FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Effiziente automatisierte Erst-Analyse von Android-Anwendungen in Bezug auf IT-Sicherheit basierend auf Open-Source-Werkzeugen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Fabian Guschlbauer BSc
Matrikelnummer 1204451

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuer: Thomas Grechenig
Mitwirkung: Florian Fankhauser

Wien, 10. Oktober 2019

(Unterschrift Verfasser) (Unterschrift Betreuer)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Effiziente automatisierte Erst-Analyse von Android-Anwendungen in Bezug auf IT-Sicherheit basierend auf Open-Source-Werkzeugen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Fabian Guschlbauer BSc
Matrikelnummer 1204451

ausgeführt am
Institut für Information Systems Engineering
Forschungsgruppe Industrial Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuer: Thomas Grechenig
Mitwirkung: Florian Fankhauser

Wien, 10. Oktober 2019

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Eidesstattliche Erklärung

Fabian Guschlbauer BSc
1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Danksagung

An dieser Stelle möchte ich all jenen danken, die mir während meines Studiums und letztlich bei dieser Arbeit durch fachliche und persönliche Unterstützung sowie Motivation zur Seite standen, und zum erfolgreichen Abschluss beigetragen haben.

Mein Dank gilt meinem Betreuer Univ. Prof. DI Dr. Thomas Grechenig sowie den betreuenden Assistenten Florian Fankhauser und Clemens Hlauschek für die Bereitstellung des Themas, ihre Unterstützung und konstruktive Kritik, die zum Erfolg dieser Diplomarbeit beigetragen haben.

Herzlich bedanken möchte ich mich auch bei meiner Familie, insbesondere meinen Eltern und Großeltern, die mir mein Studium ermöglicht und mich in all meinen Entscheidungen unterstützt haben.

Schließlich gilt noch ein besonderer Dank meinen Studienkollegen und Freunden Thomas und Bernhard, mit denen ich erfolgreiche Projekte und Prüfungen feiern durfte, ebenso wie Maria, Sabina, Michael und Christian, die mir mit Motivation und Ausgleich zum Erfolg verholfen haben und mir sowohl in guten, als auch schlechten Momenten treu zur Seite standen. Danke für die vielen schönen und spannenden Erinnerungen, die ich während meiner Studienzeit mit euch sammeln durfte.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Kurzfassung

Das Betriebssystem Android liegt mit knapp 85% Marktanteil weltweit an führender Position, und der Android eigene Google Play Store verzeichnet rund 2.8 Millionen mobile Anwendungen, die vielen Menschen den Mehrwert ihrer tragbaren Geräte aufzeigen. Gefahren, die diese Anwendungen mit sich bringen, bleiben allerdings oftmals im Verborgenen.

In den vergangenen Jahren wurden im Bereich der mobilen Anwendungssicherheit zahlreiche Publikationen und Werkzeuge veröffentlicht. Dabei wurden die unterschiedlichsten Sicherheitsbereiche wie etwa Kommunikation, Kryptografie oder Datenzugriff und Speicherung fokussiert und Sicherheitslücken aufgezeigt. Um die Analysefunktionalität von unterschiedlichen Werkzeugen zu bündeln und eine automatisierte Sicherheitseinstufung in Bezug auf IT-Sicherheit von Android-Anwendungen durchzuführen, wurde innerhalb dieser Arbeit ein webbasiertes Testing as a Service (TaaS) Framework entwickelt. Dieses integriert unterschiedliche existierende Sicherheitsanalyse-Werkzeuge und führt basierend auf deren Ergebnissen eine Sicherheitseinstufung der Anwendung in Bezug auf die Empfehlungen des Open Web Application Security Project (OWASP) Top 10 sowie der CERT-Abteilung des Software-Engineering Institut der Carnegie Mellon Universität (CMU) durch. Um die am häufigsten auftretenden Sicherheitsprobleme, die sichersten sowie gefährdetsten Google Play Store-Kategorien und Veränderungen der Sicherheit im Bereich Kryptografie in den vergangenen sechs Jahren aufzuzeigen, wurde im Rahmen einer durchgeführten Evaluierung automatisiert die Sicherheit von jeweils 100 Android-Anwendungen aus den 58 unterschiedlichen Google Play Store-Kategorien (e.g. *Android Wear, Business oder Finance*) bestimmt.

Bei der Analyse der 5.800 Google Play Store-Anwendungen durch die angebundenen Werkzeuge sowie die Sicherheitseinstufung, der in der Arbeit definierten Kategorien mangelnder Angriffsschutz, Sicherheitsinvalidierung, anwendungsübergreifende Zugriffskontrolle, Datenlecks und Eingabevalidierung, wurde bei rund 76% der untersuchten Anwendungen mindestens ein schwerwiegender Sicherheitsverstoß festgestellt. Das Ergebnis der sicherheitskritisch eingestuften Anwendungen variiert dabei in den unterschiedlichen Google Play Store-Kategorien. In sicherheitskritischen Kategorien wie *Business, Family Education, Finance* und *Medical* konnten geringere Häufigkeiten kritischer Anwendungen festgestellt werden. Die meisten Probleme zeichneten sich im Bereich der Sicherheitsinvalidierung beziehungsweise genauer in den Bereichen der Zertifikatvalidierung sowie Kryptografie ab. Im Gebiet der Kryptografie wurden Regeln zur Verwendung des *Electronic Code Book Mode (ECB)*, *Cipher Block Chaining Mode (CBC)* mit konstant definierten Initialisierungsvektor (IV) sowie statischen Seeds evaluiert. Dabei konnte insgesamt ein Rückgang der Sicherheitsprobleme in diesem Bereich in den letzten sechs Jahren manifestiert werden.

Schlüsselwörter

Android-Anwendungssicherheit, automatisierte Sicherheitseinstufung, Google Play Store Sicherheitsevaluierung, TaaS, Microservice-basiertes Framework, OWASP Mobile Top 10, Regeln der CMU, Android API-Versionen, nativer Quellcode, Repacking, Berechtigungen, Kryprografie, Zertifikatvalidierung, Anwendungsverteilung, Inter-Component Communication, Intent, Debuggable Release, Datenlecks, SQL-Injection, XSS

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

The operating system Android is with almost 85% market share worldwide in a leading position and the Android Google Play Store counts around 2.8 million mobile applications. However, the threats these applications involve remain often untold. In the last few years publications and tools with a focus on security areas such as communication, cryptography and data access or data storage have been published. To compose the functionality of different analysis tools and to accomplish an automated security classification of Android applications with a focus on IT-Security, a web-based Testing as a Service (TaaS) framework has been implemented, which integrates various existing security analysis tools. Based on the analysis results of the integrated tools and the recommendations of the Open Web Application Security Project (OWASP) Top 10 as well as the rules of the CERT Department of the Software Engineering Institute of Carnegie Mellon University (CMU) the framework identifies the security class of the analyzed applications. In order to identify the most common security threats and issues, the most secure and vulnerable Google Play Store categories as well as the changes in the security field of cryptography in the last six years, an evaluation was conducted as part of the thesis to automatically determine the security of 100 android applications from 58 different Google Play Store categories that include categories such as *Android Wear, Business or Finance*. Analyzing the 5.800 Google Play Store applications using the framework attached tools as well as the security classification of the defined categories *Insufficient Attack Protection, Security Invalidation, Access Control, Sensitive Data Leakage* and *Input Validation* resulted in at least one security issue in about 76% of the evaluated applications. The result of critical or insecure classified applications varied in different Google Play Store categories, with lower rates of critical applications found in security-critical categories such as *Business, Family Education, Finance* and *Medical*. Most of the issues were located in the *Security Invalidation* category or more precisely in the areas of *Certificate Validation* and *Cryptography*. In the field of Cryptography rules concerning the usage of *Electronic Code Book Mode (ECB)*, *Cipher Block Chaining Mode (CBC)* with non-random Initialization Vector (IV) as well as static seeds have been evaluated and resulted overall in a decline of problems in the past six years.

Keywords

Android Application Security, Automated Security Classification, Google Play Store security Evaluation, TaaS, Microservice-basiertes Framework, OWASP Mobile Top 10, CMU-Rules, Android API-Versionen, Native-Code, Repacking, Permissions, Cryptography, Certificate Validation, Application Distribution, Inter-Component Communication, Intent, Debuggable Release, SQL-Injection, XSS

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Problemstellung . . . 1
1.2 Motivation . . . 1
1.3 Zielsetzung . . . 2
1.4 Methodisches Vorgehen . . . 3
1.5 Aufbau der Arbeit . . . 4

2 Related Work **5**

3 Grundlagen der IT-Sicherheit und Sicherheitsanalyse **10**
3.1 Einführung in die IT-Sicherheit . . . 10
3.2 Unterschiedliche Software-Analyseverfahren . . . 11
3.2.1 Statische Analyseverfahren . . . 12
3.2.2 Dynamische Analyseverfahren . . . 13
3.2.3 Hybride Analyseverfahren . . . 15
3.3 Einsatz von Cloud-basierten Services . . . 15

4 Einführung in mobile Anwendungen **17**
4.1 Unterschiedliche Typen mobiler Anwendungen . . . 17
4.1.1 Native Anwendungen . . . 17
4.1.2 Webanwendungen . . . 17
4.1.3 Hybride Anwendungen . . . 18
4.2 Struktur und Aufbau . . . 18
4.2.1 Struktur von Android-Anwendungen . . . 19
4.2.2 Android-Anwendungskomponenten . . . 20
4.2.3 Inter-Component Communication (ICC) . . . 22
4.2.4 Berechtigungen . . . 24
4.2.5 Speicher . . . 25

5 Architektur und Sicherheitskonzept von Android **28**
5.1 Architektur . . . 28
5.1.1 Linux-Kernel-Schicht . . . 28
5.1.2 Bibliotheken-Schicht . . . 29
5.1.3 Laufzeitschicht . . . 29
5.1.4 Anwendungsframework-Schicht . . . 30
5.1.5 Anwendungsschicht . . . 31
5.2 Sicherheitskonzepte . . . 31
5.2.1 Berechtigungskonzept . . . 31
5.2.2 Kapselung/Sandboxing von Anwendungen . . . 32
5.2.3 Anwendungsverteilung . . . 33
5.2.4 Anwendung/Code Signing . . . 33
5.2.5 Datenverschlüsselung . . . 33

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6 Sicherheitsstandards von Android-Anwendungen nach OWASP Top 10 und der CERT-Abteilung des Software-Engineering Instituts der CMU 34
6.1 Falsche Plattform Verwendung . . . 35
6.2 Unsichere Datenspeicher . . . 37
6.3 Unsichere Kommunikation . . . 38
6.4 Unsichere Authentifizierung . . . 39
6.5 Unzureichende Kryptografie . . . 40
6.6 Unsichere Autorisierung . . . 40
6.7 Schlechte Codequalität . . . 41
6.8 Codemanipulation . . . 41
6.9 Reverse Engineering . . . 42
6.10 Irrelevante Funktionalität . . . 42

7 Identifizierung und Kategorisierung ausgewählter Sicherheitsrisiken zur automatisierten Analyse 43
7.1 Mangelnder Angriffsschutz . . . 43
7.1.1 API-Versionen, Bibliotheken und Drittquellen . . . 43
7.1.2 Nativer Quellcode . . . 44
7.1.3 Repacking . . . 44
7.1.4 Berechtigungen . . . 45
7.2 Sicherheitsinvalidierung im Zusammenhang mit Zertifikaten, Kryptografie und Anwendungsverteilung . . . 45
7.2.1 Kryptografie . . . 45
7.2.2 Zertifikatvalidierung . . . 46
7.2.3 Anwendungsverteilung und Anwendungsmodifizierung . . . 46
7.3 Anwendungsübergreifende Zugriffskontrolle . . . 47
7.3.1 Inter-Component Communication (ICC) . . . 47
7.3.2 Intent . . . 47
7.3.3 Debuggable Release . . . 48
7.4 Datenlecks – Unbefugter Zugriff auf sensible Informationen . . . 48
7.4.1 Sicherheitslücken bezüglich Hardware-, Softwareidentität sowie unerwünschten Zugriff auf sensible Daten . . . 48
7.4.2 Unsichere Übertragungskanäle . . . 48
7.4.3 Unsichere Datenspeicherung . . . 49
7.4.4 Unsichere Aufbewahrung von Zugangsdaten . . . 49
7.5 Eingabevalidierung . . . 49
7.5.1 WebView und XSS . . . 50
7.5.2 SQL-Injection . . . 50

8 Open-Source-Werkzeuge zur Sicherheitsanalyse von Android-Anwendungen 51
8.1 Reverse Engineering und Reengineering Werkzeuge . . . 51
8.1.1 Assembler/Disassembler . . . 51
8.1.2 Java Decompiler . . . 52
8.2 Android Open-Source-Analysewerkzeuge . . . 53
8.2.1 APK-Parser . . . 53
8.2.2 Argus Statisches Analyse-Framework (SAF) / Argus-Amandroid . . . 53
8.2.3 Androguard . . . 54
8.2.4 AndroBugs . . . 55
8.2.5 Androwarn . . . 56
8.2.6 ApkAnalyser . . . 57
8.2.7 Permission-Lint (P-Lint) . . . 58

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

8.2.8 Quick Android Review Kit (QARK) 59
8.2.9 Secure, Unified, Powerful and Extensible Rust Android Analyzer (SUPER) 59
8.2.10 Mobile Security Framework – MobSF 60
8.2.11 M-Perm 60
8.2.12 FlowDroid 61

9 Beschreibung des Frameworks zur automatisierten Analyse der Anwendungssicherheit 63
9.1 Anforderungen und Konzeption 63
9.2 Architektur- und Design-Entscheidungen 64
9.2.1 Architektur 64
9.2.2 Technologien 65
9.3 Implementierung 67
9.3.1 Microservice Setup 67
9.3.2 Docker Setup 69
9.3.3 Verwaltung und Verarbeitung von Android-Anwendungen 70
9.3.4 Integration von Analysewerkzeugen in des Framework 73
9.3.5 Report Verarbeitung der integrierten Analysewerkzeuge 76
9.3.6 Ablauf der Android-Anwendungsanalyse und Sicherheitseinstufung . . . 79
9.4 Auswertung der Sicherheitseinstufung 81
9.5 Verwendung des Frameworks 81
9.5.1 Analyse Modi des Frameworks 82
9.5.2 Sicherheitseinstufung von Android-Anwendungen 84
9.6 Einbindung neuer Analysewerkzeuge 84
9.6.1 Vorbedingungen 84
9.6.2 Wahl der zuständigen Analyse-Service Komponente 85
9.6.3 Integration eines Werkzeugs 85

10 Google Play Store-Anwendungsanalyse 86
10.1 Zusammenfassender Werkzeugeinsatz zur Evaluierung der definierten Sicherheitskategorien 86
10.2 Beschreibung der verwendeten Systemumgebung 87
10.2.1 Framework Anpassung für den AWS Einsatz 87
10.2.2 AWS-Setup und Deploy 87
10.3 Repräsentative Auswahl an Android-Anwendungen 89
10.4 Manuelle Validierung ausgewählter Sicherheitsprobleme 89
10.5 Automatisierte Analyse und Auswertung der Evaluierung 92
10.5.1 Übersicht des Gesamtergebnisses der Sicherheitseinstufung 93
10.5.2 Anzahl der beendeten sowie abgebrochenen Werkzeug Analysen 94
10.5.3 Benötigte Analysezeit der Werkzeuge 95
10.5.4 Auflistung der am häufigsten erkannten Sicherheitsprobleme 98
10.5.5 Anzahl der Sicherheitsrisiken, die in den einzelnen Anwendungen festgestellt werden konnten 100
10.5.6 Verteilung der Sicherheitsrisiken innerhalb der definierten Sicherheitskategorien 101
10.5.7 Sicherheitsrisiko der unterschiedlichen Anwendungskategorien 102

11 Fazit 105

12 Zusammenfassung und Ausblick 109

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Literatur **113**
Wissenschaftliche Literatur 113
Online-Referenzen 120

A Anhang **126**
A.1 Integrierte Werkzeuge 126
A.2 Software-Versionen 126
A.3 Konfiguration der Sicherheitseinstufung 126

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildungsverzeichnis

3.1 CIA Triad [93] 11

4.1 Native, webbasierte und hybride mobile Anwendungen [149] 18
4.2 Vereinfachte APK-Anwendungsstruktur [88] 20

5.1 Vereinfachte Darstellung der Android-Architektur 28
5.2 Binder IPC [88] 29
5.3 Android-Quellcode Kompilierungsprozess 30
5.4 Berechtigungsänderungen zur Laufzeit am Beispiel von WhatsApp 32

9.1 Framework Architektur 64
9.2 Einsatz von Technologien und Frameworks in den Microservice Komponenten Service-Registration-Service, Main-Service und Analysis-Service 68
9.3 Upload einer APK-Datei ins Framework über das Dateiauswahlmenü 70
9.4 Upload einer APK-Datei ins Framework über den integrierten Google Play Store Downloader 71
9.5 Vereinfachtes Sequenzdiagramm der Upload Funktionalität des Frameworks 72
9.6 Vereinfachtes Sequenzdiagramm des Cleanup Service 73
9.7 Vereinfachtes Klassendiagramm des Factory Pattern zur Erstellung von AnalysisWorkern 74
9.8 Vereinfachtes Klassendiagramm des Chain Pattern zum Parsen der Reports von unterschiedlichen Analysewerkzeugen 76
9.9 Vereinfachtes Klassendiagramm des Facade Pattern zum Persistieren der Reports von unterschiedlichen Analysewerkzeugen 78
9.10 Vereinfachtes Sequenzdiagramm der Analyse Funktionalität des Frameworks 79
9.11 Vereinfachtes Sequenzdiagramm der Qark Report Verarbeitung des Frameworks . . . 80
9.12 Framework Startseite mit Android-Anwendungsübersicht 82
9.13 Konfiguration der Analyse 82
9.14 Anwendungsdownload aus dem Google Play Store 83
9.15 Start der automatisierten Google Play Store-Anwendungsanalyse 84

10.1 AWS-Setup der Evaluierungsumgebung 88
10.2 AWS ECS Main-Service und Service-Registration Cluster 89
10.3 AWS ECS Analysis-Service Cluster 89
10.4 Ergebnis der Sicherheitseinstufung von 5.800 analysierter Google Play Store Android-Anwendungen 94
10.5 Erfolgreiche Analysezeit der eingesetzten Werkzeuge in Sekunden 96
10.6 Erfolgreiche Analysezeit der eingesetzten Werkzeuge in Sekunden 97
10.7 Erfolgreiche Analysezeit der eingesetzten Werkzeuge in Sekunden 97
10.8 Darstellung der QARK Analysezeit in Sekunden 98
10.9 Darstellung der festgestellten Problemanzahl in den analysierten Anwendungen . . . 101
10.10 Verteilung der kritischen Anwendungen aus den Google Play Store-Kategorien mit niedrigsten (grün schattiert) und höchsten (rot schattiert) Sicherheitsrisiko in den fünf definierten Sicherheitskategorien 103

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.11 Verteilung der kritischen Anwendungen aus den Google Play Store-Kategorien mit niedrigsten (grün schattiert) und höchsten (rot schattiert) Sicherheitsrisiko in der Security Invalidation Sicherheitskategorie 104

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Tabellenverzeichnis

4.1 Übersicht der 58 Google Play Store Kategorien 19
4.2 Dangerous Permissions [19] 24

8.1 Auflistung der Werkzeuge mit einem Überblick zu den jeweiligen Einsatzzwecken . 62

10.1 Verwendete EC2-Instanz Typen 88
10.2 Übersicht der Risikostufen 92
10.3 Übersicht der beendete/abgebrochene Analysen der Werkzeuge 95
10.4 Analysezeit pro Werkzeug in Sekunden (Timeout bei 1800 Sekunden) 95
10.5 Am häufigsten gefundene Sicherheitslücken mit Einstufung POTENTIALINSECURE und INSECURE 100
10.6 Top 10 der geringsten Anzahl an Sicherheitsrisiken pro Anwendung 100
10.7 Sicherheitseinstufung der Anwendungen in den fünf definierten Sicherheitskategorien 101
10.8 Verteilung der Sicherheitsrisiken innerhalb der fünf definierten Sicherheitskategorien 102
10.9 Google Play Store-Kategorien mit den meisten beziehungsweise wenigsten als kritisch eingestuften Anwendungen 103

11.1 Sicherheitsverstöße gegen die Regeln R1, R2 und R6 der Neuevaluierung im Vergleich zum Ergebnis von Egele et al. [85] 108

A.1 In das Framework integrierte Werkzeuge 126
A.2 Zur Entwicklung eingesetzte Software-Versionen 126
A.3 Verteilung der Sicherheitsrisiken in den exisiterenden Google Play Store-Kategorien 130

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Liste der Listings

4.1 Ausschnitt einer privaten Komponentendefinition im AndroidManifest.xml [16] . 21
4.2 Beispiel eines expliziten Intents [14] 22
4.3 Beispiel eines impliziten Intents der einen *ACTION_SEND* Intent sendet [14] . . 23
4.4 Beispiel eines Intent Filters einer Aktivität die einen `ACTION_SEND` Intent erhält wenn Daten vom Typ Text empfangen werden [14] 23
4.5 Berechtigungsdefinition im AndroidManifest.xml [19] 24
4.6 Ausschnitt einer Berechtigungsanforderung im AndroidManifest.xml [19] 25

8.1 Definition von unsicheren TrustManager durch Argus-SAF [30] 54
8.2 Definition von unsicheren SSLSockets durch Argus-SAF [30] 54
8.4 Ergebnis der M-Perm Analyse zur Überprivilegierung von WhatsApp 61

9.1 Einfaches Beispiel eines Dockerfile anhand des Main-Service 70
9.2 Implementierung des Aufrufs der einzelnen ReportParser innerhalb der doReportParsing-Methode der ReportParsingChain 77
9.3 Pseudoimplementierung einer typischen apply-Methode eines ReportParsers . . . 77
9.4 Beispielkonfiguration der ICC Sicherheitsevaluierung 81

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

1 Einleitung

Im ersten Kapitel der Arbeit wird die Problematik und Relevanz von Sicherheit im Bereich mobiler Anwendungen eingeführt, die Ziele und das methodische Vorgehen definiert, sowie der Aufbau der Arbeit vorgestellt.

1.1 Problemstellung

Mobile Endgeräte sind heutzutage ein fixer Bestandteil des täglichen Lebens und begleiten viele Menschen beinahe durch jeden Lebensbereich. Egal ob bei Smartphone, Tablet, Smartwatch oder einfachen Internet of Things, meist stellen vor allem die auf den Endgeräten installierten Anwendungen eine willkommene Unterstützung im Alltag und somit einen Mehrwert dar.

Das Android-Betriebssystem ist bei Smartphones mit knapp 85% Marktanteil [190] weltweit klar in marktführender Position. Ähnlich sieht es mit den Android-Anwendungen aus. Im September 2018 waren rund 2.8 Millionen [27] Android-Anwendungen im Google Play Store [127] verfügbar. Das Angebot an verfügbaren Anwendungen sowie die Zahl der Downloads steigt dabei stetig und lässt sich über viele Jahre hinweg zurückverfolgen. Wir vermuten, dass diese Zahlen, speziell aufgrund der immer weiter steigenden digitalen Vernetzung in den unterschiedlichsten Lebensbereichen, in den nächsten Jahrzehnten noch rascher ansteigen.

Über Gefahren und Sicherheitslücken, die so manche Anwendungen beispielsweise durch falsche Datenspeicherung, Validierung, unsichere APIs und Verschlüsselungsverfahren mit sich bringen wird im Allgemeinen zu wenig berichtet. Gerade weil uns „smarte“ Geräte und deren Anwendungen durch den ganzen Tag begleiten, geraten sie an hochsensible Daten und stellen einen besonders sicherheitsrelevanten Bereich dar. Auf welche Daten, Sensoren, Speicherbereiche oder andere Funktionen der hoch technischen Geräte von den installierten Anwendungen zugegriffen werden kann, ist allerdings ohne genauere Analyse nicht ersichtlich. Ebenso bleibt der interne Umgang mit Daten und die Verwaltung dieser ohne genauere Untersuchung verborgen.

1.2 Motivation

Wie in allen Bereichen der Software-Entwicklung existieren auch im Bereich der Android-Anwendungsentwicklung State-of-the-Art-Programmierstandards, die es Entwicklern und Entwicklerinnen erleichtern sollen, robuste, wartbare Software zu bauen und mit sensiblen Daten umzugehen. Neben wissenschaftlichen Quellen wie den Arbeiten von Egele et al. [85] und Colton et al. [66], die sich auf Programmierregeln und Empfehlungen im Verschlüsselungs- und Berechtigungsbereich spezialisieren, gibt es eine Reihe an weiteren Quellen, die State-of-the-Art-Programmierempfehlungen definieren. Beispielsweise von der CERT-Abteilung des Software-Engineering Instituts der Carnegie-Mellon-Universität, das unter anderem für Sicherheit und Standardisierung im Entwicklungsbereich bekannt ist, gibt es acht Android-spezifische Programmierregeln und Empfehlungen, um Anwendungen von Beginn an sicher zu gestalten [22]. Weiters gibt es auch für die Android-Entwicklung, wie aus der Webentwicklung bekannt, eine Liste der OWASP (Open Web Application Security Project) Top 10 [179], welche Standards zur sicheren Anwendungsentwicklung beschreiben und anhand deren es möglich ist, Sicherheitsprobleme aufzudecken.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

In den letzten Jahren wurde eine Vielzahl von unterschiedlichen sowohl kommerziellen als auch Open-Source-Softwareprodukten veröffentlicht, um Anwendungen beispielsweise mithilfe von statischen Analysen, auf die Einhaltung von Sicherheitsrichtlinien und Fehlkonfigurationen zu prüfen. Diese Analysewerkzeuge fokussieren sich dabei meist auf einzelne, unterschiedliche Sicherheitsbereiche. Der konkrete Funktionsumfang vieler Werkzeuge ist allerdings meist undurchsichtig, Dokumentationen unvollständig und die erhaltenen Ergebnisse einer Analyse sind schwer interpretierbar. Dadurch ist es auch für Tester und Testerinnen einer Sicherheitsabteilung kaum möglich, rasch konkrete Aussagen über die Sicherheit einer analysierten Anwendung zu treffen. Darüber hinaus ist die Installation und Verwendung solcher Analysewerkzeuge meist äußerst komplex und aufwendig. Ein zentrales und vor allem manuelles Testen aller oder zumindest einer großen Anzahl an Anwendungen ist daher umständlich und mit hohem Aufwand verbunden.

1.3 Zielsetzung

In dieser Arbeit werden Android-Anwendungen in Bezug auf IT-Sicherheit und Datenschutz automatisiert analysiert und eine Sicherheitseinstufung durchgeführt. Die automatisierte Analyse kombiniert dabei verschiedene verfügbare statische Open-Source-Analysewerkzeuge, die sich jeweils auf spezifische Sicherheitsbereiche spezialisieren, und führt weiters, basierend auf den Ergebnissen der einzelnen Analysewerkzeuge, eine eigene vergleichbare Sicherheitseinstufung von Anwendungen durch.

Die aus dem großen Angebot bestehender Analysewerkzeuge selektierten und durch manuelle Tests bewährten Werkzeuge, die für die Sicherheitseinstufung verwendet werden, stellen für weiterführende Arbeiten in diesem Bereich einen großen Mehrwert dar.

Ziel der Arbeit ist es, eine Analyse einer aussagekräftigen Selektion aus den unterschiedlichen Google Play Store-Anwendungskategorien zu präsentieren, um zu erschließen, wie weit kritische Sicherheitsprobleme in den jeweiligen Anwendungskategorien vertreten sind.

Um Android-Anwendungen im großen Stil auf eine breite Palette von sicherheitskritischen Aspekten automatisiert zu analysieren und weiters zu kategorisieren und wissenschaftlich auszuwerten, wird ein skalierendes verteiltes Framework resultieren, das die Stärken unterschiedlicher statischer ausgewählter existierender Open-Source-Analysewerkzeuge vereint und ein einheitliches beziehungsweise vergleichbares Analyseergebnis liefert. Ein für möglichst viele Anwendungen kompatibles Framework bietet dabei nicht nur die Möglichkeit, eine Klassifizierung einer großen Menge an im Google Play Store enthaltenen Anwendungen durchzuführen, sondern auch weiterführende neuartige Testanwendungen zu entwickeln und diese einfach in ein globales verteiltes Analyse-Framework mit zentraler Datenverwaltung einzubinden. Entwickler und Entwicklerinnen eines Testwerkzeugs können auf diese Weise nicht nur von allen Vorteilen einer verteilten Anwendungsanalyse, sondern auch von einer einheitlichen Benutzeroberfläche profitieren. Darüber hinaus kann das resultierende Framework für zukünftige wissenschaftliche Arbeiten und komplexe Analysen in ähnlichen Bereichen verwendet werden, um weiterführende Ergebnisse zu erlangen. Interessant wäre etwa ein Vergleich beziehungsweise die Veränderung der Anwendungssicherheit in den kommenden Jahren.

Ziel des Frameworks ist es nicht, neue Sicherheitsrichtlinien oder Entwicklungsmuster zu definieren, sondern Android-Anwendungen in Bezug auf Sicherheit und Datenschutz automatisiert zu analysieren und so eine Sicherheitseinstufung in Bereichen wie Zugriffskontrolle, Validierung, Schutz vor Missbrauch und Datenlecks durchzuführen sowie die allgemeine Sicherheit einer Anwendung zu definieren.

Hypothese 1: Basierend auf den aufgedeckten Problemen kann eine Risiko- beziehungsweise Sicherheitseinstufung von Android-Anwendungen durchgeführt werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Hypothese 2: In allen Google Play Store-Anwendungskategorien sind sicherheitskritische Probleme angesiedelt, wobei es Kategorien gibt, in denen eine durchschnittliche Anwendung ein höheres Sicherheitsrisiko darstellt.

Hypothese 3: Wie Egele et al. [85] bereits von Mai bis Juli 2012 evaluierten, sind Sicherheitslücken in Android-Anwendungen durch den Einsatz des Electronic Code Book (ECB) Mode und des Ciphertext Block Chaining (CBC) Verschlüsselungsalgorithmus weit verbreitet. Sechs Jahre später stellt sich die Frage nach dem aktuellen Stand dieser Evaluierung. Eine Neuevaluierung der Verbreitung des ECB-Mode und des CBC-Verschlüsselungsalgorithmus mit non-random Initialisierungsvektor ermöglicht einen aktuellen Vergleich dieses sicherheitsrelevanten Bereichs mit den Ergebnissen von Egele et al.

Hypothese 4: Das entwickelte Framework zur Kombination bestehender Analysewerkzeuge und automatisierten Analyse sowie Sicherheitseinstufung von Android-Anwendungen ist durch neue Werkzeuge entsprechend erweiterbar, wodurch sich sowohl der Benutzeroberfläche als auch der skalierenden verteilten Analyseknoten bedient werden kann.

1.4 Methodisches Vorgehen

Zu Beginn der Arbeit wird eine breite wissenschaftliche Literatur-Recherche durchgeführt, um Hintergrundinformation zu sammeln und bestehende Arbeiten im Bereich Android-Anwendungssicherheit zu finden. Dabei ist vor allem interessant, welche Analysewerkzeuge in anderen Arbeiten verwendet wurden, um bestimmte sicherheitskritische Aspekte zu untersuchen, sowie wissenschaftlich auszuwerten, und zu welchen Ergebnissen existierende Arbeiten dabei gekommen sind. Nachdem der aktuelle Stand der Wissenschaft analysiert wurde, wird der Fokus auf spezifische Themenbereiche oder Sicherheitslücken gelegt, um anschließend das Sicherheitsrisiko einer Anwendung zu spezifizieren und zu kategorisieren.

Ausgehend vom aktuellen Stand der Wissenschaft werden die Anforderungen an das Framework zur Einbindung bestehender Analysewerkzeuge und automatisierten Analyse sowie Sicherheitseinstufung von Android-Anwendungen definiert. Eine wichtige Anforderung wird dabei die Skalierbarkeit und Erweiterbarkeit des Frameworks sein, wodurch im nächsten Schritt tragende Design-Entscheidungen getroffen werden.

Sowohl bei Entscheidungen bezüglich Architektur, als auch Design, werden bekannte State-of-the-Art-Technologien und Entwicklungsmuster verwenden, um auf diese Weise in sich abgeschlossene Komponenten zu entwickeln, die wartbar, wiederverwendbar, skalierbar und erweiterbar sind.

Um Android-Anwendungen auf sicherheitskritische Aspekte zu analysieren, existiert eine Vielzahl verschiedener Open-Source-Analysewerkzeuge, die unterschiedliche Sicherheitsbereiche abdecken. In dieser Arbeit werden daher einige, bereits durch die wissenschaftliche Literatur-Recherche bekannte, statische Analysewerkzeuge ausgewählt, die Funktionsweise durch manuelle Tests validiert und die Relevanz für die Arbeit untersucht, um darauf basierend eine Auswahl von Analysewerkzeugen zur Einbindung in das Framework zu treffen.

Diese Komposition aus unterschiedlichen statischen Analysewerkzeugen stellt die Basis des Frameworks zur automatisierten Android-Anwendungsanalyse dar, dessen Fokus bei der Entwicklung auf Erweiterbarkeit durch neue Werkzeuge sowie Skalierbarkeit gelegt wird.

Nach der Entwicklung des Frameworks wird eine Untersuchung der im Google Play Store verfügbaren Anwendungen durchgeführt, um auf diesem Weg vergleichbare und aussagekräftige Analyseergebnisse zu erhalten. Dabei wird wegen der hohen Anzahl an verfügbaren Anwendungen nicht der gesamte Google Play Store analysiert, sondern anhand der verfügbaren Google Play Store-Schnittstellen relevante Anwendungen aus allen Google Play Store-Kategorien ausgewählt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Mit diesen Analyseergebnissen werden anschließend automatisiert Aussagen über die Sicherheit von Android-Anwendungen abgeleitet und Auffälligkeiten in bestimmten Anwendungskategorien aufgezeigt, sowie die Hypothesen belegt. Die automatisierten Sicherheitsevaluierungen der Android-Anwendungen werden stichprobenartig durch händische Auswertung der Werkzeugberichte validiert.

1.5 Aufbau der Arbeit

Zu Beginn der Arbeit werden existierende wissenschaftliche Werke im Bereich der Android-Anwendungssicherheit vorgestellt. Das darauf folgende Kapitel 3 beschäftigt sich mit den Grundbegriffen der IT-Sicherheit und State-of-the-Art Standards der Sicherheitsanalyse. Dem folgt eine Einführung in mobile Anwendungen in Kapitel 4, indem die unterschiedlichen Typen sowie die Struktur und der Aufbau mobiler Anwendungen diskutiert wird. Kapitel 5 beschäftigt sich mit der Architektur und dem Sicherheitskonzept von Android selbst. Dabei werden die fünf Schichten des Android-Systems im Überblick erklärt und wichtig Grundbegriffe sowie Funktionen im Bereich der Sicherheit von mobilen Geräten geklärt. Das darauffolgende Kapitel 6 beschäftigt sich mit der Definition von Sicherheit im Bereich der Android-Anwendungen basierend auf den Empfehlungen und Regeln des OWASP [179] sowie der CMU [22]. Basierend auf diesen werden in Kapitel 7 fünf Sicherheitskategorien und deren Unterkategorien definiert, anhand dieser nachfolgend eine automatisierte Erst-Analyse und Sicherheitseinstufung von Android-Anwendungen durchgeführt wird. Dazu werden in Kapitel 8 existierende State-of-the-Art-Werkzeuge zur Sicherheitsanalyse der definierten Kategorien recherchiert, deren Funktionsumfang analysiert und Erkenntnisse für zukünftige Arbeiten in diesem Bereich dokumentiert. Kapitel 9 beschreibt das in dieser Arbeit zur automatisierten Sicherheitseinstufung von Android-Anwendungen entwickelte Framework sowie die Integration der ausgewählten Werkzeuge. Die Ergebnisse der durchgeführten automatisierten Sicherheitsevaluierung des Frameworks, von 5.800 Google Play Store-Anwendungen aus allen Kategorien, werden in Kapitel 10 präsentiert. Am Ende der Arbeit werden die Hypothesen der Arbeit in Kapitel 11 bekräftigt, gefolgt von einem abschließenden Kapitel 12 zur Zusammenfassung der Erkenntnisse dieser Arbeit.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2 Related Work

Durch die immer rascher ansteigende Menge an Android-Anwendungen und die hohe Relevanz in unserem täglichen Umfeld, wurden in den letzten Jahren viele Arbeiten und Forschungen im Bereich Android Sicherheit durchgeführt, wodurch nicht nur viele unterschiedliche Analysewerkzeuge, sondern auch viele wichtige Erkenntnisse, Theorien und Sicherheitsstandards im Android-Anwendungsbereich entstanden sind. In diesem Abschnitt der Arbeit werden einige Arbeiten und deren Ergebnisse beschrieben, die sich in der Vergangenheit mit dem komplexen und breiten Thema der Android Sicherheit befasst haben.

Bonett et al. [39] untersuchten die Genauigkeit, Grenzen und Funktionalitäten von statischen Analysewerkzeugen, um undokumentierte und unbekannte Grenzbereiche beziehungsweise problematische Design-Entscheidungen und Fehlimplementierungen aufzuzeigen. Analysewerkzeuge, die von Sicherheitsexperten und Expertinnen verwendet und in wissenschaftlichen Arbeiten eingesetzt werden, beinhalten teilweise seitens der Entwicklung undokumentierte und unbekannte Funktionsabgrenzungen, wie die Arbeit zeigt. Dadurch besteht die Gefahr, dass Analyseergebnisse entstehen, auf deren Grundlage falsche Annahmen und wissenschaftliche Aussagen getroffen werden. Darüber hinaus wurden in der Vergangenheit eine Reihe von neuen Analysewerkzeugen entwickelt, die auf Funktionsweisen und Analyseergebnissen bekannter Werkzeuge aufbauen, ohne dabei die Genauigkeit und Aussagekraft der zugrunde liegenden Werkzeuge zu hinterfragen. Um die Grenzen von Analysewerkzeugen aufzuzeigen, wurde im Zuge der Arbeit ein Framework namens Mutation-based Soundness Evaluation (μSE) entwickelt, das gezielt Mutationen von Android-Anwendungen mit Sicherheitslücken erstellt und die Funktionsweise der Analysewerkzeuge anhand dieser Anwendungen validiert (e.g. [172], [163], [77]). Mit dem entwickelten Framework wurden von Bonett et al. sieben häufig zitierte und in der Community bekannte Werkzeuge analysiert. In ihrer Arbeit gehen sie vertiefend vor allem auf die drei Werkzeuge FlowDroid [103], Argus [30] sowie DroidSafe [119] und die bei diesen Werkzeugen entdeckten Funde ein. Sie stellten fest, dass durch eine Analyse mit FlowDroid 48.7% der eingebauten Lücken unentdeckt blieben, durch Argus-SAF sogar 73.1%. Besonders FlowDroid, das in 700 Paper zitiert wurde und auf dem basierend 13 weitere Werkzeuge sowie einige wissenschaftliche Veröffentlichungen erstellt wurden, nennen Bonett et al. als Beispiel für ein Werkzeug, das wichtige undokumentierte Analyse Limitationen beinhaltet. So unterstützt FlowDorid in ihrer Analyse zum Zeitpunkt der Untersuchung durch μSE beispielsweise keine Android Fragmente, die in 90% der untersuchten Google Play Store Top 240 Android-Anwendungen eine tragende Rolle spielen. Im Zuge der Arbeit setzten sich Bonett et al. mit den Entwicklern und Entwicklerinnen von FlowDroid in Verbindung und konnten gemeinsam diese Lücke beheben.

Cai et al. [42] befassten sich mit der Problematik, dass Android-Anwendungsanalysen beziehungsweise Taint Tracking Analysen (e.g. [175], [153], [247]) zeitintensiv sind und selbst unter der Verwendung moderner Analysewerkzeuge oftmals mehrere Stunden in Anspruch nehmen. Sie widmeten sich einer neuen Herangehensweise, die sich mit der Evolution und inkrementellen Veränderung einer Anwendung über mehrere Versionen hinweg befasst und Anwendungen inkrementell analysiert. Bei dieser neuen Analysemethode bauen Analysen von neuen Anwendungsversionen auf den existierenden Analyseergebnissen einer vorherigen Version auf und analysieren lediglich die Veränderungen zwischen den konkreten Versionen. Um zu zeigen, dass ein inkrementelles Analyseverfahren wesentlich effizienter und dennoch korrekt hinsichtlich erkannter Probleme arbeitet, entwickelten Cai et al. im Zuge ihrer Arbeit ein inkrementelles Analyse-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

werkzeug mit dem Namen EvoTaint. EvoTaint extrahiert auf Methodenebene alle Veränderungen des Quellcodes und wendet diese auf eine angebundene Implementierung des Analysewerkzeug FlowDroid an, um anschließend, anhand dieser Analyse durch FlowDroid [103], die Effizienz zu messen. Die Autoren zeigten durch die inkrementelle Analyse von den 19 populärsten Google Play Store-Anwendungen, dass sich die Analysezeit einer veränderten Anwendungsversion, unter der Verwendung des neuen entwickelten Analysealgorithmus, durchschnittlich um 78.6% reduziert. Dadurch erhöht sich die Effizienz der Analyse unter der Verwendung des inkrementellen Analyseansatzes im Vergleich zu einer kompletten Neuanalyse wesentlich. Darüber hinaus erzielte EvoTaint bei allen 19 untersuchten Anwendungen eine gleich hohe Genauigkeit wie die komplette Neuanalyse durch FlowDroid, wodurch die verbesserte Analyseeffizienz ohne Genauigkeitsverlust belegt und ein wesentlicher Grundstein für die inkrementelle Anwendungsanalyse gelegt werden konnte.

Wei et al. [236], [237], [239], [238] beschreiben die aktuellen Weiterentwicklungen des Amandroid Framework zur Datenfluss-Analyse von Android-Anwendungen, auch bekannt unter Argus-SAF. Dabei beschreiben sie, Herausforderungen der statischen Android-Anwendungsanalyse und erklären wie Amandroid diese Herausforderungen bewältigt. Die Autoren gehen vor allem auf die Veränderungen, Verbesserungen und neuen Analyseansätze im Vergleich zu ihrer 2014 veröffentlichten Arbeit [236] beziehungsweise der zu diesem Zeitpunkt existierenden Implementierung von Amandroid ein. Als wesentliche Verbesserung, auch im Vergleich zu anderen Datenfluss Analysewerkzeugen wie FlowDroid [103], nennen sie den neu entwickelten komponentenbasierten Algorithmus zur Analyse von Kommunikationswegen, der auch ICC berücksichtigt. In ihrer Arbeit verifizierten sie diese verbesserte Funktionsweise sowie die Implementierung von Amandroid anhand der Analyse von 4.600 Android-Anwendungen. Dabei fokussieren sie sich vor allem auf Skalierbarkeit und Genauigkeit im Vergleich zu anderen Werkzeugen sowie die Effektivität der ICC Analyse.

Die Arbeit von Huang et al. [137] zeigt, dass Android Malware Entwickler und Entwicklerinnen Plattformen wie VirusTotal [233] dazu verwenden, herauszufinden wie Malware entwickelt werden muss, um von Virenscannern nicht erkannt zu werden. Dabei entwickelten sie ein Framework namens Android Malware Development Hunter (AMD Hunter), mit dem sie anhand der auf VirusTotal hochgeladenen Anwendungen Malware *Submitter* identifizierten. Der entwickelte AMD Hunter besteht aus zwei Phasen, einer *Alert* und einer *Validierungsphase*. Während der Alert Phase verarbeitet die Anwendung in Echtzeit die durch VirusTotal zur Verfügung gestellten Metadaten über die hochgeladenen und untersuchten Anwendungen. Anhand dieser Metadaten erstellten Huang et al. sieben Indizien, um potenzielle Malware Development Fälle zu identifizieren. Weisen die Metadaten einen Verstoß gegen eine dieser Regeln auf, wird die untersuchte Anwendung als potenzieller Malware Development Fall eingestuft, der in der zweiten Phase genauer analysiert wird. Dazu untersucht AMD Hunter die Anwendung selbst mittels statischer und dynamischer Analyse, um anhand der Analyse die zu den jeweiligen Indizien definierten Regeln zu untersuchen. Weist eine Anwendung durch die Framework eigene Analyse einen Regelverstoß auf, konnte ein Fall von Malware Development aufgedeckt werden. Auf diese Weise konnten, durch eine viermonatige Evaluierungsphase des Frameworks, 1.623 Android Malware Development Fälle von 153 Millionen auf VirusTotal hochgeladenen Anwendungen aufgedeckt und die Hypothese bestätigt werden, dass Malware Entwickler und Entwicklerinnen Webservice wie VirusTotal verwenden, um für Antiviren-Software unsichtbare, schadhafte Anwendungen zu erstellen.

Chester et al. [59] sowie Felt et al. [96] beschäftigten sich mit dem Android Berechtigungskonzept und analysierten das Vorkommen von Über- und Unterprivilegierung bei Android-Anwendungen. Wie sie in ihrer Arbeit erklären, führt eine Überprivilegierung verständlicherweise zu einem unnötigen Sicherheitsrisiko, während sich die Unterprivilegierung durch eine möglicherweise unstabile Anwendung bemerkbar macht. Selbst in aktuellen Android-Versionen existiert keine Möglichkeit

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

festzustellen, ob Anwendungen tatsächlich alle definierten und angeforderten Berechtigungen für ihre Funktionsausübung benötigen. Daher entwickelten Chester et al. das leichtgewichtige Werkzeug M-Perm [121], um eine wissenschaftliche Evaluierung des Auftretens von Über- und Unterprivilegierung in bekannten Android-Anwendungen nachzuweisen. Dabei ließen sie sich von ähnlichen bereits bestehenden aber veralteten Werkzeugen wie das Werkzeug Stowaway [223] von Felt et al. [96], PScout [123] und Spark [157] inspirieren, deren Android Versionskompatibilität spätestens bei Version 5.1.1 endet. M-Perm wurde daher mit speziellen Fokus auf die Kompatibilität von neueren Android Versionen, speziell ab Android 6, und der damit einhergehenden Änderung am Berechtigungskonzept entwickelt. Innerhalb ihrer Arbeit untersuchten die Autoren 50 bekannte und weit verbreitete Android-Anwendungen wie Uber, Skype, LinkedIn, Twitter und Google Calendar. Dabei konnten sie feststellen, dass von den 50 untersuchten Anwendungen insgesamt 1.119 Berechtigungen angefordert wurden und sich darunter 110 Über- sowie 89 Unterprivilegierungen befanden.

Weitere Veröffentlichungen im Bereich des Android Berechtigungskonzept wie Wei et al. [241] und Colton et al. [66], aus der ein Projekt namens P-Lint [182] hervorgeht, beschäftigen sich mit der Evolution des Android Berechtigungssystems. Die Autoren von P-Lint beschäftigten sich mit Problemen der Implementierung von Anwendungen im Zusammenhang mit Berechtigungsanfragen. Wie in Abschnitt 5.2.1 beschrieben, existiert seit Android 6.0 ein neues Berechtigungsmodel, das es der Anwendung ermöglicht, zur Laufzeit Berechtigungen anzufordern. Diese Funktionalität stellt bei falscher Implementierung ein Problem dar, das sich auf die Stabilität einer Anwendung auswirken kann. Vor Android 6.0 konnten Entwickler und Entwicklerinnen davon ausgehen, dass eine Anwendung alle benötigten Berechtigungen besitzt, da eine Verweigerung von Berechtigungen weder bei der Installation, noch zur Laufzeit möglich war, wodurch zur Laufzeit keine Berechtigungsprüfung mehr erfolgen musste. Ab Android 6.0 ist dies aber anders. Daher muss vor jedem Zugriff auf eine Funktion, die spezielle Berechtigungen benötigt, eine Überprüfung der erteilten Berechtigungen erfolgen. Sind die benötigten Berechtigungen nicht vorhanden und ein Zugriff passiert ohne diese Prüfung, kommt es zu einem Laufzeitfehler der Anwendung, der die Stabilität der Anwendung beeinflusst, wie Colton et al. beschreiben. P-Lint untersucht die korrekte Implementierung dieser Funktionalität sowie weitere Probleme im Zusammenhang mit der Berechtigungsdefinition, Prüfung und Implementierung. Um das Ausmaß der Problematik darzustellen, wurden im Zuge der wissenschaftlichen Veröffentlichung zu P-Lint, 40 Android-Anwendungen hinsichtlich des korrekten Umgangs mit Berechtigungen untersucht, wobei insgesamt 79 Best Practice Abweichungen festgestellt wurden, die möglicherweise zu Problemen der Anwendungsfunktionalität führen können. Die Autoren erwähnen allerdings nicht, nach welchen Kriterien die untersuchten Anwendungen ausgewählt wurden.

Yu et al. [249] untersuchten in ihrer Arbeit die Privatsphäre und Datenschutzrichtlinien von Android-Anwendungen und entwickelten eine Software namens PPChecker, die Datenschutzbestimmungen einer Anwendung mit dem tatsächlichen Verhalten vergleicht und Abweichungen aufdeckt. Das Werkzeug unterteilt sich in drei Teilbereiche, die sich durch die automatisierte Verarbeitung von textuellen Datenschutzrichtlinien, eine statische Analyse der eigentlichen Anwendungsfunktionalität und dem Erkennen von Privatsphäre Problemen oder Unstimmigkeiten im Vergleich zu den vorhandenen Datenschutzrichtlinien einer Anwendung beschreiben lassen. Um Datenschutzrichtlinien einer Anwendung automatisiert zu verarbeiten und für die weitere Analyse aufzubereiten, entwickelten Yu et al. einen eigenen Algorithmus, der die entsprechenden Textpassagen auswertet und bestimmten Pattern zuordnet. Liegen diese Pattern sowie das Ergebnis der statischen Anwendungsanalyse vor, bei der PPChecker auf der Funktionalität des Analysewerkzeugs VulHunter [193] aufbaut, können diese im dritten und letzten Analyseteil mit dem tatsächlichen Verhalten einer Anwendung abgeglichen werden. Dazu wurden innerhalb der Arbeit ebenso fünf Algorithmen zum Erkennen solcher Abweichungen entwickelt. Durch dieses neue Analysewerkzeug konnte, in einer Evaluierungsphase, ein völlig neuer Bereich der Android-Anwendungssicherheit und Privat-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

sphäre analysiert werden, wodurch die Autoren in rund 24% der 1.197 untersuchten Anwendungen Unstimmigkeiten in diesem Bereich entdecken konnten.

Rojas et al. [201] beschäftigten sich mit dem stetigen Wandel der mobilen Geräte sowohl auf Hardware als auch auf Software beziehungsweise Betriebssystem Ebene und den damit verbundenen Qualitäts- sowie Kompatibilitätsproblemen. Durch die immer kürzeren Entwicklungszyklen neuer Betriebssystemversionen und dem immer rascheren Erscheinen von neuen mobilen Endgeräten, kommt meist nur die lokale, eingeschränkte Test-Infrastruktur von Software-Entwicklungsteams zum Einsatz, wodurch viele Hardware und System Kombinationen vor dem Erscheinen neuer oder veränderter Anwendungen meist ungetestet bleiben. Aus diesem Grund widmeten sich die Autoren in ihrer Arbeit der Entwicklung eines Test Frameworks namens Automated Mobile Testing as a Service (AM-TaaS) [4]. Dabei handelt es sich um ein Cloud-basiertes Test Framework für Android-Anwendungen, dass es dem Nutzer ermöglicht, gewählte Anwendungen auf verschiedenen Emulatoren, also Endgeräten und Betriebssystemen, zu testen. Dies ist gerade bei raschen sich wesentlich voneinander unterscheidenden Entwicklungen im Bereich mobiler Geräten besonders wichtig, da es durch verschiedene Hardware und Softwarelösungen oftmals zu unerwarteten Kompatibilitätsproblemen kommt. Das Framework ermöglicht es Benutzern und Benutzerinnen ausgewählte Anwendungen mit einer Vielzahl unterschiedlich konfigurierter Emulatoren automatisiert zu testen. Dazu bauten die Autoren eine Reihe von automatisierten Testwerkzeugen wie Monkeyrunner [171], AndroidViewClient [24] und UIAutomator [232] ein, um auf diese Weise automatisierte Testdurchläufe zu starten und darüber hinaus die, durch den Benutzer oder die Benutzerin zur Verfügung gestellten, Testskripts einzubinden.

Egele et al. [85] befassten sich mit der automatischen Analyse von Google Play Store-Anwendungen, um Probleme in Bezug auf die Verwendung von Kryptografie Bibliotheken aufzudecken, die von vielen Analysewerkzeugen außen vor gelassen wurden. Dadurch legte die Arbeit von Egele et al. in der Vergangenheit einen großen und wichtigen Stein im Bereich Android-Sicherheitsanalyse. Um Missstände im Bereich Kryptografie aufzudecken, entwickelten sie CryptoLint, ein Werkzeug das kryptografische Vorgänge untersucht und Datenflüsse von Schlüsseln sowie IV aufzeigt. Dabei wurden vor allem sechs Sicherheitsrisiken und Grundregeln im Umgang mit Kryptografie überprüft. Darunter beispielsweise die Verwendung von ECB oder CBC Verschlüsselung mit nicht zufälligen IV sowie konstanten Schlüssel und Salts. In einer Evaluierungsphase ihres neuen Werkzeugs zeigte sich, dass 88% der 11.748 untersuchten Android-Anwendungen gegen mindestens eine der sechs Grundregel zur Kryptografie verstoßen und somit ein Sicherheitsrisiko darstellen.

Rastogi et al. [194] beschäftigten sich, anders als wir, mit der dynamischen Anwendungsanalyse und entwickelten ein Framework mit dem Namen AppsPlayground zur automatischen dynamischen Analyse von Android-Anwendungen in Bezug auf unerwünschte, schadhafte Funktionalität und Privatsphäre Probleme. Dabei widmeten sie sich unter anderem dem Problem der sinnvollen, automatischen und dynamischen Analyse und der damit verbundenen nötigen Anwendungsausführung. Dies stellt gerade bei Android-Anwendungen, die anders als gewöhnliche Anwendungen nicht nur einen eindeutigen Einstiegspunkt, sondern viele unterschiedliche Einstiegspunkte haben, eine besondere Schwierigkeit dar. Um das Verhalten von Anwendungen durch die unterschiedlichsten Einstiegspunkte dynamisch zu analysieren, verwendet AppsPlayground eine Reihe von automatisierten Testdurchläufen, um mithilfe von Fuzzing [169], intelligenten Anwendungsausführungen und Event Trigger Algorithmen möglichst viele Einstiegspunkte zu testen. AppsPlayground versucht dabei durch API und Kernel Level Aufzeichnungen Sicherheitsprobleme aufzudecken sowie zu verfolgen und integriert dazu einige bereits bestehende Sicherheitsanalysewerkzeuge. In einer durchgeführten Evaluierungsphase von 3.968 Android-Anwendungen, deckten Rastogi et al. in fast 24% oder genauer in 946 Anwendungen Sicherheits- beziehungsweise Datenschutzprobleme auf, durch die Informationen wie Standort und Identität IDs ins Internet gesendet werden. Durch eine genauere Analyse stellten sie fest, dass diese Datenflüsse unter anderem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

durch eingebundene Werbung oder Analysebibliotheken verursacht wurden, die Informationen an bestimmte Domains zur Werbe- und Entwicklungsforschung sendeten.

Eine weitere interessante Arbeit ist von Taylor et al. [226], in der die Sicherheit von Android-Anwendungen aus dem Finanzbereich Thema ist. Dabei wurde die Veränderung der Sicherheit zwischen 2014 und 2016, sowie die Finanz Kategorie mit allen anderen Anwendungskategorien verglichen. Dazu untersuchten sie mithilfe von AndroBugs [111] und MobSF [170] die durch OWASP definierten Sicherheitsrichtlinien. Sie analysierten 200 Anwendungen aus dem Finanzbereich sowie im Vergleich dazu deren 2 Jahre alte Version. Diesen Ergebnissen stellten sie das Ergebnis der Analyse von 5.000 Anwendungen aus anderen Google Play Store-Kategorien sowie ebenso deren 2 Jahre alten Vorgängerversion gegenüber. In Summe untersuchten sie also 10.400 Anwendungen und stellten fest, dass Anwendungen aus dem Finanzbereich im Durchschnitt zwar weniger kritische Sicherheitslücken aufweisen als Anwendungen anderer Bereiche, der durchschnittliche Anstieg der Sicherheitslücken von Finanzanwendungen im untersuchten Zeitraum von zwei Jahren allerdings etwa dreimal so hoch ist als in den anderen untersuchten Anwendungskategorien. Die vier häufigsten entdeckten Sicherheitsrisiken waren dabei unabhängig von der Anwendungskategorie, das Erstellen von global les- und schreibbaren Dateien, die Verwendung von unsicheren Android `ContentProvider`, die Erstellung von unsicheren zufälligen Kryptografie Schlüsseln und die Verwendung impliziter Intents. In ihrer anschließenden Diskussion erklären Taylor et al., dass Entwickler und Entwicklerinnen von mobilen Anwendungen unbedingt Kenntnis der OWASP Top 10 haben müssen und Anwendungen auf jeden Fall durch diverse verfügbare Analysewerkzeuge analysiert werden sollten, bevor diese veröffentlicht werden. Sie erwähnen auch, dass solche Analysen durch den Google Play Store selbst sinnvoll wären, um die Qualität von mobilen Anwendungen weiter zu erhöhen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3 Grundlagen der IT-Sicherheit und Sicherheitsanalyse

Im diesem Abschnitt der Arbeit werden wichtige Begriffe und Definitionen der Informations- beziehungsweise IT-Sicherheit sowie die Grundpfeiler dieser beschrieben. Dabei liegt der Fokus auf den weitverbreiteten Erklärungen des NIST [176]. Im Anschluss daran wird eine Einführung in die Software-Analysetechniken gegeben. Dazu werden *statische, dynamische* und *hybride Analyseverfahren* sowie die Herausforderungen, welche die Analyse mobiler Anwendungen mit sich bringen, diskutiert. Abschließend beschäftigen wir uns damit, wie komplexe Analysetechniken skalierbar und nutzbar zur Verfügung gestellt werden können.

