster Malware ausgehen. Auch legale, über offizielle Kanäle installierbare Apps können, insbesondere im Umgang mit sensitiven, personenbezogenen oder vertraulichen Informationen, ein Risiko darstellen. Häufig verschickt die Malware auf den mobilen Geräten erhobene Daten an Provider, ohne dass die Verarbeitungswege dem Nutzer oder den Sicherheitsexperten offensichtlich sind. Außerdem wurde dargestellt, dass mobile Geräte und ihre Applikationen, wie z.B. das Smartphone für Angreifer und für Malware eine breite Angriffsfläche bieten (vgl. hierzu Abb.3.3 OWASP Top 10 Violation Rates).
Bei einer Analyse von 40.000 Applikationen durch das Unternehmen Nowsecure stellte sich heraus, das 85% der Applikationen, mindestens eine Schwachstelle aus den OWASP Top 10 Risikobereiche aufwiesen. Die OWASP, die sich auf die Bereitstellung von Analysen und Richtlinien zur Cybersicherheit konzentriert, stellt mittlerweile bei vielen Unternehmen und auch Behörden einen Standard dar (vgl. hierzu BSI-Leitfaden zur Entwicklung sicherer Webanwendungen https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Webanwendungen/Webanw_Auftragnehmer.pdf?__blob=publicationFile&v=2).
Mit den erstellten Richtlinien und benutzerdefinierten Sicherheitstools oder dem Testleitfaden für mobile Applikationen, aufbauend auf den Top 10 Mobile Risks, auf den im späteren Verlauf der Arbeit noch Bezug genommen wird, soll Entwicklern und

Sicherheitsexperten dazu verholfen werden, ein zunehmendes Sicherheitsbewusstsein zu entwickeln und zum anderen auch benutzerdefinierte Tools und eine nützliche und vertrauenwürdige Community für diesen Bereich bieten zu können.[Vel16]

Abschließend wurden wichtige Komponenten des Android-Sicherheitsmodells erläutert. Das Sicherheitsmodell muss einer Vielzahl von Bedrohungen standhalten und soll dem Nutzer gleichzeitig ein hohes Maß an Benutzerfreundlichkeit, Erreichbarkeit und Entertainment bieten.[Andh] Um das umzusetzen, wird bei Android OS ein mehrschichtiges Berechtigungskonzept verwendet, das auf dem Linux-Kernel basiert. Apps und Prozesse kommunizieren über die IPCMechanismen, den Content Providern, Intents und Services (vgl. hierzu Kapitel2.3 Aufbau und Bestandteile von Android-Applikationen.) Android isoliert einzelnen Prozesse und Applikationen mit dem Sandboxing und der Sicherheitsfunktionen des Linux-Kernels. Unter anderem erhält jeder Prozess eine eindeutige UID und ein Label. Zudem wird standardmäßig mit dem Least Privilege Prinzip gearbeitet, wodurch Prozesse und Applikationen nur sehr wenige Permissions erhalten und diese auch explizit anfordern müssen, um mit Systemdiensten, Hardwarekomponenten oder anderen Applikationen interagieren zu können. Permissions werden in der Manifestdatei (vgl. hierzu Kapitel 2.3.1 Das Android Manifest) für jede Applikation definiert und bei der Installation erteilt. Des Weiteren verwendet das Android OS die UIDs jeder Applikation, um herauszufinden, welche Permissions erteilt wurden, um sie zur Laufzeit durchzusetzen. Mit SELinux wird die Vergabe von Permissions für die einzelnen Prozesse noch weiter eingeschränkt und die Isolierung der Prozesse verstärkt. Dadurch sollen die Systemdienste noch besser geschützt und die Steuerung der Zugriffe z.B. auf Applikationsdaten granularer gestaltet werden.// Die Signierung der Applikation stellt eine weitere Sicherheitskomponente im Android OS dar. Über das in die apk-Datei enthaltene Zertifikat, wird die Applikation validiert und kann eindeutig einem Entwickler zugeordnet werden. Fehlt diese Signierung oder entspricht nicht den Vorgaben des Google Play Stores, kann diese nicht auf dem Gerät installiert werden. Auf Basis des Zertifikats wird beim Installationsprozess die UID vergeben.
Aufgrund der immer größer werdenden Anzahl an Schwachstellen bietet Android seit 2015 ein umfangreiches Patchmanagement. Allerdings gestaltet sich dieser Prozess, wie zuvor beschrieben wurde, als sehr aufwendig. Es gibt nicht mehr die eine Android-Version. Viele Hersteller von mobilen Geräten, wie Smartphones, haben das Android OS an ihre Bedürfnisse angepasst und z.B. um zusätzliche Komponenten erweitert. Zwar stellen auch die einzelnen Hersteller Patches zur Verfügung, aber oft erst nach einem langen Zeitraum oder es werden Updates mit gefälschten Versionsnummern vorgetäuscht.[Lel]

Mit den hier vorgestellten Sicherheitskomponenten und noch einigen Sicherheitsdiensten, wie u.a. Verifiy App oder Google Play Protect (siehe hierzu „https://source.android.com/security/reports/Google_Android_Enterprise_Security_Whitepaper_2018.pdf“), auf die im Rahmen dieser Arbeit aber nicht weiter ein-

gegangen wird, sollen Angriffsvektoren minimiert und so der Befall durch Malware reduziert werden. Im Allgemeinen bietet das Android-System mit seinen Sicherheitskomponenten, vor allem das Sandboxing, das durch SELinux noch erweitert bzw. verstärkt wird, einen guten Grundschutz. Trotzdem weist es kritische Sicherheitslücken auf und wie in Abb.6.1 zu sehen ist, werden täglich neue Schwachstellen gemeldet.

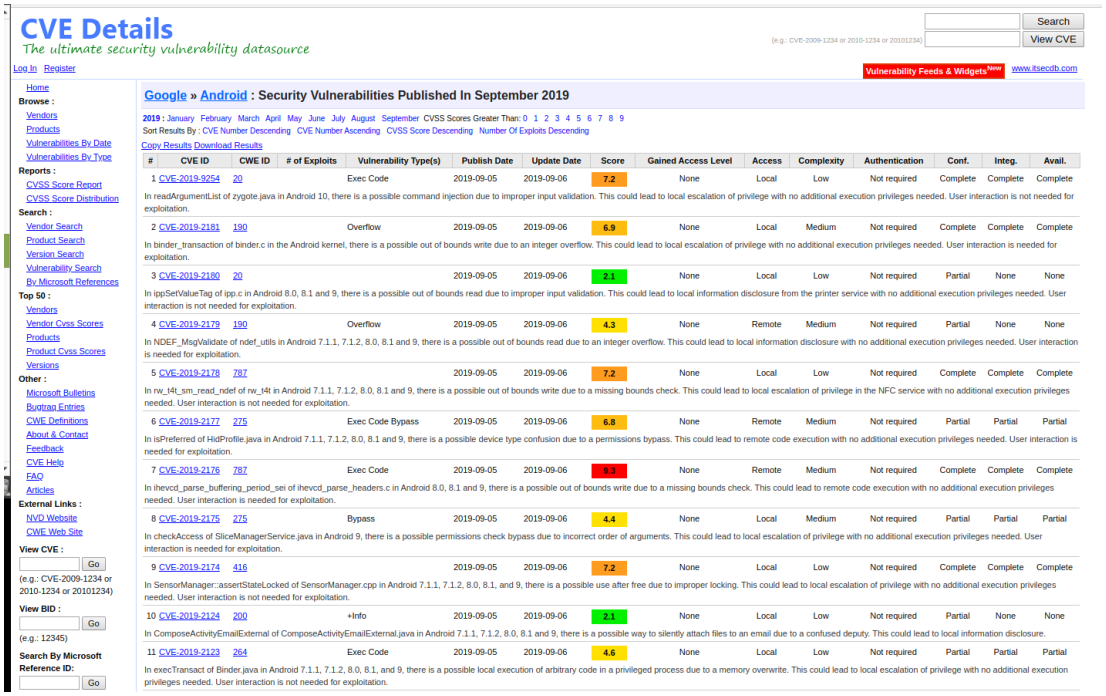


Abbildung 6.1: Gemeldete Schwachstellen Android September 2019 (Stand: 08.September 2019)[CVEc]

Häufige Bedrohungen gegen das Sicherheitsmodell stellen dar:

1) **Das Rooting**: Durch Rooting eines mobilen Gerätes kann das Sicherheitsmodell komplett außer Kraft gesetzt werden. Anwender mit Root-Rechten haben vollständige Kontrolle über die Anwendung und die Vergabe der Permissions. Ebenso können die Anwender ungehindert auf das Dateisystem zugreifen und z.B. sensible oder vertrauliche Daten auslesen.

2) **Gefahren durch App-Stores**: Durch die Möglichkeit für Anwendern Applikationen aus Drittanbieter-Stores zu laden, besteht eine große Gefahr, dass schadhafter Code bzw. Applikationen auf das Gerät gelangen und so die Sicherheitskontrollen des System aushebeln. Der Google-Play-Store wird mit Verify-App und Google-Play-Protect überwacht und stellt eine vorgeschaltete Sicherheitsfunktion, damit keine schadhaften Applikationen auf die mobilen Geräte gelangen.

3) **App-Signierung und Verifizierung**: Der App-Signierung-Prozess birgt eine große Sicherheitslücke, da es keine offizielle Zertifizierungsstelle gibt, die vertrauenswürdige Zertifikate vergibt. Die Entwickler erstellen selbst die Zertifikate mit

denen die Applikation signiert wird (unter Beachtung vorgegebener Regeln von Google) und können diese dann im Google-Play-Store veröffentlichen. Zudem ist der Entwickler dafür verantwortlich, die Schlüssel sicher zu verwahren.

