Bachelorarbeit

Carlos Alberto da Silva Gonçalves

Lokale Sicherheitsanalyse mobiler Endgeräte an Beispiel von Android

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Carlos Alberto da Silva Gonçalves

Lokale Sicherheitsanalyse mobiler Endgeräte an Beispiel von Android

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Herr Prof. Dr. Kossakowski
Zweitgutachter: Herr Prof. Dr. Hübner

Eingereicht am: 28. April 2017

Carlos Alberto da Silva Gonçalves

Thema der Arbeit

Lokale Sicherheitsanalyse mobiler Endgeräte an Beispiel von Android

Stichworte

Google, Android, Android Sicherheitsarchitektur, Sicherheitslücken, mobile Applikation

Kurzzusammenfassung

Die vorliegende Bachelorarbeit thematisiert Sicherheitsfunktionen und in mobilen Endgeräten aufgetretene Sicherheitslücken anhand von Googles mobilem Betriebssystem Android (Version 4.1 bis 6.0). Darüber hinaus werden dem Benutzer die Gefahren ebendieser Sicherheitslücken durch Angabe der bekannten Schwachstellen angezeigt. Daneben werden plattformübergreifende Schwachstellen, die auch für iOS und Windows Phone gelten, erläutert. Im Rahmen der Bachelorarbeit wird auf dieser Basis eine mobile Applikation für Android entwickelt, die den Benutzer eines entsprechenden mobilen Endgeräts vor Schwachstellen in der jeweils aktuellen Betriebssystem-Version warnt und ihm Hinweise dazu gibt.

Carlos Alberto da Silva Gonçalves

Title of the paper

Local security analysis of mobile devices on the example of Android

Keywords

Google, Android, Android Mobile security architecture, security vulnerability, mobile Application

Abstract

The present thesis deals with security functions and security vulnerabilities in mobile devices with Google's operating system Android. In addition, the dangers of the security vulnerabilities are shown by indicating, which known weak points are present. There are also vulnerabilities that are cross-platform, which means they also apply to iOS and Windows Phone. A mobile application is developed, which alerts against vulnerabilities in its current operating system version and gives hints about them.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Das Smartphone ist aus der heutigen Welt kaum mehr wegzudenken. Smartphones sind nicht nur ein Mittel, um erreichbar zu sein, sondern für viele fast wie ein Atemgerät, von dem sie regelrecht abhängig sind. Zugleich hat man damit viele verschiedene Optionen: Wer Musik hören möchte, kann dies mittels seines Smartphones tun und braucht dafür keinen MP3-Player. Man kann auf dem Smartphone auch Spiele spielen oder damit im Internet recherchieren. Wer z. B. wissen möchte, wann das Geschäft X geöffnet ist, kann dies im Browser seines Smartphones nachschauen. Man kann auch Online-Banking mit dem Smartphone betreiben und mittlerweile von überall aus auf sein Bankkonto zugreifen. Man kann somit praktisch alles, was man zuvor nur mit dem PC oder persönlich vor Ort erledigen konnte, nun auch mit dem Smartphone erledigen. Das Smartphone erleichtert somit in vieler Hinsicht den Alltag.

Es hat gleichwohl nicht nur Vorteile, denn es gibt z. B. Cyberkriminelle, die sich an Smartphones erfreuen, denn sie können diese dank deren Sicherheitslücken angreifen und z. B. Daten vom Smartphone ablesen. Smartphones werden zugleich zunehmend von jungen Menschen im Alter von 8 bis 14 Jahren und älteren Menschen über 60 Jahre benutzt, die sich in der Regel weniger mit der zugehörigen IT auskennen. Deshalb wird in der vorliegenden Bachelorarbeit die Sicherheit mobiler Endgeräte, die auf Android basieren, analysiert und erläutert.

1.2 Ziel der Arbeit

Ziel dieser Bachelorarbeit ist es, eine Sicherheitsanalyse-Applikation für Android zu entwickeln. Diese soll den Benutzern Hinweise auf mögliche Sicherheitslücken in ihrem mobilen Betriebssystem geben. Die Sicherheitslücken werden nicht von dieser Applikation geschlossen, sondern nur die notwendigen von dem Benutzer zu ergreifenden Mittel bzw. Maßnahmen erläutert. Diesbezüglich gibt es zwei verschiedene Prüfungen, die lokale und die globale. Die lokale Prüfung findet ausschließlich mit dem Gerät selbst und die globale Prüfung mit Hil-

fe eines Test-Servers statt. Dieser externe Test-Server wird in einer anderen Bachelorarbeit entwickelt.

1.3 Zielgruppe dieser Arbeit

Die vorliegende Arbeit richtet sich an Studierende im Bereich der Informatik sowie alle anderen Personen, die ein mobiles Endgerät, das auf Android basiert, besitzen und sich für die Sicherheit ihrer Geräte interessieren, eingeschlossen sind Personen, die sich allgemein mit der Sicherheit des Smartphones beschäftigen.

1.4 Struktur der Arbeit

Diese Bachelorarbeit umfasst sieben Kapitel, wovon das erste die Einleitung ist.

Das zweite Kapitel ist zunächst der Sicherheitsarchitektur sowie der Kryptografie, der Datenverschlüsselung und der Anwendungssicherheit von Android gewidmet. Anschließend werden die Sicherheitslücken, die es im Android-Betriebssystem gibt, erörtert. Schließlich wird definiert, was Common Vulnerabilities and Exposures (CVE) und Common Vulnerability Scoring System (CVSS) sind.

Im dritten Kapitel wird auf die Sicherheitsanalyse von Android fokussiert. Dabei werden einige exemplarische Sicherheitsrisiken analysiert, anschließend wird Android mit 2 anderen Smartphone-Betriebssystemen verglichen und 6 bekannte Android- sowie plattformübergreifende Sicherheitslücken werden erläutert.

In Kapitel vier werden die Anforderungen zum Erstellen der im Rahmen der vorliegenden Arbeit entwickelten Android-Anwendung, welche das Android-Smartphone auf dessen Sicherheitslücken hin durchsucht, ermittelt. Daraufhin werden die Systemarchitektur und der Ablauf der Anwendung in einem Sequenzdiagramm dargestellt. Schließlich wird der Entwurf der grafischen Benutzeroberfläche der Anwendung dargestellt.

Im fünften Kapitel wird die Entwicklungsumgebung, in der die Anwendung programmiert wird, erläutert. Anschließend wird die Realisierung der Android-Anwendung beschrieben.

Im sechsten Kapitel wird die Android-Anwendung, die im Rahmen von Kapitel fünf realisiert wurde, anhand von Testfällen geprüft.

Das siebte und letzte Kapitel gibt wieder, was im Rahmen dieser Bachelorarbeit erarbeitet wurde, ob alle Ziele dieser Arbeit erreicht wurden und was im Hinblick auf die Sicherheit rückblickend verbessert werden könnte.

2 Grundlagen

2.1 Sicherheitsarchitektur von Android

Android ist ein Open-Source-Betriebssystem, das auf einem Linux-Kernel basiert, und stellt eine Umgebung zur Verfügung, die es dem Benutzer und Betriebssystem erlaubt, mehrere Anwendungen, z. B. eine Musik-Anwendung und ein einen Browser, gleichzeitig zu betreiben. Diese werden in einer Anwendungs-Sandbox signiert und isoliert. Anwendungs-Sandboxes definieren die verfügbaren Zugriffsrechte auf die jeweilige Anwendung. Anwendungen werden mit Android Runtime erstellt und kommunizieren durch ein Framework, das die Systemdienste beschreibt, mit dem Betriebssystem. Diese Systemdienste sind Application-Programming-Interface's (APIs) und Nachrichtenformate. In Android sind auch andere High-level-Sprachen (z. B. JavaScript) und Low-level-Sprachen (z. B. ARM assembly) erlaubt und diese operieren mit derselben Anwendungs-Sandbox wie die normalen Anwendungen. Android-Systemdienstleistungen werden als Anwendungen implementiert und wie die normalen Anwendungen von einer Anwendungs-Sandbox eingeschränkt.

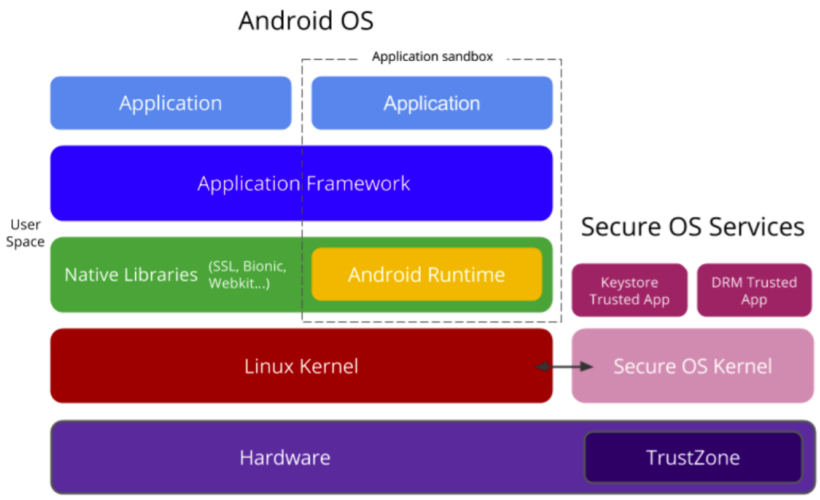

Abbildung 2.1: Android Sicherheitsarchitektur [52, S. 4]

Bis auf ein paar Zeilen vom Android-Betriebssystem-Quellcode, die als Root laufen, wird jeder Quellcode über der Linux-Kernel-Ebene von der Anwendungs-Sandbox beschränkt.

Android ist ein Mehrzweck-Betriebssystem. Android-Geräte bieten eine sekundäre isolierte Umgebung, um privilegierte oder sicherheitskritische Operationen wie das Auslesen von Speichertabellen oder Sicherheitskritische Anwendungen auszuführen, welche die Funktionalität eines Mehrzweck-Betriebssystems nicht brauchen. Diese Umgebung wird manchmal als ein sicherer Betriebssystem (BS) bezeichnet. Diese Funktionen können auf einem separaten Prozessor – wie einem Standalone-Secure-Element oder einem Trusted-Platform-Module (TPM) – realisiert oder unter dem Kernel auf einem gemeinsamen Prozessor (wie ARM TrustZone® Technik) isoliert werden.
Das sichere Betriebssystem kann vom Original-Gerätehersteller (Original Equipment Manufacturer, OEM) eingesetzt werden, um dem Benutzer gerätespezifische Dienste und Anwendungen zur Verfügung zu stellen. Die meisten Android-Geräte implementieren Widevine-DRM-geschützte Video-Wiedergabedienste innerhalb des sicheren Betriebssystems. Beginnend mit Android 4.3, basieren kryptografische Dienste auf den Secure-OS und können auch auf Android-Anwendungen über die Schlüsselbund-API eingesetzt werden. Diese API bietet die Möglichkeit, für Anwendungen Schlüssel zu erschaffen, die selbst im Falle eines Android-Kompromisses (d. h. eines anderen OS) nicht exportiert werden können. [52]

2.2 Kryptografie und Datenverschlüsselung

Kryptografie wird in Android genutzt, um Vertraulichkeit und Integrität in Bezug auf Anwendungen und Benutzerdaten zu gewährleisten.

2.2.1 Geräteverschlüsselung

Ein Android-Gerät verschlüsselt alle vom Benutzer erstellten Daten automatisch, bevor sie auf der Festplatte gespeichert werden insofern, als dieses Gerät selbst verschlüsselt ist. Diese Verschlüsselung ist ein Prozess, bei dem Benutzerdaten mit Hilfe eines verschlüsselten Schlüssels auf das Android-Gerät codiert werden.

Die Android-Festplattenverschlüsselung basiert auf dm-crypt. Dies ist eine Kernel-Funktion, die im „Block-Device-Layer“ arbeitet. Der Hauptschlüssel wird mit 128-Bit-AES über Aufrufe an die Android-OpenSSL-Bibliothek verschlüsselt. Erstausrüster können einen 128-Bit-Key oder höher(256-Bit) verwenden, um die Hauptschlüssel zu verschlüsseln.

Seit Android 5.0 gibt es neue Verschlüsselungsfunktionen wie die schnelle Verschlüsselung, welche das erste Booten des Smartphones beschleunigt, indem es nur die gebrauchten Blöcke auf der Datenpartition verschlüsselt. Außerdem wurde dem Betriebssystem eine Force-Encrypt-Flag hinzugefügt, um beim ersten Booten eine Verschlüsselung zu ermöglichen. Seit Android 5.0 gibt es eine zusätzliche Unterstützung für Muster und eine Verschlüsselung ohne Passwort. Die vier Arten des Verschlüsselungsstatus sind die Standard-, Pin-, Passwort- und Musterverschlüsselung.

Wenn die Standard-Verschlüsselung auf einem Gerät aktiviert ist, dann wird beim ersten Booten ein 128-Bit-Schlüssel erzeugt, der mit einem Standardpasswort verschlüsselt wird. Der verschlüsselte Schlüssel wird in Form von verschlüsselten Metadaten gespeichert. Die Hardware-Unterstützung wird durch die Nutzung der Trusted Execution Environment Unterzeichnungfähigkeit (TEE-Unterzeichnungsfähigkeit) realisiert. Der vom Gerät erzeugte 128-Bit-Schlüssel ist bis zum nächsten Werks-Reset gültig. Also bis die Datenpartition gelöscht worden ist. Nach dem Werks-Reset wird ein neuer 128-Bit-Schlüssel generiert.

Wenn der Benutzer die PIN oder das Passwort neu setzt, wird nur der 128-Bit-Schlüssel neu verschlüsselt und gespeichert, es findet keine erneute Verschlüsselung der Benutzerdaten statt. Das Android-5.0-Compatibility-Definition-Document (CDD) verlangt, dass das Gerät eine Full-Disk-Verschlüsselung der privaten Anwendungsdaten unterstützt, wenn eine Gerätimplementierung eine Display-Sperre hat. [52, 39]

2.2.2 KeyChain und KeyStore

Android stellt den Entwicklern der Anwendungen verschiedene kryptografische APIs, wie die standardisierten und häufig verwendeten Advanced-Encryption-Standard (AES), Rivest-Shamir-Adleman (RSA), Digital-Shamir-Adleman (DSA), Secure-Hash-Algorithm (SHA) und andere zur Verfügung. Zusätzlich werden APIs für übergeordnete Protokolle wie Secure-Socket-Layer (SSL) und Hypertext-Transfer-Protocol (HTTP) bereitgestellt.
Android 4.0 hat am 19. Oktober 2011 die KeyChain-Klasse eingeführt, damit Anwendungen wie der Play Store den Systemberechtigungsspeicher für private Schlüssel und Zertifikatketten mitbenutzen können. Die KeyChain-API wird für Wi-Fi/WLAN/WLAN- und VPN-Zertifikatketten benutzt.