3.1 Einführung in die IT-Sicherheit

Bei Informationen im IT-Bereich handelt es sich um Datensätze jeder Art sowie Wissen in unterschiedlichen Formen, die zwischen Systemen ausgetauscht werden können. Die Informationssicherheit befasst sich daher mit dem Schutz von Daten und Wissen, also Informationen, sowie dem Schutz von Informationssystemen. Sie beinhaltet zwei wichtige und zusammengehörende Bereiche, die sich in Datenschutz und Datensicherheit aufteilen, wie Deshpande et al. [76] erklären.

> ..., data security is the door keeper of the fortress of data privacy. [76]

> ..., enforcing data privacy acts as an enabler for ensuring data security of an application. [76]

> Data security is required to safeguard data privacy. Data privacy is necessary to ensure data security. [76]

Wenn die Datensicherheit eines Systems angreifbar oder schwach ist, dann sind private und schutzbedürftige Informationen, die sich in diesem System befinden, nicht mehr sicher, wodurch der Datenschutz nicht gewährleistet werden kann. Umgekehrt, wenn der Datenschutz nicht eingehalten wird, ist keine Datensicherheit möglich.

Die drei Begriffe Vertraulichkeit *(confidentiality)*, Integrität *(integrity)* und Verfügbarkeit *(availability)*, bilden die Grundpfeiler der IT-Sicherheit und sind auch unter dem in Abbildung 3.1 dargestellten CIA Triad bekannt.

Die **Vertraulichkeit** befasst sich mit dem Schutz vor unautorisierten Zugriffen auf Informationen. Informationen beziehungsweise Daten dürfen nur für Befugte zugänglich sein, um die Vertraulichkeit zu gewährleisten. Vertraulichkeit schließt auch Datenschutz mit ein, bei dem ausschließlich betroffene Entitäten festlegen können, wer und welches System Zugriff auf welche Informationen erhält. [93]

Bei **Integrität** wird zwischen Daten- und Systemintegrität unterschieden [176]. Datenintegrität bedeutet, dass Informationen vor unbefugten Veränderungen und externen Einflüssen

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

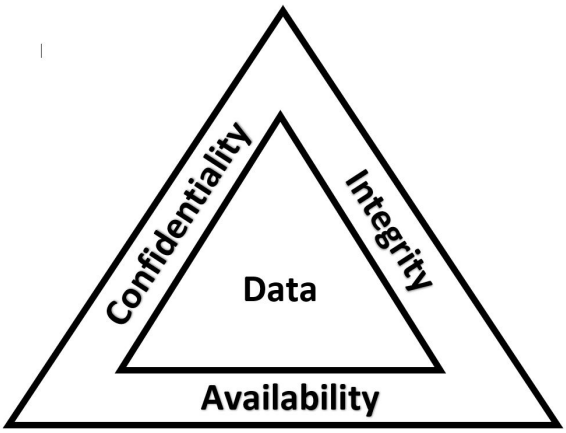


Abbildung 3.1: CIA Triad [93]

geschützt sind. Dieser Bereich der Integrität befasst sich besonders mit dem Speichern, Bearbeiten und Transport von Informationen. Systemintegrität bedeutet, dass ein System oder eine Anwendung eine konkrete Funktion korrekt und autorisiert durchführt sowie Informationen vor unbefugten Zugriffen und Manipulationen geschützt sind.

Die **Verfügbarkeit** ist eine weitere wichtige Eigenschaft von IT-Sicherheit, die beinhaltet, dass Informationen und Ressourcen eines Systems jederzeit zur Verfügung stehen [176].

Gefahren und Risiken für die Erhaltung der Vertraulichkeit, Integrität und Verfügbarkeit, und somit für die IT-Sicherheit, sind Schwächen (*Vulnerabilities*), die von Bedrohungen (*Threats*) ausgenutzt werden können.

Bei Threats unterscheidet man zwischen feindlichen Gefährdungen, also beispielsweise Personen, Gruppen oder Entitäten, die bewusst Angriffe auf die IT-Sicherheit durchführen, und nicht feindlichen Gefährdungen wie etwa Aktionen, die unbewusst zu Risiken der IT-Sicherheit führen [176].

Um die IT-Sicherheit beziehungsweise die Einhaltung der CIA Triad Schutzziele zu gewähren, ist es erforderlich Vulnerabilities zu vermeiden und möglichen Threats entgegenzuwirken. Dazu existieren je nach Bereich und Threat verschiedene Maßnahmen wie etwa Datenverschlüsselung, Zwei-Faktor-Authentifizierung, biometrische Authentifizierung, Checksummen, *Denial of Service* Verhinderungsmechanismen und redundante Systeme. Es existieren auch verschiedene Projekte und Regeln von Organisationen, um IT-Sicherheit zu gewähren. Kapitel 6 beschäftigt sich etwa mit dem OWASP [179] sowie Regeln der CMU [22] zur Android-Anwendungsentwicklung, welche die Sicherheit von mobilen Anwendungen verbessern.

Gerade in Zeiten in denen tragbare Endgeräte Teil des Alltags sind und mobile Anwendungen eine tragende Rolle spielen, ist die Informationssicherheit von diesen Geräten und Anwendungen, nicht zuletzt wegen des großen Funktionsumfangs und dem damit verbundenen Eingriff in private Bereiche, besonders relevant. Anwendungen, die Sicherheitslücken beinhalten, stellen ein großes Risiko für Datensicherheit, Datenschutz und Systemsicherheit dar und müssen daher vermieden werden. Welche Möglichkeiten existieren, um die Sicherheit von bestehenden Anwendungen zu überprüfen, bringt das nächste Kapitel näher.

3.2 Unterschiedliche Software-Analyseverfahren

Die Funktionsanalyse von Anwendungen spielt in allen Bereichen der Anwendungsentwicklung eine große Rolle. Dabei existieren verschiedene Ansätze (e.g. [159], [84], [211]), um detaillier-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

te Informationen über die Funktionsweise einer Implementierung zu sammeln. Im Wesentlichen lassen sich diese Ansätze in statische, dynamisch und hybride Analyseverfahren gliedern, die sich wesentlich voneinander unterscheiden. In diesem Abschnitt wird eine kurze Einführung in die Funktionsweise sowie Vor- und Nachteile der unterschiedlichen Herangehensweisen gegeben und vor allem auf Android-spezifische Probleme eingegangen.

3.2.1 Statische Analyseverfahren

Bei statischen Analyseverfahren (e.g. [67], [155], [58]) werden Anwendungen nach verschiedenen Ansätzen untersucht, ohne selbst ausgeführt zu werden. Dabei wird durch manuelle oder automatisierte Analyse versucht, möglichst viele Informationen aus dem vorhandenen Quellcode, Dokumentationen sowie *Binaries* zu erhalten.

Der Hauptvorteil gegenüber der dynamischen Analyse ist die Untersuchung der gesamten Funktionalität sowie aller Bereiche einer Anwendung, während bei dynamischer Analyse jeweils nur der aktuell ausgeführte Teil analysiert wird, wie Payet et al. [183] und Egele et al. [84] beschreiben.

Eine automatisierte, statische Analyse beginnt üblicherweise damit, die zu untersuchende Quelle in einem abstrakten Model, wie etwa Kontrollfluss, Datenfluss oder UML Diagrammen, darzustellen, um anschließend Informationen extrahieren und unterschiedliche Bereiche analysieren zu können, wie Li et al. [159] beschreiben. Darüber hinaus nennen sie einige Herausforderungen statischer Codeanalyse, auf die an dieser Stelle ein Blick geworfen wird.

Herausforderungen statischer Analyseverfahren in Bezug auf Android-Anwendungen

Da Android-Anwendungen hauptsächlich in Java geschrieben werden, beinhaltet die statische Analyse neben Android-spezifischen Analyse Herausforderungen auch Herausforderungen und Probleme, die aus der statischen Analyse von Java-Anwendungen bekannt sind. Zu den wichtigsten Problemen zählen dabei Reflexion und nativer Quellcode.

Reflexion
: Unter Reflexion [187] wird das dynamische Verändern und Nachladen von Codeteilen oder Ressourcen zur Laufzeit verstanden. Diese Funktionalität erschwert die statische Analyse von Codeteilen, da der vollständige zu analysierende Code nicht zur Analysezeit bekannt ist.

Native Codeteile
: Native Codeteile werden oftmals direkt in Form von kompilierten Binaries eingesetzt, wodurch entsprechende Codeteile zum Analysezeitpunkt nicht vorhanden sind und eine automatisierte statische Analyse erschwert wird.

Zusätzlich zu den Java-spezifischen Problematiken existieren Android-spezifische Herausforderungen, welche statische Codeanalysen beeinflussen [159].

Bibliotheken
: Android-Anwendungen sind vollständige, alleinstehende Anwendungen, die neben dem eigentlichen Anwendungscode auch verwendete Drittanbieter-Bibliotheken beinhalten. Werden bei der statischen Analyse Bibliotheken berücksichtigt, steigt die Anzahl der zu analysierenden Codezeilen enorm, da verfahrensabhängig auch nicht verwendete Codeteile einer Bibliothek analysiert werden. Dies erhöht nicht nur die Analysezeit, sondern führt unter Umständen zu falschen Analyseergebnissen, da Bibliotheken möglicherweise unverwendete Funktionalitäten enthalten können, die für den konkreten Anwendungsfall einer Anwendung schadhafte Eigenschaften bedeuten könnten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Dalvik Executables

Android-Anwendungen werden nach der Entwicklung in Java, in *Dalvik Executables (DEX)* [72] kompiliert, wie in Kapitel 5.1.3 anhand der Abbildung 5.3 erklärt. Statische Analysewerkzeuge müssen daher dieses Format unterstützen oder Anwendungen vor der Analyse in ein unterstütztes Format konvertieren, wodurch zusätzlicher Aufwand anfällt.

Programm Einstiegspunkte

Android-Anwendungen besitzen, im Vergleich zu Anwendungen anderer Programmiersprachen, keinen eindeutigen Einstiegspunkt. Stattdessen verfügen diese über mehrere verschiedene Einstiegspunkte, die vom Android-Anwendungsframework, auf das in Abschnitt 5.1.4 eingegangen wird, zur Laufzeit aufgerufen werden können. Daher existiert kein eindeutiger Programmablauf, wodurch das Erstellen von Kontrollflüssen, Datenflüssen und Diagrammen zur statischen Analyse erheblich erschwert wird.

Komponenten Lebenszyklus

Einzelne Komponenten einer Android-Anwendung, auf die in Abschnitt 4.2.2 eingegangen wird, besitzen ihren eigenen Lebenszyklus und können direkt vom Android-System gestartet, gestoppt oder angehalten werden. Dadurch haben diese Komponenten einen eigenen unabhängigen Kontrollfluss und stellen ebenfalls, speziell für statische Kontrollflussanalysen, eine Herausforderung dar.

Events

Benutzer- oder Systemevents stellen ebenso Einstiegspunkte und Veränderungen des Ablaufs dar, wodurch der Analyseaufwand weiter erhöht wird. Zusätzliche mögliche Einstiege und Kombinationen von Abläufen müssen analysiert werden, um die vollständige Funktionsweise einer Anwendung abzudecken.

Inter-Component Communication

Durch die *Inter-Component Communication (ICC)* bietet Android die Möglichkeit der Kommunikation und des Datenaustausches zwischen Komponenten, wie Abschnitt 4.2.3 erklärt. Diese Kommunikation stellt eine große Erschwernis für statische Analysen dar, da es automatisiert kaum möglich ist, zu ermitteln welche Komponenten wie und zu welchem Zeitpunkt kommunizieren können.

Aus diesen Gründen kommen bei statischen Analysen von Android-Anwendung neben Kontrollfluss- [3] und Datenfluss-Analysen [209] häufig vor allem regelbasierte Volltextanalysen zum Einsatz, mit denen nach bestimmten bekannten unerwünschten Funktionen oder fehlerhaften Konfigurationen gesucht wird. Dies hat den Vorteil, dass der Quellcode direkt anhand dieser Regeln analysiert werden kann, ohne zuvor abstrakte Modelle zu generieren.

Bei der statischen Analyse von Programmcode kommt statisches *Program Slicing* [242] häufig zum Einsatz, um nicht den gesamten Quellcode zu analysieren, sondern um sich auf bestimmte ausgewählte Bereiche zu fokussieren, die möglicherweise auf einen gewählten Teilbereich einer Software Einfluss nehmen. Mit dieser Herangehensweise ist es beispielsweise möglich, nur Befehle, Funktionen oder Statements zu analysieren, die eine bestimmte, definierte Variable verändert [159].

3.2.2 Dynamische Analyseverfahren

Anders als bei der statischen, wird bei dynamischen Analysen (e.g. [205], [2], [84]), bei der die inneren Strukturen einer Anwendung nicht bekannt sind, die Funktionsweise einer Anwendung zur Laufzeit in einer abgesicherten, simulierten, virtuellen Umgebung untersucht. Dabei werden immer nur die aktuell ausgeführten Aktionen oder genauer gesagt Programmabläufe untersucht,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

wodurch sich die dynamische Analyse meist auf konkrete Funktionsabläufe oder Anwendungsszenarien bezieht und nicht den gesamten Funktionsumfang abdeckt. Daraus ergibt sich ein Vorteil gegenüber den Android-spezifischen Herausforderungen des statischen Analysevorgangs. Eine Herausforderung ist bei dieser Herangehensweise allerdings das Erreichen einer hohen Codeabdeckung, da ein erheblicher Aufwand nötig ist, um sämtliche Funktionalitäten durch möglichst realistische Testfälle abzudecken.

Aus diesem Grund bauen dynamische Analysen meist auf der Untersuchung von Interaktionen durch Programmschnittstellen, sogenannten *API-Schnittstellen*, Systemaufrufen, Datei und Netzwerk Zugriffen, Methodenaufrufen, Bibliotheken Verwendungen oder dergleichen auf, wodurch ein Verhaltensmodel von verschiedenen Anwendungsszenarien einer Anwendung generiert wird. Basierend auf diesem Modell können anschließend durch eine dynamische Analyse unerwünschte Verhaltensmuster identifiziert werden. Darüber hinaus lassen sich Veränderungen innerhalb der simulierten Analyseumgebung feststellen, die durch schadhafte Funktionsweisen einer Anwendung ausgelöst werden können.

Bei der Ausführung von dynamischen Analysevorgängen *(Dynmaic Concret Execution)* existieren Ansätze [211], die sowohl auf *Fuzzing* [169], als auch *Symbolic Execution* [152] basieren. Beim Fuzzing werden bestimmte, einzelne Programmabläufe basierend auf mutierenden Eingaben analysiert, während es sich bei Symbolic Execution um eine abstrakte Ausführung handelt, bei der symbolische Werte für mögliche Eingaben basierend auf den unterschiedlichen Programmabläufen erstellt werden. Wird eine konditionelle Verzweigung des Programmablaufs festgestellt, werden alle Pfade weiterverfolgt. Dadurch entsteht ein genauer semantischer Einblick in die Funktionsweise einer Anwendung. Eine wesentliche Herausforderung von Symbolic Execution ist die Skalierbarkeit bei Anwendungen mit vielen verschiedenen konditionellen Verzweigungen.

Um den Abschnitt des dynamischen Analyseverfahrens abzuschließen, wird an dieser Stelle noch auf vier weit verbreitete, unterschiedliche dynamische Analyseansätze und Techniken eingegangen [84].

Aufzeichnung von Funktionsaufrufen

Hierbei wird versucht, anhand von Funktionsaufrufen einen abstrakten Überblick über das Verhalten einer Anwendung zu bekommen. Dazu werden unterschiedliche Analysemethoden, sogenannte *Hooks*, vor oder nach Funktionsaufrufen eingehängt. Auf diese Weise lassen sich beispielsweise System- oder API-Aufrufe festhalten, wodurch Aussagen über die Funktionalität einer Anwendung getroffen und unerwünschte Systemzugriffe aufgezeigt werden können.

Analyse von Funktionsparametern

Bei der Untersuchung von Funktionsparametern stehen die konkreten Werte und Rückgabewerte eines Funktionsaufrufs im Vordergrund, wodurch sich Verbindungen zwischen Funktionalitäten aufzeigen und Funktionsaufrufe gruppieren lassen, um weitere Einsicht in die Funktionsweise einer Anwendung zu bekommen.

Informationsfluss Überwachung

Bei der Analyse des Informationsflusses wird die Datenverarbeitung innerhalb einer Anwendung analysiert, um unerwünschten Informationsfluss aufzuzeigen. Dazu werden ausgewählte Daten mit einem Label, sogenannten *Sources*, gekennzeichnet. Wird ein gekennzeichnetes Objekt verändert oder gelangt dieses in Kontakt mit zuvor definierten Komponenten, auch bekannt unter *Sinks*, bemerkt der dynamische Analyseprozess die Datenflüsse beziehungsweise Veränderungen und kann diese verarbeiten.

Analyse von Befehlsabläufen

Die Analyse von Befehlsabläufen auf Maschinenbefehlsebene ermöglicht es, das Verhalten

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

einer Anwendung oder Teilfunktionalität im Detail zu betrachten. Auf diese Weise können unerwünschte Verhaltensmuster auf niedrigen Abstraktionslevel erkannt werden.

3.2.3 Hybride Analyseverfahren

Sowohl statische, als auch dynamische Analyseverfahren besitzen unterschiedliche Stärken und Schwächen, wie die beiden Teilbereichen zuvor zeigen. Während statischen Analysen den gesamten Quellcode einer Anwendung analysieren, was speziell bei Android-Anwendungen, die unterschiedliche Einstiegspunkte, Funktionsabläufe und unverwendete Codeteile in Form von Bibliotheken beinhalten, eine Herausforderung darstellt, werden bei dynamischen Analysen ausschließlich direkt ausgeführte Codeteile analysiert. Dies hat zum einen den Vorteil, dass nur konkrete Verhaltensmuster analysiert und auch Systemveränderungen festgestellt werden können, zum anderen den großen Nachteil, dass hohe Codeabdeckung meist nur schwer erzielt werden kann. Des Weiteren kommt es häufig vor, dass Malware dynamische Analysen erkennt und somit die eigentliche schadhafte Funktionalität vor der Analyse verbirgt, wie Tam et al. [224] erklären.

Daher bietet sich der gemeinsame Einsatz von dynamischer und statischer Analyse an, um durch die Stärken beider Analysetechniken zu profitieren. Werden sowohl statische, als auch dynamische Techniken zur Identifikation unerwünschter Funktionalität oder Sicherheitslücken verwendet, wird von hybriden Analyseverfahren gesprochen. Bei diesen werden meist zuerst statische Analysetechniken verwendet, um verdächtige Codeteile und Pfade zu finden und diese anschließend mit dynamischer Analyse zu untersuchen.

Ein Beispiel für ein hybrides Analyseverfahren ist die Kombination von statischen Program Slicing und Symbolic Execution, bei welcher der Fokus auf bestimmte Anwendungssegmente gelegt wird. So wird Problemen wie dem sogenannten *Path Explosion* Problem (exponentiell steigender Analyseaufwand bei konditionellen Verzweigung des Programmablaufs), nicht vollständig aufgeschlüsselte Programmabläufe oder fehlenden Programmkontext entgegengewirkt. Angr [211] ist ein konkretes Beispiel für ein hybrides Analyse-Framework, das diese beiden State-of-the-Art-Analyseverfahren integriert, um von den jeweiligen Stärken zu profitieren.

Der weitere Verlauf dieser Arbeit beschäftigt sich ausschließlich mit der automatisierten, statischen Analyse von Android-Anwendungen.

3.3 Einsatz von Cloud-basierten Services

Die Analyse von Anwendungen ist meist ein komplexes, aufwendiges und vor allem zeitintensives Unterfangen. Gerade bei mobilen Endgeräten, die allgegenwärtig sind und unzählige unterschiedliche Anwendungen besitzen, ist es besonders wichtig, möglichst viel über das Verhalten installierter Anwendungen zu wissen. Zur Verhaltensanalyse und Sicherheitseinstufung von Android-Anwendungen ist zwar eine große Anzahl an Analysewerkzeuge vorhanden (siehe 8.2), jedoch ist es meist für den durchschnitten Nutzer oder die durchschnittliche Nutzerin aufgrund hoher Komplexität der Werkzeuge trotzdem nicht möglich diese einzusetzen.

Ein Ansatz, diese Analysewerkzeuge zugänglich zu machen, ist die Kombination dieser innerhalb eines Frameworks, das in Form einer Webanwendung zur Verfügung gestellt wird. Diese Art von Servicemodel hat sich in den letzten Jahren stark verbreitet und ist unter dem Begriff *Software as a Service (SaaS)* beziehungsweise *Mobile Testing as a Service (Mobile TaaS)* bekannt, wie Rojas et al. [202] sowie Gupata [132] beschreiben.

SaaS sowie TaaS verfolgen grundsätzlich zwei Gedankengänge. Zum einen werden komplexe Software einfach zugänglich über ein Webservice, zum anderen Skalierbarkeit und Elastizität durch diverse Cloud Lösungen angeboten. Beides ist für den Bereich dieser Arbeit besonders

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

wichtig, da die Analyse von Anwendungen und speziell von mobilen Android-Anwendungen, sowohl Zeit, als auch Ressourcen intensiv ist und Analysetools meist ein komplexes Setup und Hintergrundwissen benötigen. Einfache Zugänglichkeit, Skalierbarkeit und Ressourcenelastizität ist daher besonders wichtig, um die Einstiegshürde für Nutzer und Nutzerinnen möglichst gering zu halten und die Stabilität des Services zu gewähren [202], wie auch Mahmood et al. [165] feststellen.

Daher bieten sich *Cloud-Computing* Lösungen an, um möglichst flexibel auf Hard- und Softwareressourcen auf Abruf (*on-demand*) zugreifen zu können. Bei dieser Art von Ressourcen Teilung durch Cloud Lösungen haben sich in den letzten Jahren vor allem folgende Servicemodelle etabliert.

Infrastructure as a Service (IaaS)
Infrastruktur wie Prozessorleistung, Arbeitsspeicher oder Festplattenspeicher wird bei IaaS on-demand angefordert und Nutzer sowie Nutzerinnen kaufen auf diesem Weg on-demand Infrastruktur Leistungen.

Platform as a Service (PaaS)
PaaS beschreibt eine zur Verfügung gestellt Umgebung, innerhalb dieser zum Beispiel Software entwickelt oder deployed werden kann. Die als Service genutzte Plattform baut auf der Infrastruktur darunter auf.

Software as a Service (SaaS)
Wenig überraschend wird beim SaaS Servicemodell Software als Service zur Verfügung gestellt, sodass Nutzer oder Nutzerinnen diese beispielsweise in Form von Webservice benutzen können.

Testing as a Service (TaaS)
Das TaaS Servicemodell stellt eine erweiterte Form von SaaS dar.

> TaaS is automated software testing as a cloud-based service. [202]

TaaS baut auf den drei zuvor genannten Services auf und wird dazu verwendet, sowohl Software als auch Plattform und Infrastruktur zu testen. Die Spezialisierung auf mobile Anwendungen, ist unter dem Begriff *Mobile TaaS* bekannt [202]. Bei dieser Art von Servicemodell wird eine Testsoftware oder ein Framework zur Analyse von mobilen Anwendungen als Service zur Verfügung gestellt und wird auf diese Weise nicht nur einfach zugänglich und nutzbar, sondern profitiert auch von den Vorteilen der darunter liegenden Servicemodelle.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

4 Einführung in mobile Anwendungen

Zu Beginn dieses Kapitels wird allgemein auf die unterschiedlichen Typen mobiler Anwendungen sowie deren Eigenschaften eingegangen, und anschließend die interne Struktur von Android-Anwendungen betrachtet.

4.1 Unterschiedliche Typen mobiler Anwendungen

Bei mobilen Anwendungen unterscheiden wird zwischen drei Anwendungstypen differenziert (e.g. [135], [235], [149]), die sich sowohl vom Aufbau, der Funktionsweise als auch vom Funktionsumfang wesentlich voneinander unterscheiden.

4.1.1 Native Anwendungen

Native Anwendungen werden Plattform- beziehungsweise Gerätespezifisch entwickelt und in Form von Binaries beispielsweise durch den Google Play Store oder den Apple App Store zur Verfügung gestellt. Diese Anwendungen zeichnet in der Regel eine hohe Performance und Verfügbarkeit aus und haben den Vorteil, direkt auf API-Schnittstellen, die das Betriebssystem zur Verfügung stellt, zugreifen zu können, wodurch sie große Teile der Funktionalität und Technologie eines Endgeräts beziehungsweise der zugrunde liegenden System-Plattform verwenden können.

Ein großer Nachteil von nativen Anwendungen ist die plattformübergreifende Portierbarkeit. Soll eine Anwendung beispielsweise sowohl für Android, als auch für Apple iOS verfügbar sein, erfordert dies zumindest das Neuentwickeln von plattformspezifischen Codeteilen. Wird also eine mobile Anwendung entwickelt, die plattformübergreifend möglichst viele Endnutzer und Nutzerinnen ansprechen soll, ist die Entwicklung und das Deployment von plattformspezifischen Anwendungsversionen nötig, wodurch sowohl der Entwicklungs-, als auch der Wartungsprozess von nativen Anwendungen länger, komplexer und aufwendiger wird, wie Heitkotter et al. [135] und Karadimce et al. [149] erwähnen.

4.1.2 Webanwendungen

Im Gegensatz zu nativen Anwendungen werden webbasierte Anwendungen über einen beliebigen kompatiblen Webbrowser aufgerufen, müssen daher nicht auf den mobilen Endgeräten installiert werden und sind geräte- und plattformunabhängig. Moderne Webanwendungen bauen vor allem auf Technologien wie HTML5 [136], CSS [68] und JavaScript [150] auf und werden im Anschluss über einen Webserver zur Verfügung gestellt [135]. Dies hat neben der breiten Kompatibilität und der damit verbundenen Reichweite den Vorteil des, im Vergleich zu nativen Anwendungen, relativ einfachen und raschen Deployment Prozesses, wodurch Webanwendungen wesentlich unkomplizierter in Bezug auf Wartung sowie Versionsanpassungen sind. Im Gegenzug dazu können Webanwendungen nicht im selben Umfang wie native Anwendungen auf Hardware- und Software-API sowie Funktionalität der Endgeräte zugreifen und eignen sich somit weder für ressourcenintensive Berechnungen noch für komplexere Anwendungen, die Hardware- und Software-Funktionalität eines Endgeräts benötigen. Für einige Funktionalitäten existieren allerdings, seit dem Einsatz von HTML5, JavaScript basierende Ersatz APIs. Ein Beispiel dafür ist etwa die API zur GPS Standort Bestimmung [234].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

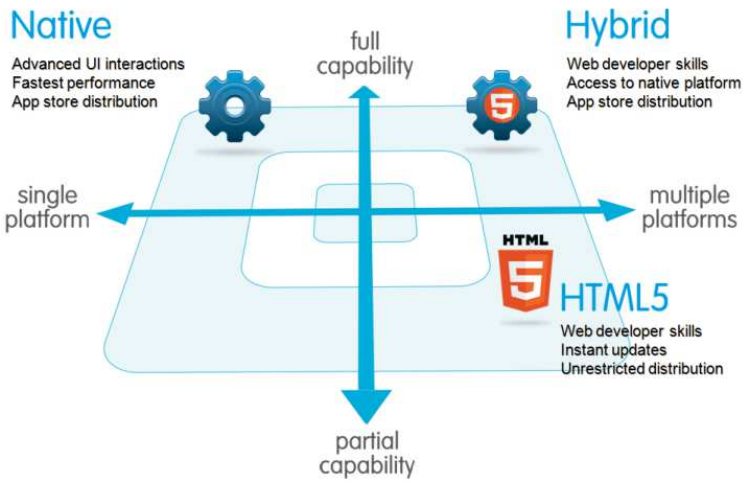


Abbildung 4.1: Native, webbasierte und hybride mobile Anwendungen [149]

4.1.3 Hybride Anwendungen

Unter hybriden Anwendungen wird die Kombination von Webanwendungen und nativen verstanden, bei denen mobile Webanwendungen in eine leichtgewichtige native Umgebung verpackt werden, die den Zugriff und die Verwendung von plattformspezifischen Funktionalitäten ermöglicht [135]. Hybride Anwendungen bestehen daher sowohl aus einem nativen Teil, als auch einer Webanwendung. Sie werden wie native Anwendungen beispielsweise über einen Anwendungsstore verteilt und direkt auf den Endgeräten installiert, nutzen aber die gleichen Technologien wie Webanwendungen. Frameworks zur Entwicklung hybrider Anwendungen sind beispielsweise *Reactive Native* [195], *Ionic* [141], *PhoneGap* [185]. Durch das Verpacken einer Webanwendung innerhalb einer nativen Anwendung wird die Entwicklung von plattformübergreifenden Anwendungen erheblich erleichtert und die Vorteile von beiden Typen vereint. Zugleich entstehen allerdings neue Schwachstellen und Sicherheitsrisiken (e.g. [109], [61], [216]).

In Abbildung 4.1 wird der Zusammenhang zwischen nativen, webbasierten und hybriden Anwendungen grafisch dargestellt. Native Anwendungen haben die Möglichkeit direkt auf plattform- und hardwarespezifische Funktionalität zuzugreifen, sind allerdings nicht plattform- und geräteübergreifend kompatibel. Dem gegenüber stehen mobile Webanwendungen, die direkt durch den Browser aufgerufen und ausgeführt werden, wodurch plattformübergreifende Kompatibilität erreicht wird. Zwischen diesen beiden Typen befinden sich hybride Anwendungen, die auf Webanwendungen basieren und in eine native Anwendung eingebettet werden. Dadurch wird der volle Funktionsumfang einer nativen Anwendung möglich und zugleich plattformübergreifende Kompatibilität gewährt.

Im nächsten Abschnitt, der sich auf den Aufbau von nativen Anwendungen bezieht, wird die Struktur von Android-Anwendungen unter die Lupe genommen.

4.2 Struktur und Aufbau von Android-Anwendungen

Bei Android-Anwendungen wird zwischen Systemanwendungen, die direkt im Betriebssystem inkludiert sind oder vom Gerätehersteller vorinstalliert wurden und benutzerspezifischen Anwendungen unterschieden. Systemanwendungen werden vom System als vertrauenswürdige und sichere Anwendungen eingestuft und können nicht durch den Nutzer oder die Nutzerin verändert sowie deinstalliert werden. Diese besitzen mehr und vor allem kritischere Berechtigungen als An-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

wendungen, die durch den Benutzer oder die Benutzerin verwaltet werden, wie Elenkov [88] erläutert. Der Aufbau von System- und Benutzeranwendungen, der im folgenden Teil näher betrachtet wird, ist allerdings ident.

4.2.1 Struktur von Android-Anwendungen

Android-Anwendungen werden in Java entwickelt, anschließend kompiliert und üblicherweise in Form eines Android-spezifischen Package Format, bekannt unter *Android Package Kit (APK) [13]*, über den Google Play Store verteilt und installiert, der diese in die durch Tabelle 4.1 aufgelisteten Anwendungskategorien unterteilt. Das APK-Dateiformat ist eine Erweiterung des *Java Archiv (JAR) [143]*, das auf dem bekannten *ZIP*-Dateiformat aufbaut [88] und besitzt die in Abbildung 4.2 dargestellte Struktur, auf die an dieser Stelle eingegangen wird.

ANDROID_WEAR	ART_AND_DESIGN	AUTO_AND_VEHICLES
BEAUTY	BOOKS_AND_REFERENCE	BUSINESS
COMICS	COMMUNICATION	DATING
EDUCATION	ENTERTAINMENT	EVENTS
FINANCE	FOOD_AND_DRINK	HEALTH_AND_FITNESS
HOUSE_AND_HOME	LIBRARIES_AND_DEMO	LIFESTYLE
MAPS_AND_NAVIGATION	MEDICAL	MUSIC_AND_AUDIO
NEWS_AND_MAGAZINES	PARENTING	PERSONALIZATION
PHOTOGRAPHY	PRODUCTIVITY	SHOPPING
SOCIAL	SPORTS	TOOLS
TRAVEL_AND_LOCAL	VIDEO_PLAYERS	WEATHER
GAME	GAME_ACTION	GAME_ADVENTURE
GAME_ARCADE	GAME_BOARD	GAME_CARD
GAME_CASINO	GAME_CASUAL	GAME_EDUCATIONAL
GAME_MUSIC	GAME_PUZZLE	GAME_RACING
GAME_ROLE_PLAYING	GAME_SIMULATION	GAME_SPORTS
GAME_STRATEGY	GAME_TRIVIA	GAME_WORD
FAMILY	FAMILY_ACTION	FAMILY_BRAINGAMES
FAMILY_CREATE	FAMILY_EDUCATION	FAMILY_MUSICVIDEO
FAMILY_PRETEND		

Tabelle 4.1: Übersicht der 58 Google Play Store Kategorien

AndroidManifest.xml

Das `AndroidManifest.xml` ist das Herzstück jeder Android-Anwendung und beinhaltet alle nötigen Deklarationen wie beispielsweise Anwendungsname, Version, Komponenten, Berechtigungen, Android-Version und andere Metadaten.

Classes.dex

Im `Classes.dex` Paket befindet sich der kompilierte und ausführbare Quellcode einer Anwendung in Form von `.dex`-Dateien, welche durch die *Dalvik Virtual Machine (DVM)* [86] (ab Android 5 *Android Runtime (ART)* [186]) verarbeitet werden.

Resources.arsc

Das `Resources.arsc` Paket einer APK-Datei beinhaltet alle kompilierten Inhalte einer Android-Anwendung wie etwa Strings und Styles.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
apk/
|-- AndroidManifest.xml❶
|-- classes.dex❷
|-- resources.arsc❸
|-- assets/❹
|-- lib/❺
|-- META-INF/❻
`-- res/❼
```

Abbildung 4.2: Vereinfachte APK-Anwendungsstruktur [88]

Assets

`Assets` beinhalten mit der Anwendung ausgelieferte Ressourcen wie Icons, Musik Dateien oder Schriftarten, die im Quellcode direkt referenziert werden.

Lib

Der `Lib` Bereich beinhaltet externe Bibliotheken, die in der Anwendung zum Einsatz kommen.

Res

Ressourcen wie Bilder, Animationen oder Menüdefinitionen, die vom Android-Quellcode direkt referenziert werden, werden im `res` Paket nach Typ verwaltet.

META-INF

Im `META-INF` Paket einer APK-Datei befinden sich Zertifikate zur Android Codesignatur. Einen Überblick zur Anwendungssignierung wird in Abschnitt 5.2.4 gegeben.

4.2.2 Android-Anwendungskomponenten

Eine Android-Anwendung setzt sich aus mehreren eigenständigen und lose gekoppelten Komponenten zusammen, die in sich abgeschlossen eigene Zuständigkeiten sowie Aufgaben besitzen und wesentlichen Einfluss auf die Sicherheit einer Anwendung nehmen [88]. Diese Komponenten haben, anders als bei gewöhnlichen Java-Anwendungen, nicht einen einzigen Einstiegspunkt, sondern mehrere verschiedene. Komponenten und deren Einstiegspunkte werden im `Android-Manifest.xml` definiert beziehungsweise konfiguriert, um bei der Installation im System für die Verwendung registriert zu werden.

Definierte Komponenten können dabei anhand des booleschen Attributes `exported` als öffentlich oder als privat definiert werden, wodurch Komponenten von anderen Anwendungen genutzt werden können, die mithilfe von *Intents* (eine abstrakte Beschreibung einer durchzuführenden komponentenübergreifenden Aktion oder Funktion), nach der Implementierung einer spezifischen Funktionalität suchen und diese wiederverwenden können [16], [88]. In der offiziellen Android Komponentendokumentation [16] wird das Beispiel genannt, dass eine fremde E-Mail-Anwendung über eine speziell zum Verfassen einer neuen E-Mail hinterlegten Aktivität, durch eine Kamera Anwendung gestartet werden kann. Dies bietet zum einen zwar die einfache Möglichkeit der anwendungsübergreifenden Wiederverwendung von bestehenden Funktionalitäten, stellt zum anderen aber eine mögliche Sicherheitsschwachstelle dar (siehe Abschnitt 7.3.2). Wird eine Komponente hingegen als privat deklariert, ist es nur der eigenen Anwendung möglich diese zu verwenden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Listing 4.1 zeigt beispielhaft die Deklaration einer privaten Komponente im `AndroidManifest-.xml`. Das Attribut `android:name` definiert den vollständigen Klassennamen der registrierten Aktivitäten Komponente während das Attribut `android:label` den für den Benutzer oder die Benutzerin sichtbaren Namen der Aktivität definiert. Das Attribut `android:exported="false"` deklariert die Komponente als privat. Darüber hinaus existieren noch weitere Attribute, die der Android Komponentendokumentation [16] entnommen werden können. Eines der wichtigsten Attribute ist `android:permission`, welches die zur Ausführung der Komponente nötigen Berechtigungen beschreibt.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3      <application android:icon="@drawable/app_icon.png" ... >
4          <activity android:name="com.my.project.Activity"
5                    android:label="@string/my_activity_label"
6                    android:exported="false"... >
7          </activity>
8           ...
9      </application>
10 </manifest>
```

Listing 4.1: Ausschnitt einer privaten Komponentendefinition im AndroidManifest.xml [16]

Bei Anwendungskomponenten unterscheidet Android vier Hauptkomponenten Gruppen [16], [88]. Diese teilen sich in *Activities*, *Services*, *Content Providers* sowie *Broadcast Receivers* und werden an dieser Stelle überblicksmäßig eingeführt.

Activities

Eine Aktivität stellt den Einstiegspunkt zur Benutzerkommunikation dar und beschreibt ein grafisches Interface oder genauer einen Screen. Jede Anwendung kann mehrere Aktivitäten und somit Screens besitzen, von denen jede einzelne einen Einstiegspunkt in die Anwendung darstellt.

Services

Ein Service ist eine Komponente, die im Hintergrund läuft und definierte Aufgaben erledigt, wodurch sich diese Art von Komponente besonders für lange Berechnungen oder Tätigkeiten eignet, die transparent im Hintergrund verarbeitet werden.

Content Providers

Ein Content Provider bietet die Möglichkeit, URI-basiert auf das Dateisystem sowie Anwendungsdaten zuzugreifen, und stellt einen Anwendungseinstiegspunkt zum Teilen von Daten dar. Dabei erfolgt eine strenge und feingranulare Berechtigungsüberprüfung, wodurch es möglich ist, einzelne spezifische Datensätze zu teilen.

Broadcast Receivers

Broadcast Receiver reagieren, wie der Name vermuten lässt, auf systemweite Events, egal ob diese vom System selbst oder von einer anderen Anwendung stammen. Dadurch können Anwendungen auf gewisse Vorkommnisse entsprechend reagieren, wodurch Broadcast Receiver ebenso weitere Einstiegspunkte in eine Anwendung darstellen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2.3 Inter-Component Communication (ICC)

Die ICC beschreibt die Kommunikation zwischen Komponenten, bei der sogenannte Intents versendet werden, welche die Kommunikation und den Datenaustausch zwischen Komponenten beschreiben. Diese Art der Kommunikation stellt vor allem zwei Gefahren dar, wie Chin et al. [62] beschreiben. Wird eine Komponente als öffentlich definiert, kann diese von jeder anderen Anwendung wiederverwendet werden, wodurch Drittanwendungen möglicherweise ungewollte Funktionalitäten und Berechtigungen erlangen können *(Intent Spoofing)*. Darüber hinaus können Intents von schadhaften Komponenten abgefangen werden, wodurch diese Einfluss und unerwünschten Zugriff auf Daten bekommen und diese missbräuchlich verarbeiten können. Speziell von impliziten Intents geht eine Gefahr aus, da die Zielanwendung eines impliziten Intents für den Entwickler oder die Entwicklerin einer Anwendung nicht bekannt ist. Gerade bei der Analyse von sicherheitskritischen Schwachstellen einer Anwendung spielt die ICC eine große Rolle und darf nicht vernachlässigt werden.

Android Intents

Intents sind abstrakte Beschreibungen von Operationen, die zur Komponenten übergreifenden ICC oder anwendungsübergreifenden Kommunikation und Wiederverwendung von Funktionalitäten verwendet werden [14], [88]. Dabei können einzelne Anwendungen auf abstrakte Weise die Implementierung von Komponenten, die eine bestimmte Funktionalität aufweisen, wiederverwenden und mithilfe von Intents zusätzliche Daten und Informationen zwischen den Komponenten austauschen, die zur Durchführung der gewünschten Operation nötig sind. Bei Android Intents wird zwischen expliziten und impliziten Intents unterschieden.

Expliziter Intent

Sendet eine Android-Anwendung oder genauer gesagt eine Komponente einen expliziten Intent, beschreibt dieser explizit die gewünschte Zielkomponente durch Package und Klassennamen und übergibt dieser die zur Verarbeitung nötigen Daten, wie in Listing 4.2 anhand der `DownloadService.class` zu sehen ist. Die Android-Anwendung spezifiziert also selbst eindeutig, welche Komponente beziehungsweise Implementierung innerhalb einer speziellen Anwendung die gewünschte Aktion durchführen soll. Besitzt die Intent-sendende Anwendung alle nötigen Berechtigungen, um die gewünschte Aktion der Zielkomponente durchzuführen, wird diese aufgerufen, ansonsten wird der Nutzer oder die Nutzerin zur Bestätigung der nötigen Berechtigungen gebeten. Dies impliziert allerdings die korrekte Definition einer Komponente, die alle erforderlichen Berechtigungen listet.

```
1
2   // Executed in an Activity, so 'this' is the Context
3   // The fileUrl is a string URL, such as "http://www.example
    ↪ .com/image.png"
4   Intent downloadIntent = new Intent(this, DownloadService.
    ↪ class);
5   downloadIntent.setData(Uri.parse(fileUrl));
6   startService(downloadIntent);
```

Listing 4.2: Beispiel eines expliziten Intents [14]

Impliziter Intent

Bei impliziten Intents spezifiziert die sendende Anwendung nur eine abstrakte Beschreibung

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

der durchzuführenden Aktion/Operation und den dafür nötigen Daten, lässt allerdings völlig offen, welche Anwendung und Implementierung zur Durchführung herangezogen wird, wie anhand der impliziten Intent Definition in Listing 4.3 zu sehen ist. Das System sucht in diesem Fall nach allen Anwendungen, die im `AndroidManifest.xml` einen geeigneten *Intent Filter*, welcher nachfolgend erklärt wird, konfiguriert haben und bietet dem Nutzer oder der Nutzerin im Falle eines Mehrfachtreffers die Möglichkeit, sich frei für eine Anwendung und somit eine spezifische Implementierung zu entscheiden.

Implizite Intents stellen dadurch ein gewisses Sicherheitsrisiko dar, da ein Entwickler oder eine Entwicklerin einer Anwendung keinen Einfluss darauf hat, welche Anwendung den gesendeten Intent und somit die enthaltenen, möglicherweise sensiblen, Daten empfängt und verarbeitet. Der Anwender beziehungsweise die Anwenderin kann sich in diesem Fall für eine verfügbare Zielanwendung entscheiden und übergibt dieser die zu verarbeitenden Daten.

```
1
2     // Create the text message with a string
3     Intent sendIntent = new Intent();
4     sendIntent.setAction(Intent.ACTION_SEND);
5     sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
6     sendIntent.setType("text/plain");
7
8     // Verify that the intent will resolve to an activity
9     if (sendIntent.resolveActivity(getPackageManager()) != null
      ↪ ) {
10        startActivity(sendIntent);
11    }
```

Listing 4.3: Beispiel eines impliziten Intents der einen *ACTION_SEND* Intent sendet [14]

Um eine Komponente, die im `AndroidManifest.xml` registriert wird, über Intents verfügbar zu machen, müssen entsprechende Intent Filter deklariert werden, wie Listing 4.4 beispielhaft an einer Aktivität (`<activity>` Tag) zeigt. Bei dieser Konfiguration können sowohl ein, als auch mehrere Filter hinterlegt werden, wobei eine Intent Filter Deklaration immer mit einem `<intent-filter>` Tag beginnt. Das Kind-Element `<action>` definiert durch das `android:name` Attribut die vom Intent akzeptierte Aktion, wie etwa die Aktivität zum Versenden einer Textnachricht (`android.intent.action.SEND`). Weiters definiert `<category>` durch das `android:name` Attribut die akzeptierte Intent Kategorie. Um einen implizite Intents zu erhalten, muss beispielsweise die Kategorie `android.intent.category.DEFAULT` akzeptiert werden. Das dritte Kind-Element `<data>` beschreibt die unterstützten Datentypen eines Intents, beispeilsweise den MIME-Type [104] `text/plain`.

```
1 <activity android:name="ShareActivity">
2     <intent-filter>
3         <action android:name="android.intent.action.SEND"/>
4         <category android:name="android.intent.category.DEFAULT"/>
5         <data android:mimeType="text/plain"/>
6     </intent-filter>
7 </activity>
```

Listing 4.4: Beispiel eines Intent Filters einer Aktivität die einen `ACTION_SEND` Intent erhält wenn Daten vom Typ Text empfangen werden [14]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

4.2.4 Berechtigungen

Android ist ein auf Berechtigungen basiertes Betriebssystem. Anwendungen müssen Berechtigungen auf API-Level anfordern, um auf spezifische Funktionalitäten zugreifen zu können. Wie die Berechtigungsdefinition sowie Anforderung funktioniert, wird in diesem Abschnitt beschrieben.

Der Android Dokumentation zum Berechtigungskonzept [19] zufolge wird zwischen *normal*, *dangerous*, *special* und *signatur level* Berechtigungen unterschieden. Als normal eingestufte Berechtigungen stellen nur ein geringes Risiko für die Beeinflussung der Privatsphäre dar und werden dazu verwendet, um auf Daten und Ressourcen außerhalb der Anwendungssandbox (siehe Abschnitt 5.2.2) zuzugreifen. Gefährliche Berechtigungen beschreiben Berechtigungen, durch die Zugriff auf benutzerspezifische Daten ermöglicht wird, wodurch Eingriff in die Privatsphäre genommen werden kann. Diese Gruppe umfasst die in Tabelle 4.2 dargestellten Berechtigungen.

Permission Group	**Permissions**
CALENDAR	READ_CALENDAR, WRITE_CALENDAR
CALL_LOG	READ_CALL_LOG, WRITE_CALL_LOG, PROCESS_OUTGOING_CALLS
CAMERA	CAMERA
CONTACTS	READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE, READ_PHONE_NUMBERS, CALL_PHONE, ANSWER_PHONE_CALLS, ADD_VOICEMAIL, USE_SIP
SENSORS	BODY_SENSORS
SMS	SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE

Tabelle 4.2: Dangerous Permissions [19]

Zu den speziellen Berechtigungen zählen *SYSTEM_ALERT_WINDOW* und *WRITE_SETTINGS*, die anders als normale und gefährliche Berechtigungen von den meisten Anwendungen nicht benötigt und mit Vorsicht erteilt werden sollten [19]. Weiters existiert noch die Gruppe der Signatur-basierten Anwendungsberechtigungen, die bei eigens definierten Berechtigungen, auf die nachfolgend eingegangen wird, eine Rolle spielen. Diese werden vom System bei der Installation einer Anwendung vergeben, wenn die Anwendung, welche die Berechtigung anfordert, mit demselben Zertifikat signiert wurde als die Anwendung, die diese Berechtigung definiert [19].

Neben den von Android vordefinierten Berechtigungen, die den Zugriff auf System und Basisfunktionalitäten definieren, ist es Anwendungsentwicklern und Entwicklerinnen möglich über das `AndroidManifest.xml` und einen `<permission>` Tag eigene Berechtigungen zu definieren, die den Zugriff auf Komponenten regeln. Wie diese Definition aufgebaut ist, wird anhand des Beispiels in Listing 4.5 erklärt.

```
1 <manifest
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   package="com.example.myapp" >
4
5     <permission
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
6         android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
7         android:label="@string/permlab_deadlyActivity"
8         android:description="@string/permdesc_deadlyActivity"
9         android:permissionGroup="android.permission-group.COST_MONEY"
10        android:protectionLevel="dangerous" />
11      ...
12 </manifest>
```

Listing 4.5: Berechtigungsdefinition im AndroidManifest.xml [19]

Das Attribut `android:name` spezifiziert erwartungsgemäß den Namen der Berechtigung, während `android:label` und `android:description` eine textuelle Beschreibung der Berechtigung darstellt, die dem Benutzer oder der Benutzerin präsentiert wird. Die Definition der Berechtigungsgruppe durch `android:permissionGroup` ist optional und dient der kategorisierten Darstellung bei der Berechtigungsanforderung. Das Attribut `android:protectionLevel` beschreibt abschließend noch die von der Berechtigung ausgehende Gefahr.