4) **Permission-Vergabe**: Bei der Installationen hat der Anwender nur begrenzte Möglichkeiten Einfluss auf die Vergabe der Permissions zu nehmen. Zudem sind diese für den Anwender nicht hinreichend transparent. Viele Permissions von Standard-Applikationen, wie z.B. E-Mail-, SMS-oder auch die Browser-Applikation haben Zugriff auf das Mikrofon, die Kamera oder die Geo-Lokation. Meist hat der Anwender auch keine Chance eine Applikation zu nutzen, wenn er dieser nicht alle Permissions erteilt.

5) **Android-Fragmentierung**: Mittlerweile bestehen nicht nur eine Vielzahl von Android-Versionen, sondern auch viele mobile Geräte, die mit älteren Android-Versionen betrieben werden und so nicht über alle Android-Schutzkomponenten verfügen. Dadurch besteht ein erhöhtes Risiko der Kompromittierung.

6) **Sonstige Sicherheitslücken**: Weitere Sicherheitslücken können u.a. durch Fehler im Code, die wiederum zu Sicherheitslücken in einzelnen Komponenten der App führen können, entstehen oder auch durch fehlende Patches für einzelne Hardwarekomponenten, die sich auf die Funktion des Sicherheitsmodells auswirken und somit Einfallstore für schadhafte Applikationen bieten.

Im fünften Abschnitt wurden wichtige Grundlagen zur Planung und Umsetzung von Sicherheitsanalysen von Android-Applikationen vorgestellt und erläutert. Dabei konnte gezeigt werden, dass die Planung von Sicherheitsanalysen entsprechend den Vorgaben und Empfehlungen des BSI, um der Forderung zur Informationssicherheit nachkommen zu können, einen umfangreichen und komplexen Prozess darstellt. Die Durchführung einer Bedrohungsmodellierung verlangt von Unternehmen und Behörden nicht nur ein klares Bild von der aktuellen Bedrohungslage und möglichen Risiken für mobile Applikationen, sondern auch ein tiefes Verständnis der gesamten mobilen Infrastruktur, um eindeutig Angriffsvektoren identifizieren und bewerten zu können (siehe hierzu Kapitel 3.2 Klassifizierung von Angriffsvektoren für Android-Smartphones).
Im Rahmen dieser Arbeit wurde die Identifikation der Angriffsvektoren unter Bezug auf die definierten Gefährdungen des BSI und die OWASP Top 10 mobile Risks vorgenommen. In der Regel fließen in diesen Prozess auch noch von den Unternehmen und Behörden ermittelte Bedrohungen, je in Abhängigkeit von der vorhandenen mobilen Infrastruktur und schon vorhandener Sicherheitsmechanismen, ein. Darauf folgend wurden aus den identifizierten Angriffsvektoren, Sicherheitsanforderungen abgeleitet (vgl. hierzu Abb.4.2 Referenztabelle BSI und OWASP Mobile-Security-Checkliste), mit denen die einzelnen Prüffälle-bzw. Kategorien für die Sicherheitsanalyse modelliert wurden (vgl. hierzu Kap.4.2.3 „Checkliste für die Sicherheitsanalyse“). Abschließend wurde anhand

des bestimmten Schutzbedarfs (vgl. hierzu Kap.4.2.2 „Schutzbedarfsfeststellung“) eine Bewertung der Angriffsvektoren vorgenommen, die über die Prüftiefe entscheiden.

Im letzten Abschnitt der Arbeit wurde prototypisch ein in Abschnitt 4 modellierter Prüffall getestet. Dafür wurde zuerst der Aufbau der Testumgebung dargestellt und eine kurze Beschreibung zu Tools gegeben, die bei der Sicherheitsanalyse eingesetzt wurden. Wie schon zu Beginn des Abschnitts angedeutet, gibt es für die Analyse von Android-Applikationen mittlerweile sehr viele und gute Tools. Da es Open-Source-Tools sind, kann es manchmal aufgrund nicht aktueller Module, Funktionen oder Pakete zu Kompatibilitätsproblemen kommen. Oder die Tools haben Fehler im Code aufgrund dessen bestimmte Module nicht einwandfrei funktionieren. Um Abhängigkeitsproblemen, die besonders schnell mit Python-Modulen auftreten können zu Umgehen, wird z.B. die Verwendung von Virtual-Enviroments empfohlen (siehe hierzu auch: venv — Creation of virtual environments-https://docs.python.org/3/library/venv.html). Überdies konnte festgestellt werden, dass viele Tests mit integrierten Kommandozeilenbefehle in Linux oder über die Android-Debug-Bridge durchgeführt werden konnten, wie z.B. eine Überprüfung der Permissions, eine Stringanalyse oder das Auslesen von Dateien.

Im Weiteren wurden zwei Tools, VirusTotal und MobSF, zur Durchführung von automatisierten statischen und dynamischen Analyse vorgestellt. Dabei konnte gezeigt werden, dass beide Tools mit den gelieferten Ergebnisse, wesentlich zur Analyse beitragen können, einen guten Überblick über den Aufbau und Komponenten einer App geben, wodurch ein Sicherheitsexperte beim Einstieg in die Analyse gut unterstützt wird.

Darauf folgend wurden in Kapitel 5.3 Testapplikationen mit unterschiedlichen Schwachstellen auf die in der erstellten Checkliste definierte Anforderung zur Datenspeicherung und Datenschutz untersucht.
Dabei wurde in einem ersten Schritt analysiert, wie die App persönliche oder vertrauliche Daten verarbeitet und in gängigen Datenspeichern, wie z.B. SharedPreferences oder SQLite Datenbanken ablegt. Des Weiteren wurde gezeigt, wie mit dem Kommandozeilen Befehl „logcat“ eine dynamische Analyse der Protokolldaten durchgeführt werden kann, um die geloggten Systemdateien zur Laufzeit auf sensible Daten zu überprüfen.
Im Zuge der Überprüfung des Datenverkehrs der App, z.B. Übermittlung der sensibler Daten an Dritte, wurden auch die Permissions der App geprüft. Die Permission entscheiden im wesentlichen darüber, auf welche Bereiche, Komponenten oder Funktionen eine App Zugriff erhält. Abschließend wurde ermittelt, ob die IPC-Mechanismen auf irgendeine Art und Weise sensible Daten zur Verfügung stellen. Dabei wurden insbesondere die Content-Provider der App analysiert. Die Content-Provider übernehmen in der App die Zugriffsteuerung auf die Daten und verwalten auch die Zugriffe von anderen Apps. Bei falscher Konfiguration der Content-Provider, kann es zur Offenlegung sensibler

Daten kommen.
Der gesamte Analyseprozess wurde umfangreich mit Screenshot dokumentiert. Ebenso wurden die Nutzung der verwendeten Tools kurz erläutert und Hinweise zur deren Installation gegeben.

Die Testung des Prüffalls „Anforderungen an die Datenspeicherung und Datenschutz“ hat gezeigt, dass ein Sicherheitsanalyse sehr zeitaufwendig ist und ein gutes Verständnis nicht nur der gesamten App-Infrastruktur, sondern auch einzelner App Funktionen oder Methoden verlangt. Zudem sollte der Tester über ein Basiswissen in Java und Python verfügen, um den Quellcode analysieren zu können.

Literaturverzeichnis

[ADA17] ADAC e.V.: *Die Evolution der Mobilität*. https://www.zukunftsinstitut.de/fileadmin/user_upload/Publikationen/Auftragsstudien/ADAC_Mobilitaet2040_Zukunftsinstitut.pdf. Version: 2017, Abruf: Mittwoch, 1. Mai 2019

[Anda] Android Developers: *The AndroidManifest.xml-File*. https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/manifest/manifest-intro.html. – [Online; Stand 15. Mai 2019]

[Andb] Android Developers: *Bound services overview*. https://developer.android.com/guide/components/bound-services.html, Abruf: [16. Mai, 20:54 UTC]

[Andc] Android Developers: *Save key-value data*. https://developer.android.com/training/data-storage/shared-preferences, Abruf: [29.September 2019, 11:03 UTC]

[Andd] Android Open Source Project: *Android Architecture*. https://source.android.com/devices/architecture, Abruf: [16. Mai, 20:54 UTC]

[Ande] Android Open Source Project: *Application Signing*. https://source.android.com/security/apksigning. – [Online; Stand 15. Mai 2019]

[Andf] Android Open Source Project: *Google Android Enterprise Security Whitepaper 2018*. https://source.android.com/security/reports/Google_Android_Enterprise_Security_Whitepaper_2018.pdf, Abruf: [28.August 2019, 20:54 UTC]

[Andg] Android Open Source Project: *Life of a Patch*. https://source.android.com/setup/contribute/life-of-a-patch, Abruf: [28.August 2019, 20:54 UTC]

[Andh] Android Open Source Project: *Secure an Android Device*. https://source.android.com/security/index.html#platform-security-architecture, Abruf: [28.August 2019, 11:03 UTC]

[Andi] Android Open Source Project: *Security Enchanced Linux in Android*. https://source.android.com/security/selinux, Abruf: [28.August 2019, 11:03 UTC]

[Andj] Android Source: *HAL-Subsystem*. https://source.android.com/devices/camera/camera3_requests_hal, Abruf: Mittwoch, 1. Mai 2019

[And12] Android User: *Mit Android Intents auf Du und Du*. https://www.android-user.de/mit-android-intents-auf-du-und-du/. Version: 2012. – [Online; abgerufen am 15. Mai 2019]

[And13] Android Wiki: *Cloud-Android Wiki,*. https://www.droidwiki.org/w/index.php?title=Cloud&oldid=6555. Version: 2013. – [Online; abgerufen am 14. Mai 2019].

[And16a] Android Wiki: *Freie Software-Android Wiki,*. https://www.droidwiki.org/w/index.php?title=Freie_Software&oldid=15485. Version: 2016. – [Online; abgerufen am 14. Mai 2019].