In Android 4.3 wurde am 24. Juli 2013 die KeyStore-Klasse eingeführt. Sinn und Zweck dieser

Klasse ist es, private Schlüssel, die Anwendungen für Anmeldeinformationen speichern, sicherer zu machen, da diese dadurch in einer Art Container gespeichert werden, was es Angreifern erschwert, sie aus dem Gerät zu extrahieren.

Anwendungen können den isBoundKeyAlgorithm in KeyChain aufrufen, bevor sie private Schlüssel eines gegebenen Algorithmus importieren oder generieren, um festzustellen, ob ein Hardware-unterstützter KeyStore unterstützt wird, um Schlüssel in einer Weise an das jeweilige Gerät zu binden, die sie nicht exportierbar macht. Mit Hilfe des isBoundKeyAlgorithm der KeyChain-Klasse können Anwendungen prüfen, ob der Hardware-unterstützte KeyStore unterstützt wird, um festzustellen, ob ein Schlüssel an das Gerät gebunden werden kann, ohne dass es exportierbar ist, um private Schlüssel eines Algorithmus sicher zu importieren oder zu generieren. [52]

2.3 Applikationssicherheit

In den folgenden Abschnitten 2.3.1 bis 2.3.3 werden die wichtigsten Sicherheitsfunktionen der Android-Anwendung definiert. Da es stetig mehr Anwendungen gibt, ist die Anwendungssicherheit für Entwickler und Benutzer immer ein wichtiges Thema. Damit Personen die Anwendungen auf ihrem Smartphone beruhigt nutzen können, bietet Android mehrere Anwendungsschutzschichten, um die Benutzer, so gut es geht, vor Malware, Exploits und anderen Angriffen zu schützen.

2.3.1 Sandbox und Berechtigung

Alle Android-Anwendungen laufen in einer sogenannten Anwendungs-Sandbox. Diese basiert auf jahrzehntelanger Unix-basierter Benutzertrennung von Prozessen sowie Datenberechtigung und umgibt die Anwendung, damit diese nicht ohne Erlaubnis seitens des Betriebssystems aus der Sandbox heraus kommen kann:

> *„Just like the walls of a sandbox keep the sand from getting out, each application is housed within a virtual sandbox to keep the sand from getting out, each application is housed within a virtual sandbox to keep it from accessing anything outside itself.“* [52, S. 6]

Damit eine Anwendung z. B. auf die Kontakte des Smartphones zugreifen kann, müssen dieser bei deren Installation die nötigen Zugriffsrechte dafür gewährt werden. Werden der Anwendung die Zugriffsrechte hingegen nicht gewährt, dann wird diese auch nicht auf dem Gerät

installiert. Ab Android 6.0 ist dies jedoch nicht mehr der Fall, denn ab dieser Version muss der Benutzer der Anwendung nur beim ersten Mal, wo diese die Zugriffsrechte benötigt, ebendiese erteilen. Wenn eine Anwendung auf die Kontakte zugreifen möchte, um das Beispiel wieder aufzunehmen, dann müssen dieser folglich die Zugriffsrechte für die Kontakte nur noch beim ersten Aufrufen der Kontakte gewährt werden. Ab dann hat die Anwendung immer volle Zugriffsrechte auf die Kontakte, wenn man diese Option nicht von Hand entfernt. Die Anwendung kann somit auch installiert werden, obwohl man der Anwendung die Zugriffsrechte auf irgendeine Smartphone-Funktion nicht gewährt. Durch diese Rechte Hinweise kann der Benutzer selbst entscheiden, ob er der Anwendung oder dem Anwendungshersteller vertraut oder nicht. Denn wenn z. B. eine Spieleanwendung auf das Telefon zugreifen möchte, um Anrufe zu tätigen, dann ist dies höchstwahrscheinlich keine seriöse Anwendung und der Benutzer kann und sollte sich dann aus Sicherheitsgründen gegen ebendiese Anwendung entscheiden.
Zur Identifizierung und Isolierung von Anwendungsressourcen nutzt die Android-Plattform die sogenannte „Linux user-based protection“. Das bedeutet, dass jede Anwendung eine eindeutige Benutzer-ID (User-ID, UID) hat und jede Anwendung wie ein separater Benutzer in einem separaten Prozess ausgeführt wird. Hierdurch unterscheidet sich Android hauptsächlich von anderen Betriebssystemen wie iOS und Windows Phone, da bei Android jede Anwendung ihre eigenen Benutzerrechte hat, anders als bei anderen Betriebssystemen wie iOS oder Windows Phone, wo mehrere Anwendungen mit denselben Benutzerrechten ausgeführt werden. Durch die Standard-Linux-Einrichtung wie Benutzer- und Gruppen-IDs wird die Sicherheit auf der Prozessebene zwischen der Anwendung und dem System vom Kernel erzwungen. Standardmäßig können alle Anwendungen bei Android nicht interagieren und haben normalerweise nur begrenzten Zugriff auf das Betriebssystem. Damit die Anwendung A auf die Anwendung B zugreifen kann, benötigt sie die Berechtigung hierzu vom Betriebssystem. Wenn eine Anwendung ohne Berechtigung auf eine andere zugreifen möchte, dann wird das Gerät vom Betriebssystem geschützt, indem es blockiert wird.

Dieses Sicherheitsmodell erstreckt sich auf die native Code- und Betriebssystem-Anwendung (BS-Anwendung), da sich die Anwendungs-Sandbox im Kernel befindet. Wie in Abbildung 2.1 zu sehen, läuft die gesamte Software einschließlich BS-Bibliotheken, Anwendungs-Framework, Android-Anwendungslaufzeit sowie aller Benutzer und vorinstallierter Anwendungen oberhalb der Linux-Kernel-Schicht in der Anwendungs-Sandbox. In Android sind die Entwickler der Applikationen nicht an eine bestimmte Entwicklungsumgebung gebunden, um die Sicherheit

des mobilen Endgerätes zu gewährleisten, so wie dies bei einigen anderen Betriebssystemen wie iOS oder Windows Phone der Fall ist:

> *„On Android, there are no restictions on how an application can be written that are required to enforce security; native code is just as secure as interpreted code.“* [52, S. 7]

Ein Speicherfehler lässt nur eine beliebige Codeausführung mit den vom Betriebssystem vergebenen Benutzerrechten für diese Anwendung zu und beeinträchtigt nicht gleich die Sicherheit des ganzen Gerätes, da alle Anwendungen und deren Ressourcen auf der Betriebssystem-Ebene (BS-Ebene) sandboxiert sind.[52]

2.3.2 Anwendungssignatur

In Android können nur Anwendungen mit einem digitalen Zertifikat installiert werden, dabei muss das Zertifikat nicht einmal von einer Zertifizierungsstelle zertifiziert worden sein, sondern die Anwendungen verwenden oft selbst signierte Zertifikate, von denen der Entwickler einen privaten Schlüssel besitzt. Das Zertifikat kann und wird zur Identifizierung des Entwicklers benutzt. Wenn eine Anwendung ein Update erhält, überprüft das System, ob das Zertifikat des Updates und dasjenige der bereits installierten Anwendung übereinstimmen. Nur wenn die Zertifikate beider Versionen übereinstimmen, wird das jeweilige Update ausgeführt.

Anwendungen, die mit demselben Zertifikat signiert sind, können im selben Prozess ausgeführt werden, wenn dies von der Anwendung so gewünscht wird. Das System sieht diese zwei Anwendungen dann als eine einzige an und führt sie in einem anstelle von zwei Prozessen aus. Dank der Signatur-basierten Berechtigungsdurchsetzung haben Anwendungen die Möglichkeit, ihre Funktionalitäten mit anderen Anwendungen, die mit einem bestimmten Zertifikat signiert sind, bekanntzugeben. Das heißt, dass die Anwendung A auf die Funktionalitäten der Anwendung B zugreifen kann, wenn die Anwendung A mit einem bestimmten Zertifikat signiert worden ist. Durch die Unterzeichnung mehrerer Anwendungen mit demselben Zertifikat und der Signatur-basierten Berechtigung kann eine Anwendung Code und Daten auf sichere Art und Weise freigeben.

Die Gültigkeitsdauer des privaten Schlüssels sollte möglichst lang sein, da eine Anwendung nicht mehr aktualisiert werden kann, sobald deren Gültigkeit abgelaufen ist, wodurch die Anwendung dann komplett neu installiert werden müsste. Android empfiehlt hier eine Min-

destlebensdauer von 25 Jahren, da Google nicht davon ausgeht, dass eine Anwendung eine so lange Lebensdauer haben wird.[52]

2.3.3 Security Enhanced Linux

Security Enhanced Linux (SELinux) wird in der Android-Sandbox verwendet, um für alle Anwendungen, die mit Root- und Superuser-Rechten ausgeführt werden, eine obligatorische Zugriffskontrolle (Mandatory-Access-Control, MAC) zu erzwingen. SELinux trennt diese Prozesse, wegen der zentralisierten und analysierbaren Politik, die SELinux anbietet, strikt voneinander.

Da Android SELinux im „Erzwingmodus“ enthält, werden Sicherheitsrichtlinien protokolliert und für die Anwendungen erzwungen. Wenn eine Anwendung eine ungesetzliche Handlung und oder eine Verletzung bzw. Verweigerung, die gegen diese Politik verstößt, ausführen möchte, wird diese verhindert und vom Kernel zu dmesg und logcat protokolliert.

Seit Android 5.0 müssen Geräte wegen des Compatibility-Definition-Documents (CDD) eine SELinux-Richtlinie implementieren, um SELinux auf einer Domänen-Basis festzulegen und damit alle Domänen im Durchführungsmodus konfiguriert werden. Die Android-5.0-CDD erfordert, dass Geräte eine SELinux-Richtlinie implementieren, welche es ermöglicht, dass der SELinux-Modus auf einer Domänen-Basis festgelegt wird und alle Domänen im Durchführungsmodus konfiguriert sind. Dabei sind keine zulässigen Modusdomänen erlaubt:[52, 41]

> *„Die Kompatibilitäts-Test-Suite (CTS) für SELinux sorgt für sicherheitspolitische Kompatibilität und erzwingt Sicherheits-Best Practices.“* [52]

2.4 Sicherheitslücken

Sicherheit ist in der IT-Branche ein bedeutendes Thema, denn heutzutage gibt es sehr viele Schadprogramme bzw. Malware wie Viren, Trojaner, Würmer und viele andere. Diese Sicherheitslücken können den Benutzer entweder „nur“ stören oder aber großen Schaden anrichten. Diese können nämlich die Sicherheit des Systems umgehen, sich als „gutartiges“ Goodwill-Programm ausgeben oder das System selbst modifizieren und ausspionieren. Nachfolgend werden sie einzeln vorgestellt.[16]

2.4.1 Malware

Als Malware werden Programme von Angreifern bezeichnet, die absichtlich schädliche oder unerwünschte Funktionen ausführen, um einem System Schaden zuzufügen. Dies erreicht der Angreifer, indem er Sicherheitslücken im Betriebssystem, in einer Anwendersoftware, einem Treiber oder einem anderen Programmcode ausnutzt. Malware (bzw. deren Benutzer) hat unterschiedliche Ziele, da sie meistens von verschiedenen Angreifern programmiert wird. Das Internet ist zwar die von Hacker am häufigsten genutzte Infektionsquelle, aber nicht die einzige, denn der schädliche Code kann durch die verschiedensten Datenträger (z. B. USB-Stick oder CD) übertragen werden. Die Malware versucht meistens, sich zu verbreiten, ohne dass der jeweilige Benutzer es merkt, um im Hintergrund das System zu beeinflussen. [8, 10]

2.4.2 Viren und Würmer

Viren können sich nicht selbst aktivieren, sie werden erst dann aktiviert, wenn der Anwender das infizierte Programm startet. Sobald dies geschieht, erfolgt erst ein Sprung auf den Viruscode und dann ein Rücksprung auf den Programmcode, anstatt so, wie es vorgesehen ist, ein direkter Sprung auf den Programmcode. So infiziert der schadhafte Code im Programm den das entsprechende Gerät. Dann hat das Virus die verschiedensten Optionen: Es kann Daten auslesen, löschen oder sogar das System und Datenträger vernichten.

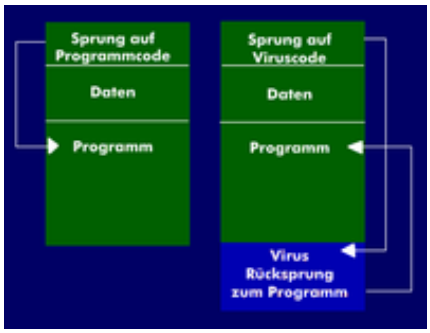

Abbildung 2.2: Programmablauf mit und ohne Virus [26]

Viren können sich folglich weder selbstständig aktivieren noch von allein andere Geräte infizieren, ohne dass irgendjemand die infizierte Datei verschickt und aktiviert. Würmer hingegen können sich von allein aktivieren und vermehren. Sie können z. B. die Kontaktliste des infizierten Gerätes benutzen, um sich selbst an alle Kontakte des Infizierten weiterzuversenden. Würmer müssen keinen fremden Code infizieren, um sich zu vermehren, können jedoch den gleichen Schaden wie Viren verursachen.

[26] [27]

2.4.3 Keylogger

Keylogger ist eine Spyware und Unterkategorie von Malware, die es dem Angreifer ermöglicht, alle Tastatureingaben des jeweiligen Benutzers zu erhalten, ohne dass dieser es bemerkt. Dies schafft der Keylogger, indem er sich wie beim sogenannten Man-in-the-Middle-Angriff quasi zwischen die Kommunikation der beiden Komponenten, Tastatur und Betriebssystem, stellt. Bei Android wäre dies zwischen dem Touch-screen und dem Betriebssystem. In diesem Fall wäre es z. B. die Swift-Key-App, die vom Angreifer so manipuliert werden kann, dass sie alle Daten, die von dem Benutzer eingetippt werden, direkt dem Angreifer schickt, so wie es der irische Entwickler Georgie Casey auf seiner Website http://www.georgiecasey.com/2013/03/06/inserting-keylogger-code-in-android-swiftkey-using-apktool/ zeigt, um vor Piraterie zu warnen.[29, 33, 6]

2.4.4 Trojaner

Der Begriff Trojaner stammt von dem hölzernen Trojanischen Pferd, welches im Jahre 600 v. Chr. vor den Toren Trojas stand. Die Griechen benutzten es, um nach Troja zu gelangen, denn im Inneren des hölzernen Pferdes waren griechische Soldaten. Das, was von Troja für ein Geschenk gehalten wurde, war in Wahrheit ein Weg, um nach Troja zu gelangen. Das gleiche Prinzip wird bei dieser Malware benutzt: Programme, die sich als scheinbar nützlich darstellen, sind in Wahrheit Trojanische Pferde, die den das Gerät infizieren. Diese können sich jedoch nicht selbstständig verbreiten, sondern sind nur in Applikationen versteckt. Das infizierte Gerät wird dann vom Angreifer ausspioniert, ohne dass der Anwender dies feststellt.[5, 20, 50]

2.4.5 Schwachstellen

Im folgenden Diagramm in der Abbildung 2.3 sind die Arten und die jeweilige Zahl der Schwachstellen, die es bei Android gibt (Stand:12.02.2017), zu sehen. Nachfolgend wird auf die vier häufigsten davon eingegangen.