Neben der Definition eigener Berechtigungen ist vor allem die Definition aller benötigten Berechtigungen einer Anwendung wichtig, die ebenfalls im `AndroidManifest.xml` durch den `<uses-permission>` Tag deklariert werden, wie im Listing 4.6 anhand der *SEND_SMS* Berechtigung beispielhaft zu sehen ist.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3           package="com.example.snazzyapp">
4
5     <uses-permission android:name="android.permission.SEND_SMS"/>
6
7     <application ...>
8         ...
9     </application>
10 </manifest>
```

Listing 4.6: Ausschnitt einer Berechtigungsanforderung im AndroidManifest.xml [19]

Diese dort deklarierten Berechtigungen werden abhängig von der Android-Version zur Laufzeit oder vor der Installation über ein Pop-up-Fenster angefordert, wodurch der Anwender sowie die Anwenderin selbst über die Erteilung oder Verweigerung entscheiden kann. Ab Android-Version 6.0 passiert dies zur Laufzeit, wodurch der Nutzer oder die Nutzerin unmittelbar vor der ersten Verwendung einer Berechtigung die Möglichkeit erhält, über die Berechtigungsvergabe zu entscheiden. Darüber hinaus können Berechtigungen auch zur Laufzeit verändert werden. Vor Android-Version 6 müssen alle angeforderten Berechtigungen vor der Installation erteilt werden. Wird eine Berechtigung nicht erteilt, bricht die Anwendungsinstallation ab [19].

4.2.5 Speicher

Wie die Android Speicherdokumentation [73] beschreibt, bietet Android mehrere unterschiedliche Möglichkeiten Daten abzuspeichern. Dazu zählen der interne sowie externe Datenspeicher, Datenbanken und geteilte Einstellungen. Darüber hinaus bietet Android einen eigenen Keystore zur Aufbewahrung sensibler Schlüssel [15]. All diese unterschiedlichen Speicher besitzen ihre eigenen Eigenschaften, wodurch sich manche Speicherarten für bestimmte Anwendungsfälle besser eignen als andere und der Entwickler oder die Entwicklerin selbst entscheiden kann, welche Speicherart für den vorgesehenen Anwendungsfall verwendet wird. An dieser Stelle wird auf die

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

wichtigsten Eigenschaften dieser Speicherarten eingegangen, um aufzuzeigen, dass die Wahl des richtigen Speichermediums sicherheitsrelevante Auswirkungen hat und bei einer Anwendungsanalyse miteinbezogen werden muss.

Interner Speicher

Der interne Speicher besitzt ein strenges Berechtigungskonzept, wodurch Daten einer Anwendung, bei intakter Berechtigungsprüfung, nicht von fremden Android-Anwendungen gelesen und bearbeitet werden können. Dieser Speicher eignet sich daher besonders, um anwendungsspezifische private Daten abzuspeichern, zu denen keine andere Anwendung am Endgerät Zugriff erlangen sollte. Üblicherweise werden Daten innerhalb des internen Speichers in anwendungsspezifischen Unterordnern verwaltet, die durch eine Deinstallation samt Inhalt entfernt werden. Aus diesen Gründen eignet sich diese Speicherart nicht für eine anwendungsübergreifende Datenspeicherung. Wie Zhang et al. [251] zeigen, gehen im Bereich der Android Anwendungsdeinstallation Risiken von zurückbleibenden Datenresten aus, die den Zugriff auf sensible Daten wie beispielsweise Credentials und Verläufe beinhalten und in weiterer Folge *Privilege-Escalation*-Attacken (e.g. [250], [40], [74]) ermöglichen können. Allgemein empfiehlt es sich, Daten ausschließlich verschlüsselt abzuspeichern (siehe Abschnitt 5.2.5), um unbefugten Zugriff beispielsweise durch gerootete Geräte oder Privilege-Escalation-Attacken zu verhindern und die Integrität sowie Vertraulichkeit von Daten zu bewahren.

Externer Speicher

Beim externen Speicher handelt es sich um einen Speicherbereich, der durch den Benutzer oder die Benutzerin hinzugefügt, entfernt sowie verwaltet werden kann und meist durch externe Speicherkarten repräsentiert wird, die den Speicher des Endgeräts erhöhen. Dieser Speicher unterscheidet sich wesentlich vom vorhin besprochenen internen Speicher, da der externe für alle Anwendungen gleichermaßen zugänglich ist und kein feingranulares Berechtigungskonzept besitzt. Daten, die auf diese Weise abgespeichert werden, besitzen die Berechtigungen `World readable/writeable` und können von beliebigen Anwendungen, anwendungsübergreifend gelesen und verändert werden.

Geteilte Einstellungen

Wenn es darum geht Key-Value Paare abzuspeichern, bietet sich die Verwendung des sogenannten *Shared Preferences* [21] Speichers an. In diesem können primitive Datentypen in Form einer Key-Value Map im internen Speicher des Gerätes abgespeichert werden. Diese Einstellungen sind demzufolge nur von der zugehörigen Anwendung selbst les- und schreibbar.

Datenbank

Android bietet außerdem die Möglichkeit, Daten innerhalb einer SQLite-Datenbank [222] abzuspeichern. Entscheidet sich ein Entwickler oder eine Entwicklerin für die Verwendung dieser Speicherart, ist die Datenbank ausschließlich von der Anwendung selbst les- und schreibbar und unbefugte Anwendungen haben keinen Zugriff auf darin gespeicherte Daten.

Android Keystore System

Ab Android 4.3 existiert ein eigenes Keystore System [15] zum Speichern von sicherheitsrelevanten Daten wie etwa Kryptografie-Schlüsseln, das vom System verwaltet wird und Keys vor unberechtigten Zugriffen schützt. Der Zugriff beziehungsweise die Verwendung von abgespeicherten Schlüsseln wird ausschließlich über das vom System verwaltete Keystore System ermöglicht, wodurch Schlüssel nicht innerhalb des Anwendungsprozesses gelangen. Auf diese Weise wird der unbefugte Zugriff auf sensible Daten erschwert. Des Weiteren bietet das Android Keystore System die Möglichkeit des Einsatzes von Hardware Security der

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Android Geräte wie etwa *Trusted Execution Environment* oder *Secure Element*. Ab Android 9 werden darüber hinaus geräteabhängig Hardware Sicherheitsmodule zur Schlüsselerzeugung und Aufbewahrung unterstützt, die aus einer eigenen CPU, sicheren Hardwarespeicher und Schlüsselgenerator bestehen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5 Architektur und Sicherheitskonzept von Android

Dieser Abschnitt gibt einen Einblick in den Aufbau und die Funktionsweise des Android-Betriebssystems, da diese für dieses Werk und die nachfolgende Definition von Sicherheitsstandard sowie für die Einstufung der Android-Anwendungssicherheit durch das Framework eine besondere Relevanz darstellt. Zu Beginn wird die schichtenweise aufgebaute Architektur von Android vereinfacht erklärt, aus der sich der gesamte Funktionsumfang des Betriebssystems zusammensetzt. Im zweiten Teil des Kapitels liegt der Fokus auf den Sicherheitskonzepten des mobilen Systems.

5.1 Architektur

Android lässt sich, wie in Abbildung 5.1 zu sehen, in fünf Schichten einteilen, die abgeschlossene Aufgaben erfüllen und aufeinander aufbauen, wie die Android eigene Dokumentation [186] und Elenkov [88] beschreiben. Im folgenden Abschnitt werden die einzelnen Schichten näher betrachtet, um auf diesem Weg einen groben Überblick über die Funktionsweise von Android-Systemen zu erhalten. Genauere und tiefer greifende Erklärungen der Android-Systemarchitektur würden an dieser Stelle den Rahmen der Arbeit sprengen und sind für die Sicherheitsanalyse nicht relevant. Um die gesamte Funktionalität am besten darstellen zu können, wird der sogenannte *Bottom-Up* Ansatz beginnend mit der tiefsten Schicht (der Linux-Kernel-Schicht) angewendet.

5.1.1 Linux-Kernel-Schicht

Am Beginn der Android Schichtenarchitektur befindet sich die Linux-Kernel-Schicht, welche die darunter verborgene Hardware abstrahiert und beschreibt [186], [88]. Dabei handelt es sich um eine speziell für Android angepasste Version eines Linux-Kernels, der für die Verwendung der zugrunde liegenden Hardware zuständig ist und die Speicher- und Prozess-Organisation, Netzwerkanbindung sowie das Android eigene Sicherheitskonzept verwaltet.

Eine der Hauptanpassungen im Vergleich zum original Linux-Kernel, ist die veränderte *Inter-Process Communication (IPC)*. In die für Android angepasste Version des Kernels wurde das

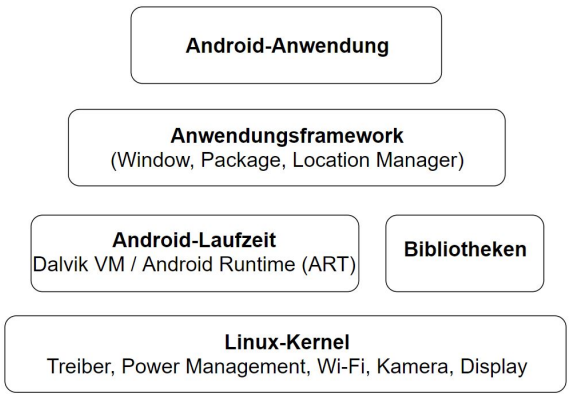


Abbildung 5.1: Vereinfachte Darstellung der Android-Architektur

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

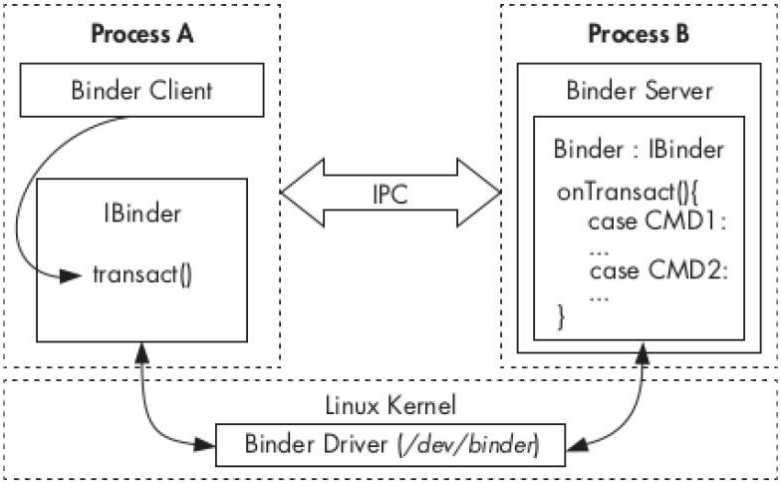


Abbildung 5.2: Binder IPC [88]

Binder Framework [38], [88] eingebaut, wodurch die lokale Kommunikation zwischen Prozessen mittels *Remote Procedure Calls (RPC)* [197] ermöglicht wird. Geräteübergreifende RPC über das Netzwerk werden bewusst nicht unterstützt.

Die Aufgabe des Binder Framework ist es eine API zur Verfügung zu stellen, über welche sämtliche IPC läuft. Das Framework übernimmt im Hintergrund Tätigkeiten wie die Zuweisung des Speichers, der für die Kommunikation benötigt wird, und den Nachrichtentransfer zwischen den einzelnen Prozessen. In Abbildung 5.2 ist die stark vereinfachte Kommunikation zwischen Prozess A und Prozess B zu sehen. Dabei interagiert Prozess A über das Binder Interface, in der Abbildung mit *IBinder* gekennzeichnet, welches die Kommunikation zu Prozess B über den Linux-Kernel übernimmt. Dafür wird ein RPC über den, in der Abbildung *Binder driver* genannten, Kommunikationskanal angestoßen. Auf der Empfängerseite in Prozess B wird dieser anschließend durch das Binder Framework verarbeitet und Informationen an den Prozess weitergeleitet [88].

5.1.2 Bibliotheken-Schicht

Die Schicht über dem Linux-Kernel bildet die Schicht der Bibliotheken [186]. Diese beschreibt C und C++ Bibliotheken, die in Android enthaltenen sind und beispielsweise der Verarbeitung von Video-, Bild- und Audiomaterial sowie SQLite-Datenbanken dienen. Der Zugriff auf die Bibliotheken findet über die direkt darüberliegende Schicht, der Anwendungsframework-Schicht, statt.

5.1.3 Laufzeitschicht

Die Laufzeitschicht befindet sich auf der gleichen Ebene wie die Bibliotheken-Schicht und erweitert diese um einige Java Hauptbibliotheken, um die Grundfunktionalität des Android-Systems zu bilden.

Jede Android-Anwendung läuft in einem eigenen für sich abgeschlossenen Prozess, der ab Android 5 in einer eigenen Android Runtime (ART) ausgeführt wird [186]. Vor Android 5 wurden Android-Anwendungen in einer DVM ausgeführt. Die ART beziehungsweise die DVM ist eine vereinfachte Android eigene Interpreter Maschine, vergleichbar mit der *Java Virtual Machine (JVM)* [227] und ist für die Android-Plattform und Endgeräte mit wenigen Hardware- beziehungsweise Energieressourcen optimiert. Im Unterschied zur JVM, welche Java `.class`-Dateien *stackbasiert* verarbeitet, besitzt die DVM eine eigene native Sprache (DEX) und verarbeitet `.dex`-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

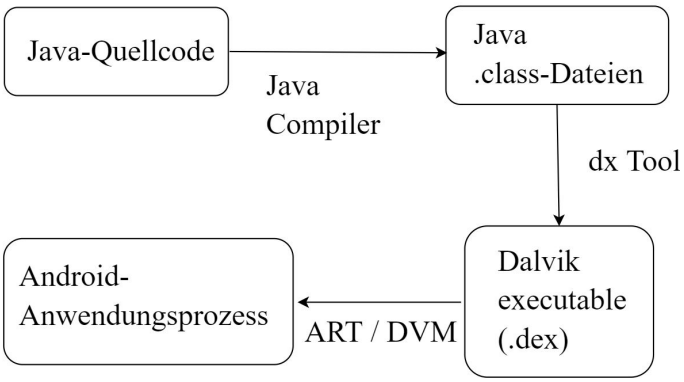


Abbildung 5.3: Android-Quellcode Kompilierungsprozess

Dateien *registerbasiert*. ART verarbeitet ebenso wie die DVM `.dex`-Dateien und optimiert Maschinenbefehle ab Android 9 weiter, um Ressourcen zu sparen.

Abbildung 5.3 zeigt die Verarbeitung von Android-Quellcode, der in Java vorliegt und zuerst in `.dex`-Dateien kompiliert wird, um im Anschluss von der DVM beziehungsweise der ART verarbeitet werden zu können. Dazu wird zunächst der Java-Quellcode mittels *Java Compiler* in gewöhnliche Java `.class`-Dateien kompiliert, bevor eine Optimierung und Konvertierung in `.dex`-Dateien durch ein in der Abbildung sogenanntes *DX Tool* durchgeführt wird.

5.1.4 Anwendungsframework-Schicht

Die nächste Schicht direkt über der Bibliotheken und Android-Laufzeitschicht stellt die Anwendungsframework-Schicht dar, die Android-spezifische Bibliotheken, die nicht in Java enthalten sind beinhaltet und zusammen mit den darunterliegenden Schichten ein Framework für die Android-Anwendungsentwicklung bildet [186].

Dadurch bietet dieses Framework Entwicklern und Entwicklerinnen die Möglichkeit, mittels API-Aufrufen auf darunterliegende Grundfunktionalitäten zuzugreifen, wodurch sich die Funktionalität von Android ergibt. Zusätzlich fördert diese Schicht vor allem die Wiederverwendung von Komponenten und erleichtert durch Abstraktion die Implementierungen von neuen komplexen Anwendungen. Auf diese Weise können sich Entwickler und Entwicklerinnen auf die eigentliche Funktionalität ihrer Anwendungen konzentrieren und auf Grundfunktionen und Services der Android Geräte unkompliziert zugreifen. Zu diesen Android-Systemservices, die im Anschluss in einigen Sätzen erklärt werden, zählen vor allem *Views*, *Notification Manager*, *Activity Manager*, *Content* und *Resource Manager* [186]. Des Weiteren können bereits existierende Komponenten durch eigene, neu entwickelte, ersetzt werden, wodurch die rasche und einfach Entwicklung von funktionalen, innovativen und moderne Anwendungen ermöglicht wird.

Die **Views** beinhalten eine Vielzahl grafischer Elemente, wie zum Beispiel Listen, Textfelder, Menüs, Schaltflächen und dergleichen. Die Wiederverwendung dieser funktionalen Elemente ermöglichen es einheitliche Benutzeroberflächen zu schaffen, ohne sich groß um die Darstellung dieser Bedienelemente kümmern zu müssen. Gerade bei mobilen Geräten mit unterschiedlichen Displaygrößen und Auflösungen stellt dies eine maßgebliche Erleichterung dar.

Der **Notification Manager** ist dafür zuständig den Nutzer oder die Nutzerin mittels Benachrichtigungen über Events und Vorgänge, die im Hintergrund passieren, zu informieren. Mithilfe

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

dieses Managers können Entwickler und Entwicklerinnen einfach und unkompliziert Benachrichtigungen für den Endnutzer oder die Endnutzerin erstellen, ganz egal, ob es sich um eine Statusleisten Benachrichtigung oder ein Signal in Form von Ton, Vibration oder dem Statuslicht handelt.

Der **Activity Manager** steuert und verwaltet das Navigieren innerhalb einer Anwendung sowie den Lebenszyklus dieser und enthält Informationen über Services und Aktivitäten, um mit diesen zu interagieren.

Die **Content Manager** und **Resource Manager/Provider** ermöglichen den anwendungsübergreifenden Zugriff auf, sowie das Teilen von Daten wie etwa Kontakten.

5.1.5 Anwendungsschicht

Direkt über der Anwendungsframework-Schicht liegt die Android-Anwendungsschicht, die ebenfalls auf allen darunter liegenden Schichten aufbaut und neben den grundlegenden Android-Funktionalitäten und integrierten System-Anwendungen auch die durch den Benutzer installierten Anwendungen enthält. Somit stellt diese Schicht die oberste und letzte Schicht der Android-Architektur dar [186].

5.2 Sicherheitskonzepte

Bevor die Definition von Sicherheitsstandards diskutiert wird, werden in diesem Abschnitt des Kapitels wichtige Sicherheitskonzepte des Android-Betriebssystems erklärt.

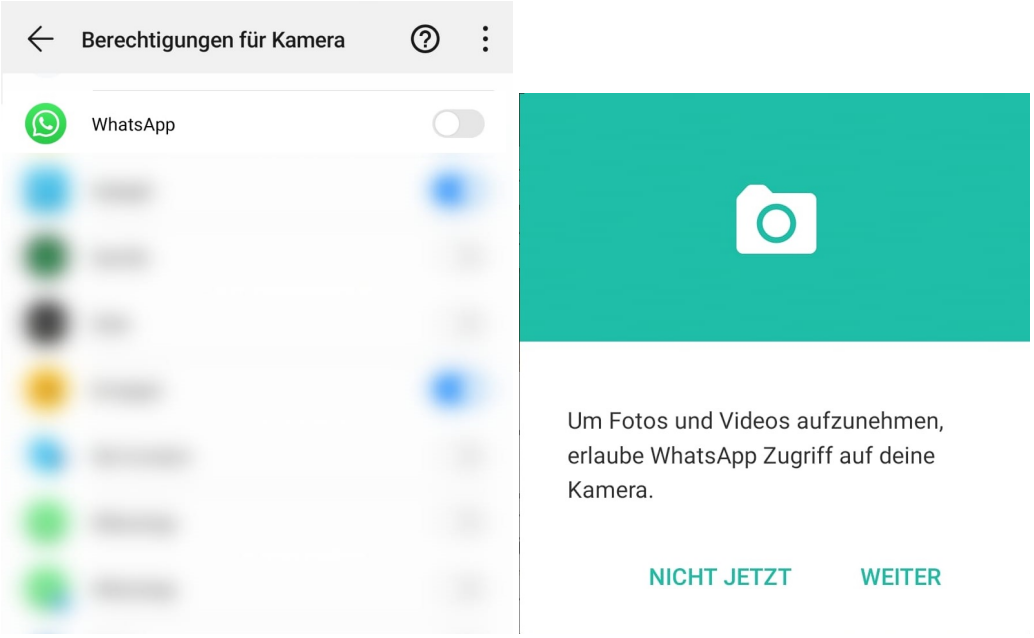
5.2.1 Berechtigungskonzept

Android verfügt über ein feingranulares Berechtigungssystem, um anwendungsübergreifende Zugriffe auf Ressourcen und Funktionalitäten zu koordinieren (e.g. [96], [241]). Dieses Berechtigungssystem basiert auf einer vom Android-Betriebssystem verteilten *UserID (UID)*, anhand dieser Zugriffe gewährt beziehungsweise verwehrt werden, wie die Android Dokumentation zu Berechtigungen [18] beschreibt.

Das Konzept des Ressourcen- und Funktionalitäten-Zugriff über ein Berechtigungssystem hat sowohl positive als auch negative Seiten, wie Felt et al. [96], [97] analysieren. Zum einen ist Android dadurch besonders offen und fördert die Wiederverwendung von Software sowie Grundfunktionalitäten, wodurch es Entwicklern und Entwicklerinnen ermöglicht wird, sich auf die eigentliche Funktionalität einer Anwendung zu konzentrieren und so in kurzer Zeit innovative Anwendungen zu bauen. Zum anderen öffnet diese Funktionalität aber auch Tür und Tor für den Einbau von schadhaften, unerwünschten Verhalten, da viele Endnutzer und Nutzerinnen unbedacht mit Berechtigungsanforderungen umgehen, zu wenig Kontext für eine bedachte Entscheidung besitzen, oder häufig schwer nachvollziehbar ist, ob Anwendungen bestimmte Berechtigungen wirklich benötigen, um ihre vollständige Funktionalität zu erhalten. Meist ist nicht ersichtlich, welche konkreten Auswirkungen eine Zugriffsfreigabe auf Funktionalität beziehungsweise Privatsphäre hat. Viele freie Android-Anwendungen binden Werbebibliotheken ein, die zusätzlich zu den tatsächlich benötigten Berechtigungen weitere Zugriffe zum Sammeln von Informationen sowie zu Werbezwecken anfordern, wie Pearce et al. [184] feststellen. Durch die Einbindung von Werbebibliotheken in Anwendungen erhalten diese Bibliotheken Zugriff mit den für die Anwendung erteilten Berechtigungen und somit möglicherweise auf sensible Daten. Besonders vor Android 6.0 (API Level 23) war es Nutzern und Nutzerinnen ohnehin nicht möglich einzelne Berechtigungen zu verweigern, wodurch sie, wenn sie eine Anwendung verwenden wollten, indirekt dazu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

gezwungen wurden alle angeforderten Berechtigungen zu erteilen. Erst ab Android 6.0 ist es möglich Berechtigungen zur Laufzeit freizugeben oder nachträglich zu verweigern [19]. Abbildung 5.4 zeigt beispielhaft die dynamische Änderung von Berechtigungen. Diese Änderung ermöglicht es dem Benutzer und der Benutzerin Berechtigungen einer Anwendung im Voraus zu beschränken und gegebenenfalls zur Laufzeit kurzfristig zu erteilen. Die Entscheidung von Nutzern und Nutzerinnen über die Bestätigung von Berechtigungsanfragen beruht auf mehreren Faktoren, zum Beispiel dem Anwendungskontext, wie Wijesekera et al. [246] untersuchten.



(a) Kamera Berechtigung zur Laufzeit entziehen (b) Berechtigungsanforderung zur Laufzeit

Abbildung 5.4: Berechtigungsänderungen zur Laufzeit am Beispiel von WhatsApp

5.2.2 Kapselung/Sandboxing von Anwendungen

Bei der Kapselung von Anwendungen, auch *Sandboxing* genannt, geht es um die Trennung voneinander unabhängiger Anwendungen, sodass jede Anwendung ihre eigene abgetrennte Umgebung besitzt. Android setzt die Kapselung von Anwendungen, die Teil der Sandboxing Strategie ist, mithilfe eines Linux basierten User und Gruppen Berechtigungssystem um, wodurch die Isolierung der Anwendungen auf unterster Schicht passiert [88], wie auch die Arbeit von Davi et al. [74] beschreibt. Jede einzelne Android-Anwendung läuft dabei in einem eigenen Prozess mit eigener UID, wodurch ein striktes Berechtigungsmuster durchgesetzt wird und eine klare Trennung und Isolation der Anwendungen sowohl auf Prozess-, als auch auf Datenebene möglich ist. Prozesse sind dadurch eigenständig, haben ihre eigene Umgebung, Struktur, ihr eigenes Dateisystem sowie Speicher oder Ressourcen. Auf diese Weise werden unterschiedliche Anwendungen voneinander abgetrennt und können nicht auf Ressourcen und Daten einer anderen Anwendung zuzugreifen, wenn dies der Nutzer oder die Nutzerin nicht bewusst anhand von Berechtigungen erlaubt, wodurch Android im Bereich Sandboxing das Berechtigungskonzept mit einschließt. Die UID und *Gruppen ID (GID)* einer Anwendung wird vom Android-System während der Installation vergeben, wobei die GID anhand der angeforderten Berechtigungen bestimmt wird.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.2.3 Anwendungsverteilung

Für das Verbreiten von Android-Anwendungen existieren unterschiedliche Ansätze [88]. Die bekannteste Variante ist die Installation von Anwendungen über den Google Play Store [127], die offizielle in Anwendungskategorien unterteilte Plattform zum Verteilen von Android-Anwendungen. Neben dem Google Play Store gibt es noch verschiedene inoffizielle Stores, die ähnlich aufgebaut sind. Darüber hinaus ermöglicht Android die Installation von externen Quellen, wodurch Anwendungen oder vielmehr APK-Dateien, die beispielsweise heruntergeladen oder selbst entwickelt wurden, direkt installiert werden können.

5.2.4 Anwendung/Code Signing

Android-Anwendungen müssen durch Entwickler oder Entwicklerinnen signiert werden, bevor diese über den Google Play Store veröffentlicht und installiert werden können, wie die Android Dokumentation zur Anwendungssignierung [213] beschreibt. Der Signatur-Prozess bei Android-Anwendungen dient dazu, sicherzustellen, dass Updates und Änderungen einer Anwendung nicht von beliebigen Quellen oder genauer Entwicklern und Entwicklerinnen durchgeführt und veröffentlicht werden können. Nur Besitzer und Besitzerinnen des Signaturzertifikats können neue Versionen veröffentlichen, wodurch eine Art *Same-Origin Policy* eingeführt wird [88], [74]. Die Anwendungen müssen allerdings nicht von einer verifizierten *Central Authority (CA)* signiert werden, da Android keine CA Validierung durchführt. Dadurch ist eine Signatur mit bekannten *Third-Party* Zertifikaten oder *Self-Signed* Zertifikaten möglich. Die Android-Anwendungssignierung ist darüber hinaus auch der erste Schritt Richtung Sandboxing und Anwendungsteilung, da anhand der Signatur die eindeutige UID erstellt wird.

5.2.5 Datenverschlüsselung

Die Verschlüsselung von Daten ist ein weiteres wichtiges Sicherheitskonzept von Android, das mit Version 3.0 das erste Mal mithilfe von 128-bit *Advanced Encryption Standard (AES)* [1] eingeführt wurde und benutzerdefiniert aktiviert werden konnte [88]. Dieses Konzept zum vollständigen Verschlüsseln des Speichers wurde durch Android 4.4 und Android 5 noch weiter verbessert und standardmäßig aktiviert wie die Android Dokumentation zur Verschlüsselung [90] beschreibt. Seit Android 7.0 unterstützt Android Verschlüsselung auf der Ebene von einzelnen Dateien, wodurch diese mit unterschiedlichen Schlüsseln verschlüsselt und nach Bedarf auch einzeln wieder entschlüsselt werden können. Mit Android-Version 9 wird eine weitere Verbesserung implementiert, die sich mit der Verschlüsselung von Metadaten beschäftigt. Dies ist allerdings nur bei entsprechender Hardwareunterstützung möglich und kann nur auf neu formatierte Speicher angewendet werden [90].

Um verschlüsselte Daten vor unbefugten Zugriffen zu bewahren, ist der Umgang mit Verschlüsselungskeys entscheidend. Android bietet hierzu verschiedene Möglichkeiten zur Aufbewahrung solch sensibler Schlüssel (siehe Abschnitt 4.2.5). Darüber hinaus ist *White-Box Cryprography* (e.g. [64], [63], [245]) ein Ansatz, bei dem die Sicherheit von Verschlüsselungskeys auch während dem Verschlüsselungsvorgang und unter Whitebox-Attacken bewahrt wird, bei denen der Angreifer vollständigen Zugriff auf das attackierte Gerät beziehungsweise die Anwendung besitzt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6 Sicherheitsstandards von Android-Anwendungen nach OWASP Top 10 und der CERT-Abteilung des Software-Engineering Instituts der CMU

Wie in allen Bereichen der Software-Entwicklung, existieren auch bei der Android-Anwendungsentwicklung definierte Regeln und Richtlinien, um den Sicherheitsstandard moderner State-of-the-Art-Anwendungen zu entsprechen. Dazu wird in diesem Abschnitt auf zwei bekannte Android-spezifische Richtliniendefinitionen eingegangen, die an dieser Stelle einleitend vorgestellt werden.

Das OWASP [229] ist eine weltweite Non-Profit Organisation zur Verbesserung und Standardisierung der Sicherheit von Anwendungen im World Wild Web. OWASP veröffentlicht dazu unterschiedliche Projekte, die sich jeweils auf bestimmte Themenbereiche fokussieren und Grundregeln zu Sicherheitsstandards behandeln. Besonders interessant für diese Arbeit ist das *OWASP Mobile Security Project* [179], dass die Entwicklung und Definition von Sicherheitsstandards für mobile Anwendungen behandelt. In diesem Projekt wird eine Liste von Standards vorgestellt, auch bekannt unter *OWASP Mobile Top 10*, die zehn wichtige Bereiche der mobilen Anwendungsentwicklung abdeckt, Probleme sowie Risiken vorstellt und Empfehlungen zur Verbesserung der Anwendungssicherheit liefert.

Darüber hinaus gibt es von der CERT-Abteilung des Software-Engineering Institut der CMU [44] Entwicklungsregeln und Empfehlungen zur Verbesserung der Sicherheit und Zuverlässigkeit von Software-Produkten. Dabei wird neben unterschiedliche Programmiersprachen wie C, C++ und Java auch speziell auf die Android Entwicklung eingegangen, wodurch acht Android-spezifische Regeln [22] zur sicheren Entwicklung existieren, auf die ebenso wie auf die OWASP Mobile Top 10 im Teil der Arbeit ein Fokus geworfen wird.

Um an dieser Stelle bekannte Sicherheitsdefinitionen diskutieren zu können, haben wir uns zum einen die Zusammenfassung und Erklärung dieser OWASP Mobile Top 10 Richtlinien [173] sowie auch die Android-spezifischen Regeln der CERT-Abteilung des SEI der CMU [22] genauer angesehen.

Die zehn Regeln von OWASP gliedern sich dabei folgendermaßen.

1. Falsche Plattform Verwendung
2. Unsichere Datenspeicher
3. Unsichere Kommunikation
4. Unsichere Authentifizierung
5. Unzureichende Kryptografie
6. Unsichere Autorisierung
7. Schlechte Codequalität

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

8. Codemanipulation
9. Reverse Engineering
10. Irrelevante Funktionalität

Die acht Regeln der CMU, die eine Reihe von Empfehlungen mit ausführlicher Beschreibung zur Einhaltung und Umsetzung dieser enthalten und dazu dienen die Sicherheit in den einzelnen Bereichen zu erhöhen, befassen sich mit den folgenden Teilgebieten.

1. Komponenten Sicherheit
2. Datei I/O und Logging
3. Intent
4. WebView
5. Netzwerk SSL/TLS
6. Berechtigungen
7. Kryptografie
8. Sonstige

Im nachstehenden Teil des Kapitels werden anhand der OWASP Kategorisierung überblicksmäßig sicherheitskritische Probleme erläutert und auf Empfehlungen des OWASP sowie der CMU eingegangen, die durch statische Analyse automatisiert getestet werden können und eine besondere Relevanz für diese Arbeit darstellen.

6.1 Falsche Plattform Verwendung

Die erste OWASP Regel [179] befasst sich mit der sicheren Verwendung von Plattform Funktionen, APIs sowie der IPC und ICC. Gerade die Kommunikation zwischen Komponenten und Prozessen über Intents, können im Android Bereich Sicherheitsprobleme darstellen, wenn sowohl bei der Verwendung, als auch bei der Implementierung bestimmte plattformspezifische Entwicklungs- und Konfigurationspraktiken nicht eingehalten werden. Im Allgemeinen können Plattform Funktionalitäten bereits bekannte Sicherheitslücken [206] beinhalten und bei Missachtung von Dokumentationen zu unerwünschtem Verhalten führen.

Innerhalb der ersten OWASP Regel werden Richtlinien und Empfehlungen zur korrekten Plattform Verwendung genannt, von denen in diesem Abschnitt einige wichtige genannt werden. Für eine vollständige Liste wird auf das Projekt [173], [179] verwiesen.

Die Android eigene Dokumentation beinhaltet des Weiteren eine Sammlung von Security *Best Practices* [29], die sich auf die richtige Verwendung der Android-Plattform und beispielsweise dem korrekten Umgang mit unterschiedlichen Intents oder Berechtigungen spezialisieren. Auf diese Empfehlungen wird in dieser Arbeit allerdings nicht weiter eingegangen.

Die CMU nennt ebenso einige Regeln und Empfehlungen aus dem Bereich der Plattform Verwendung, wobei sich vor allem die Regeln zu Berechtigungen, Intents, Komponenten Sicherheit und WebViews [22] mit diesem Bereich befassen. Daher unterteilen wir die erste OWASP Regel der falschen Plattform Verwendung noch einmal in diese Teilbereiche.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Berechtigungen

Anwendungsentwickler und Entwicklerinnen sollten dem OWASP sowie der Berechtigungsregel der CMU zufolge, besonders darauf achten, die durch Android definierten Regeln zur Berechtigungsdefinition [29] sowie die von der Version abhängigen Implementierung zur Prüfung erteilter Berechtigungen zu befolgen [19]. Weiters sollten nur wirklich benötigte Berechtigungen angefordert und definiert werden, um dadurch Sicherheits- und Privatsphären-Gefährdungen zu minimieren [96]. Während von der Überberechtigung einer Android-Anwendung klare Risiken bezüglich der Privatsphäre eines Nutzers oder einer Nutzerin ausgehen, endet eine Unterberechtigung meist in einem Laufzeitfehler, wie Chester et al. [59] feststellen. Des Weiteren empfiehlt die CMU das explizite Definieren von Berechtigungen für Dateien, die über das *Native Development Kit (NDK)* [17] erstellt werden, da dieses standardmäßig Lese- und Schreibrechte für alle Nutzergruppen erteilt.

DRD20-C. Specify permissions when creating files via the NDK [55].

Intents

Der bewusste Umgang mit Intents beziehungsweise IPC und ICC ist entscheidend, um zu verhindern, dass auf diesem Weg bösartige Anwendungen Zugriff auf sensible Daten erhalten. Besondere Vorsicht ist daher bei der Wahl von impliziten und expliziten Intents geboten, um nicht vertrauenswürdige Komponenten auszuschließen. Das Abfangen von impliziten Intents, Intent Spoofing [62] oder die unerwünschte Kombination von Teilfunktionalitäten unterschiedlicher Anwendungen, wie Elish et al. [89] beschreiben, sind nur Beispiele für mögliche Gefahren im Bereich der Intents. Eine besondere Gefahr geht von *Pending Intents* aus. Dabei handelt es sich um eine besondere Form von Intent, der einen in der Zukunft liegenden Intent samt Berechtigungen der Ursprungsanwendung verpackt. Erhält eine fremde, unerwünschte Anwendung diesen Intent, kann diese die Berechtigungen der Ursprungsanwendung missbrauchen [62], [131].

DRD03-J. Do not broadcast sensitive information using an implicit intent [47].

DRD21-J. Always pass explicit intents to a PendingIntent [56].

Komponenten Sicherheit

Der Zugriff auf Content Provider, die dazu dienen Daten zwischen Anwendungen zu teilen, sollte möglichst limitiert sein, wie die CMU empfiehlt.

DRD01-X. Limit the accessibility of an app's sensitive content provider [45].

Um zu verhindern, dass Services und Aktivitäten öffentlich für Anwendungen zugänglich gemacht werden, wodurch Intent Attacken ermöglicht werden [62], die zu unerwünschtem Informationsfluss führen, sollten Services durch bewusste Definition abgesichert werden. Besonderes Augenmerk sollte auf das Attribut `exported` gelegt werden, um Komponenten als privat zu deklarieren und den Zugriff einzuschränken, wie auch Jiang et al. [146] beschreiben.

DRD07-X. Protect exported services with strong permissions [48].

DRD09. Restrict access to sensitive activities [49].

DRD16-X. Explicitly define the exported attribute for private components [53].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Wichtig ist zu beachten, dass ab Android API-Level 17 Komponenten standardmäßig als privat deklariert sind [88], wenn diese nicht explizit über die Definition im `Manifest.xml` als öffentlich konfiguriert wurden. Bei niedrigeren Android API-Level ist dies genau umgekehrt und Komponenten sind standardmäßig öffentlich zugänglich. Bei API-Level 8 und früher ist keine Einschränkung des Zugriffs möglich.

WebView

Ebenso erwähnt das OWASP Vorsicht im Bereich Eingaben von externen Quellen und empfiehlt das Deaktivieren von JavaScript in WebViews [41], wenn dies nicht ausdrücklich erforderlich ist, um unerwünschte JavaScript Ausführung zu verhindern. Die CMU führt für dieses Teilgebiet ebenso eine eigene WebView Regel [46] an, die selbiges aussagt. Die Arbeiten von Chin et al. [61], Neugschwandtner et al. [174] und Luo et al. [164] beschäftigen sich mit den Gefahren von WebViews und beschreiben einige Angriffsszenarien.

Sonstige Plattform Probleme

Ein weiterer wichtiger Punkt des OWASP im Bereich der Plattform Verwendung bezieht sich auf das Erkennen von *rooted* / *jailbroken* Endgeräten, wodurch sensible Daten einer Anwendung massiv gefährdet werden können und der unbefugte Zugriff anderer Anwendungen auf Dateisystemebene nicht mehr verhindert werden kann.

Darüber hinaus nennt die CMU sowie Felt et al. [98] eine speziell von der W3C Geolocation-API [234] ausgehende Gefahr des unberechtigten Standortzugriffs, die Entwickler und Entwicklerinnen beachten sollten.

> DRD15-J. Consider privacy concerns when using Geolocation-API [52]

Diese Empfehlung beschreibt den korrekten und sicheren Umgang mit der Gelocation-API von W3C, mit der es für Webbrowser möglich ist, direkt über JavaScript auf Standortinformationen eines Geräts zuzugreifen. Bei einer korrekten und den Datenschutzrichtlinien dieser API entsprechenden Verwendung muss, bevor Standortinformationen zur Verfügung gestellt werden, die entsprechende Berechtigung durch den Nutzer oder die Nutzerin des betreffenden Geräts erteilt werden.

6.2 Unsichere Datenspeicher

Die zweite OWASP Regel befasst sich mit der Speicherung sensibler Daten auf mobilen Endgeräten. Gerade portable Geräte besitzen meist besonders sensible und personenbezogene Daten, wodurch der bewusste Umgang mit diesen besonders relevant ist, um zu verhindern, dass diese durch Systemfunktionen wie etwa IPC, ICC, Backups oder falsche Konfiguration sowie durch schadhafte Einflüsse gefährdet werden [250], [40], [74], [251].

Eine Verschlüsselung der Daten vor dem Speichern durch die Anwendung selbst, kann dazu beitragen die Sicherheit zu erhöhen, wie die CMU sowie das OWASP empfiehlt. Allerdings stellen Schlüssel, die zum Verschlüsseln dieser sensiblen Daten verwendet werden und der Anwendung bekannt sind, eine Gefahr dar (siehe Abschnitt 5.2.5). Die effektivste Maßnahme zur sicheren Datenspeicherung ist wirklich nur dann Daten zu speichern, wenn dies unbedingt nötig ist und dabei Verschlüsselungen und Best Practices der Android-Plattform einzusetzen. Dies beinhaltet allerdings oftmals einen *Trade-off* zwischen Sicherheit und Benutzerfreundlichkeit.

OWASP listet für den Fall, dass Datenspeicherung nötig ist einige Grundregeln, die von Entwicklern und Entwicklerinnen beachtet werden sollten [173]. Sensible Schlüssel und Zugangsdaten sollten beispielsweise ausschließlich in den vom System vorgesehenen Credential Stores (siehe Abschnitt 4.2.5) gespeichert werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Weiters nennt OWASP auch Vorsicht im Umgang mit Zwischenspeicher und Caches, um zu verhindern, dass vertrauliche Daten auf diese Weise unverschlüsselt im Speicher liegen. Auch Screens, die wichtige Daten anzeigen, sollten vor Screenshots bewahrt werden. Die CMU nennt die Empfehlung besonders darauf zu achten, den Zugriff auf gespeicherte Daten einzuschränken.

> DRD11. Ensure that sensitive data is kept secure [51].

Diese Empfehlung befasst sich vor allem damit, dass Daten ausschließlich mit dem Modus `MODE_PRIVATE` in Dateien, Shared Preferences oder Datenbanken geschrieben werden, wodurch der Zugriff auf diese beschränkt wird. Des Weiteren sollten Anwendungen keinesfalls Daten vertrauen, die aus `Mode_WORLD_READABLE` Dateien stammen und diese Funktionalität keinesfalls dazu nutzen, Daten zwischen Anwendungen zu teilen, da in diesem Fall keine Zugriffsbeschränkungen möglich sind.

Daten wie Benutzernamen, Passwörter, Cookies, Authentifizierungstoken oder andere persönliche Daten, müssen von Anwendungen mit besonderer Vorsicht behandelt werden und dürfen keinesfalls in SQLite-Datenbanken, Log-Dateien oder als XML in den internen oder gar in den externen Speicher gelangen, wie die Regel zu File I/O und Logging [22] der CMU erläutert. Ebenso vor Synchronisierung und Sicherungen, die möglicherweise sogar in die Cloud transportiert werden, müssen sensible Daten geschützt werden. Im Allgemeinen ist die automatisierte Analyse des Umgangs mit Daten und Logging schwierig, da durch automatisiere Analysewerkzeuge oftmals auch gewünschte Funktionalitäten als Datenlecks erkannt werden [248].

6.3 Unsichere Kommunikation

Bei Anwendungen auf mobilen Endgeräten spielt die Kommunikation zwischen Endpunkten wie mobilen Endgerät und Servern eine bedeutende Rolle. In Bezug auf die Anwendungssicherheit ist der Datenaustausch ein kritisches Unterfangen um Datensicherheit zu gewähren und betrifft das gesamte Spektrum an Kommunikationstechnologien wie etwa NFC, Wifi, Bluetooth, Infrarot und das Mobilfunknetz.

Gerade sensible Daten wie Verschlüsselungsschlüssel, Passwörter, Zugangstoken und private Nutzerdaten stellen nicht nur bei der Datenspeicherung, sondern auch beim Transport über Kommunikationskanäle ein besonderes Risiko dar und müssen sensibel behandelt werden, damit diese nicht in falsche Hände geraten und missbräuchlich verwendet werden können. Dazu empfiehlt es sich Daten ausschließlich über gesicherte *SSL/TLS*-Verbindungen [105], [198] zu übertragen, um sicherzustellen, dass Daten während der Kommunikationsphase nicht von Dritten abgegriffen oder verändert werden können. Wei et al. [240], Fahl et al. [91] und Fahl et al- [92] zeigen weitverbreitete Fehlkonfigurationen und Ursachen auf, die in weiterer Folge zu Sicherheitslücken führen. *Man-in-the-middle* (e.g. [43]) und *Eavesdropping* Attacken (e.g. [161]) sind etwa mögliche Angriffsszenarien, auf die in dieser Arbeit aber nicht näher eingegangen wird. Um solche Angriffe zu verhindern, ist die durchgängige, richtige Verwendung und Konfiguration der gesicherten Kommunikationsverbindungen maßgeblich sowie eine zusätzliche, anwendungsbasierte Verschlüsselung der übertragenen Daten empfehlenswert, um diese ausschließlich gesichert zu übertragen [179].

OWASP nennt dazu einige Punkte, die zur Verbesserung der Anwendungssicherheit im Bereich Kommunikation beachtet werden sollten [173]. Bei der Verwendung von SSL/TLS-Kommunikation nennt das OWASP und die Netzwerk SSL/TLS Regel der CMU [22] vor allem die Validierung der X.509 Zertifikate als wichtigen Punkt zur sicheren Kommunikation. Dabei ist es wichtig, nur validen Zertifizierungsstellen zu vertrauen und wenn möglich anwendungsspezifische zusätzliche Validierung von Zertifikaten einzubauen. Um die Gültigkeit eines Zertifikats zu überprüfen, empfiehlt die CMU die Überprüfung der Signatur, CA, Ablaufdatum des Zertifikats und die URL, für

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

die das Zertifikat ausgestellt wurde. Dazu sollten vertrauenswürdige Standardimplementierungen verwendet beziehungsweise diese nicht durch eigene Implementierungen überschrieben werden.

Gerade für Android-Anwendungen nennt das OWASP zusätzlich die dringende Empfehlung, dass veröffentlichte Anwendungen keinesfalls auf die *Hostname*-Validierung verzichten. Dies passiert durch die Verwendung von `org.apache.http.conn.ssl.AllowAllHostnameVerifier` oder `SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER` [10]. Ebenso, wenn eine Anwendung entgegen der CMU Empfehlung, die Klasse `SSLSocketFactory` überschreibt, muss die Methode zur Server-Zertifikatvalidierung `checkServerTrusted` unbedingt vertrauenswürdig und korrekt implementiert werden.

Zu diesem kritischen Bereich bietet Android als sicher geltenden Empfehlungen und ausführliche Anleitungen mit Beispielimplementierungen zum Umgang mit Zertifikaten, die als Referenz für die Entwicklung verwendet werden sollten [207].

6.4 Unsichere Authentifizierung

Bei mobilen Anwendungen ist oftmals eine Benutzer sowie Benutzerinnen Authentifizierung notwendig, um auf Server, Schnittstellen oder Nutzerdaten zugreifen zu können. Dabei ist es wichtig, dass die Authentifizierung zuverlässig und sicher abläuft, um unbefugten Zugriff auf Daten, Server und Schnittstellen zu verhindern, wie Zuo et al. [253] zeigen. Besonders wichtig ist die Betrachtung von Anwendungen als vollständige, unzuverlässige Client-Software, die etwa durch Binary-Manipulation (siehe Abschnitt 7.1.3) verändert werden kann.

Im Gegensatz zu Webanwendungen, bei denen Nutzer und Nutzerinnen eine durchgängige Internetverbindung benötigen, kann bei mobiler Software oftmals keine durchgängige stabile Internetverbindung garantiert werden. Trotzdem soll die Anwendung in diesem Fall nutzbar bleiben. Um Offlinefähigkeit zu ermöglichen, wird häufig eine alternative Authentifizierungsmethode zur regulären Client-Server Authentifizierung implementiert. Diese kann eine Sicherheitslücke darstellen, da diese alternative Authentifizierungsmethode rein clientseitig passieren muss und somit zum einen durch Manipulation umgangen werden kann und zum anderen entsprechende Daten am Gerät selbst vorhanden sind, um einen Nutzer korrekt zu erkennen [37], [179], [69]. Um fehleranfällige Authentifizierungsmechanismen von Anwendungen aufzudecken, versuchen Tester die Offline-Authentifizierung mittels Binary-Manipulation zu umgehen und in weiterer Folge unbefugte Aktionen durchzuführen.

OWASP listet einige anfällige und als unsicher geltende Authentifizierungsmuster [179], die vermieden werden sollten. Darunter befinden sich etwa folgende Empfehlungen.

a) Werden mobile Anwendungen entwickelt, die auf serverseitige Schnittstellen zugreifen, müssen diese dieselben Authentifizierungsmaßnahmen durchlaufen wie etwa eine Webanwendung, die ebenso auf dasselbe Backend zugreift. Es sollten also keine zusätzlichen, möglicherweise erleichterte und unsichere Authentifizierungsverfahren angeboten werden, nur um die Benutzerfreundlichkeit von mobilen Anwendungen zu verbessern. Die Datensicherheit muss klar im Vordergrund stehen.

b) Wichtige Daten sollten, immer authentifiziert von einem Backend-Server abgefragt werden, um zu verhindern, dass sensible Daten auf nicht vertrauenswürdigen Endgeräten abgespeichert werden.

c) Werden Daten auf einem mobilen Endgerät verschlüsselt gespeichert, empfiehlt OWASP die Verwendung von Schlüsseln, die beispielsweise von einem sicheren geheimen Login Credenti-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

al abgeleitet werden. Dadurch kann garantiert werden, dass der Entschlüsselungsschlüssel nur nach einer erfolgreichen Authentifizierung vorhanden ist.

d) Authentifizierungstoken sollten Gerätespezifisch invalidiert werden können, damit im Falle eines Diebstahls der Authentifizierungstoken eines spezifischen Geräts invalidiert werden kann.

e) Nutzer und Nutzerinnen sollten dazu verpflichtet werden, sichere Passwörter zu verwenden.

f) Entwickler und Entwicklerinnen sollten davon ausgehen, dass clientseitige Authentifizierungsmechanismen jederzeit manipuliert werden können. Serverseitige Neuauthentifizierung, zum Beispiel anhand von Refresh-Token, sollte daher möglichst häufig passieren.

g) Innerhalb des Authentifizierungscodes sollte versucht werden, mögliche Codemanipulationen festzustellen.

6.5 Unzureichende Kryptografie

Der Einsatz von zuverlässiger und sicherer Kryptografie ist von besonderer Bedeutung, um sensible Daten zu schützen. Dabei ist es wichtig, dass keine schwachen Verschlüsselungsalgorithmen verwendet werden, sondern ausschließlich solche, die sämtlichen State-of-the-Art-Sicherheitsstandards (e.g. [34]) entsprechen und von der Sicherheits-Community als unbedenklich betrachtet werden. Das OWASP schließt daher vor allem auch selbst kreierte Verschlüsselungsprotokolle und als unsicher bekannte Verschlüsselungsverfahren wie etwa symmetrische Verschlüsselung mit hart codierten Schlüssel aus, wenn Privatsphäre und Datensicherheit im Vordergrund stehen [179]. Darüber hinaus muss bei der Generierung von Schlüsseln streng auf die Zufälligkeit und Stärke dieser geachtet werden. Neben der Wahl des richtigen Kryptografie-Algorithmus und starken Schlüssels spielt die Verwaltung des Schlüssels eine bedeutende Rolle und stellt für Entwickler und Entwicklerinnen eine Herausforderung dar. Besonders hart codierte Schlüssel sowie Schlüssel, die direkt in Binaries enthalten sind, stellen ein Risiko dar, da sie durch Reverse Engineering Attacken abgegriffen werden können und müssen daher verhindert werden.

Auch die Kryptografie Regel der CMU nennt ähnliche Empfehlungen, wie etwa die DRD18.

> DRD18. Do not use the default behavior in a cryptographic library if it does not use recommended practices [54].

Die CMU referenziert dabei die Regeln von Egele et al. [85].

> Rule 1: Do not use ECB mode for encryption.
> Rule 2: Do not use a non-random IV for CBC encryption.
> Rule 3: Do not use constant encryption keys.
> Rule 4: Do not use constant salts for PBE.
> Rule 5: Do not use fewer than 1,000 iterations for PBE.
> Rule 6: Do not use static seeds to seed SecureRandom(·).

6.6 Unsichere Autorisierung

Im Vergleich zur Authentifizierung, bei der die Identität eines Nutzers oder einer Nutzerin festgestellt wird, geht es bei der Autorisierung um die Rollen und Berechtigungen dieser. Gerade beim Zugriff auf API-Schnittstellen ist eine korrekte Autorisierung wichtig, um Unbefugten den

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Zugriff auf Funktionalitäten und Daten zu verweigern. Ein Beispiel für unsicher Autorisierung ist die Möglichkeit von *Insecure Direct Object Reference* [70] innerhalb einer Anwendung. Dabei können Nutzer und Nutzerinnen beispielsweise durch Veränderung von Request Parametern unbefugte Abfragen tätigen und Zugriff auf sensible Daten erhalten. Eine Gefahr stellen auch Implementierungen dar, die Rollen oder Berechtigungen über ungesicherte Schnittstellen übertragen, auf denen basierend Autorisierungen durchgeführt werden.

Aus diesem Grund befassen sich die beiden Regeln, die OWASP zur Verhinderung unsicherer Autorisierung aufstellt, mit der serverseitigen Berechtigungsprüfung von Clients [173]. Als Erstes nennt OWASP, dass Rollen und Berechtigungen keinesfalls von clientseitigen Informationen abgeleitet werden dürfen, sondern ausschließlich serverseitige Informationen zur Autorisierung herangezogen werden sollten. Des Weiteren müssen serverseitig alle Requests auf korrekte Autorisierung geprüft werden.

6.7 Schlechte Codequalität

Die Codequalität bei der mobilen Anwendungsentwicklung ist wie bei allen Softwareprodukten essenziell, um Sicherheitslücken auszuschließen und Angriffe zu erschweren (e.g. [102] [187], [240], [110]). Dabei ist es wichtig Entwicklungsstandards sowie plattformabhängige Best Practices einzuhalten, um bekannte Risiken wie *Buffer Overflow* (e.g. [215]), *Cross-Side Scripting (XSS)* (e.g. [71]) oder ähnliche Angriffe und Schwachstellen zu verhindern. Android bietet neben bereits erwähnten Best Practices [29] ein Analysewerkzeug [139], um die Codequalität von mobilen Anwendungen zu erhöhen. OWASP erwähnt im Bereich Codequalität, dass die korrekte Signatur einer Anwendung besonders wichtig ist, um nachträglichen Veränderungen entgegenzuwirken [173].

Weiters umfasst das Android Debugging, das dazu verwendet wird, Anwendungen während der Entwicklungsphase zu debuggen und über eine Einstellung im `AndroidManifest.xml` aktiviert sowie deaktiviert wird, einen sicherheitskritischen Bereich, da schadhafte Anwendungen beispielsweise Debug Schnittstellen simulieren, und auf diese Weise Daten abgreifen können. Standardmäßig ist die Debug Funktionalität deaktiviert und wird mit `android:debuggable="true"` aktiviert. Bevor eine Anwendung in Produktion geht und veröffentlicht wird, muss diese Funktionalität deaktiviert werden, wie neben dem OWASP auch eine CMU Empfehlung [50] beschreibt, andernfalls können sensible Informationen abgegriffen werden und in falsche Hände geraten.

6.8 Codemanipulation

Bei clientseitiger Software, die ausschließlich auf unkontrollierten Umgebungen läuft, ist Codemanipulation beziehungsweise das Erkennen von modifizierten Anwendungen ein komplexes und weit verbreitetes Thema im Bereich der Sicherheit, wie etwa Li et al. [160] und Zhou et al. [252] beschreiben. Angreifer versuchen zum Beispiel einer Anwendung schadhafte Funktionalität hinzuzufügen (*Piggybacking*), Sicherheitsmechanismen zu deaktivieren sowie die Systemumgebung eines Gerätes zu verändern und zu manipulieren.

Entwickler und Entwicklerinnen von sensiblen mobilen Anwendungen sollten sich daher die Frage stellen, wie sie nachträgliche Veränderungen ihrer Anwendung zur Laufzeit aufdecken und verhindern können. OWASP listet dazu eine Reihe an unterschiedlichen Möglichkeiten [180]. Ein einfaches Beispiel wäre etwa eine Checksummenprüfung über den ausführbaren Code. Natürlich können Angreifer solche Überprüfungen wieder beeinflussen und manipulieren, allerdings erschweren derartige Maßnahmen dies erheblich. Das OWASP empfiehlt von Anwendung zu Anwendung das Risiko einer Codemanipulation abzuschätzen und abzuwiegen, wie sinnvoll einer Manipulation entgegengewirkt werden kann, beziehungsweise welchen Aufwand dies bedeutet.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Wie genau Risiken zur Codemanipulation eingeschätzt und Veränderungen verhindert werden können, beschreibt das OWASP in einem eigenen Projekt namens *Reverse Engineering and Code Modification Prevention* [181].

6.9 Reverse Engineering

Bei *Reverse Engineering* Angriffen (e.g. [60], [87]) geht es vor allem darum, Informationen über die Implementierung einer Anwendung zu erhalten, mithilfe dieser Sicherheitslücken aufzudecken und in weiterer Folge, möglicherweise auf schadhafte Art und Weise, auszunutzen. Ähnlich wie bei der Codemanipulation ist bei der Suche nach Methoden zur Verhinderung von Reverse Engineering Angriffen sowie für eine Risikoanalyse das OWASP Projekt Reverse Engineering and Code Modification Prevention [181] sowie die OWASP Reverse Engineering Threats [180] empfehlenswert.

6.10 Irrelevante Funktionalität

Die letzte OWASP Regel bezieht sich auf die irrelevanten und versteckten Funktionalitäten von Anwendungen, die durch bestimmte Konfigurationen aktiviert sowie deaktiviert werden können. Gerade bei mobilen Anwendungen kommt es häufig vor, dass Anwendungen zusätzliche Funktionalitäten hinzugefügt werden, die in weiterer Folge zum Beispiel mehr Berechtigungen fordern oder eine Sicherheitslücke inkludieren, ohne dem Nutzer beziehungsweise der Nutzerin einen echten Mehrwert zu bieten (e.g. [96], [97], [33], [184]). Ein Beispiel dafür ist das Hinzufügen von Werbefunktionalität.

Auch Codeteile, die von Entwicklern und Entwicklerinnen während der Entwicklungs- oder Testphase eingebaut und verwendet werden, um Informationen auslesen zu können sowie automatisierte Tests oder Ähnliches zu ermöglichen, stellen in einer Produktivumgebung bedeutende Risiken dar.

Um unnötige und möglicherweise gefährliche Funktionalitäten in Anwendungen zu verhindern, empfiehlt OWASP *Code Review* Prozesse mit einem Schwerpunkt auf Sicherheitsaspekte [179].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7 Identifizierung und Kategorisierung ausgewählter Sicherheitsrisiken zur automatisierten Analyse

Wie dem vorherigen Kapitel entnommen werden kann, existieren sowohl von OWASP, als auch von der CMU einige grundlegende Regeln, Hinweise und Empfehlungen zur Verbesserung der Sicherheit von Android-Anwendungen, die von Entwicklern und Entwicklerinnen eingehalten werden sollten.

Für diese Arbeit ist es wichtig, herauszufinden welche Schwachstellen und Fehlkonfigurationen einer Anwendung automatisiert analysiert und kategorisiert werden können, um anschließend Aussagen über die Sicherheit einer Anwendung treffen zu können. Dabei ist zum einen die Abwägung zwischen Analyseaufwand und der von einer Schwachstelle ausgehenden Gefahr wichtig und zum anderen das Erkennen von wirklich kritischen Schwachstellen und Schwachstellen, die sensible Daten zugänglich machen aber trotzdem gewünschtes Verhalten darstellen. Automatisiert zu analysieren, welche Anwendungen Standortinformationen preisgeben ist beispielsweise trivial, herauszufinden ob dies gewünschtes Verhalten einer Anwendung darstellt oder schadhaften, unerwünschten Hintergrund besitzt, ist allerdings automatisiert und ohne genaueren Funktionsanalyse nicht mehr möglich.

Um die Sicherheit von Android-Anwendungen einzustufen, wurden in Anlehnung an die Arbeit von Gadient [107] fünf Sicherheitskategorien eingeführt, denen einige Punkte der CMU Entwicklungsregeln und OWASP Top 10 zugeordnet wurden, die automatisiert analysiert werden können, kritische Bedeutung haben und zur Bestimmung der Anwendungssicherheit herangezogen werden können. In diesem Kapitel werden die fünf Kategorien samt Unterkategorien definiert, um darauf aufbauend im nächsten Kapitel entsprechende Analysewerkzeuge zu identifizieren, auf deren Ergebnis die Sicherheitseinstufung von Android-Anwendungen durchgeführt werden kann.

7.1 Mangelnder Angriffsschutz

Die erste Kategorie befasst sich mit dem mangelnden Schutz, dem vor allem die OWASP Regeln der falschen Plattform Verwendung und schlechten Codequalität zugrunde liegen. Ziel ist es, allgemeine Fehlkonfigurationen, unachtsame Implementierungen und Sicherheitsprobleme sowie die Verwendung von veralteten API-Versionen, Bibliotheken und Funktionalitäten, gerade auch im Zusammenhang mit älteren Android-Versionen aufzudecken. Dabei wird diese Kategorie in die nachfolgenden vier Teilbereiche geteilt.

7.1.1 API-Versionen, Bibliotheken und Drittquellen

Wie in allen Softwarebereichen ist es wichtig aktuelle Softwareversionen einzusetzen, um das Risiko bestehender Sicherheitslücken zu reduzieren. Viele Android Geräte verfügen über eine veraltete Betriebssystemversion, da speziell ältere und Hardware schwächere Geräte von neuen Betriebssystemversionen nicht mehr unterstützt werden und Gerätehersteller Android Sicherheitsupdates meist stark verzögert veröffentlichen, wie Thomas et al. [231] beschreiben. Gerade von veralteten Betriebssystem Versionen geht eine besonders große Gefahr aus Opfer von Angriffen

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

zu werden, da Google den Support von älteren Versionen im Laufe neuer Releases nach und nach einstellt und somit keine Sicherheitsupdates mehr liefert. Durch den auslaufenden Support kann es vorkommen, dass ältere API-Versionen noch Sicherheitslücken, Implementierungen oder Pakete besitzen, welche in späteren Versionen bereits behoben oder als *deprecated* markiert wurden [206]. Auch durch Android versionsabhängiges Standardverhalten oder neuen Funktionen, kann beispielsweise das Hinzufügen von Abwärtskompatibilität einer Anwendung zu groben Auswirkungen auf die Anwendungssicherheit führen.

Das OWASP erwähnt in ihrer ersten Regel [179] die besondere Wichtigkeit, Plattform und versionsspezifische Best Practices zu befolgen. Aus diesem Grund wird bei der nachfolgenden Werkzeugwahl für die Analyse dieser Kategorie besonders darauf geachtet, Werkzeuge zu selektieren, die auch versionsspezifische Fehlimplementierungen und Konfigurationen analysieren (e.g. AndroBugs [111], QARK [124]).

Durch die zunehmenden Komplexität und Funktionalität von Anwendungen, ist es für Entwickler und Entwicklerinnen wichtig auf Bibliotheken zurückzugreifen, welche gewünschte Funktionalitäten bereits implementieren. Dabei stellen diese Bibliotheken aber nicht nur eine Erleichterung dar, sondern auch eine Gefahr unerwünschtes Verhalten oder Schadsoftware unbewusst einzubinden und somit ganze Anwendungen und Systeme angreifbar zu machen [107]. Es sollte daher genauestens darauf geachtet werden, wie vertrauenswürdig eingebundene Bibliotheken tatsächlich sind, ob diese aus zuverlässigen Quellen stammen und keine Schadsoftware oder bekannten Lücken beinhalten. Für die Sicherheitseinstufung ist es uns daher ein Anliegen, auch die in Android-Anwendungen inkludierten Bibliothek zu analysieren und in die Sicherheitseinstufung mit aufzunehmen.

7.1.2 Nativer Quellcode

Der zweite Teilbereich des mangelnden Angriffsschutzes befasst sich mit nativen Codesegmenten innerhalb von Android-Anwendungen. Gerade bei rechenintensiven Operationen ist die Verwendung des NDK und das Einbinden von native C/C++ Codeteilen eine häufige Herangehensweise (e.g [156], [225]). Die Kehrseite der nativen Codeteile ist allerdings, dass diese anfälliger gegen Angriffe auf Berechtigungen oder Buffer Overflow Attacken sind. Solche Schwachstellen können von Angreifern und Angreiferinnen ausgenutzt werden, um unerwünschten Code ausführen, Systemzugriff erhalten und Daten stehlen zu können. Wie in Abschnitt 6.1 beschrieben, besteht besonders beim Rechtekonzept des NDKs Handlungsbedarf, da hier standardmäßig Lese- und Schreibrechte für alle Nutzergruppen erteilt werden, was dem Android Rechtekonzept widerspricht [22], [107] [96], wie auch Faruki et al. [94] beschreiben.

Die Verwendung von nativen Quellcode innerhalb einer Anwendung stellt demnach ein Sicherheitsrisiko dar, wenn keine zusätzlich Funktionalität zur Rechteverwaltung vorhanden ist. Wird von einem in das Framework inkludierten Analysewerkzeugen die Verwendung von nativen Codesegmenten innerhalb einer Anwendung identifiziert, stellt dies ein mögliches Sicherheitsrisiko dar, wodurch eine mobile Anwendung nicht als bedenkenlos eingestuft werden kann.

7.1.3 Repacking

Reverse Engineering und Repacking von Anwendungen ist eine weitere Gefahr von mobilen Anwendungen [160], [252], [94]. Beim Repacking werden Anwendungen durch einfaches disassemblieren, modifizieren und anschließend durch re-assemblieren verändert, wodurch Angreifer und Angreiferinnen Anwendungen beispielsweise analysieren und das ursprüngliche Verhalten manipulieren oder schadhafte Funktionalität hinzufügen können. Durch Repacking wird mehr als 80% der am Markt verfügbaren Malware erstellt [94]. Um unbefugte Analyse einer Anwendung zu erschweren, können unterschiedliche sogenannte *Obfuscation*-Mechanismen (e.g. [65], [154])

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

eingesetzt werden. Dabei wird die Lesbarkeit durch Mechanismen, wie dem Umbenennen von Variablen und Methoden oder Verschlüsselung erschwert.

Ein mögliches Indiz für Repacking, schlechte Codequalität beziehungsweise die unvertrauenswürdige Herkunft einer Anwendung ist das Anwendungszertifikat [107], das zur Signatur verwendet wurde, wie das OWASP [179] im Bereich schlechte Codequalität erwähnt. Besonders Anwendungen mit Debug Zertifikaten, die vom Google Play Store nicht akzeptiert werden und somit von alternativen Drittanbietern stammen, stellen eine besonders Gefahr sowie ein Indiz für eine modifizierte Anwendung dar. Daher wird ein Werkzeug benötigt, dass die Korrektheit der Signatur einer Anwendung untersucht. Die Verwendung von Self-Signed sowie Debug Zertifikaten oder unsicheren Zertifikat Schlüsseln stellt in Folge eine Sicherheitslücke dar.

7.1.4 Berechtigungen

Damit Anwendungen auf bestimmte APIs und Daten zugreifen können, müssen diesen explizit Berechtigungen erteilt werden, die von einem Anwendungsentwickler oder einer Entwicklerin im `AndroidManifest.xml` definiert werden. Dabei stellt eine mögliche Überberechtigung einer Anwendung eine Gefahr für den Nutzer oder die Nutzerin und in weiterer Folge für die sensiblen Daten dar [107], [94]. Es ist daher wichtig, dass tatsächlich nur Berechtigungen angefordert und erteilt werden, die eine Anwendung wirklich für ihre implementierte Funktionalität benötigt, wie auch das OWASP in ihrer ersten Regel beschreibt. Darüber hinaus existiert gerade im Bereich von Berechtigungsdefinition, Erteilung und Überprüfung eine Reihe von Möglichkeiten zur Fehlimplementierung. Vor allem auch in Zusammenhang mit versionsbedingten unterschiedlichen Standardverhalten und der ab Android 6 verfügbaren Berechtigungserteilung zur Laufzeit, wie Colton et al. [66] erklären. Auch Berechtigungen, die von Android als gefährlich eingestuft werden [19], sollten nicht leichtfertig angefordert werden, wenn die dahinter liegende Funktionalität beispielsweise durch einen Intent-Aufruf durchgeführt werden kann.

In diesem Bereich der Sicherheitsanalyse liegt der Fokus daher besonders auf dem Ergebnis von Analysewerkzeugen, die sich mit Fehlimplementierungen und Best Practice Abweichungen im Bereich der Berechtigungsanforderung, Erteilung und Überprüfung beschäftigen.

7.2 Sicherheitsinvalidierung im Zusammenhang mit Zertifikaten, Kryptografie und Anwendungsverteilung

Die zweite Sicherheitskategorie befasst sich mit Sicherheit in Bezug auf Verschlüsselung und Vertraulichkeit. Dabei wird vor allem auf die OWASP Regeln der unsicheren Kommunikation und Kryptografie eingegangen.

7.2.1 Kryptografie

Die Verschlüsselung beim Speichern und Übertragen von Daten stellt eine besondere Relevanz dar, um diese vor unerwünschten Zugriffen zu schützen. Dazu gibt es verschiedene Verschlüsselungsarten (e.g. [35], [106], [214]), die in symmetrische und asymmetrische eingeteilt werden und verschiedene Algorithmen wie etwa Base64 [147], MD4 [199], MD5 [200], SHA1 [100] oder SHA256 [101] besitzen [107]. Bei der Verschlüsselung von Daten ist es wichtig, empfohlene Sicherheitsstandards einzuhalten und als sicher geltende Algorithmen einzusetzen. Ein häufiger Fehler ist die Verwendung von schwachen oder unsicheren Verschlüsselungsalgorithmen wie etwa MD4, MD5, SHA1 sowie Base64, oder die unsichere Konfiguration dieser. Eine wichtige wissenschaftliche Arbeit im Bereich Kryptografie stammt von Egele et al. [85], in welcher sechs Regeln (siehe Abschnitt 6.5) der Kryptografie definiert werden. So stellt etwa die Verwendung des

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

ECB-Mode, CBC-Mode mit unsicheren IV, konstante sowie schwache Passwörter und Salts eine Sicherheitslücke dar.

Um Sicherheitsprobleme und Fehlkonfigurationen von Android-Anwendungen im Bereich Kryptografie aufzudecken, liegt der Fokus in diesem Bereich auf Werkzeugen, welche die Verwendung schwacher Verschlüsselungsverfahren sowie Verstöße gegen die Regeln von Egele et al. untersuchen.

7.2.2 Zertifikatvalidierung

Um eine sichere Kommunikation zwischen Endpunkten sicherzustellen, ist es unumgänglich ausschließlich gesicherte SSL/TLS-Verbindungen zu verwenden. Dazu werden serverseitige Zertifikate verwendet, um den Kommunikationspartner zu identifizieren und verschlüsselte Kommunikation sicherzustellen [105]. Um diese Zertifikate zu validieren, wird unter anderem überprüft, ob Zertifikate von bekannten CAs, die am Endgerät installiert und dadurch bekannt sind, ausgestellt wurden und somit als vertrauenswürdig gelten. Diese CAs werden unter Android von einem sogenannten `TrustManager` verwaltet [208]. Manche serverseitigen Anwendungen nutzen self-signed Zertifikate, die nicht durch bekannte CAs erstellt wurden. In diesem Fall müssen Entwickler und Entwicklerinnen die Validierung dieses Zertifikats selbst übernehmen (e.g [208], [107]), wobei es wichtig ist, dass dies nach empfohlenen Android Sicherheitsstandards, beispielsweise durch einen eigenen TrustManager dem das self-signed Zertifikat bekannt ist, geschieht. Bei fehlerhafter Implementierung der Zertifikatvalidierung bleibt die Kommunikation zwischen den Endpunkten unverschlüsselt, unauthentifiziert und somit unsicher.

Bei der Sicherheitseinstufung dieses Bereichs, werden vorliegende Analyse-Reports hinsichtlich der korrekten Zertifikatvalidierung, dem Ignorieren von SSL-Zertifikat Fehlern sowie der Akzeptanz von self-signed Zertifikaten und inkorrekten Hostnamen betrachtet.

7.2.3 Anwendungsverteilung und Anwendungsmodifizierung

Für die Verteilung von Android-Anwendungen existieren neben den offiziellen Google Play Store noch weitere Möglichkeiten der mobile Anwendungsverteilung [88], [210], [107]. Es ist beispielsweise möglich, Anwendungen von externen Quellen direkt über eine gedownloadete APK-Datei oder alternativen inoffiziellen Stores zu installieren. Gerade bei externen Quellen und alternativen Stores existieren oftmals keine Sicherheitsmaßnahmen und Zertifikatvalidierungen, um Veränderungen von Dritten und die Verbreitung von sicherheitskritischen sowie schadhaften Anwendungen zu verhindern, und werden daher zum Teil bewusst für die Verteilung von schadhaft modifizierten Anwendungen verwendet. Dabei werden reguläre Anwendungen durch einfaches Repacking verändert.

Die Anwendungsinstallation von externen Quellen oder nicht offiziellen Stores stellt daher ein besonders hohes Risiko für Sicherheit und Privatsphäre eines Nutzers oder einer Nutzerin dar. Bei der automatisierten Analyse von Anwendungen ist es allerdings kaum mehr möglich, die Herkunft einer Anwendung zu bestimmen. Einzig mobile Anwendungen mit einer Debug-Zertifikat-Signatur, die vom Google Play Store nicht akzeptiert wird, wie die Android eigene Dokumentation [213] beschreibt, kann als Indiz für eine unsichere Herkunft der Anwendung verwendet werden. Wird durch die automatisierte Anwendungsanalyse ein Debug-Zertifikat identifiziert, kann die Herkunft der Anwendung durch das Framework nicht mehr als unbedenklich eingestuft werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.3 Anwendungsübergreifende Zugriffskontrolle

Die Zugriffskontrolle ist die dritte Kategorie unserer Sicherheitseinstufung und befasst sich mit dem Datenfluss zwischen Komponenten beziehungsweise der Kommunikation und dem Datenaustausch zwischen verschiedenen Anwendungen.

7.3.1 Inter-Component Communication (ICC)

Bei der Kommunikation und der Wiederverwendung von Komponenten gibt es den sogenannten *Component Hijacking* Angriff, bei denen bösartige Anwendungen versuchen, Komponenten von höher privilegierten Anwendungen zu missbrauchen, um Zugriff auf sensible Daten zu erhalten [98], [62]. Dies ist nur dann möglich, wenn Komponenten wie Aktivitäten, Services, Content Provider oder Broadcast Receiver exportiert und somit für andere Anwendungen verfügbar gemacht werden und die definierten Komponenten über inkorrekte Berechtigungsprüfungen verfügen [107] oder andere Schwachstellen enthalten. Um zu verhindern, dass Anwendungen Komponenten missbräuchlich wiederverwenden können, sollten diese nur wenn wirklich explizit gewollt, durch einen Intent Filter oder genauer durch das Attribut `android:exported = true` für andere Anwendungen zugänglich gemacht werden.

Bei der Verwendung von Komponenten aus einer fremden Anwendung heraus, ist es aus diesem Grund auch besonders wichtig, dass die Berechtigungen der aufrufenden Client Anwendung überprüft werden. Dazu müssen Komponenten mit den für die Verwendung benötigten Berechtigungen im `AndroidManifext.xml` definiert werden, wie in Abschnitt 4.2.2 erklärt.

Um innerhalb der Framework Analyse den Bereich der Zugriffskontrolle abzudecken, ist eine genauere Analyse der Definitionen im `AndroidManifest.xml` besonders relevant, wie auch in den nächsten Teilabschnitten zur Sicherheit im Umgang mit Intents und der Android Debug Funktionalität aufgezeigt wird.

7.3.2 Intent

Mobile Android-Anwendungen können mithilfe von Intents auf die Funktionalität von anderen Anwendungen zugreifen. Dazu existieren implizite Intents, bei denen jede beliebige verfügbare Anwendung die Durchführung einer Funktion übernehmen kann, und explizite Intents, bei denen der Entwickler und Entwicklerinnen explizit eine Implementierung einer Funktion wählt. Die Gefahr oder das Risiko, das durch die Verwendung von impliziten Intents entsteht, ist durch das Verwenden von möglicherweise schadhaften Anwendungen und Implementierungen für eine bestimmte Aufgabe gegeben, wodurch durch einen impliziten Intent sensible Informationen in falsche Hände geraten können. Eine weitere Gefahr geht von empfangenen Intents aus, bei denen eine schadhafte Anwendung beispielsweise böswillig Intents an eine Zielanwendung sendet, welche die gewünschte Funktionalität durchführt und sicherheitskritische Informationen preisgibt [62], [107]. Des Weiteren stellen Pending Intents, die einen in der Zukunft liegenden Intent samt Berechtigungen der Ursprungsanwendung verpacken, eine Gefährdung der Privatsphäre von Endnutzern und Nutzerinnen dar. Wird dieser Pending Intent an andere, dritte Anwendungen weitergereicht, haben diese die Möglichkeit jederzeit den erhaltenen Intent mit den Berechtigungen der Ursprungsanwendung auszuführen [62], [131], [88]. Ein besonderes Risiko stellen implizite Pending Intents dar, da auf diesem Weg von unbekannte Anwendungen, Codeteile mit den erteilten Berechtigungen der Ursprungsanwendung ausgeführt werden.

Um eine sichere Implementierung einer Anwendung zu gewährleisten und Risiken auszuschließen, empfiehlt es sich vorwiegend explizite Intents zu verwenden, um Funktionalität nur an vertrauenswürdige Anwendungen auszulagern, wie das OWASP und die CMU im Bereich der Intent Empfehlungen (siehe Abschnitt 6.1) beschreiben. Werden Komponenten beziehungsweise Funkti-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

on anderer Anwendungen verwendet, ist eine Validierung des Rückgabewerts oder des Ergebnisses dringend notwendig.

Automatisierte Analysewerkzeuge in diesem Bereich sowie darauf aufbauende Sicherheitseinstufungen von Android-Anwendungen sind daher äußerst relevant.

7.3.3 Debuggable Release

Die Aktivierung der Android Debug Funktionalität ermöglicht es Angreifern und Angreiferinnen, über die *Android Debug Bridge (ADB)* [11] Zugriff auf sämtliche Daten einer Anwendung zu erhalten und Code mit den Berechtigungen der Zielanwendung auszuführen, wie auch aus dem OWASP [179] hervorgeht.

Gadient beschreibt dazu folgendes Angriffsszenario. „Apps shipped with debugging enabled always try to connect to a local Unix socket opened by the Android Debug Bridge (adb). While adb is not running on every consumer device, a malign app could disguise itself as an adb service and connect to random debuggable apps. Consequently, a malicious app is able to gain full access to the Java process and can execute arbitrary code in the context of the debuggable app.“ [107]

Dieser Gefahr kann durch deaktivieren des Debug-Modus vor der Veröffentlichung einer Anwendung mithilfe der Einstellung `android:debuggable = "false"` entgegengewirkt werden, die bei der Sicherheitseinstufung von Android-Anwendungen beachtet werden sollte.

7.4 Datenlecks – Unbefugter Zugriff auf sensible Informationen

Die vorletzte Kategorie der Sicherheitsevaluierung befasst sich sowohl mit dem Zugriff auf, als auch mit dem Umgang von sensiblen Daten, Metadaten sowie Geräteinformationen und Gefahren von Datenlecks in Android-Anwendungen.

7.4.1 Sicherheitslücken bezüglich Hardware-, Softwareidentität sowie unerwünschten Zugriff auf sensible Daten

Anwendungen, die Hardware Identifikationsnummern und Adressen wie etwa die *IMEI*-Nummer oder die *MAC*-Adressen von WLAN- und Bluetooth-Modulen verwenden, um User beziehungsweise Geräte zu identifizieren, verstoßen nicht nur gegen die von Android empfohlenen Best Practices [36], sondern stellen auch eine Gefahr der anwendungsübergreifenden Nutzer und Nutzerinnen sowie Aktivitäten Verfolgung und Aufzeichnung dar. Darüber hinaus kann die eindeutige IMEI beispielsweise dazu verwendet werden, infizierte Geräte innerhalb eines *Command and Control* Systems zu registrieren, wie Wei et al. [238] erwähnen. Ab Android 6.0 sind die MAC Adressen von WLAN und Bluetooth Modulen nicht mehr zugänglich [9] und der Zugriff auf die IMEI-Nummer benötigt ab Android 10 die Berechtigung `READ_PRIVILEGED_PHONE_STATE`, die nicht an Third-Party Google Play Store-Anwendungen vergeben werden kann [20].

Erkennt ein Analysewerkzeug die Verwendung solcher Identifikatoren, ist ein eindeutiger Verstoß gegen empfohlene Standards und in weiterer Folge schlechte Codequalität beziehungsweise eine mögliche Sicherheitslücke ableitbar.

7.4.2 Unsichere Übertragungskanäle

Dass Zertifikate und die Validierungen dieser eine tragende Rolle bei gesicherten Transportprotokollen darstellen, wurde bereits im Bereich Zertifikatvalidierung (siehe Abschnitt 6.3) beschrieben. Darüber hinaus stellen ungesicherte und unverschlüsselte Datentransportprotokolle wie HTTP

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[99] oder FTP [188] ein besonderes Sicherheitsrisiko bei der Übertragung von Nutzlast dar. Es besteht die Gefahr, dass Angreifer Daten ohne großen Aufwand mitlesen, abgreifen und verändern können [107]. Um zu verhindern, dass sensible Daten in die falschen Hände geraten, müssen unverschlüsselte Übertragungen sowie APIs, welche unsichere Verbindungen herstellen, verhindert werden. In heutigen modernen Anwendungen sind unverschlüsselte Kommunikationskanäle ein absolutes No-Go. Daher wird bei der anschließenden Werkzeugrecherche verstärkt darauf Rücksicht genommen, den sicherheitskritischen Bereich der Zertifikatvalidierung und Einhaltung von gesicherten Verbindungen in die Sicherheitseinstufung von Anwendungen mit aufzunehmen.

7.4.3 Unsichere Datenspeicherung

Bei Android-Geräten gibt es verschiedene Möglichkeiten Daten abzuspeichern, wie Abschnitt 4.2.5 zeigt. Das Android Rechtekonzept ermöglicht das bewusste Speichern von Daten mit den Berechtigungen `MODE_WORLD_READABLE` und `MODE_WORLD _WRITEABLE` im internen Speicher, wodurch auf diese Weise gespeicherte Daten allgemein zugänglich sind. Dies stellt in weiterer Folge eine Gefahr für die Privatsphäre dar. Ebenso problematisch ist die Verwendung des externen Speichers, der keinerlei Schutz gegen unbefugten Zugriff bietet. Aber auch SQLite-Datenbanken und Shared Preferences dürfen keinesfalls als sicher betrachtet werden [210], [250], [40], [74]. Gerade bei gerooteten Geräten besteht die Gefahr von unbefugten Zugriffen auf private Daten. Vor allem bei Verlust eines mobilen Endgeräts.

Um möglichen Gefahren entgegenzuwirken, ist bewusster Umgang mit Daten und Sicherungen im Allgemeinen sowie eine starke Verschlüsselungen wichtig. Darüber hinaus ist das öffentlich zugängliche Abspeichern von Daten zu verhindern, wie das OWASP und die CMU angeführt haben. In diesem Bereich liegt der Fokus auf Werkzeuge, die Sicherheitsprobleme von Android-Anwendungen wie etwa das öffentlich zugängliche Speichern von Daten, Sicherungen oder temporären Dateien aufdecken, um in weiterer Folge den sicherheitskritischen Bereich der Datenspeicherung in die Sicherheitseinstufung mit aufzunehmen.

7.4.4 Unsichere Aufbewahrung von Zugangsdaten

Ein weiterer kritischer Bereich ist die Speicherung von Zugangsdaten, Zertifikaten, private Schlüsseln und ähnlichen hochsensiblen Daten. Solche sensiblen Daten sollten keinesfalls ungeschützt im Speicher oder hart codiert in Codeteilen landen. Spezielle Android `KeyStore` Implementierungen, die durch Hardwareschlüssel, Zertifikate oder durch ein benutzerdefiniertes Passwort geschützt sind, das nicht am Gerät abgespeichert wird, stellen beispielsweise eine geeignete Methode zum Abspeichern sensibler Daten dar [107], [15]. Bei der Sicherheitseinstufung von Anwendungen liegt das Hauptaugenmerk auf Analyseergebnisse von Werkzeugen in Bezug auf die richtige und sichere Konfiguration sowie Verwendung der `KeyStore`-Implementierungen.

7.5 Eingabevalidierung

Eingabevalidierung egal welcher Art ist eine sicherheitsrelevante Funktionalität einer Anwendung und dient dazu, einerseits Datenqualität zu gewährleisten und andererseits unerwünschten Verhalten beziehungsweise Angriffen entgegenzuwirken, wie auch die OWASP Regeln zu schlechter Codequalität [179], bei denen vor allem dynamisch geladene Codeteile, XSS- [71] und *SQL-Injection*-Attacken [221] im Vordergrund stehen. Der Eingabevalidierungsbereich der Sicherheitseinstufung beschäftigen sich mit diesen Gefahren.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

7.5.1 WebView und XSS

Die WebView Komponente ermöglicht es, HTML Webinhalte innerhalb einer Anwendung dynamisch anzuzeigen, wie die Android Dokumentation zu WebViews [41] beschreibt. Ist die JavaScript Verarbeitung der WebView Komponente aktiviert, stellt diese bei fehlender Eingabevalidierung ein Risiko für XSS-Attacken dar, bei denen Angreifer und Angreiferinnen schadhafte Codeteile eingeben, die anschließend verarbeitet werden [71]. Auf diese Weise können Angreifer unbefugten Zugriff auf dem System erhalten, auf dem die Eingabe verarbeitet wird. Im Fall von mobilen Anwendungen kann dies dazu führen, dass Zugriff auf Anwendungsdaten und das lokale System gewährt wird, wodurch beispielsweise SMS versendet oder ähnliche Dienste eines mobilen Endgeräts missbraucht werden können [210].

7.5.2 SQL-Injection

Eine ähnliche Attacke, die auf fehlerhafter oder fehlender Eingabevalidierung basiert, ist die SQL-Injection-Attacke, bei der unvalidierte Eingaben dazu verwendet werden, Datenbankabfragen zu erstellen [221]. Auf diese Weise können unerwünschte SQL-Befehle eingeschleust werden, wodurch Angreifer und Angreiferinnen je nach Berechtigung der angegriffenen Anwendung die Möglichkeit haben, auf angebundene Datenbanken zuzugreifen und Daten zu lesen oder zu verändern [107].

In der nachfolgenden Werkzeugrecherche wird daher auch verstärkt nach Analysewerkzeugen gesucht, die den Bereich der WebView- sowie XSS- und SQL-Injection-Analyse beinhalten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8 Ausgewählte Open-Source-Werkzeuge zur Sicherheitsanalyse von Android-Anwendungen

Um die definierten Sicherheitsbereiche einer Anwendung zu analysieren und anschließend eine Sicherheitseinstufung durchführen zu können, wurde im Zuge dieser Arbeit das aktuelle Angebot an statischen Analysewerkzeugen untersucht, um für das Framework relevante Werkzeuge zu identifizieren. Dazu wurde die Funktionsweise von ausgewählten und durch die Literatur (e.g [204], [237], [138], [39]) bekannten Werkzeuge analysiert und getestet. Bereits nach kurzer Recherche stellte sich wie erwartet heraus, dass eine Vielzahl unterschiedlicher Android-Anwendungsanalyse und Reverse Engineering Werkzeuge oder Frameworks existieren, deren Architektur, Funktionsumfang sowie Arbeitsweise sich zum Teil stark voneinander unterscheiden. Die Erfahrungen, die bei der Untersuchung der gewählten Werkzeuge gesammelt wurden, werden in diesem Kapitel diskutiert, um auf diesem Weg vor allem auch anderen künftigen Arbeiten einen raschen Überblick über gute, empfehlenswerte Werkzeuge und Frameworks zur Android-Anwendungsanalyse bieten zu können.

Trotzt unterschiedlichen Funktionsumfang haben all die statischen Analysewerkzeuge allerdings eines gemeinsam. Vor der Analyse müssen die fertig gebauten Android-Anwendungen in les- beziehungsweise analysierbaren Code dekompiliert werden. Dazu werden im ersten Teil dieses Kapitels einige Hilfswerkzeuge vorgestellt.

8.1 Reverse Engineering und Reengineering Werkzeuge

Bei der Auswahl von *Assembler/Disassembler*- und *Dekompilier*-Werkzeugen, wurde auf der *Android Security Awesome* Liste von Bhatia [204] aufgebaut, die neben einer Vielzahl wertvoller Android-Analysewerkzeuge und Frameworks sowie wissenschaftlichen Veröffentlichungen im Bereich Android Sicherheit, auch einige Hilfswerkzeuge zum Dekompilieren und Disassemblieren enthält.

8.1.1 Assembler/Disassembler

Disassembler beschreiben die Funktionalität der Konvertierung von Maschine Code in für Menschen lesbaren und analysierbaren Code. Assembler hingegen befassen sich mit dem umgekehrten Vorgang.

Smali/Baksmali

Smali/Baksmali [126] ist ein Assembler/Disassembler für DEX. Dieser ermöglicht es kompilierten Android-Quellcode, der im DEX-Format vorliegt, in lesbaren *Smali*-Code umzuwandeln, der auf *Jasmin*-Syntax [142] basiert. Vorteil von Smali beziehungsweise des umgekehrten Vorgangs namens Baksamli ist, dass beim Disassemblieren von DEX in Smali-Syntax beinahe originale

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Quellen erzeugt werden, diese auf Assembler-Ebene verändert und anschließend wieder zu funktionierenden DEX zusammengebaut werden können, wodurch einfaches modifizieren von Anwendungen möglich ist, wie etwa Faruki et al. beschreiben [95].

ApkTool

Das ApkTool [117] ist ein sehr bekanntes und weit verbreitetes Reverse Engineering Werkzeug für Android-Anwendungen und baut auf Smali sowie Baksmali auf. Eine der Hauptfunktionen ist das Entpacken beziehungsweise die Disassemblierung von APK-Dateien in lesbaren Code, der in Smali-Format dargestellt wird. Dieser Smali-Code kann durch das ApkTool auch in modifizierter Form wieder in eine vollwertige Anwendung zusammengebaut werden. Somit ist es nicht nur möglich, die Funktionsweise einer Anwendung genau unter die Lupe zu nehmen, sondern diese auch zu verändern und neue, veränderte Funktionen hinzuzufügen. Durch diese Funktionalität wurde der Grundstein für statische Codeanalyse gelegt, wodurch eine Vielzahl an Reverse Engineering und statischen Analysewerkzeugen (e.g. ApkAnalyser [116], QARK [124], M-Perm [59]) auf ApkTool aufbauen.

8.1.2 Java Decompiler

Liegen kompilierte Java-Anwendungen vor, können diese durch einen Decompiler in Java-Quellcode dekompiliert werden. Der erhaltene Quellcode kann anschließend durch Java-Analysewerkzeuge weiter analysiert werden. Einige Werkzeuge, die Android-Anwendungen beziehungsweise DEX in Java-Quellcode dekompilieren, werden an dieser Stelle vorgestellt.

dex2jar

Dex2jar [118] ist ein Decompiler, der DEX in JAR dekompiliert. Dieses erhaltene Java Archiv kann im Anschluss zum Beispiel mit JD-Core, auf das anschließend eingegangen wird, weiter zu lesbaren Quellcode verarbeitet werden.

Enjarify

Enjarify [120] stellt eine Alternative zu dex2jar dar und dekompiliert ebenso Dalvik-Bytecode in Java-Bytecode, wodurch eine anschließende Analyse mit Java-Analysewerkzeugen und Frameworks möglich wird. Enjarify stellt eigenen Angaben zufolge, eine Verbesserung zu dex2jar dar, da der Dekompilierungsprozess auch in Fällen bei denen dex2jar scheitert erfolgreich korrekten Java-Bytecode erstellen kann.

JD-Core

JD-Core ist Teil des Java Decompiler (JD) Projekts [144] und stellt die Funktionalität des dekompilieren von Java-Archiven in originalen Java-Quellcode zur Verfügung, wodurch es mithilfe dieses Werkzeugs in Kombination mit dex2jar oder Enjarify möglich ist, vollständige Android-Anwendungen in Java-Quellcode zu dekompilieren. Dies ist gerade in Bezug auf statische Quellcodeanalysen eine besonders wichtige Funktionalität.

Procyon und CFR

Procyon [189] und CFR [57] sind neben JD-Core weitere Java-Decompiler, die in vielen Android-Analysewerkzeugen zum Einsatz kommen. Diese unterscheiden sich vor allem in der Kompatibilität zu neueren Java-Versionen und stellen somit eine Verbesserung zu JD-Core dar.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

8.2 Android Open-Source-Analysewerkzeuge

Dieser Abschnitt enthält eine Auswahl von Open-Source-Anwendungen und Frameworks zur Analyse der Android-Anwendungssicherheit mit dem Fokus auf die in Kapitel 7 definierten Sicherheitsbereiche. Nach einer breiten Werkzeugrecherche, haben wir uns auch in diesem Bereich dazu entschieden, vor allem die Android Security Awesome Listung [204] von Analysewerkzeugen und Frameworks als Vorselektion der für diese Arbeit interessanten Anwendungen zu verwendet und auf dieser Selektion aufzubauen.

8.2.1 APK-Parser

Der Java-basierte APK-Parser [115] wurde entwickelt, um `AndroidManifest.xml` Dateien, die gesammelt alle wichtigen Informationen über eine Android-Anwendung beinhalten, einzulesen und in lesbarer Form aufzubereiten. Der Vorteil des APK-Parsers ist, dass Android-Anwendungen nicht zuerst dekompiliert werden müssen, bevor Informationen gesammelt werden können. Diese Funktionalität ist für einen ersten und vor allem schnellen Einblick in die wichtigsten Details (e.g. Berechtigungen, Intents, Komponentendefinitionen) einer Anwendung sowie deren Konfiguration und Signatur besonders hilfreich und eignet sich daher sehr gut für Java-Anwendungen, die Informationen aus dem `AndroidManifest.xml` extrahieren und weiterverarbeiten möchten.

8.2.2 Argus Statisches Analyse-Framework (SAF) / Argus-Amandroid

Argus-Amandroid [237] ist ein Analysewerkzeug, das ebenso wie Argus-Jawa [31] in Argus-SAF [30] integriert wurde. Das Argus Framework spezialisiert sich auf Analysen von Datenflüssen, Abhängigkeiten, Kontrollflüssen und den Bereichen der Kommunikation sowie Kryptografie.

Die für unser Framework relevanten Funktionen des Argus Framework sind vor allem die *API Misuse* Checks, bei denen nach versteckten Icons und SSL- sowie Kryptografie-Problemen gesucht wird, wodurch sich das Framework für die Sicherheitsevaluierung einer Android-Anwendung in den Bereichen Kryptografie und Zertifikatvalidierung eignet. Im Bereich der Zertifikate untersucht Argus beispielsweise die Verwendung von unsicheren TrustManager Implementierungen und `SSLSockets` [12]. Die konkrete Argus Definition der entsprechenden Schwachstellen zeigen die Listings 8.1 sowie 8.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
1  val badTrustManagerNames = Set(
2    "AcceptAllTrustManager",
3    "AllTrustManager",
4    "DummyTrustManager",
5    "EasyX509TrustManager",
6    "FakeTrustManager",
7    "FakeX509TrustManager",
8    "FullX509TrustManager",
9    "NaiveTrustManager",
10   "NonValidatingTrustManager",
11   "NullTrustManager",
12   "OpenTrustManager",
13   "PermissiveX509TrustManager",
14   "SimpleTrustManager",
15   "SimpleX509TrustManager",
16   "TrivialTrustManager",
17   "TrustAllManageranager",
18   "TrustAllTrustManager",
19   "TrustAnyCertTrustManager",
20   "UnsafeX509TrustManager"
21 )
```