[And16b] Android Wiki: *Kernel-Android Wiki,*. https://www.droidwiki.org/w/index.php?title=Kernel&oldid=15877. Version: 2016. – [Online; abgerufen am 14. Mai 2019].

[And18] Android Wiki: *Android Debug Bridge — Android Wiki,*. https://www.droidwiki.org/w/index.php?title=Android_Debug_Bridge&oldid=17559. Version: 2018. – [Online; abgerufen am 30. August 2019]

[And19a] Android Developers: *Distribution dashboard*. https://developer.android.com/about/dashboards/index.html. Version: 2019. – [Online; abgerufen am 14. Mai 2019].

[And19b] Android Developers: *permissions*. https://developer.android.com/guide/topics/manifest/permission-element.html. Version: 2019. – [Online; abgerufen am 15. Mai 2019].

[And19c] Android Developers: *Platform Architecture*. https://developer.android.com/guide/platform. Version: 2019. – [Online; Stand 15. Mai 2019]

[And19d] Android Developers: *Security tips*. https://developer.android.com/training/articles/security-tips. Version: 2019. – [Online; abgerufen am 14. Mai 2019].

[And19e] Android Developers: *Sign your app*. https://developer.android.com/studio/publish/app-signing.html#certificates-keystores. Version: 2019. – [Online; abgerufen am 17. Mai 2019].

[And19f] Android Open Source Project: *Android Security & Privacy 2018 Year In Review*. https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf. Version: 2019. – [Online; Stand 16. Mai 2019]

[And19g] Android Open Source Project: *Legacy HALs*. https://source.android.com/devices/architecture/hal. Version: 2019, Abruf: Mittwoch, 1. Mai 2019

[And19h] Android Wiki: *Android-Android Wiki,*. https://www.droidwiki.org/w/index.php?title=Android&oldid=17871. Version: 2019. – [Online; abgerufen am 15. Mai 2019]

[App18] Appfigures: *Anzahl der verfügbaren Apps in den Top App-Stores im 3. Quartal 2018. Statista.* https://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/. Version: 2018, Abruf: Zugriff am 15. Mai 2019.

[Ayc06] Aycock, John: *Intrusion-Detection- und -Prevention-Systeme: Beobachterposten*. 2006. – 70 S.

[Bac16] Backes, M., Bugiel, S., Derr, E., McDaniel, P., Octeau, D., Weisgerber, S. : „On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis. In: *Proceedings of the 25th USENIX Security Symposium* (2016), 1101–1118. https://trust.cispa.saarland/~bugiel/publications/pdfs/bugiel16-usenix.pdf

[BfD] BfDI: *Das Standard-Datenschutzmodell*. "https://www.bfdi.bund.de/SharedDocs/Publikationen/Entschliessungssammlung/ErgaenzendeDokumente/95DSK_SDM_V_1_1clean_Broschuere.pdf;jsessionid=9C782C8004B3B8B317D60E17233A6AE2.1_cid319?__blob=publicationFile&v=2", Abruf: [20.September 2019, 22:51 UTC]

[BFH+11] Becher, M. ; Freiling, F. C. ; Hoffmann, J. ; Holz, T. ; Uellenbeck, S. ; Wolf, C.: *Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices*. New York : IEEE Computer Society, 2011 "http://www5.rz.rub.de:8032/imperia/md/content/wolf/ieee_mobile.pdf"

[Bir15] Bird, Tim: *Android Kernel Features*. https://elinux.org/index.php?title=Android_Kernel_Features&action=info. Version: 2015. – [Online; Stand 15. Mai 2019]

[Bit18] Bitkom: *Anzahl der Smartphone-Nutzer in Deutschland in den Jahren 2009 bis 2018 (in Millionen)*. https://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/. Version: 2018, Abruf: Mittwoch, 1.Mai 2019

[Bor08] Bornstein, Dan: *Presentation of Dalvik VM Internals*. https://14b1424d-a-62cb3a1a-s-sites.googlegroups.com/site/io/dalvik-vm-i

nternals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf. Version: 2008. – [Online; Stand 18. Mai 2019]

[BP10] Becker, Arno ; Pant, Marcus: *Android 2 - Grundlagen und Programmierung*. 2. Aufl. Köln : Dpunkt-Verlag, 2010. – ISBN 978–3–898–64677–2

[BSI] BSI: *Online-Kurs IT-Grundschutz- Lektion 4: Schutzbedarfsfeststellung*. "https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzSchulung/OnlinekursITGrundschutz2018/onlinekurs_itgrundschutz_node.html", Abruf: [20.September 2019, 22:51 UTC]

[BSI17] BSI-Bundesamt für Sicherheit in der Informationstechnik: *Mindeststandard des BSI für Mobile Device Management*. "https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Mindeststandards/Mindeststandard_Mobile-Device-Management.pdf?__blob=publicationFile&v=7". Version: 2017, Abruf: [19.September 2019, 11:32 UTC]

[BSI18] BSI-Allianz für Cybersicherheit: *Android Konfigurationsempfehlung auf Basis betriebssystemeigener Mittel für eine Nutzung mit erhöhter Sicherheit*. "https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS_109.pdf?__blob=publicationFile&v=9". Version: 2018, Abruf: [28.August 2019, 11:03 UTC]

[BSI19] BSI - Bundesamt für Sicherheit in der Informationstechnik: *IT-Grundschutz-Kompendium*. 2.Edition. Köln : Reguvis-Bundesanzeiger Verlag, 2019. – ISBN 978–3–8462–0906–6

[Bun18] Bundesamt für Sicherheit in der Informationstechnik(BSI) (Hrsg.): *Die Lage der IT-Sicherheit in Deutschland*. Bonn : Bundesamt für Sicherheit in der Informationstechnik(BSI), 2018.

[Cal18] Callaham, John: *Die Geschichte von Android OS: Name, Herkunft und mehr*. https://www.androidauthority.com/history-android-os-name-789433/. Version: 2018. – [Online; abgerufen am 14. Mai 2019].

[Car16] Carter, Jonathan: *Mobile Top 10 2016-Top 10*. https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10. Version: 2016. – [Online; Stand 13. Februar 2017]

[Cha17] Chandrayan, Pramod: *How Your Android Code Compiles To Deliver .APK Package File ?* https://codeburst.io/how-your-android-code-compiles-to-deliver-apk-package-file-9f180129c9bf. Version: 2017, Abruf: [16. Mai, 20:54 UTC]

[Chr18] Christian, R.: *Mobile Risks: M3 – Insecure communication*. https://securitygrind.com/mobile-risks-m3-insecure-communication/. Version: 2018. – [Online; Stand 27.Juni 2018]

[cod17] CODIFIEDSECURITY.COM: *OWASP Mobile Top 10 2016: M1 Improper Platform Usage*. https://codifiedsecurity.com/owasp-mobile-top-10-2016-m1-improper-platform-usage/. Version: 2017. – [Online; Stand 1. September 2019]

[CVEa] CVE DETAILS: *Top 50 Products By Total Number Of Distinct Vulnerabilities in 2017*. https://www.cvedetails.com/top-50-products.php?year=2017. – [Online; Stand 2. Juni 2019]

[CVEb] CVE DETAILS: *Top 50 Products By Total Number Of Distinct Vulnerabilities in 2018*. https://www.cvedetails.com/top-50-products.php?year=2018. – [Online; Stand 2. Juni 2019]

[CVEc] CVE DETAILS: *Vulnerability Trends Over Time*. https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224. – [Online; Stand 2. Juni 2019]

[CVEd] CVE.MITRE.ORG: *Frequently Asked Questions-What is CVE??* http://cve.mitre.org/about/faqs.html#what_is_cve_entry. – [Online; abgerufen am 02. Juni 2019].

[DA16] DALZIEL, Henry ; ABRAHAM, Ajin: *Automated Security Analysis of Android and iOS Applications with Mobile Security Framework*. 2. Aufl. Burlington, MA : Syngress, 2016. – ISBN 978–0–12–804718–7

[DDSW11] DAVI, Lucas ; DMITRIENKO, Alexandra ; SADEGHI, Ahmad-Reza ; WINANDY, Marcel: Privilege Escalation Attacks on Android. In: BURMESTER, Mike (Hrsg.) ; TSUDIK, Gene (Hrsg.) ; MAGLIVERAS, Spyros (Hrsg.) ; ILIĆ, Ivana (Hrsg.): *Information Security*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. – ISBN 978–3–642–18178–8, S. 346–360

[doc19] DOCS, MDN web: *Touch events*. https://developer.mozilla.org/en-US/docs/Web/API/Touch_events. Version: 2019. – [Online; Stand 16. Mai 2019]

[Eik16] EIKENBERG, Ronald: *Millionen Android-Smartphones mit Rootkit ab Werk*. https://www.heise.de/security/meldung/Millionen-Android-Smartphones-mit-Rootkit-ab-Werk-3493715.html. Version: 2016. – [Online; Stand 30. August 2019]

[Eil18] EILERS, Carsten: *You've been hacked! - Alles über Exploits gegen Webanwendungen. So schützen Sie sich vor Web-Hacking*. 1. Aufl. Bonn : Rheinwerk Verlag GmbH, 2018. – ISBN 978–3–836–24460–2

[Ele14] ELENKOV, Nikolay: *Android Security Internals - An In-Depth Guide to Android's Security Architecture*. 1.Auflage. München : No Starch Press, 2014. – ISBN 978–1–593–27581–5

[Eur15] EUROSTAT: *Anteil der Unternehmen in ausgewählten Ländern in Europa, die ihren Beschäftigten tragbare Geräte mit mobiler Internetverbindung für geschäftliche Zwecke zur Verfügung stellen, im Jahr 2015*. https://de.statista.com/statistik/daten/studie/417534/umfrage/ausstattung-der-mitarbeiter-mit-mobilen-endgeraeten-in-unternehmen-nach-laendern/. Version: 2015, Abruf: Mittwoch, 1.Mai 2019

[f-s19] F-SECURE.COM: *Backdoor:Android/Hummingbad*. https://www.f-secure.com/v-descs/backdoor_android_hummingbad.shtml. Version: 2019. – [Online; abgerufen am 30. August 2019]

[Fra14] FRANZ, Klaus: *Handbuch zum Testen von Web- und Mobile-Apps - Testverfahren, Werkzeuge, Praxistipps*. 2. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2014. – ISBN 978–3–662–44028–5

[G D18] G DATA: *Cyberangriffe auf Android-Geräte nehmen stark zu*. https://www.gdata.de/blog/2018/11/31254-cyberangriffe-auf-android-gerate-nehmen-stark-zu. Version: 2018, Abruf: Sonntag, 12. Mai 2019.

