FernUniversität in Hagen
Fakultät für Mathematik und Informatik

Abschlussarbeit
im Studiengang Bachelor of Science Informatik

Erstellung eines virtuellen Mobile Security Labors

von
Anja Granvogl
Matrikelnummer: 9712500

Datum der Abgabe: 11.07.2024

Erstgutachterin und Betreuerin: Dr. Carina Heßeling
Zweitgutachter/-in: Dr. Marius Rosenbaum

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

VM	Virtual Machine
VMM	Virtual Machine Monitor
App	Applikation
GPS	Global Positioning System
PC	Personal Computer
ART	Android Runtime
GB	Gigabyte
MB	Megabyte
RAM	Random Access Memory
CVE	Common Vulnerabilities and Exposure
URI	Uniform Ressource Identifier
JDK	Java Development Kit
aapt	Android Asset Packaging Tool
adb	Android Debug Bridge
ARP	Address Resolution Protocol
ZAP	OWASP Zed Attack Proxy
NAT	Network Address Translation
API	Application Programming Interface
KVM	Kernel-based Virtual Machine
HAL	Hardware Abstraction Layer
HTTP	Hypertext Transfer Protocol
SMS	Short-Message-Service
MMS	Multimedia-Message-Service
GUI	Graphical User Interface

TLS Transport Layer Security

1. Einführung

An der FernUniversität in Hagen wird von der Fakultät Mathematik und Informatik die Lehrveranstaltung Mobile Security angeboten. Diese soll Bachelor- und Masterstudenten grundlegende Sicherheitskonzepte sowie Funktionsmechanismen mobiler Endgeräte und der darauf ausführbaren Applikationen vermitteln. Dabei liegt der Fokus auf den mobilen Betriebssystemvarianten iOS und Android.
Die therotische Vermittlung der Inhalte erfolgt anhand eines Leittextes, der die Studierenden durch den Basistext *Mobile Hacking* von Dr.-Ing. Michael Spreitzenbarth [1] führt und diesen um ausgewählte Inhalte erweitert.
Die theoretisch vermittelten Inhalte werden anhand der Bearbeitung von Einsendearbeiten vertieft. Im Rahmen der Einsendearbeiten führen die Studierenden auch praktische Übungen in Form von Analysen an mobilen Android-Applikationen durch. Um diese Übungen durchführen zu können, werden den Studierenden eine virtuelle Laborumgebung mit entsprechenden Analyseprogrammen und eine zu analysierende, mobile Applikation in zwei Versionen zur Verfügung gestellt.

1.1. Motivation

Die Algorithmen künstlicher neuronaler Netze sind eine, durch Vorgänge im menschlichen Gehirn inspirierte, maschinelle Abbildung der Wechselwirkungen zwischen Synapsen.
Sowohl technische neuronale Netze, als auch das menschliche Gehirn können lernen, indem sie, aufbauend auf einem durch gezielte Eingaben entwickelten Modell, Erlerntes durch den Erwerb von Erfahrung verfestigen [2]. Dieser Vorgang entspricht der Anwendung von erlernten Entscheidungsmodellen auf Problemstellungen, wie sie in den Übungen des Kurses Mobile Security zu lösen sind.
Da die Kombination theoretischer Lehre und praktischer Vertiefung der Lehrinhalte vorteilhaft für den Lernerfolg ist, sollte diese Kombination

von Lernmethodiken im Rahmen von Lehrveranstaltungen weiterhin unterstützt werden.
Eine Eigenschaft, die Menschen von Rechensystemen unterscheidet, ist der Einfluss von Motivation auf den Lernerfolg. Neben anderen Faktoren hat die Motiviertheit der Lernenden Einfluss auf den Wissensaufbau. Insbesondere spielt dabei auch das Belohnungssystem eine Rolle. Laut Roth muss "[...] die Lernsituation dem Schüler in irgendeiner Weise attraktiv erscheinen"[3].
Eine stabil lauffähige Laborumgebung, deren Nutzung entsprechend dokumentiert ist, ermöglicht Studierenden eine Vertiefung der theoretisch erworbenen Kenntnisse anhand praktischer Übungen, ohne erhöhten Einarbeitungsaufwand.

1.2. Überblick und Related Work

Durch die Betreuung der Lehrveranstaltung Mobile Security der FernUniversität in Hagen wird bereits eine Testumgebung zur Verfügung gestellt. Aufbauend auf der durch Dr. Kubek erstellten Systematik, soll ein neues System geschaffen werden, das an veränderte Softwareumgebungen angepasst ist und unabhängig von der Art des Hostbetriebssystems stabil lauffähig ist.
Das Fachbuch *Mobile Hacking* von Dr.-Ing. Michael Spreitzenbarth beschreibt schematisch, welche Bestandteile eine Laborumgebung enthalten sollte, damit geeignete Tests bezüglich der Softwaresicherheit ermöglicht werden. Darüber hinaus wird in dem Werk beschrieben, welche Arten von Analysesoftware genutzt werden können, um statische Codeanalysen zu ermöglichen, sowie Verhalten von Applikationen zur Laufzeit auswerten zu können [1]. Die Lauffähigkeit der Laborumgebung auf unterschiedlichen Basisbetriebssystemen wurde dabei nicht betrachtet.
Grundgegenstand dieser Thesis ist die systematische Betrachtung der Erstellung einer virtuellen Laborumgebung sowie der Entwicklung einer Testapplikation, die als Prüfobjekt für entsprechende Analysen genutzt werden kann.

1.3. Problembeschreibung und Zielsetzung

Ziel dieser Bachelorthesis ist, eine überarbeitete virtuelle Laborumgebung zu erstellen, die auf unterschiedlichen Betriebssystemvarianten stabil lauffähig ist und den Studierenden entsprechende Analysen zur Vertiefung der theoretisch erarbeiteten Inhalte ermöglicht. Diese soll den Studierenden in Form fertig konfigurierter Dateien zum Download zur Verfügung gestellt werden. Für Studierende, die nicht die vorkonfigurierten Dateien nutzen, sondern die virtuelle Laborumgebung selbst erstellen wollen, soll eine entsprechende Dokumentation angeboten werden. Zusätzlich soll eine entsprechend gestaltete Nutzerdokumentation erstellt werden, die die Installation der Umgebung auf den Rechnern der Anwender, sowie deren Nutzung erleichtert.
Darüber hinaus soll eine mobile Applikation, an der die Studierenden Analysen im Rahmen der Übungen durchführen können, erstellt werden. Diese muss gewisse Eigenschaften und Funktionen aufweisen, damit die Studierenden entsprechende Erkenntnisse aus den Analysen gewinnen können.
Die aktualisierte Laborumgebung muss schlank gestaltet sein, um den Ressourcenverbrauch auf den Basissystemen der Studierenden zu minimieren. Dabei muss die Laborumgebung dennoch einen angemessenen Nutzungsumfang ermöglichen, um das Lehrziel der Veranstaltung adäquat zu unterstützen.
Um den Studierenden ein entsprechendes System mit Testapplikation zur Verfügung stellen zu können, müssen als zentrale Fragen in dieser Thesis geklärt werden:

- Wie muss eine Laborumgebung beschaffen sein, damit sie auf Linux, Windows und MacOS Systemen die Analysen stabil und performant unterstützt?
- Welche Analysesoftware muss in der Laborumgebung installiert werden, um den Studierenden entsprechende Analysen an Android Applikationen zu ermöglichen?
- Welche Eigenschaften muss die zu analysierende Applikation aufweisen, um den Studierenden Erkenntnisse im Rahmen der Übungen zu ermöglichen, die dem Lehrziel dienen?

- Wie muss die Nutzerdokumentation gestaltet werden, um die Studierenden bei der Installation und der Anwendung der Laborumgebung adäquat zu unterstützen?

1.4. Lösungsansatz und Methodik

Um die im vorherigen Abschnitt beschriebene Zielsetzung erfüllen zu können, muss eine eingehende Auseinandersetzung mit der Thematik der Virtualisierung im Allgemeinen, den Basisbetriebssystemen virtualisierter Umgebungen und deren Eigenschaften erfolgen.
Bei der Evaluation von Analysesoftware, die geeignet ist, die im Rahmen der Übungen durchzuführenden Analysen zu unterstützen, ist es notwendig, die Merkmale entsprechender Werkzeuge zu betrachten und zu bewerten. Darüber hinaus müssen die durch die Studierenden zu analysierenden Objekte (Android-Applikationen), sowie die Geräte, auf denen diese ausgeführt werden, im Rahmen der Auseinandersetzung mit der Problemstellung näher betrachtet werden, um eine geeignete Testapplikation erstellen zu können.
Außerdem muss der Erfahrungsstand der Nutzer ermittelt werden, um die Dokumentation an diese angepasst gestalten zu können.

Bei der Behandlung der eben genannten Thematiken wird der Ansatz einer Betrachtung anhand eines Schichtenmodells gewählt, wie in Abbildung 1.1 schematisch dargestellt.

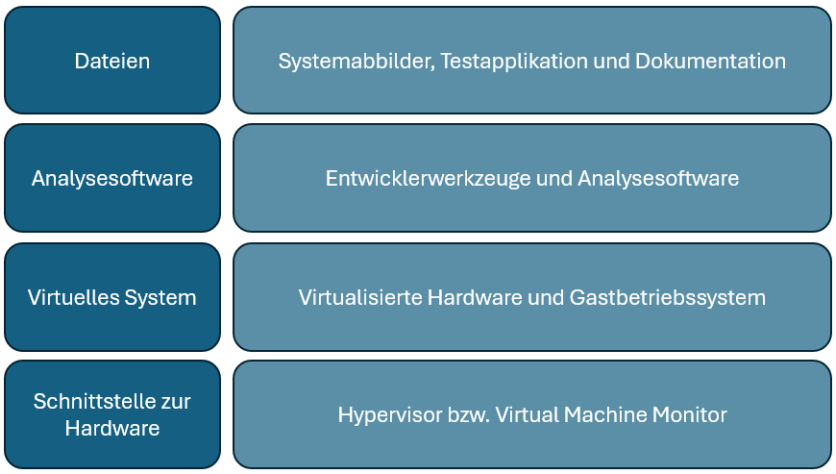

Abbildung 1.1.: Schichtenweise Problembetrachtung

Die Problemstellungen werden schichtweise von unten nach oben bearbeitet, bis hin zur Anwenderschnittstelle, die der Dokumentation und den vorkonfigurierten Dateien zum Download entspricht.

Im Rahmen der Erarbeitung von Grundlagen wird zunächst anhand einer eingehenden Recherche ein Basisverständnis für die Funktionsweise von Virtualisierung, der zu analysierenden Objekte (mobiler Geräte) sowie der auf ihnen ausführbaren Applikationen und entsprechender Analysemethoden und Werkzeuge entwickelt.
Anschließend wird mittels einer eingehenden Analyse evaluiert, welche Software für die Virtualisierung der Laborumgebung und Analyse der Applikation infrage kommt. Dieser Abschnitt beschäftigt sich auch mit den Anforderungen an die Testapplikation hinsichtlich ihrer Eigenschaften und Funktionen. Zudem wird erörtert, wie die Gestaltung der Nutzerdokumentation geschaffen sein muss und welche Systemvoraussetzungen durch die Basis- sowie Gastbetriebssysteme erfüllt werden müssen.
Aufbauend auf den im Zuge der Recherche und Analyse erarbeiteten Erkenntnissen erfolgt im Anschluss die Beschreibung der Implementierung des Systems und dessen Softwarebestandteile sowie der Entwicklung der Testapplikation. Im Zusammenhang mit der Implementierung der virtuellen Laborumgebung werden auch die Erstellung der Nutzerdokumentation sowie der Systemabbilder erläutert.
Außerdem wird betrachtet, wie das System in der Lehre genutzt werden kann, und eine experimentelle Evaluation des Systems vorgenommen.
Abschließend findet eine Bewertung unter Maßgabe des Ziels dieser Arbeit statt.

2. Grundlagen

Dieser Abschnitt befasst sich mit der Erarbeitung von theoretischen Grundlagen als Basis für die Erstellung der virtuellen Laborumgebung. Die Informationen, die für die Erstellung dieses Teils der Arbeit genutzt wurden, wurden in Form einer Recherche zusammengetragen und bezüglich ihrer Relevanz im Hinblick auf die Aufgabenstellung bewertet.

2.1. Virtuelle Systeme

Wie bereits unter Punkt 1.3 beschrieben, ist die Zielsetzung dieser Arbeit, ein virtuelles Labor zu erstellen, das der Analyse von mobilen Applikationen hinsichtlich ihrer Eigenschaften dienen soll. Im Folgenden werden eine Definition von Virtualisierung skizziert und Softwaretypen vorgestellt, die diese unterstützen. Anschließend werden Vorteile virtueller Umgebungen erläutert.

2.1.1. Defintion von Virtualisierung

IBM definiert Virtualisierung als einen "[...] Prozess, der eine effizientere Nutzung der physischen Computerhardware ermöglicht und die Grundlage des Cloud-Computings bildet" [4]. Dabei wird zwischen der Hardwareschicht des Hostsystems und dem Betriebssystem des virtuellen Rechners eine Softwareschicht implementiert, die die Kommunikation zwischen der sogenannten Virtual Machine (VM) und der physischen Systemhardware übernimmt. Virtuelle Systeme arbeiten voneinander und vom Basissystem unabhängig, unterstützt durch ihr systemeigenes Betriebssystem [4].
In den folgenden Abschnitten werden Kategorien von Software betrachtet, die zum Aufbau einer virtualisierten Umgebung genutzt werden.

2.1.2. Virtual Machine Monitor

Die unter Punkt 2.1.1 beschriebene Softwareschnittstelle stellt der sogenannte Virtual Machine Monitor (VMM), auch Hypervisor genannt, zur Verfügung. Dieser weist dem virtuellen System Hardwareressourcen, also Prozessorzeit und Speicherbereiche des zugrundeliegenden Systems, zu. Man unterscheidet zwei Arten von Hypervisoren:

Typ-1-VMM

Ein Typ-1-VMM setzt direkt auf der Hardwareschicht auf, übernimmt die Rolle eines Host-Betriebssystems und arbeitet im privilegiertesten Modus. Diese Variante wird auch als Bare-Metal-Hypervisor bezeichnet [5]. Der Virtual-Machine-Monitor emuliert alle Systemzugriffe der virtuellen Umgebungen auf die zugrundeliegende Hardware [6]. Die virtuellen Umgebungen operieren, von dieser Softwareschicht unterstützt und kontrolliert, auf der Hardware. Der schematische Aufbau eines entsprechenden Systems ist in Abbildung 2.1 dargestellt.

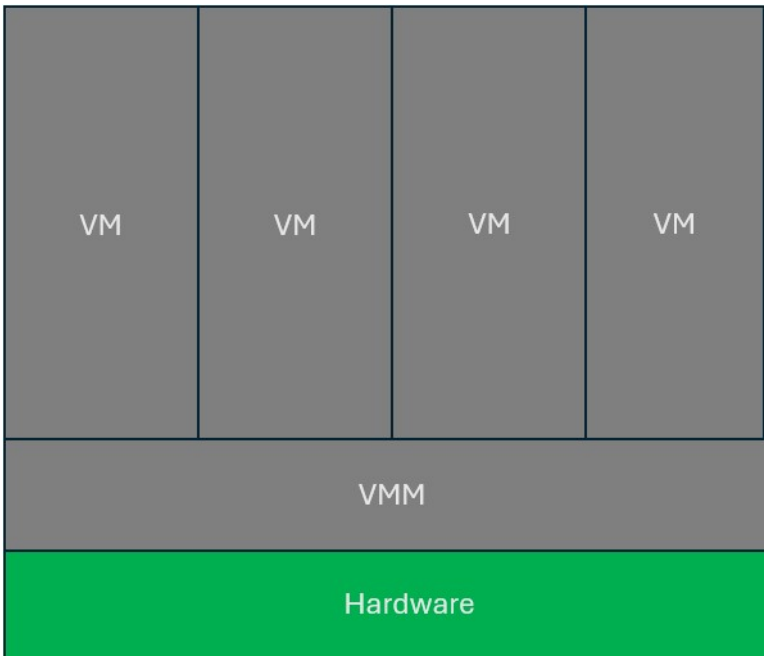

Abbildung 2.1.: Schematischer Systemaufbau VMM Typ-1

Typ-2-VMM

Ein Typ-2-VMM benötigt ein Host-Betriebssystem, das die Verwaltung der Systemressourcen für das virtuelle System übernimmt. Man bezeichnet diese Konfiguration als hosted [5]. Der Hypervisor wird vom Nutzer wie eine zusätzliche Software auf dem Basissystem installiert. Das Betriebssystem der Hostumgebung stellt die Treiber für den Zugriff auf die zugrundeliegende Host-Hardware zur Verfügung. Die Systemzugriffe auf die Hardwareressourcen des Basissystems werden durch das Host-Betriebssystem kontrolliert und organisiert. Ein schematischer Aufbau ist in Abbildung 2.2 dargestellt.

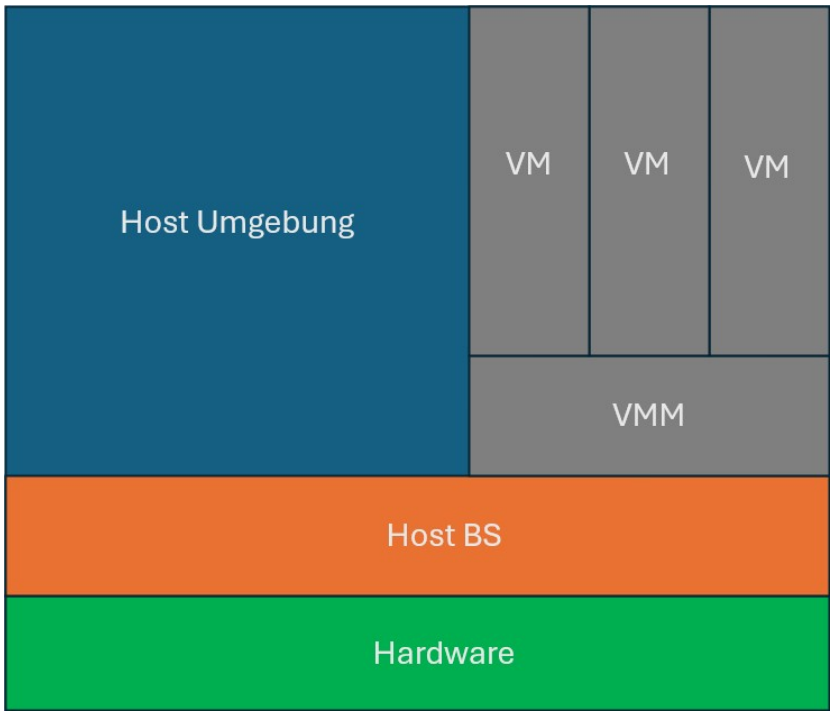

Abbildung 2.2.: Schematischer Systemaufbau VMM Typ-2

Auf Basis der Hypervisoren werden die virtuellen Maschinen ausgeführt. Im nächsten Abschnitt werden deren Aufgabe und ihre Systemauslegung im Dateisystem des Hosts dargestellt.

2.1.3. Virtuelle Maschinen

Virtuelle Maschinen simulieren, auf Basis der implementierten Softwarealgorithmen, ein physisches Rechensystem. Hypervisoren leiten die Aufgaben

an die, vom Hostsystem zur Verfügung gestellte, geteilte Hardware weiter. Die Konfiguration der virtuellen Umgebung, sowie der Speicherbereich für die virtuelle Festplatte, sind im Dateisystem des Hosts über entsprechende Unterordner dargestellt. Von virtuellen Maschinen können Snapshots erstellt werden, die ebenfalls im Ordner der VM abgelegt sind. Diese Snapshots entsprechen dem Status der virtuellen Umgebung zu einem bestimmten Zeitpunkt [5].

2.1.4. Containervirtualisierung

Leichtgewichtige Umgebungen zur Simulation von Systemen stellen Container dar. Diese benötigen keinen Hypervisor, der für sie die Systemzugriffe durchreicht. Container sind für die Ausführung einer bestimmten Applikation spezifiziert und nicht geeignet, unterschiedliche Programme in einer Instanz zu betreiben. Virtualisiert wird durch diesen Softwaretyp der Teil der Systemfunktionen eines Betriebssystems, die notwendig sind, um die Applikation zu betreiben [7]. Da Containerlösungen nicht für den Betrieb eines Portfolios an Analysesoftware, wie in den Abschnitten 3.1.5, 3.1.6 und 3.1.7 dargestellt, in Betracht kommen, wird diese Art der Virtualisierung nicht in die weiteren Überlegungen zur Erstellung des Laborsystems mit einbezogen.