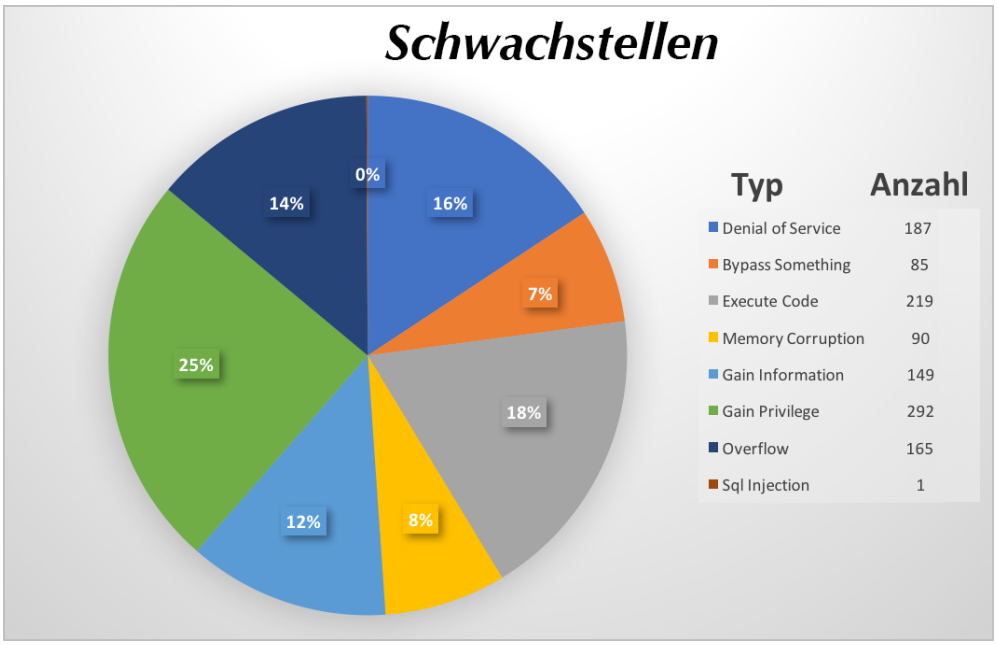

Abbildung 2.3: Schwachstellen in Android (2009-2017) [9]

Denial of Service (DoS)

DoS-Attacken haben meist das Ziel, einen Dienst wie z.B. Webserver einzuschränken, wenn nicht sogar komplett auszuschalten. Dies schaffen die Angreifer, indem sie den Server mit unnötigen Anfragen überlasten. Das können ganz normale E-Mails sein, indem der Angreifer Millionen dieser an einen Server schickt und ihn dadurch überlastet oder indem eine Seite Millionen Mal aufgerufen wird. Dadurch bekommt der Server Unmengen an Datenpaketen, die er nicht verarbeiten kann, und bricht zusammen, da er hierfür nicht genügend Ressourcen zur Verfügung hat. Deswegen kann auf den Server vorübergehend nur eingeschränkt oder gar nicht zugegriffen werden.[19]

Execute Code

Durch eine Sicherheitslücke kann der Angreifer mittels der Code-Execution die Kontrolle über ein Gerät übernehmen. Der Angreifer kann damit einen beliebigen Maschinen-Code auf dem Gerät ausführen, das er „eingenommen“ hat. Dies erlaubt dem Angreifer die vollständige Kontrolle über das Gerät, ohne dass der Benutzer dies bemerkt. Da der Angreifer damit aus der Ferne den Maschinen- und auch Shell-Code einschleusen und ausführen lassen kann, wird dies „Arbitary Code-Execution“ genannt. Die Arbitary Code-Execution wird auch oft benutzt, um vom übernommenen Gerät aus Spam-E-Mails zu verschicken oder auch um, wie in Abschnitt 2.4.5 Schwachstellen (Denial of Services (DoS)) beschrieben, DoS-Attacken zu verrichten.[28, 43]

Gain Privilege

Das Gain-Privilege, auch Rechteausweitung (Privilege-Escalation) genannt, ist eine Sicherheitslücke, dank deren der Angreifer auf Daten oder Programme zugreifen kann, für die er normalerweise keine Zugriffsberechtigung hat. Es gibt zwei Arten: die vertikale und die horizontale Rechteausweitung Bei der vertikalen Rechteausweitung (Vertical Privilege-Escalation) loggt sich der Angreifer z. B. in ein System ein und sucht dort nach einer Sicherheitslücke, die es ihm ermöglicht, seine Rechte auszuweiten, damit er auf alles zugreifen kann oder zumindest auf das, was er möchte. Bei der horizontalen Rechteausweitung (horizontal privilege escalation) erhält der Angreifer keine erhöhte Zugangsberechtigung, sondern gibt sich als jemand anderes aus, der diejenigen Zugriffsrechte hat, die er benötigt. Die Sicherheitslücke Dirty COW nutzt die vertikale Rechteausweitungs-Schwachstelle aus, um sich erweiterte Rechte (z. B. Root-Rechte) zu sichern. Diese Sicherheitslücke wird auch von vielen Android-Benutzern ausgenutzt, um ihr Android-Smartphone zu rooten. [22, 31, 32]

Overflow

Wie in Abbildung 2.3 zu sehen ist, ist der Überlauf (Overflow) eine der von den Angreifern meistgenutzten Schwachstellen bei Android. Ein Überlauf tritt dann ein, wenn von der Schadsoftware die Grenzen von Bereichen wie Speicherbereiche überschritten werden. Beim Buffer-Overflow sind die Speicherbereiche, die vom Betriebssystem vergeben wurden, diese entsprechenden Grenzen, die viele Angreifer zum Überlauf zu bringen versuchen. Dadurch werden andere Speicherbereiche mit einem schadhaften Code platziert und es werden falsche Rücksprungadressen in diese Speicheradressen kopiert, die dann auf die schadhafte Software

hinüberspringen, oder das Programm wird von dem Schadhaften Code so stark mit Müll (irgend welche Strings) überhäuft, dass es abstürzt. [37, 18, 21]

2.4.6 CVE und CVSS

Common Vulnerabilities and Exposures (CVE) vereinfachen den Informationsaustausch zwischen separaten Netzwerksicherheitsdatenbanken dank der standardisierten und einheitlichen Kennzeichnung von Sicherheitslücken und Schwachstellen in Systemen. Diese Kennzeichnung (CVE-ID) setzt sich aus einer fortlaufenden Nummer und dem Jahr, in dem die jeweilige Schwachstelle gefunden wurde, zusammen. [24]

Das Common Vulnerability Scoring System (CVSS) ist ein Standard, der von FIRST.Org dazu genutzt wird, um für den Benutzer die Gefährlichkeit der jeweiligen Schwachstelle mit Hilfe einer Punkteskala besser darzustellen. Diese Punkteskala reicht von 0 bis 10, wobei 10 die höchste Gefahrenstufe darstellt. [23]

3 Sicherheitsanalyse

In Kapitel 2 (Grundlagen) wurden die Sicherheitsarchitektur sowie die Kryptografie, Datenverschlüsselung und Applikationssicherheit von Android erläutert. Des Weiteren wurden dort einige wichtige Sicherheitslücken beschrieben.
Im folgenden Abschnitt werden die Sicherheitsrisiken von Android analysiert. Darauffolgend werden Sicherheitslücken, die nur auf Android zutreffen, erläutert und anschließend plattformübergreifende Sicherheitslücken.

3.1 Sicherheitsrisiken

In diesem Abschnitt werden zwei wichtige Sicherheitsrisiken bei Android analysiert und beschrieben.

3.1.1 Herstellerspezifisches Android Systemupdates

Weil jeder Hersteller, der Smartphones mit Android als Betriebssystem verkauft, die jeweils aktuelle Android-Version an seine Hardware anpasst, muss dieser zugleich für die entsprechenden Updates sorgen, wodurch es zu Verzögerungen bei der Verfügbarkeit kommt. Dies führt dazu, dass manche Sicherheitslücken bei einigen Herstellern länger bestehen als bei anderen, da sie möglicherweise länger brauchen, um das neue Update an ihre Hardware anzupassen. Deshalb ist ein von Google entwickeltes Handy kurzfristig sicherer als eins anderer Smartphone-Hersteller, da Google-Smartphones normalerweise immer als Erstes ein Update erhält, da diese bereits an ihre Hardware angepasst sind. Bei anderen Betriebssystemen wie IOS ist dies nicht der Fall, da die Update-Versorgung aus einer zentralen Quelle stammt und nicht extra an jedes Gerät eines anderen Herstellers angepasst werden muss. [38, 16]

3.1.2 Root

Das Rooten von Android-Handys bringt Vor- und Nachteile mit sich. Der Vorteil des Rootens ist, dass der Benutzer sein Betriebssystem dadurch besser an seine persönlichen Bedürfnisse anpassen kann, sofern er sich damit auskennt oder sich punktgenau an entsprechende Anleitungen aus dem Internet hält. Die meisten Smartphone-Hersteller geben dann jedoch keine Garantie mehr oder nur noch eine Teil-Garantie auf das Gerät. Durch das Rooten bekommen die vom Benutzer ausgewählten Apps Rootberechtigungen. So kann der Benutzer dann z. B. einige Apps deinstallieren, die er zuvor nicht deinstallieren konnte, wie es z. B. bei Facebook oft der Fall ist. Das Rooten bringt aber auch Nachteile mit sich, die nicht ungefährlich sind, denn durch das Rooten kann auch Schadsoftware Rootrechte auf die Systemanwendungen bekommen, die sie vorher nicht hatte. Wenn man z. B. einer schadhaften Software aus Versehen Rootrechte erteilt, dann kann diese das ganze System verändern, wenn nicht zerstören, da die Schadsoftware dann ggf. auf alle Dateien und Anwendungen zugreifen kann. Dadurch können immense Kosten auf den Smartphone-Besitzer kommen, da manche Schadsoftware damit kostenpflichtige Nummern anrufen oder Abonnements abschließen kann, ohne dass es der Benutzer merkt. Dies ist zwar nicht nur bei Schadsoftware, die Rootrechte erhalten hat, der Fall, aber hierbei ist es umso gefährlicher, da das Infizieren der Schadsoftware hier einfacher ist. Damit so etwas nicht passiert, ist es bei Android standardmäßig so, dass nur der Kernel komplette Rootrechte hat. Außer dem Kernel haben nur wenige Hauptprogramme Rootrechte, sofern das Smartphone (noch) nicht gerootet worden ist.[**7**, 4, 7, 40, 48]

3.2 Mobile Betriebssysteme Vergleich

Wie man in der Tabelle 3.1 sieht, ist Android aktuell das meistverkaufte und meistbenutzte Smartphone-Betriebssystem. Man erkennt anhand der Tabelle 3.2 aber auch, dass es zugleich das Smartphone-Betriebssystem mit den meisten Sicherheitslücken ist. Zusätzlich gilt Folgendes: Es ist sogar von allen möglichen Betriebssystemen dasjenige mit den meisten Sicherheitslücken. Da Android das meistbenutzte Smartphone-Betriebssystem ist, ist es auch das beliebteste Betriebssystem für Angreifer, da sie bei einem Android-Smartphone viel mehr Hardwaregeräte angreifen können. So können sie z. B. viel größere Botnetze aufbauen, als sie es z. B. beim Windows Phone können.

Zeit	Android	iOS	Windows Phone
2015 Q4	79,6 %	18,7 %	1,2 %
2016 Q1	83,5 %	15,4 %	0,8 %
2016 Q2	87,6 %	11,7 %	0,4 %
2016 Q3	86,8 %	12,5 %	0,3 %

Tabelle 3.1: Weltweiter Smartphone BS Marktanteil [25]

Betriebssystem	Anzahl Sicherheitslücken
Android	523
IOS	161
Windows 10 Mobile	172

Tabelle 3.2: Weltweiter Smartphone Sicherheitslückenanzahl im Jahr 2016 [12]

3.2.1 Android Sicherheitslücken

Wie in Abschnitt 3.2 dargestellt, gibt es stetig mehr Sicherheitsprobleme in der mobilen Welt, Android bleibt davon nicht verschont. In der Tabelle 3.3 sind einige der meistverbreiteten und gefährlichsten Android-Sicherheitslücken zu erkennen, da sie in den meistbenutzten Android-Versionen 4.1 bis 6 zu finden sind.

Name / betroffene Komponente	**CVE-ID**	**Betroffene iOS-Versionen**
Hummingbad	unbekannt	4.1 bis einschließlich Android 5.0
Dirty Cow	CVE-2016-5195	alle bekannten Android Versionen
QuadRooter	CVE-2016-2503 CVE-2016-2504 CVE-2016-2059 CVE-2016-5340	nahezu allen Android Smartphones mit Qualcomm-Chips
Stagefright	CVE-2015-1538 CVE-2015-1539 CVE-2015-3824 CVE-2015-3826 CVE-2015-3827 CVE-2015-3828 CVE-2015-3829	2.2 bis einschließlich 5.1.1

Tabelle 3.3: Sicherheitslücken in Android

Hummingbad

HummingBad ist eine Android-Sicherheitslücke, die vermehrt in den Android-Versionen 4.1 bis 4.4 vorzufinden ist. Sie wird zusätzlich, wenngleich seltener, in den Android-Versionen 5.0 gefunden. Ob ein Android-Gerät von dieser Sicherheitslücke betroffen ist, wird nicht so leicht erkannt, da die schädigenden Komponenten hierbei allesamt verschlüsselt sind, was es erschwert, den Schadcode zu erkennen. Allein in Deutschland wurden 40.000 Geräte infiziert, weltweit sogar 85 Millionen Android-Geräte. HummingBad wurde ausgehend von Porno-Webseiten per Drive-by-Download verbreitet oder auch von infizierten Anwendungen aus dem Google Playstore. Es verschafft sich höhere Nutzungsrechte, um das Smartphone mit Hilfe verschiedenster Exploits zu entsperren. HummingBad installiert dann beliebige Anwendungen, falls es das Smartphone erfolgreich rooten kann. Dann kann es infizierte Geräte zu einem

Botnetz zusammenschließen. In der Abbildung 3.1 ist das Programmablaufplan (PAP) von HummingBad zu erkennen sowie dass HummingBad den Smartphone gleich zweimal angreift, um einen erfolgreichen Angriff zu gewährleisten.