[G D19a] G DATA: *Kritische Sicherheitslücke: Erster Android-Wurm entdeckt*. https://www.gdata.de/news/2018/06/30834-kritische-sicherheitslucke-erster-android-wurm-entdeckt. Version: 2019, Abruf: Donnerstag, 29. August 2019.

[G D19b] G DATA: *Mobile Malware Report: Keine Entspannung bei Android-Malware*. https://www.gdata.de/news/2019/07/35212-mobile-malware-report-keine-entspannung-bei-android-malware. Version: 2019, Abruf: Donnerstag, 29. August 2019.

[Gar11] GARGENTA, Marko: *Einführung in die Android-Entwicklung*. 1.Auflage. Köln : O'Reilly Germany, 2011. – ISBN 978–3–868–99115–4

[Gar18] GARTNER: *Marktanteile der führenden Betriebssysteme am Absatz von Smartphonesweltweit vom 1. Quartal 2009 bis zum 4. Quartal 2018. Statista*. https://de.statista.com/statistik/daten/studie/73662/umfrage/marktanteil-der-smartphone-betriebssysteme-nach-quartalen/. Version: 2018, Abruf: Freitag, 3. Mai 2019

[Gen] GENYMOTION: *Genymotion for Fun*. "https://www.genymotion.com/fun-zone/", Abruf: [27.September 2019, 16:51 UTC]

[GH17] GRUNWALD, L. ; HERZ, A.: Intrusion-Detection- und -Prevention-Systeme: Beobachterposten. In: *IX* (07/2017), S. 70

[GN14] GARGENTA, Marko ; NAKAMURA, Masumi: *Learning Android*. 2. Auflage. Sebastopol : Ö'Reilly Media, Inc.", 2014. – ISBN 978–1–449–30724–0

[Hei18] Heise Security: *Malware-Angriffe auf Smartphones-die unterschätzte Gefahr*. https://www.heise.de/security/solutions/telekom/malware-angriffe-auf-smartphones-die-unterschaetzte-gefahr/?source=bwt. Version: 2018. – [Online; Stand 16. Mai 2019]

[Hub03] Huber, Edith: *Cyber Crime* -. 1. Aufl. 2019. Berlin Heidelberg New York : Springer-Verlag, 2003. – ISBN 978–3–658–26150–4

[HWA+17] Halton, Wolf ; Weaver, Bo ; Ansari, Juned A. ; Kotipalli, Srinivasa R. ; Imran, Mohammed A.: *Penetration Testing: A Survival Guide*. 1. Aufl. Birmingham : Packt Publishing Ltd, 2017. – ISBN 978–1–787–28988–8

[Inf19] InfoSec: *Hacken von Android-Geräten mithilfe der StageFright-Sicherheitsanfälligkeit*. https://resources.infosecinstitute.com/hack-android-devices-using-stagefright-vulnerability/#gref. Version: 2019. – [Online; Stand 31. August 2019]

[Kan18] Kantar: *Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in Deutschland im 1. Quartal der Jahre 2018 und 2019. Statista*. https://de.statista.com/statistik/daten/studie/198435/umfrage/marktanteile-der-smartphone-betriebssysteme-am-absatz-in-deutschland/. Version: 2018, Abruf: Freitag, 3. Mai 2019.

[KC13] Krajci, Iggy ; Cummings, Darren: *History and Evolution of the Android OS*. Berkeley, CA : Apress, 2013. – 1–8 S. https://doi.org/10.1007/978-1-4302-6131-5_1. – ISBN 978–1–4302–6131–5

[KM16] Knoll, Matthias ; Meinhardt, Stefan: *Mobile Computing - Grundlagen – Prozesse und Plattformen – Branchen und Anwendungsszenarien*. 1. Aufl. 2016. Berlin Heidelberg New York : Springer-Verlag, 2016. – ISBN 978–3–658–12029–0

[Kof18] Kofler, Michael et al.: *Hacking & Security - Das umfassende Handbuch*. 1. Aufl. Bonn : Rheinwerk Verlag, 2018. – ISBN 978–3–836–24548–7

[Lel] Lell, Jacob and Nohl, Carsten: *Uncovering the Android Patch Gap Through Binary-Only Patch Level Analysis*. https://srlabs.de/wp-content/uploads/2018/04/SRLabs-Mind_the_gap-Android_Patch_Gap-HITB_2018-1.pdf, Abruf: [28.August 2019, 11:03 UTC]

[Lev14] Levin, Jonathan: *Android Internals - Volume I - A Confectioner's Cookbook*. 3. Aufl. Cambridge, MA, USA : Jonathan Levin, 2014 http://newandroidbook.com/AIvI-M-RL1.pdf. – ISBN 978–0–9–91–05–5

[LH05] Lipner, Steve ; Howard, Michael: *Entwicklungszyklus für sichere Software*. https://docs.microsoft.com/de-de/previous-versions/technical-content/ms995349(v=msdn.10). Version: 2005. – [Online; Stand 15. Mai 2019]

[LS17] Luber, Stefan ; Schmitz, Peter: *Was ist ein Exploit?* https://www.security-insider.de/was-ist-ein-exploit-a-618629/. Version: 2017. – [Online; Stand 31. August 2019]

[LSLS17] Labudde, Dirk ; Spranger, Michael ; Labudde, Dirk ; Spranger, Michael: *Forensik in der digitalen Welt - Moderne Methoden der forensischen Fallarbeit in der digitalen und digitalisierten realen Welt*. 1. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2017. – ISBN 978–3–662–53801–2

[Mac16] Macke, Simeon: *Android Kernel Tuning – Teil 1: Die Grundlagen erklärt, vier Kernel im Detail*. https://www.overleaf.com/project/5cbb25a003073944b64b6851. Version: 2016. – [Online; abgerufen am 15. Mai 2019]

[Man10] Mandl, Peter: *Grundkurs Betriebssysteme - Architekturen, Betriebsmittelverwaltung, Synchronisation, Prozesskommunikation*. 2. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2010. – ISBN 978–3–834–89384–0

[Mue19] Mueller, Bernhard et al.: *Mobile Security Testing Guide*. 1.1.2. OWASP Foundation, 2019 978-0-359-47489-9https://github.com/OWASP/owasp-mstg/. – ISBN 978–0–359–47489–9

[Nar16] Narmatha, M. and KrishaKuma, S.: „Study on Android Operating System And Its Versions. In: *International Journal of Scientific Engineering and Applied Science (IJSEAS)* 02 (2016), Febr, Nr. 2, 439–444. http://ijseas.com/volume2/v2i2/ijseas20160254.pdf. – ISSN 2395–3470

[Ope07] Open Handset Alliance: *Was ist die Open Handset Alliance?* https://www.openhandsetalliance.com/oha_faq.html. Version: 2007. – [Online; abgerufen am 14. Mai 2019].

[OWA] OWASP: *Data Storage on Android*. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md, Abruf: [29.September 2019, 11:03 UTC]

[Ris15] Rishabh: *An Overview of Android Binder Framework*. http://codetheory.in/an-overview-of-android-binder-framework/. Version: 2015, Abruf: [16. Mai, 20:54 UTC]

[Rix17] Rixecker, Kim: *Aggressive Mining-Malware zerstört den Akku von Smartphones*. https://t3n.de/news/aggressive-mining-malware-889723/. Version: 2017. – [Online; Stand 31. August 2019]

[Roe] Roeske, Thomas: *Mit Schutzklassenkonzepten zur DSGVO-Compliance.* "https://www.datenschutzbeauftragter-info.de/mit-schutzklassenkonzepten-zur-dsgvo-compliance/", Abruf: [20.September 2019, 22:51 UTC]

[RSM15] Riddle, Bryan ; Schwarz-Manalo, Sydney: *Java und Dalvik Bytecode - Ein Vergleich*, Fachhochschule Köln, Fakultät für Informatik und Ingenieurwissenschaften, Diss., Jul 2015

[Sch13] Schmidt, Harald: *OpenCV Basiswissen.* 1. Auflage. Open Source Press, Ebook, 2013. – ISBN 978–3–95539–028–0

[Sch15] Schiefer, Gunther: *Sicherer mobiler Zugriff auf Unternehmensdaten.* 1. Aufl. Norderstedt : BoD – Books on Demand, 2015. – ISBN 978–3–739–25572–9

[Sch18] Schurr, Amy: *A decade in, how safe are your iOS and Android apps?* https://www.nowsecure.com/blog/2018/07/11/a-decade-in-how-safe-are-your-ios-and-android-apps/. Version: 2018. – [Online; Stand 1. September 2019]

[Sch19a] Schirrmacher, Dennis: *Trojaner-App CamScanner auf mehr als 100 Millionen Android-Geräten installiert.* https://www.heise.de/security/meldung/Trojaner-App-CamScanner-auf-mehr-als-100-Millionen-Android-Geraeten-installiert-4508174.html. Version: 2019. – [Online; Stand 30. August 2019]

[Sch19b] Schleier, Sven and Mueller, Bernard and Willemsen, Jeroen and owasp: *OWASP Mobile Security Testing Guide.* 1.1.3 Release. OWASP, 2019 "https://leanpub.com/mobile-security-testing-guide"