Nachdem sich in den vorangegangenen Abschnitten mit unterschiedlichen Arten der Virtualisierung befasst wurde, werden in den folgenden Abschnitten Vorteile der Virtualisierung von Systemen betrachtet.

2.1.5. Vorteile virtueller Umgebungen

Systemauslegung

Unabhängig vom Typ des Hypervisors besitzen virtualisierte Systeme den Vorteil, dass sie hinsichtlich ihrer Konfiguration nicht durch die Auslegung des Hostsystems begrenzt sind [8]. Durch die in der Definition (vgl. 2.1.1) genannte Abstraktionsschicht, wird die flexible Auslegung eines Systems, aufsetzend auf dem Basissystem ermöglicht. Es können vollständige Systemumgebungen mit ihren Kommunikationsschnittstellen simuliert werden.

Neben der Simulation hardwarespezifischer Aspekte können unterschiedliche Betriebssystemvarianten virtualisiert genutzt werden.

Sicherheit

In Abschnitt 2.1.3 wurde bereits beschrieben, dass die Konfiguration sowie die gespeicherten Daten einer VM, auf dem Dateisystem des Hosts in Unterordnern abgelegt sind. Die Integrität dieser Dateien kann von Analyseprogrammen überwacht werden. Eine Erstellung von Systemsnapshots erlaubt die Wiederherstellung der Integrität des virtuellen Systems nach einer Infektion. Darüber hinaus kann eine virtuelle Umgebung unkompliziert gelöscht und schnell wieder aufgesetzt werden [5].
Virtuelle Umgebungen arbeiten voneinander und vom Hostsystem isoliert (vgl. 2.1.3). Die Prozesse, die in der VM ausgeführt werden, haben keinen Zugriff auf das Dateisystem des Hostsystems. Wenn ein infiziertes Programm auf einem virtuellen System ausgeführt wird, so wirkt sich das nicht auf andere VMs oder das übergeordnete System aus.
Diese Trennung kann nicht aufrechterhalten werden, wenn ein Angreifer den zugrundeliegenden Hypervisor kompromittiert. Nicht nur der Zugriff auf alle Gastsysteme ist dann möglich, auch der Zugriff auf das Hostsystem kann nicht ausgeschlossen werden [6].

Weitere Komponenten, die für die Implementierung einer Laborumgebung auf Grundlage eines virtualisierten Systems notwendig sind, werden in den folgenden Ausführungen beschrieben.

2.2. Softwarebestandteile eines virtuellen Mobile Security Labors

Zu einer Laborumgebung gehört neben der Virtualisierungssoftware und dem Gastsystem auch Analysesoftware. Um einen adäquaten Umfang an Analysemöglichkeiten in der virtuellen Laborumgebung bereitzustellen, wird zunächst betrachtet, welche Objekte analysiert werden sollen und mit welchen Methoden die Untersuchung erfolgen kann. Diesen Fragen widmen sich die folgenden Abschnitte.

2.2.1. Mobile Devices

Seit Mitte der 1990er Jahre ist die Nutzung von Mobiltelefonen kaum aus dem Alltag der Menschen wegzudenken. Während diese Geräte zunächst beschränkt waren auf die Nutzung zum Zwecke der Telefonie, wurde mit der Zeit der Nutzungsumfang schrittweise erweitert. Neben dem Versenden von Textnachrichten als Short-Message-Service (SMS) wurde auch das Versenden von Bildmitteilungen als Multimedia-Message-Service (MMS) möglich. Mit zunehmender Weiterentwicklung wurden mehr Anwendungsmöglichkeiten erschlossen. Heute können mit Hilfe moderner Mobiltelefone Fotos erstellt und z. B. via Messenger Services versendet oder auf Onlineplattformen wie Instagram, X oder Snapchat hochgeladen und bearbeitet werden. Navigationsapplikationen können, unterstützt durch den in mobile Geräte integrierten Global Positioning System (GPS)-Empfänger genutzt werden, um den Standort des Nutzers zu bestimmen oder ihn auf dem Weg zu seinem Ziel zu leiten. Darüber hinaus speichert ein Mobiltelefon nützliche Informationen, wie zum Beispiel die Telefonnummern und E-Mail-Adressen der Kontakte des Nutzers. Mobiltelefone sind über das Mobilfunknetz oder WLAN mit dem Internet verbunden. Die Nutzungsmöglichkeiten der Geräte können über den Download von Applikationen flexibel erweitert werden.
Mobiltelefone benötigen, wie andere Rechensysteme auch, ein Basisbetriebssystem. Geeignete Betriebssystemvarianten sind Android, iOS und Windows Mobile. Heute gibt es neben Mobiltelefonen auch größere mobile Devices, die als Tablets bezeichnet werden.
Da es inzwischen eine Vielzahl an mobilen Geräten und für diese entwickelte Applikationen gibt, gibt es damit einhergehend auch diverse Möglichkeiten, deren implementierte Mechanismen auszunutzen. Gerade deshalb ist es wichtig, dass sich Analysten mit der Sicherheit von mobilen Geräten und der für diese Plattformen entwickelten Applikationen auseinandersetzen.

Betriebssysteme von mobilen Geräten

Ebenso wie Personal-Computer benötigen mobile Geräte ein Betriebssystem, um die Nutzung systemkritischer Operationen zu organisieren. Häufig werden Android und iOS auf mobilen Plattformen genutzt (72,15 % bzw. 27,19 %). Windows Mobile hingegen wird nur selten als Betriebssystemvariante für mobile Geräte genutzt (0,02 %) [9].

Die Lehrveranstaltung Mobile Security der FernUniversität Hagen beschränkt sich, wie bereits unter 1 erwähnt, in der theoretischen Aufarbeitung auf die Analyse von Applikationen für die zwei am häufigsten genutzten Varianten iOS und Android. Die praktischen Übungen werden für die Analyse von Android Applikationen konzipiert. Da der Nutzungsumfang der Laborumgebung sich auf diesen Bereich konzentriert, wird der Fokus der Arbeit auf die Betrachtung von Android-Plattformen gelegt.

Android Betriebssysteme sind auf Linux basierende Open-Source Plattformen. Geschützt und organisiert durch die Android Runtime (ART), einer Art VM für den auszuführenden Applikationsprozess, können für Android konzipierte Applikationen auf systemkritische Operationen, wie die Zuteilung von Prozessorzeit oder das Speichermanagement, des zugrundeliegenden Linux-Kernels zurückgreifen. Der Hardware Abstraction Layer (HAL) stellt Applikationen über eine entsprechende Bibliothek Schnittstellen für den Zugriff auf periphere Hardwarekomponenten wie z. B. Kamera oder Bluetooth zur Verfügung [10].

Das Betriebssystem stellt Basisfunktionen zur Verfügung, die durch die Applikationen genutzt und erweitert werden. Wie Android-Applikationen aufgebaut sind und welche Programmiersprachen in ihre Entwicklung eingebunden werden können, beleuchtet der nächste Abschnitt.

2.2.2. Android Applikationen

Generell bestehen Applikationen für mobile Systeme ebenso wie Programme für Personal Computer (PC) aus mehreren Teilen, die in eine, für die Recheneinheit des ausführenden Geräts, interpretierbare Form übersetzt und zu einer ausführbaren Datei gepackt werden.
Der Code von für die Plattform Android konzeptionierten Applikationen wird in den Sprachen Java oder Kotlin verfasst. Nach Mateus und Martines ist Kotlin eine Weiterentwicklung von Java, die objektorientierte und funktionale Programmierkonzepte vereint[11]. Die Verwendung von nativen Bibliotheken, die in C oder C++ geschrieben wurden, wird ebenfalls unterstützt.
Applikationscode, der für die Ausführung auf Androidversionen ab 5.0

(Lollipop) erstellt wird, wird für die Ausführung in der Laufzeitumgebung ART übersetzt. Aus Gründen der Aktualität wird die Betrachtung auf Applikationen, die für den Betrieb in der ART konzipiert sind, beschränkt [10]. Zum Zeitpunkt der Erstellung der Arbeit stellt Android 14 - Application Programming Interface (API) Level 34 - die aktuellste Release-Version dar. Android 15 (API Level 35) kann derzeit als Beta-Version getestet werden [12].
Elementare Teile einer Android Applikation im Hinblick auf die Analyse, bezogen auf sicherheitskritische Auffälligkeiten, sind unter anderem:

- Das Android Manifest, das Informationen über die für den Betrieb der Applikation notwendigen Berechtigungen sowie über Applikationskomponenten wie Activities, Services und Content-Provider enthält [13].
- Das .dex-File, das den kompilierten ausführbaren Code der Applikation enthält [1, S.5].

Zur Entwicklung mobiler Applikationen für Android-Plattformen stehen entsprechende Werkzeuge in Form von Entwicklungsumgebungen zur Verfügung. Diese werden im Folgenden näher betrachtet.

2.2.3. Entwicklungsumgebungen

Entwicklungsumgebungen, auch Development Tools genannt, unterstützen Entwickler, indem sie Basisfunktionen zur Erstellung von Applikationen, wie sprachspezifische Compiler und die Möglichkeit zur Einbindung von Standardbibliotheken, zur Verfügung stellen [14]. Werkzeuge, die eine statische Analyse von Applikationscode unterstützen, greifen zum Teil auf Funktionen zurück, die durch Entwicklungsumgebungen zur Verfügung gestellt werden.
Integrierte Entwicklungsumgebungen (IDE) ergänzen diese grundlegenden Hilfsmittel. Sie sind mit einer grafischen Benutzeroberfläche ausgestattet und unterstützen Syntaxhighlighting, das die statische Analyse des Codes erleichtert. Zudem ermöglichen IDE die simulierte Ausführung des Codes auf Emulatorinstanzen, um dynamische Fehleranalysen zur Laufzeit durchzuführen [15].

Die Möglichkeit der Nutzung von Emulatoren im Rahmen der dynamischen Analyse wird im nächsten Abschnitt näher beschrieben.

2.2.4. Emulatoren

Um Applikationen zur Laufzeit testen zu können, müssen mobile Plattformen simuliert werden. Dies geschieht mit Hilfe von Emulatoren. Emulatoren bieten eine Testumgebung für unterschiedliche Plattformen und Versionen. Die Möglichkeiten der Simulation umfassen dabei Funktionen wie z. B. GPS-Unterstützung oder die Emulation von Kameras und Geschwindigkeitssensoren [16]. Unterstützt durch Emulatoren kann das Verhalten von Programmen zu ihrer Laufzeit auf unterschiedlichen Plattformversionen unter Simulation verschiedener Hardwarekonfigurationen analysiert werden [17].

2.2.5. Analysemethoden

Um geeignete Analysemethoden für die Prüfung von Android-Applikationen im Hinblick auf ihre sicherheitskritischen Bestandteile identifizieren zu können, wird zunächst betrachtet, welche Schwachstellen Applikationen für mobile Geräte aufweisen können. Darauf aufbauend werden geeignete Methoden zur Untersuchung identifiziert.

Schwachstellen

Die für Anwender unter 2.2.1 aufgeführten nützlichen Eigenschaften von mobile Devices bieten Ansatzpunkte für Angriffsvektoren. Die Geräte verbinden sich über Funk mit Netzwerken. Daten, die auf diesem Weg ausgetauscht werden, können während des Informationstransports abgefangen oder manipuliert werden. Darüber hinaus kann die Funktion von Hintergrundkommunikation zu externen Instanzen durch Applikationen (Apps) implementiert sein, ohne dass dies für den Nutzer nachvollziehbar ist.
Die Implementierung von exported Activities erlaubt die programmübergreifende Nutzung von Funktionen. Über diesen Mechanismus sind Datenzugriffe möglich, die dem Anwender unter Umständen nicht bewusst sind [18]. Kritisch zu betrachten ist ebenfalls eine nicht dem Nutzungsumfang

der Applikation entsprechende, erweiterte Abfrage von Berechtigungen. Diese kann ebenfalls den Zugriff auf sensible Nutzerdaten durch Dritte ermöglichen. Über exported Content-Provider werden Zugriffe auf applikationsinterne Datenbereiche erlaubt. Oft sind mit diesen Content-Providern SQLite-Datenbanken[1] verbunden. Potenziell können diese eine Anfälligkeit für SQL-Injektion aufweisen [1, S. 86 ff.].
OWASP führt unter den Top 10 der Risiken bei fehlerhafter Implementierung von Funktionen mobiler Applikationen unter anderem unsichere Kommunikation mit remote Servern, nicht ausreichende Validierung von Dateneingaben und Ausgaben und fehlerhafte Implementierung der Verwaltung und Überprüfung von Zugangsdaten bzw. Schlüsseln auf [20].
Diese Aufzählung von Schwachstellen ist nur beispielhaft und zeigt nur einen geringen Teil der Mechanismen auf, die durch Dritte genutzt werden können, um an sensible Daten zu gelangen oder mobile Systeme anderweitig zu kompromittieren. Eine umfassende Aufführung würde den Umfang der Arbeit sprengen.

Manche Schwachstellen können bereits durch eingehende Betrachtung des Applikationscodes erkannt werden, andere wiederum können nur während der Ausführung des Codes entdeckt werden. Welche Analysemethoden generell geeignet sind Schwachstellen möglichst umfassend zu identifizieren, behandeln die nächsten Abschnitte.

Analysemethoden

Die statische Analyse von Applikationen, auch Reverse Engineering genannt, umfasst Techniken, die eine Analyse der Programme ohne eine Ausführung des Codes ermöglichen. Um die statische Analyse des Quellcodes einer Applikation durchführen zu können, muss der ausführbare Code der Applikation zunächst dekompiliert werden. Danach kann im Quellcode nach Mustern gesucht werden, die auf unerwünschtes Verhalten des Programms zur Laufzeit hindeuten. Diese Betrachtung kann manuell durch den Analysten oder automatisiert durch Programme geschehen. Bei der automatisierten Analyse des Quellcodes können nur bekannte Muster erkannt werden. Neue Muster können durch Experten oder automatisiert, unter

[1] SQLite-Datenbanken sind prozessinterne, leichtgewichtige Bibliotheken, die eine programminterne Datenverwaltung auf mobilen Plattformen erlauben [19]

Nutzung von Methoden des maschinellen Lernens, identifiziert werden. Wenn bei der Erstellung von Schadcode Verschleierungstechniken (engl. Obfuscation) oder polymorpher Code genutzt werden, wird die statische Analyse zusätzlich erschwert. Eine Verschlüsselung des Quellcodes macht die statische Analyse einer Applikation ohne Kenntnis des Schlüssels unmöglich [1, S. 39].

Bei der dynamischen Analyse von Applikationen wird der Applikationscode in einer kontrollierten Umgebung, der Sandbox, ausgeführt. Die Sandbox entspricht der Simulation eines ausführenden Gerätes und arbeitet, wie eine VM, isoliert vom Hostsystem. Während der Ausführung des Programms wird das Verhalten automatisiert protokolliert. Das Protokoll kann anschließend von Experten ausgewertet und interpretiert werden. Die Reaktion einer Applikation auf externe Ereignisse kann im Rahmen einer dynamischen Analyse nur dann beobachtet werden, wenn auslösende Ereignisse, wie z. B. das Senden einer SMS, simuliert werden [1, S. 40].
Applikationen, deren Code verschlüsselt vorliegt, können über die Auswertung ihres Verhaltens zur Laufzeit inspiziert werden. Zudem liegt der Code zur Ausführungszeit unverschlüsselt vor. Auch die Anwendung von Verschleierungstechniken kann bei der dynamischen Analyse von Software nicht verhindern, dass das Verhalten des Programms in Ausführung beobachtet und ausgewertet werden kann. Es gibt jedoch Techniken, gängige Sandboxes zu erkennen und das Verhalten der Applikation in derartigen Umgebungen entsprechend anzupassen, um den vollen Funktionsumfang oder schadhaftes Verhalten zu verbergen [1, S. 40].

2.3. Zusammenfassung

In diesem Kapitel wurde durch eingehende Recherche ein grundlegender Überblick darüber geschaffen, wie virtuelle Umgebungen funktionieren und welche Vorteile virtualisierte Systeme im Hinblick auf die Analyse von mobilen Applikationen auf diversen Host-Betriebssystemen bieten. Darüber hinaus wurde erörtert, welche grundlegenden Eigenschaften die zu analysierenden Objekte, mobile Applikationen, aufweisen, und welche Methoden genutzt werden können, um diese eingehend auf Schwachstel-

len hin zu untersuchen. Dieser oberflächlichen Betrachtung anschließend beschäftigt sich der folgende Teil der Arbeit, auf Basis der zusammengetragenen Grundlagen, mit einer detaillierten Analyse der Bestandteile und Konfiguration des Mobile Security Labors.

3. Analyse

In den folgenden Abschnitten wird beschrieben, wie die Entscheidung für eine Software zur Unterstützung der Virtualisierung und das Basisbetriebssystem des virtuellen Gastsystems getroffen wurde. Darüber hinaus wird beleuchtet, auf welche Weise Entwicklungsumgebungen, Emulatoren und entsprechende Analysesoftware für die Untersuchung der Testapplikation ausgewählt wurden. Probleme, die bei der Entwicklung der Testumgebung erkannt wurden, sowie Lösungsoptionen werden dargestellt.
Anschließend wird diskutiert, welche Eigenschaften die Testapplikation aufweisen muss, um Erkenntnisse aus deren Analyse zu erlauben, die dem Lehrziel dienen.
Darüber hinaus wird beleuchtet, wie die Nutzerdokumentation gestaltet sein muss, um den Studierenden ausreichend Hilfestellung zu bieten.
Abschließend wird betrachtet, welche Systemvoraussetzungen die Hostsysteme der Nutzer und die VMs bieten müssen, damit das Mobile Security Labor betrieben werden kann.

3.1. Softwareauswahl

Um Kandidaten bewerten zu können, wird zunächst ein Kriterienkatalog erstellt, der zur Evaluierung von Software im Allgemeinen geeignet ist. Dabei werden in den Unterpunkten die Ausführungen auf relevante Informationen beschränkt. Unterscheiden sich die Kandidaten, bezogen auf ein Kriterium, nicht, wird auf die Aufzählung verzichtet. Sollten für einen Softwaretyp spezifische Kriterien mit berücksichtigt werden müssen, so werden diese im jeweiligen Unterpunkt definiert.

3.1.1. Kriterien für die Softwareauswahl

- Funktionalität:
  Software muss in der Lage sein, die durch die Aufgabenstellung gefor-

derten Funktionalitäten bereitzustellen. Die Laborumgebung muss den Studierenden die Ausführung von Software zum Zwecke der Durchführung der im Rahmen der Übungen geforderten Analysen ermöglichen.

- Integrationsmöglichkeit in bestehende Systeme:
  Die Rechner der Studierenden besitzen unterschiedliche Basisbetriebssysteme, daher sollte das virtuelle Labor auf Linux-, Windows- und MacOS-Betriebssystemen stabil betreibbar sein.

- Sicherheit:
  Um den Schutz der Integrität des Basissystems, auf dem die Laborumgebung aufgesetzt wird, zu gewährleisten, sollte die Aktualisierung der implementierten Software durch regelmäßige Updates unterstützt werden.

- Benutzerfreundlichkeit:
  Den Studierenden soll wie bereits unter 1.1 beschrieben eine Umgebung zur Verfügung gestellt werden, mit der sie die Analysen im Rahmen des Kurses Mobile Security durchführen können, ohne dass eine eingehende Einarbeitung in die Funktion des virtuellen Systems und umfangreiche Konfigurationsmaßnahmen notwendig sind.

- Kostenbetrachtung:
  Kostenpflichtige Softwarevarianten oder Programme, die nur eine bestimmte Zeit lang kostenfrei genutzt werden können, tragen zu einer Erhöhung des Verwaltungsaufwandes und der für die Bearbeitung des Kurses entstehenden Kosten bei.

3.1.2. Virtualisierungssoftware

Wie bereits in 2.1.2 beschrieben, ist die grundlegende Softwarebasis einer virtuellen Umgebung der Virtual Machine Monitor oder Hypervisor. Er vermittelt zwischen der virtuellen Maschine und dem Basissystem. In den folgenden Abschnitten werden VMM-Kandidaten anhand der Kriterien zur Softwareauswahl bewertet.