Listing 8.1: Definition von unsicheren TrustManager durch Argus-SAF [30]

```
1  val VUL_PARAM = "@@org.apache.http.
↪  conn.ssl.SSLSocketFactory.
↪  ALLOW_ALL_HOSTNAME_VERIFIER"
2
3  val badSSLSocketFactoryNames = Set(
4    "AcceptAllSSLSocketFactory",
5    "AllTrustingSSLSocketFactory",
6    "AllTrustSSLSocketFactory",
7    "AllSSLSocketFactory",
8    "DummySSLSocketFactory",
9    "EasySSLSocketFactory",
10   "FakeSSLSocketFactory",
11   "InsecureSSLSocketFactory",
12   "NonValidatingSSLSocketFactory",
13   "NaiveSslSocketFactory",
14   "SimpleSSLSocketFactory",
15   "SSLSocketFactoryUntrustedCert",
16   "SSLUntrustedSocketFactory",
17   "TrustAllSSLSocketFactory",
18   "TrustEveryoneSocketFactory",
19   "NaiveTrustManagerFactory",
20   "LazySSLSocketFactory",
21   "UnsecureTrustManagerFactory"
22 )
```

Listing 8.2: Definition von unsicheren SSLSockets durch Argus-SAF [30]

Im Bereich Kryptografie fokussiert sich das Argus Framework auf die Verwendung von ECB-Mode und CBC-Mode mit nicht zufällig gewählten IV und geht somit auf die Erkenntnisse von Egele et al. [85] ein.

Durch den Einsatz von Argus-SAF wurden etwa von Wei et al. im Bereich Datenfluss-Analyse [236], [237], [239], Android Malware [238] sowie von Lin et al. im Bereich der forensischen Android-Anwendungsanalyse [162] veröffentlicht. Die Analyseergebnisse des Frameworks sowie weitere auf Argus-SAF basierenden Werkzeuge und wissenschaftlichen Veröffentlichungen sind allerdings mit Vorsicht zu genießen, wie Bonett et al. [39] feststellen. Auf diese Arbeit wurde in Abschnitt 2 genauer eingegangen.

8.2.3 Androguard

Androguard [112] ist ein Python-Werkzeug das sich auf Android Malware Analyse und Reverse Engineering spezialisiert. Die Hauptfunktionalität liegt auf Manifest- und Quellcodeanalysen, bei denen Berechtigungen, Aktivitäten, Zertifikate, Instruktionen und weitere Anwendungsinformationen extrahiert werden. Des Weiteren bietet das Python-Werkzeug Funktionen zur Erkennung von Obfuscation sowie zur Malware Risiko Bestimmung und Erstellung von Kontrollflussgraphen. Die Funktionalität selbst stellt sich aus der Verbindung von in sich abgeschlossene Teilwerkzeuge zusammen, wodurch sich Androguard auf den ersten Blick besonders für die Integration in andere Analysetools eignet.

Bei den ersten Funktionstests, die im Rahmen der Recherche durchgeführt wurden, kamen allerdings einige Probleme sowie Unstimmigkeiten des Androguard Analysewerkzeugs ans Licht. Den Git Commits [112] zufolge, passierten in der Vergangenheit größere Umstrukturierungen, wodurch auch die zu den Releases verfügbaren Dokumentationen stellenweise stark veraltet scheinen und

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

auf Python-Skripts verweisen, die nicht mehr in der beschriebenen Form existieren. Dies führte zu erheblichen Aufwand bei unseren Testversuchen. Selbst bei Release-Versionen waren noch gröbere Probleme zu verzeichnen. Ein Beispiel dafür ist die für unsere Analyse interessante Funktionalität der Kontrollflussgraph Generierung, die der Dokumentation zufolge mit dem in Androguard enthaltenen `apkviewer.py` Skript durchgeführt wird. Dieses Skript bricht unabhängig von der zu untersuchenden Anwendung beispielsweise mit folgender Fehlermeldung ab.

> 'DalvikVMFormat' object has no attribute 'dotbuff'.

Zu diesem Fehler gibt es auch bereits ein Ticket auf Github [113] mit dem Ergebnis, dass 2013 offenbar noch benötigte Codezeilen entfernt wurden. Wir schließen daraus, dass die Kontrollflussgraph Erstellung auch seit diesem Zeitpunkt nicht mehr funktioniert.

Gerade bei einem Projekt wie diesem, mit weitreichend interessanten Funktionsumfang, wäre eine durchgängige, ausführliche Dokumentation und Funktionsauflistung sowie Versionierung wünschenswert. Da aber andere Werkzeuge, wie etwa AndroBugs [111] zumindest Teilweise auf Androguard aufbauen und das Projekt auch noch aktiv scheint, dürfte dieses Tool zumindest nach größerer Funktions- und Quellcodeanalyse sowie möglichen notwendigen Anpassungen wiederverwendbar und nutzbar sein. Im Rahmen dieser Arbeit wurde das Werkzeug allerdings nicht weiter verwendet.

8.2.4 AndroBugs

AndroBugs [111] ist ein Framework zur Analyse der Android-Anwendungen in Bezug auf Sicherheitslücken und potenzieller Probleme, die während der Entwicklung einer Anwendung durch falsche Design-Entscheidungen sowie fehlerhafte Implementierungen entstehen können. AndroBugs deckt nicht nur eine sehr breite Palette von Sicherheitslücken ab, die große Bereiche der OWASP Top 10 beinhaltet, sondern berücksichtigt bei der Analyse auch eingebundene und innerhalb einer Anwendung verwendete Bibliotheken. Durch den breiten Analysebereich, der alle in Kapitel 7 definierten Sicherheitskategorien und viele darin enthaltenen Unterbereiche abdeckt, ist AndroBugs besonders interessant für die Verwendung in unserem Framework.

Zur Analysefunktionalität im Bereich mangelnder Angriffsschutz gehören die folgenden Punkte.

a) Verwendung als unsicher geltender Android-APIs

b) Fehlkonfigurationen, Kompatibilitätsprobleme sowie Fehler in Bezug auf bestimmte Android-Versionen

c) Verwendung von nativen Codesegmenten

d) Repacking in Bezug auf fehlende Signatur oder Verwendung von einem Debug Zertifikat

e) Anforderung von kritischen Systemberechtigungen

f) Fehlkonfigurationen von eigens erstellten Berechtigungen

Im Bereich Kryptografie fokussiert sich AndroBugs auf die Verwendung von verschlüsselten Datenbanken und dem Auftreten von Base64 Verschlüsselung. Des Weiteren wird der Bereich der Zertifikatvalidierung in Bezug auf bekannte unsichere Implementierungen untersucht. In der definierten Kategorie der anwendungsübergreifenden Zugriffskontrolle wird durch AndroBugs die Definition von öffentlichen Komponenten und die Verwendung von impliziten Intents untersucht. Auch der Bereich der Datenlecks wird durch folgende Punkte abgedeckt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

a) Verwendung von IMEI[1] und AndroidID

b) Funktionalität zum Rooten eines Geräts

c) Prüfung von Root Rechten und Ausführung von Kommandos als Root

d) Unsicheres Löschen von Daten

e) Unsichere HTTP-Kommunikation

f) Implementierung zu öffentlich les- und schreibbaren Daten

g) Verwendung von unverschlüsselten Datenbanken

h) Erlaubnis von ADB-Backups

i) Konfiguration verwendeter KeyStores

Ebenso werden im Bereich der Eingabevalidierung die Einstellungen bezüglich der JavaScript WebView Aktivierung, WebView SSL-Zertifikatvalidierung und die Funktionalität des Datenzugriffs über die WebView überprüft.

Das Framework baut auf der Funktionalität des Analysewerkzeugs Androguard auf und erweitert dieses um wertvolle Sicherheitseinstufungen sowie Kategorisierungen. Ein ausführlicher Report wird in Form einer Textdatei oder auf Wunsch in einer NoSQL-Datenbank zur Verfügung gestellt. AndroBugs zeichnet eine kurze Analysedauer aus, die der offiziellen Github Framework Beschreibung zufolge durchschnittlich bei unter zwei Minuten pro Scan einer Anwendung liegt [111].

Darüber hinaus ist das Framework portabel und aufgrund einer plattformübergreifenden Anleitung zur Installation und Verwendung tatsächlich sehr einfach einsetzbar, und eignet sich daher besonders gut für Experten und Expertinnen, Entwickler sowie Entwicklerinnen, Hacker und Hackerinnen oder Interessierte, um in kurzer Zeit einen möglichst breiten Bereich an Sicherheitslücken oder Entwicklungsproblemen einer Anwendung aufzuzeigen. AndroBugs basiert größtenteils auf Volltext-Analysen beziehungsweise *Pattern Matching*-Analysen, wodurch das Werkzeug auch relativ einfach um weitere Sicherheitsanalyse Parameter erweitert werden könnte. Allerdings passiert die gesamte Definition der Sicherheitslücken, die AndroBugs analysiert, ebenso wie die Implementierung der Analyse selbst, in einem einzigen Python-Skript, das mehrere tausend Zeilen lang ist. Demzufolge ist die bestehende Funktionsweise entsprechend unübersichtlich und Veränderungen an der existierenden Funktionalität ohne tief greifendes Verständnis nicht empfehlenswert.

An dieser Stelle ist die Arbeit von Khadiranaikar et al. [151] erwähnenswert, in der unter anderem mit AndroBugs versucht wurde, Android Intent Sicherheitslücken zu untersuchen, sowie die Arbeit von Ibrar et al. [138], in welcher der Fokus auf die Analyse von Anwendungen im Finanzbereich gelegt wurde.

8.2.5 Androwarn

Ein weiteres Werkzeug, das in dieser Arbeit genauer betrachtet wurde, ist Androwarn [114]. Die Python-basierte Anwendung versucht mit einer statischen Codeanalyse möglichst viel Information über eine Android-Anwendung zu sammeln, um dem Anwender oder der Anwenderin in

[1] AndroBugs sucht nach der Verwendung der Funktion `getDeviceId()` des `TelephonyManager`, die seit API Level 26 als *deprecated* gilt. Stattdessen wird durch die Android Dokumentation des `TelephonyManager` die Funktion `getImei()` empfohlen [23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

weiterer Folge zu ermöglichen, anhand eines detaillierten Berichts, unerwünschte Anwendungen und Funktionalitäten zu identifizieren. Androwarn ist, ebenso wie beispielsweise AndroBugs, ein Analysewerkzeug, dass auf Androguard aufbaut und dieses erweitert. Bei der statischen Quellcodeanalyse fokussiert sich Androwarn neben der `AndroidManifest.xml` Auswertung vor allem auf Struktur- und Datenfluss-Analysen, um auf diesem Weg mögliches unerwünschtes Verhalten aufzudecken. Ein großer Teil der Untersuchungen basiert, der übersichtlichen Dokumentation zufolge, auf Pattern Matching sowie Volltextsuchen und deckt die nachfolgend gelisteten Sicherheitsrisiken auf, wodurch sich das Werkzeug besonders für die Evaluierung der Datenlecks-Kategorie eignet.

> **Telephony identifiers exfiltration:** IMEI, IMSI, MCC, MNC, LAC, CID, operator's name ...
>
> **Device settings exfiltration:** software version, usage statistics, system settings, logs ...
>
> **Geolocation information leakage:** GPS/WiFi geolocation ...
>
> **Connection interfaces information exfiltration:** WiFi credentials, Bluetooth MAC adress ...
>
> **Telephony services abuse:** premium SMS sending, phone call composition ...
>
> **Audio/video flow interception:** call recording, video capture ...
>
> **Remote connection establishment:** socket open call, Bluetooth pairing, APN settings edit ...
>
> **PIM data leakage:** contacts, calendar, SMS, mails ...
>
> **External memory operations**: file access on SD card ...
>
> **PIM data modification:** add/delete contacts, calendar events ...
>
> **Arbitrary code execution:** native code using JNI, UNIX command, privilege escalation ...
>
> **Denial of Service:** event notification deactivation, file deletion, process killing, virtual keyboard disable, terminal shutdown/reboot ...
>
> [114]

Das Ergebnis der Analyse speichert Androwarn auf Wunsch in einer Textdatei oder generiert einen Report in HTML-Format. Durch die einfache Architektur, Logik und leichte Erweiterbarkeit eignet sich dieses Werkzeug besonders zur Wiederverwendung innerhalb anderer Frameworks oder zur Erweiterung und Einbindung neuer Analyse- beziehungsweise Suchkriterien. Die Entwicklung des Werkzeugs selbst ist allerdings seit 2013 eingefroren.

8.2.6 ApkAnalyser

Der ApkAnalyser [116] ist ein Java-basiertes Werkzeug mit grafischer Oberfläche, dient der manuellen Analyse von Android-Anwendungen und baut auf Smali sowie ApkTool auf. Kernfunktionalität ist das Erforschen und Verändern von Anwendungsquellcode sowie das Darstellen von Architektur und Abhängigkeiten auf Basis von Klassendiagrammen oder Methoden-basierten Kontrollflussgraphen.

Nach erfolgreicher Installation des Analysewerkzeugs lässt sich dieses über ein JAR starten. Für eine Analyse von Android-Anwendungen ist neben der APK-Datei selbst, noch die Definition des zur Anwendung passenden Android Software Development Kit (SDK) und weiteren, in der

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Android-Anwendung verwendeten, Bibliotheken nötig, was bei der Analyse von unbekannten Anwendungen nahezu unmöglich scheint. Dadurch eignet sich dieses Analysewerkzeug eher für Entwickler und Entwicklerinnen zur Analyse eigens entwickelter Anwendungen. Dies geht auch aus der Beschreibung des ApkAnalyser Projekts hervor.

> ApkAnalyser is a static, virtual analysis tool for examining and validating the development work of your Android™ app [116].

Im Bereich dieser Arbeit wurde die Funktionalität dieses Werkzeugs nicht weiter analysiert, da die Hauptfunktionalitäten an der benötigten Funktionalität zur automatisierten Sicherheitsanalyse vorbeigehen.

8.2.7 Permission-Lint (P-Lint)

P-Lint [182] analysiert Berechtigungen von Android-Anwendungen auf Basis von Android Best Practice Richtlinien und versucht Abweichungen von gängigen Programmierstandards aufzuzeigen. Dabei konzentriert sich P-Lint auf 16 Richtlinien, von denen die folgenden vier für die Evaluierung des sicherheitskritischen Bereichs der Berechtigungen besonders relevant sind.

> 5. APK does NOT initialize checkSelfPermission method even if permission requests are made (Bad) [182]

Beim Zugriff auf Funktionalitäten, die bestimmte Berechtigungen benötigen, ist es notwendig durch *CheckSelfPermission* zu überprüfen, ob die Anwendung zum aktuellen Zeitpunkt noch die nötigen Berechtigungen besitzt, wie Dennis et al. [75] sowie die Android Dokumentation zur Berechtigungsanforderung beschreiben, da die Berechtigungen einer Anwendung ab API Level 23 beziehungsweise Android 6.0 (siehe Abschnitt 5.2.1) durch den Nutzer oder die Nutzerin zur Laufzeit verändert werden können. Die nachfolgenden Richtlinien 9 und 14 weisen auch auf mögliche Probleme im Bereich der dynamischen Berechtigungserteilung hin, wodurch eine genauere Untersuchung der Anwendung empfehlenswert ist.

> 7. Checks to see if multiple permission requests are made in close proximity in the code, possibly overwhelming user with requests (Bad)
>
> 9. Permission declaration has a min SDK version set below 23, and no max SDK version. (Check that it works for BOTH permission models)
>
> 14. APK's min SDK version is below 23, and max SDK version is at or above 23 (Should check how it functions on BOTH models)
>
> [182]

Die Funktionsweise von P-Lint baut auf den durch ApkTool sowie dex2jar dekompilierten Quellen auf. Besonders hilfreich ist die Best Practice Richtlinien Analyse für Android-Anwendungsentwickler und Entwicklerinnen, um auf diesem Weg die Qualität einer Anwendung in Bezug auf die Berechtigungskonfiguration zu prüfen. Auch für Frameworks die Codequalität analysieren, ist das Tool empfehlenswert, da der eigentliche Analyseprozess wenig Zeit in Anspruch nimmt und meist innerhalb Sekunden ein Ergebnis in HTML-Format vorliegt. Gerade in Frameworks, bei denen Android-Anwendungen ohnehin in dekompilierter Form vorliegen, stellt P-Lint eine einfache Analyseerweiterung dar.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

8.2.8 Quick Android Review Kit (QARK)

QARK [124] ist ein weiteres auf Github veröffentlichtes Open-Source-Werkzeug zur Analyse von Android-spezifischen Sicherheitsproblemen, mit dem durch statische Quellcodeanalysen Probleme in unterschiedlichen Bereichen einer Anwendung aufdeckt werden können. Zu den Kernfunktionalitäten gehört die Dekompilierung mithilfe von JD-Core, Procyon und CFR sowie Disassemblierung durch ApkTool gefolgt von der anschließenden Analyse. Bei dieser stehen vor allem die definierten Evaluierungskategorien zur richtige Verwendung von Komponenten, Aktivitäten und Intents sowie die Zertifikatvalidierung, das Erstellung von Daten, die WebView Konfiguration und Kryptografie im Vordergrund. Dadurch werden von diesem Werkzeug nicht nur viele der OWASP Top 10 Kategorien abgedeckt, sondern auch viele der für unsere Sicherheitseinstufung relevanten Bereiche. Das Werkzeug eignet sich daher besonders sowohl für einzelne Anwendungsanalysen, als auch für die Wiederverwendung in Analyse-Frameworks.

QARK basiert auf Python und stellt sowohl eine automatisierte als auch eine interaktive Analyse zur Verfügung. Was QARK eindeutig von anderen Werkzeugen abhebt, ist die auf Wunsch automatisierte Generierung von *Exploits* in Form von modifizierten Anwendungen oder ADB-Kommandos, um mögliche gefundene Angriffsszenarien direkt anhand funktionierender *Proof of Concepts* aufzuzeigen. Darüber hinaus liefert QARK einen vollständigen Bericht über mögliche gefundene Probleme in HTML- und CSV-Format und eignet sich sowohl als alleinstehende Analyseanwendung, als auch als Teil eines noch größeren Frameworks.

Zusammenfassend ist QARK ein übersichtliches, funktionales Werkzeug für eine rasche und vor allem breite Analyse von Android-Anwendungen.

8.2.9 Secure, Unified, Powerful and Extensible Rust Android Analyzer (SUPER)

SUPER [125] ist ein Framework zur Analyse des Sicherheitskonzepts von Android-Anwendungen, das auf den durch dex2jar und JD-Core dekompilierten Quellcode aufbaut, und sich von anderen Frameworks speziell durch die Wahl der Architektur und Technologien abhebt. Anders als die meisten anderen verfügbaren Analysewerkzeuge basiert SUPER nicht auf Java oder Python, sondern auf Rust [203].

Rust ist eine Programmiersprache, die von Mozilla Research entwickelt wurde und vor allem auf Geschwindigkeit und Parallelisierung ausgelegt ist, wie Matsakis et al. [167] beschreiben. Dabei werden Ansätze aus der objektorientierten und funktionalen Programmierung verwendet, wodurch eine mit C++ vergleichbare Effizienz erreicht wird. Rust bietet darüber hinaus einen besonders einfachen Umgang mit *Regular Expressions*, Pattern Matching und Daten im Allgemeinen, was bei statischen Analysen von Android-Anwendungen eine wichtige Funktionalität darstellt.

Das Kommandline Analyse-Framework SUPER basiert auf Regular Expression Suchen und bietet die vollständige Konfigurierbarkeit dieser in Form einer zentralen Konfigurationsdatei. Dadurch ist dieses Werkzeug, anders als die meisten von uns untersuchten Werkzeugen, beliebig über eine Datei im JSON-Format, die alle Sicherheitsregeln der Analyse enthält, anpass- und erweiterbar. Die Implementierung von SUPER liefert eine Standardkonfiguration, die 37 Regeln enthält [125]. Einige davon beschäftigen sich vorwiegend mit den für unsere Sicherheitsevaluierung relevanten Bereichen der Eingabevalidierung oder genauer mit XSS und SQL-Injection Attacken, schwachen Algorithmen wie MD4, MD5 sowie SHA1 und Base64 im Bereich Kryptografie, unsicheren Konfigurationen und Implementierungen zur Zertifikatvalidierung und Datenlecks. Dadurch stellt SUPER ein interessantes Werkzeug zur Anbindung an unser Framework dar.

Wie die Definition einer Analyseregel durch SUPER aussieht, zeigt beispielhaft Listing 8.3 anhand der Regel zur Untersuchung von SQL-Injection Angriffsmöglichkeiten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
22 {
23 "regex": "android\\.database\\.sqlite",
24 "forward_check": "(?:rawQuery|execSQL)\\(.*\"\\s*\\+\\s*.*\\)",
25 "criticality": "critical",
26 "label": "SQL injection",
27 "description": "This application is vulnerable to SQL injection. Any
   ↪ data stored in database can be exposed as any attacker is able
   ↪ to retrive, modify and delete the stored information.",
28 "include_file_regex": ".java"
29 }
```

Listing 8.3: Definition der SUPER Regel zur Untersuchung von SQL-Injection Angriffsmöglichkeiten [125]

Das Ergebnis der Analyse wird sowohl in HTML- und JSON-Format ausgegeben und eignet sich daher auch zur Wiederverwendung in anderen Anwendungen.

8.2.10 Mobile Security Framework – MobSF

MobSF [170] ist ein wichtiges Python-basierte statisches und dynamisches *Pen-Test* Framework, das sich im Gegensatz zu dieser Arbeit auch auf die Analyse von Windows Mobile und iOS-Anwendungen fokussiert. Das Framework spezialisiert sich, den Angaben des Projekts zufolge, vor allem auf die Integration in *Continuous Integration (CI)* und *Continuous Delivery (CD)* Pipelines. Leider beinhaltet die Dokumentation keine genaue Auflistung der untersuchten Sicherheitsbereiche. Ersten Funktionstests zufolge werden aber große Bereiche der OWASP Top 10 abgedeckt. Weiters scheint das Framework ohne große Umstrukturierungen allerdings nicht auf Anpassungen und Einbindungen durch andere externe Werkzeuge ausgelegt zu sein. Auch die Möglichkeit der konfigurierbaren Sicherheitseinstufung besteht nicht, die bezüglich der Integration in eine CI/CD Pipeline unserer Meinung nach eine wichtige Funktion darstellt. Für diese Arbeit stellen vor allem die innerhalb des Projekts eigens entwickelten Analyseverfahren eine besondere Relevanz dar, die sich möglicherweise selektiert und gezielt in unser Framework einbinden lassen könnten.

8.2.11 M-Perm

Wie in dieser Arbeit bereits festgestellt wurde, spielen Berechtigungen in Android-Systemen eine bedeutende Rolle (e.g. [96], [98], [97], [241]). M-Perm [121] ist ein mächtiges Werkzeug, dass sich mit der Über- und Unterprivilegierung von Anwendungen befasst. Also der Untersuchung von Berechtigungen, die angefordert aber nicht verwendet beziehungsweise umgekehrt nicht angefordert aber benötigt werden. Das Werkzeug unterteilt sich in zwei Teilbereiche, die durch die Dekompilierung und die nachstehende Analyse repräsentiert werden. Dazu bedient sich M-Perm der Funktionalität der zuvor beschrieben Werkzeuge ApkTool, dex2jar und JD-Core, um Android-Anwendungen zu dekompilieren. Wenn die zu analysierenden Android-Anwendungen bereits in dekompilierter Form vorliegen, kann dieser Schritt übersprungen werden, wodurch die Analysezeit optimiert wird und ein rasches Ergebnis vorliegt.

M-Perm wurde im Zuge der wissenschaftlichen Arbeit von Chester et al. [59] entwickelt, in der die Autoren 50 Anwendungen auf über- und unterprivilegierte Berechtigungen untersuchten. Dabei fanden sie, ihren eigenen Aussagen zufolge, eine Überprivilegierung von rund 2,2% der untersuchten Anwendungen.

Bei den Funktionstests des Analysewerkzeugs, bei denen die Anwendungen Facebook, WhatsApp sowie Instagram analysiert wurden, musste allerdings festgestellt werden, dass alle als überprivilegiert gekennzeichneten Berechtigungen, also Berechtigungen die laut M-Perm Definition

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

angefordert aber im Anwendungscode nicht verwendet werden, für die normale Verwendung der Anwendungsfunktionen unter Garantie benötigt werden. Dadurch stellt das Analyseergebnis von M-Perm für das Framework keinen Mehrwert dar. Listing 8.4 zeigt den entsprechenden Report Ausschnitt der Untersuchung von WhatsApp durch M-Perm.