[Sch19c] Schmerer, Kai: *Google Play Store: Bösartige Android-App mit mehr als 100 Millionen Downloads.* https://www.zdnet.de/88367713/google-play-store-boesartige-android-app-mit-mehr-als-100-millionen-downloads/. Version: 2019. – [Online; Stand 28.August 2019, 12:04 Uhr]

[SFSP10] Schwab, Fabian ; Findeisen, Alexander ; Sakal, Peter ; Pohl, Hartmut: Bedrohungsmodellierung (Threat Modeling) in der Softwareentwicklung. In: Freiling, Felix C. (Hrsg.): *Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit.* Bonn : Gesellschaft für Informatik e.V., 2010, S. 253–264

[Spr17] Spreitzenbarth, Michael: *Mobile Hacking - Ein kompakter Einstieg ins Penetration Testing mobiler Applikationen – iOS, Android und Windows Mobile.* 1.Aufl. Heidelberg : dpunkt.verlag, 2017. – ISBN 978–3–960–88125–4

[Spr18] Springer Fachmedien Wiesbaden: „Machen wir uns nichts vor, auf unserem Handy ist unser Leben drauf“. In: *Wirtschaftsinformatik & Management* 10 (2018), Aug, Nr. 4, 100–103. http://dx.doi.org/10.1007/s35764-018-0082-1. – DOI 10.1007/s35764–018–0082–1. – ISSN 1867–5913

[Sta17] Statistisches Bundesamt: *Einwohnerzahl-Anzahl der Einwohner von Deutschland von 1990 bis 2017 (in Millionen)*. https://de.statista.com/statistik/daten/studie/2861/umfrage/entwicklung-der-gesamtbevoelkerung-deutschlands/. Version: 2017, Abruf: Mittwoch, 1.Mai 2019

[Ste19a] Stefanko, Lukas: *2FA: Phishing-Apps umgehen Googles Berechtigungsrichtlinien*. https://www.welivesecurity.com/deutsch/2019/06/17/android-malware-umgeht-2fa-bypass-technik/. Version: 2019. – [Online; Stand 31. August 2019]

[Ste19b] Stefanko, Lukas: *Android-Ransomware ist zurück*. https://www.welivesecurity.com/deutsch/2019/07/30/android-ransomware-filecoder-c/. Version: 2019. – [Online; Stand 31. August 2019]

[Tae07] Taeumel, Marcel: *Threat Modeling- Was kann schon schief gehen?* "https://hpi.de/fileadmin/_migrated/content_uploads/paper2.pdf". Version: 2007, Abruf: [19.September 2019, 11:32 UTC]

[The18] The Linux Kernel Organization: *Active kernel releases*. https://www.kernel.org/category/releases.html. Version: 2018. – [Online; abgerufen am 15. Mai 2019]

[Tok15] Tokarski, Andrzej: *Infografik: Das ist die komplette Geschichte von Android undTablets*. https://www.tabletblog.de/android-geschichte/. Version: 2015, Abruf: Sonntag, 12. Mai 2019.

[Tre18] Tremmel, Moritz: *Spyware liest Whatsapp-Konversationen aus*. https://www.golem.de/news/android-spyware-liest-whatsapp-konversationen-aus-1809-136856.html. Version: 2018. – [Online; Stand 30. August 2019]

[Vel16] Velu, Vijay K.: *Mobile Application Penetration Testing* -. 1. Aufl. Birmingham : Packt Publishing Ltd, 2016. – ISBN 978–1–785–88869–4

[Vira] VirusTotal: *About us*. "https://support.virustotal.com/hc/en-us/categories/360000160117-About-us", Abruf: [27.September 2019, 16:51 UTC]

[Virb] VirusTotal: *Documentation*. "https://support.virustotal.com/hc/en-us/categories/360000162878-Documentation", Abruf: [27.September 2019, 16:51 UTC]

[VLP11] Verclas, Stephan ; Linnhoff-Popien, Claudia: *Smart Mobile Apps - Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. 2012. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2011. – ISBN 978–3–642–22259–7

[Vol17] Vollmer, Guy: *Mobile App Engineering - Eine systematische Einführung – von den Requirements zum Go Live*. 1. Auflage. Heidelberg : dpunkt.verlag, 2017. – ISBN 978–3–960–88212–1

[Wes19] Westenthanner, Marianne: *Millionen Nutzer betroffen: Diese bösartigen Apps sollten Sie sofort löschen*. https://www.focus.de/digital/internet/schon-millionen-nutzer-betroffen-diese-boesartigen-android-apps-sollten-sie-schnell-von-ihrem-smartphone-loeschen_id_11049705.html. Version: 2019. – [Online; Stand 31. August 2019]

[Wik16] Wiki, Android: *Systemapps — Android Wiki,*. https://www.droidwiki.org/w/index.php?title=Systemapps&oldid=14129. Version: 2016. – [Online; abgerufen am 16. Mai 2019]

[Wik17] Wikipedia: *Klickbetrug — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Klickbetrug&oldid=171442854. Version: 2017. – [Online; Stand 30. August 2019]

[Wik18a] Wiki, Android: *App — Android Wiki,*. https://www.droidwiki.org/w/index.php?title=App&oldid=17395. Version: 2018. – [Online; abgerufen am 16. Mai 2019]

[Wik18b] Wikipedia: *Framework-Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Framework&oldid=179886028. Version: 2018. – [Online; Stand 14. Mai 2019].

[Wik19a] Wikipedia: *Adware — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Adware&oldid=189930544. Version: 2019. – [Online; Stand 31. August 2019]

[Wik19b] Wikipedia: *Botnet — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Botnet&oldid=188629252. Version: 2019. – [Online; Stand 31. August 2019]

[Wik19c] Wikipedia: *Google LLC-Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Google_LLC&oldid=188424702. Version: 2019. – [Online; Stand 15. Mai 2019].

[Wik19d] Wikipedia: *Mirai (Malware) — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Mirai_(Malware)&oldid=188230785. Version: 2019. – [Online; Stand 31. August 2019]

[Wik19e] Wikipedia: *Pharming (Internet) — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Pharming_(Internet)&oldid=189465127. Version: 2019. – [Online; Stand 1. September 2019]

[Wik19f] Wikipedia: *Ransomware — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Ransomware&oldid=191806402. Version: 2019. – [Online; Stand 1. September 2019]

[Wik19g] Wikipedia: *Spyware — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Spyware&oldid=187077895. Version: 2019. – [Online; Stand 31. August 2019]

[Wol06] Wolf, Jürgen: *Linux-UNIX-Programmierung - das umfassende Handbuch ; [Einstieg, Praxisbeispiele, Referenz ; System- und Netzwerkprogrammierung ; X Window, GTK+, SDL, Werkzeuge, Sicherheit ; inkl. Openbooks zu C, UNIX und Debian]*. 2. Aufl. Bonn : Galileo Press, 2006. – ISBN 978–3–898–42749–4

[Zha18] Zhauniarovich, Yury: *Learn Android Security Stack*. 1. Auflage. New York : Apress, 2018. – ISBN 978–1–484–21681–1

Abbildungsverzeichnis

1.1 Anzahl der verfügbaren Apps in den Top App-Stores 2018[App18] . . . 2
1.2 Marktanteil mobiler Betriebssysteme in Deutschland[Kan18] 3
1.3 Marktanteil mobiler Betriebssysteme Weltweit[Gar18] 3
1.4 Neue Android-Schaddateien pro Jahr[G D18] 4

2.1 Geschichte von Android[Tok15] 7
2.2 The Android software stack[And19a] 9
2.3 The Android software stack[And19c] 10
2.4 PID und die zugeordnete Anwendung[Ele14] 12
2.5 Aufbau der task-struct[Man10] 12
2.6 Ausschnitt-HAL Libaries Galaxy S8 13
2.7 Die Dalvik-VM im Vergleich zur Java-VM[Gar11] 14
2.8 Aufbau Android-Applikations-Framework[Andd] 15
2.9 Der Android-Build-Prozess[Cha17] 17
2.10 Inhalt einer Android-Applikation 18
2.11 AndroidManifest.xml 20
2.12 Shared-Preference-xml 23

3.1 Weltweite Verbreitung von mobilen Malware-Typen im Jahr 2018 25
3.2 OWASP Top 10 Mobile Risks 35
3.3 OWASP Top 10 Violation Rates[Sch18] 38
3.4 Benutzer-ID 39
3.5 Zuordnung der Applikation zur Domäne 40
3.6 Benutzer-ID 40
3.7 list permissions 42
3.8 list permissions description 42
3.9 Ausschnitt-CERT.RSA Wikipedia-App 43
3.10 Ausschnitt-Signierung APK 44
3.11 APK-Signaturüberprüfungsprozess[Ande] 45
3.12 Ausschnitt-Liste CVE 2019 (Stand: 30. August 2019[CVEc] 47
3.13 Trend Android-Vulnerabilities (Stand: 30. August 2019[CVEc] 47

4.1 Ablauf einer Bedrohungsmodellierung 53
4.2 BSI Referenztabelle 55
4.3 BSI Anforderungen zum Schutz von mobilen Apps 56
4.4 Ermittlung des Schutzbedarfs 58
4.5 Checkliste für die Sicherheitsanalyse mobiler Apps 59
4.6 Ablaufplan für die Sicherheitsanalyse einer mobilen App 61
4.7 Tabellarische Beschreibung des Ablaufs der Sicherheitsanalyse 62

5.1 Startseite VirusTotal 70
5.2 Ausschnitt: Detection InsecureBankv2-App 71
5.3 Ausschnitt: Details InsecureBankv2-App 71
5.4 Ausschnitt: Relations InsecureBankv2-App 72
5.5 Ausschnitt: Behavior InsecureBankv2-App 73
5.6 Bildung Hash-Summe 73