Evaluierung der Kandidaten

Die Studierenden sollen das virtuelle Laborsystem auf ihren Rechnern installieren können. Da in Betrieb befindliche PC und Laptops bereits ein

Basisbetriebssystem installiert haben, kommen für die Nutzung von virtuellen Umgebungen auf diesen Plattformen Typ-2-Hypervisoren in Frage. Bekannte Kandidaten des Hypervisortyps 2 sind die Hypervisorvarianten von VMware und Oracle VirtualBox. Aufgrund der geforderten Flexibilität bezüglich der Unterstützung diverser Hostbetriebssysteme werden für bestimmte Betriebssystemvarianten spezifizierte Virtualisierungslösungen wie Windows Hyper-V, Linux Kernel-based Virtual Machine (KVM) und Parallels Desktop für MAC sowie UTM[1] von der Betrachtung ausgeschlossen.

Die neuesten Versionen von VirtualBox sowie VMware sind, unabhängig vom Hostbetriebssystem, nur auf 64-Bit-Plattformen lauffähig.
VirtualBox 7.X ist auf Windows-, Linux- und MacOS-Plattformen lauffähig. Darüber hinaus werden auch Solaris- und OpenSolaris-Plattformen unterstützt. VirtualBox 6.X kann auf Windows-Versionen ab 7 aufwärts genutzt werden. Auf Hostsystemen mit Windowsversionen ab 8.1 kann VirtualBox 7.X betrieben werden. Generell werden durch VirtualBox Linuxkernel ab Version 2.6 unterstützt. Voraussetzung für die Lauffähigkeit von VirtualBox auf MacOS-basierten Systemen ist eine Intel-Hardwareplattform. MacOS-Varianten ab 10.13 (High Sierra) werden durch VirtualBox-Versionen bis 6.1, ab MacOS Catalina (10.15) durch die Versionen 7.X unterstützt [21].
VMware Workstation ist auf den Windows-Versionen 8.1 bis 11 und Linux Plattformen nutzbar. Die Version Windows 8.1 ist auf VMware-Versionen bis 16.2.X lauffähig. Windows 11 Host-Systeme werden ab Version 16.2.X unterstützt. VMware Workstation ist außerdem auf Linux-basierten Betriebssystemvarianten von Ubuntu, Red Hat Enterprise Linux, CentOS, Oracle Linux und openSUSE lauffähig [22]. Darüber hinaus unterstützt VMware Fusion die Virtualisierung auf MacOS-Plattformen [23]. Auf MacOS-Systemen, die auf einer ARM-Architektur basieren, ist weder VirtualBox noch VMware lauffähig.
Diese Aufzählung entspricht nicht der Vollständigkeit. Auf die Betrachtung der Nutzbarkeit auf Serverbetriebssystemen wird aufgrund fehlender Relevanz im Hinblick auf die Aufgabenstellung verzichtet.

Wenn Schwachstellen erkannt werden, werden sowohl durch VMware als auch durch Oracle zeitnah Aktualisierungen angeboten [24] [25]. Für VirtualBox 7.X gibt es regelmäßige Updates sowie Upgrades alle drei Monate [26].

[1] UTM ist ein Hypervisor auf Quemu-Basis für Apple-Betriebssysteme

Ältere Versionen auf Basis von 6.0.X werden nicht mehr durch Updates unterstützt [27]. Für VMware Workstation 17.X gibt es ebenfalls turnusmäßig Aktualisierungen sowie Upgrades alle zwei bis fünf Monate [28]. Bekannte Schwachstellen von Software sind in der Common Vulnerabilities and Exposures (CVE) Datenbank aufgeführt. Ein Eintrag aus dem Jahr 2020 weist auf die Schwachstelle CVE-2020-6100 hin, die einen Guest-to-Host-Escape[2] begünstigen kann [29]. Diese Möglichkeit besteht laut CVE ausschließlich bei AMD-Prozessoren und einer bestimmten Treibervariante und betrifft den Betrieb von virtuellen Umgebungen generell. Diese Schwachstelle ist also nicht einer bestimmten Typ-2-Hypervisorvariante zuzuordnen.
Die Funktion Secure-Boot, die eine zusätzliche Barriere für Angriffe auf die Integrität des Gastsystems darstellt, wird sowohl durch VMware als auch durch VirtualBox zur Verfügung gestellt. Voraussetzung für die Nutzung dieses Mechanismus ist, dass auch das entsprechende Gastbetriebssystem diese Möglichkeit unterstützt.
VMware Workstation sowie Fusion werden zur nicht kommerziellen Nutzung kostenfrei angeboten [30]. Um den Download der Software vornehmen zu können, muss eine Registrierung auf der Broadcom-Website erfolgen.
Varianten von VirtualBox werden unter einer GNU-Public-License kostenfrei zur Verfügung gestellt [31] [32].
Im Überblick sind Gemeinsamkeiten und Unterschiede der aktuellsten VMware- und VirtualBox-Varianten vereinfacht in Abbildung 3.1 dargestellt.

Typ-2 Hypervisor-Variante	Unterstützte Hostbetriebs-systeme	Updates / Upgrades	Secure-Boot Unterstützung	Registrierung notwendig
VMware 17.X	Linux-, MacOS- und Windows-Varianten	Zeitnah / regelmäßig	Ja	Ja
VirtualBox 7.X	Linux-, MacOS- und Windows-Varianten	Zeitnah / regelmäßig	Ja	Nein

Abbildung 3.1.: Tabelle Eigenschaften Typ-2 Hypervisor-Varianten

Nachdem die für die Auswahl einer Hypervisorvariante relevanten Informationen in den vorhergehenden Abschnitten zusammengetragen wurden, wird im nächsten Abschnitt begründet, welche VMM-Variante als Basis für den Betrieb des Mobile Security Labors genutzt wird.

[2]Guest-to-Host-Escape nennt man eine Manipulation des Host-Systems durch ein virtuelles Gastsystem

Auswahl der Hypervisorvariante

Im Hinblick auf die Aufgabenstellung, die die Nutzung der VM durch Studierende ins Zentrum stellt, bieten die Varianten der Oracle VirtualBox ein breites Portfolio an unterstützten Hostbetriebssystemen, sowie eine kostengünstige Lösung. Für die Nutzung von VirtualBox ist keine Registrierung bei einem Anbieter notwendig.
Die neueste Version VirtualBox 7.X ist auch auf Windows-11-Systemen lauffähig. Somit werden die Windowsvarianten 8.1 bis 11 als Hostsysteme unterstützt. Ältere Versionen von VirtualBox kommen aus Sicherheitsgründen nicht in Frage, da diese nicht mehr durch Updates unterstützt werden. Als VMM-Plattform für das Mobile Security Lab wird daher die VirtualBox Version 7.0.18 gewählt, die zum Zeitpunkt der Erstellung der Arbeit die aktuellste Variante darstellt.

3.1.3. Gastbetriebssystem

Basis der Laborumgebung ist das zugrundeliegende Gastsystem, das mit Unterstützung der Virtualisierungssoftware simuliert wird. In den folgenden Abschnitten werden relevante Informationen über Betriebssystemkandidaten zusammengetragen und eine Bewertung anhand des Kriterienkatalogs durchgeführt.

Evaluierung der Betriebssystemkandidaten

Beim Vergleich von virtualisierten Windows 11, Windows 10 und Ubuntu-22.04-Betriebssystemen ist auffällig, dass die Ubuntu Variante, den Speicherplatz betreffend, ressourcenschonender zu betreiben ist [33] [34] [35] [36]. Die Systemanforderungen unterschiedlicher Varianten im Überblick zeigt Abb. 3.2.

Betriebssystem Variante	Win 11 Developer	Win 11 Home / Pro	Win 10 Home / Pro	Ubuntu 22.04
Prozessor	-	1 GHz	1 GHz	2 GHz / Dual Core
Arbeitsspeicher	8 GB	4 GB	2 GB	4 GB
Festplattenspeicher	70 GB	> 64 GB	32 GB	25 GB

Abbildung 3.2.: Tabelle Systemanforderungen

Die Version Ubuntu 20.04 LTS Focal Fossa wird bis April 2025 und Ubuntu 22.04 LTS Jammy Jellyfish bis April 2027 im Standardsupport betreut. Darüber hinaus gibt es für beide Varianten einen um fünf Jahre verlängerten Pro-Support [37]. Der Suport für Windows 10 wird im Oktober 2025 eingestellt [38], für Windows 11 Versionen Home und Pro ist noch kein Enddatum für den Support angegeben [39].
Aktuelle Windowsversionen können 90 Tage lang als kostenfreie Testversionen genutzt werden. Ubuntu-Varianten sind kostenfrei verfügbar.
Bei der Verwendung von Analysetools wie Drozer wurden im Rahmen der Durchgeführten Tests Probleme in Zusammenhang mit der neuesten Ubuntu-Version - 24.04 Noble Numbat - erkannt. Diese Version wurde deshalb nicht bei den weiteren Überlegungen berücksichtigt.

Auswahl der VM-Abbildvariante

Der Vergleich zeigt, dass Ubuntu 22.04 Jammy Jellyfish ressourcenschonend und kostenfrei auf Hostsystemen zu betreiben ist. Diese Variante bietet eine ausreichende Supportdauer für den Betrieb als Gastsystem der Laborumgebung. Darüber hinaus sind Ubuntu Betriebssystem-Varianten in ihrer kostenfreien Nutzungsdauer nicht eingeschränkt. Für die Erstellung des virtuellen Mobile Security Labors wurde daher Ubuntu 22.04 Jammy Jellyfish als Gastbetriebssystem für die Labor-VM gewählt.

3.1.4. Herausforderungen

Auch wenn die derzeit aktuellste Variante von VirtualBox ein breites Portfolio an Host-Betriebssystemen unterstützt und die gewählte Ubuntu-Version die Nutzung der Analysetools unterstützt, schließt das nicht aus, dass es auf unterschiedlichen Softwareplattformen und zugrundeliegenden Hardwarekonfigurationen zu Problemen beim Betrieb der virtuellen Umgebung kommen kann. Welche Herausforderungen bei den Systemtests identifiziert wurden, beleuchten die nächsten Abschnitte.

Methodik

Um Herausforderungen für den Betrieb der Laborumgebung auf unterschiedlichen Betriebssystemversionen sowie Prozessorvarianten auszuschließen, wurde die Lauffähigkeit auf heterogenen Systemkonstellationen getestet.

Die Labor-VM wurde auf den Hostbetriebssystemvarianten Windows 10, Windows 11, Ubuntu 22.04 Jammy Jellyfish und MacOS (Intel Plattform) Ventura 13.6.7 ausgeführt. Dabei wurden bei den Tests auf Windows-Betriebssystemen Rechnervarianten mit AMD-Ryzen-Prozessoren sowie mit Intel-Pentium-Prozessoren verwendet, um unterschiedliche Hardwarekonfigurationen der Hostsysteme mit einzubeziehen. Die Tests erfolgten schichtenweise, so dass zunächst die Funktion der Virtualisierungssoftware mit den unterschiedlichen Systemvarianten überprüft wurde. Anschließend wurden VM-Varianten mit Implementierung von Secure-Boot und ohne diese Funktion getestet, um die Auswirkungen dieses Sicherheitsfeatures auf die Lauffähigkeit zu bestimmen. Danach wurde die VM mit der Analysesoftware bestückt und entsprechenden Tests unterzogen, um sowohl die Lauffähigkeit auf unterschiedlichen Systemvarianten als auch ihre Eignung im Hinblick auf die Durchführung entsprechender Analysen, wie im Rahmen des Kurses Mobile Security gefordert, zu evaluieren.

Einschränkungen bezüglich der Virtualisierung

Bei der Erstellung der virtuellen Laborumgebung mit der Fähigkeit, über Secure Boot zu starten, ist die Übertragbarkeit des VM-Abbildes auf andere Systeme durch die Signatur eingeschränkt. Daher wurde auf die Implementierung von Secure Boot verzichtet, damit die Übertragbarkeit der Abbilder auf die Systeme der Studierenden gewährleistet ist. Die Umgebung dient den Studierenden dazu, die Übungsinhalte des Kurses anhand der Analyse einer Applikation, deren Funktionen durch die Lehrenden implementiert wurden, zu vertiefen. Damit ist eine Gefahr für die Kompromittierung des Systems mit Hilfe der in der Testapplikation implementierten Funktionen nicht gegeben.
Die Tests haben keine Einschränkungen bezüglich der Lauffähigkeit von VirtualBox auf den Hostsystemvarianten Windows 10 und 11 aufgezeigt. Bei der Virtualisierung auf MacOS- und Linux-basierten Hostsystemen, die den Secure-Boot-Modus aktiviert haben, kann es zu Problemen beim Betrieb von virtuellen Umgebungen kommen. Die Problematik kann durch Deaktivierung des Secure-Boot-Modus im BIOS des Hostsystems behoben werden.
Im Rahmen der Ausarbeitung wurde außerdem festgestellt, dass es nach der unbeaufsichtigten Installation des VM-Abbildes dazu kommen kann,

dass sich kein Terminalfenster in der virtuellen Umgebung öffnen lässt. Dieses Problem kann umgangen werden, indem die Installation der VM im Modus einer beaufsichtigten Installation durchgeführt wird.
Nach Einbindung der Gasterweiterungen sind Probleme bezüglich der Nutzung von Drag&Drop bzw. Copy&Paste Funktionen sowie der gemeinsamen Nutzung von Ordnern zwischen Host- und Gastsystem aufgetreten. Da im Rahmen der Analyse von Applikationen mit potenziell ungewünschtem Verhalten diese Funktionen ohnehin Potenzial für eine Kompromittierung des Hostsystems darstellen können [40] wird dieser Umstand nicht als Einschränkung gewertet.
Wenn VirtualBox VMs und Hyper-V gleichzeitig auf dem Hostsystem ausgeführt werden, kann es zu Konflikten kommen [41]. Probleme können außerdem entstehen, wenn der Secure Virtual Machine Mode nicht im BIOS aktiviert ist.
Die Emulation, unterstützt durch Hardwareacceleration, ist innerhalb einer virtuellen Umgebung nicht möglich [42]. Sowohl bei AMD- als auch Intel-basierten Plattformen konnte beobachtet werden, dass der VM keine Hardwareacceleration ermöglicht wurde, auch wenn VT-x oder AMD-V[3] sowie Nested Hardware Virtualization[4] aktiviert waren. Dieses Problem wurde durch die Erarbeitung einer Lösung mit zwei virtuellen Umgebungen, wie unter 3.1.6 beschrieben, gelöst.

Lösungsmöglichkeiten für Probleme, die zu Einschränkungen bei der Nutzung der Laborumgebung führen können, werden in der Nutzerdokumentation dargelegt. Auf sicherheitskritische Einschränkungen wird durch Warnhinweise in der Dokumentation aufmerksam gemacht.

3.1.5. Entwicklungsumgebungen

Um statische und dynamische Analysen an der Testapplikation zu unterstützen, muss die Laborumgebung bestimmte Entwicklungsumgebungen als Basis für die Funktionsweise der Analysesoftware implementieren. Die Auswahl der Entwicklungsumgebungen wird im Folgenden beschrieben.

[3] Virtualisierungsfunktionen von Intel- bzw. AMD-Prozessoren
[4] Funktion, die die Ausführung eines Hypervisors innerhalb einer virtuellen Maschine ermöglicht

Java Development Kit

Für die Entwicklung von Java-Code stellt Oracle das Java Development Kit (JDK) zur Verfügung. Diese Entwicklungsumgebung gibt es in einer proprietären (JDK) sowie Open-Source-Variante (OpenJDK). Oracle JDK und OpenJDK können seit JDK 11 analog genutzt werden [43] .
OpenJDK wird von Oracle, Red Hat und freien Entwicklern betreut. Der Vorteil von offen zugänglichem Code ist, dass auch Entwickler, die nicht direkt mit der Entwicklung des Sourcecodes betraut sind, die Implementierung unter unterschiedlichen Gesichtspunkten überprüfen können und somit Fehler oder Schwachstellen erkennen. Dies ist bei proprietärer Software wie Oracle JDK nicht der Fall. Ein Vergleich beider Varianten bei CVE zeigt, dass mehr Schwachstellen bei OpenJDK als bei Oracle JDK identifiziert und damit auch behoben werden konnten [44] [45].
Die JDK-Versionen 8, 11, 17, 21 und 25 sind Langzeitsupport-Versionen (LTS). JDK 17 wird im Premier Support bis September 2026 unterstützt, JDK 21 bis September 2028 und JDK 25 bis September 2030 [46]. OpenJDK 8 wird bis November 2026, OpenJDK 11 bis Oktober 2024, OpenJDK 17 bis Oktober 2027 und OpenJDK 21 bis Dezember 2029 im Full Support betreut [47].
Das JDK und OpenJDK stellen Basisfunktionen sowohl für die im folgenden Abschnitt beschriebene Android Entwicklungsumgebung als auch für einige Analysetools, wie Jadx und Drozer, die in Abschnitt 3.1.7 näher beschrieben werden, zur Verfügung. Jadx und Drozer benötigen die Unterstützung durch JDK 11 oder eine aktuellere Version [48] [49].
Unter anderem ist für die unkomplizierte und sichere Anwendung entscheidend, dass der Download des JDK und die Aktualisierung aus Paketquellen reibungslos darstellbar sind. Auf Linux-Betriebssystemen, wie Ubuntu 22.04 Jammy Jellyfish, erlaubt OpenJDK die Installation über einen Paketmanager. JDK unterstützt diese Installationsvariante für Debian Versionen nicht [50]. In Abbildung 3.3 wird eine Übersicht an Paketquellen, die für Ubuntu-Betriebssysteme zur Verfügung stehen, gezeigt [51].

Ubuntu Version	OpenJDK 11	OpenJDK 13	OpenJDK 16	OpenJDK 17	OpenJDK 18	OpenJDK 19	OpenJDK 21
20.04 Focal Fossa	✓	✓	✓	✓	✗	✗	✗
22.04 Jammy Jellyfish	✓	✗	✗	✓	✓	✓	✓

Abbildung 3.3.: Ubuntu Paketquellen für OpenJDK

Android Development Tools

Die Android Development Tools (ADT) stellen Entwicklern eine integrierte Entwicklungsumgebung mit entsprechenden Werkzeugen zur Erstellung und zum Testen von Android-Apps zur Verfügung. Android-Studio kann von Anwendern über eine grafische Schnittstelle bedient werden. Eine weitere Möglichkeit zur Untersuchung von Android-Applikationen bieten die sogenannten Commandline-Tools. Wie sich aus dem Namen ableiten lässt, unterstützen diese keine grafische Nutzerschnittstelle und sind damit ressourcenschonender zu betreiben als Android-Studio. Die Commandline-Tools implementieren alle für die Einrichtung des Labors und im Rahmen der Übungen geforderten Funktionen.

Auswahl

Für die Implementierung in der Labor-VM wurde die OpenJDK-Version 17 ausgewählt. Für Linux-Plattformen werden die Installation und Updates über die Repositories angeboten. Der Code dieser Anwendung wird regelmäßig von der Community auf Schwachstellen überprüft. Außerdem wird diese Version noch bis Oktober 2027 supported. Analysewerkzeuge wie Jadx werden durch OpenJDK 17 aufgrund der ausreichenden Aktualität unterstützt. Um die Ressourcen der Hostsysteme zu schonen, werden die Commandline-Tools anstelle von Android-Studio in der Labor-VM implementiert.

3.1.6. Emulator

Bereits im Rahmen der Beschreibung der Grundlagen wurde unter Punkt 2.2.1 dargelegt, wie sich der Nutzungsumfang von mobilen Geräten seit

den 1990er Jahren entwickelt hat. Um die offensichtlichen und verborgenen Funktionen von Applikationen zur Laufzeit zu testen, muss der Nutzungsumfang mobiler Devices simuliert werden können. Diese Aufgabe übernehmen, wie unter 2.2.4 dargestellt, Emulatoren. Die Auswahl eines Emulators wird in den folgenden Abschnitten diskutiert.

Emulation mit Unterstützung der Android Development Tools

Die ADT bieten unterschiedliche Plattformversionen zur Emulation von Android-Instanzen an. So kann Applikationscode auf älteren sowie aktuellen Android-Plattformen zur Laufzeit auf seine Funktion getestet werden. Applikationscode, der für neuere Plattformen entwickelt wird, unterliegt entsprechend angepassten Sicherheitsanforderungen. Bei Code, der für Android 34 (API Level 34) oder höher entwickelt wurde, kann z. B. nicht auf implizite Intents zur App-internen Nutzung zurückgegriffen werden [52]. Die neueste Plattform die durch ADT zur Emulation zur Verfügung gestellt wird, ist 34.2.16 [53].
Die Tests zeigten, wie bereits unter 3.1.4 erwähnt, dass die Ausführung eines über die ADT installierten Emulators in der Labor-VM nicht stabil möglich ist.

Weitere Möglichkeiten zur Emulation

Eine weitere Möglichkeit zur Emulation von Android-Geräten stellen containerbasierte Softwarelösungen wie Genymotion, Waydroid, Anbox oder Memu dar. Auch diese Softwarevarianten waren nicht in der virtuellen Ubuntu-Umgebung lauffähig.
Darüber hinaus kann mit Hilfe von VirtualBox und einem Android-Systemabbild eine Android-VM erstellt werden, die für die Simulation zum Zwecke der dynamischen Analyse genutzt werden kann.

Auswahl

Eine Lösung über die Ausführung zweier virtueller Umgebungen wurde erarbeitet. Eine Ubuntu-VM, die zur Ausführung der Analysesoftware dient, ist dabei über ein NAT-Netzwerk mit einer Android-Umgebung verbunden. Beide Umgebungen werden über VirtualBox direkt auf dem Hostsystem betrieben und müssen somit keine Nested Hardware Virtualization nutzen.

Diese Lösung wird unter 4.1.2 näher beschrieben.

Für die Analyse von Android-Software ist neben den Entwicklungsumgebungen, die Basisfunktionen zur Analyse des Codes zur Verfügung stellen, und den Emulatoren, entsprechende Analysesoftware notwendig. Auf diese wird in den folgenden Abschnitten näher eingegangen.

3.1.7. Analysesoftware

Um möglichst viele Eigenschaften von mobilen Applikationen bewerten zu können, empfiehlt es sich, sowohl statische als auch dynamische Analysetechniken zu nutzen, damit sowohl Eigenschaften, die bereits aus dem Quellcode oder den Konfigurationsdateien ersichtlich sind, als auch Verhaltensweisen, die erst bei Ausführung der Applikation erkennbar sind, in die Analyse mit einbezogen werden [1, S.41].

Da im Rahmen der Lehrveranstaltung Mobile Security Testapplikationen, die durch die Betreuung über die Moodle-Umgebung zur Verfügung gestellt werden, analysiert werden, wird auf die Vorstellung von Werkzeugen, die den Download von Applikationen aus dem Google-Play-Store oder deren Extraktion von Mobiltelefonen ermöglichen, verzichtet.

Statische Analyse

Die Analyse des Android-Manifestes zur Überprüfung von Berechtigungen, die zur Installationszeit vom Nutzer angefragt werden, kann durch das Android Asset Packaging Tool (aapt) geschehen. Das Tool bietet keine grafische Benutzeroberfläche an. Die Eingabe von Befehlen erfolgt über die Konsole [1, S.46].
Für eine tiefergehende Untersuchung von Applikationen muss deren .apk-Datei mit Werkzeugen wie Unzip entpackt werden und der ausführbare Bytecode, der in der .dex-Datei enthalten ist, in eine für den Analysten interpretierbare Form übersetzt werden. Diese Übersetzung können Hilfsmittel wie Dex2jar [54], Enjarify [55] oder Baksmali [56] vornehmen. Wie aapt werden diese Tools auch über die Kommandozeile bedient.
Der in eine lesbare Form übersetzte Applikationscode kann mit Hilfe von JD-GUI inspiziert werden. Diese Applikation dient der Betrachtung des auf-

bereiteten Codes. Sie besitzt ein Graphical User Interface (GUI) und ist für Nutzende intuitiver bedienbar als Werkzeuge, die über das Command-Line-Interface gesteuert werden [57]. Analysten, die sich Arbeitsschritte sparen wollen, können Jadx-GUI oder Classyshark nutzen. Diese Werkzeuge sind ohne externe Bytecodeübersetzer in der Lage, eine Android-Applikation in Java-Code zu übersetzen und dem Analysten die extrahierten Informationen grafisch aufbereitet zur Verfügung zu stellen [58] [59]. Beide Werkzeuge können über GitHub bezogen werden. Für den Download von Classyshark muss eine Registrierung auf GitHub vorgenommen werden.
Ein umfangreicheres Tool, das die Vorteile der unterschiedlichen Übersetzungsmechanismen mehrerer Bytecodecompiler vereint, ist der Bytecodeviewer. Die Aufbereitung der über dieses Tool zur Verfügung gestellten Informationen sowie die Bedienungsoberfläche werden ebenfalls grafisch dargestellt [60].

Pentesting

Drozer ist ein Analysetool, mit dessen Unterstützung Schwachstellen in Android-Applikationen gefunden und auch ausgenutzt werden können. Das Tool gibt es als Open-Source-Variante. Der Nutzungsumfang von Drozer ist über den Download und die Installation von zusätzlichen Modulen flexibel erweiterbar. Außerdem kann Drozer verwendet werden, um auf zu testenden mobilen Endgeräten nach installierten Applikationen zu suchen, die bestimmte Berechtigungen bei der Installation fordern. Im Rahmen von dynamischen Analysen können mit Hilfe dieses Tools Informationen über Berechtigungen, exported Activities, Content-Provider und Broadcast-Receiver gesammelt werden [49].
Inspeckage bietet eine Alternative zu Drozer. Über API-Hooking[5] können die Nutzung von Shared Preferences, SQLite-Zugriffe, der Austausch von Daten über HTTP, die Nutzung der WebView-Komponente[6] sowie die Inter-Prozess-Kommunikation in Ausführung beobachtet werden [62].
Im Gegensatz zu Inspeckage wird Drozer derzeit noch aktualisiert und betreut [62] [63].

[5] Als API-Hooking wird die Manipulation appinterner API-Aufrufe zum Zwecke der Ausführung von eingeschleustem Code bezeichnet

[6] Über WebView kann der applikationsinterne Aufruf eines Browsers über einen Intent realisiert werden [61]

Auswertung von Logfiles

Die Android Debug Bridge (adb) ist als Client-Server-Architektur aufgebaut. Durch den adb-Client, der auf dem Computer des Entwicklers installiert ist, werden Befehle über den adb-Server, der als Hintergrundprozess läuft, an das Android-Gerät oder den Emulator gesendet. adb wirkt somit als Proxy für die Analyse zum Zwecke des Debuggings. Die Logdatei, die während des Prozesses von adb erstellt wird, kann vom Analysten auf die Dokumentation kritischer Verhaltensmuster hin untersucht werden [64].
Das Tool Logcat ermöglicht die Auswertung von Logfiles, die über adb zur Laufzeit von Applikationen erstellt wurden. Entsprechende Filterfunktionen werden zur Erleichterung der Auswertung angeboten [65].
Alternativen zu Logcat wie Recat oder Logcat-color, die ein Highlighting von Logfilepositionen unterstützen, werden aktuell nicht betreut. Logcat ist in die Android-Commandline-Tools integriert, die unter Punkt 3.1.5 näher beschrieben wurden.

Analyse und Manipulation des Datenverkehrs

Den Netzwerkverkehr von Applikationen kann man über Address Resolution Protocol (ARP) Spoofing oder die Nutzung eines Proxy-Servers, über den der Verkehr an externe Instanzen weitergeleitet wird, analysieren [1, S. 200]. In Bezug auf die Aufgabenstellung wird auf die Aufzählung rein kommerzieller Softwarevarianten verzichtet.
Die Analyse von Datenverkehr, der mittels Transport Layer Security (TLS) geschützt wird, ist nur dann möglich, wenn das sendende Gerät dem Proxy-Server vetraut, d. h. ein entsprechendes Zertifikat des Servers kennt. Daher ist ein entscheidendes Kriterium, dass die Proxy-Software entsprechende Zertifikate zur Installation anbietet.
Das Tool Burp Suite kann als lokaler Proxy-Server auf dem Analyserechner installiert werden. Das Werkzeug gibt es als kostenpflichtige und Community-Variante.
Alternativen zu Burp-Suite sind Mitmproxy oder OWASP Zed Attack Proxy (ZAP). Alle drei Werkzeuge stellen eine grafische Benutzeroberfläche zur Bedienung sowie Zertifikate zur Untersuchung und Manipulation von verschlüsseltem Datenverkehr zur Verfügung [1, S. 202] [66] [67].

Auswahl

Zur Installation in der Laborumgebung wird Jadx als Werkzeug für die statische Codeanalyse ausgewählt. Jadx besitzt einen ausreichenden Funktionsumfang für die durchzuführenden Analysen und ist durch die Unterstützung in Form eines grafischen User-Interfaces anwenderfreundlich in der Bedienung. Darüber hinaus ist das Werkzeug auch ohne Registrierung über GitHub beziehbar.

Der Nutzungsumfang von Drozer ist für die automatisierte Suche nach Schwachstellen im Rahmen der praktischen Übungen angemessen. Inspeckage wird derzeit nicht aktualisiert. Aus diesem Grund wird als Pentesting-Tool Drozer in der Laborumgebung implementiert.

Die Anwendung der Funktionen von adb und Logcat, sind für die Analysen im Rahmen der Übungen hinlänglich. Über die Installation der Commandline-Tools können beide Werkzeuge ins System integriert werden. Da die Alternativen keine Aktualisierungen mehr unterstützen, wird auf die Implementierung von dieser verzichtet.

Alle drei Proxyvarianten bieten den notwendigen Nutzungsumfang für die Durchführung der Übungen. Die Nutzung von Burp Suite ist bereits in der Literatur zur Lehrveranstaltung beschrieben. Um die mit Hilfe der Kursliteratur erarbeiteten theoretischen Erkenntnisse über die Nutzung dieses Tools weiter zu vertiefen, wird daher für die Anwendung in der Laborumgebung Burp Suite ausgewählt. Die Nutzungsbeschreibung in der Anwenderdokumentation dient einer Ergänzung und zusätzlichen Hilfestellung.

Nachdem die benötigte Ausstattung für die Labor-VM zusammengestellt wurde, wird betrachtet, wie die zu analysierende Testapplikation beschaffen sein muss.

3.2. Testapplikation

Um den Umfang der zu implementierenden Funktionen und Eigenschaften der Test-App zu ermitteln, muss zunächst untersucht werden, welches

Lehrziel durch die praktischen Übungen erreicht werden soll. Auf Basis der Definition dieses Lehrziels kann eine Aufstellung der zu implementierenden Eigenschaften und Funktionen erfolgen, die die Testapplikation aufweisen muss, um den Studierenden im Rahmen der praktischen Übungen entsprechende Erkenntnisse zu erlauben. Diese Betrachtungen werden in den folgenden zwei Sektionen dargestellt.

3.2.1. Zugrundeliegendes Lehrziel

Die Testapplikation soll von den Studierenden auf Schwachstellen, wie sie bereits in Abschnitt 2.2.5 genannt wurden, überprüft werden. An ihr sollen sowohl statische als auch dynamische Analysemethoden angewendet werden. Über die Analyse des Manifests sollen die Studierenden erkennen, welche Berechtigungen durch die Applikation angefragt werden, und diese Anfragen kritisch im Hinblick auf ihre Notwendigkeit und mögliche Sicherheitsrisiken bewerten. Durch statische Analyse des ausführbaren Codes sollen fehlerhafte Implementierungen, die eine Schwachstelle darstellen können, erkannt werden. Darüber hinaus sollen die Studierenden mit Hilfe der Methode des Penetrationtesting mögliche Ansatzpunkte für SQL-Injection[7] oder den Zugriff auf Content-Provider entdecken. Außerdem sollen die Studierenden über die Auswertung der durch die adb erstellten Logfiles sicherheitskritische Verhaltensweisen des Programms erkennen. Darüber hinaus soll die Analyse und Manipulation des Datenverkehrs mit einer externen Instanz ermöglicht werden.

Welche Bedingungen beachtet werden müssen, um eine Applikation zu erstellen, deren Analyse dem eben beschriebenen Lehrziel dient, wird in den nächsten Abschnitten beleuchtet.

3.2.2. Eigenschaften der Testapplikation

Schutz des Codes

Um statische Codeanalysen kritischer Programmbestandteile zu ermöglichen, muss der Quellcode unverschlüsselt und unverschleiert vorliegen.

[7] SQL Injektion (SQLi) ist eine Schwachstelle, die Angreifern die Manipulation von Zugriffen auf Datenbanken erlaubt [68]

Auf Maßnahmen zum Schutz des Codes gegen unerlaubtes Kopieren oder eingehende Analyse muss verzichtet werden.

Berechtigungsmanagement über das Androidmanifest

Damit die Analyse des Manifests durch die Studierenden es ermöglicht, Schwachstellen in Form einer zu umfangreichen Gewährung von Berechtigungen zu entdecken, müssen sowohl Berechtigungen implementiert sein, die dem Nutzungszweck der App dienen, als auch solche, die über den eigentlichen Nutzungszweck nicht notwendigerweise hinausgehen.

Aufbau der Applikation

Um den Studierenden im Rahmen einer statischen Analyse anhand der Struktur der Applikation bereits erste Hinweise zu liefern, welche Aktionen die Applikation zur Laufzeit durchführen wird, sollte die App mehrere Klassen enthalten, die den Funktionscode entsprechend strukturieren.

Nicht offensichtliche Funktionen

Neben den Funktionen, die Nutzern bei der Ausführung der Applikation offensichtlich sind, sollten Funktionen implementiert werden, die nicht dem eigentlichen Nutzungszweck dienen und einem potenziellen Angreifer erlauben, Informationen zu erhalten, deren Preisgabe an Unbekannte der Nutzer nicht zustimmen würde.

Schwächen in der Implementierung

Häufige Fehler bei der Implementierung von Funktionen in mobilen Applikationen wurden bereits unter 2.2.5 beschrieben. Es sollten Mechanismen wie die Verwaltung von Schlüsseln zum Schutz der Kommunikation mit externen Instanzen fehlerhaft implementiert werden [69].
Da eine unzureichende Überprüfung von Dateneingaben in Verbindung mit SQL-Datenbanken SQL-Injection ermöglicht, sollten auch derartige Mechanismen unzureichend implementiert sein [70].
Um den Zugriff auf sensible Daten über Loggingfunktionen zu erlauben, sollten entsprechende Logeinträge durch die Applikation generiert werden. Zusätzlich sollte ein Zugriff auf kritische Funktionen über das Setzen der

Debuggable-Flag erlaubt werden [71]. Darüber hinaus sollten Backups erlaubt werden, um auch diesen Angriffsvektor zu erlauben [72].

Applikation in zwei Versionen

Da die Studierenden über die Kombination von statischen und dynamischen Analysemethoden die Funktionsweise der Applikation ergründen sollen, sollte die Applikation in zwei Versionen erstellt werden. Dadurch ist sichergestellt, dass bei der Durchführung der statischen Codeanalyse nur ein Teil der implementierten Funktionsmechanismen erkannt werden kann. Die zweite Version der Applikation, die ein Upgrade der Grundversion darstellt, sollte zusätzliche Funktionen implementieren, die über eine dynamische Analyse von den Studierenden erkannt und ausgenutzt werden können.

3.3. Skalierung des Systems

Bei der Skalierung des Laborsystems müssen zwei Komponenten betrachtet werden. Die Voraussetzungen, die für die Installation von VirtualBox notwendig sind, und die Ressourcen, die für den Betrieb der virtuellen Laborumgebung und der Emulator-VM, mit den implementierten Softwarebestandteilen vorgehalten werden müssen. Wie sich diese zusammensetzen, beleuchten die nächsten Abschnitte.

3.3.1. Systemauslegung der Labor-VM

In den durch Canonical beschriebenen Systemanforderungen für Ubuntu 22.04 Jammy Jellyfish, ist eine Prozessorkonfiguration mit einer Taktung von 2 GHz unter Nutzung von zwei Kernen empfohlen [36]. Für die Nutzung von Burp Suite wird ein System mit mindestens zwei Prozessorkernen und 4 Gigabyte (GB) Random Access Memory (RAM) für die Durchführung der Analyse von Netzwerkverkehr empfohlen [73].
Weitere notwendige Anpassungen an den durch VirtualBox definierten Default Werten, konnten nicht nachgewiesen werden.
Um ausreichend Speicherplatz für die Installation des Betriebssystems vorzuhalten, sollten dem virtuellen System mindestens 25 GB Speicherplatz zugewiesen werden.
Darüber hinaus muss noch die entsprechende Analysesoftware Platz finden.

Der Platzbedarf der einzelnen Komponenten ist der folgenden Aufzählung zu entnehmen.

- Die Installation von jadx benötigt 501 Megabyte (MB) Speicherplatz.
- Für die Standardinstallation von Burp Suite wird die Reservierung von 1 GB Speicher empfohlen, bei der Analyse größerer Projekte 2 GB.
- Speicherbereiche für die Drozerkomponenten Konsole und Server besitzen eine Größe von 431 MB.
- Die Installation der Android Commandline-Tools benötigt 310 MB.
- Der Platzbedarf von OpenJDK beträgt 377 MB.

Damit beträgt der Platzbedarf aller für den Betrieb der Labor-VM notwendigen Komponenten gerundet 29 GB.
Bei Annahme eines benötigten Puffers von 20 Prozent der Speicherkapazität werden aufgerundet 37 GB für die Installation der Komponenten der Labor-VM benötigt.

3.3.2. Systemauslegung der Android-VM

Die Android-VM sollte entsprechend der Dokumentation der Entwickler mit mindestens zwei virtuellen Prozessoren und 2048 MB virtuellem Arbeitsspeicher ausgestattet werden. Im Rahmen der Tests wurden bei dieser Konfiguration Performanceprobleme festgestellt, daher wird eine Konfiguration mit 4096 MB virtuellem RAM empfohlen.
Aus der Dokumentation der Android-x86-VM geht hervor, dass 8 GB Speichervolumen für die virtuelle Festplatte ausreichend sind. Dies konnte im Rahmen der Tests bestätigt werden.
Als Grafikcontroller muss in den Einstellungen VBoxVGA für den Betrieb des Systems hinterlegt werden [74].

3.3.3. Systemvoraussetzungen des Hostsystems

Grundsätzlich kann die entwickelte Systemkonstellation mit Unterstützung der Virtualisierung durch VirtualBox auf folgenden Hostsystemen genutzt werden [75]:

- Windows-Versionen ab 8.1
- Linux-Versionen der Kernelversion 2.6 aufwärts
- MacOS-Versionen ab 10.15 (Catalina)

Die Installation von VirtualBox selbst benötigt etwa 30 MB Speicherplatz [76]. Darüber hinaus müssen zusätzlich ausreichend Ressourcen für den Betrieb der virtuellen Umgebungen, wie in den vorhergehenden Abschnitten beschrieben, vorgehalten werden. Um das System, wie für die Durchführung der Übungen benötigt, auf den Hostsystemen der Nutzer installieren und performant betreiben zu können, werden folgende Anforderungen empfohlen:

- 8 GB RAM
- 45 GB freie Festplattenkapazität
- 4 Prozessorkerne

Da SSE-2[8] Unterstützung durch aktuelle Prozessoren der Hersteller Intel und AMD implementiert ist, wurde auf diese in der Aufzählung der Voraussetzungen verzichtet [77].

3.4. Gestaltung der Nutzerdokumentation

Um den Studierenden die Erstellung und die Anwendung der Laborumgebung zu erleichtern, muss eine entsprechend umfassende Nutzerdokumentation erstellt werden. Die Erklärungstiefe muss dabei dem Erfahrungsstand der Anwender angepasst werden.
Die Anwender besitzen unterschiedliche Vorbildung in Bezug auf die Verwendung von virtuellen Systemen, den Umgang mit Linux-Betriebssystemen, die Konfiguration von Netzwerken und die Verwendung von Hilfsmitteln und Analyseprogrammen, wie sie in der Laborumgebung zur Verfügung gestellt werden. Da der Wissensstand des Einzelnen schwer abzuschätzen ist, sollte der unerfahrenste Nutzer angenommen werden, um allen Studierenden eine ausreichende Dokumentationsbasis zu bieten.

[8] SSE steht für Single Instruction Multiple Data Extensions

Die Beschreibung sollte ausführlich genug sein, um für häufig auftretende Probleme bei Installation und Betrieb von virtuellen Maschinen Lösungsstrategien anzubieten. Darüber hinaus sollte sie den Nutzenden, die keine entsprechende Vorerfahrung mit Linux-Betriebssystemen haben, die wichtigsten Befehle für die Nutzung und Instandhaltung der virtuellen Umgebung nahebringen.
Eine entsprechende Bebilderung der Dokumentation kann den Studierenden zusätzliche Unterstützung dort bieten, wo Erklärungen in Textform nicht ausreichen, um Sachverhalte eindeutig zu beschreiben.
Für diejenigen, die keine tiefergehende Unterstützung durch die Dokumentation benötigen, sollte die Beschreibung entsprechend übersichtlich gegliedert sein, so dass diese Anwender problemspezifisch auf Erklärungen zugreifen können.

3.5. Zusammenfassung

In diesem Kapitel wurde eine eingehende Analyse der Systembestandteile vorgenommen. Auf Basis dieser wurde eine Hypervisor- sowie eine Betriebssystemvariante für die Erstellung der Laborumgebung ausgewählt. Im Anschluss fand eine Betrachtung und Auswahl von Analysesoftwarekandidaten zur Implementierung in der virtuellen Analyseumgebung statt. Darauf folgend wurden Herausforderungen bezüglich der Lauffähigkeit des Systems auf unterschiedlichen Plattformen identifiziert und entsprechende Lösungsmöglichkeiten diskutiert. Dabei konnte für MacOS-ARM-Architekturen keine Lösung erarbeitet werden, da dies den Umfang der Arbeit gesprengt hätte.
Außerdem wurden die Anforderungen an die Testapplikation zusammengetragen, um einen Überblick zu schaffen, welche Eigenschaften und Funktionalitäten bei der Erstellung der Anwendung zu berücksichtigen sind.
Darüber hinaus wurde beleuchtet, wie die Nutzerdokumentation erstellt sein muss, um die Studierenden bei der Erstellung und Nutzung des Analysesystems entsprechend zu unterstützen. Abschließend wurde betrachtet, welche Voraussetzungen das Host- sowie das Gastsystem erfüllen müssen, damit die Laborumgebung ausreichend performant lauffähig ist.

Aufbauend auf dem eben Beschriebenen werden im nächsten Kapitel die

Implementierung der virtuellen Laborumgebung und der Analysesoftware sowie die Entwicklung der Testapp dokumentiert. Außerdem werden die Erstellung der virtuellen Images, die Zusammenstellung der Nutzerdokumentation und die Nutzungsmöglichkeiten des Systems näher betrachtet.

4. Implementierung

4.1. Installation der Virtualisierungssoftware

4.1.1. Einrichtung der VMM-Variante

Wie bereits in Abschnitt 3.1.2 erwähnt, wurde als Hypervisor für das Laborsystem VirtualBox 7.0.18 implementiert.
Auf die Installation des VirtualBox-Extension-Packs wurde verzichtet, da dieses keine Nutzungserweiterungen enthält, die für die Leistungsfähigkeit in Bezug auf die Nutzung als Plattform für den Betrieb des virtuellen Mobile Security Labors einen Vorteil bieten [78].
Für die dynamische Analyse der Testapplikation benötigen die virtuellen Umgebungen Zugriff auf das Internet, um Daten mit einem externen Server austauschen zu können. Daher wurde ein Netzwerkaufbau über VirtualBox mit Network Address Translation (NAT)-Network realisiert. Bei der in der Nutzerdokumentation beschriebenen Konfiguration, dient der Hostrechner als Gateway für die Labor-VM und diese wiederum als Proxyserver für die Android-VM.

Neben dem Hypervisor benötigt man, wie im Analyseteil unter 3.1.3 erläutert, das virtuelle Abbild eines Betriebssystems. Die Konfiguration der Systemabbilder wird in der folgenden Sektion beschrieben.

4.1.2. Einrichtung der VM-Abbilder

Als Analysesystem wurde eine Ubuntu-VM unter Nutzung der Betriebssystemversion 22.04 Jammy Jellyfish erstellt. Als Emulator wurde eine Android-Umgebung eingerichtet, auf Basis des aktuellsten zur Verfügung stehenden Android-x86-Abbildes, das Android 9 (Pie - API Level 28) implementiert.
Um eine Kompromittierung des Gastsystems zu erschweren, werden für die

Software virtueller Umgebungen regelmäßige Updates empfohlen. VirtualBox weist darauf hin, dass es unabhängig vom installierten Betriebssystemabbild elementar ist, dass aus Sicherheitsgründen sowohl VirtualBox als auch die Gasterweiterungen aktuell gehalten werden sollten [40]. Die Gasterweiterungen wurden in der Ubuntu-Umgebung implementiert und die Empfehlung der regelmäßigen Aktualisierung des Systems wurde in die Nutzerdokumentation aufgenommen. Für die Android-VM existiert die Möglichkeit zur Installation der Gasterweiterungen nicht.
Insbesondere im Hinblick auf die Integrität und Vertraulichkeit der Daten von Basissystemen bei Typ-2-Umgebungen empfiehlt es sich, die Schnittstelle zu den Hostsystemen so schmal wie möglich zu halten, also auf Funktionalitäten, die den Austausch von Daten zwischen Host- und Gastsystem auf der Anwenderebene erleichtern, zu verzichten. Ein entsprechender Hinweis erfolgt im Rahmen der Nutzerdokumentation zur Ubuntu-Umgebung.
Wenn das Betriebssystem den VM Secure-Boot unterstützt, kann auch über die Nutzung dieser Konfiguration die Kompromittierung der virtuellen Umgebung auf Systemebene erschwert werden [79]. Da mit der Implementierung von Secure Boot neben Vorteilen für die Sicherheit des virtuellen und Host-Systems, wie bereits in 3.1.4 beschrieben, Probleme bezüglich der Übertragbarkeit der Abbilder auf andere Systeme einhergehen, wurde auf die Einrichtung dieser Funktion verzichtet. Ein entsprechender Warnhinweis wurde in der Anleitung zum Aufsetzen des virtuellen Systems gegeben. Dieser weist Anwender darauf hin, dass die Nutzung des Systems zur Analyse von Applikationen, deren Nutzungsumfang nicht vollständig bekannt ist, ein Sicherheitsrisiko darstellt.

4.2. Implementierung der Analysesoftware

Die Systemabbilder, deren Konfiguration in Abschnitt 4.4 beschrieben wird, enthalten alle für die Durchführung im Rahmen der Übungen notwendigen Komponenten der Analysesoftware sowie die Applikation Sorglos1 in zwei Versionen.

4.2.1. Einrichtung der Entwicklungsumgebungen

Wie bereits unter 2.2.3 beschrieben, enthalten Entwicklungsumgebungen Basismethoden für die Funktion der Analysesoftware. Um einen

entsprechenden Nutzungsumfang zu erlauben, wurden in der Ubuntu-Laborumgebung OpenJDK 17 und die zum Zeitpunkt der Erstellung aktuellste Version der Android-Development-Commandline-Tools implementiert. Die Plattformtools wurden in der Version Android 34 installiert. Diese beinhalten unter anderem die Android-Debug-Bridge, die sowohl zur Installation von Software auf dem virtuellen Android-Gerät benötigt wird, als auch zur dynamischen Analyse mittels Drozer oder Logcat.

4.2.2. Installation der Analysesoftware

Um entsprechende Analysen im Rahmen der Übungen zu unterstützen, wurden die aktuellste Version der Burp Community Edition-Software, zum Zeitpunkt der Erstellung der Arbeit 2024.5.5, sowie Jadx und Drozer installiert. Um Drozer nutzen zu können, wurden in der Labor-VM die Drozer Client-Software und, unterstützt durch adb, eine Applikationskomponente, genannt Agent, auf der Android-Umgebung installiert [49].
Eine Anpassung der Proxyeinstellungen der Android-VM ist notwendig, um Analysen des Datenverkehrs, unterstützt durch Burp Suite, durchführen zu können. Zur Ermittlung der Netzwerkadresse des Ubuntu-Systems wurden die Linux-Net-Tools in der Labor-VM installiert.

4.3. Entwicklung der Testapplikation

Die Testapplikation Sorglos1, wurde auf Basis der Android API 31 erstellt, kompatibel für Betriebssystemvarianten bis Android API 24, um die Lauffähigkeit auf der Android-VM sicherzustellen.
Das Layout wurde entsprechend der Android-Umgebung, auf der die Applikation ausgeführt werden soll, angepasst, damit Komponenten wie Eingabefelder und Buttons korrekt dargestellt werden.
Die Applikation ermöglicht den Nutzenden das Versenden von Einkaufslisten über den Short-Message-Service. Die Zielrufnummern werden mit Aktivierung der Funktion *Senden* in einer SQLite-Datenbank abgelegt. Darüber hinaus werden durch die Applikationsversion Sorglos1.1 bei jedem Sendevorgang die aktuelle Position und die Nummer des Empfängers an einen externen Server gesendet.
Nachdem die Hardwaresimulation eines GPS-Empfängers in der Android-Umgebung nicht möglich ist, wurde die Funktion zum Positionsabruf

entsprechend angepasst und die Übermittlung der Position über eine festgeschriebene Standortvariable simuliert.

4.3.1. Komponenten

Die Basisversion der Applikation Sorglos1.0 enthält neben der Hauptklasse *MainActivity* zwei weitere Klassen, *NummernDbSchema* und *NummendDatenbankHelfer*. *MainActivity* ist im Paket *com.example.sorglos1* implementiert. Die Klasse *MainActivity* enthält neben den für die Darstellung und Bedienung der Applikation verantwortlichen Funktionen auch Funktionen zur Abfrage der Vergabe von Berechtigungen zur Nutzung des Short-Message-Service und der Standortdaten. Zudem besitzt diese Klasse eine Funktion, die den Versand von Kurznachrichten ermöglicht. Darüber hinaus sind Methoden zum Eintrag der Nummern in die Datenbank und zum Abruf der Standortdaten in *MainActivity* hinterlegt.
Die beiden weiteren Klassen, die für die Organisation von Datenbankeinträgen zuständig sind, sind in einem untergeordneten Paket, *com.example.sorglos1.datenbank* zusammengefasst. Die Klasse *NummernDBSchema* legt das Schema der Nummern-Datenbank fest. Diese enthält eine Tabelle zur Speicherung der Nummerndaten, die in zwei Spalten gegliedert ist. Die Spalte *ID* nimmt Werte eines Counters auf, die der Anzahl der Tabelleneinträge entsprechen. Die Spalte *Nummer* speichert die Nummerndaten aus dem Eingabefeld. Die Klasse *NummendDatenbankHelfer* organisiert die Zugriffe auf die SQLite Datenbank zum Zwecke der Datenspeicherung.
Der Code dieser Applikationsversion ist im Anhang unter A.3.1 einzusehen.

Das Upgrade Sorglos1.1 enthält eine Erweiterung um zwei Klassen, um den Versand von Nachrichten über das Internet zu einem externen Server und die Bereitstellung von Daten aus der Nummern-Datenbank an applikationsexterne Services zu ermöglichen. Die Hauptklasse *MainActivity* ist für den Aufruf der Funktionen der zusätzlichen Klassen erweitert.
Die Klasse *HTTPSend* ist gemeinsam mit *MainActivity* im Paket *com.example.sorglos1* enthalten.
Das Paket *com.example.sorglos1.datenbank* enthält zusätzlich die Klasse *Provider*. In der Klasse *Provider* wurden alle notwendigen Methoden für den Zugriff auf die Datenbank über einen Content-Provider implementiert.
Die Klasse *HTTPSend* enthält neben der Methode zur Kommunikation

über das Hypertext Transfer Protocol (HTTP) auch eine Funktion zur oberflächlichen symmetrischen Verschlüsselung der Positionsdaten. Der Schlüssel ist in Klartext hinterlegt. Der Funktionscode dieser Methode lässt sich mit Hilfe von Jadx nur unzureichend im Rahmen der statischen Codeanalyse wiederherstellen, sodass die Funktionsweise nicht vollständig nachvollzogen werden kann.
Um den Datenverkehr mit einer externen Instanz auswerten und manipulieren zu können, wurde als Zieladresse für den POST-Request in der Funktion *postData* `http://httpbin.org/anything` hinterlegt. Der Webserver erwidert den POST-Request mit einem identischen GET-Request.
Der Applikationscode von Sorglos 1.1 ist unter A.3.2 dargestellt.
Die Erstellung einer dritten Applikationsversion, die die Möglichkeit zur Erprobung der Umgehung von Certificate-Pinning eröffnet, war nicht in der vorgegebenen Zeit möglich.

4.3.2. Android Manifest

Zur Nutzung von Funktionen wie dem Versand von Kurznachrichten, oder dem Abruf der Positionsdaten, müssen entsprechende Berechtigungen erteilt werden. Um die Voraussetzungen zu erfüllen, welche die durch Sorglos 1.0 und Sorglos 1.1 implementierten Methoden für die korrekte Funktion zur Laufzeit benötigen, wurde die Anfrage folgender Berechtigungen implementiert:

1. SEND_SMS
2. ACTIVITY_RECOGNITION
3. ACCESS_COARSE_LOCATION
4. ACCESS_FINE_LOCATION
5. ACCESS_BACKGROUND_LOCATION
6. INTERNET

Der Content-Provider *com.example.sorglos1.datenbank.Provider* ist für den applikationsexternen Zugriff freigegeben. Darüber hinaus werden Backup- und Debuggingfunktionen, sowie der Versand von Klartextnachrichten über Internetverbindungen von der Applikation zugelassen.

4.4. Portierung der Systemabbilder

Zur Portierung der Systembestandteile auf die Rechner der Studierenden werden Abbilder der virtuellen Endgeräte, der Ubuntu-Laborumgebung und der Android-VM in Form von .ova-Dateien erstellt.
Nach Einrichtung der virtuellen Systeme auf den Basissystemen der Nutzer muss, wie in Abschnitt 4.1.1 beschrieben, ein NAT-Netzwerk in VirtualBox eingerichtet werden, damit Kommunikation zwischen den virtualisierten Systemen und mit der externen Serverinstanz stattfinden kann.
Die Installation von VirtualBox, der Systemabbilder und die Einrichtung der Kommunikationskanäle zwischen den VMs sind in der Nutzerdokumentation beschrieben.

Welche Bestandteile die Dokumentation enthält, wird im folgenden Abschnitt dargelegt.

4.5. Zusammenstellung der Nutzerdokumentation

Um den Studierenden die Einrichtung der Laborumgebung sowie deren Nutzung ausführlich zu beschreiben, wurde eine umfangreiche Dokumentation zu folgenden Themen erstellt:

- Installation von VirtualBox: Diese Anleitung enthält Informationen für die Installation von VirtualBox auf den Betriebssystemvarianten Windows, Linux und MacOS.
- Einrichtung der virtuellen Systeme: In diesem Teil der Dokumentation wird beschrieben, wie die Gastbetriebssysteme in VirtualBox installiert und konfiguriert werden. Diese Anleitung dient den Anwendern als Hilfestellung, die nicht die vorkonfigurierten Abbilddateien zur Installation des Systems nutzen möchten.
- Aufsetzen des Laborsystems über .ova-Dateien: Hier wird beschrieben, wie die vorkonfigurierten Systemabbilder in VirtualBox eingebunden werden können. Anschließend wird auf die Anleitung *Vernetzung der VMs über VirtualBox* verwiesen, um den Studierenden Hilfestellung bei der Vernetzung der virtuellen Umgebungen zu bieten. Außerdem wird auf die Dokumentation zur *Einrichtung der virtuellen Systeme*

hingewiesen, die Informationen über den Umgang mit Problemen, zur Änderung von Passwörtern oder auch zur Durchführung von Updates enthält. Abschließend wird auf die Dokumentation *Nutzung der Analysetools* hingewiesen, um die Durchführung der Analysen zu erleichtern.

- Vernetzung der VMs über VirtualBox: Diese Anleitung beschreibt die Einrichtung der NAT-Verbindung zwischen den VMs.
- Installation von OpenJDK und ADT: Dieser Dokumentationsteil beschreibt die Installation von OpenJDK-17 sowie den Android-Development-Commandline-Tools auf dem Ubuntu Labor-System.
- Installation einer App auf dem Android-Emulator: Hier wird beschrieben, wie .apk Dateien mittels adb ausgehend von der Ubuntu-VM auf der Android-VM installiert werden können.
- Installation der Analysetools: In dieser Anleitung wird erklärt, wie die zur Durchführung der Analysen notwendigen Analysewerkzeuge auf den beiden Systemabbildern implementiert werden können.
- Nutzung der Analysetools: Diese Dokumentation enthält Informationen für die Durchführung von statischen, sowie dynamischen Analysen an Sorglos1.0 und Sorglos1.1. Dazu zählen unter anderem die Einrichtung einer Verbindung mittels adb um ein Portforwarding für die Analysen, unterstützt durch Drozer, einzurichten. Auch die Einrichtung der Proxykonfiguration zur Umleitung des Netzwerkverkehrs von der Android-VM über die Ubuntu-Umgebung für die Analyse und Manipulation des Netzwerkverkehrs wird hier beschrieben.

Im Rahmen der Tests wurde durch zwei Studierende bestätigt, dass die Dokumentation verständlich gestaltet ist. Das Aufsetzen der Systeme durch die Test-Nutzer war erfolgreich und Demonstrationen, wie unter A.1 beschrieben, konnten durch diese nachvollzogen werden.

4.6. Einsatz des Labors in der Lehre

In der Lehre kann das Labor zu Demonstrationszwecken genutzt werden. Im Rahmen einer statischen Analyse, unterstützt durch Jadx-GUI, können

die Struktur der Applikation sowie Inhalte des Manifests analysiert werden. Über die Betrachtung des Codes der .dex-Datei können Erkenntnisse über die enthaltenen Pakete und Klassen, sowie Informationen über implementierte Funktionen gewonnen werden. Die Betrachtung des Manifests erlaubt Einblicke in das Berechtigungsmanagement der Applikation. Hier können Berechtigungen identifiziert werden, die dem offensichtlichen Nutzungszweck von Sorglos1.0 dienen und solche, die zusätzlich zum Zwecke des Datenaustauschs im Hintergrund angefragt werden. Die statische Analyse erfolgt durch die Ubuntu-Labor-VM, auf der Jadx installiert ist (vgl. A.3).
Um mittels Drozer weitere Informationen über die erweiterte Version Sorglos1.1 zu erhalten, muss zunächst die Netzwerkverbindung zwischen der Android-VM und der Laborumgebung eingerichtet werden, damit die Drozerkomponenten miteinander kommunizieren können. Bei der anschließenden Analyse können anschließend Informationen über Paketbezeichnungen, implementierte exported Activities und exported Content-Provider entdeckt werden. Darüber hinaus können mit Hilfe von Drozer die Uniform Ressource Identifier (URI)s der Content-Provider entdeckt und angesprochen werden (vgl. A.2.1).
Anhand einer Auswertung des Netzwerkverkehrs zwischen Sorglos1.1 und einer externen Serverinstanz mittels Burp Suite, kann durch die Studierenden der Abfluss von sensiblen Informationen an Dritte beobachtet werden (vgl. A.2.2). Bevor der Netzwerkverkehr zwischen der Testapplikation und dem Server beobachtet und manipuliert werden kann, müssen zunächst, unterstützt durch adb, die Proxyeinstellungen der Android-VM angepasst werden, so dass der Datenverkehr über die Labor-VM geleitet wird. Im Anschluss können die Datenpakete inspiziert und verändert werden.
Im Rahmen der Auswertung der Logfiles, unterstützt durch Logcat, kann über Debugginginformationen partiell nachvollzogen werden, welche Aktionen die Applikation zur Laufzeit ausführt (vgl. A.2.3).

4.7. Zusammenfassung

In diesem Abschnitt wurde, aufbauend auf den in Kapitel 3 (Analyse) zusammengetragenen Informationen, betrachtet, wie das virtuelle Laborsystem eingerichtet und konfiguriert wurde.
Darüber hinaus wurde erörtert, für welche Plattformen die Testapplika-

tion Sorglos1 entwickelt wurde und deren Struktur und Funktionsweise beleuchtet.
Anschließend wurde dargelegt, wie die Übertragbarkeit des Systems auf die Rechner der Studierenden sichergestellt wird.
Darüber hinaus wurde beleuchtet, welche Teile die Nutzerdokumentation enthält und welche Informationen in den einzelnen Dokumenten enthalten sind.
Abschließend wurde dargestellt, wie das Laborsystem in der Lehre eingesetzt werden kann.

Im nächsten Abschnitt wird abschließend diskutiert, ob die in der Einführung unter 1.3 dargelegten Forschungsfragen im Rahmen der Arbeit beantwortet und das Ziel der Arbeit umfassend erreicht werden konnten.

5. Zusammenfassung und Ausblick

Der Fokus dieser Arbeit lag darauf, eine virtuelle Systemumgebung zu erstellen, die auf diversen Betreibssystemen stabil und performant lauffähig ist. Für den Betrieb auf den Betreibssystem-Varianten Linux, Windows und MacOS wurde ein System entwickelt, das im Hinblick auf die durch die Studierenden im Rahmen der Einsendearbeiten durchzuführenden praktischen Übungen stabil auf ihren Systemplattformen zu betreiben ist. Im Rahmen der Tests konnten Probleme nachvollzogen werden, die durch Studierende bereits im Kommunikationsforum der Lehrveranstaltung diskutiert worden waren. Hierzu zählen z. B. Probleme bei der Virtualisierung, wenn Secure-Boot auf dem Hostsystem aktiviert ist, oder Schwierigkeiten bei der VM-internen Emulation. Lösungsmöglichkeiten für diese und weitere Problemstellungen wurden in der im Rahmen dieser Arbeit erstellten Nutzerdokumentation beschrieben. Die Problematik bezüglich der VM-internen Emulation wurde durch die Systemkonfiguration mit zwei vernetzten virtuellen Systemen umgangen. Diese Lösung hat den Nachteil, dass nicht auf die aktuellsten Emulatoren und umfassende Möglichkeiten der Hardwarekonfiguration zurückgegriffen werden kann, da diese nicht durch die Entwickler der Android-X86-Systemabbilder zur Verfügung gestellt werden.
Einschränkend muss erwähnt werden, dass für die Durchführung der Übungen auf ARM-basierten MacOS-Plattformen keine Lösung gefunden wurde.

Die im Rahmen dieser Arbeit entwickelten, vorkonfigurierten Systemabbilder sind so gestaltet, dass sie die erforderliche Software enthalten, um die Studierenden bei ihren Analysen an der Testapplikation *Sorglos1* zu unterstützen. Es wurden Werkzeuge zur Unterstützung von statischen und dynamischen Analysemethoden implementiert, um eine umfassende Betrachtung der Testobjekte zu ermöglichen. Dabei wurde darauf geachtet, die Ressourcen der Hostsysteme zu schonen, indem nur notwendige Softwarebestandteile implementiert wurden.

Im Rahmen der experimentellen Evaluation wurden Funktionsweise und Möglichkeiten der Laborumgebung demonstriert. Übungen, die Studierende bisher im Rahmen des Kurses absolvieren konnten, sind auch mit dem neu gestalteten Setup möglich. Die zu analysierende Applikation wurde in zwei Versionen (Sorglos1.0 und Sorglos1.1) gestaltet. Als Grundlage für die Entwicklung diente eine Analyse des Lehrziels, das unterstützt durch die Laborumgebung und die Applikation erreicht werden soll. Angelehnt an dieses wurden notwendige Eigenschaften und Funktionen der Applikation in Abschnitt 3.2.2 identifiziert und implementiert. Um die Möglichkeit der Umgehung von Certificate-Pinning darzustellen, müsste die Klasse *HTTPSend* und das Manifest der Applikation angepasst werden. Diese Implementierung konnte in der gegebenen Zeit nicht vorgenommen werden.

Den Studierenden wird eine umfangreiche Dokumentation zur Verfügung gestellt, die sowohl die Implementierung als auch die Nutzung des Systems beschreibt. Die Rückmeldungen der Test-Nutzer bestätigen, dass die Dokumente verständlich sind und bei der Erstellung und Nutzung des Systems entsprechend unterstützen.

Anknüpfend an die Ausführungen unter 1.1 bezüglich des Einflusses der Motivation auf den Lernerfolg bestünde eine Möglichkeit zur Weiterentwicklung in der Erprobung eines Gamification-Ansatzes. In einem ersten Schritt wäre ein solcher über den Entwurf einer Storyline zu den Übungen möglich.

A. Anhang

A.1. Statische Analyse der Applikationsversion Sorglos 1.0

Die Android-VM ist so eingerichtet, dass sowohl die Testapplikation Sorglos1 sowie die Drozer-Agent Komponente für die Studierenden über den Startbildschirm gut auffindbar sind (vgl. Abb. A.1).

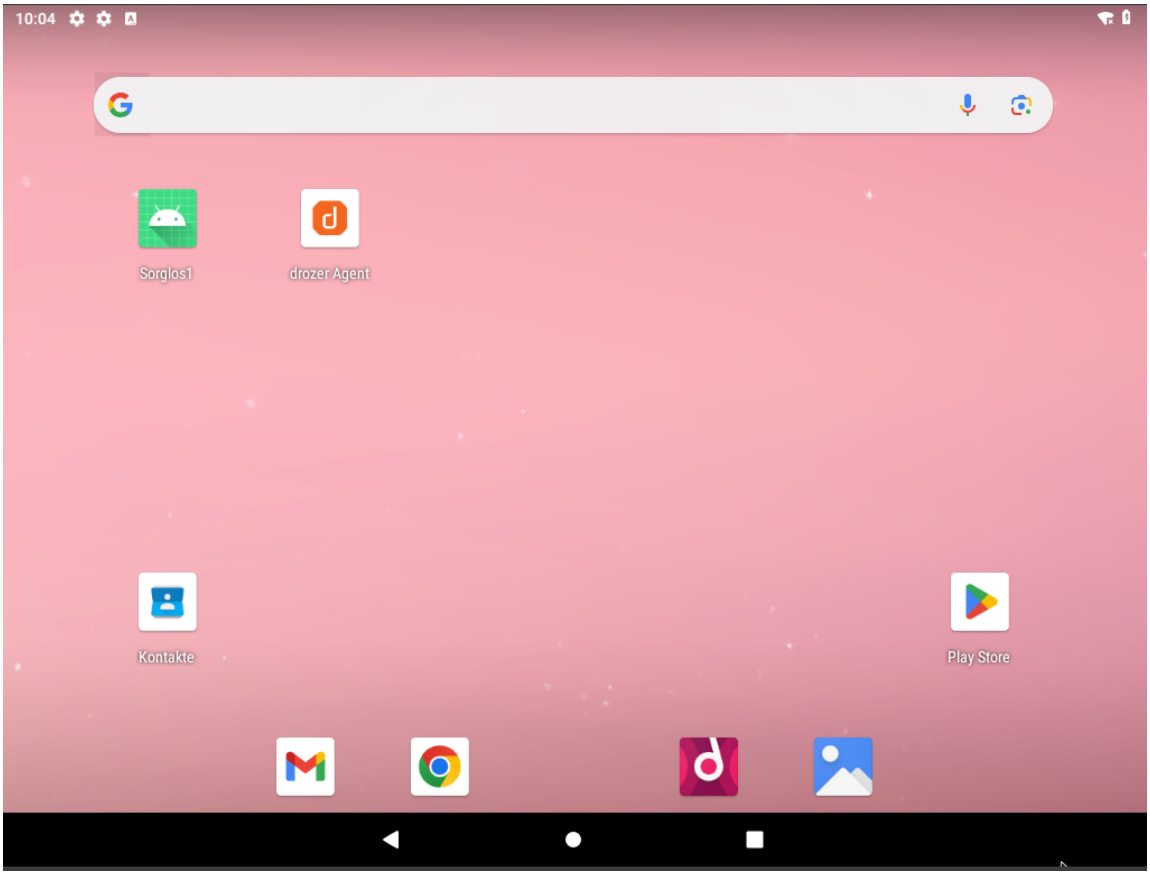

Abbildung A.1.: Oberfläche der Emulator-VM

Das Layout der Testapplikation enthält zwei Texteingabefelder. Die Dateineingabe kann über die Tastatur des Hostsystems erfolgen. Ein Warnhinweis

weist Nutzende auf den Versand von Daten an einen extrenen Server hin (vgl. Abb. A.2).

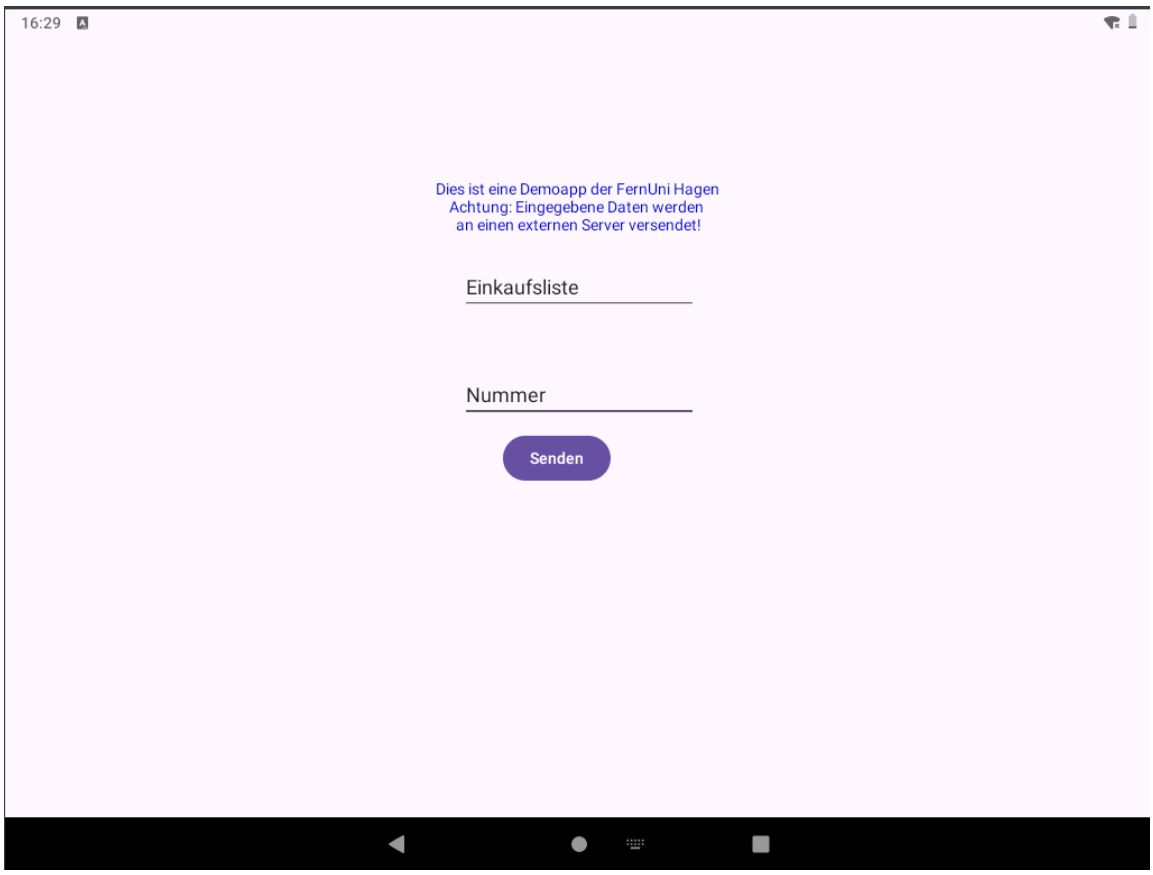

Abbildung A.2.: Layout der Testapplikation

Bei Nutzung des Buttons Senden, werden die eingegebenen Nummern in der implemetierten LiteSQL-Datenbank gespeichert. Darüber hinaus werden die Positions- sowie Nummerndaten an den externen Server gesendet.

Über die statische Analyse der Applikationsversion Sorglos1.0 mittels Jadx können Studierende erste Informationen über die Struktur der Applikation sowie implementierte Basisfunktionen erhalten (vgl. Abb. A.3). Der Applikationscode ist, wie bereits unter 4.3.1 erläutert, in zwei Paketen strukturiert, “com.example.sorglos1” und “datenbank”. Informationen über implementierte Funktionen, enthalten die Klassen des Pakets “datenbank” sowie die Klasse “MainActivity” (vgl. A.3.1).

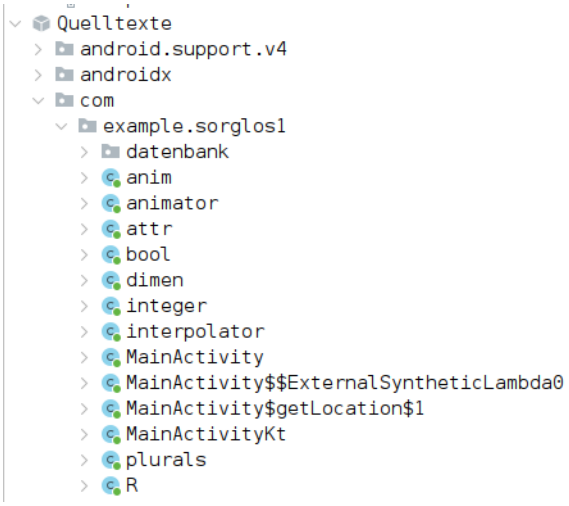

Abbildung A.3.: Analyse der Applikationsstruktur mittels Jadx

Anhand einer Analyse des Manifestes kann nachvollzogen werden, dass die Applikation über Berechtigungen hinaus, die dem offensichtlichen Nutzungszweck dienen, weitere Berechtigungen anfragt (vgl. Abb. A.4).

```
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.ACTIVITY_RECOGNITION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
<permission
```

Abbildung A.4.: Analyse des Berechtigungsmanagements über das Manifest

Nicht dem offensichtlichen Nutzungszweck entsprechen die Berechtigungen, die mit der Abfrage von Standortdaten in Zusammenhang stehen und die Berechtigung android.permission.INTERNET, da die Applikation die Daten der Einkaufsliste mittels SMS an das durch die Eingabedaten des Nummernfeldes spezifizierte Gerät des Empfängers sendet.

Darüber hinaus zeigt die Analyse Manifestes, dass kritsche Funktionen wie Backup- und Debuggingfunktionen sowie der Versand von Nachrichten über das Internet in Klartext durch die Applikation ermöglicht werden (vgl. Abb. A.5).

```
<application
    android:theme="@style/Theme.Sorglos1"
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:debuggable="true"
    android:allowBackup="true"
    android:supportsRtl="true"
    android:extractNativeLibs="false"
    android:fullBackupContent="@xml/backup_rules"
    android:usesCleartextTraffic="true"
```

Abbildung A.5.: Analyse kritischer Funktionen über das Manifest

Die Betrachtung der Signatur von Sorglos1.0 zeigt, dass die Applikation als Debuggingversion durch die Entwicklungsumgebung signiert wurde [80]. Diese Version ist nicht durch den Google-Playstore geprüft worden.

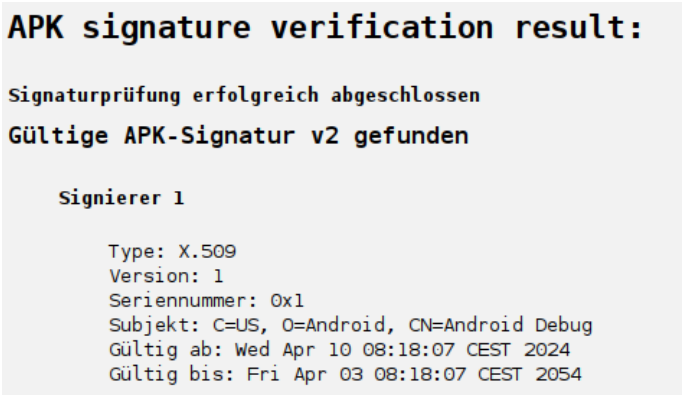

Abbildung A.6.: Analyse der Zertifikatsinformationen

A.2. Analyse an Sorglos 1.1

A.2.1. Entdeckung von Schwachstellen mittels Drozer

Mit Hilfe des Tools Drozer können die Paketnamen der Applikation über den Befehl '`run app.package.list -f <Applikationsname>`' identifiziert werden. In Abbildung A.7 ist dies für die Applikation Sorglos1 dargestellt.

```
dz> run app.package.list -f Sorglos1
Attempting to run shell module
com.example.sorglos1 (Sorglos1)
```

Abbildung A.7.: Identifikation von Paketnamen mittels Drozer

Mögliche Schwachstellen können durch Drozer unter Verwendung des Befehls '`run app.package.attacksurface <Paketname>`' entdeckt werden. Wie in Abbildung A.8 gezeigt, implementiert Sorglos1.1 einen exported Content Provider. Außerdem ist auch mit Hilfe dieses Tools zu erkennen, dass Debuggingfunktionen durch die Applikation zugelassen werden.

```
dz> run app.package.attacksurface com.example.sorglos1
Attempting to run shell module
Attack Surface:
  1 activities exported
  1 broadcast receivers exported
  1 content providers exported
  0 services exported
    is debuggable
```

Abbildung A.8.: Identifikation möglicher Schwachstellen mittels Drozer

Zugreifbare Content-URI sind mit Drozer anhand des Befehls '`run scanner.provider.finduris -a <Paketname>`' auffindbar (vgl. Abb. A.9).

```
dz> run scanner.provider.finduris -a com.example.sorglos1
Attempting to run shell module
Scanning com.example.sorglos1...
No respone from content URI:      content://com.example.sorglos1.androidx-startup
Got a response from content Uri:  content://com.example.sorglos1.datenbank
No respone from content URI:      content://com.example.sorglos1.androidx-startup/
Got a response from content Uri:  content://com.example.sorglos1.datenbank/

For sure accessible content URIs:
  content://com.example.sorglos1.datenbank
  content://com.example.sorglos1.datenbank/
```

Abbildung A.9.: Indentifikation aufrufbarer Conten-URIs

Anhand dieser Analyse an Sorglos1.1 lässt sich feststellen, dass auf die Adresse `content://com.example.sorglos1.datenbank` zugegriffen werden kann.

Mit Hilfe des Befehls '`run app.provider.query <Content-URI>`' kann die Datenbank eingesehen werden (vgl. Abb. A.10). Die Betrachtung der Datenbank liefert Informationen über den Aufbau.

```
dz> run app.provider.query content://com.example.sorglos1.datenbank/
Attempting to run shell module
| id | nummer |
| 1  | 123456 |
| 2  | 123456 |
```

Abbildung A.10.: Analyse des Tabellenaufbaus

Diese Erkenntnisse können bei der Manipulation der Datenbank genutzt werden. Über den Befehl '`run app.provider.insert <Content-URI> --<Datentyp> <Spaltenname> <Wert> --<Datentyp> <Spaltenname> <Wert>`' können der Datenbank, wie in Abbildung A.11 demonstriert, Einträge hinzugefügt werden.

```
dz> run app.provider.insert content://com.example.sorglos1.datenbank/ --string id 3 --string
 nummer 23456
Attempting to run shell module
Done.

dz> run app.provider.query content://com.example.sorglos1.datenbank/
Attempting to run shell module
| id | nummer |
| 1  | 123456 |
| 2  | 123456 |
| 3  | 23456  |
```

Abbildung A.11.: Manipulation der Nummerntabelle - neuer Eintrag

Unterstützt durch den Befehl '`run app.provider.delete <Content-URI> --selection "<Spaltenname>=?" --selction-args <Wert>`' können Einträge mit dem entsprechenden Spaltenwert gelöscht werden (vgl. Abb. A.12).

```
dz> run app.provider.delete content://com.example.sorglos1.datenbank/ --selection "nummer=?"
 --selection-args 23456
Attempting to run shell module
Done.

dz> run app.provider.query content://com.example.sorglos1.datenbank/
Attempting to run shell module
| id | nummer |
| 1  | 123456 |
| 2  | 123456 |
```

Abbildung A.12.: Manipulation der Nummerntabelle - Entfernung eines Eintrags

Content-Provider, die eine Anfälligkeit für SQL-Injection aufweisen, können über den Befehl '`run scanner.provider.injection -a <Paketname>`' identifiziert werden. Wie in Abbildung A.13 demonstriert, weist der in Sorglos1.1 implementierte Content-Provider diese Sicherheitslücke auf.

```
dz> run scanner.provider.injection -a com.example.sorglos1
Attempting to run shell module
Scanning com.example.sorglos1...
Not Vulnerable:
  content://com.example.sorglos1.androidx-startup/
  content://com.example.sorglos1.androidx-startup

Injection in Projection:
  content://com.example.sorglos1.datenbank/
  content://com.example.sorglos1.datenbank

Injection in Selection:
  content://com.example.sorglos1.datenbank/
  content://com.example.sorglos1.datenbank
```

Abbildung A.13.: Analyse bezüglich Anfälligkeit für SQL-Injection

A.2.2. Betrachtung und Manipulation des Netzwerkverkehrs mittels Burp

Bei der Analyse des Netzwerkverkehrs mit Hilfe des Tools Burp im Intercept Modus ist zu erkennen, dass Positions- und Nummerndaten durch die Applikation mittels POST-Request, über Port 80, an die Adresse `http://httpbin.org` versendet werden (vgl. Abb. A.14).

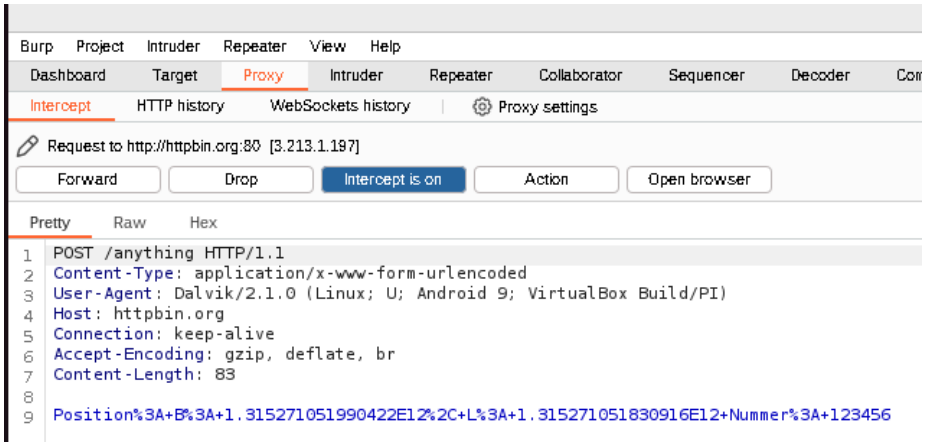

Abbildung A.14.: Analyse POST-Nachrichten mittels Burp

Die Nachrichten können vor der Weiterleitung manipuliert werden, indem die Daten im Textfeld angepasst werden. In Abb. A.15 ist eine Anpassung der Nummerndaten vor der Weiterleitung dargestellt.

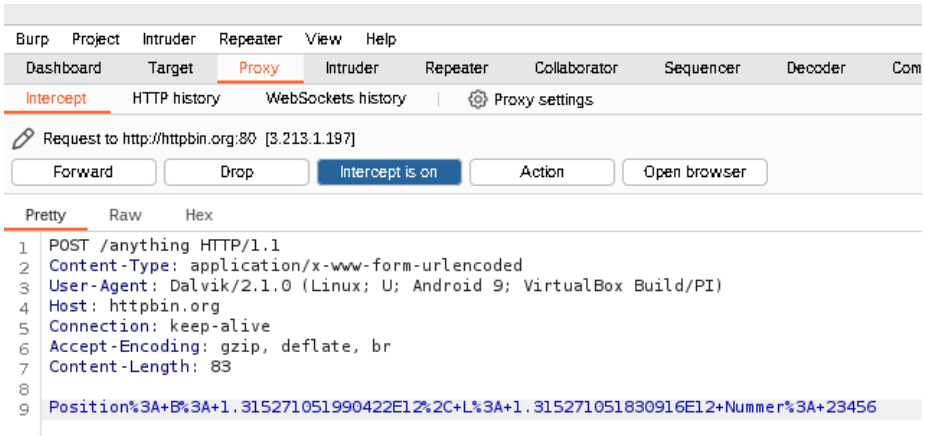

Abbildung A.15.: Manipulation von POST-Nachrichten mittels Burp

Die Manipulation ist mittels Burp anhand der Analyse der Serverantwort nachzuvollziehen (vgl. Abb. A.16).

Abbildung A.16.: Analyse der Serverantwort mittels Burp Repeater

A.2.3. Auswertung der Logeinträge mittels Logcat

Um die Anzeige der Logeinträge auf die mit der Applikation Sorglos 1.1 in Verbindung stehenden Einträge zu reduzieren, kann mit Hilfe des Tools adb über den Befehl '`./adb shell`' eine externe Shell, die mit der Android-VM verbunden ist, eingerichtet werden. Über den Befehl '`logcat | grep Sorglos-Debug`' kann anschließend das Logging gestartet werden. Damit Funktionen, wie z.B. der Austausch von Nachrichten über das Internet, oder das Speichern von Nummern in der Datenbank, über entsprechende Einträge nachvollogen werden können, muss als auslösendes Ereignis die Aktion "Senden" auf der Applikation ausgeführt werden (vgl. Abb. A.17).

```
mobilesec@mobilesec-VirtualBox:~/Android/platform-tools$ ./adb shell
x86_64:/ $ logcat | grep Sorglos-Debug
07-07 11:11:29.866  3129  3129 D Sorglos-Debug: Location: B: 37.42199833, L: -122.08400000
07-07 11:11:58.501  3129  3129 D Sorglos-Debug: Nummer 123456 erfolgreich gespeichert
07-07 11:11:58.531  3129  3129 D Sorglos-Debug: Location: B: 37.42199833, L: -122.08400000
07-07 11:11:58.582  3129  3129 D Sorglos-Debug: Nummer: 123456 Text: Ein belegtes Brot mit Schinken
07-07 11:11:58.743  3129  3164 D Sorglos-Debug: HTTPsend Position: B: 1.315271051990422E12, L: 1.315
```

Abbildung A.17.: Analyse von Debuggingnachrichten mittels Logcat

Die erfolgreiche Übermittlung von Nachrichten des externen Servers an die Applikation kann, wie in Abbildung A.18 dargestellt, anhand von Debuggingnachrichten nachvollzogen werden. Zu sehen ist, dass die Nachricht in Klartext versendet wird. Die Positionsdaten sind durch die implementierte Verschlüsselungsfunktion unzureichend verfremdet.

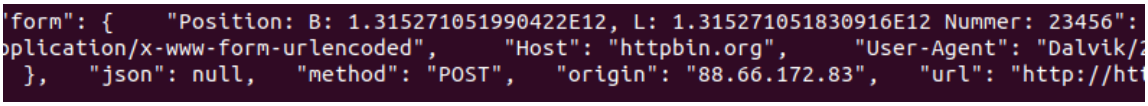

Abbildung A.18.: Analyse der Serverantwort mittels Logcat

A.3. Code der Testapplikation

A.3.1. Sorglos 1.0

Klasse MainActivity

Die Codebestandteile, die zur Überprüfung der Erteilung von Berechtigungen, zum Versenden von Nachrichten über den Short Message Service, den Abruf der Position über den Location Provider und den Eintrag von Nummern in der Datenbank implementiert wurden, sind in Anlehnung an Beispiele, die durch Thomas Theis in "Einstieg in Kotlin" beschrieben wurden, entwickelt worden [81].

```
1  package com.example.sorglos1
2
3  import android.Manifest
4  import android.annotation.SuppressLint
5  import android.content.ContentValues
6  import android.content.Context
7  import android.content.pm.PackageManager
8  import android.database.sqlite.SQLiteDatabase
9  import android.location.Location
10 import android.location.LocationListener
11 import android.location.LocationManager
12 import android.os.Bundle
13 import android.telephony.SmsManager
14 import android.util.Log
15 import android.view.View
16 import android.widget.EditText
17 import androidx.activity.enableEdgeToEdge
18 import androidx.appcompat.app.AppCompatActivity
19 import androidx.core.app.ActivityCompat
20 import androidx.core.view.ViewCompat
21 import androidx.core.view.WindowInsetsCompat
22 import com.example.sorglos1.datenbank.NummernDatenbankHelfer
23 import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.Columns.
       KEY_NUMMERN_WERT
24 import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.TABLE_NAME
25
26 const val KONST_LOCATION=242
27
28 class MainActivity : AppCompatActivity() {
29
30     //Variablendeklaration
31     private val datenbank = NummernDatenbankHelfer(this)
```

```
32      private var lis:LocationListener? = null
33      private var man: LocationManager? = null
34      private var loc = ""
35
36      override fun onCreate(savedInstanceState: Bundle?) {
37          super.onCreate(savedInstanceState)
38          enableEdgeToEdge()
39          setContentView(R.layout.activity_main)
40          ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
                insets ->
41              val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
42              v.setPadding(systemBars.left, systemBars.top, systemBars.right,
                    systemBars.bottom)
43              insets
44          }
45
46          //Berechtigung fuer Zugriff auf Locationservices wird ueberprueft
47          if (ActivityCompat.checkSelfPermission(this,
48                  Manifest.permission.ACCESS_FINE_LOCATION
49              ) != PackageManager.PERMISSION_GRANTED
50          )
51              //Nutzeranfrage wird gestellt, falls Berechtigung nicht bereits erteilt
52              ActivityCompat.requestPermissions(
53                  this,
54                  arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
55                  KONST_LOCATION
56              )
57          else {
58              //Bei bereits erteilter Berechtigung wird der Standort abgerufen
59              getLocation()
60          }
61      }
62
63
64      /*Wenn die Nutzerabfrage ein positives Ergebnis meldet, wird die Position
            abgerufen,
65        anderenfalls eine Debugging-Nachricht ausgegeben*/
66      @SuppressLint("MissingSuperCall")
67      override fun onRequestPermissionsResult(
68          requestCode: Int,
69          permissions: Array<out String>,
70          grantResults: IntArray
71      ) {
72              if (requestCode == KONST_LOCATION && grantResults.isNotEmpty() &&
                    grantResults[0] == PackageManager.PERMISSION_GRANTED)
73                  getLocation()
74              else Log.d("Sorglos-Debug", "Kein␣Recht␣auf␣Nachricht␣Standortdaten␣
                    erteilt")
75      }
76
77      //Versand der SMS mit den Einkaufslistendaten
78      fun sendSMS(view: View) {
79          writeNumber()
80          // Pruefen ob die Berechtigung zum Senden von SMS gesetzt wurde
81          if (ActivityCompat.checkSelfPermission(this, android.Manifest.permission.
                SEND_SMS) == PackageManager.PERMISSION_DENIED) {
```

```
82              Log.d("Sorglos-Debug", "Kein␣Recht␣auf␣Versenden␣von␣SMS␣erteilt")
83              //Berechtigung erfragen ueber Nutzerabfrage
84              ActivityCompat.requestPermissions(this, arrayOf(android.Manifest.
                    permission.SEND_SMS), 2)
85          }
86
87          else{
88              //Bei Berechtigung erteilt, Einkaufsliste wird versendet
89              val message = findViewById<EditText>(R.id.einkaufsliste).getText().
                    toString();
90              val number = findViewById<EditText>(R.id.nummer).getText().toString();
91              SmsManager.getDefault().sendTextMessage(number,null,message,null,null)
92              Log.d("Sorglos-Debug", "Nummer:␣" + number + "␣Text:␣" + message + "␣
                    Nachricht␣erfolgreich␣versendet")
93          }
94      }
95
96
97      fun writeNumber(){
98          //Nummer wird in Datenbank gespeichert
99          val db: SQLiteDatabase = datenbank.writableDatabase
100         val number = findViewById<EditText>(R.id.nummer).getText().toString()
101         val values = ContentValues()
102         values.put(KEY_NUMMERN_WERT, number)
103         db.insert(TABLE_NAME, null, values)
104         Log.d("Sorglos-Debug", "Nummer␣" + number + "␣erfolgreich␣gespeichert")
105         db.close()
106     }
107
108     //Abruf der Positionsdaten (funktioniert auf X86 VM nicht Korrekt)
109     private fun getLocation(){
110         man= getSystemService(Context.LOCATION_SERVICE) as LocationManager
111         lis = object: LocationListener {
112             override fun onLocationChanged(location: Location) {
113                 loc = "B:␣%.8f,␣L:␣%.8f".format(location.latitude, location.longitude
                        )
114                 Log.d("Sorglos-Debug", "Location:␣" + loc)
115             }
116             override fun onStatusChanged(provider: String, status: Int, extras:
                    Bundle){}
117             override fun onProviderEnabled(provider: String) {}
118             override fun onProviderDisabled(provider: String) {}
119         }
120     }
121
122     /*Bei Aktivierung nach Pause (Activity im Hintergrund), werden die
            Locationupdates wieder abgerufen*/
123     override fun onResume() {
124         super.onResume()
125         if (ActivityCompat.checkSelfPermission(this,
126                 Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.
                        PERMISSION_GRANTED) {
127                     //NETWORK_PROVIDER fuer x86-VM anwaehlen, GPS_Provider nicht
                            implementiert, Location Update wird simuliert
128                     man?.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                            2000, 0f, lis!!)
```

```
129                     //man?.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000,
                            1f, lis!!)
130
131                 }
132         }
133
134     //Activity Funktion im Hintergrund: keine Locationupdates generiert
135     override fun onPause() {
136         super.onPause()
137         if (ActivityCompat.checkSelfPermission(this,
138                 Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.
                    PERMISSION_GRANTED){
139             man?.removeUpdates(lis!!)
140         }
141     }
142 }
```

Klasse NummernDbSchema

Das Schema wurde in Anlehnung an ein Beispiel von Kodeco entwickelt [82].

```
1  package com.example.sorglos1.datenbank
2
3  object NummernSchema {
4      /*Beschreibung des Datenbankschemas (Name der Datenbank,
5        Name der Tabelle  und Spaltenbezeichnungen)*/
6      const val DATABASE_VERSION = 1
7      const val DATABASE_NAME = "nummern.db"
8      object NummernTabelle {
9          const val TABLE_NAME = "nummern"
10         object Columns {
11             const val KEY_NUMMERN_ID = "id"
12             const val KEY_NUMMERN_WERT = "nummer"
13         }
14     }
15 }
```

Klasse NummernDatenbankHelfer

Diese Codebestandteile wurden ebenfalls in Anlehnung an Beispielcode von Kodeco entwickelt [82].

```
1  package com.example.sorglos1.datenbank
2
3  import android.content.Context
4  import android.database.sqlite.SQLiteDatabase
5  import android.database.sqlite.SQLiteOpenHelper
6  import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.Columns.
       KEY_NUMMERN_ID
7  import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.Columns.
       KEY_NUMMERN_WERT
```

```
8  import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.TABLE_NAME
9
10 class NummernDatenbankHelfer(context:Context): SQLiteOpenHelper (context,
       NummernSchema.DATABASE_NAME, null,NummernSchema.DATABASE_VERSION) {
11
12     /*Tabelle wird angelegt mit Namen TABLE_NAME und 2 Spalten (NUMMERN_ID und
           NUMMERN_WERT)*/
13     override fun onCreate(db: SQLiteDatabase?) {
14         val createNummernTable = """
15 ␣␣␣␣CREATE␣TABLE␣$TABLE_NAME␣␣(
16 ␣␣␣␣␣␣$KEY_NUMMERN_ID␣INTEGER␣PRIMARY␣KEY,
17 ␣␣␣␣␣␣$KEY_NUMMERN_WERT␣␣TEXT);
18 ␣␣"""
19         db?.execSQL(createNummernTable)
20     }
21
22     //Bei Datenbankupgrade wird Datenbank neu angelegt
23     override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
24         db?.execSQL("DROP␣TABLE␣IF␣EXISTS␣$TABLE_NAME")
25         onCreate(db)
26     }
27
28 }
```

A.3.2. Sorglos 1.1

Angepasste Klasse MainActivity

```
1  package com.example.sorglos1
2
3  import android.Manifest
4  import android.annotation.SuppressLint
5  import android.content.ContentValues
6  import android.content.Context
7  import android.content.pm.PackageManager
8  import android.database.sqlite.SQLiteDatabase
9  import android.location.Location
10 import android.location.LocationListener
11 import android.location.LocationManager
12 import android.os.Bundle
13 import android.telephony.SmsManager
14 import android.util.Log
15 import android.view.View
16 import android.widget.EditText
17 import androidx.activity.enableEdgeToEdge
18 import androidx.appcompat.app.AppCompatActivity
19 import androidx.core.app.ActivityCompat
20 import androidx.core.view.ViewCompat
21 import androidx.core.view.WindowInsetsCompat
22 import com.example.sorglos1.datenbank.NummernDatenbankHelfer
23 import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.Columns.
       KEY_NUMMERN_WERT
24 import com.example.sorglos1.datenbank.NummernSchema.NummernTabelle.TABLE_NAME
```

```
25
26  const val KONST_LOCATION=242
27
28  class MainActivity : AppCompatActivity() {
29
30      //Variablendeklaration
31      private val datenbank = NummernDatenbankHelfer(this)
32      private val httpsend = HTTPSend()
33      private var lis:LocationListener? = null
34      private var man: LocationManager? = null
35      private var loc = ""
36
37      override fun onCreate(savedInstanceState: Bundle?) {
38          super.onCreate(savedInstanceState)
39          enableEdgeToEdge()
40          setContentView(R.layout.activity_main)
41          ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
                insets ->
42              val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
43              v.setPadding(systemBars.left, systemBars.top, systemBars.right,
                    systemBars.bottom)
44              insets
45          }
46          //Berechtigung fuer Zugriff auf Locationservices wird ueberprueft
47          if (ActivityCompat.checkSelfPermission(this,
48                  Manifest.permission.ACCESS_FINE_LOCATION
49              ) != PackageManager.PERMISSION_GRANTED
50          )
51              //Nutzeranfrage wird gestellt, falls Berechtigung nicht bereits erteilt
52              ActivityCompat.requestPermissions(
53                  this,
54                  arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
55                  KONST_LOCATION
56              )
57          else {
58              //Bei bereits erteilter Berechtigung wird der Standort abgerufen
59              getLocation()
60          }
61      }
62
63
64      /*Wenn die Nutzerabfrage ein positives Ergebnis meldet, wird die Position
            abgerufen, anderenfalls eine Debugging-Nachricht ausgegeben*/
65      @SuppressLint("MissingSuperCall")
66      override fun onRequestPermissionsResult(
67          requestCode: Int,
68          permissions: Array<out String>,
69          grantResults: IntArray
70      ) {
71              if (requestCode == KONST_LOCATION && grantResults.isNotEmpty() &&
                    grantResults[0] == PackageManager.PERMISSION_GRANTED)
72                  getLocation()
73              else Log.d("Sorglos-Debug", "Kein␣Recht␣auf␣Nachricht␣Standortdaten␣
                    erteilt")
74      }
75
```

```
76      //Versand der SMS mit den Einkaufslistendaten
77      fun sendSMS(view: View) {
78          writeNumber()
79          /*Bei jeder Nutzung der Sendefunktion wird die Location an den Server
                uebertragen*/
80          getLocation()
81          //Pruefen ob die Berechtigung zum Senden von SMS gesetzt wurde
82          if (ActivityCompat.checkSelfPermission(this, android.Manifest.permission.
                SEND_SMS) == PackageManager.PERMISSION_DENIED) {
83              Log.d("Sorglos-Debug", "Kein␣Recht␣auf␣Versenden␣von␣SMS␣erteilt")
84              //Berechtigung erfragen ueber Nutzerabfrage
85              ActivityCompat.requestPermissions(this, arrayOf(android.Manifest.
                    permission.SEND_SMS), 2)
86          }
87
88          else{
89              //Bei Berechtigung erteilt, Einkaufsliste wird versendet
90              val message = findViewById<EditText>(R.id.einkaufsliste).getText().
                    toString();
91              val number = findViewById<EditText>(R.id.nummer).getText().toString();
92              SmsManager.getDefault().sendTextMessage(number,null,message,null,null)
93              Log.d("Sorglos-Debug", "Nummer:␣" + number + "␣Text:␣" + message + "␣
                    Nachricht␣erfolgreich␣versendet")
94          }
95      }
96
97      fun writeNumber(){
98          //Nummer wird in Datenbank gespeichert
99          val db: SQLiteDatabase = datenbank.writableDatabase
100         val number = findViewById<EditText>(R.id.nummer).getText().toString()
101         val values = ContentValues()
102         values.put(KEY_NUMMERN_WERT, number)
103         db.insert(TABLE_NAME, null, values)
104         Log.d("Sorglos-Debug", "Nummer␣" + number + "␣erfolgreich␣gespeichert")
105         db.close()
106         /*Hardcoded Positionsdaten werden versendet, nach unzureichender
                Verschluesselung*/
107         val pos = httpsend.postionEncryptor(37.42199833, -122.08400000)
108         httpsend.postData("Position:␣" + pos +"␣Nummer:␣"+ number)
109     }
110
111     //Abruf der Positionsdaten (funktioniert auf X86 VM nicht Korrekt)
112     private fun getLocation(){
113         man= getSystemService(Context.LOCATION_SERVICE) as LocationManager
114         lis = object: LocationListener {
115             override fun onLocationChanged(location: Location) {
116                 loc = "B:␣%.8f,␣L:␣%.8f".format(location.latitude, location.longitude
                        )
117                 Log.d("Sorglos-Debug", "Location:␣" + loc)
118             }
119             override fun onStatusChanged(provider: String, status: Int, extras:
                    Bundle){}
120             override fun onProviderEnabled(provider: String) {}
121             override fun onProviderDisabled(provider: String) {}
122         }
123         Log.d("Sorglos-Debug", "Location:␣B:␣37.42199833,␣L:␣-122.08400000")
```

```
124     }
125
126     /*Bei Aktivierung nach Pause (Activity im Hintergrund), werden die
            Locationupdates wieder abgerufen*/
127     override fun onResume() {
128         super.onResume()
129         if (ActivityCompat.checkSelfPermission(this,
130                 Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.
                    PERMISSION_GRANTED) {
131                 //NETWORK_PROVIDER fuer x86-VM anwaehlen, GPS_Provider nicht
                        implementiert, Location Update wird simuliert
132                 man?.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                        2000, 0f, lis!!)
133                 //man?.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000,
                         1f, lis!!)
134             }
135         }
136
137     //Activity Funktion im Hintergrund: keine Locationupdates generiert
138     override fun onPause() {
139         super.onPause()
140         if (ActivityCompat.checkSelfPermission(this,
141                 Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.
                    PERMISSION_GRANTED){
142             man?.removeUpdates(lis!!)
143         }
144     }
145 }
```

Zusätzliche Klasse HTTPSend

```
1  package com.example.sorglos1
2
3  import android.util.Log
4  import kotlinx.coroutines.Runnable
5  import java.io.BufferedReader
6  import java.io.InputStreamReader
7  import java.io.OutputStreamWriter
8  import java.net.HttpURLConnection
9  import java.net.URL
10 import java.net.URLEncoder
11
12
13 class HTTPSend {
14
15     //Variablendeklaration
16     private var serverResponse = ""
17
18     //Defintion der Methode zur Behandlung des Sendevorgangs der POST-Nachricht
19     fun postData(text: String) {
20         /*Behandlung der Funktion in einem Thread um Funktion der Applikation nicht
               durch Wartezeiten zu beeintraechtigen*/
21         Thread(Runnable{
22             val serverURL = "http://httpbin.org/anything"
```

```
23              val url = URL(serverURL)
24              var submit = URLEncoder.encode(text, "UTF-8")
25              try {
26                  with(url.openConnection() as HttpURLConnection) {
27                      /*Versand der durch die MainActivity uebergebenen Daten als POST-
                            Nachricht*/
28                      requestMethod = "POST"
29                      val wr = OutputStreamWriter(outputStream)
30                      wr.write(submit)
31                      wr.flush()
32                      //Nach erfolgreichem Versand ausgabe einer Debugging-Nachricht
33                      Log.d("Sorglos-Debug", "HTTPsend " + text +" successful")
34
35                      //Entgegennahme der Serverantwort
36                      BufferedReader(InputStreamReader(inputStream)).use {
37                          val response = StringBuffer()
38                          var inputLIne = it.readLine()
39                          while (inputLIne != null) {
40                              response.append(inputLIne)
41                              inputLIne = it.readLine()
42                          }
43                          it.close()
44                          //Ueberschreiben von serverResponse mit den empfangenen Daten
45                          serverResponse = "$response"
46                          //Ausgabe der Serverantwort als Debugging-Nachricht
47                          Log.d("Sorglos-Debug", "Response" + response)
48
49                      }
50                  }
51              } finally {
52                  //nicht definiert
53              }
54          }).start()
55      }
56
57      //Funktion zur Verschluesselung der Positionsdaten mittels Hardcoded Secret
58      fun postionEncryptor(posB:Double, posL:Double): String{
59          val key = 1315271051953
60          val bEncrypted = (posB + key).toString()
61          val lEncrypted = (posL + key).toString()
62          val posEncrypted = "B: "+ bEncrypted +", L: "+ lEncrypted
63          return posEncrypted
64      }
65  }
```

Zusätzliche Klasse Provider

Die Codebestandteile wurden in Anlehnung an ein Beispiel der Website Medium entwickelt [83].

```
1  package com.example.sorglos1.datenbank;
2
3  import android.content.ContentProvider;
4  import android.content.ContentUris;
```

```
5  import android.content.ContentValues;
6  import android.database.Cursor;
7  import android.database.sqlite.SQLiteDatabase;
8  import android.net.Uri;
9
10 public class Provider extends ContentProvider {
11
12     /*Variablendeklaration NummernDatenbankHelfer Objekt und Datenbankauthority
13       sowie Pfad zur Tabelle*/
14     private NummernDatenbankHelfer datenbank;
15     private static final String AUTHORITY = "com.example.sorglos1.datenbank";
16     private static final String BASE_PATH = "nummern";
17     //Deklaration des Content-URI
18     Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + BASE_PATH);
19
20     //Initialisierung des NummernDatenbankHelfer Objektes
21     @Override
22     public boolean onCreate() {
23         datenbank = new NummernDatenbankHelfer(getContext());
24         return true;
25     }
26
27     //Definition der Methode fuer Abfragen auf com.example.sorglos1.datenbank
28     @Override
29     public Cursor query(Uri uri, String[] projection, String selection, String[]
           selectionArgs, String sortOrder) {
30         SQLiteDatabase db = datenbank.getReadableDatabase();
31         //Defintion eines Curserobjektes fuer Abragen aus der Nummern-Tabelle
32         Cursor cursor = db.query("nummern", projection, selection,
33                 selectionArgs, null, null, sortOrder);
34         cursor.setNotificationUri(getContext().getContentResolver(), uri);
35         return cursor;
36     }
37
38     /*Definition der Methode fuer Dateneingaben auf com.example.sorglos1.datenbank/
           nummern*/
39     @Override
40     public Uri insert(Uri uri, ContentValues values) {
41         SQLiteDatabase db = datenbank.getWritableDatabase();
42         long id = db.insert("nummern",null,values);
43         Uri itemUri = ContentUris.withAppendedId(CONTENT_URI, id);
44         getContext().getContentResolver().notifyChange(itemUri, null);
45         return itemUri;
46     }
47
48     /*Definition der Methode fuer die Loeschung von Daten auf com.example.sorglos1.
           datenbank/nummern*/
49     @Override
50     public int delete(Uri uri, String selection, String[] selectionArgs) {
51         SQLiteDatabase db = datenbank.getWritableDatabase();
52         int count = db.delete("nummern", selection, selectionArgs);
53         getContext().getContentResolver().notifyChange(uri, null);
54
55         return count;
56     }
57
```

```
58      //Definition der Methode zum Update von com.example.sorglos1.datenbank/nummern
59      @Override
60      public int update(Uri uri, ContentValues values, String selection, String[]
            selectionArgs) {
61          SQLiteDatabase db = datenbank.getWritableDatabase();
62          int count = db.update("nummern", values, selection, selectionArgs);
63          getContext().getContentResolver().notifyChange(uri, null);
64          return count;
65      }
66
67      //Gibt den MIME Typ des Content-URI zurueck
68      @Override
69      public String getType(Uri uri) {
70          return null;
71      }
72  }
```

Literatur

[1] M. Spreitzenbarth, *Mobile Hacking - Ein kompakter Einstieg ins Penetration Testing mobiler Applikationen – iOS, Android und Windows Mobile*. Heidelberg: dpunkt.verlag, 2017, ISBN: 978-3-960-88124-7.

[2] G. E. Hinton, „How neural networks learn from experience," *Scientific American*, Jg. 267, Nr. 3, S. 144–151, 1992, ISSN: 0036-8733. DOI: 10.1038/scientificamerican0992-144.

[3] G. Roth, „Warum sind Lehren und Lernen so schwierig? Zeitschrift für Pädagogik 50 (2004) 4, S. 496-506," *Zeitschrift für Pädagogik*, Jg. 50, 2004, ISSN: 0044-3247. DOI: 10.25656/01:4823.

[4] IBM. „Was sind virtuelle Maschinen." (10. Juli 2024), Adresse: https://www.ibm.com/de-de/topics/virtualization.

[5] IBM. „Was ist Virtualisierung." (10. Juli 2024), Adresse: https://www.ibm.com/de-de/topics/virtual-machines.

[6] RedHat. „Was ist ein Hypervisor." (10. Juli 2024), Adresse: https://www.redhat.com/de/topics/virtualization/what-is-a-hypervisor.

[7] IBM. „Containers versus virtual machines (VMs): What's the Difference?" (10. Juli 2024), Adresse: https://www.ibm.com/think/topics/containers-vs-vms.

[8] R. P. Goldberg, *Architecture of Virtual Machines*, Billeria, MA, USA, 1973. DOI: 10.1145/1499586.1499669. Adresse: https://dl.acm.org/doi/pdf/10.1145/1499586.1499669.

[9] statcounter. „Mobile Operating System Market Share Worldwide." (10. Juli 2024), Adresse: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[10] Android Developers. „Platform architecture." (10. Juli 2024), Adresse: https://developer.android.com/guide/platform.

[11] B. G. Mateus und M. Martinez, „On the adoption, usage and evolution of Kotlin features in Android development," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, New York, NY, USA: ACM, 10052020, S. 1–12, ISBN: 9781450375801. DOI: 10.1145/3382494.3410676.

[12] Android Developer. „https://developer.android.com/about/versions." (10. Juli 2024), Adresse: https://developer.android.com/about/versions.

[13] Android Developers. „App manifest overview." (10. Juli 2024), Adresse: https://developer.android.com/guide/topics/manifest/manifest-intro.

[14] Google for Developers. „Meet Android Studio." (10. Juli 2024), Adresse: https://developer.android.com/studio/intro.

[15] RedHat. „What is an IDE?" (10. Juli 2024), Adresse: https://www.redhat.com/en/topics/middleware/what-is-ide.

[16] Google for Developers. „Extended controls, settings, and help." (10. Juli 2024), Adresse: https://developer.android.com/studio/run/emulator-extended-controls.

[17] Google for Developers. „Run apps on the Android Emulator." (10. Juli 2024), Adresse: https://developer.android.com/studio/run/emulator.

[18] Google for Developers. „android:exported." (10. Juli 2024), Adresse: https://developer.android.com/privacy-and-security/risks/android-exported.

[19] SQLite. „About SQLite." (10. Juli 2024), Adresse: https://www.sqlite.org/about.html.

[20] OWASP. „OWASP Mobile Top 10." (10. Juli 2024), Adresse: https://owasp.org/www-project-mobile-top-10/.

[21] Oracle. „1.4. Supported Host Operating Systems." (10. Juli 2024), Adresse: https://www.virtualbox.org/manual/ch01.html#hostossupport.

[22] Broadcom. „Supported host operating systems for Workstation Pro 16.x, 17.x and Workstation Player 16.x, 17.x." (10. Juli 2024), Adresse: https://knowledge.broadcom.com/external/article?legacyId=80807.

[23] Broadcom. „VMware Fusion 13." (10. Juli 2024), Adresse: https://www.vmware.com/products/fusion/fusion-evaluation.html.

[24] VMware. „VMware Security Advisories." (10. Juli 2024), Adresse: https://support.broadcom.com/web/ecx/security-advisory?.

[25] Oracle. „Critical Patch Updates, Security Alerts and Bulletins.“ (10. Juli 2024), Adresse: https://www.oracle.com/security-alerts/.

[26] Oracle. „Changelog for VirtualBox 7.0.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/wiki/Changelog.

[27] Oracle. „Download VirtualBox (Old Builds): VirtualBox 6.0.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/wiki/Download_Old_Builds_6_0.

[28] Broadcom. „VMware Workstation 17.5.1 Pro Release Notes.“ (10. Juli 2024), Adresse: https://docs.vmware.com/en/VMware-Workstation-Pro/17.5.1/rn/vmware-workstation-1751-pro-release-notes/index.html.

[29] NIST. „CVE-2020-6100 Detail.“ (10. Juli 2024), Adresse: https://nvd.nist.gov/vuln/detail/CVE-2020-6100.

[30] Broadcom. „VMware Workstation Pro: Now Available Free for Personal Use.“ (10. Juli 2024), Adresse: https://blogs.vmware.com/workstation/2024/05/vmware-workstation-pro-now-available-free-for-personal-use.html.

[31] Oracle. „Welcome to VirtualBox.org!“ (10. Juli 2024), Adresse: https://www.virtualbox.org/.

[32] GNU. „GNU General Public License.“ (10. Juli 2024), Adresse: https://www.gnu.org/licenses/gpl-3.0.html.

[33] Microsoft. „Get a Windows 11 development environment.“ (10. Juli 2024), Adresse: https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/.

[34] Microsoft. „Windows 11 Home und Pro.“ (10. Juli 2024), Adresse: https://www.microsoft.com/de-de/windows/windows-11-specifications.

[35] Microsoft. „Systemanforderungen für die Installation von Windows 10.“ (10. Juli 2024), Adresse: https://www.microsoft.com/de-de/windows/windows-10-specifications.

[36] Canonical Ltd. „Installation/SystemRequirements.“ (10. Juli 2024), Adresse: https://help.ubuntu.com/community/Installation/SystemRequirements.

[37] Canonical Ltd. „The Ubuntu lifecycle and release cadence.“ (10. Juli 2024), Adresse: https://ubuntu.com/about/release-cycle.

[38] Microsoft. „Windows 10 Home und Pro.“ (10. Juli 2024), Adresse: https://learn.microsoft.com/de-de/lifecycle/products/windows-10-home-and-pro.

[39] Microsoft. „Windows 11 Home und Pro.“ (10. Juli 2024), Adresse: https://learn.microsoft.com/de-de/lifecycle/products/windows-11-home-and-pro.

[40] Oracle. „Chapter 13. Security Guide.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/manual/ch13.html.

[41] Canonical Ltd. „Chapter 14. Known Limitations.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/manual/ch14.html.

[42] Android Developers. „Configure hardware acceleration for the Android Emulator.“ (10. Juli 2024), Adresse: https://developer.android.com/studio/run/emulator-acceleration.

[43] Red Hat. „OpenJDK im Vergleich zu Oracle JDK.“ (10. Juli 2024), Adresse: https://www.redhat.com/de/topics/application-modernization/openjdk-vs-oracle-jdk.

[44] CVE. „Security Vulnerabilities, CVEs, published since 2023-04-14.“ (10. Juli 2024), Adresse: https://www.cvedetails.com/vulnerability-list/vendor_id-93/product_id-19116/Oracle-JDK.html.

[45] CVE. „Security Vulnerabilities, CVEs, published since 2023-04-14.“ (10. Juli 2024), Adresse: https://www.cvedetails.com/vulnerability-list/vendor%5C_id-93/product%5C_id-23642/Oracle-Openjdk.html.

[46] Oracle. „Oracle Java SE Support Roadmap.“ (10. Juli 2024), Adresse: https://www.oracle.com/de/java/technologies/java-se-support-roadmap.html.

[47] Red Hat. „OpenJDK Life Cycle and Support Policy.“ (10. Juli 2024), Adresse: https://access.redhat.com/articles/1299013#OpenJDK_Life_Cycle.

[48] GitHub. „jadx.“ (10. Juli 2024), Adresse: https://github.com/skylot/jadx/releases.

[49] WithSecure. „drozer.“ (10. Juli 2024), Adresse: https://labs.withsecure.com/tools/drozer.

[50] Oracle. „Installation of the JDK on Linux Platforms.“ (10. Juli 2024), Adresse: https://docs.oracle.com/en/java/javase/11/install/installation-jdk-linux-platforms.html#GUID-CF001E7F-7E0D-49D4-A158-9CF3ED4C247C.

[51] Ubuntu. „Ubuntu-Packages.“ (10. Juli 2024), Adresse: https://packages.ubuntu.com/search?keywords=openjdk.

[52] Android Developers. „Android 14 features and changes list.“ (10. Juli 2024), Adresse: https://developer.android.com/about/versions/14/summary.

[53] Android Developers. „Emulator release notes.“ (10. Juli 2024), Adresse: https://developer.android.com/studio/releases/emulator.

[54] GitHub. „dex2jar.“ (10. Juli 2024), Adresse: https://github.com/pxb1988/dex2jar/tree/2.x.

[55] GitHub. „enjarify.“ (10. Juli 2024), Adresse: https://github.com/google/enjarify.

[56] GitHub. „smali/baksmali.“ (10. Juli 2024), Adresse: https://github.com/JesusFreke/smali.

[57] GitHub. „JD-GUI.“ (10. Juli 2024), Adresse: https://github.com/java-decompiler/jd-gui.

[58] GitHub. „jadx.“ (10. Juli 2024), Adresse: https://github.com/skylot/jadx.

[59] GitHub. „ClassyShark.“ (10. Juli 2024), Adresse: https://github.com/google/android-classyshark.

[60] Kalen Kinloch. „Bytecode Viewer.“ (10. Juli 2024), Adresse: https://bytecodeviewer.com/.

[61] GitHub. „WebView.“ (10. Juli 2024), Adresse: https://developer.android.com/reference/android/webkit/WebView.

[62] GitHub. „Inspeckage - Android Package Inspector.“ (10. Juli 2024), Adresse: https://github.com/ac-pm/Inspeckage.

[63] GitHub. „drozer.“ (10. Juli 2024), Adresse: https://github.com/WithSecureLabs/drozer.

[64] Google for Developers. „Android Debug Bridge (adb).“ (10. Juli 2024), Adresse: https://developer.android.com/tools/adb.

[65] Google for Developers. „Logcat command-line tool.“ (10. Juli 2024), Adresse: https://developer.android.com/tools/logcat.

[66] mitmproxy. „mitmproxy is a free and open source interactive HTTPS proxy.“ (10. Juli 2024), Adresse: https://mitmproxy.org/.

[67] ZAP. „Server Certificates.“ (10. Juli 2024), Adresse: https://www.zaproxy.org/docs/desktop/addons/network/options/servercertificates/.

[68] PortSwigger. „SQL injection.“ (10. Juli 2024), Adresse: https://portswigger.net/web-security/sql-injection.

[69] Android Developers. „Hardcoded Cryptographic Secrets.“ (10. Juli 2024), Adresse: https://developer.android.com/privacy-and-security/risks/hardcoded-cryptographic-secrets.

[70] Android Developers. „SQL injection.“ (10. Juli 2024), Adresse: https://developer.android.com/privacy-and-security/risks/sql-injection.

[71] Android Developers. „android:debuggable.“ (10. Juli 2024), Adresse: https://developer.android.com/privacy-and-security/risks/android-debuggable.

[72] Android Developers. „Backup leaks.“ (10. Juli 2024), Adresse: https://developer.android.com/privacy-and-security/risks/backup-leaks.

[73] PortSwigger. „Burp Suite system requirements.“ (10. Juli 2024), Adresse: https://portswigger.net/burp/documentation/desktop/getting-started/system-requirements.

[74] Android-x86. „Android-x86 Run Android on your PC - VirtualBox How To.“ (10. Juli 2024), Adresse: https://www.android-x86.org/documentation/virtualbox.html.

[75] VirtualBox. „Oracle® VM VirtualBox® User Manual.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/manual/UserManual.html#hostossupport.

[76] VirtualBox. „End-user documentation.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/wiki/End-user_documentation.

[77] VirtualBox. „Oracle® VM VirtualBox® User Manual.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/manual/UserManual.html#hostcpurequirements.

[78] Virtualbox. „Installing Oracle VM VirtualBox and Extension Packs.“ (10. Juli 2024), Adresse: https://www.virtualbox.org/manual/ch01.html#intro-installing.

[79] Broadcom Inc. „Securing Virtual Machines.“ (10. Juli 2024), Adresse: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-60025A18-8FCF-42D4-8E7A-BB6E14708787.html.

[80] Android$_{D}$*evelopers*. „Sign your app.“ (10. Juli 2024), Adresse: https://developer.android.com/studio/publish/app-signing.

[81] T. Theis, *Einstieg in Kotlin - Apps entwickeln mit Android Studio*. Bonn: Rheinwerk Verlag, 2019, ISBN: 978-3-836-26873-8.

[82] Kodeco. „SQLite Database.“ (10. Juli 2024), Adresse: https://www.kodeco.com/books/saving-data-on-android/v2.0/chapters/3-sqlite-database.

[83] Sec Zone. „Content Provider in Android.“ (10. Juli 2024), Adresse: https://medium.com/@sec.zone64/content-provider-in-android-a9d0450bdeab.