Die erste Attacke erfolgt immer nach der Ausführung eines der folgenden drei Events: Das erste Event „BOOT_COMPLETED“ ist das Booten des Gerätes. Wenn das Gerät eingeschaltet wird, wird bereits ein Angriff durchgeführt. Das zweite ist das Event „TIME_TICK“. Bei diesem Event wird immer dann ein Angriff durchgeführt, wenn eine Minute vergangen ist. Das dritte ist das Event „SCREEN_ON“. Hier wird immer dann ein Angriff durchgeführt, wenn der Bildschirm angemacht wird. Wenn eines dieser drei Events ausgelöst wird, wird von HummingBad geprüft, ob das Smartphone gerootet ist. Wenn dies zutrifft, wird das Smartphone mit dem Server des Angreifers verbunden, um Schadsoftware herunterzuladen und zu installieren. Wenn das Smartphone nicht gerootet ist, wird die „right_core.apk“ mit XOR entschlüsselt. Danach wird eine native Bibliothek aus der „support.bmp“-Datei mit Hilfe der „right_core.apk“ entschlüsselt. Danach werden mehrere Exploits mit der nativen Bibliothek gestartet. Dann wird versucht, einen Root-Zugriff zu erhalten, indem der Schadhafte Code mit Hilfe der Exploits die Privilegien eskalieren lässt. Wenn das Smartphone jetzt erfolgreich gerootet wurde, dann versucht sich das Smartphone mit dem Server zu verbinden, um Schadsoftware herunterzuladen.

Die zweite Attacke namens „qs“ nutzt Social Engineering, um ihr Ziel zu erreichen, und wird nur dann ausgeführt, wenn die erste Attacke das Handy noch nicht erfolgreich rooten konnte. Der Benutzer wird mit einem Fake-System-Update hereingelegt. Wenn der Benutzer diesem zustimmt, dann wird „module_encrypted.jar“ entschlüsselt und HummingBad hat fortan dieselbe Macht über das Smartphone, als wenn die erste Attacke gelungen wäre. Dann wird, wie bei der ersten Attacke, das Smartphone mit dem Server verbunden, um Schadsoftware herunterzuladen.[34, 35, 30]

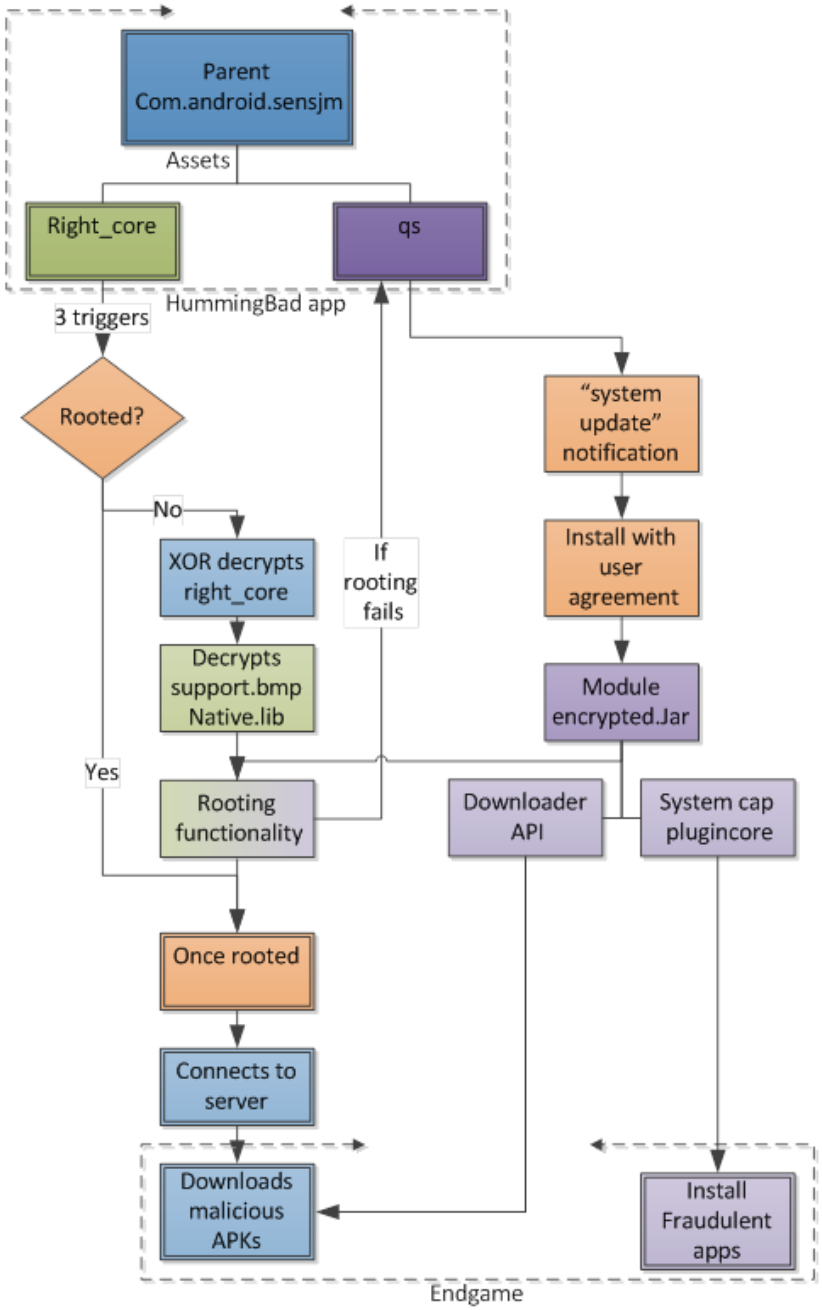

Abbildung 3.1: Hummingbad PAP [34]

Dirty COW

Die in Kapitel 2.4.5 kurz erwähnte Sicherheitslücke Dirty COW hat die Nummer CVE-2016-5195 und trägt den Namen Dirty COW wegen eines Fehlers in der Kernel-Funktion Copy-on-Write (COW). Es ist eine bei allen Android-Geräten anfällige Sicherheitslücke, da diese seit der Linux-Kernel-Version 2.6.22 besteht und Android selbst auf der Linux-Kernel-Version 2.6.25 aufbaut. Dirty COW erlaubt es normalen Prozessen, ohne Rootrechte Speicherbereiche zu verändern, die normalerweise nur von Prozessen mit Rootrechten verändert werden können. Wenn diese Prozesse Zugriff auf den Speicherbereich bekommen und die veränderten Daten ausführen, dann können sie das komplette Smartphone übernehmen. Mit dem auf diese Weise übernommenen Smartphone können die Angreifer dann alles machen, was der Benutzer selbst auch damit tun könnte. [13, 45]

QuadRooter

QuadRooter nutzt eine von den Angreifer präparierte App aus, um den Rootzugriff zu erhalten. QuadRooter besteht aus insgesamt vier Sicherheitslücken und waren insgesamt mehr als 900 Millionen Geräte weltweit davon betroffen. Die genannten Sicherheitslücken sind wegen des Qualcomm-Chips, der für den Long Term Evolution-Empfang (LTE-Empfang) zuständig ist, auf dem Smartphone aufzufinden. Die Sicherheitslücke wurde von Qualcomm behoben und muss dann an alle Hardware-Hersteller verteilt werden, damit diese die entsprechenden Sicherheitslücken in ihren Geräten schließen können.
Die Sicherheitslücken mit den CVE-Nummern CVE-2016-2503 und CVE-2016-2504 ermöglichen es Angreifern, über eine präparierte App den Rootzugriff auf einen Nexus 5, 5x, 6, 6P und 7 zu erhalten.[53, 2, 17, 36]

Stagefright

95 Prozent aller Geräte, welche die Android-Versionen 2.3 bis 5.1.1 haben, sind von der Sicherheitslücke Stagefright betroffen. Stagefright wird seit der Android-Version 2.3 als Standardbibliothek zum Verarbeiten von Multimediadateien genutzt. Seit Android 4.0 ist das Ausnutzen dieser Sicherheitslücke durch die Schutzfunktion Address-Space-Layout-Randomization (ASLR) erschwert worden.

Eine präparierte Multimediadatei wie eine MP4-Videodatei kann das Stagefright-Framework zum Absturz bringen. Diese MP4-Datei kann von einer MMS oder Hangouts verschickt werden und wird normalerweise gleich beim Downloaden der Datei ausgeführt. Deshalb sollte

man sicherheitshalber die automatische MMS-Download-Funktion ausschalten. Man kann die präparierte MP4-Datei jedoch nicht nur von durch MMS oder Hangouts „einfangen“, sondern auch über einen Messenger, eine App, eine E-Mail, Bluetooth, vcard, sd-Karte, nfc oder eine präparierte Website. Stagefright nutzt die Overflow-Schwachstelle, die in Abschnitt 2.4.5 (Schwachstellen) im Grundlagenteil erläutert wurde. Der Angreifer kann nach erfolgreichem Ausführen des Schadcodes das Handy als Abhörgerät missbrauchen und die präparierte MP4-Datei an alle Kontakte verschicken, die der betreffende Benutzer hat. [49, 3]

3.2.2 Plattformübergreifende Sicherheitslücken

In der Tabelle 3.4 sind zwei Sicherheitslücken zu sehen, die es sowohl bei Android als auch den Betriebssystemen Windows Phone und IOS gibt.

Name	**Android**	**iOS**	**Windows Phone**	**CVE-ID**
Freak-Attacken	✓	✓	✓	CVE-2015-0204 CVE-2015-1637 CVE-2015-1067
Adobe Flash / Air	✓	✓	✓	CVE-2015-7663 CVE-2015-8042 CVE-2015-8043 CVE-2015-8044 CVE-2015-8046

Tabelle 3.4: Plattformübergreifende Sicherheitslücken in Mobilen Betriebssystemen

Adobe Flash/Air

Adobe Flash/Air ist auf allen Betriebssystemen verfügbar, deshalb ist es bei Angreifern beliebt, da hierbei eine Sicherheitslücke gleich für alle Betriebssysteme offen ist. So können Angreifer gleich mehrere Betriebssysteme angreifen und z. B. nicht nur Android. Auch Exploit-Kits nutzen gern die Schwachstellen von Adobe Flash. Viele Webserver wie z.B. Youtube setzen hingegen auf neue Webtechnologien wie HTML 5, da für Adobe Flash/Air ständig neue Sicherheitslücken auftauchen und es deshalb langsam ausstirbt. [11]

FREAK-Attack

FREAK-Attack nutzt eine kryptografische Sicherheitslücke im SSL/TLS-Protokoll. Eine FREAK-Attacke ist dann möglich, wenn ein anfälliger Browser eine Verbindung zu einem anfälligen Webserver herstellt. Anfällig sind Server, welche eine Export-Grade-Verschlüsselung akzeptieren. Die Angreifer fangen HTTPS-Verbindungen zwischen Server und Client ab und zwingen diese dann dazu, eine schwächere Verschlüsselung zu verwenden, die leicht entschlüsselt werden kann. Der Angreifer kann dann Daten stehlen oder manipulieren. Smartphones, die Anwendungen nutzen, die ungepatchte TLS-Bibliotheken betreiben oder RSA_EXPORT-Chiffre-Suiten anbieten, sind durch FREAK-Attacken gefährdet. Man sollte immer darauf achten, Anwendungen mit den neuesten Bibliotheken zu benutzen. Auch beim Browser sollte man darauf achten, dass dieser immer mit den neuesten Plug-in-Versionen betrieben wird. Auch wenn Browser wie Firefox dafür bekannt sind, dass sie sicher sind, können Schwachstellen durch Drittanbieter-Plug-ins oder Antiviren-Plug-ins für FREAK-Attacken geöffnet werden.[42]

4 Konzept

4.1 Anforderungsanalyse

Im 2. Kapitel wurde sowohl auf den Aufbau der Sicherheitsarchitektur von Android eingegangen als auch auf die Gefahren von Schwachstellen hingewiesen. Im 3. Kapitel wurden einige relevante aktuelle Schwachstellen von Android und anderen mobilen Betriebssystemen dargestellt. Im Hinblick auf den höheren Marktanteil (Tab. 3.1) von Android (86,8 %) im Vergleich zu dem von iOS (12,5 %) und Windows Phone (0,3 %, Q3 2016) sowie die Zahl der Sicherheitslücken weltweit im Jahr 2016 ist ersichtlich, dass Android zwar mehr Sicherheitslücken als iOS und Windows Phone aufweist, aber auch, dass die anderen Betriebssysteme nicht sehr viel sicherer sind. Der Benutzer selbst sollte daher aktiv zur Sicherheit beitragen und sich nicht blind auf die entsprechenden Maßnahmen der Smartphone-Hersteller verlassen.
Auf Basis der gesammelten Informationen aus dem 3. Kapitel wird im Rahmen der vorliegenden Arbeit eine Anwendung konstruiert, die den Android-Benutzer vor den bei seinem Smartphone (potenziell) bestehenden und bekannten Sicherheitslücken warnt und, wenn möglich, zeigt, wie er sich davor schützen kann. Die Anwendung zeigt jedoch nur die Sicherheitslücken und entfernt diese nicht.

Die Anwendung bietet zwei verschiedene Überprüfungen: Die lokale Überprüfung testet das Smartphone mit Hilfe einer kleinen, bereits in die Anwendung integrierten Datenbank. Die serverseitige Überprüfung erfolgt mit Hilfe eines Servers, der im Zuge einer anderen Bachelorarbeit von einem Kommilitonen programmiert wurde. Bei dieser serverseitigen Überprüfung verbindet sich die Anwendung mit dem genannten Server und überprüft das Smartphone dann mit Hilfe der von dem Server ständig aktualisierten Datenbank. In beiden Fällen werden dann die Sicherheitslücken (Common Vulnerabilities and Exposures, CVEs) in einer Liste aufgezeigt. Klickt der User dann auf eine dieser Sicherheitslücken, so sieht er, wenn dies möglich ist, wie er in Bezug auf diese Sicherheitslücke vorgehen kann, um sein Smartphone zu schützen.

4.1.1 Funktionale Anforderungen

Um die geplanten Ziele der zu Entwickelnden mobilen Applikation verfolgen zu können werden nachfolgend die Funktionalen und nicht Funktionalen Anforderungen erläutert.