5.7 VirusTotal Hash-Suche . . . 74
5.8 Startseite MobSF . . . 75
5.9 Ablauf statische Analyse MobSF InsecureBankv2-App . . . 76
5.10 Ablauf statische Analyse MobSF InsecureBankv2-App II . . . 76
5.11 Weboberfläche MobSF . . . 77
5.12 Bericht App-Informationen InsecureBankv2-App . . . 77
5.13 Bericht App-Scan Options InsecureBankv2-App . . . 78
5.14 Bericht App-Signer Certificate InsecureBankv2-App . . . 78
5.15 Bericht App-Signer Certificate InsecureBankv2-App . . . 79
5.16 Bericht Manifest Analysis InsecureBankv2-App . . . 80
5.17 Bericht Code Analysis InsecureBankv2-App . . . 81
5.18 Bericht Reconnaissance InsecureBankv2-App . . . 82
5.19 Bericht Components InsecureBankv2-App . . . 82
5.20 Dynamic Analysis InsecureBankv2-App . . . 83
5.21 Dynamic Analysis Exported Activity Tester . . . 84
5.22 Dynamic Analysis Logcat Stream . . . 84
5.23 Dynamic Analysis Bericht . . . 85
5.24 Übersicht Filesystem Emulator AndroL4b . . . 86
5.25 Übersicht Partitionen Emulator AndroL4b . . . 87
5.26 Beispiel Zugriff auf die Shared-Preferences . . . 87
5.27 Prüfung der Shared-Preferences der DIVA-App . . . 88
5.28 Prüfung des Java-Codes InsecureBankv2-App . . . 89
5.29 Prüfung der Modi InsecureBankv2-App . . . 89
5.30 Beispiel 1: Prüfung der Datenbanken . . . 90
5.31 Beispiel 2: Auslesen der Datenbanken . . . 90
5.32 Beispiel: Prüfung der Datenbanken . . . 91
5.33 ClassyShark: Überprüfung Permissions External Storage . . . 92
5.34 Dynamische Analyse der Protokolldaten mit Logcat . . . 92
5.35 Drozer: Überprüfung Permissions der InsecureBank-App . . . 93
5.36 MobSF Bericht- Überprüfung der Permissions . . . 93
5.37 Analyse des Datenverkehrs der App . . . 94
5.38 Mittschnitt Wireshark . . . 94
5.39 Analyse der Content-Provider . . . 95
5.40 Drozer Analyse Content-Provider 1 . . . 96
5.41 Drozer Analyse Content-Provider 1 . . . 97

6.1 Gemeldete Schwachstellen Android September 2019 (Stand: 08.September 2019)[CVEc] . . . 101

A.1 BSI Referenztabelle . . . xxiii
A.2 BSI Anforderungen zum Schutz von mobilen Apps . . . xxiv
A.3 Checkliste für die Sicherheitsanalyse mobiler Apps . . . xxv
A.4 Ablaufplan für die Sicherheitsanalyse einer mobilen App . . . xxvi

Listings

Abkürzungsverzeichnis

AAC Advanced Audio Coding.

AIDL Android Interface Definition Language.

apk Android Package Kit.

ART Android-Runtime.

Ashmen Android Shared Memory.

BDSG Bundesdatenschutzgesetz.

Blob Binary Large Object.

bzw. beziehungsweise.

DSGVO Datenschutzgrundverordnung.

EMM Enterprise Mobile Management.

GID Group Identifier.

HAL Hardwareabstraktionsschicht.

HTML Hypertext Markup Language.

IMEI International Mobile Equipment Identity.

IPC Interprozesskommunikation.

JPEG Joint Photographic Experts Group.

JVM Java-Virtual-Machine.

LAN Local Area Network.

MAC Mandatory Access Control.

MDM Mobile Device Management.

MITM Man-In-The-Middle.

MobSF Mobile-Security-Framework.

MPEG Moving Picture Experts Group.

OS Operating System.

OWASP Open Web Application Security Project.

PID Process Identifier.

RPC Remote Procedure Call.

SELinux Security Enhanced Linux.

SSDLC Secure Software Development Life Cycle.

StGB Strafgesetzbuch.

UID User Identifier.

VM Virtual-Machine.

VPN Virtual Private Network.

WLAN Wireless Local Area Network.

A Weitere Bilder und Tabellen

Schutzbedarf	BSI Referenztabelle APP.1.4 Mobile Apps	Gefährdungen	G0.9 Ausfall oder Störung von Kommunikationsnetzen	G 0.14 Ausspähen von Informationen (Spionage)	Datenträgerverlust von Geräten, Datenträgern oder Dokumenten	G0.18 Fehlplanung oder fehlende Anpassung	G0.19 Offenlegung schützenswerter Informationen	G0.20 Informationen oder Produkte aus unzuverlässiger Quelle	G0.21/0.23 Manipulation/ unbefugtes Eindringen von	G0.25/0.26 Ausfall/Störung von Geräten oder Systemen	G0.28 Software-Schwachstellen oder -Fehler	G0.29/0.32 Verstoß gegen Gesetze Mißbrauch von Berechtigungen Oder der Regelungen	G0.30/0.38 Unerlaubte Verarbeitung/ Missbrauch personenbezogener Daten	G0.39 Schadprogramme	G0.42 Social Engineering
		Anforderungen													
Basisanforderungen	**APP.1.4.A1 Anforderungsanalyse f. d. Nutzung v. Apps**	- Welche Geschäftsprozesse werden unterstützt? - An welche IT-Komponenten wird d. App angebunden? - Festlegung von Sicherheitsanforderungen - Betrachtung d. Schutzbedarfs	X	X	X	X	X			X		X	X		
	APP.1.4.A2 Regelungen f. d. Verwendung v. mobilen Endgeräten und Apps	- Erstellung von Nutzerrichtlinien: > Welche Daten dürfen verarbeitet werden > Ist die private Nutzung gestattet? > Wie muss d. Nutzer sich in öffentl. Netzen verhalten > Verhalten bei Geräteverlust		X	X	X	X		X			X			X
	APP.1.4.A3 Verwendung Sicherer Quellen für Apps	-Bezug von Apps aus sicheren u. vertrauenswürdigen Quellen - interne Apps u. Apps d. schutzbedürftige Informationen verarbeiten, sollten über gesonderten Store bezogen werden		X		X	X	X	X			X		X	
	APP.1.4.A4 Test und Freigabe von Apps	- App muss Anforderungen d. Fachlichen Bereiche , der Informationssicherheit und d. Datenschutzes erfüllen - App wird vor Freigabe getestet - Definition von Freigabekriterien - Tests sollten gesamte MDM umfassen				X				X	X	X	X	X	X
	APP.1.4.A5 Minimierung und Kontrolle von App-Berechtigungen	- Kontrolle d. App-Permission-> Erhält die App nur die minimal nötigen Permissions? - Fixierung der Permissions bei sicherheitsrelevanten Einstellungen - regelmäßigen Überprüfung der Permissions		X			X		X			X	X	X	X
	APP.1.4.A6 Patchmanagement für Apps	- zeitnahe Einspielung von Updates - zentralisierte Updatesteuerung über vorhandene MDM-Lösung - Formulierung von Ausnahmebedingung, wenn Schwachstelle nicht zeitnah geschlossen werden kann								X	X				
	APP.1.4.A7 Sichere Speicherung Lokaler App-Daten	- wenn App auf interne Sharepoints zugreift = Sicherstellung der lokalen Datenhaltung - Daten sollten hinreichend verschlüsselt sein - keine Speicherung von vertraulichen Daten auf externen Speichermedien (SD-Card)		X	X		X								
	APP.1.4.A8 Verhinderung von Datenabfluss	- Sicherstellung der Verschlüsselung d. Kommunikation von sensiblen u. vertraulichen Daten - Testung der Kryptographie während d. Sicherheitsanalyse/Pentest - Testung der App auf ungewollte Protokollierungs-und Hilfsdateien		X			X						X		
Standardanforderungen	**APP.1.4.A9 Sichere Anbindung An Backend-Systeme**	- hinreichende kryptographische Absicherung d. Datenverkehrs zw. Backend u. anderen Systemen	X	X			X		X						
	APP.1.4.A10 Sichere Authentisierung von Apps	- Sicherstellung der Authentisierungsmechanismen - Werden die Daten mit einem sicheren Protokoll übertragen?	X	X			X		X						
	APP.1.4.A11 Zentrales Management von Apps	- Es SOLLTE eine MDM-Lösung etabliert werden - Es SOLLTE über Prüfmechanismen und Whitelists sichergestellt werden, dass nur geprüfte und freigegebene Apps verwendet werden können				X						X			
	APP.1.4.A12 Sichere Deinstallation von Apps	Wenn Apps deinstalliert werden, SOLLTEN auch alle von der App generierten Dateien gelöscht werden. Dabei SOLLTEN ebenso vom Betriebssystem zwischengespeicherte Daten mit Bezug zur App gelöscht werden, etwa Caches. Außerdem SOLLTEN auch Daten gelöscht werden, die auf externen Systemen, beispielsweise beim App-Anbieter, gespeichert wurden.		X		X	X					X	X		
Erhöhter Schutzbedarf	**APP.1.4.A13 Entwicklung von Fallback-Lösungen für Apps**	- Bereitstellung redundanter Systeme	X			X			X	X					
	APP.1.4.A14 Unterstützung zusätzlicher Authentisierungsmerkmale bei Apps	- Nutzung einer Zwei-Faktor Authentifizierung - Überprüfung der verwendeten Mechanismen der Authentisierung auf Sicherheit		X	X	X	X		X			X	X		
	APP.1.4.A15 Durchführung Von Penetrationstests für Apps	Durchführung von Penetrationtests/Sicherheitsanalysen vor dem Go-Live einer App. Dabei soll die gesamte Infrastruktur auf Sicherheitslücken untersucht werden. Die Test sollen in regelmäßigen Abständen wiederholt werden.							X	X	X				