```
1
2 ———————— Unrequested Dangerous (Under) ————————
3
4 ————————— Requested Dangerous (Over) —————————
5 ['READ_EXTERNAL_STORAGE', 'WRITE_EXTERNAL_STORAGE']
6 ['RECORD_AUDIO']
7 ['SEND_SMS', 'RECEIVE_SMS', 'READ_SMS', 'RECEIVE_WAP_PUSH', '
  ↪ RECEIVE_MMS']
8 ['ACCESS_FINE_LOCATION', 'ACCESS_COARSE_LOCATION']
9 ['CAMERA']
10 ['READ_CONTACTS', 'WRITE_CONTACTS', 'GET_ACCOUNTS']
11 ['READ_PHONE_STATE', 'CALL_PHONE', 'READ_CALL_LOG', 'WRITE_CALL_LOG
  ↪ ', 'ADD_VOICEMAIL', 'USE_SIP', 'PROCESS_OUTGOING_CALLS']
```

Listing 8.4: Ergebnis der M-Perm Analyse zur Überprivilegierung von WhatsApp

8.2.12 FlowDroid

FlowDroid [103] wurde innerhalb einer Arbeit von Arzt et al. [32] entwickelt und ist ein Werkzeug, das sich auf die statische Datenfluss-Analyse mithilfe von *Taint Tracking* (e.g. [175], [153], [247]) fokussiert. Um den Datenfluss einer Anwendung zu analysieren, benötigt FlowDroid neben der APK-Datei selbst, die Konfiguration der Ausgangspunkte und Endpunkte *Sources and Sinks* sowie das zur Anwendung passende Android-SDK. Das Ziel von FlowDroid ist es, eine Basis für tiefer greifende Forschungen darzustellen und es anderen Werkzeugen zu ermöglichen, auf dieser Basis aufzubauen, die zum Beispiel in Form von einer Bibliothek eingebunden werden kann. FlowDroid basiert auf Soot [217], einem Compiler Optimierungsframework, das Java-Bytecode vor der Analyse in ein eigenes, leichter zu analysierendes, Format umwandelt. FlowDroid wird in vielen Arbeiten, aus denen unter anderem auch neue Analysewerkzeuge hervorgehen, als Vergleichswerkzeug herangezogen (e.g. [158], [129], [153]). Eine besondere Arbeit ist dabei von Bonett et al. [39], die unter anderem die Genauigkeit von FlowDroid analysiert und feststellt, dass durch FlowDroid rund die Hälfte der untersuchten Sicherheitslücken unentdeckt bleiben. Aus diesem Grund wurde das Werkzeug innerhalb dieser Arbeit nicht weiter untersucht.

Am Ende dieses Kapitels wird anhand der Tabelle 8.1 ein zusammenfassender Überblick über die Funktionen und Einsatzzwecke der untersuchten Open-Source-Werkzeuge zur Sicherheitsanalyse von Android-Anwendungen gegeben.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Werkzeug	**Einsatzzweck**
APK-Parser	Aufbereitung und Darstellung des AndroidManifest.xml.
Argus-SAF	Analyse von Datenflüssen, Abhängigkeiten, Kontrollflüssen. Analyse von API-Missbrauch im Bereich Kommunikation und Kryptografie.
Androguard	Manifest- und Quellcodeanalysen. Extraktion von Berechtigungen, Aktivitäten, Zertifikate, Instruktionen und weitere Anwendungsinformationen. Erkennung von Obfuscation sowie Malware Risiko Bestimmung.
AndroBugs	Analyse von Sicherheitslücken und potenzieller Probleme, die während der Entwicklung durch falsche Design-Entscheidungen sowie fehlerhafte Implementierungen entstehen können. Deckt einen Großteil der OWASP Top 10 ab.
Androwarn	AndroidManifest.xml Auswertung und Struktur- beziehungsweise Datenfluss-Analysen. Untersucht beispielsweise den Zugriff auf Hardware IDs und externen Speicher, Missbrauch von Telefonservices und Standortinformation sowie Ausführung von Native Code.
ApkAnalyser	Manuellen Analyse von Android-Anwendungen und ermöglicht Einblicke in die Implementierung.
P-Lint	Analyse von Abweichungen der Android Best Practice Richtlinien und gängigen Programmierstandards im Bereich von Berechtigungen.
QARK	Analyse der richtigen Verwendung von Komponenten, Aktivitäten, Intents, Kryptografie sowie die Zertifikatvalidierung, der Erstellung von Daten und der WebView Konfiguration. Ermöglicht die Generierung von Exploits in Form von modifizierten Anwendungen oder ADB-Kommandos.
SUPER	Analyse basierend auf Regular Expression Suchen, die vollständige konfigurierbar sind. Die mitgelieferte Standardkonfiguration beschäftigt sich vor allem mit den Bereichen der Eingabevalidierung, schwachen Kryptografie Algorithmen sowie unsicheren Konfigurationen und Implementierungen zur Zertifikatvalidierung.
MobSF	Statisches und dynamisches Pen-Test Framework von Android, Windows Mobile und iOS-Anwendungen. Spezialisiert sich auf die Integration in CI/CD Pipelines.
M-Perm	Analyse von Über- und Unterprivilegierung.
FlowDroid	Statische Datenfluss-Analyse mithilfe von Taint Tracking.

Tabelle 8.1: Auflistung der Werkzeuge mit einem Überblick zu den jeweiligen Einsatzzwecken

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

9 Beschreibung des Frameworks zur automatisierten Analyse der Android-Anwendungssicherheit

In diesem Kapitel wird die Implementierung des Analyse-Frameworks sowie wichtige Architektur- und Design-Entscheidungen beschrieben. Darüber hinaus beschreibt dieses Kapitel den Funktionsumfang, die Erweiterung und die Verwendung des Frameworks. Um im Anschluss Architektur- und Design-Entscheidungen zu argumentieren, wird als Erstes mit der Beschreibung des Funktionsumfangs begonnen.

9.1 Anforderungen und Konzeption

Ziel dieses Frameworks ist die Kombination bestehender Android-Analysewerkzeuge innerhalb einer einheitlichen Anwendung, um die Verwendung dieser einzelnen Werkzeuge zu erleichtern und durch die Kombination der einzelnen Analyseergebnisse eine Sicherheitseinstufung sowie Kategorisierung von Android-Anwendungen durchführen zu können. Dabei soll das Framework eine möglichst einfache, erweiterbare und skalierende SaaS/TaaS [202] Analyselösung darstellen, um auf diesem Weg einen Mehrwert für eine breite Zielgruppe darzustellen. Der Aufbau und die Architektur des Frameworks ist daher essenziell, um diese Anforderungen zu erfüllen.

Um eine möglichst große Anzahl an Android-Anwendungen zu untersuchen, benötigt das Framework neben einer Upload Möglichkeit für ausgewählte Android-Anwendungen auch eine Anbindung an den Google Play Store [127], damit beispielsweise basierend auf einer gewählten Anwendungskategorie automatisiert entsprechende APKs für die spätere Analyse downgeloadet werden können.

Das Framework lässt sich also in drei funktionale Anforderungsbereiche aufteilen.

1. Eine Upload und Google Play Store Download Funktionalität, um Anwendungen zur Analyse hoch beziehungsweise herunterladen zu können.
2. Die Kombination ausgewählter Analysewerkzeuge und automatisierte Verarbeitung der Analyseergebnisse und Reports.
3. Die dynamische Kategorisierung basierend auf den Analyse-Reports.

Weiters bestehen folgende nicht-funktionale Anforderungen.

1. Möglichst einfache Erweiterbarkeit sowohl durch neue Analysewerkzeuge, als auch neue oder veränderte Sicherheitseinstufungen.
2. Skalierende Anwendungsanalysen sowie problemlose Portierbarkeit des Frameworks.
3. Hohe Verfügbarkeit durch Bereitstellung in Form eines Webservices.
4. Funktionales Benutzerinterface.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

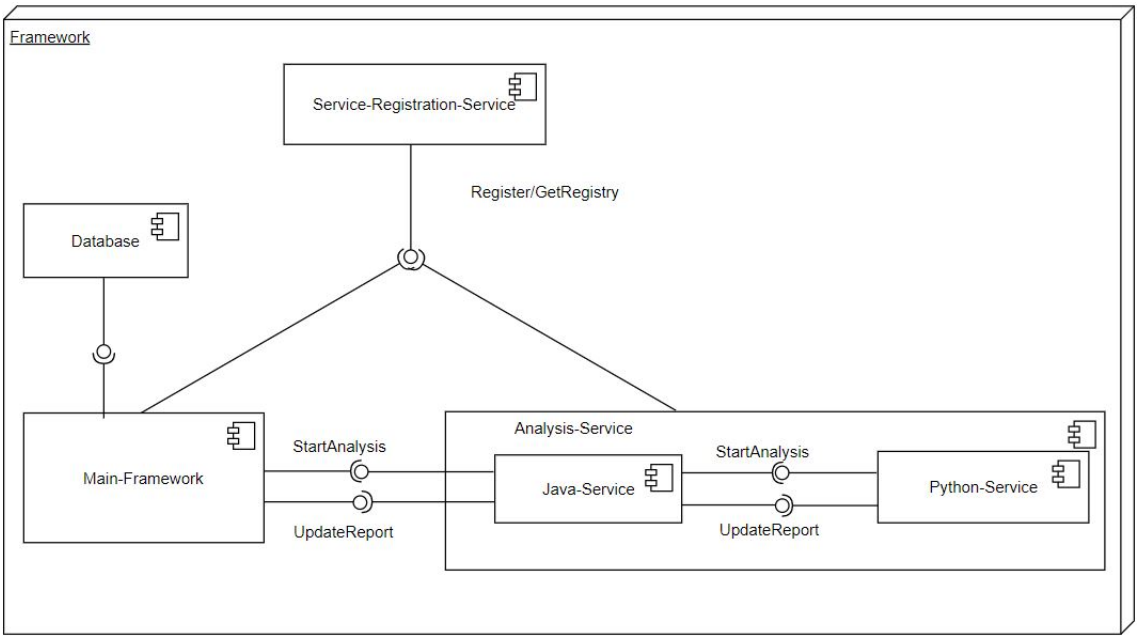


Abbildung 9.1: Framework Architektur

5. Zentrales Speichern von Analyseergebnissen zur Wiederverwendung dieser als Eingabe zukünftiger Auswertungen oder Analysewerkzeuge.

9.2 Architektur- und Design-Entscheidungen

Dieser Abschnitt beschreibt die wesentlichen Architektur- sowie Design-Entscheidungen und begründet die Wahl der verwendeten Technologien beziehungsweise Frameworks.

9.2.1 Architektur

Um ein großes komplexes monolithisches System zu verhindern, das nicht nur schwer zu warten, sondern auch kaum Veränder- und Wiederverwendbar ist, fiel bei der Entwicklung des Frameworks die Entscheidung auf ein *Microservice* (e.g. [168], [83]) basierendes System. Dabei wurde die Funktionalität des Frameworks in abgeschlossene Teilgebiete aufgeteilt, die jeweils durch ein eigenständiges, abgeschlossenes Microservice repräsentiert werden. Die Kommunikation zwischen diesen Services passiert über REST-Schnittstellen [244], das auch für weitere Entwicklungen ein schmales, smartes API darstellt.

Um zum einen eine zentrale Datenverwaltung und zum anderen eine skalierende Analyse zu ermöglichen, wurde das System als Erstes in ein zentrales Datenverwaltung und Benutzerinterface Microservice, im weiteren `Main-Service` genannt, und weiters in ein Microservice zur Analysefunktionalität `(Analysis-Service)` geteilt. Dies bietet den Vorteil, dass einzelne Analyse-Microservices beziehungsweise Analyseknoten dynamisch an das zentrale Main-Service gekoppelt werden können und dadurch eine skalierende Analyse erreicht wird. Die Kommunikation und die Lastverteilung zwischen den einzelnen Analyseknoten übernimmt ein weiteres eigenständiges zentrales Microservice mit dem Namen `Service-Registration`.

Die Abbildung 9.1 gibt einen Überblick über die Architektur, Komponenten und die Kommunikation zwischen den Komponenten des Frameworks, die anschließend der Reihe nach besprochen werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Service-Registration

Die Service-Registration Komponente verwaltet die Instanzen der anderen Services und übernimmt die Lokation sowie die Lastverteilung, wodurch die Kommunikation zwischen den Instanzen ermöglicht wird. Diese Funktionalität wird vereinfacht durch das Interface `Register/FindRegistry` abgebildet. Abgesehen von dieser Funktionalität besitzt die Service-Registration Komponente keine weitere Funktionalität.

Main-Service

Die nächste Hauptkomponente des Frameworks stellt die Main-Service Komponente dar, welche die Funktionalität der Benutzeroberfläche, Benutzerkommunikation sowie die Datenverwaltung von Android-Anwendungen und Reports abbildet. Das Main-Service kommuniziert dazu mit der Service-Registration Komponente, um Analyseknoten zu identifizieren und auf diesen anschließend Analyseaufgaben zu starten.

Analysis-Service

Die Analyse-Microservice Komponente besteht jeweils aus zwei unterschiedlichen Teilkomponenten. Einer Python- und einer auf Java-basierenden Teilkomponente, die unterschiedliche Analyseaufgaben und Werkzeuge besitzen.

Hintergrund der Aufteilung in zwei getrennte, aber eng zusammenarbeitenden Komponenten ist die vereinfachte und erhöhte Flexibilität bei der Anbindung von bestehenden Analysewerkzeugen, die meist in Python oder Java existieren. Aus diesem Grund fiel bei der Entwicklung des Frameworks die Entscheidung, die Analysefunktionalität in zwei Komponenten zu unterteilen, um so diese Werkzeuge möglichst effizient einbinden und verwenden zu können. Beide Analysekomponenten greifen dabei auf das gleich darunterliegende Dateisystem zu, wodurch nur einmal eine Disassemblierung beziehungsweise eine Dekompilierung der zu untersuchenden Android-Anwendungsquellen nötig ist. Die Kommunikation zwischen der Python und Java Analyseeinheit passiert ebenso über einheitliche interne REST-Schnittstellen. Die Python Teilkomponente kommuniziert ausschließlich über die Java Teilkomponente, die sämtliche Kommunikation Richtung Main-Service verwaltet. Welche Analyse Funktionalitäten die beiden Teilkomponenten genau beinhalten, wird in Abschnitt 9.3 noch genauer erläutert.

9.2.2 Technologien

Welche Technologien und Frameworks bei der Implementierung der einzelnen Komponenten zum Einsatz kamen, wird in diesem Abschnitt beschrieben. Bei der Wahl wurden vor allem weitverbreitete und wartbare State-of-the-Art-Technologien bevorzugt.

Maven 3

Bei allen Komponenten des Frameworks kommt Maven [28] zur Versionierung- und Abhängigkeitsverwaltung zum Einsatz. Grund für die Wahl von Maven ist die mehrjährige positive Entwicklungserfahrung im Umgang mit diesem Werkzeug.

Java 8 mit Spring Framework

Eine Java 8 [145] Spring Boot [218] Anwendung bildet die Basis der drei Framework Komponenten. Bei der Wahl der Java-Version wurde die zum Projektstart aktuelle und von der Community empfohlene Version 8 verwendet. Das Spring Framework stellt mittlerweile State-of-the-Art Standard für Java-basierte Anwendungen und speziell für Webanwendungen und Microservices

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

dar, da es die Entwicklung von effizienten komplexen Systemen enorm erleichtert. Nicht zuletzt deswegen, weil Spring einen breiten Funktionsbereich zur Entwicklung moderner Anwendungen abdeckt und dadurch nicht nur in einem, sondern in vielen verschiedenen Bereichen eine Erleichterung darstellt.

Python 3 mit Flask

Neben der Java-basierenden Hauptkomponente des Analyse-Service existiert die eigenständige Python-basierte [191] Teilkomponente, bei der das Framework Flask [192] verwendet wurde, um rasch und einfach REST-Schnittstellen und die Kommunikation mit der Java-Analysekomponente umzusetzen.

Spring Data Java Persistence API (JPA)

Um alle nötigen Daten zu persistieren, wurde innerhalb des zentralen Main-Frameworks JPA [220] verwendet. JPA kapselt Datenbankzugriffe und ermöglicht maximale Flexibilität bei der Wahl der konkreten Datenbank Implementierung. Mit JPA ist es beispielsweise über wenige Konfigurationszeilen möglich zwischen unterschiedlichen Datenbanken wie MariaDB [228], H2 [133] oder Amazon RDS [7] zu wechseln, wodurch nicht nur maximale Flexibilität während der Entwicklung innerhalb des Entwicklungsteams, sondern auch der Deployment Prozess mit beispielsweise Cloud-basierten Datenbanklösungen erleichtert wird. Darüber hinaus stellt JPA in Kombination mit Java Spring Boot Anwendungen eine große Erleichterung bei der Entwicklung und Fehlervermeidung dar.

Spring Cloud Netflix Eureka

Gerade bei verteilten Microservices mit redundanten Knoten zur Lastverteilung ist eine vernünftige zentrale *Service Registry/Discovery* unumgänglich. Hierbei fiel die Entscheidung der Kapselung dieser Funktion innerhalb eines eigenen Microservices. Zum einen hat dies den Vorteil, der noch besseren Funktionskapselung und zum anderen bleibt für zukünftige Weiterentwicklungen die Möglichkeit offen, ebenso mehrere Main-Framework Microservice-Instanzen anzubinden. Bei der Wahl des Frameworks zur Service Registrierung fiel die Entscheidung auf die Verwendung von Eureka [219], ein von Netflix selbst entwickeltes und verwendetes Projekt zur Verwaltung und Lokation von Microservice-Instanzen. Eureka arbeitet REST-basiert und besteht aus einem zentralen Eureka Server, der mehrere Clients verwaltet, die sich beispielsweise mit einem Servicenamen beim Eureka Server registrieren. Werden von mehreren Microservice-Instanzen gleiche Servicenamen verwendet, lässt sich Eureka neben einer einfachen Serviceverwaltung auch direkt zur Lastverteilung verwenden und deckt dadurch den gesamten benötigten Funktionsumfang ab. Darüber hinaus wird ein übersichtliches Webinterface geboten, um alle verfügbaren sowie nicht mehr verfügbaren Instanzen zu verwalten.

Angular 5, NodeJS und Node Package Manager

Im Frontend Bereich fiel die Wahl auf das mächtige Framework Angular [25] mit der zum Projektstart aktuellen Version 5. Angular bietet zahlreiche Features, von denen vor allem Routing, Backend Kommunikation, Validierung und Skalierbarkeit besonders hilfreich für dieses Projekt sind. Darüber hinaus werden Entwickler und Entwicklerinnen durch Angular-CLI [26] mit Code Generierung und Build Prozess der Frontend Anwendung unterstützt. Weiters wird die State-of-the-Art Entwicklung von Anwendungen gefördert, Wartung vereinfacht und auch große komplexe Systeme durch Komponenten übersichtlich und verständlich gegliedert. NodeJS [177] und der Package Manager NPM [178] ist für die Kompilierung des Angular Typescript Quellcode zustän-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

dig und wird ab Angular 2 benötigt. Darüber hinaus ermöglicht NPM die einfache Installation von vorgefertigten, existierenden Frontend Modulen.

Material Components

Um ein plattformunabhängiges, *responsive* und funktionales Benutzerinterface zur Verfügung zu stellen, wurden Material Components [166] eingesetzt. Diese verfügen über eine Vielzahl von Komponenten und Funktionsabläufe, um mit adäquatem Aufwand moderne, flexible und skalierende Oberflächen zu bauen und eignen sich dadurch besonders für die Anforderungen des Frameworks.

Docker

Um Elastizität und Skalierbarkeit ebenso wie eine hohe Verfügbarkeit zu ermöglichen, bietet sich die Nutzung von Cloud-Diensten an. Aus diesem Grund wurde bei der Entwicklung des Frameworks ein Fokus auf Cloud Kompatibilität geworfen und die weit verbreitete State-of-the-Art Plattform Docker [78] verwendet.

Docker ist eine Plattform, um Software plattformunabhängig in Form von *Containern* zu verteilen. Ziel ist es, moderne Software-Entwicklungsprozesse und Marktanforderungen einfacher umsetzbar zu machen, die durch die heutige Digitalisierung und den raschen technologischen Wechsel kurze Entwicklungszyklen und agile Herangehensweisen in allen Bereichen fordern.

Docker bietet die Möglichkeit der Unabhängigkeit von Anwendung und Infrastruktur, wodurch der Grundbaustein für agile Entwicklung von portabler Software in flexiblen und kurzen *Time-To-Market* Durchläufen gelegt wird. Weiters geht Docker auf die Anforderungen moderner Software-Entwicklung ein und ermöglicht Infrastruktur unabhängiges Deployment. Ebenso fördert Docker die Zerlegung von großen monolithischen Systemen in kleinere, wartbare Microservices. Unter anderem durch die flexible Wahlmöglichkeit der Infrastruktur und den damit verbundenen Vorteilen, ist auch eine starke Prozess- und Kostenoptimierung möglich und unterstützt zugleich beim Deployment in der Cloud.

Abbildung 9.2 zeigt abschließend grafisch die verwendeten Technologien und State-of-the-Art-Frameworks, die in den einzelnen Komponenten des Frameworks zum Einsatz kommen.

9.3 Implementierung

Dieser Teilkapitel der Arbeit widmet sich wichtigen Implementierungsdetails des Frameworks. Dazu wird neben dem Setup der drei Hauptkomponenten, die Anbindung und Anpassung von den inkludierten Analysewerkzeugen beschrieben, und auf die Implementierung der Analyse sowie der Evaluierung der Analyse-Reports eingegangen.

9.3.1 Microservice Setup

Wie im vorherigen Abschnitt beschrieben wurde, teilt sich das Framework zur Analyse der Android-Anwendungssicherheit in drei eigenständige, aber eng zusammenarbeitende Microservices auf. Darum möchten wir diesen Abschnitt der Arbeit damit beginnen, einen kurzen Überblick über das Grundsetup der Framework Komponenten zu geben.

Das grundlegende Setup der drei Microservices Main-Service, Analysis-Service und Service-Registration unterscheidet sich kaum voneinander und besteht aus einer durch Maven verwalteten Spring Boot Java Anwendung.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

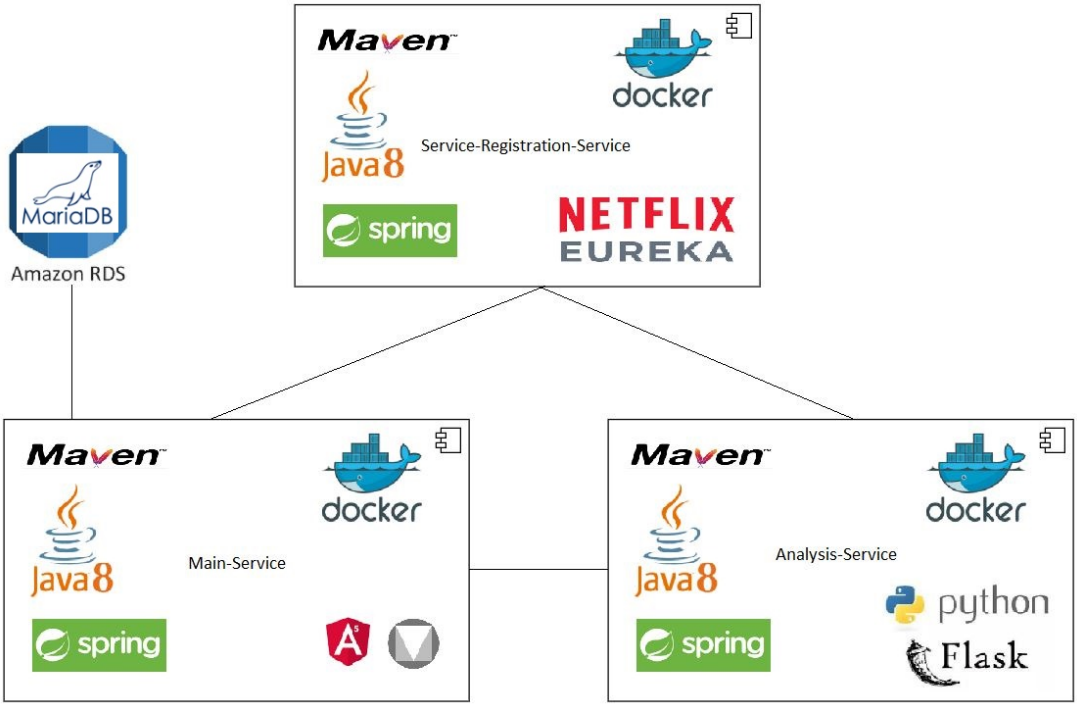


Abbildung 9.2: Einsatz von Technologien und Frameworks in den Microservice Komponenten Service-Registration-Service, Main-Service und Analysis-Service

Die Framework eigene Funktion zur Sicherheitseinstufung von Android-Anwendungen anhand der vorliegenden Werkzeug-Reports, wurde in ein eigenes Modul mit dem Namen `Evaluation` ausgelagert und in das Main-Service sowie das Analysis-Service integriert. Dies ermöglicht die flexible Durchführung der Einstufung innerhalb der unterschiedlichen Microservices. Anhand des `evaluation`-Profils kann beim Start der Microservices entschieden werden, ob die Sicherheitseinstufung durch die verteilten Analyse-Instanzen stattfinden soll, oder innerhalb des Main-Services. Dies bietet den Vorteil, dass neue Evaluierungen der vorhandenen Werkzeug-Reports mit einer einzigen Instanz möglich sind, wodurch anfallende Kosten wesentlich reduziert werden können.

Das Main-Service beinhaltet zusätzlich eine Anbindung zu einer zentral verwalteten Datenbank, wozu das Grundsetup mithilfe von Maven um einen entsprechenden Datenbanktreiber sowie um Spring Boot Data JPA erweitert wurde, das eine flexible Anbindung unterschiedlicher Datenbank Implementierungen und den einfachen Zugriff auf beziehungsweise die Speicherung von Daten ermöglicht.

Kommunikation zwischen den Microservices

Um die Kommunikation zwischen Main und Analysis-Service zu vereinfachen, wurde ein eigenständiges Modul entworfen, das den Namen `Shared` trägt und die sogenannten *Data-Transfer-Objects (DTOs)* enthält. Dieses eigenständige Modul ist sowohl im Main-Service als auch im Analysis-Service über Maven als Abhängigkeit eingebunden, wodurch die beiden Services stets unter Verwendung dieser eindeutigen DTOs kommunizieren können.

Wie in Abschnitt 9.2 erwähnt, wird Spring Cloud Netflix Eureka zur Registrierung und Lokation der Microservice-Instanzen verwendet.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Eureka wurde durch die Abhängigkeit `spring-cloud-starter-eureka-server` mittels Maven in die Spring Boot Anwendung eingebunden. Um den Eureka Service Registrierungsserver aufzusetzen, ist eine einzige Annotation nötig, die den Namen `@EnableEurekaServer` trägt.

Die beiden Microservices Main-Service und Analysis-Service wurden für die Verwendung des Eureka Service-Registration Microservice über die Annotation `(@EnableEurekaClient)` konfiguriert.

Um Requests über den Eureka Service-Registration laufen zu lassen, muss lediglich eine `RestTemplate`-Implementierung mittels der Annotation `@LoadBalanced` gekennzeichnet werden. Sämtliche Requests, die mit dieser RestTemplate-Implementierung versendet werden, laufen über die Eureka Service-Registration [219] und werden auf diese Weise nach dem *Round-Robin* [212] Lastverteilungsprinzip verteilt.

9.3.2 Docker Setup

Um die Anforderungen des Frameworks an Skalierbarkeit und Elastizität umzusetzen, wurde Docker [80] eingesetzt.

In diesem Abschnitt werden einige Grundbegriffe erklären und im Anschluss aufgezeigt, wie Docker in den Microservices verwendet wird.

Docker Container und *Image*

Docker-basiert auf einem Container System [79], um auf diesem Weg Software komplett von allen Abhängigkeiten und Umgebungen zu isolieren. Durch diese Isolation ist es möglich, auf einem Gastsystem mehr Container gleichzeitig zu starten, wodurch beispielsweise die Verwendung von mehreren Microservices auf dem gleichen darunterliegenden System ermöglicht wird. Ein Container ist eine Instanz eines Docker Image und stellt ein standardisiertes Format von Software dar.

Ein Image hingegen ist ein leichtgewichtiges, ausführbares, wiederverwendbares Softwarepaket und beinhaltet alle nötigen Systemwerkzeuge, Bibliotheken, Umgebungseinstellungen und dergleichen, um darin enthaltene Software ausführen zu können. Auf einzelne Docker Images kann ausschließlich lesend zugegriffen werden, jedoch können diese aufeinander aufbauen, wodurch je nach Anforderung ein beliebiges Zusammenbauen von Images möglich ist. Solche Docker Images können mithilfe eines Docker eigenen Formats, sogenannten *Dockerfiles* spezifiziert werden.

Dockerfile

Ein Dockerfile ist eine Textdatei, die zur Erstellung eines Docker Images benötigt wird. Daher wurde für jede der drei Hauptkomponenten des Frameworks ein Dockerfile erstellt, um jeweils ein eigenes Image zu erhalten, das die entsprechende Microservice Implementierung enthält.

Diese Dockerfiles beinhaltet eine Reihe von Befehlen, die auf ein Basisimage ausgeführt werden und ein neues Image beschreiben, wie der Dokumentation [81] entnommen werden kann. Exemplarisch wird an dieser Stelle das relativ einfach aufgebaute Dockerfile des Main-Service beschrieben, das in Listing 9.1 zu sehen ist.

Der Befehl `FROM` beschreibt das Basisimage, auf dem das mit diesem Dockerfile erstellte Image aufbaut und im konkreten Fall bereits eine Java 8 Installation enthält. Als nächsten Schritt wird der Port der Anwendung mithilfe von `EXPOSE` definiert, auf dem das Docker Image lauscht. Um dem Image Artefakte hinzuzufügen, wird der Befehl `ADD` oder `COPY` verwendet, der Daten vom lokalen Dateisystem in das Docker Image einfügt, wie an Hand der JAR-Datei zu sehen ist,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

welche die gebaute Anwendung darstellt. Als letzten Schritt wird der Befehl `CMD` verwendet, der beim Start des Docker Containers einen Kommandline Befehl innerhalb des Containers absetzt und dadurch die Anwendung startet.

Kurz zusammengefasst, erstellt das Dockerfile, das in Listing 9.1 betrachtet werden kann, also ein Image, das auf `frolvlad/alpine-oraclejdk8:slim` aufbaut und um die vollständig gebaute Anwendung erweitert wird. Als letzten Schritt wird die Anwendung innerhalb des Containers gestartet.

```
1 FROM frolvlad/alpine-oraclejdk8:slim
2 EXPOSE 8090:8090
3 COPY ./main-service-0.0.1-SNAPSHOT.jar /main-service.jar
4 CMD ["java","-jar","/main-service.jar"]
```

Listing 9.1: Einfaches Beispiel eines Dockerfile anhand des Main-Service

9.3.3 Verwaltung und Verarbeitung von Android-Anwendungen

Um Anwendungen analysieren zu können, müssen diese zuerst in das Framework oder genauer in die Analyseumgebung gelangen, wozu zwei unterschiedliche Varianten implementiert wurden. Zum einen kann ein Nutzer oder eine Nutzerin Anwendungen über das Benutzerinterface durch Drag&Drop oder einem Dateiauswahlmenü hochladen, wie in Abbildung 9.3 zu sehen, zum anderen direkt durch einen integrierten Google Play Store Downloader nach bestimmten Android-Anwendungen oder Anwendungskategorien suchen und diese zur Analyse ins Framework einfügen, wie Abbildung 9.4 zeigt. Eine erweiterte Funktionalität ermöglicht es, automatisiert Google Play Store-Anwendungen zu laden, die anschließend untersucht werden und deren Sicherheitseinstufung automatisiert durchgeführt wird. Diese Funktionalität ist über die in Abbildung 9.3 dargestellte Schaltfläche `Automated Analysis Mode` erreichbar. Das zentrale Main-Service verarbeitet und verwaltet alle Metadaten sowie Reports zu den einzelnen Anwendungen.

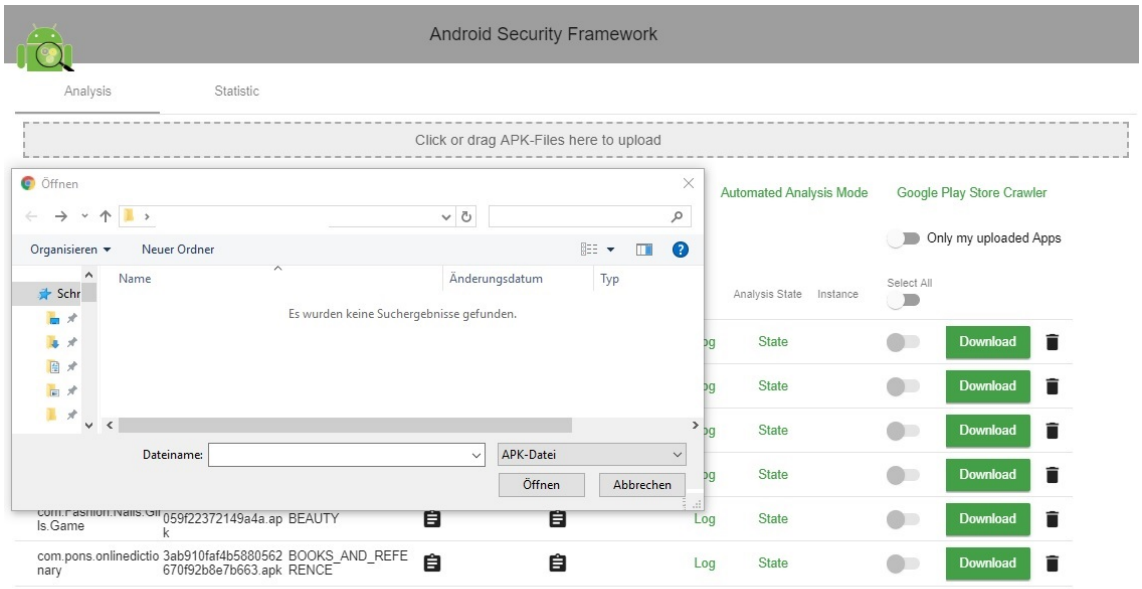


Abbildung 9.3: Upload einer APK-Datei ins Framework über das Dateiauswahlmenü

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

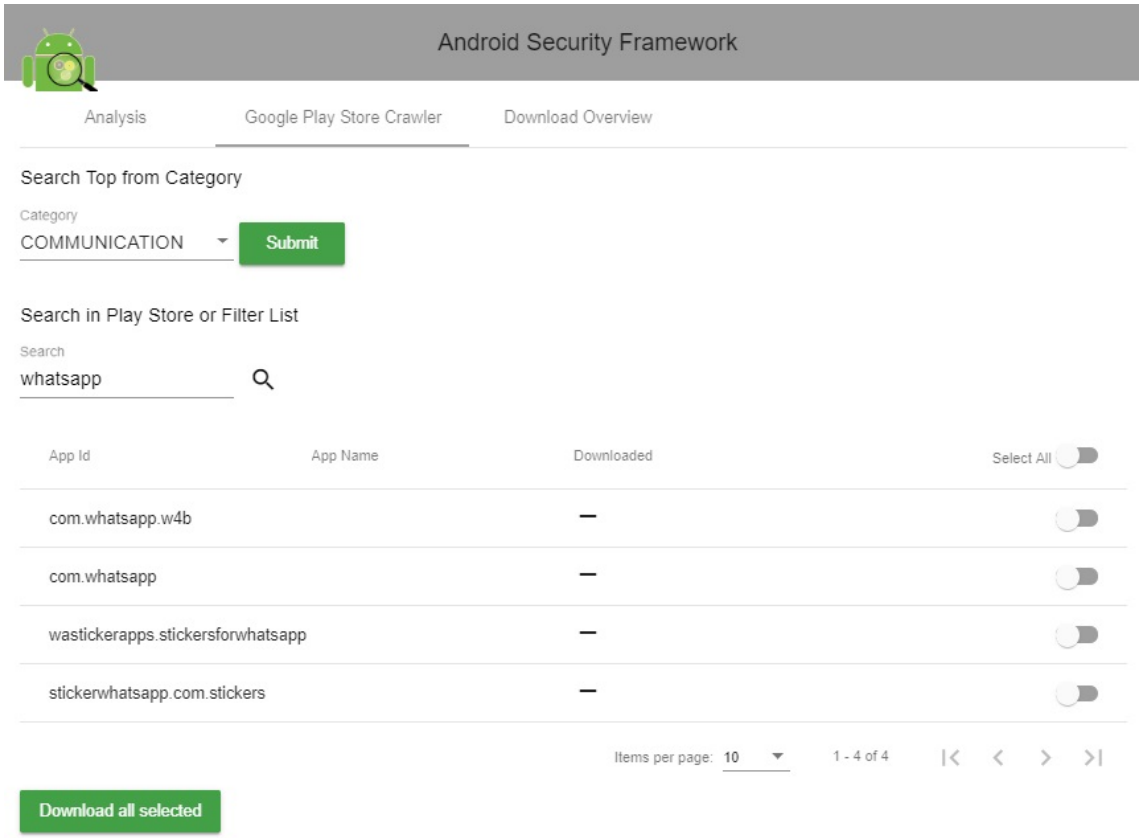


Abbildung 9.4: Upload einer APK-Datei ins Framework über den integrierten Google Play Store Downloader

Da die Architektur des Frameworks eine klare Trennung der einzelnen Analysis-Service Instanzen vorsieht und diese jeweils über einen eigenen abgetrennten Speicher verfügen, muss das Main-Service mithilfe der Service-Registration Komponente bereits beim Hinzufügen einer neuen Anwendung bestimmen, welche Instanz die Analyse der Anwendung übernimmt, wie Abbildung 9.5 zeigt. Erhält das Main-Service durch den `ApkController` die Aufgabe, eine Anwendung zur Analyse hinzuzufügen, wird durch das `UploadService` beziehungsweise das `AnalysisServiceCommunicationService` die Anwendung an die durch die Service-Registation Komponente definierte Analyse-Instanz gesendet. Dort wird durch den `ApkUpload-Controller` und das zugehörige Service die Anwendung letzten Endes ins Dateisystem geschrieben und Metadaten über die Anwendung gesammelt, die im Anschluss an das Main-Service zur Persistierung zurückgegeben werden. Die auf diese Weise gewählte Analysis-Service Instanz ist im Anschluss für das komplette Analyseverfahren zuständig. Zu diesem Zweck speichert das Main-Service als zusätzliches Metadatum zu jeder Anwendung, die eindeutige Route der zuständigen Instanz `(availableOnServer)`.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

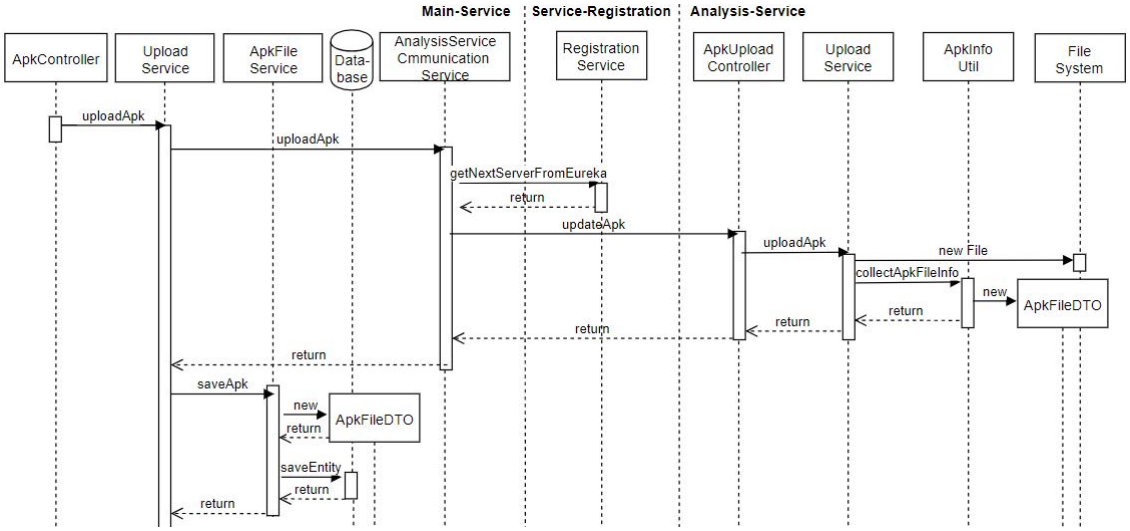


Abbildung 9.5: Vereinfachtes Sequenzdiagramm der Upload Funktionalität des Frameworks

Bei einer bevorstehenden Analyse leitet das Main-Service die Analyseanfrage an die entsprechende Analyse-Instanz weiter, welche die gewünschte Anwendung untersucht und die jeweiligen erhaltenen Berichte an das Main-Service zur Verwaltung und Persistierung zurückgibt. Da während der Analyse sämtliche Quellen einer Anwendung dekompiliert werden, benötigen die Analyse-Instanzen entsprechenden Speicherplatz.

Um die Menge der vorliegenden Quellen sowie Android-Anwendungen in den einzelnen Services zu reduzieren und somit Speicherplatz zu sparen, wurde im Main-Service das `Cleanup Service` zum automatisierten Löschen veralteter Quellen implementiert, wie die Abbildung 9.6 darstellt. Wichtig dabei sind die im Main-Service definierten Parameter `apk.cleanup.rate` und `apk.max.fileage`. Bei jedem Intervall, das mithilfe der Spring Annotation `@Schedule` [218] implementiert wurde, werden anhand des konfigurierten maximalen Alters, alle Anwendungsquellen, deren letzte Verwendung weiter als konfiguriert zurückliegt, entfernt. Dazu besitzt jede Anwendung das Feld `lastUsed`, das in Form eines Zeitstempels angibt, wann die Anwendung hochgeladen oder das letzte Mal analysiert wurde. Im Anschluss werden die entsprechenden Analyse-Instanzen durch das `CleanupService` dazu aufgefordert, die ausgewählten Quellen sowie die Anwendung selbst aus dem Dateisystem zu entfernen. Analyseergebnisse bleiben weiter verfügbar.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

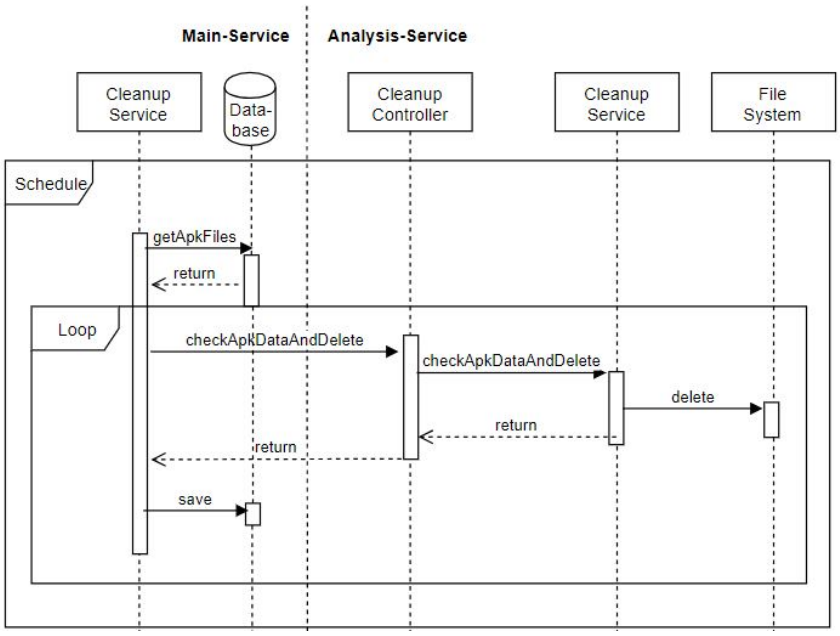


Abbildung 9.6: Vereinfachtes Sequenzdiagramm des Cleanup Service

9.3.4 Integration von Analysewerkzeugen in des Framework

Die Auswahl an Android-Analyse und Reverse Engineering Werkzeugen ist enorm, dennoch ähneln sich kaum Werkzeuge im Funktionsumfang oder der Verwendung, wie in Abschnitt 8.2 beschrieben wurde. Während der Untersuchung und Analyse der ausgewählten Anwendungen stellten sich einige als besonders relevant für die zur Sicherheitsevaluierung von Android-Anwendungen definierten Kategorien und Bereiche dar. Dabei handelt es sich um die Werkzeuge APK-Parser [115], Teilfunktionalitäten des Argus-SAF [30], AndroBugs [111], Androwarn [114], P-Lint [182], QARK [124] sowie SUPER [125].

Bevor auf die Integration dieser konkreten Analysewerkzeuge eingegangen wird, wird das allgemeine Konzept zur Einbindung und Verwendung von Werkzeugen innerhalb des Frameworks näher diskutiert.

Um ein Werkzeug in das Framework einzubetten, muss die abstrakte Klasse `AnalysisWorker` und die definierte `run`-Methode implementiert werden, die den Aufruf eines Analysewerkzeugs ermöglicht sowie für die Weiterleitung des Analyseergebnisses Richtung Report Verarbeitung zuständig ist.

Bei der Analyse einer Android-Anwendung werden die einzelnen konkreten `AnalysisWorker` Implementierungen erstellt und im Anschluss parallelisiert aufgerufen. Dabei kapselt und vereinheitlicht diese Erstellung eine Factory Pattern [108] Implementierung, wodurch eine einfache und rasche Erweiterung durch neue Analysewerkzeuge ermöglicht wird. Dieses Pattern wird im Framework sowohl im Java-Teil, als auch im Python-Teil des Analysis-Service eingesetzt. In Abbildung 9.7 ist anhand des `PlintWorker` sowie `ManifestWorker` ein stark vereinfachtes UML-Klassendiagramm zur Verwendung des Factory Pattern im Java-Teil des Analysis-Services zu sehen. Die Implementierung in der Python-Komponente ist dabei äquivalent aufgebaut. Über den `AnalysisStart-Controller` wird die Factory angestoßen, die anhand der `getAnalysisWorker`-Methode und den als Parameter bereitgestellten Informationen, den entsprechenden konkreten `Analysis-Worker` erstellt und dessen `run`-Implementierung anschließend in einem eigenen Thread gestartet wird.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

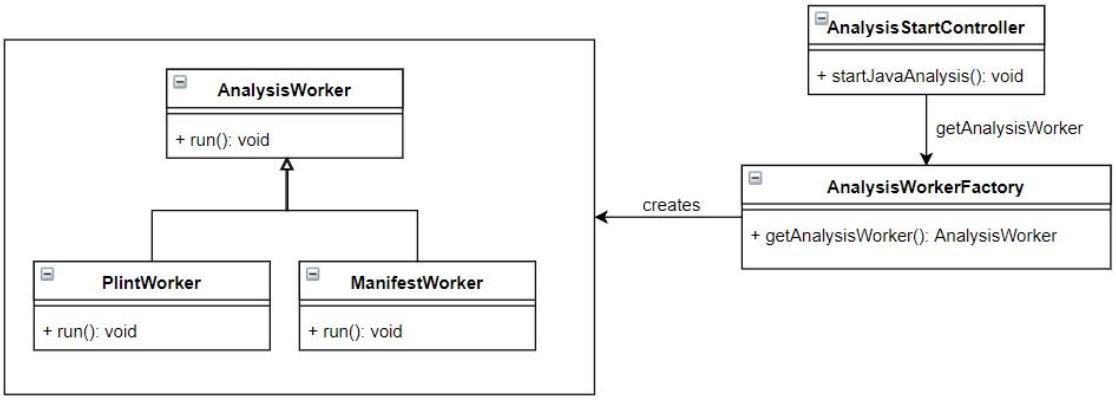


Abbildung 9.7: Vereinfachtes Klassendiagramm des Factory Pattern zur Erstellung von Analysis-Workern

Im nachfolgenden Teil wird auf die konkreten Anbindungen und nötigen Anpassungen der einzelnen Analysewerkzeuge eingegangen.

APK-Parser

Um zu Beginn der Analyse, Informationen über eine Android-Anwendung aus dem `Android-Manifest.xml` zu extrahieren, wurde das APK-Parser-Werkzeug [115] mit Version 2.2.0 als Abhängigkeit in die Java Analyse-Service Komponente eingebunden. Dieser Parser wird in einem eignen Worker dazu verwendet, sämtliche Informationen des `AndroidManifest.xml`, wie Berechtigungen, Komponenten und Intent Definitionen sowie Konfigurationen, zu lesen und für den finalen Report aufzubereiten.

Argus Statisches Analyse-Framework (SAF) / Argus-Amandroid

Leider scheint die recht ausführlich wirkende Dokumentation [30] zum Zeitpunkt unserer Integration im August 2018 zu ungenau oder möglicherweise zu veraltete, um Argus-SAF in ein Java-Projekt als Bibliothek einzubinden und so direkt den vollständigen Funktionsumfang verwenden zu können. Es sind zwar zum Teil Codebeispiele und Tutorials in *Scala* [230] verfügbar, die laut Dokumentation auch in ähnlicher Weise in Java funktionieren sollten, die Codeausschnitte lassen sich aber ohne die genauere Funktionsweise einzelner Klassen zu analysieren nicht in Java anwenden. Um die für das Framework relevante Funktionsweise des Argus-SAF zu integrieren, wurde die Version 3.1.3 des Analysewerkzeugs [30] direkt als JAR aufgenommen und über die Python-Analysekomponente angebunden. Dabei wurde die `API Misuse` Check Funktionalität angebunden, bei der Argus `HIDE_ICON`, `CRYPTO_MISUSE`, `TLS_MISUSE` Probleme aufdeckt, wie in Abschnitt 8.2.2 beschrieben wurde.

Um Argus-SAF verwenden zu können, wurde wie für alle angebundenen Werkzeuge, ein eigener Worker erstellt, der für den Aufruf der Argus-SAF Analyse zuständig ist. Dabei werden der Reihe nach die Funktionalitäten von Argus-SAF zu den API-Verwendungen angewendet. Das Ergebnis der Analyse durch wird im Anschluss in der Java-Analysekomponente verarbeitet und an das Main-Service zum Speichern weitergereicht. Da Argus-SAF direkt ein Analyseergebnis in Text-Format ausgibt, ist hier kein weitere Aufbereitung des Ergebnisses für die anschließende Verarbeitung nötig.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

AndroBugs

AndroBugs [111] Version 1.0.0 stellt für das Framework und die Sicherheitseinstufung ein besonders wichtiges Analysewerkzeug dar, da dieses eine breite Menge der OWASP Top 10 beziehungsweise der CMU Regeln abdeckt. Für den Einbau des Python Frameworks wurden zwei wichtige Anpassungen der Funktionsweise des Werkzeugs durchgeführt. Zum einen Anpassung die Klartext Ausgabe auf JSON-Format, um die Analyseergebnisse auf diese Weise mit geringen Aufwand im Analyse-Framework auslesen und weiterverarbeiten zu können. Zum anderen wurde das Analyseergebnis von AndroBugs dahingehend angepasst, dass nicht gefundene Sicherheitslücken auch nicht dokumentiert werden. Dies erleichtert dem Framework die weitere Verarbeitung, da jeder dokumentierte Eintrag eine Sicherheitslücke darstellt. Die trivial wirkende Anpassung der bestehenden Funktionalität stellte sich allerdings, aufgrund fehlender Struktur der AndroBugs Implementierung, doch als komplexer als erwartet dar.

Den Aufruf von AndroBugs übernimmt ein eigener `AndroBugsWorker` in der Python-Analysekomponente. Nach erfolgreicher Analyse liest dieser Worker den JSON Analyse-Report ein und reicht diesen über die Java-Analysekomponente weiter an das Main-Service.

Androwarn

Für die Einbindung von Androwarn [114] wurden die für das Framework weniger relevanten oder bereits durch andere Werkzeuge abgedeckten Analysen wie die Untersuchung des `Android-Manifest.xml` entfernt, um Ausführungszeit zu sparen.

Wie bei der Werkzeugbeschreibung von Androguard [112] in Abschnitt 8.2.3 beschrieben, scheiterten alle Versuche die interessante Funktionalität von Androguard zu testen und in weiterer Folge zu integrieren. Da jedoch Androwarn auf Androguard aufbaut, konnten im Zuge der Integration von Androwarn, dieses um weitere Androguard Funktionen erweitert werden. Dadurch analysiert die von uns modifizierte Version von Androwarn zusätzlich die Verwendung von *Native-, Dynamic-* und *Reflection Code* sowie Obfuscation einer Android-Anwendung.

Darüber hinaus wurde das Ausgabeformat des Berichts von Androwarn auf JSON angepasst, um wie auch bei anderen Integrationen, die Ergebnisse direkt ins Framework einlesen und weiter verarbeiten zu können. Zur letzten Anpassung des Analysewerkzeugs selbst, zählt die Entfernung der Report Generierung in HTML-Format, die eine irrelevante Funktionalität des Werkzeugs für diesen Anwendungsfall darstellt und wodurch die Abhängigkeit von Jinja2 [243] entfernt und die Komplexität und Größe des Frameworks gering gehalten werden konnte.

Die Integration in das Framework selbst wurde durch einen eigenen Worker in der Python-Analysekomponente umgesetzt, der Androwarn startet und im Anschluss den Report in JSON-Format verarbeitet und zur zentralen Persistierung weiter reicht.

P-Lint

Da die Analyse von P-Lint [122] direkt dekompilierten Quellcode von Android-Anwendungen verarbeitet und die Java Implementierung der Analyse selbst sehr überschaubar ist, wurde die Analysefunktionalität direkt in die Java-Analysekomponente aufgenommen. Dadurch konnte die Implementierung von P-Lint mit geringen Aufwand auf die Anforderungen des Frameworks angepasst werden.

QARK

QARK ist ein sehr aktuelles Analysewerkzeug, das während der Framework Entwicklung nicht nur laufend weiterentwickelt wurde, sondern in einer eigenen Branch Version auch völlig um-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

strukturiert wird. Da die Entwickler und Entwicklerinnen selbst darauf aufmerksam machen, dass die umstrukturierte Branch Version noch sehr unstabil läuft, wurde in dieser Arbeit auf der Master Version [124] aufgebaut.

Zur Einbindung des Python-basierten Analysewerkzeugs in das Framework mussten am Werkzeug selbst keine nennenswerten Anpassungen durchführt werden, da dieses über eine breite Konfigurationsmöglichkeit verfügt. Mithilfe des `AnalyseWorker`-Konzepts konnte das Werkzeug problemlos eingehängt werden. QARK erhält zur Analyse direkt die bereits vorliegenden dekompilierten Quellen einer Android-Anwendung, wodurch hier kein erneutes Dekompilieren nötig ist und ein großer Teil der Analysedauer eingespart werden kann.

Um den Report von QARK, der in HTML-Format vorliegt, weiter verarbeiten zu können, wurde ein eigener Parser implementiert, der für die Sicherheitsevaluierung relevanten Informationen extrahiert und aufbereitet.

SUPER

Da Veränderungen der bestehenden SUPER Implementierung ohne fundierte Kenntnisse von Rust [203] kaum möglich sind, wurde bei der Anbindung des SUPER Analysewerkzeugs die Version 0.4.1 [125] unverändert in Binary Form inkludiert. Dennoch konnte der SUPER Analyseprozess dahingehend konfiguriert werden, dass bereits vorliegende dekompilierte Quellen untersucht werden, um ein erneutes dekompilieren einzusparen.

Ähnlich wie bei den anderen beschriebenen Integrationen der unterschiedlichen Werkzeuge liegt bei einer Analyse durch SUPER der Report in Form einer JSON-Datei vor, die nach der Analyse eingelesen und weiterverarbeitet wird.

9.3.5 Report Verarbeitung der integrierten Analysewerkzeuge

Nach jeder Anwendungsanalyse durch ein konkretes Analysewerkzeug, wird der Report des Werkzeugs, durch den `AnalyseWorker` über die dafür bereitgestellte Framework Logik, an die zuständige Implementierung des `ReportParser` weitergeleitet.

Um den Ablauf sowie die Implementierung des dafür eingesetzten Chain of Responsibility Pattern verständlich darzustellen, wird an dieser Stelle auf das UML-Diagramm in Abbildung 9.8 eingegangen und die Verarbeitung der Reports erklärt.

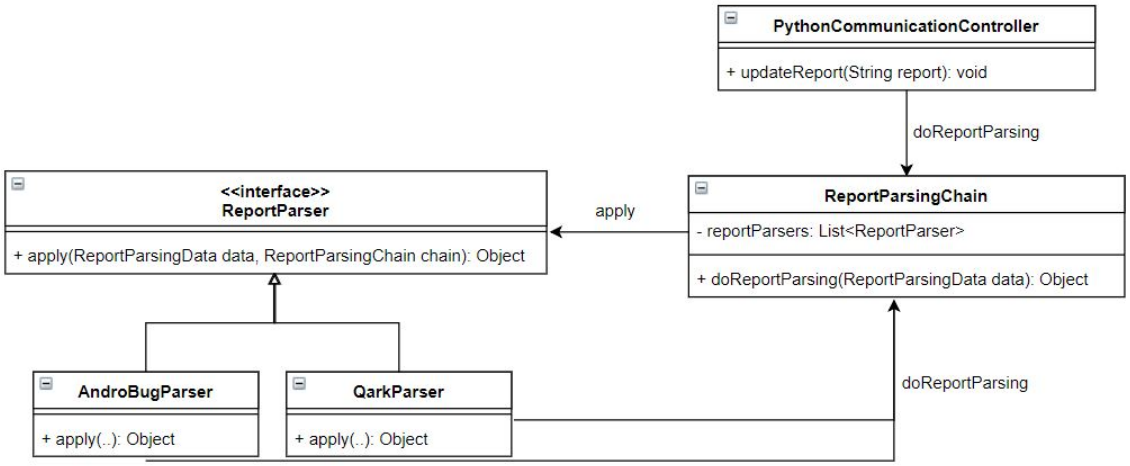


Abbildung 9.8: Vereinfachtes Klassendiagramm des Chain Pattern zum Parsen der Reports von unterschiedlichen Analysewerkzeugen

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Der `PythonCommunicationController` erhält die Ergebnisse der einzelnen Analysewerkzeuge in Textform und verwendet eine für das Analysis-Service global konfigurierte `ReportParsingChain`, die für jedes Analysewerkzeug einen eigenen `ReportParser` beinhaltet und dessen `doReportParsing`-Methode für das Aufrufen dieser Parser in der vorkonfigurierten Reihenfolge verantwortlich ist.

Die Implementierung der `doReportParsing`-Methode ist Listing 9.2 zu entnehmen. Die konfigurierte Liste an `ReportParser` wird durch `reportParsers` abgebildet. Innerhalb der `doReportParsing`-Methode wird jeweils der Parser aufgerufen, der sich an der aktuellen Position der Kette befindet. Die Position innerhalb der Kette wird dabei inkrementiert, um im nächsten Durchlauf den darauffolgenden konfigurierten Parser zu adressieren.

```
1  public class ReportParsingChain
2  {
3    private final List<ReportParser> reportParsers;
4
5    public DefaultReportParsingChain(List<ReportParser> reportParsers)
6    {
7      this.reportParsers = reportParsers;
8    }
9
10   public Object doReportParsing(ReportParsingData reportParsingData)
11   {
12       final int routingChainPosition =
13           reportParsingData.getReportChainPosition();
14
15       if (reportParsers.size() > routingChainPosition)
16       {
17           reportParsingData
18               .setReportChainPosition(routingChainPosition + 1);
19
20           return reportParsers.get(routingChainPosition)
21               .apply(reportParsingData, this);
22       }
23       return null;
24   }
25 }
```

Listing 9.2: Implementierung des Aufrufs der einzelnen ReportParser innerhalb der doReportParsing-Methode der ReportParsingChain

Jeder Parser, in Abbildung 9.8 vereinfacht durch die beiden Implementierungen `AndroBugParser` und `QarkParser` dargestellt, implementiert eine `apply`-Methode, die im Falle einer erkannten Zuständigkeit, das Parsen des Report Klartexts in ein entsprechendes Objekt übernimmt oder andernfalls die Zuständigkeit an die `ReportParsingChain` zurückgibt. Listing 9.3 zeigt eine Pseudoimplementierung einer typischen `apply`-Methode.