FA-1 Android-Version des Gerätes muss ermittelt werden

FA-2 Lokale System-überprüfung mit der von der Applikation ermittelten Android-Version durchführen.
FA-2.1 Zuordnung der Sicherheitslücken zur im Gerät benutzten Android-Version
FA-2.2 Sicherheitslücken des betroffenen mobilen Endgerät anzeigen

FA-3 Detaillierte Ansicht der Sicherheitslücken aus der Lokalen Überprüfung
FA-3.1 Informationen der CVE-ID, Sicherheitslückentyp, Sicherheitslücken beschreibung und Risiko Einstufung anzeigen

FA-4 Serverseitige Überprüfung mit der von der Applikation ermittelten Android-Version durchführen
FA-4.1 Verbindung zum Server Herstellen
FA-4.2 Anfrage an den Server senden mit der Android-Version des Gerätes
FA-4.3 Serverantwort auswerten
FA-4.4 Sicherheitslücken des betroffenen mobilen Endgerät anzeigen

FA-5 Detaillierte Ansicht der Sicherheitslücken aus der serverseitigen Überprüfung
FA-5.1 Informationen der CVE-ID, Sicherheitslückentyp, Sicherheitslücken beschreibung und Risiko Einstufung anzeigen

4.1.2 Nicht Funktionale Anforderungen

NFA-1 Applikation sollte auf den Android-Versionen 4.1 bis 6 laufen
NFA-1.1 Applikation muss eine Korrekte zu der Android-Version passende Schwachstelle anzeigen

NFA-2 Applikation sollte Benutzerfreundlich sein

NFA-2.2 Informationen der CVE-ID, Sicherheitslückentyp, Sicherheitslücken beschreibung und Risiko Einstufung müssen gut lesbar sein

NFA-3 Lokale Überprüfung sollte stets erfolgreich sein
NFA-3.1 Lokale Überprüfung sollte innerhalb kürzester Zeit erfolgen

NFA-4 Server sollte jederzeit erreichbar sein muss aber nicht
NFA-4.1 Datenbank des Servers sollte jederzeit erreichbar sein muss aber nicht
NFA-4.2 Serverseitige Überprüfung sollte innerhalb kürzester Zeit erfolgen

4.1.3 Kontextabgrenzung

Die in den Kapiteln 4.1.2 und 4.1.3 dargestellten Anforderungen decken nur den Funktionalitätsumfang dieser Anwendung ab. Die Anwendung wird vom Autor dieser Arbeit entwickelt, dieser ist zugleich weder für die Anforderungen noch die Entwicklung des Servers zuständig. Die Funktionen, die vom Server bereitgestellt werden, sind dem Autor, wie beschrieben, von einem Kommilitonen und Verfasser einer anderen Bachelorarbeit zur Verfügung gestellt worden und daher nicht Teil der vorliegenden Bachelorarbeit.

4.1.4 Anwendungsfälle

Die Funktionalität der Anwendung wird anhand eines Unified Modeling Language-Diagramms (UML-Diagramms) dargestellt. Das Diagramm in der Abbildung 4.1 stellt ein Anwendungsfalldiagramm dar, welches alle möglichen Szenarien zwischen dem Akteur und dem System zum Erreichen des Ziels aufzeigt. Dieses Diagramm verdeutlicht die Beziehung zwischen dem Benutzer des mobilen Endgerätes (Akteur) und der Anwendung (System) auf dem mobilen Endgerät. Dieses Diagramm dient jedoch lediglich der Verdeutlichung und beschreibt weder das Verhalten noch das Systemdesign der Anwendung.

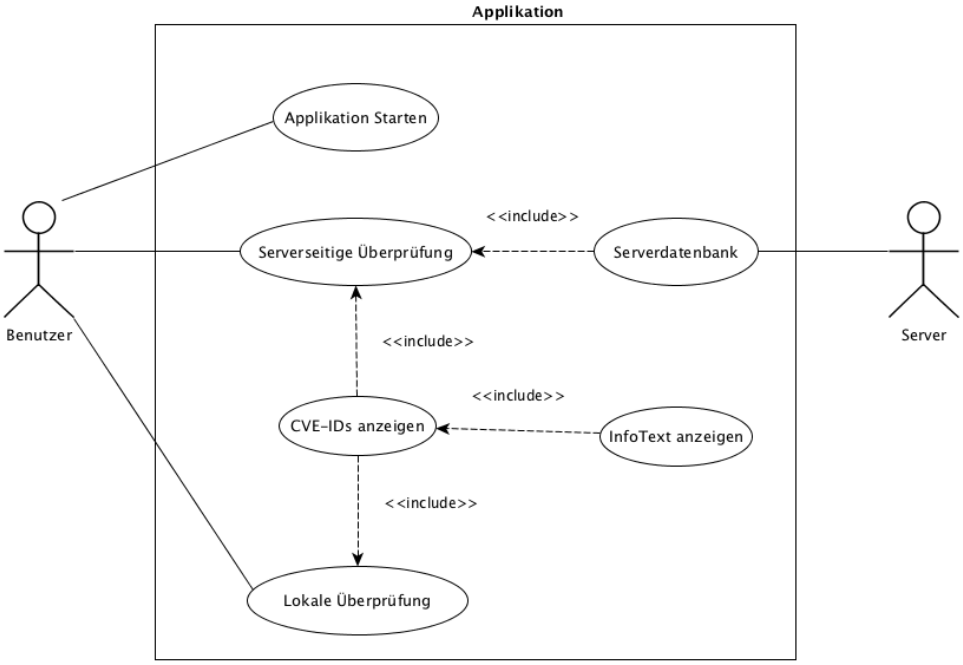

Abbildung 4.1: Anwendungsfalldiagramm für die Anforderungen

Titel	Applikation Starten
Anwendungsfall - Nr.	1
Akteur	Benutzer, Applikation
Ziel	Starten der Applikation.
Auslöser	Benutzer drückt auf die Applikation.
Vorbedingung	Benutzer hat die Applikation ist Installiert.
Nachbedingung	Der Benutzer kann eine Überprüfung auswählen
Ablaufschritte	1.) Der Benutzer startet die Applikation indem er auf das Applikations-Icon auf seinem Smartphone drückt. 2.) Es wird auf eine Überprüfungswahl gewartet.

Tabelle 4.1: Anwendungsfall von Applikation Starten

Titel	Lokale Überprüfung
Anwendungsfall - Nr.	2
Akteur	Benutzer, Applikation
Ziel	Durchführung der Lokalen Überprüfung.
Auslöser	Der Benutzer drückt auf den Button „Lokal“.
Vorbedingung	Anwendung ist an.
Nachbedingung	Lokale Überprüfung wurde ausgeführt und die Liste der CVEs werden dem Benutzer angezeigt.
Ablaufschritte	1.) Der Benutzer drückt den “Lokale” Button und startet damit die Lokale Überprüfung. 2.) Es werden die CVEs angezeigt. 3.) Der Benutzer wählt eine CVE aus. 4.) Der Benutzer bekommt eine Beschreibung und einen Hinweis zu der ausgewählten Sicherheitslücke.
Anforderungen	FA-1, FA-2, FA-3

Tabelle 4.2: Anwendungsfall der lokale Überprüfung

Titel	Serverseitige Überprüfung
Anwendungsfall - Nr.	3
Akteur	Benutzer, Applikation, Server
Ziel	Durchführung der Server Überprüfung.
Auslöser	Der Benutzer drückt auf den Button „Server“.
Vorbedingung	Anwendung ist an und der Smartphone befindet sich im selben Netzwerk wie der Server.
Nachbedingung	Server Überprüfung wurde ausgeführt und die Liste der CVEs werden dem Benutzer angezeigt.
Ablaufschritte	1.) Der Benutzer drückt den “Server” Button und startet damit die Serverseitige Überprüfung. 2.) Die Applikation sendet eine Anfrage auf Sicherheitslücken mit Betriebssystem und Systemversion zum Server. 3.) Der Server verarbeitet die Anfrage, wertet diese aus und sendet eine Antwort an die Applikation. 4.) Die Applikation verarbeitet die Antwort und zeigt die CVE Liste an mit den Sicherheitslücken 5.) Der Benutzer wählt eine CVE aus. 6.) Der Benutzer bekommt eine Beschreibung und einen Hinweis zu der ausgewählten Sicherheitslücke.
Anforderungen	FA-1, FA-4, FA-5

Tabelle 4.3: Anwendungsfall der Server Überprüfung

4.2 Systemarchitektur

In der Abbildung 4.2 findet sich ein Überblick über die Architektur und die wichtigsten Komponenten, die für dieses System benutzt werden. Wie zu erkennen ist, besteht die Architektur dieses Systems aus zwei Teilen, Client und Server. Der Client ist das Smartphone, auf dem die Anwendung läuft. Der Client führt die lokale Überprüfung mit einer kleinen Datenbank aus, die aus den Sicherheitslücken besteht, die im Kapitel 3.2.1 aufgezählt wurden. Der Server verfügt über einen Schwachstellen-Scanner und hat eine größere Datenbank, die sich ständig aktualisiert. Der Client verbindet sich über die Anwendung mit dem Server, um mit Hilfe dessen eine bessere Sicherheitslückenüberprüfung durchzuführen. Da der Server nicht über das Internet verfügbar ist, muss der Client im selben Netzwerk wie der Server sein, damit eine Überprüfung stattfinden kann.

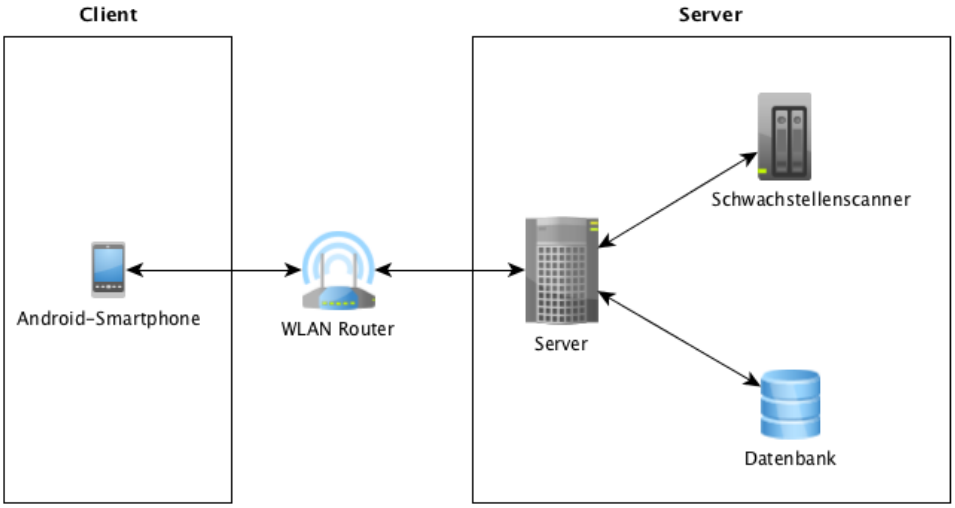

Abbildung 4.2: Systemarchitektur Überblick

4.3 Fachliches Datenmodell

Das in der Abb. 5.4 dargestellte Klassendiagramm wurde entsprechend den in den Kapiteln 4.1.1 und 4.1.2 skizzierten Anforderungen entwickelt. Dieses Diagramm zeigt seine Eigenschaften und die Assoziationen.

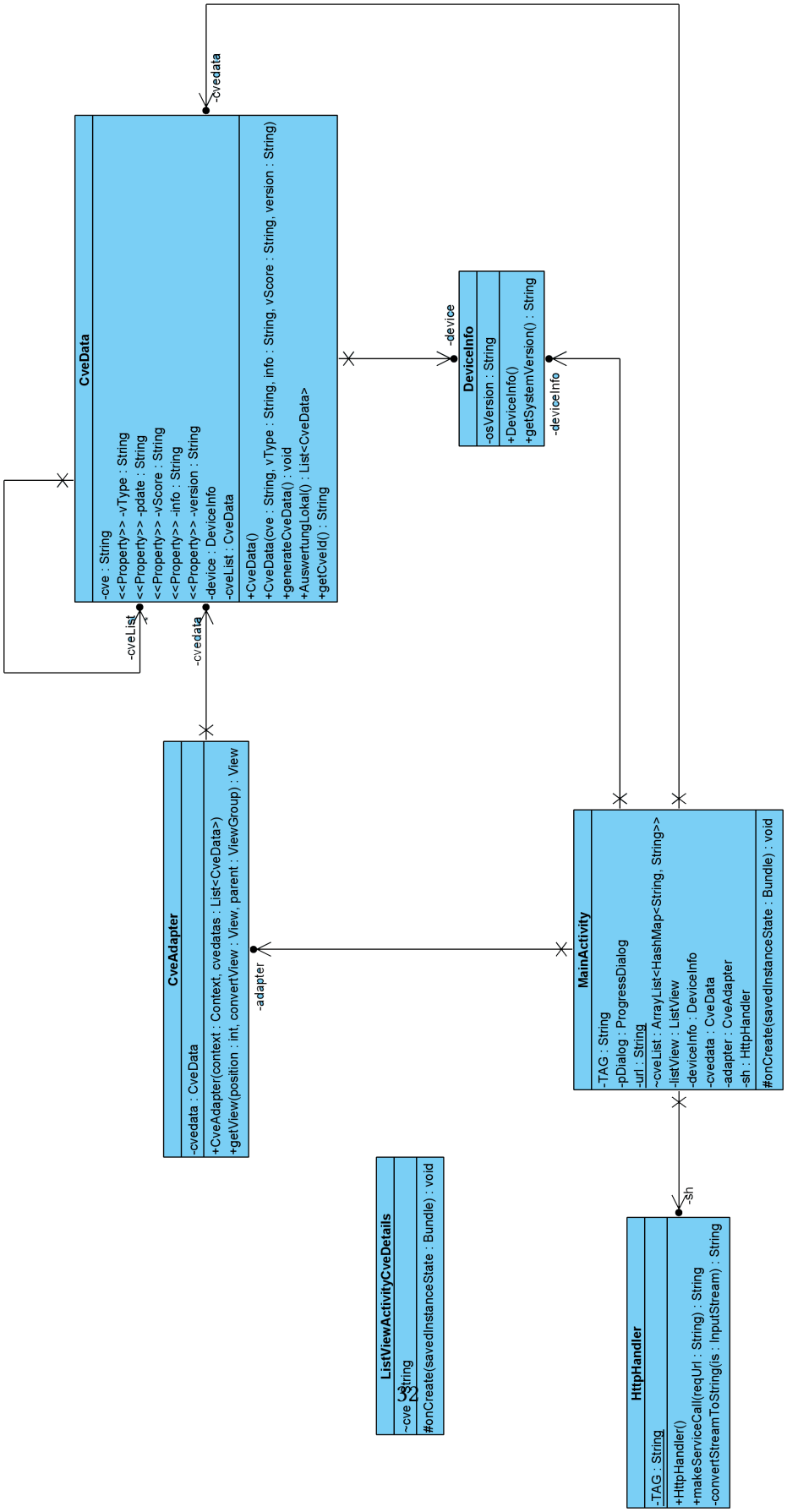

Abbildung 4.3: Klassendiagramm

4.3.1 Activities

MainActivity.java
Diese Klasse dient als Einstiegspunkt für die Benutzerinteraktion mit der Applikation. Es werden die Listener für Buttons (setOnClickListener-Methode) und Liste (setOnItemClickListener) definiert und implementiert.