Abbildung A.1: BSI Referenztabelle

BSI Anforderungen zum Schutz von mobilen Apps (weitere relevante Bausteine)
ISMS (Grundlage für alle weiteren Aktivitäten im Sicherheitsprozess)
ISMS.1 Sicherheitsmanagement
ORP (Organisatorische und personelle Sicherheitsaspekte)
ORP.4 Identitäts- und Berechtigungsmanagement
CON (Konzepte und Vorgehensweisen)
CON.1 Kryptokonzept
CON.2 Datenschutz
CON.3 Datensicherungskonzept
CON.4 Auswahl und Einsatz von Standardsoftware
CON.5 Entwicklung und Einsatz von Allgemeinen Anwendungen
CON.6 Löschen und Vernichten
OPS (Sicherheitsaspekte des operativen IT-Betriebs)
OPS.1.1.2 Ordnungsgemäße IT-Administration
OPS.1.1.3 Patch- und Änderungsmanagement
OPS.1.1.4 Schutz vor Schadprogrammen
OPS.1.1.5 Protokollierung
OPS.1.1.6 Software-Tests und -Freigaben
OPS.2.1 Outsourcing für Kunden
APP (Anwendungen und Dienste)
APP.3.1 Webanwendungen
APP.3.2 Webserver
SYS.1.1 Allgemeiner Server
APP.3.5 Webservices
OPS.2.2 Cloud-Nutzung
APP.1.2 Web-Browser
APP.4.3 Relationale Datenbanksysteme
SYS (IT-Systeme)
SYS.3.2.1 Allgemeine Smartphones und Tablets
SYS.3.2.3 iOS
SYS.3.2.4 Android
SYS.3.3 Mobiltelefon

Abbildung A.2: BSI Anforderungen zum Schutz von mobilen Apps

Checkliste Sicherheitsanalyse Mobile Apps

ID	Anforderungen nach Schutzbedarf	SK1	SK2	SK3	Status	Bemerkungen
V1	**Anforderungen an Architektur, Design und Bedrohungsanalysen/ M10 Fremdfunktionalitäten**					
1.1	Alle Komponenten der mobilen App sind identifiziert und für den Betrieb der App erforderlich.	✓	✓	✓	Pass	
1.2	Die App stammt aus einer sicheren Quelle.	✓	✓	✓		
1.3	Sicherheitsfunktionen sind niemals nur auf Client-Seite implementiert sondern immer auch im entsprechenden entfernten API-Endpunkt.	✓	✓	✓	Fail	
1.4	Es existiert eine Architekturübersicht über die mobile App und alle verbundenen API-Endpunkte und Security wurde in der Gesamtarchitektur berücksichtigt.		✓	✓	N/A	
1.5	Alle sensiblen Daten im Kontext der mobilen App wurden klar identifiziert.		✓	✓		
1.6	Für jede Komponente der App ist der angebotene fachliche Funktionsumfang und/oder Sicherheitsfunktionen/-mechanismen klar definiert.		✓	✓		
1.7	Für die mobile App und die genutzten API-Endpunkte wurde eine Bedrohungsanalyse durchgeführt und potentielle Bedrohungen und Gegenmaßnahmen identifiziert.		✓	✓		
1.8	Alle Sicherheitsfunktionen wurden in Form von zentralen Komponenten implementiert.		✓	✓		
1.9	Eine dedizierte Richtlinie zum Management von kryptographischen Schlüsseln (falls in der App genutzt) beschreibt den sicheren Umgang mit Schlüsseln über den gesamten Lebenszyklus, idealerweise basierend auf Standards (siehe hierzu BSI-IT-Grundschutz)		✓	✓		
1.10	Über Prüflisten und Whitelists sollte sichergestellt werden, dass nur geprüfte Apps verwendet werden.	✓	✓	✓		
1.11	Es gibt ein Mobile Device Management, über das Apps zentral gemanagt und aktualisiert werden. Updates können, falls notwendig erzwungen werden.	✓	✓	✓		
1.12	Es besteht im MDM eine klare und wirksame Mandantentrennung.	✓	✓	✓		
1.13	Es besteht eine zentrale Schnittstellen Verwaltung, die über das MDM administrierbar ist. Darunter fallen insbesondere Schnittstelle, wie Bluetooth, Infrarot, WLAN, SMS, MMS, GPS und USB.	✓	✓	✓		
V2	**Anforderungen an Datenspeicherung und Datenschutz/ M2 Unsicherer Datenspeicher**					
2.1	Die App speichert sensible Daten wie personenbezogene Daten, Anmeldedaten oder kryptographische Schlüssel unter Nutzung der vom jeweiligen Betriebssystem angebotenen sicheren Speichermechanismen.	✓	✓	✓		
2.2	Es werden keine sensiblen Daten außerhalb des App-Containers oder außerhalb des vom jeweiligen Betriebssystem angebotenen sicheren Speichermechanismus abgelegt.	✓	✓	✓		
2.3	Es werden keine sensiblen Daten in die Logfiles der App geschrieben.	✓	✓	✓		
2.4	Es werden keine sensiblen Daten mit Dritten geteilt - es sei denn dies wurde in der App-Architektur definiert und ist zur Erfüllung des Zwecks der App erforderlich.	✓	✓	✓		
2.5	Der Tastatur-Cache ist für alle sensiblen Texteingaben deaktiviert.	✓	✓	✓		
2.6	Es werden keine sensiblen Daten über IPC-Mechanismen zur Verfügung gestellt.	✓	✓	✓		
2.7	Es werden keine sensiblen Daten wie Passwörter oder Pins über die Benutzeroberfläche exponiert.	✓	✓	✓		
2.8	Es ist sichergestellt, dass betriebssystemgesteuerte Backups keine sensiblen App-Daten enthalten.	✓	✓	✓		
2.9	Die App entfernt sensible Daten aus der aktuellen Ansicht wenn der Hintergrundmodus aktiviert wird.	✓	✓	✓		
2.10	Die App hält sensible Daten nur solange wie nötig im Speicher und betroffene Speicherbereiche werden nach Nutzung explizit gelöscht.		✓	✓		
2.11	Die App erzwingt ein Minimum an Geräteschutz-Richtlinien wie das Definieren eines Gerätepassworts.		✓	✓		
2.12	Die App klärt den Nutzer über die Art und Weise der verarbeiteten personenbezogenen Daten auf und gibt dem Nutzer Security-Best-Practice-Empfehlungen zum Umgang mit der App.			✓		
V3	**Anforderungen an Kryptographie/ M5 Unzureichende Kryptographie**					
3.1	Verschlüsselung in der App basiert nicht nur auf symmetrischer Kryptographie mit hart-codierten Schlüsseln.	✓	✓	✓		
3.2	Die App nutzt bewährte Implementierungen zur Umsetzung kryptographischer Primitive.	✓	✓	✓		
3.3	Die App nutzt kryptographische Primitive passend zum spezifischen Anwendungsfall, konfiguriert nach Best-Practice Vorgaben dem Stand der Technik entsprechend.	✓	✓	✓		
3.4	Die App nutzt keine kryptographischen Protokolle oder Algorithmen die allgemein als veraltet und unsicher gelten.	✓	✓	✓		
3.5	Die App verwendet einen kryptographischen Schlüssel für genau einen Zweck und nicht für mehrere Zwecke.	✓	✓	✓		
3.6	Alle Zufallswerte werden über einen ausreichend sicheren kryptographischen Zufallszahlengenerator erzeugt.		✓	✓		
V4	**Anforderungen an Authentifizierung und Session Management/ M4 Unsichere Authentifizierung, M6 Unsichere Autorisierung**					
4.1	Falls die App Nutzern Zugriff auf entfernte Service APIs bietet wird am API-Endpunkt eine Authentifizierung z.B. mit Nutzername/Passwort durchgeführt.	✓	✓	✓		
4.2	Kommt Session-Management am API-Endpunkt zum Einsatz, so werden zufällig generierte Session-IDs erzeugt um Client-Anfragen zu authentifizieren und keine Nutzer-Anmeldedaten versandt.	✓	✓	✓		
4.3	Kommt statuslose Token-basierte Authentifizierung zum Einsatz, so werden die Token am Server mit einem sicheren Algorithmus signiert.	✓	✓	✓		
4.4	Der API-Endpunkt beendet die existierende Nutzersitzung sobald sich der Nutzer abmeldet.	✓	✓	✓		
4.5	Es existiert eine Passwort-Richtlinie die am entfernten API-Endpunkt erzwungen wird.	✓	✓	✓		
4.6	Der entfernte API-Endpunkt implementiert einen Mechanismus um sich gegen eine exzessive Anzahl von Login-Versuchen zu schützen.	✓	✓	✓		
4.7	Nach einer definierten Inaktivitätsdauer werden Nutzersitzungen am entfernten API-Endpunkt beendet und Zugriffs-Tokens werden nach Ablauf der Gültigkeitsdauer abgelehnt.	✓	✓	✓		
4.8	Biometrische Authentifizierung basiert auf dem Betriebssystem-basierten Entsperren des Keystores und nicht auf einer z.B. event-basierten API die einfach "true" oder "false" zurück liefert.		✓	✓		
4.9	Es gibt einen 2. Authentifizierungsfaktor und er wird am entfernten API-Endpunkt konsistent erzwungen.		✓	✓		
4.10	Sensible Transaktionen erfordern eine zusätzliche Authentifizierung (durch einen weiteren Authentifizierungsfaktor).			✓		
4.11	Die App informiert den Nutzer über alle Anmelde-Vorgänge am Nutzerkonto. Nutzer können eine Liste aller mit dem Konto verbundenen Geräte sehen und ausgewählte Geräte blockieren.			✓		
V5	**Anforderungen an Netzwerkkommunikation/M3 Unsichere Kommunikation**					
5.1	Daten werden zur Netzwerkkommunikation innerhalb der App durchgängig mit TLS verschlüsselt.	✓	✓	✓		
5.2	Die TLS-Einstellungen entsprechen aktuellen Best-Practices, oder erfüllen die Best-Practices so gut wie möglich falls das mobile Betriebssystem die Empfehlung nicht unterstützt.	✓	✓	✓		
5.3	Die App überprüft das X.509-Zertifikat des Servers beim TLS-Handshake. Die App akzeptiert nur Zertifikate die von einer vertrauenswürdigen Zertifizierungsstelle signiert wurden.	✓	✓	✓		
5.4	Die App nutzt entweder ihren eigenen Zertifikatspeicher oder nutzt Public Key-/ Zertifikats-Pinning und akzeptiert deshalb keine Verbindungen zu Endpunkten die andere Zertifikate/Schlüssel präsentieren - selbst wenn die Zertifikate von einer vertrauenswürdigen Zertifizierungsstelle signiert sind.		✓	✓		
5.5	Die App nutzt keine unsicheren Kommunikationskanäle wie Email oder SMS für kritische Operationen wie Konten-Registrierung oder Konten-Reaktivierung.		✓	✓		
5.6	Die App nutzt aktuelle Bibliotheken zum Aufbau von sicheren Kommunikationsverbindungen.		✓	✓		
V6	**Anforderungen zur Plattform-Interaktion/ M1 Unsachgemäße Plattformnutzung**					
6.1	Die App fordert vom Nutzer nur die unbedingt erforderlichen App-Berechtigungen.	✓	✓	✓		
6.2	Alle Eingaben von externen Quellen und dem Nutzer werden validiert und falls erforderlich bereinigt. Dazu gehören Daten aus der GUI, IPC Mechanismen wie Intents oder spezifische URL-Schemas und Netzwerk-Daten.	✓	✓	✓		
6.3	Die App bietet keine sensible Funktionalität über App-eigene URL-Schemas an - wenn doch, ist der Mechanismus angemessen geschützt.	✓	✓	✓		
6.4	Die App bietet keine sensible Funktionalität über Interprozesskommunikation (IPC) an - wenn doch, ist der Mechanismus angemessen geschützt.	✓	✓	✓		
6.5	JavaScript ist in WebViews deaktiviert wenn es nicht unbedingt erforderlich ist.	✓	✓	✓		
6.6	WebViews sind so konfiguriert, dass nur die minimal nötigen Protokoll-Handler erlaubt sind (Idealerweise nur HTTPS). Potentiell gefährliche Handler wie file, tel und app-id sind deaktiviert.	✓	✓	✓		
V7	**Anforderungen zu Code Qualität und Build-Einstellungen/M7 Schlechte Code-Qualität**					
7.1	Die App ist signiert und mit einem gültigen Zertifikat provisioniert, dessen privater Schlüssel angemessen geschützt ist.	✓	✓	✓		
7.2	Bibliotheken und Frameworks von Drittanbietern die die App nutzt wurden auf Schwachstellen geprüft.	✓	✓	✓		
7.3	Die App führt eine sichere Fehlerbehandlung durch, indem sie Exceptions abfängt und kontrolliert behandelt.	✓	✓	✓		
7.4	Treten in Sicherheitsfunktionen Fehler auf lehnt die App-Fehlerbehandlung die Zugriffe standardmäßig ab.	✓	✓	✓		
7.5	Das Speichermanagement (Allokation und Freigabe von Speicher) erfolgt in unmanaged code auf sichere Weise.	✓	✓	✓		
7.6	Angebotene Sicherheitsfunktionen der Entwicklungsumgebung wie Byte-Code Minimierung, Stack-Protection, PIE-Support und automatisches Reference-Counting sind aktiviert.	✓	✓	✓		
V8	**Anforderungen an Manipulationssicherheit/Resilienz/ M8 Code Manipulation/ M9 Reverse Engineering**					
8.1	Die App erkennt und reagiert auf ein gerootetes Gerät oder Geräte mit jailbreak indem der Nutzer gewarnt oder die App beendet wird.		✓	✓		
8.2	Die App verhindert Debugging und/oder erkennt und reagiert auf einen Debugger. Alle verfügbaren Debugging-Protokolle müssen abgedeckt werden.		✓	✓		
8.3	Die App erkennt und reagiert auf Manipulation von Code und Daten im eigenen Arbeitsspeicherbereich.		✓	✓		
8.4	Programmatische Abwehrmaßnahmen werden durch Obfuskierung geschützt, was wiederum De-Obfuskierung mittels dynamischer Analyse verhindert.			✓		
8.5	Die App implementiert einen Mechanismus zur Gerätebindung bei dem der Geräte-Fingerabdruck aus mehreren einzigartigen Geräteeigenschaften ermittelt wird.			✓		
8.6	Alle ausführbaren Dateien und Bibliotheken der App sind entweder auf Dateiebene verschlüsselt und/oder wichtige Code- und Datenabsegmente in ausführbaren Dateien sind verschlüsselt oder durch Packing obfuskiert. Triviale statische Analyse offenbart keinen wichtigen Code oder Daten.			✓		
8.7	Wenn das Ziel der Obfuskierung der Schutz sensibler Logik wie Algorithmen oder Berechnungen ist, so wird ein angemessener Obfuskierungsmechanismus, dem Stand der Technik entsprechend, genutzt der resilient gegen manuelle und automatisierte De-Obfuskierungsangriffe ist. Die Wirksamkeit der Obfuskierungsmethode muss durch manuelle Tests überprüft werden. Es ist zu beachten, dass hardware-basierte Isolations-Mechanismen softwarebasierter Obfuskierung vorzuziehen sind.			✓		

Legende

Symbole	Definition
Pass	Die App erfüllt die genannte Anforderung.
Fail	Die App kann die genannte Anforderung nicht erfüllen.
N/A	Anforderung konnte nicht getestet werden.

Abbildung A.3: Checkliste für die Sicherheitsanalyse mobiler Apps

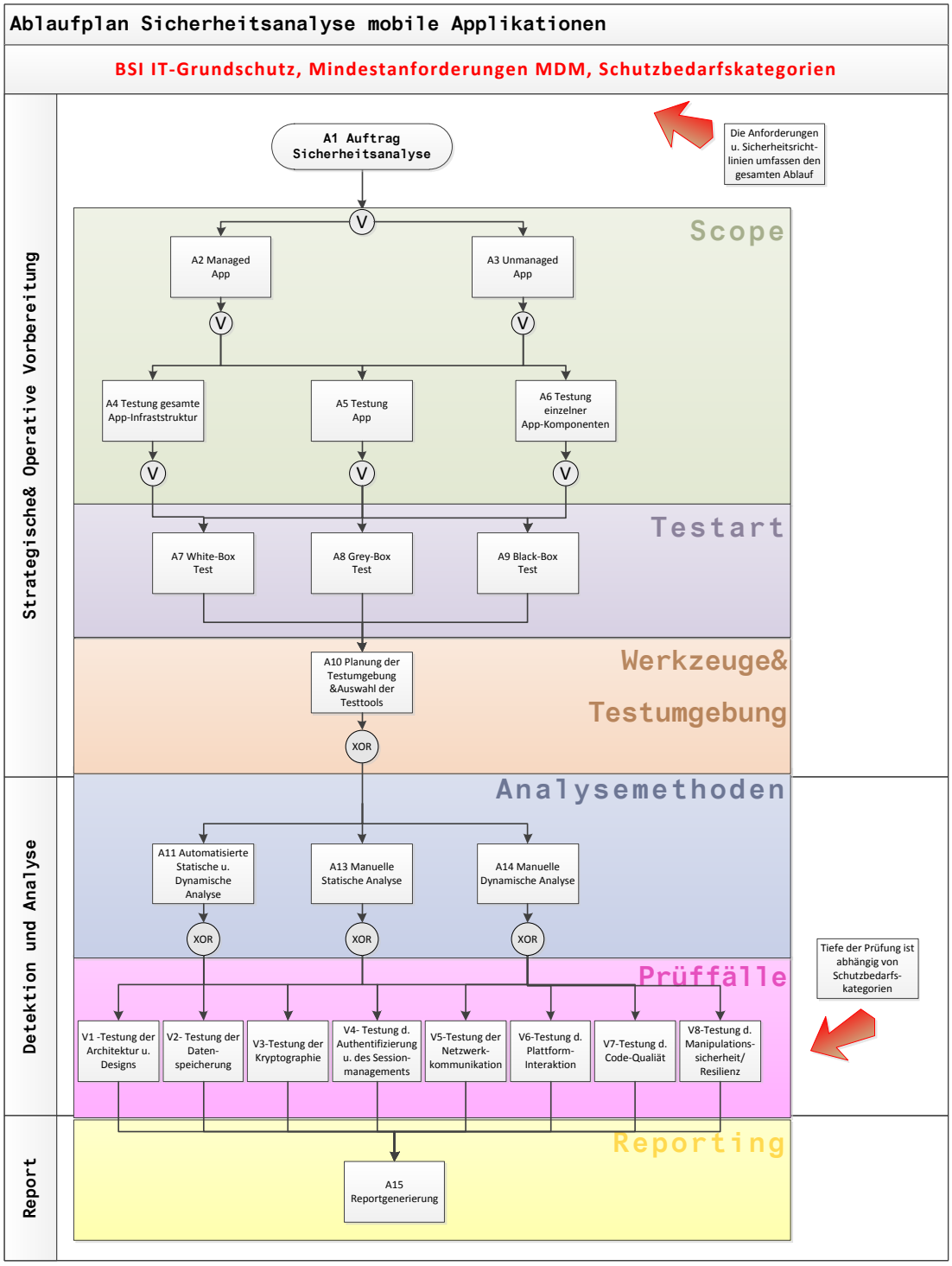


Abbildung A.4: Ablaufplan für die Sicherheitsanalyse einer mobilen App

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der aufgeführten Hilfsmittel angefertigt habe.

Ort, Datum Unterschrift