```
1 public Object apply(ReportParsingData reportParsingData,
  ↪ ReportParsingChain chain)
2 {
3     if (i_am_responsible)
4     {
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
5            return doReportParsing(reportParsindData.getPlainReport());
6        }
7        return chain.doReportParsing(reportParsingData);
8    }
```

Listing 9.3: Pseudoimplementierung einer typischen apply-Methode eines ReportParsers

Die Verwendung des Chain Pattern [108] an dieser Stelle der Framework Implementierung, ermöglicht eine einfache Erweiterbarkeit durch neue Analysewerkzeuge sowie einen möglichst geringen Wartungsaufwand bei Änderungen an den einzelnen Parsern. Darüber hinaus sind die unabhängigen Implementierungen getrennt und das Parsen der Report Klartexte passiert transparent in einem abgetrennten Modul.

Nach dem Parsen der einzelnen Reports werden diese durch ein zentrales Transferobjekt an das Main-Service zur Persistierung weitergereicht. Die Persistierung der einzelnen Reports im Main-Service ist unter der Verwendung des Facade Pattern [108] umgesetzt, wodurch die dahinterliegende Logik der Persistierung weitgehend von der Kommunikation mit dem Analysis-Service getrennt bleibt und die Persistierung hinter der Fassade relativ einfach durch neue Reporttypen neuer Analysewerkzeuge erweitert werden kann. Wie das Facade Pattern dabei implementiert wurde, zeigt Abbildung 9.9.

Der `ReportController` nimmt die Aufrufe des Analysis-Service entgegen und leitet die Anfragen zum Report Update weiter an die `UpdateReportFacade`, welche eine `updateReport`-Methode implementiert. Diese Methode übernimmt die Kommunikation zu den Analysewerkzeugspezifischen Implementierungen des `GernericReportServices`, welche abhängig von der Struktur der einzelnen Analysewerkzeug-Reports die Persistierung übernehmen.

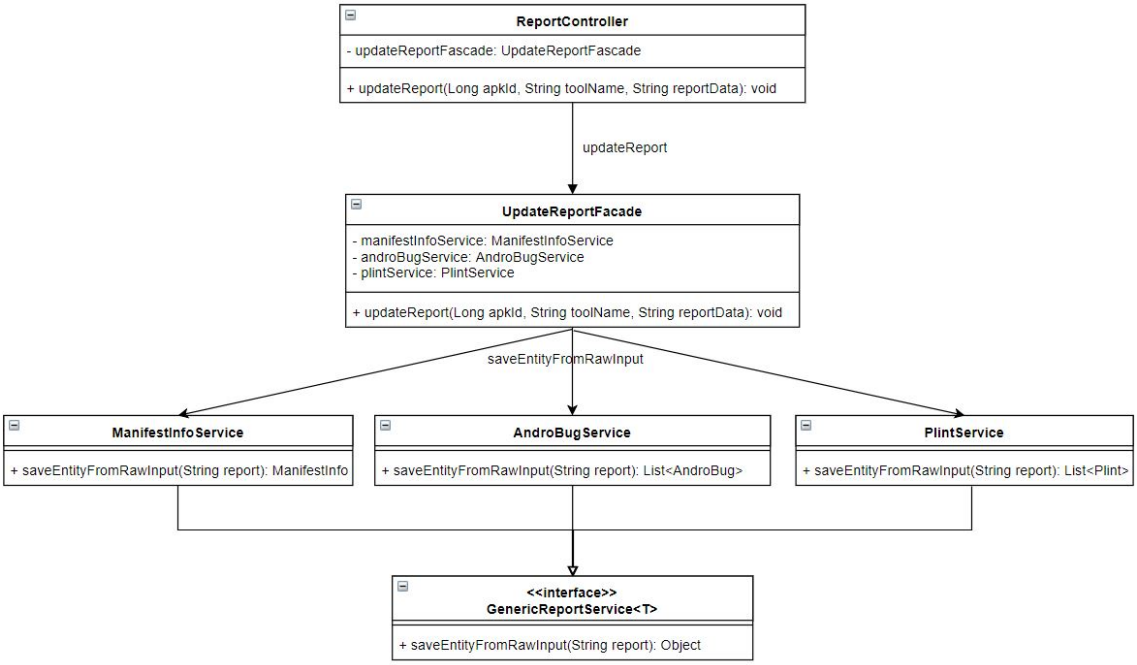


Abbildung 9.9: Vereinfachtes Klassendiagramm des Facade Pattern zum Persistieren der Reports von unterschiedlichen Analysewerkzeugen

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

9.3.6 Ablauf der Android-Anwendungsanalyse und Sicherheitseinstufung

An dieser Stelle werden zusammenfassend, anhand der vereinfachten Sequenzdiagramme in den Abbildungen 9.10 und 9.11, die wichtigsten Bereiche der Android-Anwendungsanalyse durch das Framework dargestellt.

Erhält das Analysis-Service durch den `AnalysisStartController` den Aufruf zum Analysieren einer Android-Anwendung, leitet die Java-Komponente diesen Aufruf an die Python-Komponente weiter, die im ersten Schritt die Dekompilierung der Android-Anwendung mithilfe des `DecompileWorker` durchführt. Dieser verwendet die `DecompilationWorkerFactory`, um die einzelnen `AnalyseWorker` der Werkzeuge ApkTool, dex2jar und JD-Core zu erstellen, die im Anschluss die Android-Anwendung für die weitere Analyse dekompilieren und vorbereiten. Am Ende dieses Vorgangs liegt der analysierbare Quellcode der Android-Anwendung vor und die eigentliche Anwendungsanalyse der integrierten Werkzeuge kann beginnen. Dazu startet das `AnalysisService` der Python-Komponente parallel die einzelnen angebundenen Werkzeuge. Gleichzeitig wird die Java-Komponente benachrichtigt, welche die durch die Java-Komponente angebundenen Werkzeuge startet.

Sobald ein Werkzeug ihre Analyse beendet hat, wird vom Worker selbst über die vom Framework bereitgestellten Schnittstellen, der Report des Analysewerkzeugs an das `ReportService` übermittelt. An dieser Stelle beginnt die Verarbeitung des jeweiligen Reports mithilfe der implementierten `ReportParsingChain`. In dem dargestellten Prozess wird ein Report des Werkzeugs QARK verarbeitet. Demnach übernimmt der `QarkParser` die Aufbereitung des Reports. Der fertig geparste und weiter verarbeitbare Report wird im Anschluss durch eine vom Framework bereitgestellte einheitliche Schnittstelle an das Main-Service zur Persistierung weitergeleitet. Im Main-Service wird anhand dieses Reports, durch die `UpdateReportFacade`, das für die Persistierung verantwortliche Service bestimmt. Dieses Service abstrahiert durch die Implementierung des Facade Pattern die Erstellung und Persistierung des entsprechenden Objekts, wie anhand des `QarkService` beziehungsweise `QarkReport` in Abbildung 9.11 zu sehen ist.

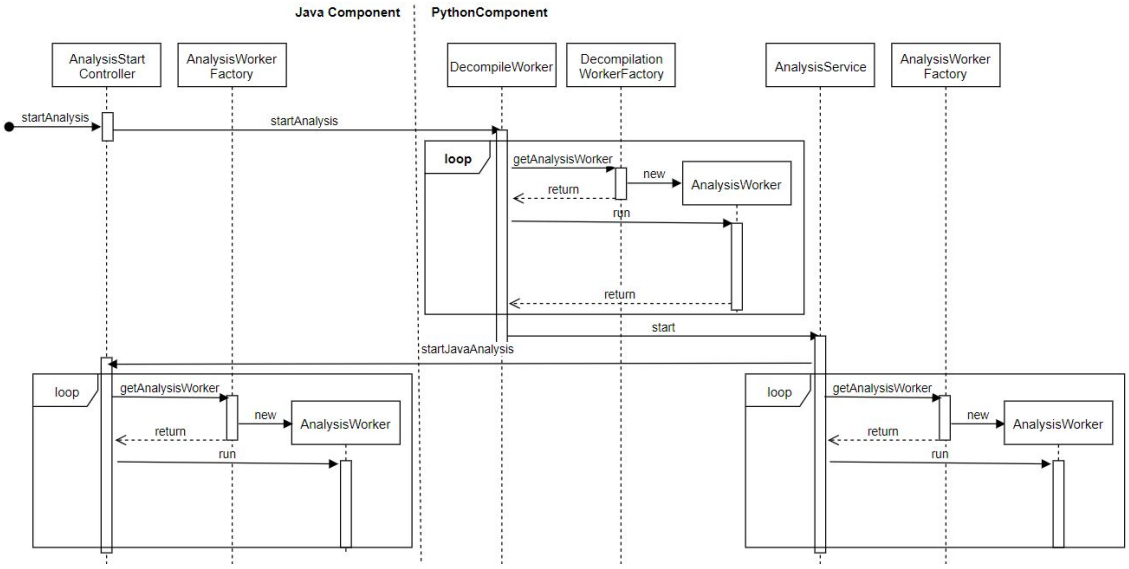


Abbildung 9.10: Vereinfachtes Sequenzdiagramm der Analyse Funktionalität des Frameworks

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

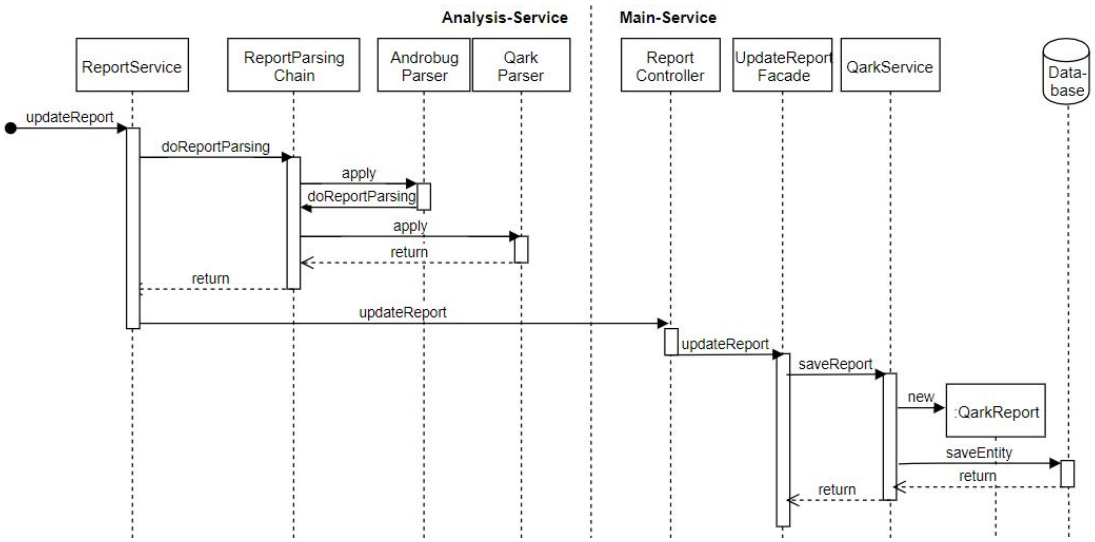


Abbildung 9.11: Vereinfachtes Sequenzdiagramm der Qark Report Verarbeitung des Frameworks

Die persistierten Reports der einzelnen Analysewerkzeuge werden in einem weiteren Schritt zur Framework eigenen Sicherheitseinstufung verwendet, bei der die von den einzelnen Werkzeugen analysierten Teilbereiche für die Sicherheitseinstufung der definierten Kategorien, mangelnder Angriffsschutz, Sicherheitsinvalidierung, Zugriffskontrolle, Datenlecks und Eingabevalidierung, verwendet werden. Welche erkannten Schwachstellen eines Analysewerkzeugs welcher Kategorie zugeordnet werden und welches Risiko eine erkannte Schwachstelle für diese Kategorie bedeutet, wurde frei konfigurierbar implementiert. Auf diese Weise können vorhandene Analyse-Reports jederzeit mit unterschiedlichen Sicherheitsrisikokonfigurationen neu ausgewertet werden. Um das Sicherheitsrisiko einer Kategorie zu definieren, wurden vier verschiedene Risikostufen (`SECURE`, `WARNING`, `POTENTIALINSECURE`, `INSECURE`) eingeführt. `SECURE` beschreibt dabei einen Fund, der im weiteren Report nicht als relevant eingestuft und somit ignoriert wird. Als `WARNING` werden Funde von Codeteilen eingestuft, die vermieden werden sollten, und deren Funktionalität sowie Relevanz für die Anwendungsfunktionalität überprüft und hinterfragt werden sollte. `POTENTIALINSECURE` definiert ein potenzielles Sicherheitsrisiko, bei dem es sich empfiehlt, mögliche Auswirkungen auf die Anwendungssicherheit durch weitere manuelle Analyse zu überprüfen. `INSECURE` deutet auf eine Abweichung von State-of-the-Art Empfehlungen wie etwa des OWASP [179], der Android-spezifischen Regeln der CMU [22] oder der Android Best Practice Dokumentation [29] hin, die direkte Auswirkungen auf die Anwendungssicherheit und somit ein Sicherheitsrisiko darstellen.

Jedem Analysebereich und jedem sicherheitsrelevanten Fund eines Analysewerkzeugs wird eine dieser Risikostufen zugeordnet. Listing 9.4 zeigt eine Beispielkonfiguration für die Evaluierung des ICC Teilbereichs (siehe Abschnitt 7.3.1) anhand der Analyseergebnisse der Werkzeuge AndroBugs, QARK und SUPER. Die Konfiguration selbst wird in YAML-Format vorgenommen, die beim Start des entsprechenden Microservice durch Spring verarbeitet und in Form einer Map eingelesen wird. Als Schlüssel dieser Map werden jeweils die Unterkategorien der fünf Evaluierungsbereiche (siehe Abschnitt 7) verwendet. Im angeführten Beispiel 9.4 durch die Unterkategorie Inter-Component Communication (`ICC`) dargestellt. Als Wert ist jedem Schlüssel eine weitere Map zugeordnet, welche als Schlüssel die einzelnen Analysewerkzeuge enthält und als Wert wiederum ein Mapping von den im Analyse-Report erkannten Sicherheitslücken sowie das durch das Framework zugeteilte Sicherheitsrisiko. Enthält etwa der Analyse-Report des Werkzeugs AndroBugs die Sicherheitslücke mit dem Tag `PERMISSION_EXPORTED`, so wird diese erkannte Sicherheitslücke vom Framework als Warnung eingestuft. Auf diese Weise können Anwender und

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Anwenderinnen des Frameworks selbst definieren, welche erkannten Sicherheitslücken der einzelnen Werkzeuge für den konkreten Anwendungsfall relevant sind und wie kritisch diese in weiterer Folge eingestuft werden.

```
1  evaluation:
2    issueDefinition:
3      ICC:
4        callAndroBug:
5          PERMISSION_EXPORTED: WARNING
6          PERMISSION_EXPORTED_GOOGLE: WARNING
7          PERMISSION_PROVIDER_EXPLICIT_EXPORTED: WARNING
8          PERMISSION_INTENT_FILTER_MISCONFIG: WARNING
9          PERMISSION_NO_PREFIX_EXPORTED: POTENTIALINSECURE
10         PERMISSION_PROVIDER_IMPLICIT_EXPORTED: POTENTIALINSECURE
11       callQark:
12         manifest: WARNING
13       callSuper:
14         Exported provider: WARNING
15         Exported receiver: WARNING
16         Exported activity: WARNING
17         Exported activity-alias: WARNING
18         Exported service: WARNING
```

Listing 9.4: Beispielkonfiguration der ICC Sicherheitsevaluierung

9.4 Auswertung der Sicherheitseinstufung

Um am Ende der automatisierten Anwendungsanalyse die vorliegenden Sicherheitseinstufungen weiter auszuwerten und zu evaluieren, wurde eine Funktionalität zur statistischen Auswertung hinzugefügt, die sich maßgeschneidert auf die für diese Arbeit relevanten Bereiche fokussiert. Ein Teilbereich der statistischen Auswertung beschäftigt sich mit der Erfolgsrate der einzelnen Werkzeuganalysen sowie mit der Ausführungszeit dieser. Das Hauptaugenmerk der Evaluierungsstatistik liegt bei der Auswertung der fünf, in Abschnitt 6 definierten Sicherheitskategorien sowie deren Unterkategorien, um herauszufinden, von welchen Bereichen am häufigsten sicherheitskritische Risiken ausgehen. Darüber hinaus werden die am häufigsten auftretenden konkreten Sicherheitsrisiken hervorgehoben. Ebenso wird anhand der Evaluierungsergebnisse erhoben, in welchen Google Play Store-Anwendungskategorien am meisten Sicherheitslücken vertreten sind, und in welchen der definierten Sicherheitskategorien diese auftreten. Dies ermöglicht den direkten Vergleich der einzelnen Google Play Store-Kategorien. Die statistische Zusammenfassung der Sicherheitsevaluierung wurde so aufgebaut, dass die jeweiligen konkreten Evaluierungsreports hinter den statistischen Auswertungen verlinkt sind. Auf diese Weise kann ein rascher Einblick in die zugrunde liegenden Daten genommen werden. Ebenso wird eine stichprobenartige händische Auswertung der Ergebnisse ermöglicht.

9.5 Verwendung des Frameworks zur Analyse von Anwendungen

In diesem Abschnitt wird die Verwendung des Frameworks sowie der beiden Analysemodi besprochen und auf die anschließende Anwendungseinstufung eingegangen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

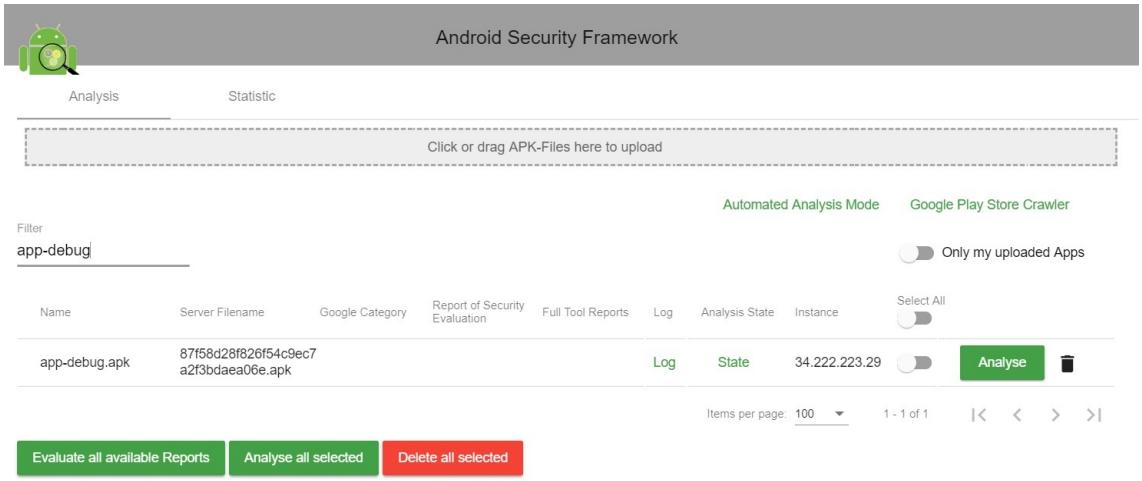


Abbildung 9.12: Framework Startseite mit Android-Anwendungsübersicht

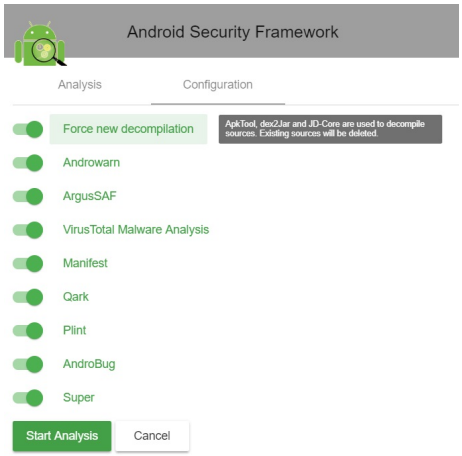


Abbildung 9.13: Konfiguration der Analyse

9.5.1 Analyse Modi des Frameworks

Manuelle Analyse

Der Analysemodus von manuell ausgewählten Anwendungen bietet dem Anwender und der Anwenderin zwei verschiedene Einsatzmöglichkeiten. Android-Anwendungen können im APK-Format per Drag&Drop beziehungsweise Dateiauswahl zur Analyse hochgeladen werden. Dabei wird bereits beim Upload die Analyse-Instanz fixiert, die diese Anwendung verarbeitet. In Abbildung 9.12 ist dies anhand der manuell ins Framework hochgeladenen Anwendung `app-debug.apk` im `Instance`-Feld zu sehen. Nach erfolgreichem Upload kann die Analyse der Anwendung durch einen Klick auf `Analyse` gestartet werden. Vor dem eigentlichen Start ist allerdings noch die Konfiguration beziehungsweise Werkzeugauswahl zur Analyse erforderlich, wie Abbildung 9.13 zeigt.

Alternativ zum manuellen Upload von Android-Anwendungen, können diese auch direkt aus dem Google Play Store geladen werden. Dazu kann mithilfe der durch *GPlayCli* [130] Version 3.25 angebundenen Google Play Store-API nach Anwendungen gesucht und diese heruntergeladen werden, wie in Abbildung 9.14 zu sehen ist. Das blau markierte Textfeld dient dabei zugleich als Filter des Suchergebnisses, der durch Tippen aktiv wird, als auch als Eingabemöglichkeit des Google Play Store Suchbegriffs. Die Suche im Google Play Store

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

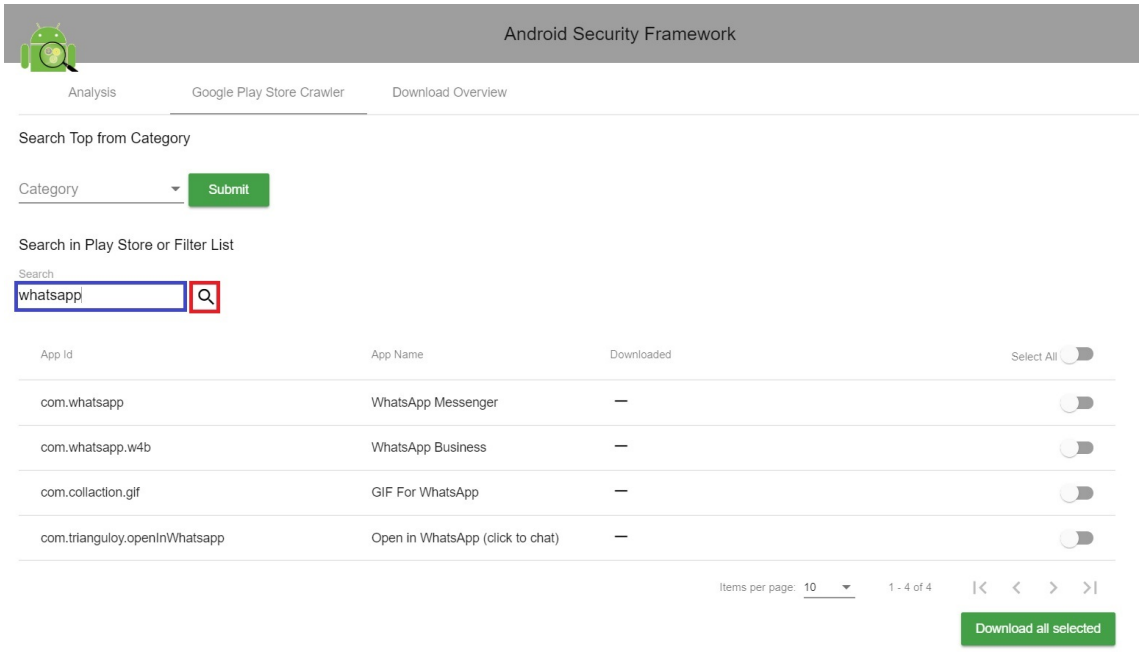


Abbildung 9.14: Anwendungsdownload aus dem Google Play Store

kann unter Verwendung des rot markierten Knopfs gestartet werden, wodurch im Anschluss alle Funde in der darunterliegenden Liste dargestellt werden. Als zweite Suchfunktion wurde die Suche der Top-Anwendungen aus den unterschiedlichsten Kategorien eingebaut. Dazu kann unter der Verwendung des Auswahlmenüs eine entsprechende Google Play Store-Kategorie ausgewählt werden. Heruntergeladene Anwendungen erscheinen ebenso, wie in Abbildung 9.12 zu sehen, in der Anwendungsübersicht, von wo aus die Analyse gestartet werden kann.

Automatische Analyse

Neben der manuellen Analyse ist es möglich, pro Framework Setup beziehungsweise genauer pro Main-Service, eine automatische Analyse zu starten. Aus technischen Gründen ist während eines automatisierten Analysedurchlaufs die manuelle Funktionalität nur eingeschränkt verfügbar. Dies ist durch ausgegraute Knöpfe bemerkbar. Zur Steuerung der automatisierten Analyse existiert die in Abbildung 9.15 dargestellte Bedienoberfläche. Nach dem Start des automatisierten Modus, der durch einen Klick auf `Start automated GPlay Analysis` ausgelöst wird, wird die Warteschlange der Analyse mit jeweils zwei Anwendungen jeder der 58 Google Play Kategorie (116 Anwendungen) befüllt. Die aktuelle Länge der Warteschlange wird durch `Apps to analyse` dargestellt. Sind die ersten zwei Anwendungen jeder Kategorie fertig analysiert und ist die Warteschlange leer, wird diese mit den nächsten beiden Anwendungen jeder Kategorie befüllt. Diese Iteration wiederholt sich so lange, bis die durch `google.play.category.numberofapps` und `google.play.category.iterations` anpassbare Menge an Anwendungen analysiert ist. Während der automatisierten Analyse wird im Bereich `Instances Load`, zu jeder Analysis-Service Instanz, die aktuell laufende Anwendungsanalyse gelistet. Jede Instanz bearbeitet dabei jeweils eine Anwendung gleichzeitig. Wird die automatisierte Analyse gestoppt, werden laufenden Anwendungsanalysen nicht abgebrochen, jedoch keine neuen Analysen mehr automatisiert gestartet.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

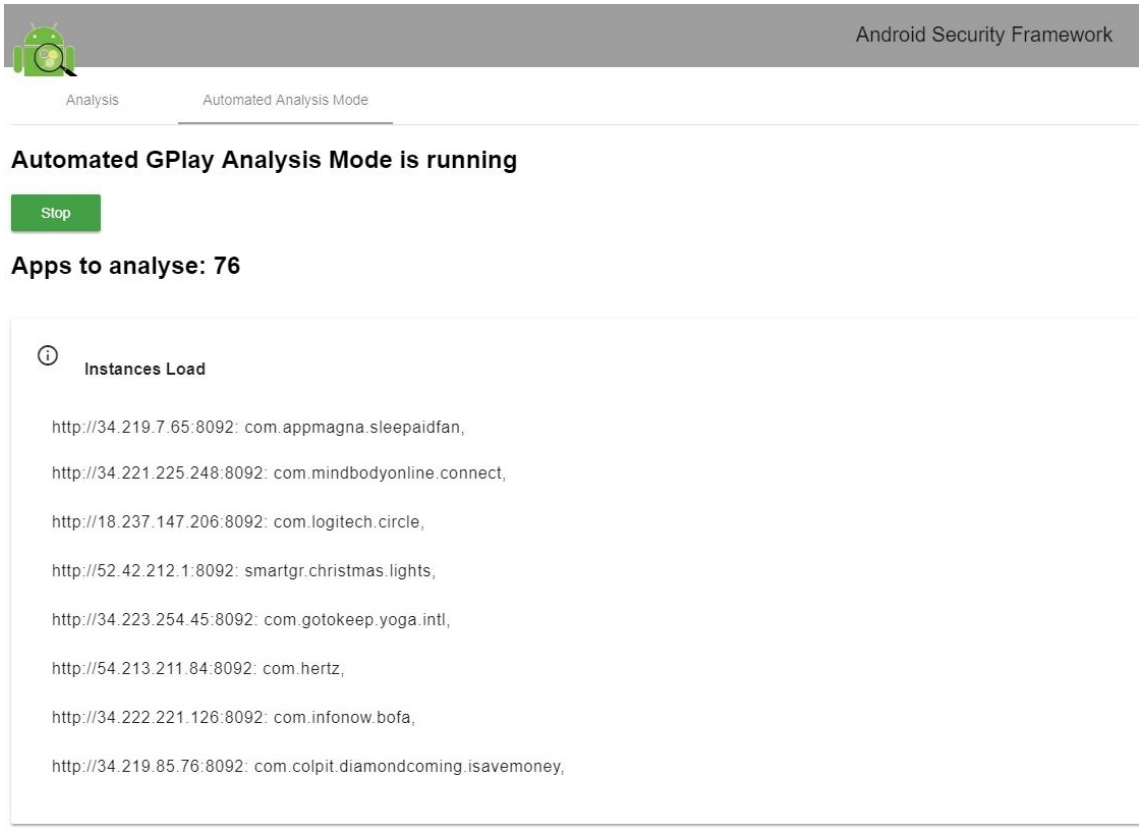


Abbildung 9.15: Start der automatisierten Google Play Store-Anwendungsanalyse

9.5.2 Sicherheitseinstufung von Android-Anwendungen

Sobald die ersten Werkzeug-Reports vorhanden sind, erscheint in den Spalten `Full Tool Reports` und `Report of Security Evaluation`, zu sehen in Abblidung 9.12, ein Report Symbol, mit dem zum entsprechenden zusammengefassten Report beziehungsweise der Sicherheitseinstufung weitergeleitet wird. Die Evaluierung wird beim Öffnen auf die Aktualität geprüft und gegebenenfalls neu ausgewertet. Nur ausgewertete Reports und somit evaluierte Anwendungen werden anschließend bei der Zusammenfassung und statistischen Auswertung der Sicherheitseinstufung unter `Statistic` berücksichtigt. Die Auswertung aller verfügbaren Reports ist durch `Evaluate all available Reports` möglich und wird zusätzlich alle vier Stunden automatisiert durchgeführt.

9.6 Einbindung neuer Analysewerkzeuge

In Abschnitt 9.3.4 und 9.3.5 wird vor allem darauf eingegangen, wie die gewählten konkreten Analysewerkzeuge an das Framework angebunden, angepasst und deren Analyseergebnisse in das Framework eingebracht wurden. In diesem Kapitel wird ein kurzes allgemeines Konzept für die Integration neuer Analysewerkzeuge beschrieben.

9.6.1 Vorbedingungen

Um ein Analysewerkzeug an das Framework anbinden zu können, muss dieses zwei Voraussetzungen erfüllen. Das Werkzeug muss erstens einen geeigneten automatisierten Analysedurchlauf von Android-Anwendungen bieten, der entweder direkt ins Framework aufgenommen oder über einen zentralen Befehl gestartet werden kann, und zweitens ein Analyseergebnis in einer Form von

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

einlesbaren, serialisierbaren sowie weiter verarbeitbaren Report liefern. Erfüllt ein Analysewerkzeug diese Anforderungen oder ist es, wie bereits in Abschnitt 9.3.4 gezeigt, durch Anpassungen in der Lage diese Voraussetzungen zu erfüllen, kann das Werkzeug integriert werden.

9.6.2 Wahl der zuständigen Analyse-Service Komponente

Durch die bewusst gewählte Architektur und die Trennung des Analysis-Service in eine Python- und eine Java-Komponente, bietet das Framework zwei verschiedene Möglichkeiten der Anbindung von Werkzeugen. Bevor die Integration und die dafür nötigen Implementierungen durchgeführt werden können, muss entschieden werden, welche Komponente die Anbindung des neuen Analysewerkzeugs übernimmt. Im Allgemeinen empfiehlt es sich Java-basierte Werkzeuge, die als Bibliothek verfügbar sind oder über eine klar definierte API verfügen, mithilfe der Java-Komponente anzubinden. Alle anderen automatisierten Analysewerkzeuge, die über einen direkten Befehl oder ein Skript aufgerufen werden, können über die Python-Komponente angebunden werden, wie anhand der bereits integrierten Werkzeugen zu sehen ist. Die weitere Integration sowie alle dafür nötigen Schritte sind unabhängig von der Komponentenwahl äquivalent, da der Aufbau beider Analysekomponenten weitgehend ident gehalten wurde.

9.6.3 Integration eines Werkzeugs

Die nachfolgende Liste stellt eine Art Checkliste für den Integrationsprozess dar. In diesem Teilabschnitt wird darüber hinaus nicht mehr auf die einzelnen Schritte und deren Implementierung eingegangen. Für näheres Verständnis und konkrete Beispielimplementierungen wird an dieser Stelle direkt auf den Quellcode des Frameworks verwiesen.

a) Konfigurationserweiterung des Enum `AvailableCalls`

b) Anbindung und Aufruf des Werkzeugs mithilfe einer entsprechenden `Analysis-Worker` Implementierung

c) Report Verarbeitung durch Implementierung des `ReportParser`-Interface

d) Persistierung und Definition von Entitäten sowie DTOs

e) Anpassung der Sicherheitsevaluierung durch Implementierung des `ReportEvaluation-Service`-Interface

f) Erweiterung der Benutzeroberfläche

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10 Google Play Store-Anwendungsanalyse

Um die Hypothesen dieser Arbeit zu bestätigen, wurden innerhalb einer Evaluierungsphase im November und Dezember 2018, unter Verwendung des in dieser Arbeit entwickelten und in Kapitel 9 beschriebenen Analyse-Frameworks, 5.800 Google Play Store [127] Android-Anwendungen automatisiert analysiert, deren Sicherheit bestimmt und in weiterer Folge ausgewertet. Dazu wird in diesem Kapitel die Methodologie der Analyse beschrieben. Das Kapitel unterteilt sich in den zusammengefassten Werkzeugeinsatz zur Sicherheitseinstufung, die Beschreibung der Systemumgebung der Evaluierung, die automatisierten Auswahl der Anwendungen sowie die Durchführung der Sicherheitskategorisierung und Auswertung dieser.

10.1 Zusammenfassender Werkzeugeinsatz zur Evaluierung der definierten Sicherheitskategorien

Bereits in Kapitel 8.2 wurde genauer auf die Funktionsweisen und Analysebereiche der einzelnen Werkzeuge eingegangen. An dieser Stelle wird noch einmal zusammenfassend aufgezeigt, welche Werkzeuge das Framework für die Analyse der in Kapitel 7 definierten Sicherheitskategorien verwendet. Dies ermöglicht im Fall von einzelnen gescheiterten Werkzeuganalysen, das vereinfachte Erkennen von Auswirkung auf die allgemeine Sicherheitseinstufung der analysierten Anwendungen. Einige Werkzeuge, vor allem jene mit einem breiten Analysebereich wie AndroBugs, QARK oder SUPER, analysieren gleiche oder ähnliche Sicherheitsbereiche, wodurch das Scheitern eines einzigen Werkzeugs oftmals geringe Auswirkungen auf das gesamte Analyseergebnis darstellt.

AndroBugs [111]: Analysiert Intent Filter, WebView, Debug-Modus und Komponenten Fehlkonfigurationen sowie Probleme im Bereich von Berechtigungen, Kryptografie, Zertifikatvalidierung, Datenlecks, unsichere Kommunikation, Repacking und API-Versionen. AndroBugs untersucht daher Bereiche aller definierten Sicherheitskategorien.

QARK [124]: Fokussiert sich auf Probleme im `AndroidManifest.xml`, vor allem im Zusammenhang mit Komponenten, Berechtigungen und Intent Definitionen, ebenso wie auf API-Versionsprobleme, Zertifikatvalidierung und WebView Konfigurationen. QARK nimmt daher ebenso wie AndroBugs Einfluss auf die Analyseergebnisse aller definierten Sicherheitskategorien.

SUPER [125]: Untersucht Probleme der Komponenten Definitionen, Zertifikatvalidierung, Eingabevalidierung, Datenspeicherung beziehungsweise Datenlecks und die Verwendung unsicherer Verschlüsselungsalgorithmen. Ebenso wie die beiden zuvor genannten Werkzeuge ist der SUPER Analyse-Report für alle Sicherheitskategorien relevant.

Androwarn [114]: Analysiert das Auftreten von nativen Codeteilen und unsicheren Kommunikationsimplementierungen sowie Datenlecks. Das Ergebnis von Androwarn fließt daher in die Sicherheitskategorien mangelnder Angriffsschutz, Sicherheitsinvalidierung und Datenlecks mit ein.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

P-Lint [122]: Befasst sich ausschließlich mit Problemen der Berechtigungsdefinition und Prüfung. P-Lint nimmt Einfluss auf die Kategorie des mangelnden Angriffsschutzes.

Argus-SAF [30]: Untersucht unsichere Kryptografie Algorithmen und Zertifikatvalidierungs-Probleme. Dadurch nimmt Argus-SAF Einfluss auf die Sicherheitskategorie Sicherheitsinvalidierung.

APK-Parser [115]: Liest das `AndroidManifest.xml` und bereitet Informationen zu Komponenten Definitionen und Signaturen auf. APK-Parser unterstützt daher mit Informationen zur Signatur beim Erkennen von Repacking Problemen der Sicherheitskategorie mangelnder Angriffsschutz.

10.2 Beschreibung der verwendeten Systemumgebung

Um eine möglichst große Menge an Anwendungen zu analysieren, wurde eine Cloud-basierte, verteilte Installation des Analyse-Frameworks unter Verwendung von *Amazon Web Service (AWS)* [8] Ressourcen verwendet, um von typischen Vorteilen wie Elastizität, Skalierbarkeit, Verfügbarkeit, der Infrastruktur sowie Plattform und nicht zuletzt geringen Kosten zu profitieren. Welches konkrete Setup gewählt wurde, wird nachfolgend beschrieben.

10.2.1 Framework Anpassung für den AWS Einsatz

Um die Kommunikation zwischen den einzelnen Microservices innerhalb der AWS-Cloud zu ermöglichen, ist eine AWS-spezifische Konfiguration des Eureka Clients sowohl innerhalb des Main-, als auch des Analysis-Service nötig, um für die Registrierung des Services eine öffentlich erreichbare IP-Adresse zu verwenden. Dazu wird die IPv4-Adresse sowie der Hostname der entsprechenden AWS-Instanz aus den Umgebungsvariablen ausgelesen und der Eureka-Instanz Konfiguration übergeben. Auf diese Weise registriert sich die jeweilige Instanz mit der korrekten öffentlich zugänglichen Adresse, wodurch die Kommunikation zwischen den Instanzen ermöglicht wird. Standardmäßig würde sich die Instanz durch den Eureka Client mit der Adresse des Docker Containers registrieren.

10.2.2 AWS-Setup und Deploy

Um das Framework durch AWS zur Verfügung zu stellen, wurde von folgenden drei Services Gebrauch gemacht.

- *Amazon Elastic Container Registry (ECR) [5]*
- *Amazon Elastic Container Service (ECS) [6]*
- *Amazon Relational Database Service (RDS) [7]*

Als ersten Teil des Cloud-Setups zur Evaluierung wurde für jeden Microservice-Typ ein Repository in der ECR erstellt, bei der es sich um eine vollständige verwaltete Docker Container Registry handelt [5]. ECR ermöglicht auf diese Weise das Speichern, Verwalten und Bereitstellen von eigenen Docker Images, die anschließend auf beliebigen Instanzen gestartet werden können.

Der zweite Teil des AWS-Setups besteht aus zwei Clustern, die in Amazon ECS [6] erstellt wurden, um die Microservice-Instanzen zu verwalten. Ein Cluster enthält eine beliebige Anzahl an *Amazon Elastic Compute (EC2)* Nodes eines zuvor definierten Typs und kann dynamisch an die benötigten Ressourcen angepasst werden. Die Spezifikationen der für die Evaluierung relevanten

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

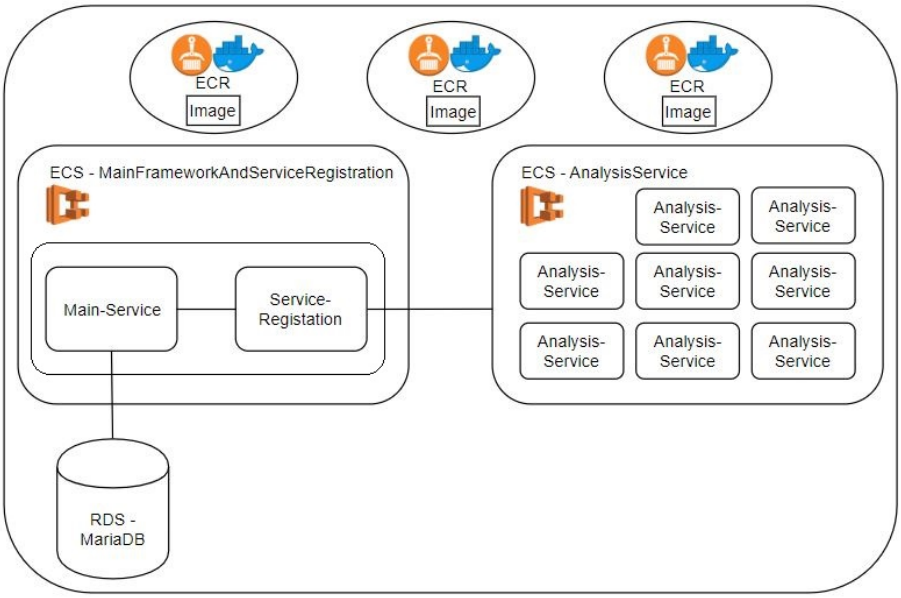


Abbildung 10.1: AWS-Setup der Evaluierungsumgebung

EC2 Instanz-Typen sind in Tabelle 10.1 anhand der AWS Angaben [8] abgebildet. Die *vCPU* beschreibt dabei die Anzahl der virtuellen CPUs, also Threads einer Instanz, während *EC2 Compute Unit (ECU)* die Anzahl der Kerne beschreibt. Variable ECU eignet sich besonders für Services mit variabler Auslast. Darüber hinaus beschreibt *RAM* den verfügbaren Arbeitsspeicher sowie *Speicher* die Größe und den Typ des vorhandenen Speichers. Dabei wird neben SSD Speicher ein *Amazon Elastic Block Store (EBS)* angeboten, dem ebenso SSD-Technologien zugrunde liegen.

Name	**vCPU**	**ECU**	**RAM**	**Speicher**
t2.small	1	Variable	2 GiB	Nur EBS
c5d.large	2	9	4 GiB	1 x 50 NVMe-SSD

Tabelle 10.1: Verwendete EC2-Instanz Typen

Da sowohl das zentrale Main-Service, als auch das Service-Registration Microservice relativ wenig Rechenkapazität benötigen, wurden diese innerhalb eines Clusters mit dem Namen `MainFrameworkAndServiceRegistration` zusammengefasst, der EC2-Instanzen des Typs t2.small verwaltet. Der zweite Cluster des Container Services trägt den Namen `AnalysisService` und verwaltet EC2-Instanz des Typs c5d.large, die für die Analysis-Service Instanzen bereitstehen.

Weiters beinhaltet ECS eine Reihe von Task Definitionen, die sich aus einer Beschreibung der zugrunde liegende Umgebung und des gewünschten ECR Docker Images zusammensetzt. Diese Tasks können nach Belieben innerhalb der definierten Cluster beziehungsweise der darin enthaltenen EC2-Instanzen gestartet werden.

Um keine Ressourcen zu verschwenden, wurde das Main-Service und Service-Registration Docker Image auf einer einzigen Amazon EC2 t2.small Instanz gestartet, während jede c5d.large Instanz des `AnalysisService` Clusters nur einen Analysis-Service Docker Container verwaltet. Abbildung 10.1 zeigt das finale Framework Setup der Evaluierung mit einer EC2 t2.small Instanz zur Serviceregistrierung und Datenverwaltung sowie acht c5d.large Analyse-Instanzen.

Der dritte und letzte Teil des AWS Setup befasst sich mit Amazon RDS [7]. Wie in Abbildung 10.1 zu sehen, wird Amazon RDS verwendet, um das Main-Service Microservice an eine zentrale

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

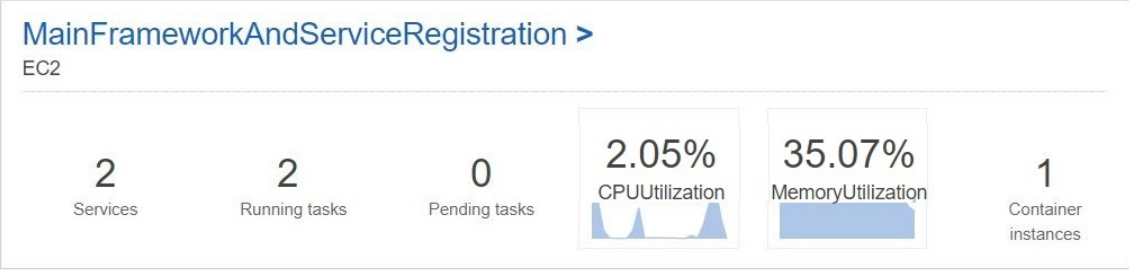


Abbildung 10.2: AWS ECS Main-Service und Service-Registration Cluster

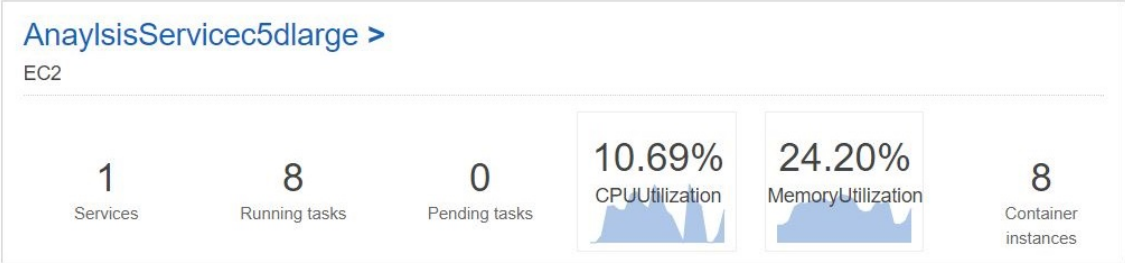


Abbildung 10.3: AWS ECS Analysis-Service Cluster

Datenbank anzubinden. Konkret kommt eine Standardkonfiguration für MariaDB von Amazon zum Einsatz, um die Daten zu verwalten.

10.3 Repräsentative Auswahl an Android-Anwendungen

Um das Framework innerhalb einer Evaluierung für eine möglichst breite Sicherheitseinstufung von Google Play Store-Anwendungen beziehungsweise der Sicherheitsevaluierung der unterschiedlichen Google Play Store-Anwendungskategorien einzusetzen und aussagekräftige Ergebnisse zu erhalten, ist es essenziell, automatisiert und unbehaftet eine Auswahl an Anwendungen zu treffen und diese nach und nach an die Analysis-Service Instanzen weiter zu reichen. Dazu wurde das Main-Service um einen automatischen Task erweitert, der in definierten Zeitintervallen die Auslast der einzelnen Instanzen prüft und diese gegebenenfalls mit neuen Anwendungen zur Analyse versorgt.

Um bei der Wahl der Android-Anwendungen möglichst aktuelle und weit verbreitete Anwendungen zu analysieren, wurde auf die von Google Play Store geführten Top kostenlosen Anwendungen (e.g. [128]) zurückgegriffen. Damit jeweils eine vergleichbare Anzahl an Anwendungen analysiert wird, werden jeweils die ersten 100 Top Anwendungen jeder der 58 existierenden und in Tabelle 4.1 aufgezählten Google Play Store-Kategorien ausgewählt. Dadurch beschränkt sich die Anzahl der zu analysierenden Anwendungen dieser Evaluierung auf 5.800 Google Play Store-Anwendungen.

10.4 Manuelle Validierung ausgewählter Sicherheitsprobleme

Um die Funktionsweise des Frameworks sowie der Werkzeug- und Report Komposition zu validieren, wurde vor der eigentlichen automatisierten Evaluierungsphase ein Probedurchlauf mit einer kleinen Versuchsgröße von rund 100 Anwendungen durchgeführt. Zur Validierung der angebundenen Werkzeuge und der Evaluierung der fünf in Kapitel 7 definierten Sicherheitskategorien, wurde jeweils ein konkretes entdecktes Problem eines Werkzeugs gewählt und das Ergebnis der automa-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

tisierten Analyse mittels manueller Untersuchung der betroffenen Android-Anwendung validiert. Im nachfolgenden Teil werden keine konkreten Namen der untersuchten Anwendungen genannt.

Im Bereich der Berechtigungsdefinition der Kategorie mangelnder Angriffsschutz wurde folgendes durch P-Lint erkanntes mögliches Sicherheitsrisiko manuell untersucht.

Custom permission is declared in manifest, could fail depending on its dependencies to dangerous permissions if they're not granted

Die erkannte Berechtigungsdefinition konnten im `AndroidManifest.xml` gefunden werden, wodurch die Funktionsweise von P-Lint sowie die Evaluierung in diesem Bereich untermauert werden konnte.

```
30 <permission android:name="com.spotify.music.permission.C2D_MESSAGE"
   ↪ android:protectionLevel="signature" />
```

Listing 10.1: Eigens erstellte und im `AndroidManifest.xml` definierte Berechtigung

Ebenso konnte die Funktionsweise von Argus-SAF der Kategorie Sicherheitsinvalidierung wie folgt bestätigt werden. Die durch Argus-SAF gemeldete Sicherheitslücke der Verwendung von fixierten IV im Bereich Kryptografie, wurde durch eine manuelle Untersuchung der betroffenen Anwendung anhand des in Listing 10.2 verkürzt dargestellten Quellcodeausschnitts belegt, der die Instanzierung eines fixierten IV durch `new IvParameterSpec('F1D3A567894BC2E0'.getBytes('UTF-8'));` zeigt.

```
31 private byte[] a(byte[] paramArrayOfByte)
32 {
33   ...
34   if (a())
35   {
36     e.init(1, f, g);
37     byte[] arrayOfByte = e.doFinal(paramArrayOfByte);
38     return arrayOfByte;
39   }
40   ...
41 }
42 private boolean a()
43 {
44   ...
45   f = new SecretKeySpec(new byte[] { -91, 48, -17, -86, 80, 82, 118,
     ↪ -105, -121, 43, 23, -17, -17, -12, -85, -11, 56, -126, -114,
     ↪ -96, -36, -122, 67, -71, 65, 64, 112, -69, 85, -40, 24, 112 },
     ↪ "AES");
46   g = new IvParameterSpec("F1D3A567894BC2E0".getBytes("UTF-8"));
47   e = Cipher.getInstance("AES/CBC/PKCS5Padding");
48   return true;
49 }
```

Listing 10.2: Verwendung eines fixierten IV

Im Bereich der Zugriffskontrolle entdeckte QARK die Erstellung eines PendingIntent durch die in Listing 10.3 dargestellte Funktion.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
50 public static PendingIntent a(Context paramContext)
51 {
52     return PendingIntent.getBroadcast(paramContext, 0, new Intent("
       ↪ com.appboy.action.receiver.APPBOY_GEOFENCE_UPDATE").setClass(
       ↪ paramContext, AppboyActionReceiver.class), 134217728);
53 }
```

Listing 10.3: Erstellung eines PendingIntent

Bei einer der untersuchten Anwendungen wurden durch Androwarn drei mögliche Auffälligkeiten im Bereich der Datenlecks gemeldet, die auf den folgenden Methodenzugriffen des `TelephonyManagers` basieren.

1. *getNetworkOperator() - This application reads the numeric name (MCC+MNC) of current registered operator*
2. *getNetworkType() - This application reads the radio technology (network type) currently in use on the device for data transmission*
3. *getPhoneType() - This application reads the device phone type value*

Diese Funde konnten durch manuelle Analyse der betroffenen Android-Anwendung durch den Codeteil 10.4 bestätigt werden, der den Zugriff auf die genannten Gerätedaten abbildet.

```
54 private final void zzl(Context paramContext)
55 {
56     TelephonyManager localTelephonyManager = (TelephonyManager)
       ↪ paramContext.getSystemService("phone");
57     ConnectivityManager localConnectivityManager = (
       ↪ ConnectivityManager)paramContext.getSystemService("
       ↪ connectivity");
58     zzVI = localTelephonyManager.getNetworkOperator();
59     zzVT = localTelephonyManager.getNetworkType();
60     zzVU = localTelephonyManager.getPhoneType();
61     ...
62 }
```

Listing 10.4: Zugriff auf sensible Daten des TelephonyManagers

Ebenso wurde im Bereich der Datenlecks eine Anwendung mit aktivierter Backup-Funktion durch QARK, SUPER und AndroBugs entdeckt, sowie im Bereich Zugriffskontrolle die aktivierte Android Debugging Funktion durch AndroBugs. Diese Funde konnten anhand der manuellen Auswertung des `AndroidManifest.xml` bestätigt werden, wie in Listing 10.5 anhand der Zeilen `android:allowBackup="true"` und `android:debuggable="true"` zu sehen ist.

```
63 <application
64     android:allowBackup="true"
65     android:debuggable="true"
66     android:icon="@mipmap/ic_launcher"
67     android:label="Cell Info"
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
68        android:name="at.cellinfo.AndroidCellInfo"
69        android:supportsRtl="true"
70        android:theme="@style/AppTheme">
71    ...
72    </application>
```

Listing 10.5: Datenleck und Android Debugging Funde durch QARK SUPER und AndroBugs

SUPER erkannte richtigerweise die WebView Javascript Aktivierung [61] einer Anwendung, die in Listing 10.6, [174] erkennbar ist und durch das Framework der Sicherheitskategorie Eingabevalidierung im Bereich XSS-Gefährdungen zugeordnet wurde.

```
73 WebView b(Context context) {
74     WebView webView = new WebView(context);
75     webView.getSettings().setJavaScriptEnabled(true);
76     return webView;
77 }
```

Listing 10.6: WebView Konfiguration mit aktivierter JavaScript Funktionalität

10.5 Automatisierte Analyse von Android-Anwendungen und Auswertung der Evaluierung

Zur Evaluierung der Android-Anwendungssicherheit wurde mithilfe des automatisierten Analysemodus eine Analyse von 5.800 Anwendungen durchgeführt. Diese 5.800 Anwendungen wurden durch das Framework über ein Zeitraum von 14 Tagen unter Verwendung der in Abschnitt 10.2 genannten Infrastruktur untersucht. In diesem Teil der Arbeit wird auf die Ergebnisse der Analyse beziehungsweise auf die statistische Auswertung der durchgeführten Sicherheitsevaluierung genauer eingegangen sowie Auffälligkeiten hervorgehoben und die Hypothesen dieser Arbeit bekräftigt.

Die Konfiguration der Risikoeinstufung der in Kapitel 7 definierten Kategorien in die vier Risikostufen `SECURE`, `WARNING`, `POTENTIALINSECURE`, `INSECURE` (siehe Abschnitt 9.3.6) erfolgte über das `appliaction-evaluation.yaml`. Die vollständige Konfiguration der Einstufung kann dem Kapitel Appendix A entnommen werden. Tabelle 10.2 stellt an dieser Stelle noch einmal zusammenfassend die Bedeutung der Risikostufen in Bezug auf die Anwendungssicherheit dar.

Risikostufe	**Bedeutung in Bezug auf die Anwendungssicherheit**
SECURE	Nicht relevanter Fund eines Werkzeugs.
WARNING	Codeteile, die vermieden oder deren Funktionalität und Relevanz überprüft werden sollte.
POTENTIALINSECURE	Mögliche Auswirkungen auf die Anwendungssicherheit. Weitere manuelle Analyse empfehlenswert.
INSECURE	Abweichung von State-of-the-Art Empfehlungen mit Auswirkungen auf die Anwendungssicherheit.

Tabelle 10.2: Übersicht der Risikostufen

Alle zur Analyse gewählten Anwendungen, wurden jeweils durch die eingebundenen Werkzeuge APK-Parser, ArgusSAF, AndroBugs, Androwarn, P-Lint, QARK und SUPER analysiert.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die statistische Auswertung des Evaluierungsergebnisses durch das Framework fokussiert sich auf die folgenden Bereiche.

1. Übersicht des Gesamtergebnisses der Sicherheitseinstufung
2. Anzahl der erfolgreichen sowie fehlerhaften Werkzeug Analysen
3. Benötigte Analysezeit der einzelnen Werkzeuge
4. Auflistung der am häufigsten erkannten Sicherheitsprobleme
5. Anzahl der Sicherheitsprobleme, die in den einzelnen Anwendungen festgestellt werden konnten.
6. Häufigkeit von Sicherheitsrisiken in den definierten Sicherheitskategorien
7. Sicherheitsrisiko ausgehend von den einzelnen Google Play Store-Anwendungskategorien

10.5.1 Übersicht des Gesamtergebnisses der Sicherheitseinstufung

Bevor detaillierte Ergebnisse der durchgeführten Sicherheitseinstufung von 5.800 Google Play Store Android-Anwendungen präsentiert werden, wird an dieser Stelle das Gesamtergebnis zusammengefasst und diskutiert. Abbildung 10.4 zeigt den entsprechenden Ausschnitt der statistischen Auswertung durch das Framework.

Von den 5.800 untersuchten Android-Anwendungen konnte bei 621 der Anwendungen kein Ergebnis erzielt werden. Dies bedeutet, dass bei 10.71% der untersuchten Anwendungen nach der Analyse durch die einzelnen Werkzeuge bei der Evaluierung der Reports keine Werkzeug-Reports vorlagen. Zum einen beinhaltet dies den Fall, dass alle Werkzeuge an der Analysedurchführung der entsprechenden Anwendung gescheitert sind, zum anderen den wesentlich wahrscheinlicheren Fall, dass die Dekompilierung vor der eigentlichen Analyse nicht erfolgreich durchgeführt werden konnte. In 568 (9.79%) der untersuchten Anwendungen konnte eine Sicherheitswarnung verzeichnet werden und in 217 (3.74%) eine mögliche Sicherheitslücke, die noch eine genauere manuelle Bewertung der erkannten Unregelmäßigkeit zur konkreten Bestimmung der Gefahr nach sich zieht. In 4394 (75.76%) und somit in den mit Abstand meisten Anwendungen konnte mindestens ein Sicherheitsrisiko in einem der untersuchten Sicherheitsbereiche entdeckt werden. Keine einzige Anwendung konnte am Ende der Sicherheitseinstufung als sicher und bedenkenlos beschrieben werden.