ListViewActivityCveDetails.java
Diese Klasse dient zur Darstellung der detaillierten Informationen aus Lokaler und serverseitger Überprüfung in einer neuen Activity.

4.3.2 Models

HttpHandler.java
Diese Klasse stellt sowohl die makeServiceCall-Methode zum Aufbau einer Serververbindung als auch die convertStreamToString-Methode zum speichern der Serverantwort bereit.

DeviceInfo.java
Dieser Klasse stellt die Systemversion des Gerätes über eine getter-Methode zur Verfügung.

CveAdapter.java
Diese Klasse bildet das Bindeglied zwischen anzuzeigenden Daten und der Grafischen Benutzeroberfläche (item_cve.xml).

CveData.java
Diese Klasse dient zur Modellierung der Schwachstellen, hierzu werden die einzelnen Eigenschaften wie CVE-ID (cveId), Beschreibung (info), Risikoeinstufung(vScore) und ein Schwachstellen Typ(vType) als Klassenattribut definiert. Des weiteren werden in der Klasse über die generateCveData-Methode, Metadaten für die Lokale Sicherheitsüberprüfung generiert und können über die auswertungLokal-Methode der jeweiligen Systemversionen eines Gerätes zugewiesen werden.

GetCve.java
Die Klasse wird verwendet um die Serverantwort (HttpHandler.java) bei der Serverseitigen Sicherheitslückenanalyse zu Parsen und beim entsprechend Aufruf über Listener in der Liste anzuzeigen.

4.3.3 Layout (UI)

Die Grafische Benutzeroberfläche wird durch folgende Layout XML Dateien realisiert:

activity_customlist.xml
Dieses Layout definiert eine Liste, zur Auflistung der gefundenen Sicherheitslücken.

activity_cve_detail.xml
Dieses Layout stellt die detaillierten Information zu einer Sicherheitslücke welche aus der Liste (activity_customlist.xml) ausgewählt wurde dar (cveId, vScore, vType, info).

activity_main.xml
Dieses Layout stellt die Start bzw. Hauptseite der Applikation dar, in welcher über zwei Buttons entweder die Lokale oder Serverseitige Sicherheitsanalyse gestartet werden können.

item_cve.xml
Mit diesen Layout wird das aussehen der einzelnen Listeneinträge, der CVE-IDs, definiert.

4.3.4 Sequenzdiagramme

Mit Hilfe von zwei Sequenzdiagrammen wird der Aufruf in der Applikation bei den Szenarien „lokale Überprüfung“ und „Server-Überprüfung“ beschrieben. In der Abb. 4.4 wird die „lokale Überprüfung“ beschrieben. Die „lokale Überprüfung“ erfolgt ohne eine Verbindung zum Server, da hier auf die in der Anwendung vordefinierte Datenbank zugegriffen wird. Die Abb. 4.5 beschreibt die „Server-Überprüfung“, die eine Verbindung zum Server aufbauen muss, da die Applikation hierbei mit Hilfe der Server-Datenbank auf Sicherheitslücken geprüft wird.

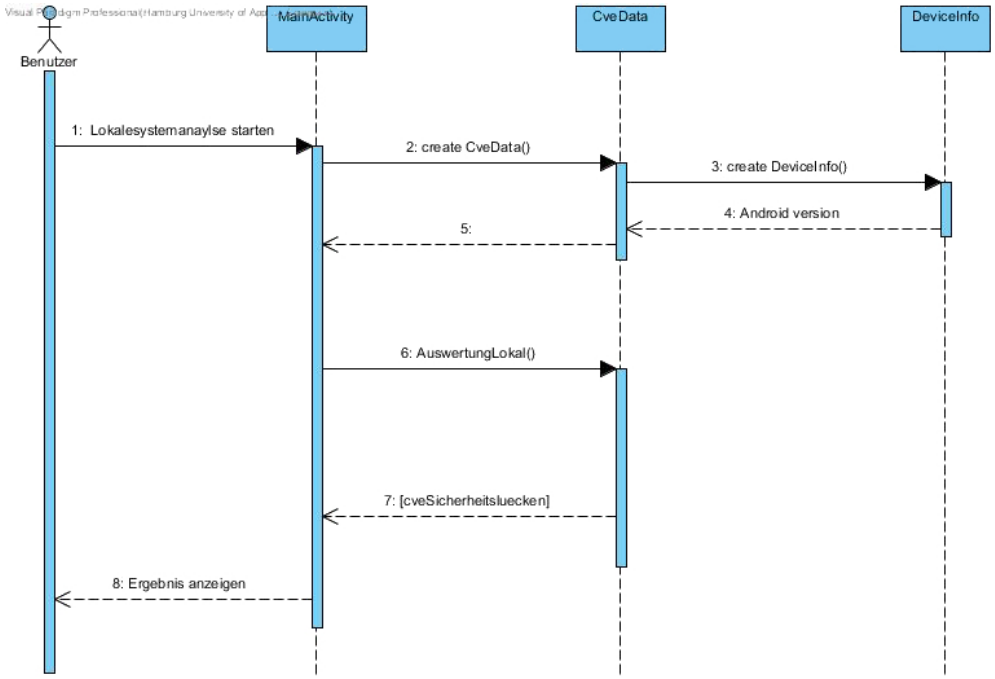

Abbildung 4.4: Sequenzediagramm der Lokalen Überprüfung

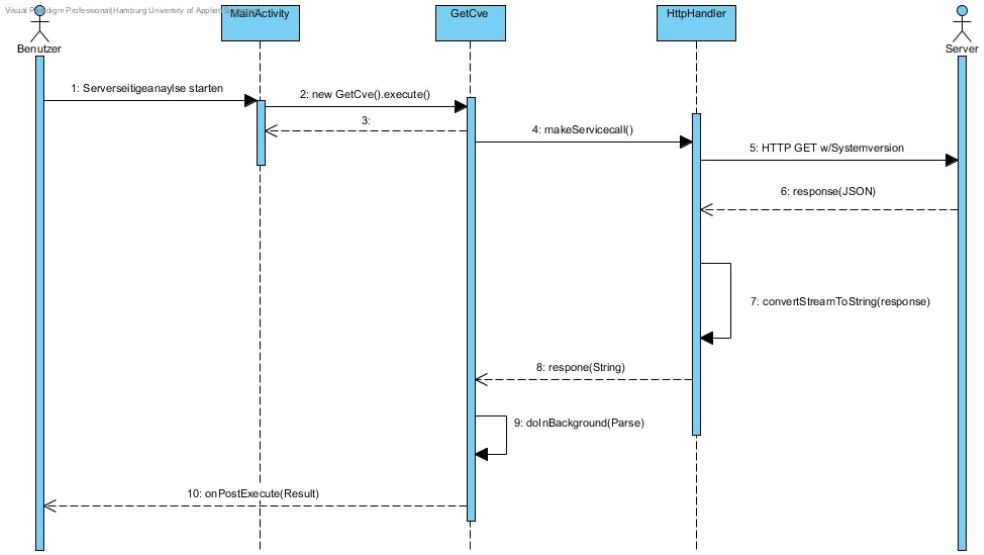

Abbildung 4.5: Sequenzediagramm der serverseitigen Überprüfung

4.4 Entwurf GUI

Damit ein Benutzer mit der Anwendung interagieren kann, werden diesem grafische Bedienelemente zur Verfügung gestellt. Diese sind Bestandteil der grafischen Oberfläche und werden nachfolgend einzeln vorgestellt sowie erläutert.

4.4.1 Lokale und serverseitige Systemüberprüfung

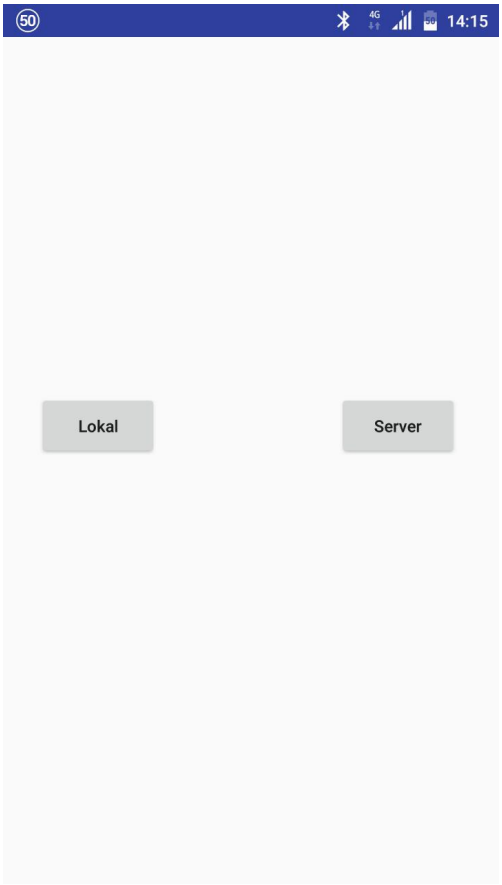

Abbildung 4.6: Startbildschirm

So bald der Benutzer die Applikation startet gelangt wird der Startbildschirm der Applikation aufgerufen. Der Startbildschirm ist ganz Simple gehalten und leicht verständlich. Im Startbildschirm hat der Benutzer die Wahl zwischen einer Lokale Überprüfung (Lokal) und einer serverseitigen Überprüfung (Server).

Nachdem der Benutzer eine Überprüfung ausgewählt hat gelangt er auf ein neues Fenster das Links in der Abbildung 4.7 angezeigt wird.

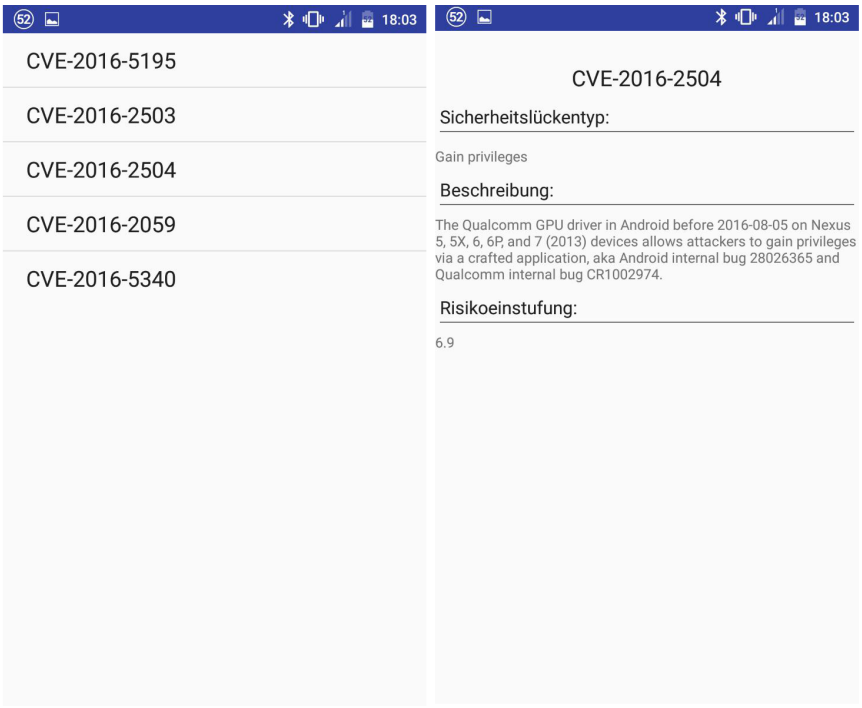

Abbildung 4.7: Überprüfung gestartet

In diesem neuen Fenster werden dem Benutzer die Sicherheitslücken eingeblendet welche auf seinem Smartphone entdeckt wurden. Für detaillierte Informationen zu einer Sicherheitslücke wird die CVE-ID angeklickt. Die Informationen werden dann in einem neuem Fenster geladen Abbildung 4.7. Hier kann der Benutzer den Typ der Sicherheitslücke, sowie die Beschreibung und die Risikoeinstufung einsehen.

Dies trifft sowohl auf die Lokale sowie auf die serverseitige Überprüfung zu, jedoch gibt es bei der serverseitigen Überprüfung zwischen dem Startbildschirm und dem Fenster das die CVEs anzeigt ein Ladefenster das Links in der Abbildung 4.8 dargestellt wird. Wenn die Applikation sich nicht mit dem Server verbinden kann wird eine Fehlermeldung ausgegeben die man in der Abbildung 4.8 sehen kann.

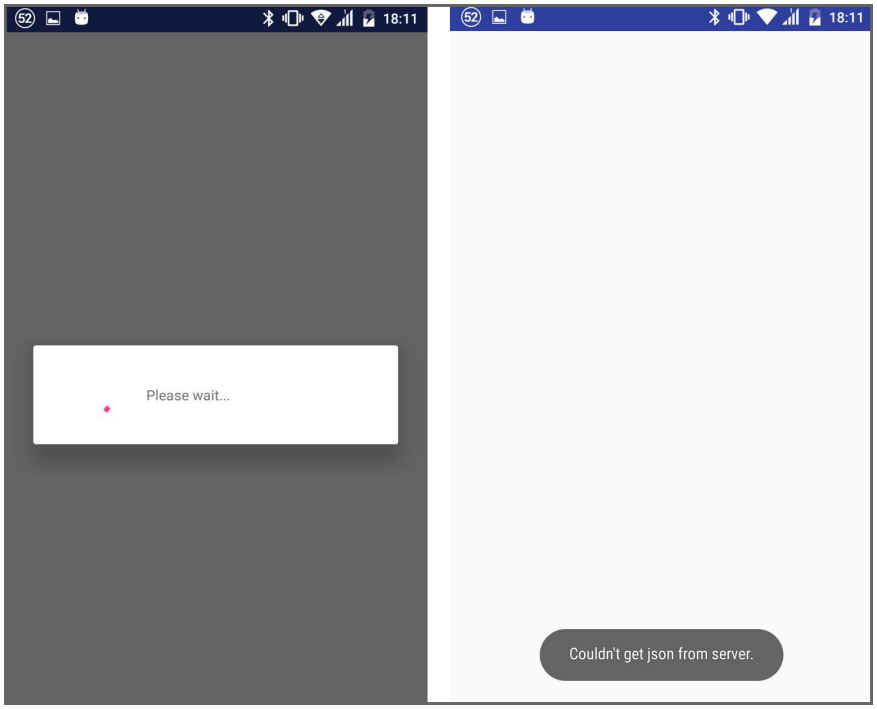

Abbildung 4.8: Serverseitige Überprüfung fehlgeschlagen

5 Implementierung

5.1 Entwicklungsumgebung

Nachdem im 4. Kapitel die funktionalen und nicht funktionalen Anforderungen sowie die Anwendungsfälle und die Systemarchitektur der Applikation festgehalten worden sind, werden in diesem Kapitel die Applikationen, wie in Kapitel 4 festgehalten implementiert.s

Android Studio

Zur Programmierung dieser Anwendung wurde die Entwicklungsumgebung Android Studio (Version 2.3.1) von Google eingesetzt. Diese basiert auf der Entwicklungsumgebung IntelliJ IDEA der Firma JetBrains, die für Java benutzt wird, aber mit verschiedenen Ablegern (wie Android Studio) auch für andere Programmiersprachen wie Python, Ruby, c/c++ oder, wie in diesem Fall, für Android (Java) benutzt werden kann. Android Studio kann sowohl auf den Betriebssystemen Windows, Mac oder Linux laufen, wobei für diese Arbeit ein Mac-Betriebssystem (Version 10.11.6) benutzt wurde.[44]

Java

Java ist eine objektorientierte Programmiersprache, die grundsätzlich aus zwei Teilen besteht, Java Development Kit (JDK) und Java Runtime Environment (JRE). Die JDK dient dem Erstellen der Java-Programme, wohingegen die JRE zur Ausführung der Java-Programme dient. Die JRE umfasst die JVM und die mitgelieferten Bibliotheken. Java wurde 1995 von Sun Microsystems erfunden, welches 2010 von Oracle aufgekauft wurde. Java ist plattformunabhängig und läuft auf jedem Rechner, wenn auf diesem die passende Laufzeitumgebung installiert worden ist. [47]

5.2 Realisierung Lokale- und Serverfunktion

5.2.1 Lokale Systemüberprüfung

Die Lokale Systemüberprüfung wird im wesentlich durch die Klasse CveData realisiert. Um ein Smartphone lokal zu prüfen werden Informationen zu Sicherheitslücken benötigt. Hierzu wird eine Liste (cveList) mit CVE samples gefüllt, auf die das Smartphone geprüft werden soll (siehe Listing 5.1).

```
cveList.add(new CveData("CVE-2016-5195", "Gain_privileges",
"Race_condition_in_mm/gup.c_in_the_Linux_kernel_2.x_through_4.x
before_4.8.3_allows_local_users_to_gain_privileges_by_leveraging
incorrect_handling_of_a_copy-on-write_(COW)_feature_to_write_to_a
read-only_memory_mapping,_as_exploited_in_the_wild_in_October_2016,
aka_\"Dirty_COW.\"", "7.2", "712"));
```

Listing 5.1: Codeausschnitt aus der Methode generateCveData (CveData-Klasse)

Um eine Sicherheitslücken einen betroffenen Gerät zuordnen zu können wird eine Auswertungsmethode implementiert (siehe Listing 5.2). Um die Auswertung zu erleichtern wird die Nummer der Systemversion des Gerätes zu einem Integer gecastet (int version). Ist die Systemversion der Sicherheitslücke großer gleich der Systemversion des Gerätes, so ist das Gerät von der Sicherheitslücke betroffen und wird der Liste cveSicherheitsluecken hinzugefügt. Sobald der Auswertungsvorgang abgeschlossen ist wird die Liste (cveSicherheitsluecken) der MainActivity übergeben, um diese dann den Benutzer Grafisch sichtbar zu machen.

```
List<CveData> cveSicherheitsluecken = new ArrayList<>();
String osVersion = device.getSystemVersion().replaceAll("\\.", "");

if (osVersion.length() == 2) {
osVersion = osVersion + "0";
}

int version = 0;
int cveVersion = 0;

version = Integer.parseInt(osVersion);

for (int i = 0; i < cveList.size(); i++) {
try {
cveVersion = Integer.parseInt(cveList.get(i).getVersion().toString());
```

```
16 } catch (NumberFormatException e) {
17 }
18
19 if ((cveVersion >= version)) {
20 cveSicherheitsluecken.add(cveList.get(i));
21 }
22 }
```

Listing 5.2: auswertungLokal-Methode (CveData-Klasse)

```
1 String osVersion;
2 osVersion = android.os.Build.VERSION.RELEASE;
```

Listing 5.3: Systemversion ermitteln

5.2.2 Server Systemüberprüfung

Bei der Serverseitigen Systemüberprüfung wird zunächst eine Verbindung zum Server aufgebaut. Über den GET Request wird die Systemversion des Smartphones übermittelt, der Server wertet diese aus und antwortet mit einem JSON-Response. Diese Serverantwort wird im String response gespeichert. (siehe Listing 5.4)

```
1 String response = null;
2 try {
3 URL url = new URL(reqUrl);
4 HttpURLConnection conn = (HttpURLConnection) url.openConnection();
5 conn.setRequestMethod("GET");
6 // read the response
7 InputStream in = new BufferedInputStream(conn.getInputStream());
8 response = convertStreamToString(in);
9 } catch (MalformedURLException e) {
10 Log.e(TAG, "MalformedURLException: " + e.getMessage());
11 } catch (ProtocolException e) {
12 Log.e(TAG, "ProtocolException: " + e.getMessage());
13 } catch (IOException e) {
14 Log.e(TAG, "IOException: " + e.getMessage());
15 } catch (Exception e) {
16 Log.e(TAG, "Exception: " + e.getMessage());
17 }
18 return response;
```

Listing 5.4: makeServiceCall-methode (HttpHandler-Klasse)

Das String response (jsonStr) wird in der doInBackground-Methode geparst und dann in eine Liste gespeichert (cveList) (siehe Listing 5.5). Sobald die doInBackground-Methode mit ihrer Arbeit fertig ist, werden die gefunden Sicherheitslücken dem Benutzer Grafisch sichtbar gemacht.

```
HttpHandler sh = new HttpHandler();

// Making a request to url and getting response
String jsonStr = sh.makeServiceCall(url);

Log.e(TAG, "Response from url: " + jsonStr);

if (jsonStr != null) {
try {
// Getting JSON Array node
JSONArray cveArray = new JSONArray(jsonStr);

// looping through All Cve's
for (int i = 0; i < cveArray.length(); i++) {
JSONObject c = cveArray.getJSONObject(i);

String id = c.getString("cveId");
String score = c.getString("vScore");
String info = c.getString("info");
String typ = c.getString("vType");

// tmp hash map for single cve
HashMap<String, String> cveTmpList = new HashMap<>();

// adding each child node to HashMap key => value
cveTmpList.put("cveId", id);
cveTmpList.put("vScore", score);
cveTmpList.put("info", info);
cveTmpList.put("vType", typ);

// adding cve to cve list
cveList.add(cveTmpList);
}
```

Listing 5.5: Codeausschnitt doInBackground-methode (GetCve-Klasse)

6 Test

Da die Implementierung der Applikation im vorherigen Kapitel erfolgt ist, wird die entwickelte Applikation im Anschluss mehreren Tests unterzogen, um deren Funktionsfähigkeit zu prüfen. Zu diesem Zweck wurde ein Gerät der Marke ZTE, welches über die Android-Version 6.0 verfügt, benutzt. Android Studio bietet selbst jedoch auch einen Emulator, mit dem man ein Smartphone der verschiedensten Marken simulieren kann. Im Rahmen dieser Bachelorarbeit wurde aufgrund der nicht vorhandenen Hardware mit einer alten Android-Version eine Nexus-6-Simulation gewählt. Das simulierte Smartphone sieht von der Optik her genauso wie ein richtiges Smartphone aus und verhält sich auch genauso wie ein solches. Zudem kann man auf dem simulierten Smartphone die Android-Version frei auswählen, was vorliegend auch genutzt wurde, um die Applikation auch auf anderen Android-Versionen, die beim ZTE nicht verfügbar waren, zu testen. In den nachfolgend dargestellten Testszenarien wurden jeweils die Hardware (Smartphone ZTE Blade v7 mit der Android-Version 6) und der Emulator von Android Studio verwendet.

6.1 Verhalten und Darstellung

In diesem Testszenario wird das GUI-Verhalten der Applikation getestet. Dabei soll sichergestellt werden, dass die Darstellung der einzelnen Elemente korrekt ist und die Funktionalität der Navigation geprüft wird.

Anforderungen

Die Applikation wurde erfolgreich gestartet.

Kriterien

Beim Drücken auf „Lokal“ soll die Seite mit den CVEs angezeigt werden, die in der Applikations-Datenbank einprogrammiert sind. Beim Drücken auf „Server“ soll hingegen die Seite mit den CVEs angezeigt werden, die von der Serverdatenbank bereitgestellt werden. Wenn man auf

eine der beiden Seiten gelangt und dann eine CVE anklickt, sollen alle Hinweise zu ebendieser CVE angezeigt werden.

Testablauf

Beim ersten Anlauf wird auf einen der beiden vorhandenen Buttons gedrückt und beim zweiten Anlauf auf den jeweils anderen. Bei beiden Anläufen wird sowohl geprüft, ob die richtige Seite angezeigt wird, als auch, ob vom Smartphone die richtigen Hinweise bezüglich der CVE, die man angeklickt hat, angezeigt werden.

Auswertung

In beiden Fällen wurde erfolgreich zu der jeweils zu erwartenden Seite sowie der CVE-Seite gewechselt und die Hinweise bezüglich der CVE wurden ebenfalls korrekt angezeigt.

6.2 Lokale Testszenarien

Dieser Lokalen Testszenarien orientiert sich an die gestellten Funktionalen Anforderungen in Kapitel 4.1.1 (FA-1 bis FA-3.1).

6.2.1 Lokale Überprüfung

Hier wird die korrekte Funktionalität der lokalen Überprüfung getestet. Es wird geprüft ob die Daten aus der fest Implementierten Liste richtig ausgelesen und korrekt angezeigt werden.

Anforderungen

Die Applikation wurde erfolgreich gestartet.

Kriterien

Beim Drücken auf „Lokal“ soll die Seite mit den CVEs angezeigt werden, die in der Applikations-Datenbank einprogrammiert sind. Dann sollte die Seite mit der Liste der CVEs der lokalen Überprüfung angezeigt werden. Wenn man auf eine CVE klickt, sollten alle Hinweise bezüglich dieser CVE korrekt angezeigt werden.

Testablauf

Es wird geprüft, ob die Liste der CVEs korrekt angezeigt wird. Dann soll geprüft werden, ob die Hinweise bezüglich der CVEs korrekt angezeigt werden. Anschließend werden beide Ergebnisse, diejenigen bezüglich der Hardware und der Simulation, verglichen, um herauszufinden, ob die CVEs nicht in der falschen Android-Version angezeigt werden.

Auswertung

Das Auslesen der fest implementierten Liste funktioniert einwandfrei. Alle CVEs werden ordnungsgemäß angezeigt und die Hinweise ebenfalls. Es verläuft alles ohne Fehlermeldung und die verglichenen CVEs werden alle in den richtigen Android-Versionen angezeigt.

6.3 Testszenarien des Servers

Der nachfolgende dargestellte Testszenario sind bis auf die Verbindung zum Server die gleichen wie bei der lokalen Überprüfung. Für dieses Testszenario ist dem Verfasser der vorliegenden Arbeit das Netzwerk des Entwicklers, der den Server für eine andere Bachelorarbeit programmiert hat, zur Verfügung gestellt worden, da dieses Testszenario ohne ein eigenes Netzwerk und ohne den Server nicht durchführbar wäre. Dieses Testszenario widmet sich der Verbindung der Applikation zum Server mit Hilfe der Applikation.

6.3.1 Verbindung zum Server

Dieses Szenario beinhaltet den Versuch der Applikation, sich mit dem Server zu verbinden. Dabei wird geprüft, ob sich die Applikation mit dem Server verbinden kann, und getestet, was passiert, wenn dies nicht der Fall ist.

Anforderungen

Die Applikation wird erfolgreich gestartet und der Server war aktiv. Sowohl der Server als auch das Smartphone befinden sich im selben Netzwerk.

Kriterien

Die Applikation soll mit einem REST-Aufruf überprüfen, ob eine Verbindung mit dem Server hergestellt werden kann. Wenn die Verbindung erfolgreich hergestellt worden ist, soll die Applikation zu der Seite mit der Liste der CVEs springen.

Testablauf

Zuerst wird der positive Ablauf getestet: Die Applikation stellt erfolgreich eine Verbindung zum Server her und springt auf die Seite mit der Liste der CVEs. Beim zweiten, dem negativen Ablauf wird das Smartphone aus dem Netzwerk entfernt und dann getestet, ob sich der Ladekreis ununterbrochen weiter dreht, bis eine Fehlermeldung erscheint.

Auswertung

Die Applikation springt bei Verbindung mit dem Server erfolgreich auf die erwartete Seite mit der Liste der CVEs und im Falle des Nicht-Verbindens mit dem Server erscheint die zu erwartende Fehlermeldung.

7 Fazit

Nachdem im 6. Kapitel die Implementierung der Applikation aus dem 5. Kapitel getestet worden ist, wird nachfolgend zuerst der Inhalt der vorliegenden Bachelorarbeit zusammengefasst. Anschließend wird der aktuelle Stand der in dieser Bachelorarbeit entwickelten Applikation aufgezeigt, um sie weiterentwickeln zu können, wenn entsprechender Bedarf besteht.

7.1 Zusammenfassung

Die vorliegende Bachelorarbeit hat sich mit der Sicherheit von Android-Geräten beschäftigt, wobei auch andere mobile Endgeräte kurz erwähnt wurden und auch über deren Sicherheit geschrieben wurde. Im Rahmen dieser Bachelorarbeit wurde im Anschluss eine Applikation für Android entwickelt, die den Endnutzer vor potenziellen Schwachstellen auf seinem mobilen Endgerät warnen soll.

Zu Beginn wurde in den Grundlagen in Kapitel 2 die Sicherheitsarchitektur von Android beschrieben. Dabei wurde ein Einblick in die von Android verwendeten Sicherheitsmechanismen für die System-, Applikations- und Datensicherheit gewährt. Im Folgenden wurden Sicherheitslücken wie die Malware Keylogger und Trojaner sowie wichtige Schwachstellen-Typen wie „Denial of Service“ (DoS), „Execute Code“ und „Gain Privilege“ aufgezeigt und erläutert, wie diese Schwachstellen von CVE-Details mit den CVEs gekennzeichnet werden.

Im Anschluss wurden in Kapitel 3 die Sicherheitsaspekte des mobilen Betriebssystems Android analysiert, indem zwei wichtige Sicherheitsrisiken (herstellerspezifische Android-System-Updates und das Rooten) näher erläutert wurden. Darauf folgend wurde das Betriebssystem Android mit iOS und Windows Phone verglichen und deren gemeinsame Sicherheitslücken (FREAK-Attack und Adobe Flash/Air) sowie die alleinigen Sicherheitslücken von Android wurden erläutert.

Im Konzept in Kapitel 4 wurden in Bezug auf die Sicherheitsanalyse die funktionalen und nicht funktionalen Anforderungen zum Aufbau der Applikation ermittelt. Danach wurden die

Anwendungsfälle basierend auf der Anforderungsanalyse erstellt. Im nächsten Schritt wurde mit Hilfe einer Abbildung (4.2) ein Überblick über die Systemarchitektur und die wichtigsten Komponenten, die für dieses System benutzt werden, gewährt . Danach wurde ein Klassendiagramm zur Veranschaulichung des Aufbaus der Applikation erstellt. Zuletzt wurde die grafische Oberfläche der Applikation gezeigt.

Im Anschluss wurden in Kapitel 5 die für die Applikation benötigte Entwicklungsumgebung und die Programmiersprache vorgestellt. Entsprechend den Anforderungen, die in Kapitel 4 erläutert wurden, wurde die Applikation entwickelt und die wichtigsten Methoden im Hinblick auf die Funktionalität aufgezeigt.

Im 6. und letzten Kapitel wurden die Testfälle der Applikation beschrieben.

7.2 Aktueller Stand

Die in dieser Bachelorarbeit entwickelte mobile Applikation für Android (Versionen 4.1 bis 6) prüft das mobile Endgerät auf bekannte Schwachstellen und gibt über diese detailliert Auskunft. Der Benutzer kann dabei zwischen einer lokalen und einer serverseitigen Überprüfung auswählen. Je nachdem, welche Option der Benutzer auswählt, wird die lokale oder serverseitige Überprüfung, bei der sich die Applikation mit dem Server verbinden muss, ausgeführt und die Hinweise bezüglich der jeweiligen Sicherheitslücken werden angezeigt.

7.3 Ausblick

Das Ziel dieser Bachelorarbeit war es, eine Sicherheitsanalyse-Applikation für Android zu entwickeln, die dem Benutzer Hinweise auf die möglichen Sicherheitslücken in seinem mobilen Betriebssystem gibt. Hierzu wurde im Rahmen dieser Bachelorarbeit, in den Abschnitten 4.1.1 und 4.1.2, funktionale und nicht funktionale Anforderungen ermittelt, die im Rahmen der Bachelorarbeit allesamt erfüllt werden konnten.

Die Applikation wurde vorliegend speziell für Android fertiggestellt, jedoch könnte man diese erweitern. So könnte beispielsweise für die Applikation ein WebView entwickelt werden, um einen Netzwerk-Schwachstellen-Scanner (Open Vulnerability Assessment System, OpenVAS), der vom selben Server bereitgestellt wird und die Datenbank für die serverseitige

Überprüfung zur Verfügung stellt, aufzurufen, um Schwachstellen im Netzwerk zu suchen. Außerdem könnte bei der lokalen Überprüfung eine XML-Datei benutzt werden, um die Sicherheitslücken der lokalen Überprüfung zu speichern, anstatt sie in Form einer Liste fest in der Applikation zu implementieren.

Literatur

[1] aasche. *dmesg*. 23. Apr. 2017. URL: https://wiki.ubuntuusers.de/dmesg/ (besucht am 27. 04. 2017).

[2] Check Point Mobile Research Team Adam Donenfeld. *QuadRooter: New Android Vulnerabilities in Over 900 Million Devices*. 7. Aug. 2016. URL: http://blog.checkpoint.com/2016/08/07/quadrooter/ (besucht am 03. 03. 2017).

[3] Patrick Beuth. *Sicherheitslücke gefährdet fast alle Android-Smartphones*. 28. Juli 2015. URL: http://www.zeit.de/digital/mobil/2015-07/android-stagefright-mms-sicherheit (besucht am 28. 02. 2017).

[4] BSI. *Überblickspapier Android*. 20. Jan. 2014. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Download/Ueberblickspapier_Android_pdf.pdf?__blob=publicationFile (besucht am 28. 02. 2017).

[5] bsi-fuer buerger.de. *Trojanische Pferde*. 2017. URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/Risiken/Schadprogramme/TrojanischePferde/trojanischepferde_node.html;jsessionid=A169328DBA1B4E77F2A6697D9702_cid360 (besucht am 13. 02. 2017).

[6] Georgie Casey. *Inserting keylogger code in Android SwiftKey using apktool*. 6. März 2013. URL: http://www.georgiecasey.com/2013/03/06/inserting-keylogger-code-in-android-swiftkey-using-apktool/ (besucht am 09. 02. 2017).

[7] chip.de. *Android rooten: Vorteile und Nachteile*. 6. März 2017. URL: http://www.chip.de/news/Android-rooten-Vorteile-und-Nachteile_62681106.html (besucht am 28. 02. 2017).

[8] computervirus.de. *Was ist Malware?* 2017. URL: http://computervirus.de/was-ist-malware/ (besucht am 31. 01. 2017).

[9] cvedetails.com. *Vulnerability Statistics*. 2017. URL: http://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224Zeit:16.Februar.2017 (besucht am 06. 02. 2017).

[10] Saki Athanassios Danoglidis. *Malware in Deutschland 2015*. 26. Feb. 2015. URL: https://entwickler.de/online/webmagazin/malware-deutschland-2015-40432.html (besucht am 02. 02. 2017).

[11] Zero Day. *Adobe Flash AS2 Sound loadSound Use-After-Free Remote Code Execution Vulnerability*. 10. Nov. 2015. URL: http://www.zerodayinitiative.com/advisories/ZDI-15-563/ (besucht am 09. 03. 2017).

[12] CVE Details. *Total Number Of Vulnerabilities Of Top 50 Products By Vendor*. 2016. URL: https://www.cvedetails.com/top-50-products.php?year=2016 (besucht am 17. 03. 2017).

[13] dirtycow.ninja. *Dirty COW*. 2017. URL: https://dirtycow.ninja (besucht am 05. 03. 2017).

[14] droidwiki.org. *Logcat*. 2017. URL: https://www.droidwiki.org/wiki/Logcat (besucht am 27. 04. 2017).

[15] droidwiki.org. *Root*. 2017. URL: https://www.droidwiki.org/wiki/Root (besucht am 25. 04. 2017).

[16] Prof. Dr. Claudia Eckert. *IT-Sicherheit. Konzepte-Verfahren-Protokolle*. Hrsg. von Prof. Dr. Claudia Eckert. Oldenbourg Verlag, 2014. ISBN: 978-3-486-72138-6. (Besucht am 11. 01. 2017).

[17] Hauke Gierow. *Qualcomm-Schwachstelle bedroht 900 Millionen Android-Geräte*. 8. Aug. 2016. URL: https://www.golem.de/news/quadrooter-qualcomm-schwachstelle-gefaehrdet-900-000-android-geraete-1608-122564.html (besucht am 03. 03. 2017).

[18] DATACOM Buchverlag GmbH. *buffer overflow*. 12. März 2015. URL: http://www.itwissen.info/buffer-overflow-Speicherueberlauf.html (besucht am 16. 02. 2017).

[19] DATACOM Buchverlag GmbH. *DoS (denial of service)*. 5. Juli 2016. URL: http://www.itwissen.info/DoS-denial-of-service-DoS-Attacke.html (besucht am 02. 03. 2017).

[20] DATACOM Buchverlag GmbH. *Trojaner*. 16. Juni 2013. URL: http://www.itwissen.info/Trojaner-trojan.html (besucht am 13. 02. 2017).

[21] DATACOM Buchverlag GmbH. *Überlauf*. 15. Okt. 2006. URL: http://www.itwissen.info/definition/lexikon/Ueberlauf-OV-overflow.html (besucht am 15. 02. 2017).

[22] DAN GOODIN. *Android phones rooted by "most serious" Linux escalation bug ever*. 24. Okt. 2016. URL: https://arstechnica.com/security/2016/10/android-phones-rooted-by-most-serious-linux-escalation-bug-ever/ (besucht am 21. 03. 2017).

[23] Seth Hanford und Max Heitman. *About CVSS*. URL: https://www.first.org/cvss (besucht am 07. 03. 2017).

[24] http://cve.mitre.org/about/. *About CVE*. 23. Feb. 2017. URL: http://cve.mitre.org/about/ (besucht am 07. 03. 2017).

[25] IDC. *Smartphone OS Market Share, 2016 Q3*. Nov 2016. URL: http://www.idc.com/promo/smartphone-market-share/os (besucht am 16. 03. 2017).

[26] itwissen.info. *Virus*. 2017. URL: http://www.itwissen.info/definition/lexikon/Virus-virus.html (besucht am 06. 02. 2017).

[27] itwissen.info. *Wurm*. 20. Aug. 2012. URL: http://www.itwissen.info/definition/lexikon/Wurm-worm.html (besucht am 06. 02. 2017).

[28] Justin C. Klein Keane. *Web Hacking Lesson 6 - Arbitrary Code Execution Vulnerabilities*. URL: http://www.madirish.net/tag/arbitrary%20code%20execution (besucht am 27. 01. 2017).

[29] Serge Malenkovich. *Was ist ein Keylogger?* 5. Apr. 2013. URL: https://blog.kaspersky.de/was-ist-ein-keylogger/884/ (besucht am 12. 02. 2017).

[30] ZEIT ONLINE. *HummingBad befällt Millionen Android-Geräte*. 6. Juli 2016. URL: http://www.zeit.de/digital/2016-07/trojaner-hummingbad-malware-android-geraete-klickbetrug (besucht am 02. 03. 2017).

[31] Elizabeth Palermo. *What is Privilege Escalation?* 17. Dez. 2013. URL: http://www.tomsguide.com/us/privilege-escalation,review-1983.html (besucht am 28. 02. 2017).

[32] Dave Piscitello. *What is Privilege Escalation?* Hrsg. von ICANN Blog. 18. Feb. 2016. URL: https://www.icann.org/news/blog/what-is-privilege-escalation (besucht am 28. 01. 2017).

[33] Steffen Pochanke. *Piraterie: So leicht implementiert man einen Keylogger in SwiftKey.* 12. März 2013. URL: http://www.giga.de/downloads/swiftkey-6/news/piraterie-so-leicht-implementiert-man-einen-keylogger-in-swiftkey/ (besucht am 14. 02. 2017).

[34] Andrey Polkovnichenko und Oren Koriat. *HummingBad: A Persistent Mobile Chain Attack.* 4. Feb. 2016. URL: http://blog.checkpoint.com/2016/02/04/hummingbad-a-persistent-mobile-chain-attack/ (besucht am 04. 04. 2017).

[35] Dennis Schirrmacher. *85 Millionen Android-Geräte von HummingBad-Malware befallen.* 5. Juli 2016. URL: https://www.heise.de/security/meldung/85-Millionen-Android-Geraete-von-HummingBad-Malware-befallen-3254664.html (besucht am 01. 03. 2017).

[36] Dennis Schirrmacher. *QuadRooter: Verwundbare LTE-Chips sollen über 900 Millionen Android-Geräte gefährden.* 8. Aug. 2016. URL: https://www.heise.de/security/meldung/QuadRooter-Verwundbare-LTE-Chips-sollen-ueber-900-Millionen-Android-Geraete-gefaehrden-3289647.html (besucht am 03. 03. 2017).

[37] Daniel Schröter. *Das Sicherheitsloch Buffer-Overflows und wie man sich davor schützt.* 5. Nov. 2001. URL: https://www.heise.de/ct/artikel/Das-Sicherheitsloch-285320.html (besucht am 11. 02. 2017).

[38] source. *Security Updates and Resources.* 2017. URL: https://source.android.com/security/overview/updates-resources.html (besucht am 24. 01. 2017).

[39] source.android.com. *Encryption.* 2017. URL: https://source.android.com/security/encryption/ (besucht am 20. 01. 2017).

[40] source.android.com. *Rooting of Devices.* 2017. URL: https://source.android.com/security/overview/kernel-security#rooting-devices (besucht am 28. 02. 2017).

[41] source.android.com. *Security Enhancements in Android 4.3.* 2017. URL: https://source.android.com/security/enhancements/enhancements43 (besucht am 21. 01. 2017).

[42] Censys Team. *The FREAK Attack.* 2015. URL: https://censys.io/blog/freak (besucht am 09. 03. 2017).

[43] PC Tools. *Arbitrary Code Execution.* 2016. URL: http://www.pctools.com/security-news/arbitrary-code-execution/ (besucht am 27. 01. 2017).

[44] wikipedia.org. *Android Studio.* 2017. URL: https://de.wikipedia.org/wiki/Android_Studio (besucht am 20. 03. 2017).

[45] wikipedia.org. *Dirty COW.* 2017. URL: https://en.wikipedia.org/wiki/Dirty_COW (besucht am 05. 02. 2017).

[46] wikipedia.org. *dm-crypt.* 2017. URL: https://de.wikipedia.org/wiki/Dm-crypt (besucht am 10. 04. 2017).

[47] wikipedia.org. *Java (Programmiersprache).* 2017. URL: https://de.wikipedia.org/wiki/Java_(Programmiersprache) (besucht am 01. 04. 2017).

[48] wikipedia.org. *Root.* 2017. URL: https://www.droidwiki.org/wiki/Root (besucht am 28. 02. 2017).

[49] wikipedia.org. *Stagefright(Sicherheitslücke).* 2017. URL: https://de.wikipedia.org/wiki/Stagefright_(SicherheitslÃijcke) (besucht am 28. 02. 2017).

[50] wikipedia.org. *Trojanisches Pferd.* 22. Jan. 2017. URL: https://de.wikipedia.org/wiki/Trojanisches_Pferd (besucht am 13. 02. 2017).

[51] wikipedia.org. *Trusted Execution Environment.* 2017. URL: https://de.wikipedia.org/wiki/Trusted_Execution_Environment (besucht am 28. 04. 2017).

[52] Google for Work|Android. *Android security white paper.* May 2015. URL: https://static.googleusercontent.com/media/1.9.24.55/en/US/work/android/files/android-for-work-security-white-paper.pdf (besucht am 04. 05. 2016).

[53] zLabs. *What is Quadrooter?* 10. Aug. 2016. URL: https://blog.zimperium.com/what-is-quadrooter/ (besucht am 03. 03. 2017).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Glossar

Booten Booten nennt man auch das Hochfahren/Starten des Smartphones. 6

dm-crypt mit dm-crypt kann man mit Hilfe verschiedener Algorithmen Daten ver- und entschlüsseln. [46]. 5

dmesg steht für display message und kann die meldungen des Kernel-Ringpuffers auf dem Bildschirm ausgeben. [1]. 10

logcat logcat ist ein Befehl der ADB und das gleichnamige Systemlogbuch... [14]. 10

Root Root definiert einen Benutzer der vollen Zugriff auf das Betriebssystem und dessen Ressourcen hat. [15]. 5

Trusted Execution Environment stellt eine sichere Laufzeitumgebung für Applikationen zur Verfügung[51]. 6

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. April 2017 Carlos Alberto da Silva Gonçalves