Bereits Arbeiten der vergangenen Jahre (e.g. [62], [96], [61], [129]), zeigen die Zustände im Bereich der Sicherheit von mobilen Anwendungen, wie auch Kapitel 2 verdeutlicht. So stellten beispielsweise Egele et al. [85] bereits 2012 in 88% von 11.748 untersuchten Anwendungen einen sicherheitskritischen Zustand im Bereich Kryptografie fest, Yu et al. [249] in 24% von 1.197 untersuchten Anwendungen im Bereich der Datenschutzrichtlinien und Rastogi et al. [194] Datenlecks in ebenso 24% von 3.968 untersuchten mobilen Anwendungen. Das Ergebnis der durchgeführten Analyse und die hohe Anzahl an unsicheren Android-Anwendungen ist demzufolge wenig überraschend. Überraschend hingegen ist, dass keine einzige Anwendung tatsächlich als unbedenklich eingestuft werden konnte. Jede untersuchte Anwendung verzeichnete zumindest eine Warnung, mögliche oder tatsächliche Sicherheitslücke beziehungsweise ein fehlendes Ergebnis in einen der fünf definierten Sicherheitsbereiche `mangelnder Angriffsschutz`, `Sicherheitsinvalidierung`, `anwendungsübergreifende Zugriffskontrolle`, `Datenlecks` und `Eingabevalidierung`. Dies verdeutlicht die Relevanz, bei einer allgemeinen Sicherheitseinstufung von Anwendungen, nicht nur bestimmte, sondern viele unterschiedliche sicherheitskritische Bereiche zu berücksichtigen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

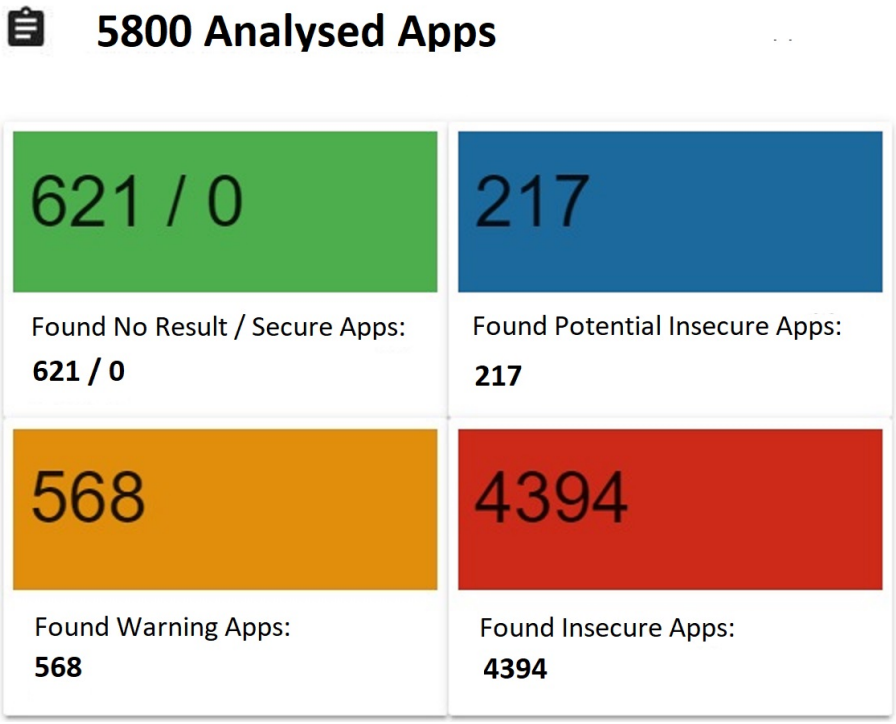


Abbildung 10.4: Ergebnis der Sicherheitseinstufung von 5.800 analysierter Google Play Store Android-Anwendungen

10.5.2 Anzahl der beendeten sowie abgebrochenen Werkzeug Analysen

Wie der Tabelle 10.3 entnommen werden kann, variiert die Zahl der beendete/abgebrochene Analysen der unterschiedlichen Werkzeuge stark. Die niedrigsten Anzahl an beendeten Analysen verzeichnet mit klarem Abstand Androwarn mit rund 11%. Von den 5187 gescheiterten Analyseversuchen wurden 77 durch den konfigurierten Timeout (30 Minuten) beendet und 5110 durch einen Fehler des Werkzeugs selbst. Ähnliche Zahlen an beendeten Analysen weisen die ähnlich funktionierenden Werkzeuge AndroBugs (49%), QARK (52%) und SUPER (66%) auf. Diese drei Werkzeuge basieren großteils auf statischen Pattern-Matching und Regel-basierten Analysemechanismen und wurden durch das entwickelte Framework bereits mit dekompilierten Quellcode versorgt, wodurch eine ähnliche Werte der Erwartungshaltung entsprechen. Argus-SAF, das sich vor allem auf Daten, Kontrollflüsse und Abhängigkeiten fokussiert und anhand dieser die Verwendung unsicherer Implementierungen erkennt, wies in unserer Evaluierung, trotzt wie in Kapitel 8.2 und 9.3.4 beschriebener Probleme im Bereich Dokumentation und Integration, überraschenderweise die höchste Zahl der beendeten Tests unter den Analysewerkzeugen mit vielseitigen untersuchten Sicherheitsbereichen auf, wodurch zukünftigen Arbeiten einen Fokus auf dieses Werkzeug nahegelegt wird. Auch die Ergebnisse von P-Lint (82%) und APK-Parser (91%) liegen der einfachen und ähnlichen Funktionsweise bedingt nahe beieinander und bestätigen somit die hohe Erwartungshaltung. In 532 der untersuchten Android-Anwendungen, konnte durch APK-Parser kein gültiges `AndroidManifest.xml` ausgelesen werden, wodurch auch das Scheitern von P-Lint, dessen Analyse auf den im `AndroidManifest.xml` definierten Berechtigungen basiert, in diesen Fällen nahe liegt. Darüber hinaus konnte P-Lint in weiteren 9% der Analysen kein erfolgreiches Analyseergebnis liefern.

Zusammenfassend lässt sich erkennen, dass Werkzeuge, die sich auf einzelne Sicherheitsbereiche beschränken, wesentlich mehr Analysen beenden konnten, als Werkzeuge, die sich auf mehrere unterschiedliche Bereiche fokussieren. Weiters weisen ähnlich arbeitende Werkzeuge auch ähnli-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

che Ergebnisse auf. Argus-SAF, das bei der durchgeführten Recherche äußerst komplex und undurchsichtig erschien und aufgrund dessen auch nur Teilfunktionen des Werkzeugs innerhalb des entwickelten Frameworks eingebunden wurden, weist eine deutlich erhöhte Anzahl an beendeten Analysen auf.

Werkzeug	Beendete Analysen	Abgebrochene Analysen	Beendete Analysen in Prozent
AndroBugs	2851	2949	49%
Androwarn	613	5187	11%
APK-Parser	5268	532	91%
Argus-SAF	5145	807	88%
P-Lint	4780	1020	82%
QARK	3033	2767	52%
SUPER	3800	2000	66%

Tabelle 10.3: Übersicht der beendete/abgebrochene Analysen der Werkzeuge

10.5.3 Benötigte Analysezeit der Werkzeuge

Um die Analysezeiten der unterschiedlichen Analyse-Werkzeuge über die gesamte Evaluierung hinweg zu vergleichen, wurden bei jeder untersuchten Anwendung Start- und Endzeitpunkt einer Werkzeuganalyse festgehalten. Durch die statistische Auswertung der aufgezeichneten Daten des Frameworks, wurden die in Tabelle 10.4 dargestellten Analysezeiten in Sekunden anhand der erfolgreich analysierten Anwendungen extrahiert und berechnet. Gescheiterte Analysevorgänge werden in dieser Berechnung vernachlässigt, um möglichst genaue Daten über eine tatsächliche beziehungsweise erfolgreiche Analyse zu sammeln. Dies dient auch der Erfahrungssammlung für die Konfiguration der Werkzeug-Timeouts für nachfolgende Analysen, Evaluierungen oder Weiterentwicklungen des Frameworks.

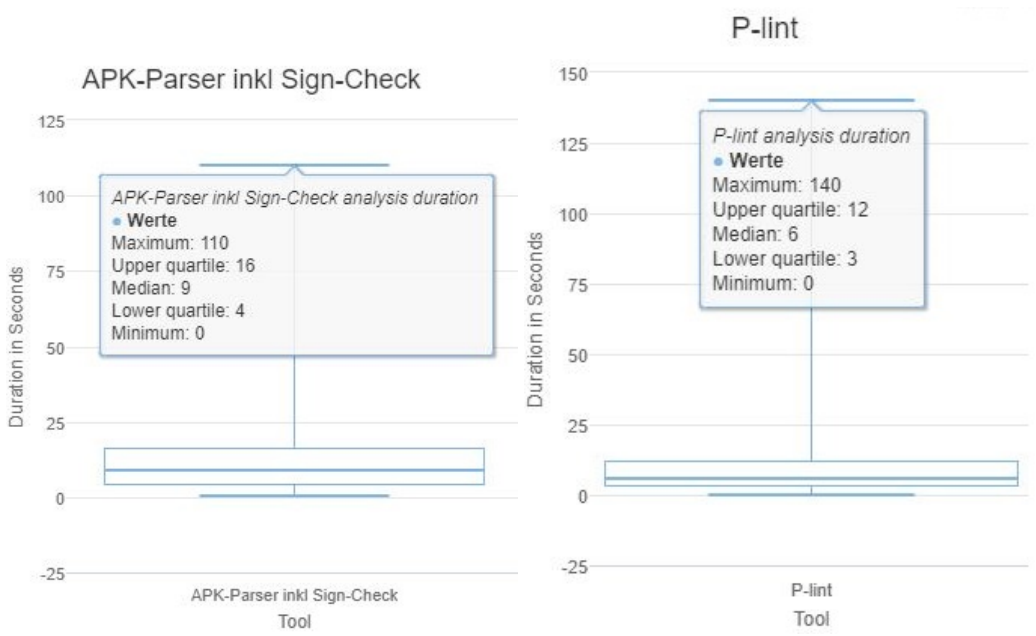
Werkzeug	Minimum	Unteres Quartil	Median	Oberes Quartil	Maximum
AndroBugs	2	74	121	181	650
Androwarn	1	27	80	158	732
APK-Parser	0	4	9	16	110
Argus-SAF	1	39	54	76	310
P-lint	0	3	6	12	140
QARK	18	121	335	521	1800
SUPER	3	51	67	80	450

Tabelle 10.4: Analysezeit pro Werkzeug in Sekunden (Timeout bei 1800 Sekunden)

Im Bereich der statistischen Auswertung des Frameworks, werden die Analysezeiten der Werkzeuge anhand der nachfolgend beschriebenen Boxplot-Diagrammen dargestellt. Auf diese Weise können möglichst genaue und vergleichbare Werte zur Verteilung und Streuung der Analysezeiten dargestellt werden.

Ähnlich wie bei den Erfolgsraten, weisen APK-Parser und P-Lint ähnliche und sehr kurze Ausführungszeiten auf, da beide Werkzeuge einfache Verarbeitungsvorgänge des `AndroidManifest` sowie weitere triviale Analysen durchführen. Die Ausführungszeit der beiden Werkzeuge beträgt unter Berücksichtigung des jeweils unteren und oberen Quartils üblicherweise zwischen drei und 16 Sekunden und wird grafisch in Abbildung 10.5 dargestellt. Damit benötigen die beiden Werkzeuge die niedrigste Ausführungszeit, der ins Framework angebundenen Werkzeuge.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub



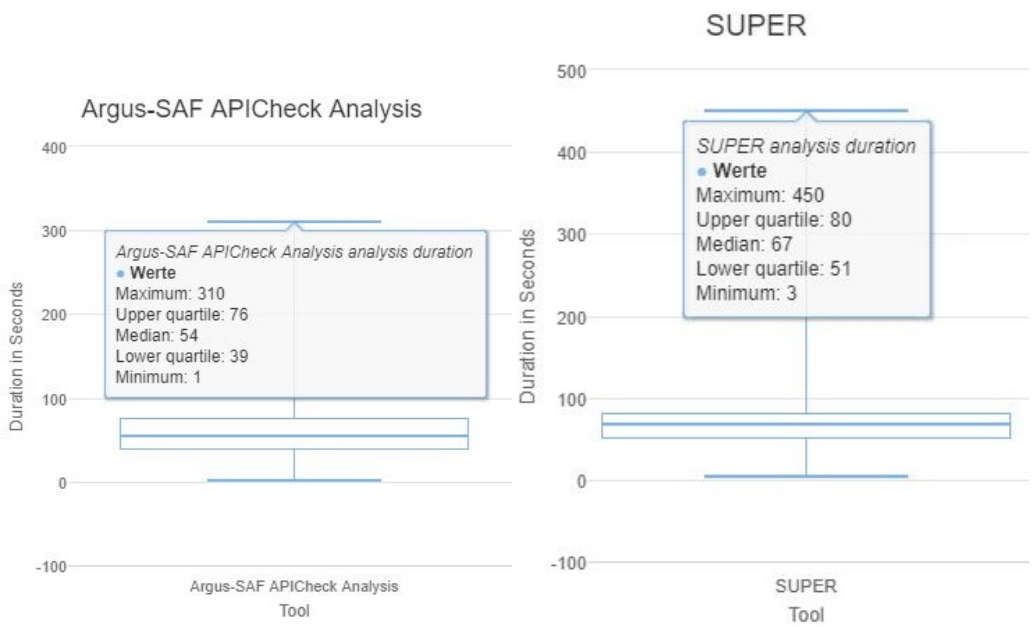
(a) Darstellung der APK-Parser Analysezeit in Sekunden

(b) Darstellung der P-Lint Analysezeit in Sekunden

Abbildung 10.5: Erfolgreiche Analysezeit der eingesetzten Werkzeuge in Sekunden

Die Ausführungszeiten einer erfolgreichen Analyse durch Argus-SAF und SUPER befinden sich üblicherweise im Bereich zwischen 39 und 80 Sekunden, durch Abbildung 10.6 grafisch dargestellt. In Einzelfällen sind, wie bei allen Analyse-Werkzeugen, grobe Abweichungen von diesen Werten möglich, die Streuung zwischen unteren und oberen Quartil ist hingegen gering. Betrachtet man die nachfolgenden Abbildungen 10.7 und 10.8 zeigt sich, dass das Werkzeug SUPER, das vom Funktionsumfang mit den Werkzeugen AndroBugs und QARK verglichen werden kann, eine deutlich geringere Ausführungszeit verzeichnet, was ein Indiz für die Effizienz der eingesetzten Programmiersprache Rust darstellt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

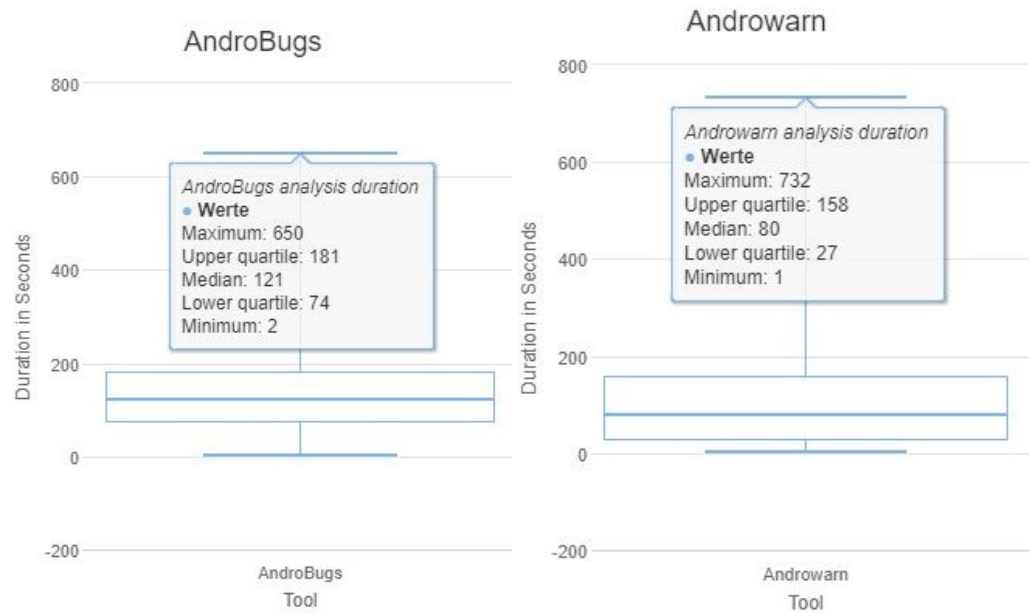


(a) Darstellung der Argus-SAF Analysezeit in Sekunden

(b) Darstellung der SUPER Analysezeit in Sekunden

Abbildung 10.6: Erfolgreiche Analysezeit der eingesetzten Werkzeuge in Sekunden

Verglichen zu SUPER weisen die Werkzeuge AndroBugs und Androwarn mit Ausführungszeiten zwischen 27 Sekunden im unteren Quartil und 181 Sekunden im oberen Quartil etwas höhere Analysezeiten auf. Die Streuung der Analysezeiten dieser beiden Werkzeuge ist ebenso wesentlich höher.



(a) Darstellung der AndroBugs Analysezeit in Sekunden

(b) Darstellung der Androwarn Analysezeit in Sekunden

Abbildung 10.7: Erfolgreiche Analysezeit der eingesetzten Werkzeuge in Sekunden

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

QARK verzeichnet die höchste Ausführungszeit, die selbst im Erfolgsfall in Einzelfällen die Timeout Konfigurationszeit von 1800 Sekunden erreicht. Die Ausführungszeit des unteren Quartil liegt bei rund 121 und des Oberen bei 521 Sekunden. Dadurch ist nicht nur die Analysedauer von QARK am höchsten, sondern auch die Streuung der erfolgreichen Ausführungszeiten, wie durch Abbildung 10.8 verdeutlicht wird.

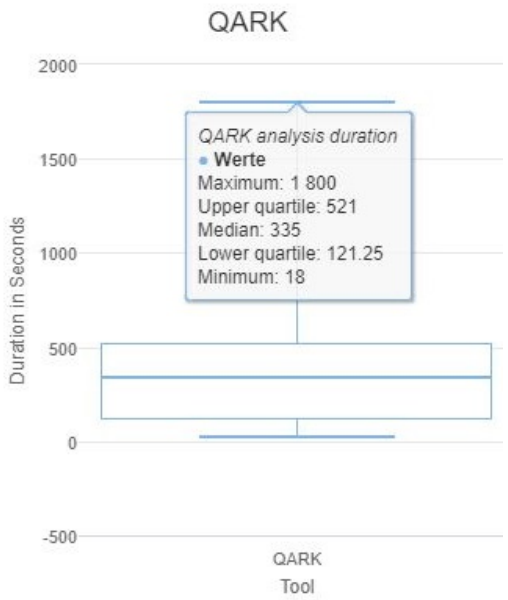


Abbildung 10.8: Darstellung der QARK Analysezeit in Sekunden

Im Gesamten betrachtet lässt sich erkennen, dass die Ausführungszeiten der einzelnen angepassten und integrierten Werkzeuge in ähnlicher Größenordnung ausfallen, je nach Werkzeug aber durchaus variieren. Um die Kosten einer Evaluierung zu reduzieren, kann die konfigurierbare Timeout Zeit von aktuell 1800 Sekunden wesentlich reduziert werden. SUPER verzeichnet im Verhältnis zu den Analysezeiten und der Breite der analysierten Sicherheitsbereiche das beste Verhältnis zwischen Aufwand und Mehrwert einer allgemein gehaltenen Sicherheitsanalyse.

10.5.4 Auflistung der am häufigsten erkannten Sicherheitsprobleme

Tabelle 10.5 bildet die zehn am häufigsten aufgetretenen Sicherheitsrisiken der Gefahrenstufen `POTENENTIALINSECURE` und `INSECURE` ab, auf die im folgenden Abschnitt eingegangen wird.

In 2.495 Android-Anwendungen wurden durch die von AndroBugs durchgeführte Prüfung *SSL Connection Checking* ungesicherte *HTTP*-Verbindungen entdeckt, die dringend durch *HTTPS*-Verbindungen ersetzt werden sollten, um dem Standard einer modernen Anwendung zu entsprechen und möglichen Risiken der ungesicherten Datenübertragung entgegenzuwirken [91], [240], [179].

Im WebView und XSS Bereich konnte ebenso durch AndroBugs in 2.129 Anwendungen ein potenzielles durch die WebView Methode `addJavascriptInterface` ausgehende Remote Code Execution Sicherheitsrisiko (*RCE Vulnerability Checking*) (CVE-2013-4710) [196], [164] entdeckt werden, die besonders bis Android-Version 4.1.2 kritischen Zugriff und Kontrolle des Hostgeräts ermöglicht.

Bei 1.492 Anwendungen konnte Code zur Überprüfung von `root`-Rechten (*Executing ROOT or System Privilege Checking*) entdeckt werden, was auf mögliche bösartige Hintergrundfunktionali-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

täten einer Anwendungen schließen lässt, von denen enorme Gefahren und Auswirkungen auf die Systemintegrität ausgehen können [250].

Unregelmäßigkeiten im Bereich SSL-Zertifikatvalidierung (*SSL Certificate Verification Checking*), wie das Erlauben von self-signed sowie abgelaufenen Zertifikaten oder Zertifikaten für falsche Domains, konnten bei 1.220 Anwendungen festgestellt werden, wodurch diese als unsicher eingestuft wurden.

Im sicherheitskritischen Bereich der Kryptografie fokussierte sich die Analyse auf die bereits 2012 untersuchten Regeln von Egele et al. [85]. Argus-SAF konnte bei 1.050 der untersuchten Google Play Store-Anwendungen die Verwendung des als unsicher geltenden ECB-Mode nachweisen (*Use ECB mode*), während QARK in 848 untersuchten Anwendungen den ECB-Mode entdeckte (*getInstance should not be calledwith ECB as the cipher mode, as it is insecure*). Die Abweichung von etwa 20% lässt sich durch die 36% höhere Erfolgsrate von Argus-SAF im Vergleich zu QARK erklären. Das zweite erkannte Sicherheitsproblem im Bereich der Kryptografie beschreibt die Verwendung von als unsicher geltenden IV (*Use non-random IV*) und wurde in 945 Anwendungen entdeckt.

Von Pending Intents geht eine besondere Gefahr aus [62], [131]. In 677 Anwendungen konnte durch QARK eine Pending Intent Instanz (*Empty Pending instance found*) entdeckt werden, wodurch diese Anwendungen als unsicher eingestuft wurden.

Das nächst häufigste Problemen (*Cert issued: bad hash*) befasst sich mit der Android Anwendungssignatur. Bei 631 der 5.800 untersuchten Anwendungen konnten SHA1 mit RSA Signaturen [148] entdeckt werden.

Das letzte der Top 10 häufigsten Problemen (*App Sandbox Permission Checking, Security issues "MODE_WORLD_READABLEör "MODE_WORLD_WRITEABLE"found*) konnte bei 593 Anwendungen festgestellt werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Sicherheitskategorie	Risiko Level	Beschreibung	Häufigkeit
Insecure_ Communication	INSECURE	SSL Connection Checking	2495
WebviewAndXSS	POTENTIALINSECURE	RCE Vulnerability Checking	2129
Data_Leakage	POTENTIALINSECURE	Executing ROOT or System Privilege Checking	1492
Certificate_Validation	INSECURE	SSL Certificate Verification Checking	1220
Crypto	INSECURE	Use ECB mode	1050
Crypto	INSECURE	Use non-random IV	945
Crypto	INSECURE	getInstance should not be called with ECB as the cipher mode, as it is insecure	848
Intent	INSECURE	Empty Pending instance found	677
Repacking	INSECURE	Cert issued: bad hash	631
Data_Storage	INSECURE	App Sandbox Permission Checking, Security issues MODE_WORLD_READABLE or MODE_WORLD_WRITEABLE found	593

Tabelle 10.5: Am häufigsten gefundene Sicherheitslücken mit Einstufung POTENTIALINSECURE und INSECURE

10.5.5 Anzahl der Sicherheitsrisiken, die in den einzelnen Anwendungen festgestellt werden konnten

Die nachstehende Grafik beschreibt die Anzahl der erkannten Warnungen, potenziellen Sicherheitsrisiken und erkannten Probleme in den untersuchten Anwendungen. Dabei werden mehrfach auftretende Probleme einer Anwendung nicht aggregiert. Tatsächlich zählt jede Meldung eines Werkzeugs mit entsprechender Risikoeinstufung als eigenständiges Problem. Werden beispielsweise durch ein Werkzeug bei einer Anwendung 20 exportierte Komponenten entdeckt, zählen diese auch als 20 eigenstehende Sicherheitsrisiken.

Anzahl der gefundenen Sicherheitsrisiken	Android-Anwendungen
0	633
1	309
2	91
3	41
4	46
5	60
6	70
7	61
8	84
9	88
10	95

Tabelle 10.6: Top 10 der geringsten Anzahl an Sicherheitsrisiken pro Anwendung

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Bei 633 Anwendungen konnte keines der eingesetzten Werkzeuge ein Analyseergebnis erlangen, daher wurden in diesen Anwendungen auch keine Probleme entdeckt, wie der Tabelle 10.6 entnommen werden kann. In 309 Anwendungen konnte jeweils eine Meldung der Werkzeuge verzeichnet werden, gefolgt von 91 Anwendungen die zwei Meldungen verzeichnen.

In den meisten Anwendungen, konkret in 132, wurden 12 Sicherheitsprobleme festgestellt, wie der grafischen Datenauswertung 10.9 entnommen werden kann. Die Anwendung mit den meisten Funden wies 609 Warnungen, potenzielle Sicherheitsrisiken und Probleme auf. Ein Großteil der 609 Sicherheitsmeldungen befasst sich mit exportierten Komponenten [62], [53], [146].

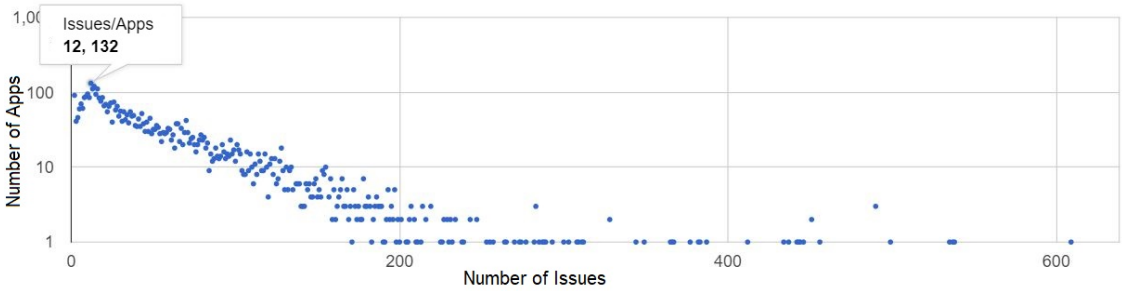


Abbildung 10.9: Darstellung der festgestellten Problemanzahl in den analysierten Anwendungen

10.5.6 Verteilung der Sicherheitsrisiken innerhalb der in Kapitel 7 definierten Sicherheitskategorien

Die folgenden zwei Tabellen 10.7 und 10.8 beschäftigen sich mit der Verteilung der Sicherheitsrisiken, die innerhalb der Evaluierung durch das entwickelte Framework entdeckt werden konnten. Im Bereich `Security Invalidation` wurden am meisten, konkret 3.337 sicherheitskritische Anwendungen entdeckt, gefolgt von `Sensitiv Data Leakage` mit 2.517 kritischen Anwendungen.

Kategorie	**Kein Ergebnis**	**Sicher**	**Warnung**	**Potenziell unsicher**	**Unsicher**
Insufficient Attack Protection	3382	70	1465	1	882
Security Invalidation	1999	401	0	63	3337
Access Control	1073	5	3905	69	748
Sensitive Data Leakage	2812	0	17	454	2517
Input Validation	2268	223	1169	2053	87

Tabelle 10.7: Sicherheitseinstufung der Anwendungen in den fünf definierten Sicherheitskategorien

Wie durch Tabelle 10.8 dargestellt, liegen die Risiken im Bereich `Security Invalidation` vor allem im Bereich `Crypto` (2.454 kritisch eingestufte Anwendungen) und `Certificat_Validation` (1.584). Im Bereich `Sensitive Data Leakage` befindet sich der größte Teil der Probleme im Bereich `Insecure_Communication` mit 2.446 kritisch eingestuften Anwendungen. Dies verdeutlicht zusätzlich zu der zuvor genannten Tabelle 10.5 der häufigsten auftretenden Sicherheitslücken, dass von absolut sicherheitskritischen Bereichen wie Kryptografie und Zertifikatvalidierung die häufigsten Sicherheitsrisiken und Lücken ausgehen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Kategorie	Unterkategorie	Kein Ergebnis	Sicher	Warnung	Potenziell unsicher	Unsicher
Insufficient Attack Protection	API_Version	2894	0	427	0	200
Insufficient Attack Protection	Native_Code	3794	298	1595	0	0
Insufficient Attack Protection	Repacking	3393	1668	0	0	626
Insufficient Attack Protection	Permission	3438	2096	41	3	109
Security Invalidation	Certificate_ Validation	3323	780	0	0	1584
Security Invalidation	Crypto	2482	631	0	120	2454
Security Invalidation	App_Distribution	526	5161	0	0	0
Access Control	ICC	1356	7	4247	77	0
Access Control	Intent	2655	351	1948	0	733
Access Control	Debug	2894	16	0	0	0
Sensitive Data Leakage	Insecure_ Communication	3032	207	0	2	2446
Sensitive Data Leakage	Data_Storage	2879	2	16	2207	583
Sensitive Data Leakage	Secret_Storage	4930	406	4	0	347
Sensitive Data Leakage	Data_Leakage	2913	132	724	1918	0
Input Validation	WebviewAndXSS	2222	220	1150	2010	85
Input Validation	SQL	1960	3723	0	4	0

Tabelle 10.8: Verteilung der Sicherheitsrisiken innerhalb der fünf definierten Sicherheitskategorien

10.5.7 Sicherheitsrisiko der unterschiedlichen Google Play Store-Anwendungskategorien

In den 58 existierenden Google Play Store-Kategorien wurden gleichmäßig verteilt jeweils 100 Anwendungen und somit insgesamt 5.800 Anwendungen analysiert. Das vollständige Ergebnis beziehungsweise die Verteilung der Sicherheitsrisiken auf die einzelnen Kategorien wird durch Tabelle A.3 dargestellt. Die nachfolgende Tabelle 10.9 zeigt jeweils die zehn Kategorien mit den meisten beziehungsweise wenigsten als kritisch eingestuften Anwendungen. Es zeigt sich, dass in kritischeren Bereichen wie `BUSINESS`, `FAMILY_EDUCATION`, `FINANCE` und `MEDICAL` die Anzahl der kritischen Anwendungen zwar geringer ist, als in Kategorien wie `BEAUTY`, `MUSIC_AND_AUDIO` oder `WEATHER`, dennoch aber ein alarmierendes Ergebnis liefern. Dies deckt sich auch mit den Ergebnissen von Taylor et al. [226], die ebenso feststellten, dass der Finanzbereich den eher sicheren Kategorien angehört. Die Kategorien `PERSONALIZATION` und `SOCIAL` zählen mitunter zu den gefährdetsten Kategorien.

Google Play Kategorie	Kein Ergebnis	Sicher	Warnung	Potenziell unsicher	Unsicher
BEAUTY	6	0	10	1	83

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

BUSINESS	18	0	8	7	67
FAMILY_BRAINGAMES	7	0	4	5	84
FAMILY_EDUCATION	16	0	12	7	65
FAMILY_MUSICVIDEO	7	0	5	4	84
FAMILY_PRETEND	16	0	12	3	69
FINANCE	14	0	18	3	65
GAME	9	0	16	7	68
GAME_ACTION	13	0	18	8	61
GAME_EDUCATIONAL	14	0	13	4	69
GAME_MUSIC	4	0	9	1	86
GAME_RACING	14	0	17	5	64
GAME_SIMULATION	18	0	11	6	65
GAME_SPORTS	4	0	8	5	83
MEDICAL	16	0	15	7	62
MUSIC_AND_AUDIO	12	0	3	0	85
PERSONALIZATION	7	0	7	1	85
SOCIAL	10	0	5	0	85
VIDEO_PLAYERS	8	0	5	1	86
WEATHER	8	0	1	3	88

Tabelle 10.9: Google Play Store-Kategorien mit den meisten beziehungsweise wenigsten als kritisch eingestuften Anwendungen

Fokussiert auf die Google Play Store-Kategorien mit den zehn wenigsten sicherheitskritischen Anwendungen (grün schattierte Säulen) sowie die zehn Kategorien mit den häufigsten sicherheitskritischen Anwendungen (rot schattiert) geht Abbildung 10.10 noch einmal auf die Verteilung der Risiken in den fünf Sicherheitskategorien ein. Wie in Abschnitt 10.5.6 ist auch hier zu erkennen, dass aus dem Bereich der `Security Invalidation` das höchste Risiko ausgeht, gefolgt von `Sensitive Data Leakage`. Eine Auswertung des `Security Invalidation` Bereichs 10.11 zeigt darüber hinaus abschließend das deutliche Risiko durch Kryptografie Algorithmen oder Konfigurationen dieser.

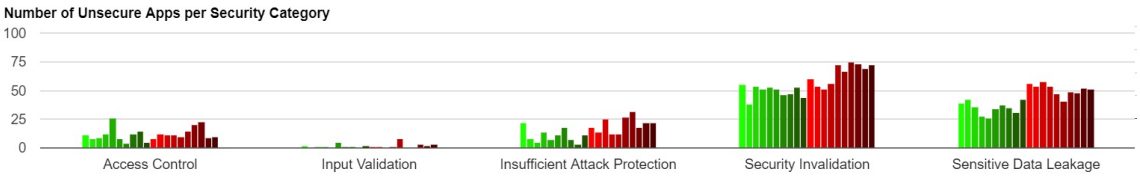


Abbildung 10.10: Verteilung der kritischen Anwendungen aus den Google Play Store-Kategorien mit niedrigsten (grün schattiert) und höchsten (rot schattiert) Sicherheitsrisiko in den fünf definierten Sicherheitskategorien

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Abbildung 10.11: Verteilung der kritischen Anwendungen aus den Google Play Store-Kategorien mit niedrigsten (grün schattiert) und höchsten (rot schattiert) Sicherheitsrisiko in der Security Invalidation Sicherheitskategorie

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

11 Fazit

In diesem Kapitel wird zusammenfassend noch einmal auf die wichtigsten der in Kapitel 2 vorgestellten bestehenden Arbeiten und Werkzeuge im Bereich Android-Sicherheitsanalyse eingegangen und Ähnlichkeiten, als auch Unterschiede zu dieser Arbeit aufzeigt. Im Anschluss werden die in dieser Arbeit definierten Hypothesen besprochen.

Das Ziel dieser Arbeit war es die Sicherheit von Android-Anwendungen anhand bestehender Analysewerkzeuge allgemein zu analysieren, ohne dabei den Fokus auf nur einen einzigen speziellen Sicherheitsteilbereich zu legen, wie es beispielsweise Egele et al. [85] mit dem Bereich Kryptografie, Chester et al. [59] sowie Colton et al. [66] mit Anwendungsberechtigungen und Yu et al. [249] mit Datenschutzrichtlinien machten. Eine klare Abtrennung dieser Arbeit gibt es auch zu der Arbeit von Bonett et al. [39], die sich ausschließlich mit der Evaluierung der Genauigkeit von Analysewerkzeugen befassten. Ebenso zu Huang et al. [137] die zeigten, dass Malware Entwickler und Entwicklerinnen Webservice wie VirusTotal missbrauchen, um unentdeckte schadhafte Software zu entwickeln. Viele der in Kapitel 2 genannten Arbeiten sowie in Kapitel 8 vorgestellten bestehenden Tools haben diese Arbeit und die Funktionalität des Frameworks inspiriert, wodurch auf den Erkenntnissen der vergangenen Jahre aufgebaut wurde. Cai et al. [42] zeigten beispielsweise wie die Analysezeit unter der Verwendung von Analyseergebnissen älterer Anwendungsversionen drastisch reduziert werden kann. Bei der Entwicklung des Frameworks stand daher auch die Reduktion der Analysezeit im Fokus, wodurch erkannt werden konnte, dass durch die Entfernung von mehrfach Dekompilierungen der unterschiedlichen Analysewerkzeuge Ausführungszeit gespart werden kann. Rojas et al. [201] entwickelten ebenso ein TaaS Framework, das allerdings durch die Nutzung von Cloud Ressourcen eine skalierende Lösung zum Testen der Anwendungskompatibilität mit unterschiedlichen Hardware und Betriebssystemversionen bereitstellt, die sowohl für Einzelprojekte, als auch für Großprojekte und Entwicklungsunternehmen interessant und hilfreich erscheint. Eine weitere besonders wichtige Arbeit ist die von Taylor et al. [226], die zeigte, dass selbst in Zeiten, in denen unzählige Analysewerkzeuge zur Verfügung stehen, trotzdem noch gravierende Sicherheitsprobleme in hoch kritischen Bereichen wie dem Finanzbereich bestehen. Wir sind ebenso wie Taylor et al. der dringenden Meinung, dass jedes Unternehmen und jedes Entwicklungsteam vor der Veröffentlichung einer Anwendung diese unbedingt auf aktuelle Sicherheitsstandards testen soll und muss. Es darf nicht vorkommen, dass Sicherheitslücken in Umlauf gebracht werden, die bereits seit Jahren bekannt sind.

Inspiriert von all diesen Arbeiten wurde in dieser Arbeit ein Framework entwickelt, das auf den Erkenntnissen und Werken der vergangenen Jahre basiert und auf die funktionalen sowie nichtfunktionalen und technischen Anforderungen der Zeit eingeht. Das entwickelte Framework ist durch eine moderne intelligente Microservice-Architektur, Cloud-fähig und skalierbar und wird als TaaS zur Verfügung gestellt, was einen wichtigen Schritt in Richtung der globalen Verbesserung von Anwendungen im Bereich der Sicherheit darstellt. Die Bereitstellung eines breiten Analyse-Frameworks als Service ist unserer Meinung nach besonders wichtig, denn wie Arbeiten aus der Vergangenheit zeigten, sind Sicherheitslücken in Android-Anwendungen trotz der Existenz vieler unterschiedlicher Analysewerkzeuge immer noch Standard. Wir vermuten daher, nicht zuletzt auch unserer Erfahrung mit diversen Analysewerkzeugen zufolge, dass die Installation und Verwendung solcher Tools nach wie vor eine enorme Einstiegshürde darstellen, die durch das Bereitstellen eines TaaS Frameworks umgangen werden kann. Das entwickelte Framework besitzt mehrere Informationsebenen beziehungsweise Detailgrade zur Einstufung von Android-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Anwendungen, wodurch es einen Mehrwert für Anwender und Anwenderinnen mit unterschiedlichen Wissensstand bietet. Die breit gefächerte Analyse durch mehrere integrierte Analysewerkzeuge bietet eine in fünf Kategorien aufgeteilte Sicherheitseinstufung, die kurz und prägnant diese Kategorien und in weiterer Folge die untersuchte Anwendung selbst, in drei Risikostufen einteilt. Dadurch ist für den Laien auf den ersten Blick erkennbar, wie sicher eine analysierte Anwendung ist. Hinter der Einstufung befinden sich allerdings detaillierte Berichte der einzelnen Analysewerkzeuge, wodurch je nach Anwendungsfall ein genauerer Einblick in die Analyseergebnisse möglich ist. Darüber hinaus ist die Sicherheitseinstufung der einzelnen Funde frei konfigurierbar, was gerade für Sicherheitsspezialisten oder Unternehmen, die eine lokale Installation des Frameworks verwenden, besonders interessant sein könnte. Durch diese Funktionalitäten sowie die leichte Erweiterbarkeit durch neue Analysewerkzeuge, die in Abschnitt 9.6 beschrieben wurde, unterscheidet sich das Framework nicht nur von sämtlichen alleinstehenden Analysewerkzeugen, sondern stellt auch eine Verbesserung zu Frameworks wie MobSF [170] dar, das zur statischen und dynamischen Analyse und vor allem zur Integration in Continuous Integration und Continuous Delivery Pipelines entwickelt wurde. Ähnlich wie die meisten Arbeiten im Bereich Android Sicherheit, aus denen ein Werkzeug oder ein Framework zur Analyse eines bestimmten Sicherheitsbereichs hervorgeht, wurde in dieser Arbeit ebenso einen Evaluierungsteil durchgeführt. Bei dieser Evaluierung wurde aufgezeigt, dass in allen Anwendungsbereichen sicherheitskritische Probleme angesiedelt sind. Darüber wurde, ähnlich wie durch Taylor et al. [226] gezeigt, dass trotz dem großen und immer weiter steigenden Angebot an Analysewerkzeugen nach wie vor kritische Implementierungen in älteren sowie modernen Android-Anwendungen enthalten sind.

An dieser Stelle wird der Focus auf die in dieser Arbeit definierten Hypothesen gelegt und diese anhand der im vorherigen Kapitel präsentierten Evaluierungsergebnisse bekräftigt.

Hypothese 1: Basierend auf den aufgedeckten Problemen kann eine Risiko- beziehungsweise Sicherheitseinstufung von Android-Anwendungen durchgeführt werden.

In Kapitel 7 wurden anhand der OWASP Mobile Top 10 und den Empfehlungen der CMU, die in Kapitel 6 beschrieben wurden, fünf sicherheitsrelevante Kategorien definiert. Um anhand dieser Bereiche die Sicherheit von Android-Anwendungen zu evaluieren, wurden in Kapitel 8 bestehende Werkzeuge untersucht und anhand des Sicherheitsbereichs und der Funktionsweise der Analyse für die Verwendung innerhalb des in dieser Arbeit entwickelten Frameworks gewählt. Durch die Kombination der unterschiedlichen Werkzeuge gelang es, alle definierten Sicherheitsbereiche zu analysieren. Weiters konnte, wie in Abschnitt 9.5.2 beschrieben, eine Funktionalität zur automatischen Auswertung und Zusammenfassung der einzelnen Werkzeug-Reports in ein einheitliches Format entwickelt werden. Die auf den durch die eingebundenen Werkzeuge gefundenen Probleme basierende Sicherheitseinstufung wurde darüber hinaus konfigurierbar umgesetzt, wodurch sich ein wesentlicher Mehrwert für zukünftige Anwendungsevaluierungen bietet.

Das mithilfe des in dieser Arbeit entwickelten Frameworks eine Sicherheitseinstufung von Android-Anwendungen durchgeführt werden kann, wurde neben der durchgeführten Evaluierung und Präsentation der Analyseergebnisse zur Sicherheitseinstufung unterschiedlicher Google Play Store-Anwendungen, auch durch eine stichprobenartige manuelle Untersuchung und Validierung der automatisierten Einstufung in Abschnitt 10.4 gezeigt.

Dadurch konnte die Hypothese, dass anhand der durch unterschiedliche Werkzeuge aufgedeckten Probleme eine Sicherheitseinstufung von Android-Anwendungen durchgeführt werden kann, belegt werden.

Hypothese 2: In allen Google Play Store-Anwendungskategorien sind sicherheitskritische Probleme angesiedelt, wobei es Kategorien gibt, in denen eine durchschnittliche Anwendung ein höheres Sicherheitsrisiko darstellt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die automatische Auswahl der Top Google Play Store-Anwendungen, die in Abschnitt 10.3 erklärt wurde, berücksichtigt die gleich verteilte Auswahl an Anwendungen über die 58 existierenden Google Play Store-Kategorien. Bei der durchgeführten Evaluierung wurden 100 Anwendungen aus allen Google Play Store-Kategorien untersucht und die Sicherheit der einzelnen Anwendungen ausgewertet.

Auf diese Weise konnte, wie in Abschnitt 10.5.7 zusammengefasst, festgestellt werden, dass in allen Anwendungskategorien sicherheitskritische Anwendungen vertreten sind. Dennoch konnte eine Abweichung zwischen den einzelnen Google Play Store-Kategorien in Bezug auf die durchschnittliche Anwendungssicherheit verifiziert werden. Die niedrigste Anzahl von als sicherheitskritisch eingestuften Anwendungen verzeichnet die Kategorie `GAME_ACTION` mit 61 kritischen von 100 untersuchten Anwendungen. Das Risiko eine kritische Anwendung aus diesem Bereich zu laden beträgt daher 61%. Das höchste Risiko geht von der Kategorie `WEATHER` mit 88% aus. Zwischen den Google Play Store-Kategorien ist daher bei jeweils 100 untersuchten Anwendungen eine Schwankung von 27 kritischen Anwendungen zwischen den Kategorien zu verzeichnen. Allgemein lässt sich in kritischeren Bereichen wie `BUSINESS (67%)`, `FAMILY_EDUCATION (65%)`, `FINANCE (65%)` und `MEDICAL (62%)` eine geringere Anzahl an kritisch eingestuften Anwendungen erkennen, als in Kategorien wie `BEAUTY (83%)`, `GAME_MUSIC (86%)`, `MUSIC_AND_AUDIO (85%)` oder `WEATHER (88%)`.

Die Kategorien `PERSONALIZATION (85%)` und `SOCIAL (85%)`, von denen ebenso wie von `BUSINESS`, `FAMILY_EDUCATION`, `FINANCE` und `MEDICAL` ein geringeres Risiko erwartet wird, zählen allerdings mitunter zu den gefährdetsten Kategorien. Die Hypothese, dass in allen Google Play Store-Kategorien kritische Anwendungen vertreten sind und die Anwendungssicherheit in unterschiedlichen Kategorien variiert konnte durch diese Evaluierung bestätigt werden. Sicherheitskritische Kategorien beinhalten dabei mit Ausnahme von `PERSONALIZATION` und `SOCIAL` eine geringere Anzahl kritisch eingestufter Anwendungen.

Hypothese 3: Wie Egele et al. [85] bereits von Mai bis Juli 2012 evaluierten, sind Sicherheitslücken in Android-Anwendungen durch den Einsatz des ECB-Mode und des CBC Verschlüsselungsalgorithmus weit verbreitet. Sechs Jahre später stellt sich die Frage nach dem aktuellen Stand dieser Evaluierung. Eine Neuevaluierung der Verbreitung des ECB-Mode und des CBC Verschlüsselungsalgorithmus mit non-random IV ermöglicht einen aktuellen Vergleich dieses sicherheitsrelevanten Bereichs mit den Ergebnissen von Egele et al.

Egele et al. [85] stellte im Jahre 2012 fest, dass Anwendungen die gegen eine der folgenden sechs Regeln aus dem Bereich der Kryptografie verstoßen, nicht als sicher eingestuft werden können.

Rule 1 (R1): Do not use ECB mode for encryption.

Rule 2 (R2): Do not use a non-random IV for CBC encryption.

Rule 3 (R3): Do not use constant encryption keys.

Rule 4 (R4): Do not use constant salts for PBE

Rule 5 (R5): Do not use fewer than 1,000 iterations for PBE.

Rule 6 (R6): Do not use static seeds to seed SecureRandom(x).

Anhand eines eigens entwickelten Werkzeugs namens CryptoLint wurden durch Egele et al. 11.748 Android-Anwendungen untersucht und festgestellt, dass 88% der untersuchten Anwendungen gegen mindestens eine dieser Regeln verstoßen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Mithilfe des in dieser Arbeit entwickelten Frameworks, beziehungsweise genauer durch die Anbindung der Werkzeuge Argus-SAF und QARK, konnten im Zuge der durchgeführten Evaluierung Verstöße von aktuellen State-of-the-Art Android-Anwendungen gegen die Regeln R1, R2, R6 erneut untersucht werden. Dabei wurden durch Argus-SAF von 5.145 erfolgreich untersuchten Anwendungen 1.050 Verstöße gegen die Regel R1, 945 gegen die Regel R2 und durch QARK von 3.033 erfolgreich analysierten Anwendungen 36 Verstöße gegen die Regel R6 vermerkt. Diese Ergebnisse wurden prozentuell in der Tabelle 11.1 mit den Ergebnissen von Egele et al. [85] gegenüber gestellt.

Regel	**Funde Egele et al.**	**Funde dieser Arbeit**
R1	65.17%	20.41%
R2	16.45%	18.37%
R6	13.87%	1.19%

Tabelle 11.1: Sicherheitsverstöße gegen die Regeln R1, R2 und R6 der Neuevaluierung im Vergleich zum Ergebnis von Egele et al. [85]

Durch die im November und Dezember 2018 durchgeführte Neuevaluierung der Regeln R1, R2, und R6 - bei der neue und sich von der durch Egele et al. durchgeführten Analyse unterscheidende Werkzeuge zum Einsatz kamen - konnte eine Verbesserung der Anwendungssicherheit im Bereich der Regeln R1 und R6 sowie eine leichte Verbesserung im Bereich der Regel R6 festgestellt werden. Insgesamt wurden im Zuge der Neuevaluierung bei 2.498 Android-Anwendungen Verstöße gegen mindestens eine der drei untersuchten Regeln im Bereich Kryptografie festgestellt. Dies entspricht einer Häufigkeit von 43.01% beziehungsweise etwa 48.55%, wenn nur erfolgreich analysierte Anwendungen berücksichtigt werden.

Hypothese 4: Das entwickelte Framework zur Kombination bestehender Analysewerkzeuge und automatisierten Analyse sowie Sicherheitseinstufung von Android-Anwendungen ist durch neue Werkzeuge entsprechend erweiterbar, wodurch sich sowohl der Benutzeroberfläche, als auch der skalierenden verteilten Analyseknoten bedient werden kann.

Durch den Einsatz moderner State-of-the-Art-Entwicklungsmuster und klarer Schnittstellendefinitionen wurde das Framework hinsichtlich der Erweiterung um weitere Analysewerkzeuge entwickelt. Speziell durch die Verwendung von Factory, Chain of Responsibility und Facade Pattern wurde für einer Erweiterbarkeit vorgesorgt. Anhand des einheitlichen Konzepts der Werkzeuganbindung sowohl innerhalb der Java-, als auch der Python-Analysekomponente, wurden innerhalb der Framework Implementierung, wie in Abschnitt 9.3.4 beschrieben, acht unterschiedliche Analysewerkzeuge angebunden, erfolgreich in eine einheitliche Reporterstellung aufgenommen und konnten in weiterer Folge zu einer einheitlichen Sicherheitseinstufung von Android-Anwendungen beitragen. Jede einzelne Integration, der acht Analysewerkzeuge, stellt daher eine Werkzeugerweiterung dar und kann beispielhaft für die Erweiterbarkeit des Frameworks herangezogen werden. Darüber hinaus wurde innerhalb der Arbeit in Abschnitt 9.6 eine tief greifende Beschreibung zur Integration neuer Werkzeuge erstellt, die nicht nur die konkrete Erweiterung zukünftiger Entwicklungen erleichtert, sondern auch eine frühzeitige Erkennung der Kompatibilität zwischen dem zu integrierenden Werkzeug und der Framework Architektur ermöglicht. Zusammenfassend lässt sich anhand der in Kapitel 9 beschriebenen Architektur, der Framework Funktionalität und der existierenden Beschreibung zur Integration von Analysewerkzeugen sowie anhand der durchgeführten Integration von acht Werkzeugen, die gegebene Erweiterbarkeit des Frameworks belegen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

12 Zusammenfassung und Ausblick

Erst die mobilen Anwendungen machen smarte Endgeräte zu den täglichen nützlichen Begleitern vieler Menschen, wodurch diesen Anwendungen Zugriff auf sensible Daten der unterschiedlichsten Lebensbereiche gewährt wird. Android entwickelte über die verschiedenen Versionen hinweg immer wieder neue Sicherheitsmaßnahmen (e.g. [90], [241], [18]), um sensible Daten zu schützen und die Entwicklung als sicher geltender Anwendungen zu ermöglichen. Einige wichtige Sicherheitskonzepte von Android sowie die Architektur selbst wurden in Kapitel 5 näher beschrieben. Die korrekte Verwendung, Konfiguration und der bewusste Umgang mit Daten liegt allerdings in der Verantwortung der Anwendungsentwickler und Entwicklerinnen sowie der Endnutzer und Nutzerinnen. Anwendungssicherheit im Bereich Android stellte bereits in der Vergangenheit ein wichtiges Thema dar, wodurch wissenschaftliche Arbeiten wie etwa von Egele et al. [85] oder Regeln zur sicheren Anwendungsentwicklung durch die CMU [22], ebenso wie OWASP mobile Top 10 [179] veröffentlicht wurden, die alle zusammen State-of-the-Art Empfehlungen und Standards für als sicher geltende Android-Anwendungsentwicklung beschreiben. Trotzt dieser existierenden Regeln und Empfehlungen, zeigten vergangene wissenschaftliche Forschungen im Bereich Android-Anwendungssicherheit erhebliche Mängel in den unterschiedlichsten sicherheitskritischen Bereichen von Android-Anwendungen (e.g. [62], [96], [61], [129], [42], [237], [137], [85], [194], [226]). Durch diese Arbeiten entstanden unter anderem auch unzählige Werkzeuge zur Sicherheitsanalyse.

In dieser Arbeit wurde die Analyse der IT-Sicherheit von aktuellen Android-Anwendungen in den Fokus gestellt, um festzustellen, ob eine automatisierte Sicherheitseinstufung basierend auf solchen existierenden State-of-the-Art-Werkzeugen möglich ist, in allen Google Play Store-Anwendungskategorien sicherheitskritische Gefährdungen vertreten und langjährig bekannte Sicherheitslücken im Bereich der Kryptografie, die durch Egele et al. [85] untersucht wurden, noch immer präsent sind. Zu diesem Zweck wurde ein skalierendes Analyse-Framework entwickelt, dass die in Kapitel 7 definierten Sicherheitsbereiche analysiert und in weiterer Folge die Sicherheit von Android-Anwendungen auswertet.

Im Zuge der durchgeführten Recherche, um potenzielle Werkzeuge zur Integration in das entwickelte Analyse-Framework zu finden, musste zum Teil eine hohe Komplexität sowohl im Setup, als auch in der Verwendung vieler Anwendungen festgestellt werden. Auch der genaue Funktionsumfang mancher Werkzeuge ist ohne eine genauere Quellcode-Untersuchung kaum nachvollziehbar. Wodurch den gesammelten Erfahrungen zufolge eine hohe Einstiegshürde in die Sicherheitsanalyse von Android-Anwendungen durch viele existierende Werkzeuge gegeben ist. Auch Bonett et al. [39] zeigen, dass Analysewerkzeuge, die auch alleinstehend mehrfach in wissenschaftlichen Arbeiten zum Einsatz kamen, undokumentierte Grenzbereiche, Design-Entscheidungen oder Fehlimplementierungen beinhalten, wodurch auch die daraus resultierenden Ergebnisse Ungenauigkeiten und möglicherweise verzerrte Tatsachen darstellen. Diese und immer wieder neu veröffentlichte Arbeiten, Werkzeuge und Frameworks im wissenschaftlichen Bereich der Android-Sicherheitsanalyse, geben darüber hinaus zum einen den Einblick in die dringende Relevanz dieses Themas und zum anderen einen Ausblick auf das existierende Angebot an vorwiegend durch wissenschaftliche Arbeiten entstandenen Werkzeuge, die sich teilweise nicht über einen prototypischen Entwicklungsstatus hinaus bewegt haben und dadurch verständlicherweise Abgrenzungen, Grenzbereiche sowie problematische Design-Entscheidungen bis hin zu Fehlimplementierungen beinhalten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Das in dieser Arbeit entwickelte Framework zur Risiko- beziehungsweise Sicherheitseinstufung von Android-Anwendungen, stellt eine skalierbare Microservice-basierte State-of-the-Art Lösung dar, um verschiedene Werkzeuge an ein globales Framework anzubinden und diese über eine TaaS Plattform zur Verfügung zu stellen. Durch die Integration unterschiedlicher existierender Werkzeuge wird durch das Framework nicht nur ein breiter Bereich an sicherheitsrelevanten Bereichen untersucht, sondern erhöht sich auch die Erfolgsrate der Analyse. Da in vielen Bereichen mehrere Werkzeuge ähnliche und gleiche Bereiche abdecken, halten sich die Auswirkungen von Ausfällen einzelner Werkzeuge auf das Gesamtergebnis in Grenzen. Welche Erfahrungen im Bereich der Werkzeugrecherche, Werkzeuganpassung und Frameworkintegration gesammelt werden konnten, kann den Kapiteln 8.2 beziehungsweise 9.3.4 entnommen werden, welche eine Basis für zukünftige Arbeiten in diesem Bereich darstellen sollen.

Bei der Entwicklung des Frameworks wurde speziell sowohl im Bereich der Werkzeugintegration, als auch im Bereich der Report Evaluierung auf Erweiterbarkeit und Konfigurierbarkeit geachtet, um zum einen Wiederverwendbarkeit, Wartbarkeit und Erweiterbarkeit zu gewährleisten und zum anderen, um eine möglichst breite Nutzergruppe anzusprechen. So ist es beispielsweise durch Konfigurationsanpassung möglich, die Sicherheitseinstufung bestimmter, durch die eingebundenen Werkzeuge entdeckten Probleme, beliebig anzupassen. Dies stellt besonders für Unternehmen, aber auch Wissenschaftsexperten und Expertinnen die Möglichkeit dar, den Fokus auf bestimmte Sicherheitsbereiche zu legen und die Einstufung anzupassen. Die Zeit und kostenintensive Dekompilierung, die sich speziell bei Android-Anwendungen durch bekannte in Kapitel 3 beschriebenen Herausforderungen komplex darstellt, sowie die Analyse der Anwendungen durch die einzelnen Werkzeuge muss dabei nur initial durchgeführt werden. Im Anschluss kann die Sicherheitseinstufung mit beliebigen veränderten Risikoeinstufungen aktualisiert und erneuert werden.

Um die Sicherheit von Android-Anwendungen aus dem Google Play Store zu analysieren, Forschungsfragen zu beantworten und gleichzeitig die Funktionsweise des entwickelten Frameworks zur automatisierten Erst-Analyse von Android-Anwendungen in Bezug auf IT-Sicherheit und Datenschutz zu validieren, wurde unter Nutzung von AWS Ressourcen über 14 Tage eine Evaluierung von jeweils 100 Anwendungen aus allen 58 existierenden Google Play Store-Anwendungskategorien durchgeführt.

Dabei wurden die einzelnen Anwendungen jeweils von den durch das Framework angebundenen Werkzeugen AndroBugs [111], Androwarn [114], APK-Parser [115], Argus-SAF [30], P-Lint [122], QARK [124] und SUPER [125] untersucht. Das Analyseergebnis der Werkzeuge wurde im Anschluss durch das Framework weiter verarbeitet und somit die Sicherheit der Sicherheitskategorien `mangelnder Angriffsschutz`, `Sicherheitsinvalidierung`, `anwendungsübergreifende Zugriffskontrolle`, `Datenlecks` und `Eingabevalidierung` unter Verwendung der Sicherheitsstufen `kein Ergebnis`, `Sicher`, `Warnung`, `Potenziell unsicher` und `Unsicher` eingestuft.

Auf diese Weise konnten durch das entwickelte Framework sowohl die erste Hypothese der Arbeit, dass basierend auf den durch existierende Werkzeuge aufgedeckten Probleme eine Sicherheitseinstufung von Android-Anwendungen möglich ist, als auch die vierte Hypothese, die sich auf die Erweiterbarkeit des Frameworks bezüglich der Integration von existierenden Werkzeugen fokussiert, belegt werden.

Bei der Auswertung des Evaluierungsergebnisses wurde neben dem Gesamtergebnis der Einstufung ein besonderer Fokus auf die Erfolgsrate der einzelnen Werkzeuge, die benötigte Analysezeit, die häufigsten erkannten Sicherheitsprobleme, die Anzahl der Probleme pro Anwendung sowie die Verteilung dieser in den einzelnen Sicherheitskategorien und die von den unterschiedlichen Google Play Store-Anwendungskategorien ausgehende Gefahr gelegt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Zusammenfassend lässt sich erkennen, dass die Erfolgsrate der Werkzeuge variiert, wobei Androwarn mit 11% eine deutlich geringere Erfolgsrate verzeichnet, als die anderen eingesetzten Werkzeuge. Mit der höchsten Erfolgsrate zeichnen sich APK-Parser (91%) und Argus-SAF (88%) aus.

Wie erwartet variiert auch die Analysezeit der unterschiedlichen Werkzeuge durch die verschiedenen Funktionsweisen und Analysebereiche der Werkzeuge. Werkzeuge, die ähnliche Funktionsweisen sowie Analysebereiche aufweisen, besitzen dabei vergleichbare Analysezeiten. Das obere Quartil der erfolgreichen Analysezeiten aller Werkzeuge befindet sich, durch die Optimierung der Werkzeuge beziehungsweise die Entfernung der Dekompilierungsschritte, im Bereich von 12 bis 521 Sekunden.

Durch die Kombination der unterschiedlichen Analyseergebnisse konnte im Zuge der Evaluierung ein breiter Sicherheitsbereich der Android-Anwendungen untersucht werden. Die Evaluierung zeigt, dass jede der jeweils 100 Android-Anwendungen aus den unterschiedlichen Google Play Store-Kategorien mindestens einen Hinweis auf ein Sicherheitsproblem aufweist und somit nicht als bedenkenlos beziehungsweise sicher eingestuft werden kann. Bei rund 11% der 5.800 untersuchten Anwendungen konnte kein Werkzeug eine erfolgreiche Analyse abschließen, wodurch in diesen Fällen keine Sicherheitseinstufung durchgeführt werden konnte. Rund 10% verzeichneten zumindest eine Sicherheitswarnung, etwa 4% ein potenzielles Sicherheitsproblem und gerundet 76% ein kritisches Problem.

Am häufigsten konnten kritische Anwendungen im Bereich unsichere Kommunikation, Kryptografie, WebView, und Zertifikatvalidierung manifestiert werden. Die Anzahl der als kritisch eingestuften Anwendungen variiert je nach Google Play Store-Kategorie zwischen 61 und 88 Anwendungen. Eine Tendenz, dass in sicherheitsrelevanten Kategorien wie `BUSINESS`, `FAMILY_EDUCATION`, `FINANCE` und `MEDICAL` weniger sicherheitskritische Anwendungen vorgefunden werden, lässt sich feststellen. Das Ergebnis ist aber dennoch deutlich höher als erwartet. Durch diese Auswertung konnte die zweite Hypothese 2 belegt werden, die sich mit der Verteilung der sicherheitskritischen Probleme in den unterschiedlichen Google Play Store-Anwendungskategorien befasst.

Der enorm kritische Bereich der Kryptografie reiht sich bedenklicherweise unter die am häufigsten auftretenden Sicherheitsprobleme und bestätigt auch sechs Jahre nach der von Egele et al. durchgeführten Evaluierung [85] in diesem Bereich, nach wie vor dramatische Zahlen und beantwortet die in der dritten Hypothese gestellte Frage nach der Veränderung der Sicherheit im Bereich der Kryptografie in den vergangenen sechs Jahren. Während die Verwendung des ECB-Mode von rund 65% 2012 auf rund 20% 2018 und ebenso die Verwendung von statischen Seeds von etwa 14% auf 1% gesunken ist, steigt die Zahl der unsicheren IVen bei CBC-Mode leicht von zirka 16% auf etwa 18%. Zumindest die drei in dieser Arbeit neu evaluierten, der insgesamt sechs von Egele et al. 2012 als kritisch definierten Sicherheitsregeln, sind demnach nach wie vor äußerst aktuell und immer noch in einer Vielzahl an Anwendungen vertreten. Konkret konnte in 43% von 5.800 untersuchten Anwendungen mindestens ein Verstoß unter Berücksichtigung der drei Sicherheitsregeln verzeichnet werden. Dies zeigt einmal mehr die absolute Relevanz von regelmäßigen Sicherheitstests im Bereich der Android Entwicklung.

Allgemein hinterlässt das Ergebnis der Evaluierung den Eindruck des mangelnden Sicherheitsbewusstseins vieler Anwendungsentwickler und Entwicklerinnen. Die durch diese Arbeit bewiesene Tatsache, dass viele Sicherheitsanalyse-Werkzeuge zudem oftmals eine hohe Einstiegshürde in dem Bereich der Sicherheitsanalyse darstellen, verbessert weiterhin kaum die Lage, wenngleich vergleichbare Ergebnisse zumindest im Bereich der Kryptografie eine Verbesserung der Sicherheit in den vergangenen sechs Jahren verzeichnen. Das Angebot an einzelnen Analysewerkzeugen, die sich auf bestimmte Bereiche spezialisieren, ist zwar durchaus groß, jedoch scheint das Angebot

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

an webbasierten Framework Lösungen, die wie das in dieser Arbeit entwickelte Framework einen breiten Sicherheitsbereich abdeckt, überaus gering.

Um zukünftig die Sicherheit und dadurch bedingte Qualität von Android-Anwendungen zu erhöhen, würde sich eine Integration von Frameworks wie diesem für die Integration in Build Automation Pipelines beziehungsweise CI/CD Pipelines anbieten. Dies stellt beispielsweise neben der Integration weiterer Werkzeuge eine mögliche zukünftige Weiterentwicklung des entwickelten Frameworks dar. Gerade die frei konfigurierbare Evaluierung ermöglicht Entwicklern und Entwicklerinnen sowie Unternehmen eine Anpassung der Sicherheitsstufen in den unterschiedlichsten Bereichen. Bei der direkten Integration in eine derartige Pipeline würde zudem, durch den bereits vorliegenden Quellcode, wesentlicher Aufwand der Dekompilierung entfallen und zugleich die Genauigkeit der Analyse steigen. Des Weiteren würden Probleme direkt bei der Entwicklung aufgedeckt werden, wodurch das Problem der mangelnden Sicherheit von mobilen Anwendungen bereits beim Ursprung behandelt werden könnte.

Zudem könnte in zukünftigen Arbeiten Anpassungen am Framework oder einzelnen Werkzeugen, angelehnt an die Arbeit [42], durchgeführt werden, um Analysezeiten durch gezielte Analysen veränderter Quellcode Bereiche zu optimieren. Cai et al. fokussierten sich dabei auf die historische Analyse von Anwendungsversionen und konnten auf diese Weise 78% an Analyseaufwand einsparen.

Ebenso würde sich das entwickelte Framework dazu eignen, innerhalb einer weiterführenden Arbeit beziehungsweise Weiterentwicklung, die Funktionsweise ähnlicher Werkzeuge, deren Analysebereiche sich überschneiden, zu vergleichen und auf diese Weise die Funktionsweise der Werkzeuge zu validieren.

Am Framework selbst kann neben weiteren Werkzeugintegrationen noch das Konzept der Analyseauslast verbessert werden. Aktuell wird jeweils von einer Analyse-Instanz nur eine Anwendung zur gleichen Zeit verarbeitet. Dies bedeutet, dass Werkzeuge, deren Ausführungszeit unter Umständen wesentlich geringer sind als andere, entsprechenden Leerlauf aufweisen. Für Evaluierungen wie die innerhalb dieser Arbeit durchgeführte, verändert sich dadurch die gesamte Ausführungszeit aber nur vernachlässigbar, da letzten Endes ohnehin auf das gesamte Analyseergebnis aller Werkzeuge gewartet werden muss. Die Timeout Zeit, die aktuell mit 1.800 Sekunden definiert ist, kann allerdings den in Abschnitt 10.5.3 präsentierten Ausführungszeiten angepasst werden, um frühzeitig scheiternde Analysen abzubrechen und Kosten zu sparen.

Abschließend lässt sich zusammenfassen, dass ein Großteil der analysierten Anwendungen zumindest einen kritischen Verstoß gegen eine Sicherheitsrichtlinie beinhaltet. Des Weiteren ist das Angebot an Werkzeugen zur Sicherheitsanalyse von mobilen Anwendungen zwar groß, jedoch benötigen diese meist ein komplexes Setup, verfügen über kaum Dokumentation und stellen eine hohe Einstiegshürde in den Bereich der Sicherheitsanalyse dar. Neuevaluierungen und Vergleiche von Ergebnissen einer Evaluierung des Sicherheitsbereichs der Kryptografie aus dem Jahre 2012 zeigen, dass nach wie vor langjährig bekannte Sicherheitslücken in modernen Anwendungen präsent sind und somit die absolute Relevanz besteht, Wissen und möglichst effiziente Analysewerkzeuge in die Unternehmen sowie an die Entwickler und Entwicklerinnen mobiler Anwendungen zu bringen. Eine klare Tendenz in diesem Bereich wird sich zukünftig vermutlich Richtung Integration von Sicherheitstests in CI/CD Pipelines manifestieren.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Literatur

Wissenschaftliche Literatur

[2] H. Agrawal und J. R. Horgan. “Dynamic Program Slicing”. In: *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*. PLDI ’90. New York, NY, USA: ACM, 1990, S. 246–256.

[3] F. E. Allen. “Control Flow Analysis”. In: *SIGPLAN Not.* 5.7 (Juli 1970), S. 1–19.

[32] S. Arzt u. a. “Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps”. In: *ACM SIGPLAN Not.* 49.6 (2014), S. 259–269.

[33] M. Backes, S. Bugiel und E. Derr. “Reliable third-party library detection in android and its security applications”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, S. 356–367.

[34] E. Barker. *Guideline for using cryptographic standards in the federal government: cryptographic mechanisms*. Techn. Ber. National Institute of Standards und Technology, 2016.

[35] M. Bellare u. a. “A concrete security treatment of symmetric encryption”. In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE. 1997, S. 394–403.

[37] A. Bianchi u. a. “Exploitation and mitigation of authentication schemes based on device-public information”. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM. 2017, S. 16–27.

[39] R. Bonett u. a. “Discovering Flaws in Security-Focused Static Analysis Tools for Android using Systematic Mutation”. In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, S. 1263–1280.

[40] S. Bugiel u. a. “Towards Taming Privilege-Escalation Attacks on Android.” In: *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*. 2012, S. 19.

[42] H. Cai und J. Jenkins. “Leveraging Historical Versions of Android Apps for Efficient and Precise Taint Analysis”. In: *Proceedings of the 15th International Conference on Mining Software Repositories*. MSR ’18. New York, NY, USA: ACM, 2018, S. 265–269.

[43] F. Callegati, W. Cerroni und M. Ramilli. “Man-in-the-Middle Attack to the HTTPS Protocol”. In: *IEEE Security & Privacy* 7.1 (2009), S. 78–81.

[58] B. Chess und G. McGraw. “Static Analysis for Security”. In: *IEEE Security and Privacy* 2.6 (Nov. 2004), S. 76–79.

[59] P. Chester u. a. “M-Perm: A Lightweight Detector for Android Permission Gaps”. In: *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2017, S. 217–218.

[60] E. J. Chikofsky und J. H. Cross. “Reverse engineering and design recovery: A taxonomy”. In: *IEEE Software* 7.1 (1990), S. 13–17.

[61] E. Chin und D. Wagner. “Bifocals: Analyzing webview vulnerabilities in android applications”. In: *International Workshop on Information Security Applications*. Springer. 2013, S. 138–159.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[62] E. Chin u. a. “Analyzing Inter-application Communication in Android”. In: *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’11. New York, NY, USA: ACM, 2011, S. 239–252.

[63] S. Chow u. a. “A white-box DES implementation for DRM applications”. In: *ACM Workshop on Digital Rights Management*. Springer. 2002, S. 1–15.

[64] S. Chow u. a. “White-box cryptography and an AES implementation”. In: *International Workshop on Selected Areas in Cryptography*. Springer. 2002, S. 250–270.

[65] C. S. Collberg und C. Thomborson. “Watermarking, tamper-proofing, and obfuscation-tools for software protection”. In: *IEEE Transactions on Software Engineering* 28.8 (2002), S. 735–746.

[66] D. Colton, D. Krutz und M. Mkaouer. “P-Lint: A Permission Smell Detector for Android Applications”. In: *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2017, S. 219–220.

[67] P. Cousot und R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL ’77. New York, NY, USA: ACM, 1977, S. 238–252.

[74] L. Davi u. a. “Privilege escalation attacks on android”. In: *International Conference on Information Security*. Springer. 2010, S. 346–360.

[75] C. Dennis, D. E. Krutz und M. W. Mkaouer. “P-lint: a permission smell detector for android applications”. In: *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE. 2017, S. 219–220.

[76] V. M. Deshpande, M. K. Nair und D. Shah. “Web Application Threats for Data Privacy Security – Detection, Analysis and Mitigation Strategies”. In: *IJSRST* 3.7 (2017), S. 1–17.

[77] D. Di Nardo, F. Pastore und L. Briand. “Generating complex and faulty test data through model-based mutation analysis”. In: *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2015, S. 1–10.

[83] N. Dragoni u. a. “Microservices: yesterday, today, and tomorrow”. In: *Present and Ulterior Software Engineering*. Springer, 2017, S. 195–216.

[84] M. Egele u. a. “A Survey on Automated Dynamic Malware-analysis Techniques and Tools”. In: *ACM Computing Survey* 44.2 (2008), S. 1–42.

[85] M. Egele u. a. “An Empirical Study of Cryptographic Misuse in Android Applications”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS ’13. New York, NY, USA: ACM, 2013, S. 73–84.

[86] D. Ehringer. “The dalvik virtual machine architecture”. In: *Techn. report (March 2010)* 4.8 (2010).

[87] E. Eilam. *Reversing: secrets of reverse engineering*. John Wiley & Sons, 2011.

[88] N. Elenkov. *Android Security Internals: An In-Depth Guide to Android’s Security Architecture*. 1st. San Francisco, CA, USA: No Starch Press, 2014.

[89] K. O. Elish, D. Yao und B. G Ryder. “On the need of precise inter-app ICC classification for detecting Android malware collusions”. In: *Proceedings of IEEE Mobile Security Technologies (MoST), In Conjunction with the IEEE Symposium on Security and Privacy*. 2015.

[91] S. Fahl u. a. “Rethinking SSL development in an appified world”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM. 2013, S. 49–60.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[92] S. Fahl u. a. “Why Eve and Mallory love Android: An analysis of Android SSL (in) security”. In: *Proceedings of the 2012 ACM conference on Computer and Communications Security*. ACM. 2012, S. 50–61.

[93] M. Farooq u. a. “A Critical Analysis on the Security Concerns of Internet of Things (IoT)”. In: *International Journal of Computer Applications* 111.7 (2015), S. 1–6.

[94] P. Faruki u. a. “Android Security: A Survey of Issues, Malware Penetration, and Defenses”. In: *IEEE Communications Surveys Tutorials* 17.2 (2015), S. 998–1022.

[95] P. Faruki u. a. “AndroSimilar: robust statistical feature signature for Android malware detection”. In: *Proceedings of the 6th International Conference on Security of Information and Networks*. ACM. 2013, S. 152–159.

[96] A. P. Felt u. a. “Android permissions demystified”. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM. 2011, S. 627–638.

[97] A. P. Felt u. a. “Android permissions: User attention, comprehension, and behavior”. In: *Proceedings of the 8th Symposium on Usable Privacy and Security*. ACM. 2012, S. 3.

[98] A. P. Felt u. a. “Permission Re-Delegation: Attacks and Defenses.” In: *USENIX Security Symposium*. Bd. 30. 2011, S. 88.

[99] R. Fielding u. a. *RFC-2616: Hypertext transfer protocol–HTTP/1.1*. Techn. Ber. 1999.

[100] PUB FIPS. “180-1. secure hash standard”. In: *National Institute of Standards and Technology* (1995).

[101] PUB FIPS. “180-4. secure hash standard”. In: *National Institute of Standards and Technology* (2015).

[102] F. Fischer u. a. “Stack overflow considered harmful? the impact of copy&paste on android application security”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, S. 121–136.

[104] N. Freed und J. Klensin. *RFC-4288: Media Type Specifications and Registration Procedures*. 2005.

[105] A. Freier, P. Karlton und P. Kocher. “RFC-6101: The secure sockets layer (SSL) protocol version 3.0”. In: *The Internet Engineering Task Force (IETF)* (2011).

[106] E. Fujisaki und T. Okamoto. “Secure integration of asymmetric and symmetric encryption schemes”. In: *Annual International Cryptology Conference*. Springer. 1999, S. 537–554.

[107] P. Gadient. *Security in Android Applications*. University of Bern, Switzerland, 2017.

[108] E. Gamma u. a. *Design Patterns Elements of Reusable Object-Oriented Software*. USA: Pearson, 1994.

[109] M. Georgiev, S. Jana und V. Shmatikov. “Breaking and fixing origin-based access control in hybrid web/mobile application frameworks”. In: *Proceedings of 2014 the Network and Distributed System Security Symposium (NDSS)*. NIH Public Access. 2014, S. 1.

[110] M. Ghafari, P. Gadient und O. Nierstrasz. “Security smells in Android”. In: *2017 IEEE 17Th International Working conference on Source Code Analysis and Manipulation (SCAM)*. IEEE. 2017, S. 121–130.

[129] M. I. Gordon u. a. “Information Flow Analysis of Android Applications in DroidSafe”. In: *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS*. 2015, S. 110.

[131] S. Gross, A. Tiwari und C. Hammer. “PIAnalyzer: A Precise Approach for PendingIntent Vulnerability Analysis”. In: *European Symposium on Research in Computer Security*. Springer. 2018, S. 41–59.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[132] N. K. Gupta. “Advancements in Cloud Computing Software Testing Research”. In: *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*. 2018, S. 1–6.

[135] H. Heitkötter, S. Hanschke und T. A. Majchrzak. “Evaluating cross-platform development approaches for mobile applications”. In: *International Conference on Web Information Systems and Technologies*. Springer. 2012, S. 120–138.

[137] H. Huang u. a. “A Large-scale Study of Android Malware Development Phenomenon on Public Malware Submission and Scanning Platform”. In: *IEEE Transactions on Big Data* (2018), S. 1–1.

[138] F. Ibrar u. a. “A Study of Static Analysis Tools to Detect Vulnerabilities of Branchless Banking Applications in Developing Countries”. In: *Proceedings of the Ninth International Conference on Information and Communication Technologies and Development*. ICTD ’17. New York, NY, USA: ACM, 2017, 30:1–30:5.

[146] Y. Jiang und Z. Xuxian. “Detecting passive content leaks and pollution in android applications”. In: *Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)*. 2013.

[147] S. Josefsson. *The base16, base32, and base64 data encodings*. Techn. Ber. 2006.

[148] B. Kaliski und J. Staddon. *PKCS# 1: RSA cryptography specifications version 2.0*. Techn. Ber. 1998.

[149] A. Karadimce und D. Bogatinoska. “Using hybrid mobile applications for adaptive multimedia content delivery”. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2014, S. 686–691.

[150] P. Kent und J. Kent. *Official NetScape Javascript Book 1.2*. 11th. Ventana Communications Group, Incorporated, 1996.

[151] B. Khadiranaikar, P. Zavarsky und Y. Malik. “Improving Android application security for intent based attacks”. In: *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2017, S. 62–67.

[152] J. C. King. “Symbolic Execution and Program Testing”. In: *Commun. ACM* 19.7 (Juli 1976), S. 385–394.

[153] W. Klieber u. a. “Android taint flow analysis for app sets”. In: *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*. ACM. 2014, S. 1–6.

[154] A. Kovacheva. “Efficient code obfuscation for Android”. In: *International Conference on Advances in Information Technology*. Springer. 2013, S. 104–119.

[155] W. Landi. “Undecidability of Static Analysis”. In: *ACM Lett. Program. Lang. Syst.* 1.4 (Dez. 1992), S. 323–337.

[156] S. Lee und J. W. Jeon. “Evaluating performance of Android platform using native C for embedded systems”. In: *ICCAS 2010*. IEEE. 2010, S. 1160–1163.

[157] O. Lhotak und L. Hendren. “Scaling Java points-to analysis using S park”. In: *International Conference on Compiler Construction*. Springer. 2003, S. 153–169.

[158] L. Li u. a. “Iccta: Detecting inter-component privacy leaks in android apps”. In: *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press. 2015, S. 280–291.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[159] L. Li u. a. “Static Analysis of Android Apps”. In: *Information and Software Technology* 88.C (2017), S. 67–95.

[160] L. Li u. a. “Understanding android app piggybacking: A systematic study of malicious code grafting”. In: *IEEE Transactions on Information Forensics and Security* 12.6 (2017), S. 1269–1284.

[161] X. Li u. a. “On modeling eavesdropping attacks in wireless networks”. In: *Journal of Computational Science* 11 (2015), S. 196–204.

[162] X. Lin u. a. “Automated forensic analysis of mobile applications on Android devices”. In: *Digital Investigation* 26 (2018), S59–S66.

[163] M. Linares-Vasquez u. a. “Enabling mutation testing for android apps”. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM. 2017, S. 233–244.

[164] T. Luo u. a. “Attacks on WebView in the Android system”. In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM. 2011, S. 343–352.

[165] R. Mahmood u. a. “A whitebox approach for automated security testing of Android applications on the cloud”. In: *2012 7th International Workshop on Automation of Software Test (AST)*. 2012, S. 22–28.

[167] N. Matsakis und S. Klock. “The Rust Language”. In: *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*. HILT ’14. New York, NY, USA: ACM, 2014, S. 103–104.

[169] B. P. Miller u. a. *Fuzz revisited: A re-examination of the reliability of UNIX utilities and services*. Techn. Ber. University of Wisconsin-Madison Department of Computer Sciences, 1995.

[172] T. Mouelhi, Y. Le Traon und B. Baudry. “Mutation analysis for security tests qualification”. In: *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE. 2007, S. 233–242.

[174] M. Neugschwandtner, M. Lindorfer und C. Platzer. “A view to a kill: Webview exploitation”. In: *Presented as part of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats*. 2013.

[175] J. Newsome und D. X. Song. “Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software.” In: *Proceedings of the 5th Network and Distributed System Security Symposium (NDSS)*. 2005, S. 3–4.

[176] M. Nieles, K. Dempsey und V. Y. Pillitteri. *An Introduction to Information Security*. Techn. Ber. 2017.

[183] É. Payet und F. Spoto. “Static analysis of Android programs”. In: *International Conference on Automated Deduction* (2011).

[184] P. Pearce u. a. “Addroid: Privilege separation for applications and advertisers in android”. In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. Acm. 2012, S. 71–72.

[187] S. Poeplau u. a. “Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications”. In: ACM, 2014.

[188] J. Postel und J. Reynolds. “RFC-959: File transfer protocol (ftp)”. In: *InterNet Network Working Group* (1985).

[193] C. Qian u. a. “Vulhunter: toward discovering vulnerabilities in android applications”. In: *IEEE Micro* 35.1 (2015), S. 44–53.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[194] V. Rastogi, Y. Chen und W. Enck. “AppsPlayground: Automatic Security Analysis of Smartphone Applications”. In: *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*. CODASPY ’13. New York, NY, USA: ACM, 2013, S. 209–220.

[198] E. Rescorla. *The transport layer security (TLS) protocol version 1.3*. Techn. Ber. 2018.

[199] R. Rivest. *The MD4 message-digest algorithm*. Techn. Ber. 1992.

[200] R. Rivest. *The MD5 message-digest algorithm*. Techn. Ber. 1992.

[201] I. Rojas, S. Meireles und A. Dias-Neto. “Cloud-Based Mobile App Testing Framework: Architecture, Implementation and Execution”. In: *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. SAST. New York, NY, USA: ACM, 2016, 10:1–10:10.

[202] I. Rojas, S. Meireles und A. Dias-Neto. “Cloud-Based Mobile App Testing Framework: Architecture, Implementation and Execution”. In: *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. SAST. New York, NY, USA: ACM, 2016, 10:1–10:10.

[205] E. J. Schwartz, T. Avgerinos und D. Brumley. “All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask)”. In: *2010 IEEE Symposium on Security and Privacy*. 2010, S. 317–331.

[209] M. Sharir und A. Pnueli. “Two approaches to interprocedural data flow analysis”. In: *Program Flow Analysis: Theory and Applications*. New York University. Courant Institute of Mathematical Sciences, 1978, S. 189–211.

[210] F. Shezan, S. Afroze und A. Iqbal. “Vulnerability detection in recent Android apps: An empirical study”. In: *2017 International Conference on Networking, Systems and Security (NSysS)*. 2017, S. 55–63.

[211] Y. Shoshitaishvili u. a. “SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis”. In: *IEEE Symposium on Security and Privacy*. 2016.

[212] M. Shreedhar und G. Varghese. “Efficient fair queuing using deficit round-robin”. In: *IEEE/ACM Transactions on networking* 3 (1996), S. 375–385.

[214] G. J. Simmons. “Symmetric and asymmetric encryption”. In: *ACM Computing Surveys (CSUR)* 11.4 (1979), S. 305–330.

[216] W. Song, Q. Huang und J. Huang. “Understanding JavaScript Vulnerabilities in Large Real-World Android Applications”. In: *IEEE Transactions on Dependable and Secure Computing* (2018).

[224] K. Tam u. a. “The Evolution of Android Malware and Android Analysis Techniques”. In: *ACM Computing Survey* 49.4 (2017), 76:1–76:41.

[225] G. Tan und J. Croft. “An Empirical Security Study of the Native Code in the JDK.” In: *Usenix Security Symposium*. 2008, S. 365–378.

[226] V. F. Taylor und I. Martinovic. “Short Paper: A Longitudinal Study of Financial Apps in the Google Play Store”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, S. 302–309.

[231] D. R. Thomas, A. R Beresford und A Rice. “Security metrics for the android ecosystem”. In: *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM. 2015, S. 87–98.

[235] T. Wasserman. “Software engineering issues for mobile application development”. In: *FoSER 2010* (2010).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[236] F. Wei u. a. “Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’14. New York, NY, USA: ACM, 2014, S. 1329–1341.

[237] F. Wei u. a. “Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps”. In: *ACM Trans. Priv. Secur.* 21.3 (2018), 14:1–14:32.

[238] F. Wei u. a. “Deep ground truth analysis of current android malware”. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2017, S. 252–276.

[239] F. Wei u. a. “JN-SAF: Precise and Efficient NDK/JNI-aware Inter-language Static Analysis Framework for Security Vetting of Android Applications with Native Code”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, S. 1137–1150.

[240] X. Wei und M. Wolf. “A survey on HTTPS implementation by Android apps: issues and countermeasures”. In: *Applied Computing and Informatics* 13.2 (2017), S. 101–117.

[241] X. Wei u. a. “Permission evolution in the android ecosystem”. In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012, S. 31–40.

[242] M. Weiser. “Program Slicing”. In: *Proceedings of the 5th International Conference on Software Engineering*. ICSE ’81. Piscataway, NJ, USA: IEEE Press, 1981, S. 439–449.

[246] P. Wijesekera u. a. “The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, S. 1077–1093.

[247] Z. Yang und M. Yang. “Leakminer: Detect information leakage on android with static taint analysis”. In: *2012 Third World Congress on Software Engineering*. IEEE. 2012, S. 101–104.

[248] Z. Yang u. a. “Appintent: Analyzing sensitive data transmission in android for privacy leakage detection”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM. 2013, S. 1043–1054.

[249] L. Yu, X. Luo und T. Zhang. “Can We Trust the Privacy Policies of Android Apps?” In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2016, S. 538–549.

[250] H. Zhang, D. She und Z. Qian. “Android root and its providers: A double-edged sword”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, S. 1093–1104.

[251] X. Zhang u. a. “Life after App Uninstallation: Are the Data Still Alive? Data Residue Attacks on Android.” In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2016.

[252] Y. Zhou und X. Jiang. “Dissecting android malware: Characterization and evolution”. In: *2012 IEEE Symposium on Security and Privacy*. IEEE. 2012, S. 95–109.

[253] C. Zuo u. a. “Automatic Forgery of Cryptographically Consistent Messages to Identify Security Vulnerabilities in Mobile Services.” In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2016.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Online-Referenzen

[1] *Advanced Encryption Standard*. URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (besucht am 28. 06. 2019).

[4] *AM-TaaS Github Projekt*. URL: https://github.com/IKVillanes/amtaas (besucht am 28. 06. 2019).

[5] *Amazon Elastic Container Registry*. URL: https://aws.amazon.com/de/ecr/ (besucht am 28. 06. 2019).

[6] *Amazon Elastic Container Registry*. URL: https://aws.amazon.com/de/ecs/ (besucht am 28. 06. 2019).

[7] *Amazon Relational Database Service*. URL: https://aws.amazon.com/de/rds/ (besucht am 28. 06. 2019).

[8] *Amazon Web Services*. URL: https://aws.amazon.com (besucht am 28. 06. 2019).

[9] *Android 6.0 Changes*. URL: https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html (besucht am 28. 06. 2019).

[10] *Android AllowAllHostnameVerifier*. URL: https://developer.android.com/reference/org/apache/http/conn/ssl/AllowAllHostnameVerifier (besucht am 28. 06. 2019).

[11] *Android Debug Bridge (adb)*. URL: https://developer.android.com/studio/command-line/adb (besucht am 28. 06. 2019).

[12] *Android Documentation SSLSocket*. URL: https://developer.android.com/reference/javax/net/ssl/SSLSocket (besucht am 28. 06. 2019).

[13] *Android Fundamentals*. URL: https://developer.android.com/guide/components/fundamentals (besucht am 28. 06. 2019).

[14] *Android Intent Dokumentation*. URL: https://developer.android.com/guide/components/intents-filters.html (besucht am 28. 06. 2019).

[15] *Android keystore system*. URL: https://developer.android.com/training/articles/keystore (besucht am 28. 06. 2019).

[16] *Android Komponentendokumentation*. URL: https://developer.android.com/guide/components/fundamentals.html (besucht am 28. 06. 2019).

[17] *Android NDK*. URL: https://developer.android.com/ndk/ (besucht am 28. 06. 2019).

[18] *Android Permission System*. URL: http://www.androiddocs.com/guide/topics/security/permissions.html (besucht am 28. 06. 2019).

[19] *Android Permissions Overview*. URL: https://developer.android.com/guide/topics/permissions/overview (besucht am 28. 06. 2019).

[20] *Android Q privacy: Changes to data and identifiers*. URL: https://developer.android.com/preview/privacy/data-identifiers (besucht am 28. 06. 2019).

[21] *Android Shared Preferences*. URL: https://developer.android.com/reference/android/content/SharedPreferences (besucht am 28. 06. 2019).

[22] *Android Sicherheitsregelen des CERT-Abteilung des Software Engineering Institut der Carnegie Mellon Universität*. URL: https://wiki.sei.cmu.edu/confluence/display/android/Android+Only (besucht am 28. 06. 2019).

[23] *Android TelephonyManager*. URL: https://developer.android.com/reference/android/telephony/TelephonyManager (besucht am 28. 06. 2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[24] *AndroidViewClient Github Projekt*. URL: https://github.com/dtmilano/AndroidViewClient (besucht am 28. 06. 2019).

[25] *Angular*. URL: https://angular.io/ (besucht am 28. 06. 2019).

[26] *Angular-CLI*. URL: https://cli.angular.io/ (besucht am 28. 06. 2019).

[27] *Anzahl der verfügbaren Apps im Google Play Store in ausgewählten Monaten von Juli 2015 bis Juni 2018*. URL: https://de.statista.com/statistik/daten/studie/74368/umfrage/anzahl-der-verfuegbaren-apps-im-google-play-store/ (besucht am 28. 06. 2019).

[28] *Apache Maven Project*. URL: https://maven.apache.org/docs/3.3.9/release-notes.html (besucht am 28. 06. 2019).

[29] *App security best practicesl*. URL: https://developer.android.com/topic/security/best-practices.html (besucht am 28. 06. 2019).

[30] *Argus Static Analysis Framework*. URL: http://pag.arguslab.org/argus-saf (besucht am 28. 06. 2019).

[31] *Argus Static Analysis Framework*. URL: http://pag.arguslab.org/jawa-language (besucht am 28. 06. 2019).

[36] *Best practices for unique identifiers*. URL: https://developer.android.com/training/articles/user-data-ids (besucht am 28. 06. 2019).

[38] *Binder Documentation*. URL: https://developer.android.com/reference/android/os/Binder (besucht am 28. 06. 2019).

[41] *Building web apps in WebView*. URL: https://developer.android.com/guide/webapps/webview (besucht am 28. 06. 2019).

[44] *Cert Division - Carnegie Mellon Universität*. URL: https://www.sei.cmu.edu/about/divisions/cert/index.cfm (besucht am 28. 06. 2019).

[45] *Cert Regel DRD01-X*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD01-X.+Limit+the+accessibility+of+an+app\%27s+sensitive+content+provider (besucht am 28. 06. 2019).

[46] *Cert Regel DRD02-J*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD02-J.+Do+not+allow+WebView+to+access+sensitive+local+resource+through+file+scheme (besucht am 28. 06. 2019).

[47] *Cert Regel DRD03-J*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD03-J.+Do+not+broadcast+sensitive+information+using+an+implicit+intent (besucht am 28. 06. 2019).

[48] *Cert Regel DRD07-X*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD07-X.+Protect+exported+services+with+strong+permissions (besucht am 28. 06. 2019).

[49] *Cert Regel DRD09*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD09.+Restrict+access+to+sensitive+activities (besucht am 28. 06. 2019).

[50] *Cert Regel DRD10-X*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD10-X.+Do+not+release+apps+that+are+debuggable (besucht am 28. 06. 2019).

[51] *Cert Regel DRD11*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD11.+Ensure+that+sensitive+data+is+kept+secure (besucht am 28. 06. 2019).

[52] *Cert Regel DRD15-J*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD15-J.+Consider+privacy+concerns+when+using+Geolocation+API (besucht am 28. 06. 2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[53] *Cert Regel DRD16-X*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD16-X.+Explicitly+define+the+exported+attribute+for+private+components (besucht am 28. 06. 2019).

[54] *Cert Regel DRD18*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD18.+Do+not+use+the+default+behavior+in+a+cryptographic+library+if+it+does+not+use+recommended+practices (besucht am 28. 06. 2019).

[55] *Cert Regel DRD20-C*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD20-C.+Specify+permissions+when+creating+files+via+the+NDK (besucht am 28. 06. 2019).

[56] *Cert Regel DRD21-J*. URL: https://wiki.sei.cmu.edu/confluence/display/android/DRD21-J.+Always+pass+explicit+intents+to+a+PendingIntent (besucht am 28. 06. 2019).

[57] *CFR Dekompiler Homepage*. URL: http://www.benf.org/other/cfr/ (besucht am 28. 06. 2019).

[68] *CSS Specification*. URL: https://www.w3.org/Style/CSS/specs.de.html (besucht am 28. 06. 2019).

[69] *CWE-287: Improper Authentication*. URL: https://cwe.mitre.org/data/definitions/287.html (besucht am 28. 06. 2019).

[70] *CWE-639: Authorization Bypass Through User-Controlled Key*. URL: http://cwe.mitre.org/data/definitions/639.html (besucht am 28. 06. 2019).

[71] *CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)*. URL: http://cwe.mitre.org/data/definitions/79.html (besucht am 28. 06. 2019).

[72] *Dalvik Executable format*. URL: https://source.android.com/devices/tech/dalvik/dex-format (besucht am 28. 06. 2019).

[73] *Data and file storage overview*. URL: https://developer.android.com/guide/topics/data/data-storage (besucht am 28. 06. 2019).

[78] *Docker*. URL: https://www.docker.com/ (besucht am 28. 06. 2019).

[79] *Docker Container Dokumentation*. URL: https://www.docker.com/resources/what-container (besucht am 28. 06. 2019).

[80] *Docker Dokumentation*. URL: https://www.docker.com/what-docker (besucht am 28. 06. 2019).

[81] *Docker File Dokumentation*. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ (besucht am 28. 06. 2019).

[82] *Docker Release Notes*. URL: https://docs.docker.com/engine/release-notes/ (besucht am 28. 06. 2019).

[90] *Encryption*. URL: https://source.android.com/security/encryption/ (besucht am 28. 06. 2019).

[103] *FlowDroid Data Flow Analysis Tool*. URL: https://github.com/secure-software-engineering/FlowDroid (besucht am 28. 06. 2019).

[111] *Github Androbugs Projekt*. URL: https://github.com/AndroBugs (besucht am 28. 06. 2019).

[112] *Github Androguard Projekt*. URL: https://github.com/androguard/androguard (besucht am 28. 06. 2019).

[113] *Github Androguard Projekt Bug Report*. URL: https://github.com/androguard/androguard/issues/234 (besucht am 28. 06. 2019).

[114] *Github Androwarn Analyzer Projekt*. URL: https://github.com/maaaaz/androwarn/commit/e0e5ad0451be361e4cc74b6507c57b5bf0eacf9e (besucht am 28. 06. 2019).

[115] *Github APK-Parser Projekt*. URL: https://github.com/hsiafan/apk-parser (besucht am 28. 06. 2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[116] *Github ApkAnalyser Projekt*. URL: https://github.com/sonyxperiadev/ApkAnalyser/wiki (besucht am 28. 06. 2019).

[117] *Github APKTool Projekt*. URL: https://ibotpeaches.github.io/Apktool/ (besucht am 28. 06. 2019).

[118] *Github dex2jar Projekt*. URL: https://github.com/pxb1988/dex2jar (besucht am 28. 06. 2019).

[119] *Github DroidSafe Projekt*. URL: https://github.com/MIT-PAC/droidsafe-src (besucht am 28. 06. 2019).

[120] *Github Enjarify Projekt*. URL: https://github.com/google/enjarify (besucht am 28. 06. 2019).

[121] *Github M-Perm Projekt*. URL: https://mperm.github.io/index.html (besucht am 28. 06. 2019).

[122] *Github P-Lint Best Practices Analyzer Projekt*. URL: https://github.com/P-Lint/P-Lint/commit/7ccdd4aafa49997f050a5fc7ec733d31710bb63e (besucht am 28. 06. 2019).

[123] *Github PScout Projekt*. URL: https://security.csl.toronto.edu/pscout/ (besucht am 28. 06. 2019).

[124] *Github Quick Andrid Review Kit Projekt*. URL: https://github.com/linkedin/qark/commit/f3a77c8f9062f70b4201b67018f5abeb5ff5ad89 (besucht am 28. 06. 2019).

[125] *Github Secure, Unified, Powerful and Extensible Rust Android Analyzer Projekt*. URL: http://superanalyzer.rocks/ (besucht am 28. 06. 2019).

[126] *Github Smali Projekt*. URL: https://github.com/JesusFreke/smali (besucht am 28. 06. 2019).

[127] *Google Play Store*. URL: https://play.google.com/store (besucht am 28. 06. 2019).

[128] *Google Play Store Top Selling Free*. URL: https://play.google.com/store/apps/category/ANDROID_WEAR/collection/topselling_free (besucht am 28. 06. 2019).

[130] *GPlayCLI*. URL: https://github.com/matlink/gplaycli (besucht am 28. 06. 2019).

[133] *H2 Database Engine*. URL: https://www.h2database.com/html/main.html (besucht am 28. 06. 2019).

[134] *H2 Database Engine - Maven Dependency*. URL: https://mvnrepository.com/artifact/com.h2database/h2/1.4.196 (besucht am 28. 06. 2019).

[136] *HTML5 Specification*. URL: https://www.w3.org/TR/html5/ (besucht am 28. 06. 2019).

[139] *Improve your code with lint checks*. URL: https://developer.android.com/studio/write/lint.html (besucht am 28. 06. 2019).

[140] *IntelliJ IDEA*. URL: https://www.jetbrains.com/idea/download/ (besucht am 28. 06. 2019).

[141] *Ionic*. URL: https://ionicframework.com/ (besucht am 28. 06. 2019).

[142] *JASMIN*. URL: http://jasmin.sourceforge.net/ (besucht am 28. 06. 2019).

[143] *Java Archiv Specification*. URL: https://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html (besucht am 28. 06. 2019).

[144] *Java Dekompiler (JD) Homepage*. URL: http://jd.benow.ca/ (besucht am 28. 06. 2019).

[145] *Java SE Development Kit 8*. URL: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html (besucht am 28. 06. 2019).

[166] *Material Components*. URL: https://material.io/ (besucht am 28. 06. 2019).

[168] *Microservices - A definition of this new architectural term*. URL: https://martinfowler.com/articles/microservices.html (besucht am 28. 06. 2019).

[170] *Mobile-Security-Framework MobSF*. URL: https://github.com/MobSF/Mobile-Security-Framework-MobSF (besucht am 28. 06. 2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[171] *Monkeyrunner*. URL: https://developer.android.com/studio/test/monkeyrunner/index (besucht am 28. 06. 2019).

[173] B. Mueller und S. Schleier. *Mobile AppSec Verification*. 2016. URL: https://www.owasp.org/images/f/f2/OWASP_Mobile_AppSec_Verification_Standard_v0.9.2.pdf (besucht am 28. 06. 2019).

[177] *Node-JS*. URL: https://nodejs.org/en/blog/release/v8.9.4/ (besucht am 28. 06. 2019).

[178] *Node Package Manager*. URL: https://www.npmjs.com/package/npm/v/5.6.0 (besucht am 28. 06. 2019).

[179] *OWASP Mobile Top 10*. URL: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10 (besucht am 28. 06. 2019).

[180] *OWASP Reverse Engineering and Code Modification Threats*. URL: https://www.owasp.org/index.php/Technical_Risks_of_Reverse_Engineering_and_Unauthorized_Code_Modification#Reverse_Engineering_and_Code_Analysis_Technical_Risks (besucht am 28. 06. 2019).

[181] *OWASP Reverse Engineering Threats*. URL: https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project (besucht am 28. 06. 2019).

[182] *P-Lint Projekt Homepage*. URL: https://p-lint.github.io/index.html (besucht am 28. 06. 2019).

[185] *PhoneGap*. URL: https://phonegap.com/ (besucht am 28. 06. 2019).

[186] *Platform Architecture*. URL: https://developer.android.com/guide/platform (besucht am 28. 06. 2019).

[189] *Procyon Bitbucket Projekt*. URL: https://bitbucket.org/mstrobel/procyon/wiki/Java/Decompiler (besucht am 28. 06. 2019).

[190] *Prognose zu den Marktanteilen der Betriebssysteme am Absatz vom Smartphones weltweit in den Jahren 2018 und 2022*. URL: https://de.statista.com/statistik/daten/studie/182363/umfrage/prognostizierte-marktanteile-bei-smartphone-betriebssystemen/ (besucht am 28. 06. 2019).

[191] *Python 3.4.0*. URL: https://www.python.org/downloads/release/python-340/ (besucht am 28. 06. 2019).

[192] *Python Flask*. URL: http://flask.pocoo.org/ (besucht am 28. 06. 2019).

[195] *React Native*. URL: http://www.reactnative.com/ (besucht am 28. 06. 2019).

[196] *Remote Code Execution Sicherheitslücke*. URL: https://nvd.nist.gov/vuln/detail/CVE-2013-4710 (besucht am 28. 06. 2019).

[197] *Remote Procedure Call*. URL: https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.progcomc/ch8_rpc.html (besucht am 28. 06. 2019).

[203] *Rust*. URL: https://www.rust-lang.org/ (besucht am 28. 06. 2019).

[204] *Sammlung großartiger Android-Analysewerkzeuge*. URL: https://github.com/ashishb/android-security-awesome (besucht am 28. 06. 2019).

[206] *Security Vulnerabilities - CVE Details*. URL: https://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html (besucht am 28. 06. 2019).

[207] *Security with HTTPS and SSL*. URL: https://developer.android.com/training/articles/security-ssl (besucht am 28. 06. 2019).

[208] *Security with HTTPS and SSL*. URL: https://developer.android.com/training/articles/security-ssl (besucht am 28. 06. 2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[213] *Sign Your App*. URL: https://developer.android.com/studio/publish/app-signing (besucht am 28. 06. 2019).

[215] *Smashing the stack for fun and profit*. URL: http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf (besucht am 28. 06. 2019).

[217] *Soot - A framework for analyzing and transforming Java and Android applications*. URL: https://sable.github.io/soot/ (besucht am 28. 06. 2019).

[218] *Spring Boot*. URL: https://spring.io/projects/spring-boot (besucht am 28. 06. 2019).

[219] *Spring Cloud - Netflix Eureka*. URL: https://www.baeldung.com/spring-cloud-netflix-eureka (besucht am 28. 06. 2019).

[220] *Spring Data JPA*. URL: https://spring.io/projects/spring-data-jpa (besucht am 28. 06. 2019).

[221] *SQL Injection Knowledge Base*. URL: https://www.websec.ca/kb/sql_injection (besucht am 28. 06. 2019).

[222] *SQLite*. URL: https://www.sqlite.org/index.html (besucht am 28. 06. 2019).

[223] *Stowaway Analysis Tool*. URL: http://www.android-permissions.org/ (besucht am 28. 06. 2019).

[227] *The Java Virtual Machine Specification*. URL: https://docs.oracle.com/javase/specs/jvms/se8/html/ (besucht am 28. 06. 2019).

[228] *The MariaDB Foundation*. URL: https://mariadb.org/ (besucht am 28. 06. 2019).

[229] *The OWASP Foundation*. URL: https://www.owasp.org/index.php/Main_Page (besucht am 28. 06. 2019).

[230] *The Scala Programming Language*. URL: https://www.scala-lang.org/ (besucht am 28. 06. 2019).

[232] *UI Automator*. URL: https://developer.android.com/training/testing/ui-automator (besucht am 28. 06. 2019).

[233] *Virustotal Homepage*. URL: https://www.virustotal.com/#/home/upload (besucht am 28. 06. 2019).

[234] *W3C Geolocation API Spezifikation*. URL: https://www.w3.org/TR/geolocation-API/\#api_description (besucht am 28. 06. 2019).

[243] *Welcome to Jinja2*. URL: http://jinja.pocoo.org/docs/2.10/ (besucht am 28. 06. 2019).

[244] *What Are RESTful Web Services?* URL: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html (besucht am 28. 06. 2019).

[245] *White-box cryptography: hiding keys in software*. URL: http://whiteboxcrypto.com/files/2012_misc.pdf (besucht am 28. 06. 2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

A Anhang

A.1 Integrierte Werkzeuge

Name	Version	Referenz
AndroBugs	1.0.0	[111]
Androwarn	Commit e0e5ad0451be361e4cc74b6507c57b5bf0eacf9e	[114]
APK-Parser	2.2.0	[115]
Argus-SAF	3.1.3	[30]
P-Lint	Commit 7ccdd4aafa49997f050a5fc7ec733d31710bb63e	[122]
QARK	Commit f3a77c8f9062f70b4201b67018f5abeb5ff5ad89	[124]
SUPER	0.4.1	[125]
GPlayCli	3.25	[130]

Tabelle A.1: In das Framework integrierte Werkzeuge

A.2 Software-Versionen

Name	Version	Referenz
IntelliJ IDEA	2017.3.4	[140]
Java	1.8.0_121	[145]
Python	3.4	[191]
Maven	3.3.9	[28]
NodeJS	8.9.4	[177]
NPM	5.6.0	[178]
H2	1.4.196	[134]
Docker	18.06.1-ce	[82]

Tabelle A.2: Zur Entwicklung eingesetzte Software-Versionen

A.3 Konfiguration der Sicherheitseinstufung

```
evaluation:
  issueDefinition:
    ICC:
      callAndroBug:
        PERMISSION_EXPORTED: WARNING
        PERMISSION_EXPORTED_GOOGLE: WARNING
        PERMISSION_PROVIDER_EXPLICIT_EXPORTED: WARNING
        PERMISSION_INTENT_FILTER_MISCONFIG: WARNING
        PERMISSION_NO_PREFIX_EXPORTED: POTENTIALINSECURE
        PERMISSION_PROVIDER_IMPLICIT_EXPORTED: POTENTIALINSECURE
      callQark:
        manifest: WARNING
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
90          callSuper:
91            Exported provider: WARNING
92            Exported receiver: WARNING
93            Exported activity: WARNING
94            Exported activity-alias: WARNING
95            Exported service: WARNING
96        Intent:
97          callAndroBug:
98            PERMISSION_IMPLICIT_SERVICE: WARNING
99          callQark:
100           pendingintent: INSECURE
101           Implicit Intent: WARNING
102       Debug:
103         callAndroBug:
104           DEBUGGABLE: INSECURE
105         callSuper:
106           Manifest Debug: INSECURE
107       WebviewAndXSS:
108         callAndroBug:
109           WEBVIEW_RCE: POTENTIALINSECURE
110           SSL_WEBVIEW: INSECURE
111           WEBVIEW_ALLOW_FILE_ACCESS: WARNING
112           WEBVIEW_JS_ENABLED: WARNING
113         callQark:
114           webview: WARNING
115         callSuper:
116           WebView ignores SSL errors: INSECURE
117           WebView XSS: POTENTIALINSECURE
118       SQL:
119         callSuper:
120           SQL injection: POTENTIALINSECURE
121           API_Version:
122         callAndroBug:
123           DB_DEPRECATED_USE1: INSECURE
124           FRAGMENT_INJECTION: INSECURE
125           MANIFEST_GCM: WARNING
126           HTTPURLCONNECTION_BUG: WARNING
127           DB_SQLITE_JOURNAL: WARNING
128       Permission:
129         callAndroBug:
130           USE_PERMISSION_ACCESS_MOCK_LOCATION: INSECURE
131           PERMISSION_GROUP_EMPTY_VALUE: INSECURE
132           USE_PERMISSION_SYSTEM_APP: INSECURE
133           USE_PERMISSION_CRITICAL: INSECURE
134           PERMISSION_DANGEROUS: POTENTIALINSECURE
135           PERMISSION_NORMAL: WARNING
136         callPlint:
137           2: SECURE
138           3: SECURE
139           4: SECURE
140           8: SECURE
141           9: WARNING
142           14: WARNING
143           5: SECURE
144           7: POTENTIALINSECURE
145       Native_Code:
146         callAndrowarn:
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
147            Native_code: WARNING
148            callAndroBug:
149            NATIVE_METHODS: WARNING
150            NATIVE_LIBS_LOADING: WARNING
151        Repacking:
152          callAndroBug:
153            CERT_SIGNED: INSECURE
154            callSuper:
155            Android Debug Certificate: POTENTIALINSECURE
156            Expired certificate: POTENTIALINSECURE
157        Crypto:
158          callAndroBug:
159            HACKER_DB_KEY: SECURE
160            HACKER_BASE64_STRING_DECODE: POTENTIALINSECURE
161            HACKER_BASE64_URL_DECODE: POTENTIALINSECURE
162          callSuper:
163            Weak Algorithms: INSECURE
164            Base64 Encode: POTENTIALINSECURE
165          callQark:
166            crypto: INSECURE
167        Certificate_Validation:
168          callSuper:
169            WebView ignores SSL errors: INSECURE
170            Accepting all SSL certificates: INSECURE
171            SSL getInsecure method: INSECURE
172          callAndroBug:
173            SSL_CN1: INSECURE
174            SSL_CN2: INSECURE
175            SSL_CN3: INSECURE
176            SSL_WEBVIEW: INSECURE
177            SSL_X509: INSECURE
178          callQark:
179            certificate: INSECURE
180        Data_Leakage:
181          callAndroBug:
182            MASTER_KEY: INSECURE
183            MASTER_KEY_SYSTEM_APP: INSECURE
184            SENSITIVE_DEVICE_ID: WARNING
185            SENSITIVE_SECURE_ANDROID_ID: WARNING
186            SENSITIVE_SMS: WARNING
187            SHARED_USER_ID: POTENTIALINSECURE
188            COMMAND: POTENTIALINSECURE
189            COMMAND_SU: POTENTIALINSECURE
190            COMMAND_MAYBE_SYSTEM: POTENTIALINSECURE
191            FILE_DELETE: SECURE
192          callSuper:
193            Hidden fields: POTENTIALINSECURE
194            IP Disclosure: POTENTIALINSECURE
195            Sending sms-mms: WARNING
196            Cell Location (Base Stations): POTENTIALINSECURE
197            Super user privileges.: POTENTIALINSECURE
198            GPS location: POTENTIALINSECURE
199            System command execution: POTENTIALINSECURE
200            Rooted device detection: INSECURE
201            Get Device ID: WARNING
202            Get SIM Serial: WARNING
203            Get SIM Operator: WARNING
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
204            Get SIM OperatorName: WARNING
205          callAndrowarn:
206            telephony_identifiers_leakage: POTENTIALINSECURE
207            telephony_services_abuse: POTENTIALINSECURE
208            Insecure_Communication:
209          callAndroBug:
210            SSL_URLS_NOT_IN_HTTPS: INSECURE
211            SSL_DEFAULT_SCHEME_NAME: INSECURE
212          callAndrowarn:
213            suspicious_connection_establishment: POTENTIALINSECURE
214        Data_Storage:
215          callAndroBug:
216            MODE_WORLD_READABLE_OR_MODE_WORLD_WRITEABLE: INSECURE
217            HACKER_DB_KEY: POTENTIALINSECURE
218            ALLOW_BACKUP: SECURE
219          callQark:
220            filepermission: INSECURE
221            manifest: WARNING
222          callSuper:
223            World readable permissions: INSECURE
224            World writable permissions: INSECURE
225            Write-Read in external storage: INSECURE
226            Temp File Use: POTENTIALINSECURE
227            Allows Backup: SECURE
228        Secret_Storage:
229          callAndroBug:
230            HACKER_KEYSTORE_SSL_PINNING: INSECURE
231            HACKER_KEYSTORE_NO_PWD: INSECURE
232            KEYSTORE_TYPE_CHECK: INSECURE
233          callAndrowarn:
234            connection_interfaces_exfiltration: INSECURE
235          callSuper:
236            Certificate or Keystore disclosure: WARNING
```

Listing A.1: Zur Evaluierung eingesetzte Konfiguration der Sicherheitseinstufung

Google Play Kategorie	Kein Ergebnis	Sicher	Warnung	Potenziell unsicher	Unsicher
ANDROID_WEAR	10	0	11	4	75
ART_AND_DESIGN	11	0	10	3	76
AUTO_AND_VEHICLES	13	0	12	2	73
BEAUTY	6	0	10	1	83
BOOKS_AND_REFERENCE	10	0	5	3	82
BUSINESS	18	0	8	7	67
COMICS	6	0	13	0	81
COMMUNICATION	16	0	5	2	77
DATING	7	0	9	4	80
EDUCATION	14	0	9	3	74
ENTERTAINMENT	8	0	10	0	82
EVENTS	9	0	8	2	81
FAMILY	11	0	8	11	70
FAMILY_ACTION	8	0	13	9	70
FAMILY_BRAINGAMES	7	0	4	5	84
FAMILY_CREATE	12	0	2	7	79

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

FAMILY_EDUCATION	16	0	12	7	65
FAMILY_MUSICVIDEO	7	0	5	4	84
FAMILY_PRETEND	16	0	12	3	69
FINANCE	14	0	18	3	65
FOOD_AND_DRINK	11	0	13	3	73
GAME	9	0	16	7	68
GAME_ACTION	13	0	18	8	61
GAME_ADVENTURE	11	0	6	2	81
GAME_ARCADE	12	0	13	8	67
GAME_BOARD	7	0	11	6	76
GAME_CARD	8	0	14	4	74
GAME_CASINO	11	0	12	3	74
GAME_CASUAL	10	0	7	1	82
GAME_EDUCATIONAL	14	0	13	4	69
GAME_MUSIC	4	0	9	1	86
GAME_PUZZLE	11	0	14	5	70
GAME_RACING	14	0	17	5	64
GAME_ROLE_PLAYING	10	0	10	6	74
GAME_SIMULATION	18	0	11	6	65
GAME_SPORTS	4	0	8	5	83
GAME_STRATEGY	13	0	13	2	72
GAME_TRIVIA	10	0	14	1	75
GAME_WORD	11	0	8	5	76
HEALTH_AND_FITNESS	11	0	12	0	77
HOUSE_AND_HOME	17	0	7	2	74
LIBRARIES_AND_DEMO	9	0	9	10	72
LIFESTYLE	8	0	13	1	78
MAPS_AND_NAVIGATION	11	0	9	1	79
MEDICAL	16	0	15	7	62
MUSIC_AND_AUDIO	12	0	3	0	85
NEWS_AND_MAGAZINES	13	0	5	1	81
PARENTING	9	0	10	6	75
PERSONALIZATION	7	0	7	1	85
PHOTOGRAPHY	13	0	6	1	80
PRODUCTIVITY	11	0	12	3	74
SHOPPING	13	0	7	6	74
SOCIAL	10	0	5	0	85
SPORTS	12	0	10	0	78
TOOLS	7	0	10	3	80
TRAVEL_AND_LOCAL	6	0	10	2	82
VIDEO_PLAYERS	8	0	5	1	86
WEATHER	8	0	1	3	88

Tabelle A.3: Verteilung der Sicherheitsrisiken in den exisiterenden Google Play Store-Kategorien

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub