HOCHSCHULE
MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Frau
Sandra Lindner, B.Sc.

Digitale Spuren in IoT-Umgebungen: Auswertung von Speicher und Netzwerkverkehr der Bosch Smart Camera App

Mittweida, September 2025

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Digitale Spuren in IoT-Umgebungen: Auswertung von Speicher und Netzwerkverkehr der Bosch Smart Camera App

Autorin:
Sandra Lindner

Studiengang:
Cybercrime/Cybersecurity

Seminargruppe:
CY23wC-M

Erstprüfer:
Prof. Dr. rer. nat. Dirk Labudde

Zweitprüfer:
Felix Fischer, M.Sc.

Einreichung:
Mittweida, 24.09.2025

Verteidigung/Bewertung:
Mittweida, 2025

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Digital Traces in IoT Environments: Analysis of Storage and Network Traffic of the Bosch Smart Camera App

Author:
Sandra Lindner

Course of Study:
Applied Computer Science

Seminar Group:
CY23wC-M

First Examiner:
Prof. Dr. rer. nat. Dirk Labudde

Second Examiner:
Felix Fischer, M.Sc.

Submission:
Mittweida, 24.09.2025

Defense/Evaluation:
Mittweida, 2025

Bibliografische Beschreibung

Lindner, Sandra:
Digitale Spuren in IoT-Umgebungen: Auswertung von Speicher und Netzwerkverkehr der Bosch Smart Camera App. – 2025. – 89 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Masterarbeit, 2025.

Referat

In dieser Arbeit wird anhand der Bosch Eyes Innenkamera II und der dazugehörigen Bosch Smart Camera App exemplarisch untersucht, welche forensischen Spuren in IoT-Kamerasystemen entstehen. Ziel ist es, herauszufinden, welche Daten während der Nutzung erzeugt und übertragen werden und wo diese gespeichert sind. Die analysierten Daten werden im Hinblick auf ihre forensische Relevanz bewertet, um zu prüfen, inwieweit diese Rückschlüsse auf das Nutzerverhalten oder potenzielle Sicherheitsrisiken zulassen. Zur Analyse des Netzwerks wurden ein Netzwerk- und ein Portscan durchgeführt sowie der Netzwerkverkehr aufgezeichnet. Durch einen MITM-Angriff wurde versucht, verschlüsselte Kommunikation sichtbar zu machen. Es erfolgte eine statische und eine dynamische Untersuchung der App. Zudem wurden die logisch gesicherten Daten mit App-Bezug ausgewertet. Für die Untersuchungen wurden anerkannte Werkzeuge wie Nmap, Wireshark, Burp Suite, MobSF und das ADB-Tool genutzt. Die Ergebnisse dieser Untersuchungen zeigen, dass sich aus Netzwerkverkehr, App-Nutzung und den von der Anwendung gespeicherten Daten eine Reihe von forensisch relevanten Informationen gewinnen lassen. Dazu gehören unter anderem Nutzerdaten, Synchronisationsprozesse und Schwachstellen. Die Arbeit kommt zu dem Schluss, dass IoT-Geräte und ihre Apps, wie die Bosch Eyes Innenkamera II und die dazugehörige Bosch Smart Camera App, nicht nur praktische Funktionen bieten, sondern auch relevante Daten für forensische Analysen liefern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abo . . . . . Abonnement

ACK . . . . . Acknowledgement

ADB . . . . . Android Debug Bridge

adbd . . . . . Android Debug Bridge Daemon

AES . . . . . Advanced Encryption Standard

AP . . . . . Access Point

API . . . . . Application Programming Interface

APK . . . . . Android Package Kit

ASCII . . . . . American Standard Code for Information Interchange

AVD . . . . . Android Virtual Device

AWS . . . . . Amazon Web Services

BLE . . . . . Bluetooth Low Energy

BSI . . . . . Bundesamt für Sicherheit in der Informationstechnik

CA . . . . . Certificate Authority

CBC . . . . . Cipher Block Chaining

CDN . . . . . Content Delivery Network

CIDR . . . . . Classless Inter-Domain Routing

CLI . . . . . Command Line Interface

CoAP . . . . . Constrained Application Protocol

CPU . . . . . Central Processing Unit

CSRF . . . . . Cross-Site-Request-Forgery

CSS . . . . . Cascading Style Sheets

CVE . . . . . Common Vulnerabilities and Exposures

CVSS . . . . . Common Vulnerability Scoring System

DAC . . . . . Discretionary Access Control

dirb . . . . . Directory Buster

DNS . . . . . Domain Name System

ECB . . . . . Electronic Codebook

EDRM . . . . . Electronic Discovery Reference Model

ffuf ..... Fuzz Faster U Fool

Fin ..... Finish

GCM ..... Galios/Counter Mode

GMT ..... Greenwich Mean Time

GPS ..... Global Positioning System

GUI ..... Graphical User Interface

hCAPTCHA ..... Human Completely Automated Public Turing test to tell Computers and Humans Apart

HTML ..... Hypertext Markup Language

HTTP ..... Hypertext Transfer Protocol

HTTPS ..... Hypertext Transfer Protocol Secure

I/O ..... Input/Output

ID ..... Identification

IoE ..... Internet of Everything

IoT ..... Internet of Things

IP ..... Internet Protocol

IPA ..... iOS App Package

IPv4 ..... Internet Protocol Version 4

IPv6 ..... Internet Protocol Version 6

IT ..... Informationstechnik

IV ..... Initialisierungsvektor

JPEG ..... Joint Photographic Experts Group

JSON ..... JavaScript Object Notation

JTAG ..... Joint Test Action Group

JWT ..... JSON Web Token

LAN ..... Local Area Network

MAC ..... Mandatory Access Control

MAC-Adresse ... Media Access Control-Adresse

MASVS ..... Mobile Application Security Verification Standard

MD5 ..... Message-Digest Algorithm 5

MITM ..... Man-in-the-Middle

MobSF ..... Mobile Security Framework

MQTT . . . . . Message Queuing Telemetry Transport

MVVM . . . . . Model View ViewModel

NIST . . . . . National Institute of Standards and Technology

Nmap . . . . . Network Mapper

NSE . . . . . Nmap Scripting Engine

NTP . . . . . Network Time Protocol

NVM . . . . . Non-Volatile Memory

OAEP . . . . . Optimal Asymmetric Encryption Padding

OAuth . . . . . Open Authorization

OHTTP . . . . . Oblivios HTTP

OS . . . . . Operating System

OWASP . . . . . Open Web Application Security Project

PC . . . . . Personal Computer

pcap . . . . . packet capture

PIN . . . . . Persönliche Identifikationsnummer

PKCE . . . . . Proof Key for Code Exchange

PKI . . . . . Public Key Infrastructure

Protobuf . . . . . Protocol Buffer

pSAP . . . . . Prepare-Secure-Analyze-Present

QUIC . . . . . Quick UDP Internet Connections

RAM . . . . . Random Access Memory

RSA . . . . . Rivest–Shamir–Adleman

SAP . . . . . Secure-Analyze-Present

SD . . . . . Secure-Digital

SDK . . . . . Software Development Kit

SELinux . . . . . Security-Enhanced Linux

SHA . . . . . Secure Hash Algorithm

SMS . . . . . Short Message Service

SNI . . . . . Server Name Indication

SOA . . . . . Start of Authority

SQL . . . . . Structured Query Language

SQLite . . . . . Structured Query Language lite

SSH . . . . . Secure Shell

SSL . . . . . Secure Sockets Layer

SYN . . . . . synchronize

TCP . . . . . Transmission Control Protocol

Telnet . . . . . teletype network

TLS . . . . . Transport Layer Security

TLSv1.2 . . . . . Transport Layer Security Version 1.2

TLSv1.3 . . . . . Transport Layer Security Version 1.3

UART . . . . . Universal Asynchronous Receiver Transmitter

udev . . . . . userspace /dev

UI . . . . . User Interface

URL . . . . . Uniform Resource Locator

USB . . . . . Universal Serial Bus

UUID . . . . . Universally Unique Identifier

VM . . . . . Virtuelle Maschine

WLAN . . . . . Wireless Local Area Network

WOFF2 . . . . . Web Open Font Format 2.0

XML . . . . . Extensible Markup Language

XSS . . . . . Cross-Site-Scripting

1 Einleitung

Die Digitalisierung durchdringt nahezu alle Lebensbereiche. Besonders die zunehmende Verbreitung von Geräten des Internet of Things (IoT) in privaten Haushalten, Unternehmen und öffentlichen Infrastrukturen führt zu einer stetig wachsenden Menge digitaler Spuren, die forensisch bedeutsam sein können. Diese Spuren entstehen bei alltäglichen Prozessen, etwa durch die Kommunikation zwischen Sensoren, Kameras und Cloud-Systemen und gewinnen in der Sicherheitsforschung wie auch in der Kriminalistik zunehmend an Bedeutung. [1, S. 3-4]

Mit dem Aufkommen smarter Geräte wie beispielsweise netzwerkfähiger Sicherheitskameras entstehen nicht nur neue Anforderungen an Datenschutz und Informationstechnik (IT)-Sicherheit, sondern auch zusätzliche Möglichkeiten zur digitalen Beweissicherung. In der forensischen Praxis werden digitale Spuren heute als zentrale Informationsquelle betrachtet, sowohl zur Aufklärung sicherheitsrelevanter Vorfälle als auch zur strafrechtlichen Verfolgung. [2, S. 113-166, 3]

Gleichzeitig bringt die Untersuchung solcher Spuren erhebliche Herausforderungen mit sich. Die technischen Gegebenheiten moderner IoT-Umgebungen, darunter verteilte Speicherorte, verschlüsselte Datenströme und die Nutzung von Cloud-Diensten, erschweren die klassische forensische Analyse erheblich [4]. Dies zeigt sich besonders deutlich bei Geräten, die mehrere Schnittstellen kombinieren – etwa Sicherheitskameras, die über mobile Apps gesteuert werden, netzwerkbasiert kommunizieren und proprietäre Speicherstrukturen nutzen.

Ein konkretes Beispiel hierfür ist die Bosch Smart Camera App, die in dieser Arbeit exemplarisch untersucht wird. Sie steht stellvertretend für moderne IoT-Videosysteme, die eine Vielzahl forensisch relevanter Spuren erzeugen. Diese Daten können sowohl im lokalen Gerätespeicher als auch im Netzwerkverkehr festgestellt werden. Die Relevanz solcher Analysen zeigt sich nicht nur im forensischen Alltag, sondern auch in zahlreichen aktuellen Abschlussarbeiten und Forschungsprojekten im deutschsprachigen Raum [5, 6, 7, 8].

Ziel dieser Arbeit ist es aufzuzeigen, welche Arten von digitalen Spuren im Zusammenspiel von App, Kamera und Infrastruktur entstehen, wie diese technisch gesichert werden können und welche Herausforderungen sich dabei für die forensische Praxis ergeben. Weiterhin soll eine Relevanzeinschätzung der gesicherten Spuren erfolgen.

Für eine klare Nachvollziehbarkeit der Untersuchungsmethoden und der daraus abgeleiteten Erkenntnisse ist diese Arbeit in neun Kapitel gegliedert.

Die Kapitel im Überblick:
Kapitel 2 enthält theoretische Grundlagen zur digitalen Forensik und deren Vorgehensmodelle sowie Informationen zum IoT und dem Vorgehen bei der forensischen Analyse in diesem Bereich.
Kapitel 3 bietet einen Überblick über die genutzten Methoden und die zur Analyse der Bosch Eyes Innenkamera II und der Bosch Smart Camera App verwendete Software.

Kapitel 4 liefert eine Zusammenfassung der Internetrecherche zu Schwachstellen und Eigenschaften sowie zur Funktionsweise der Bosch Eyes Innenkamera II und der Bosch Smart Camera App. Diese Internetrecherche dient als Grundlage der weiteren Analyse.
Kapitel 5 beschreibt die Analyse des Netzwerkverkehrs der Bosch Eyes Innenkamera II und der Bosch Smart Camera App sowie deren Ergebnisse.
Kapitel 6 widmet sich der Appanalyse der Bosch Smart Camera App und deren Ergebnisse.
Kapitel 7 liefert eine Zusammenfassung der in Kapitel 5 und Kapitel 6 erlangten Ergebnisse und eine Bewertung ihrer forensischen Relevanz.
Kapitel 8 enthält die Darstellung der Probleme und Grenzen bei den Untersuchungen der Bosch Eyes Innenkamera II und der Bosch Smart Camera App.
Kapitel 9 liefert abschließend eine Kurzzusammenfassung der Ergebnisse und zeigt einige Ansätze für zukünftige Untersuchungen auf.

2 Grundlagen

Dieses Kapitel bietet einen Einblick in das Internet der Dinge. Zudem wird auf wichtige Begriffe, Methodiken und Vorgehensmodelle der digitalen Forensik eingegangen. Diese können mit Anpassungen auch im IoT-Bereich angewendet werden. Zusätzlich liefert das Kapitel eine praxisnahe Beschreibung des Vorgehens bei der forensischen Analyse von IoT-Systemen.

2.1 Digitale Forensik und digitale Spuren

In der heutigen Zeit werden in allen Bereichen der Gesellschaft immer mehr digitale Geräte genutzt. Damit steigt die Menge an Daten, die auf unterschiedlichste Art gespeichert und verarbeitet werden, rasant. Damit verbunden wird auch die Erkennung, Sicherung und Auswertung digitaler Spuren im Rahmen der Aufklärung von Sicherheitsvorfällen und Cybercrimetaten immer wichtiger. Mit diesem Teilgebiet der Forensik beschäftigt sich die *Digitale Forensik*. Als Synonyme zum Begriff *Digitale Forensik* finden sich auch die Begriffe *IT-Forensik*, *Computerforensik*, seltener auch *Personal Computer (PC)-Forensik* oder im englischsprachigen Bereich *digital forensics* oder *digital forensic science*. [1, S. 5–6]

Die *digitale Forensik* beschäftigt sich mit der Datenanalyse auf Datenträgern, in Computernetzen und in Cloudsystemen. IT-Forensik dient der Aufklärung von Vorfällen nach strengen methodischen Vorgaben. Dabei sollte bereits im Vorfeld von solchen Ereignissen eine strategische Vorbereitung durch den Betreiber der Anlage erfolgen, um bessere forensische Untersuchungsergebnisse zu erhalten. [1, S. 5f. 9, S. 8] Einige Modelle für ein solches strategisches Vorgehen werden im Abschnitt 2.3 kurz vorgestellt.

Digitale Spuren, englisch auch *digital evidence*, sind alle Daten, die in IT-Systemen gespeichert, verarbeitet oder übertragen werden. Sie lassen Rückschlüsse auf Nutzerhandlungen oder Systemzustände zu. Dazu zählen Logdaten, Metadaten, gespeicherte Inhalte, Netzwerkverkehr und anderes. [10, S. 8, 1, S. 9f.]

In der Literatur wird neben digitalen Spuren auch von *digitalen Artefakten* gesprochen. Bei diesen handelt es sich um eine Sammlung verwandter digitaler Spuren, die eine digitalforensisch auswertbare Einheit bilden. Um zu verdeutlichen, was damit gemeint ist, bieten sich Protokolldateien an. Dort wäre die einzelne digitale Spur ein einzelner Eintrag, die Protokolldatei, die mehrere Einträge umfasst, hingegen ein digitales Artefakt. [1, S. 9f.]

Digitale Spuren haben elementare Eigenschaften, die bei der Untersuchung berücksichtigt werden müssen. Diese sind die Flüchtigkeit, die technische Vermeidbarkeit (zum Beispiel von Systemdaten), die Manipulierbarkeit und die Kopierbarkeit. *Digitale Spuren* können auf Basis ihrer Flüchtigkeit in drei Gruppen eingeteilt werden. Die erste Gruppe sind die *persistenten Daten*. Diese sind dauerhaft gespeichert und können damit auch nach dem Beenden des Programms oder dem Ausschalten eines Systems gesichert werden. Die zweite Gruppe sind *semipersistente Daten*. Diese Daten werden nur für einen begrenzten Zeitraum gespeichert und gehen nach dem Neustart oder dem Beenden des Programms verloren. *Flüchtige Daten*

sind nur für die Dauer der aktuellen Sitzung bzw. der aktuellen Ausführung des Programms im Speicher und gehen danach verloren. Beispiele für *persistente Daten* sind Daten, die auf der Festplatte gespeichert sind. *Semipersistente Daten* findet man hingegen zum Beispiel im Arbeitsspeicher, der auch als Random Access Memory (RAM) bezeichnet wird, oder im Cache-Speicher. Dies ist ein Zwischenspeicher, der dazu dient, den Zugriff von Mikroprozessoren auf häufig verwendete Daten zu beschleunigen. *Flüchtige Daten* sind beispielsweise Netzwerkdaten und Prozessorregister. [1, S. 55-57, 10, S. 9, 11, S. 101]

Aus diesen Darstellungen geht hervor, dass nicht alle Daten bei ausgeschaltetem System gesichert werden können. Bei der Analyse des ausgeschalteten Systems spricht man von *Post-Mortem-Forensik*, *Offline-Forensik* oder auch *Dead-Forensik*. Bei dieser kann man nur *persistente Daten* sichern. [1, S. 57f.] Eine andere Möglichkeit der Datensicherung ist, die Daten bei laufendem System zu sichern. Dabei können auch *flüchtige* und *semipersistente Daten* erfasst werden. Dies kann jedoch zu Veränderungen am System führen, die protokolliert werden müssen. Ein solches Vorgehen bezeichnet man auch als *Live-Forensik* bzw. *Online-Forensik*. [1, S. 57]

2.2 Internet der Dinge

In diesem Abschnitt soll dargestellt werden, was das Internet der Dinge ist und welche Möglichkeiten und Risiken es mit sich bringt.

2.2.1 Begiffsbestimmung

Mit dem Begriff IoT oder deutsch: *Internet der Dinge* bezeichnet man ein Netzwerk von physischen Objekten, die miteinander über verschiedene Kommunikationsschnittstellen verbunden sind. So können diese Objekte Daten versenden oder Kommandos empfangen. Dabei kann es sich um eine Vielzahl verschiedener Geräte handeln. Die Bandbreite reicht von ganzen Steuerungseinheiten von Industriemaschinen über Haushaltsgeräte bis hin zu Internet Protocol (IP)-fähigen Kameras oder einzelnen Sensoren. Solche Geräte machen Abläufe schneller, effizienter und bequemer. So melden Kameras Bewegungen selbstständig, Staubsauger saugen zu bestimmten Zeiten und die Kaffeemaschine hat zum Frühstück den Kaffee schon fertig. In der Industrie bieten IoT-Lösungen vielfältige Möglichkeiten, Prozesse zu automatisieren und somit produktiver und effizienter zu gestalten. Man spricht dabei auch von der Industrie 4.0, die durch die Einhaltung vier wesentlicher Prinzipien gekennzeichnet ist [12, S. 997-999]:

1. die Vernetzung von Mensch, Maschine sowie Steuerung und Sensoren
2. transparente Aufnahme von Daten und Parametern zur Optimierung von Produktions- und Unternehmensprozessen
3. Unterstützung für menschliche Arbeitskräfte
4. dezentrale Entscheidungen von kleinen Steuereinheiten [12, S. 997-999]

Bei Geräten, die eine solche Vernetzung ermöglichen, spricht man auch von *IoT-Devices* oder *IoT-Geräten*. Diese verfügen meist über einen bzw. mehrere Sensoren zum Beispiel zur Messung von Bewegung, Temperatur oder zur Aufnahme von Bildern. Des Weiteren

haben sie verschiedene Kommunikationsschnittstellen zum Beispiel für Local Area Network (LAN), Wireless Local Area Network (WLAN) oder Mobilfunk. Zunehmend wird auch Bluetooth-Kommunikation genutzt. Um eine vollständige Funktionsfähigkeit der Geräte zu ermöglichen, sind diese im Regelfall mit einem Server verbunden, der die Daten empfängt, die Kontrollkommandos absetzt und die angeschlossenen Geräte verwaltet. Eine weitere Bezeichnung, die für IoT in der Literatur auftaucht, ist der von Cisco geprägte Begriff Internet of Everything (IoE). Dieser sollte ursprünglich eine Weiterentwicklung des IoT-Ansatzes sein und den Schwerpunkt auf die intelligente Vernetzung von kleinen Einheiten legen. Er wird heute jedoch kaum noch verwendet und wird als Synonym zu IoT gesehen. [12, S. 999]

2.2.2 Forensische Analyse in IoT-Umgebungen

Nicht nur in der Industrie benutzt man solche Technologien. Auch im privaten Bereich verwenden immer mehr Menschen vernetzte Geräte, um das tägliche Leben einfacher zu machen. Unabhängig davon, ob Sicherheitskameras, Küchengeräte oder Staubsaugroboter in immer mehr Haushalten finden sich IoT-Geräte. Auch hier fällt eine Vielzahl von Daten an. Die Geräte erheben, nutzen und speichern Bewegungsdaten, Bilder, Logins, Steuerbefehle und andere Daten. Diese können genutzt werden, um Verhaltensweisen und Zustände zu rekonstruieren und so wichtige Beweise für Ermittlungen liefern. [13, S. S22f.]

Aus diesem Grund ist es wichtig, sich mit der forensischen Sicherung und Analyse solcher Daten zu beschäftigen. Jedoch stößt man dabei schnell auf ein zentrales Problem. Die große Heterogenität der Systeme und die fehlende Standardisierung in diesem Bereich erschweren forensische Untersuchungen. So nutzen unterschiedliche Hersteller teils nicht kompatible oder nicht dokumentierte Schnittstellen, durch die Daten ausgelesen werden können. Auch speichern die einzelnen Systeme Daten an unterschiedlichen Orten. Dies kann lokal, auf einem mobilen Endgerät oder in einer Cloud erfolgen. Zudem nutzen die unterschiedlichen IoT-Systeme verschiedenste Protokolle. Gerade in der Gebäudeautomatisierung tauchen so immer wieder neue Kommunikationsprotokolle auf, die teilweise nicht über offene Standards verfügen und somit nicht von unabhängigen Experten überprüft werden können. Dies sollte bei einer forensischen Untersuchung immer im Hinterkopf behalten werden, auch wenn der Trend inzwischen zu Kommunikationsprotokollen mit offenen Standards tendiert. Einige Beispiele für Protokolle mit offenen Standards sind Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Message Queuing Telemetry Transport (MQTT) und Constrained Application Protocol (CoAP). [13, S. S23f. 14, S. 350]

Das HTTP-Protokoll dient dazu, den Datenaustausch zwischen Webbrowsern und Webservern zu regeln. Es bietet damit die Grundlage für den Datenaustausch im Internet. [14, S. 58]

Das HTTPS-Protokoll ist eine Erweiterung des HTTP-Protokolls mit Verschlüsselung und Verifizierung. Es gestaltet dadurch den Datenaustausch zwischen Webbrowsern und Webservern sicher. [14, S. 58]

Das MQTT-Protokoll ist ein für IoT-Umgebungen entwickeltes Protokoll. Es dient der Nachrichtenübermittlung zwischen einem Sender und unterschiedlich vielen Empfängern. [14, S. 343-345]

Beim CoAP-Protokoll handelt es sich ebenfalls um ein für den IoT-Bereich entwickeltes Protokoll. Es dient der Nachrichtenübermittlung in ressourcenbeschränkten Einsatzszenarien, ermöglicht jedoch nur eine Punkt-zu-Punkt-Kommunikation. [14, S. 339-343]

Wie bereits aufgezeigt, erfolgt die Datenspeicherung in IoT-Umgebungen an unterschiedlichen Orten. Damit müssen bei der Speicheranalyse neben dem eigentlichen IoT-Gerät auch das dazugehörige Smartphone und alle anderen verbundenen Geräte untersucht werden. Je nach IoT-Gerät stehen dazu verschiedene Möglichkeiten zur Verfügung. So können Daten direkt über Hardwarezugriffstechniken vom Gerät extrahiert werden. Typische Hardwarezugriffsmöglichkeiten sind Universal Asynchronous Receiver Transmitter (UART), Joint Test Action Group (JTAG) und Chip-Off. Diese drei Techniken bieten bei forensischen Analysen wertvolle Möglichkeiten, auf die gespeicherten Informationen eines IoT-Geräts zuzugreifen. [13, S. S23f.]

UART ist eine serielle Schnittstelle, die es erlaubt, direkt mit der Konsole eines Geräts zu kommunizieren. Oft lässt sich darüber ohne Authentifizierung auf Systeminformationen, Bootloader oder sogar eine Root-Shell zugreifen. Bei einem Bootloader handelt es sich um ein kleines Programm, mit dessen Hilfe das Betriebssystem gestartet wird. Eine Root-Shell ist eine Schnittstelle zum Betriebssystem mit Root-Rechten. Sie verfügt über eine Programmiersprache, mit der unter anderem Prozesse und Programme gesteuert und auf Dateien zugegriffen werden kann. Root-Rechte bezeichnet man auch als Superuser-Rechte. Sie sind die höchste Art der Zugriffsrechte. Mit diesen hat der Nutzer Zugriff auf alle Systemressourcen und kann systemweite Änderungen vornehmen. Gerade bei unsachgemäß konfigurierten Geräten kann so ein direkter Zugriff auf Dateisysteme und Nutzerdaten erfolgen. Dies macht UART zu einer besonders niedrigschwelligen und oft ersten forensischen Zugriffsmöglichkeit. [15, S.59–80]

JTAG hingegen ist ein standardisiertes Debug-Interface. Diese Schnittstelle ermöglicht einen tiefen Zugriff auf das Innenleben eines Chips und dient normalerweise zur Fehlerbehebung. Über das Debug-Interface lassen sich der Arbeitsspeicher, laufende Prozesse und auch der Flash-Speicher direkt auslesen oder manipulieren. Bei einem Flash-Speicher handelt es sich um einen nicht flüchtigen, ständig mit Strom versorgten Speicher. Ein nicht flüchtiger Speicher wird kurz als Non-Volatile Memory (NVM) bezeichnet. Es ist möglich, Flash-Speicher in Blöcke aufzuteilen. Diese Speichereinheiten können gelöscht oder neu programmiert werden. Damit ist JTAG eine besonders mächtige Methode zur Analyse oder zum Dumping von Firmware. Mit dem Dumping von Firmware beschreibt man das Auslesen der Firmware eines Gerätes und die Speicherung der so gewonnenen Datei. Diese Kopie bezeichnet man auch als Dump. Bei der Firmware handelt es sich um eine spezielle Software, die direkt auf der Hardware des Gerätes läuft und deren Funktion steuert. Sie ist dafür verantwortlich, dass alle Komponenten des Gerätes zusammenarbeiten und enthält oft wichtige Daten und Steuerbefehle. Voraussetzung für die Nutzung von JTAG ist allerdings die Identifikation der JTAG-Pinbelegung sowie spezielle Hardware-Tools wie ein JTAGulator oder ein über OpenOCD angebundener Debugger. [15, S. 109–138, 16, S. 22, 47, 214, 246f.]

Eine weitere Zugriffsmöglichkeit stellt das sogenannte Chip-Off-Verfahren dar. Hierbei wird der Speicherchip physikalisch vom Mainboard entfernt und mit spezialisierten Lesegeräten oder einem zweiten Gerät ausgelesen. Diese Methode ist besonders dann von Bedeutung,

wenn weder UART noch JTAG zugänglich oder funktional sind, etwa bei stark beschädigten Geräten. Das technisch anspruchsvolle Verfahren ermöglicht eine vollständige Sicherung des Speicherinhalts und ist daher besonders in kriminaltechnischen Laboren verbreitet. Dieses Vorgehen erfordert fundierte Hardwarekenntnisse und ist kostenintensiver als die anderen beiden Zugriffsmöglichkeiten. Zudem wird bei einem Chip-Off das Gerät unwiederbringlich zerstört. [17, S. 100, 16, S.22, 47, 214, 248–250]

Auch die Analyse der zugehörigen Smartphone-App kann wichtige Spuren bringen. So können zum Beispiel Structured Query Language lite (SQLite)-Datenbanken, Logs und Cache fallrelevante Daten enthalten. [18, S. 547, 13, S. S24–S26, 16] So stellen Servida und Casey zum Beispiel bei der Untersuchung der *iSmartAlarm-App* fest, dass diese unter anderem Ereignisprotokolle, Cloud-Zugangsdaten und Thumbnails von Videostreams auf dem Gerät speichert. Bei Thumbnails handelt es sich um verkleinerte Abbildungen von Mediendateien. Diese dienen dazu, schnelle und ressourcenschonende Vorschauen zu ermöglichen. [19, 13, S. S25] Bei der Bosch Smart Camera App ist anzunehmen, dass ähnliche Artefakte (Videodaten, Benutzerbefehle, Verbindungslogs) in der App vorliegen. Dies muss in einer kontrollierten Testumgebung validiert werden.

Da IoT-Geräte ständig über das Netzwerk mit anderen Geräten kommunizieren, lohnt sich auch hier eine forensische Analyse der Netzwerkkommunikation. Gängige Methoden, um an versendete Pakete zu gelangen und das Netzwerk zu analysieren, sind Sniffing, Man-in-the-Middle (MITM)-Angriffe und Port-Scans. [13, S. S24, 14, S. 199]

Beim Sniffing werden Netzwerkpakete passiv mitgeschnitten, zum Beispiel mit Tools wie Wireshark oder spezialisierten Bluetooth Low Energy (BLE)-Sniffern. Dies erlaubt es, unverschlüsselte Daten, Steuerbefehle oder Geräteinteraktionen sichtbar zu machen. Gerade bei IoT-Geräten, die auf drahtlose Kommunikation (zum Beispiel ZigBee, BLE, WLAN) setzen, ist diese Technik besonders effektiv. So haben Tests beispielsweise gezeigt, dass über Bluetooth-Paketmitschnitte Steuerbefehle für smarte Glühbirnen rekonstruiert werden können. Dies stellt eine potenzielle Grundlage für Replay-Angriffe dar. Bei Replay-Angriffen werden durch Sniffing mitgeschnittene Pakete erneut an das System gesendet. Dadurch ist es Angreifern potenziell möglich, das entsprechende Gerät fernzusteuern. [15, S. 297–306, 14, S. 199, 20, S. 135]

Die MITM-Technik ermöglicht es, den Datenstrom zwischen zwei Kommunikationspartnern abzufangen und sogar zu manipulieren. In forensischen Testumgebungen kann dies helfen, verschlüsselte oder gesicherte Kommunikation sichtbar zu machen, indem der Verkehr gezielt umgeleitet wird. Dies kann mit Hilfe von Tools wie Burp Suite oder Domain Name System (DNS)-Spoofing-Methoden erfolgen. DNS-Spoofing ist eine Methode, bei der Angreifer die Namensauflösung im DNS manipulieren. Normalerweise übersetzt DNS menschenlesbare Domainnamen in numerische IP-Adressen, die Computer verstehen. Beim DNS-Spoofing fälschen Angreifer diese DNS-Antworten und leiten den Datenverkehr des Opfers auf eine bösartige oder gefälschte IP-Adresse um. Für IoT-Geräte, die häufig über das Netzwerk kommunizieren und auf bestimmte Server angewiesen sind, bedeutet dies, dass sie unwissentlich mit einem vom Angreifer kontrollierten Server kommunizieren könnten. Dies ermöglicht es dem Angreifer, die Kommunikation abzufangen, zu überwachen oder sogar zu manipulie-

ren, da das IoT-Gerät glaubt, mit dem vorgesehenen Dienst zu interagieren. Insbesondere Web-Interfaces von IoT-Geräten wie zum Beispiel Kameras sind häufig nicht ausreichend abgesichert und damit anfällig für solche Angriffe. [15, S. 211–213, 14, S. 199f. 20, S. 136f.]

Port-Scans sind ein klassisches Mittel zur Identifikation von offenen Netzwerkdiensten auf einem Zielgerät. Mit Programmen wie Network Mapper (Nmap) lässt sich prüfen, ob zum Beispiel Secure Shell (SSH), teletype network (Telnet), HTTP oder andere Dienste aktiv sind. Bei Telnet handelt es sich um ein Protokoll, das zur Fernsteuerung von Rechnern dient. Dabei werden alle Daten unverschlüsselt übertragen. Es gilt deshalb heute als unsicher. SSH stellt eine sichere Alternative zu Telnet dar. Hier läuft die Kommunikation verschlüsselt ab. Ein offener Port kann Hinweis auf einen potenziellen Angriffsvektor sein oder erklären, wie ein Angreifer ins System gelangt ist. Forensiker nutzen Port-Scans, um das Angriffspotenzial zu analysieren oder das Verhalten kompromittierter Systeme nachzuvollziehen. [15, S. 207, 14, S. 336-339]

In der Praxis ist eine Kombination dieser Methoden oft notwendig, um ein vollständiges Bild der Netzwerkaktivitäten zu erhalten. Die Auswertung der Kommunikation bietet dabei nicht nur Hinweise auf mögliche Angriffe, sondern auch auf die Art der genutzten Protokolle, beteiligte IP-Adressen und verwendete Authentifizierungsverfahren. Darüber hinaus lässt sich mit solchen Methoden feststellen, ob ein Gerät beispielsweise regelmäßig mit externen Servern kommuniziert, was Hinweise auf Datenabfluss oder Fernsteuerung liefern kann.

2.3 Vorgehensmodelle in der IT-Forensik

Wie bereits in Kapitel 2.1 erwähnt, ist für die Forensik ein strategisches Vorgehen von großer Bedeutung, um die Qualität der Ergebnisse zu gewährleisten. Daher orientiert sich die IT-Forensik an systematischen Modellen, um strukturiert, gerichtsfest und reproduzierbar zu arbeiten. [1, S. 19] In diesem Abschnitt sollen nun die wichtigsten theoretischen Vorgehensmodelle der IT-Forensik kurz vorgestellt und ein praxisnahes Modell für die forensische Analyse von IoT-Umgebungen aufgezeigt werden.

2.3.1 Theoretische Vorgehensmodelle der IT-Forensik

Die wesentlichen Vorgehensmodelle der IT-Forensik sind:

- das Secure-Analyze-Present (SAP)-Modell
- das Prepare-Secure-Analyze-Present (pSAP)-Modell
- das National Institute of Standards and Technology (NIST)-Modell
- das Bundesamt für Sicherheit in der Informationstechnik (BSI)-Modell
- das Electronic Discovery Reference Model (EDRM)-Modell
- das Casey-Modell.

Das SAP-Modell ist leicht verständlich und wird daher in der Praxis häufig genutzt. Es besteht aus drei Phasen. Diese sind in Abbildung 2.1 dargestellt. In der ersten Phase, der Sicherungsphase, werden alle Daten gesammelt und gesichert. In der Analysephase wird eine Sicherheitskopie der Daten angelegt. Danach korreliert der Forensiker diese und bewertet

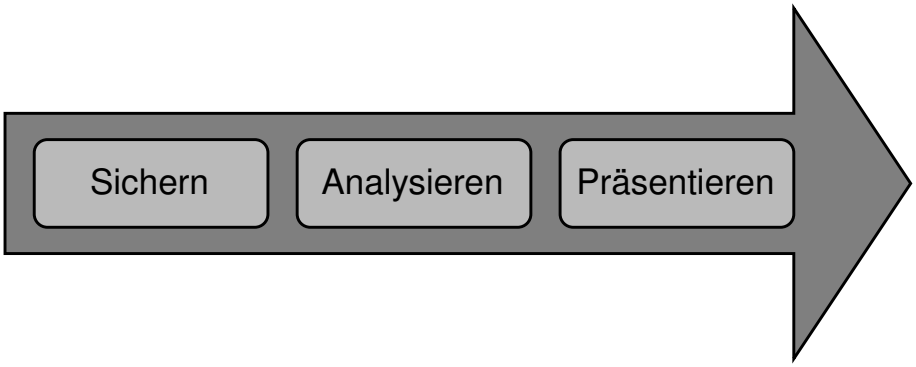

Abbildung 2.1: SAP-Modell nach [1, S. 20]

die Daten hinsichtlich ihrer Relevanz. Die letzte Phase ist die Präsentationsphase. In dieser werden die Untersuchungsergebnisse zusammengefasst und dokumentiert. Wichtig ist dabei, die Präsentation an die Zielgruppe anzupassen. [1, S. 19-21]

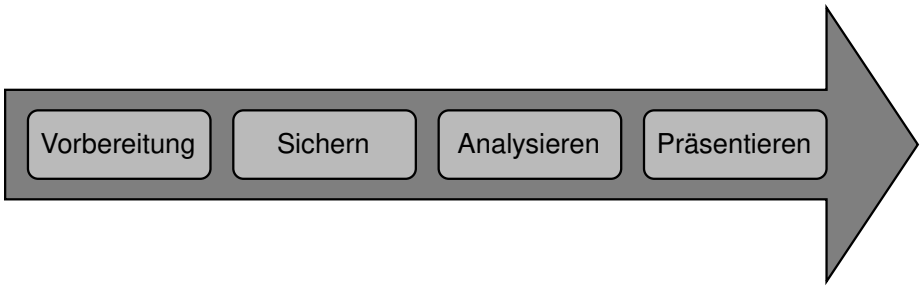

Abbildung 2.2: pSAP-Modell nach [1, S. 30]

Beim pSAP-Modell handelt es sich um eine Erweiterung des SAP-Modells. Es beinhaltet zusätzlich eine Prepare- bzw. Vorbereitungsphase. Dies ist in Abbildung 2.2 zu sehen. In dieser Phase werden strategische und technische Vorbereitungen getroffen wie zum Beispiel die Auswahl forensischer Tools und rechtliche Klärungen. Eine solche Vorbereitungsphase ist besonders bei komplexen Umgebungen wie im IoT-Bereich notwendig. Trotzdem ist das Modell noch leicht verständlich und praxisnah. [1, S. 30f.]

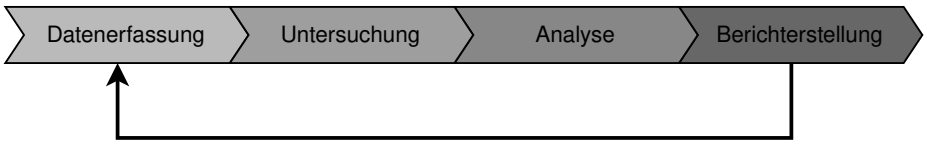

Abbildung 2.3: NIST-Modell nach [21, S. 25]

Das NIST-Modell stammt, wie aus dem Namen hervorgeht, vom US-amerikanischen National Institute of Standards and Technology und stellt ein standardisiertes, international anerkanntes Vorgehensmodell dar. Es weist viele Parallelen zum SAP-Modell auf. Das NIST-Modell umfasst vier Phasen: die Datenerfassung, die Untersuchung, die Analyse und die Berichterstellung. Die Struktur des Modells ist in Abbildung 2.3 dargestellt. Im Rahmen der Datenerfassung werden die Daten gesammelt. Bei der Untersuchung werden diese bewertet und die relevanten Inhalte extrahiert. Im Rahmen der Analyse werden die so gewonnen Informationen genutzt, um logische Schlussfolgerungen zu treffen. Untersuchung und Analyse

in diesem Modell entsprechen also inhaltlich der Analysephase im SAP- bzw. pSAP-Modell. In der vierten Phase erfolgt zum Abschluss eine Dokumentation des Vorgehens und der Ergebnisse. [1, S. 25, 21, S. 25-32]

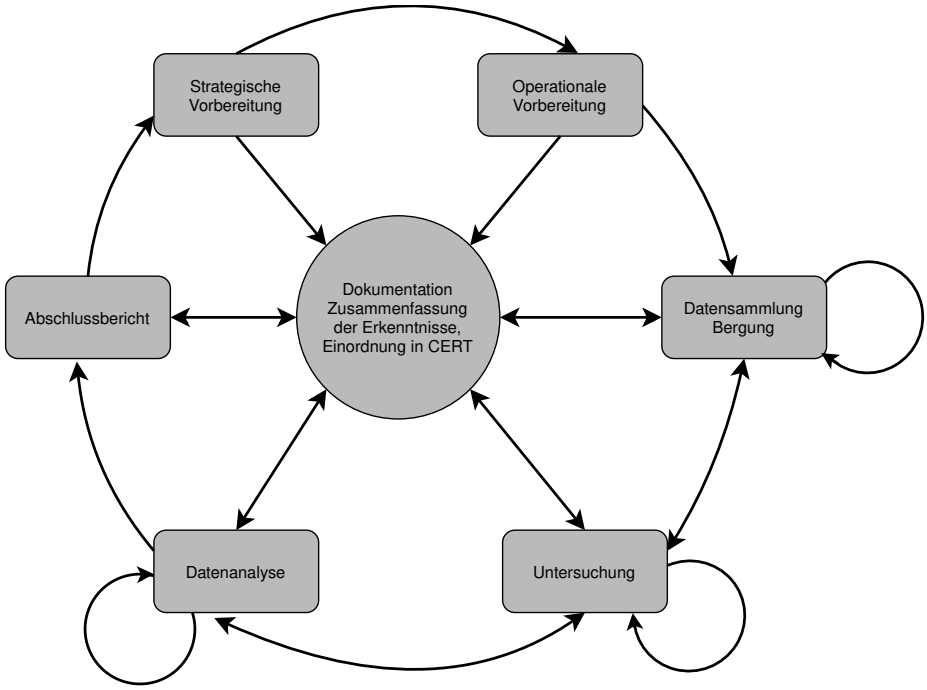

Abbildung 2.4: BSI-Modell nach [1, S. 21, 9, S. 61]

Im Vergleich zu den bereits vorgestellten Modellen verlangt das BSI-Modell ein sehr komplexes und detailliert beschriebenes Vorgehen zur forensischen Untersuchung. Dabei berücksichtigt das BSI-Modell auch Vorbereitungen, die bereits vor dem Eintreten eines Ereignisses getroffen werden können, um Vorfälle zu verhindern. In der Abbildung 2.4 ist dieses Modell dargestellt. Die einzelnen Schritte werden im Folgenden näher beschrieben. Zur Vorbereitung gehören die strategische und die operative Vorbereitung. Im Rahmen der strategischen Vorbereitung werden Maßnahmen getroffen, um Vorfälle zu verhindern. Die operative Vorbereitung umfasst alle Maßnahmen, die nach dem Eintreten des Ereignisses und vor der Datensammlung erfolgen. Der nächste Schritt ist die eigentliche Datensammlung. Dort werden alle wichtigen Daten von potenziell betroffenen Komponenten gesammelt. Anschließend wird die Datenextraktion vorgenommen. Dabei werden von den wichtigen Daten nur die herausgezogen, die tatsächlich mit dem Vorfall in Verbindung stehen. Danach wird die Datenanalyse durchgeführt. In diesem Schritt werden die extrahierten Daten inhaltlich untersucht, zeitlich eingeordnet und in Zusammenhang gebracht. Der Dokumentation des Vorgehens und der Ergebnisse kommt im BSI-Modell eine besondere Bedeutung zu. Die Dokumentation muss parallel zu den anderen Schritten fortlaufend im gesamten Prozess erfolgen. [1, S. 21f. 9, S. 60-65]

Das EDRM-Modell ist ein Modell aus dem US-amerikanischen Bereich, das für sogenannte *eDiscovery-Prozesse* entwickelt wurde. Unter eDiscovery-Prozessen versteht man die Identifikation, Aufbereitung sowie die Bereitstellung von elektronischen Informationen mit einem bestimmten Sachverhaltsbezug. Der Begriff wird jedoch auch im Kontext der Massendatenverarbeitung genutzt. Dort beschreibt man damit das Auffinden und Strukturieren von

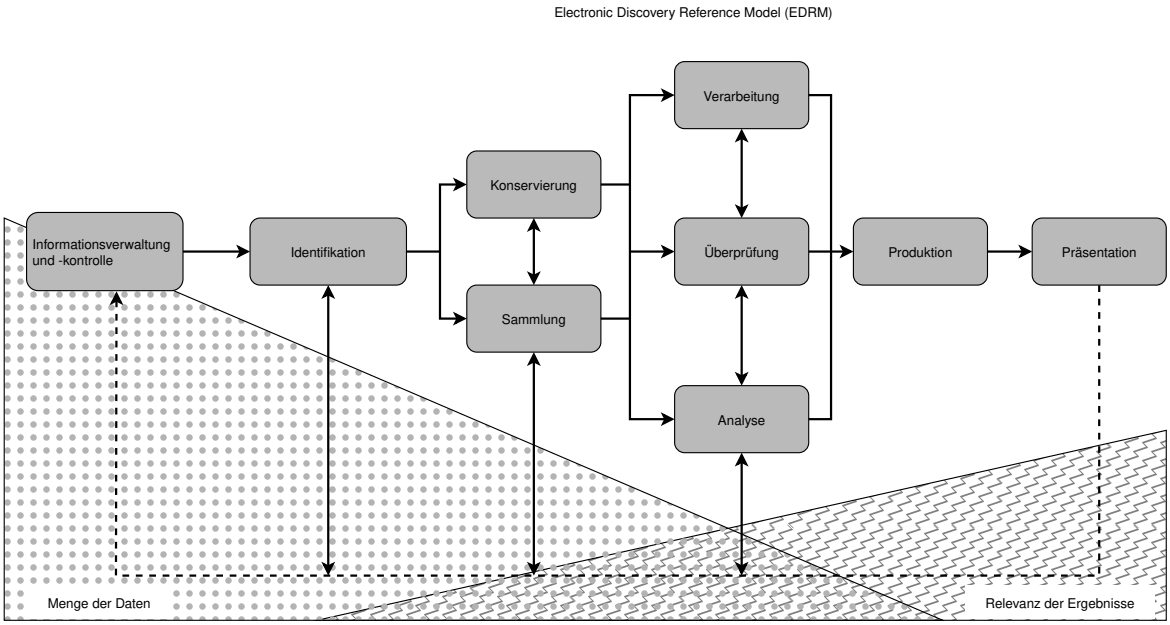

Abbildung 2.5: EDRM-Modell nach [1, S. 24, 22]

Informationen in großen Datenmengen. Ausgehend von dieser Beschreibung lässt sich ableiten, dass das EDRM-Modell auch für forensische Auswertungen geeignet ist. Anders als die anderen Modelle ist es kein Modell mit einzelnen Schritten, die in einer festgelegten chronologischen Reihenfolge durchgeführt werden müssen. Es besteht aus Bausteinen, die einzeln oder im Zusammenhang ausgeführt werden. Dies wird in Abbildung 2.5 dargestellt. Die Bausteine des Modells sind die Informationsverwaltung und Kontrolle, die Identifizierung, die Konservierung, die Datensammlung, die Datenverarbeitung, die Datensichtung, die Datenanalyse, die Bereitstellung und die Dokumentation. [1, S. 23-25]

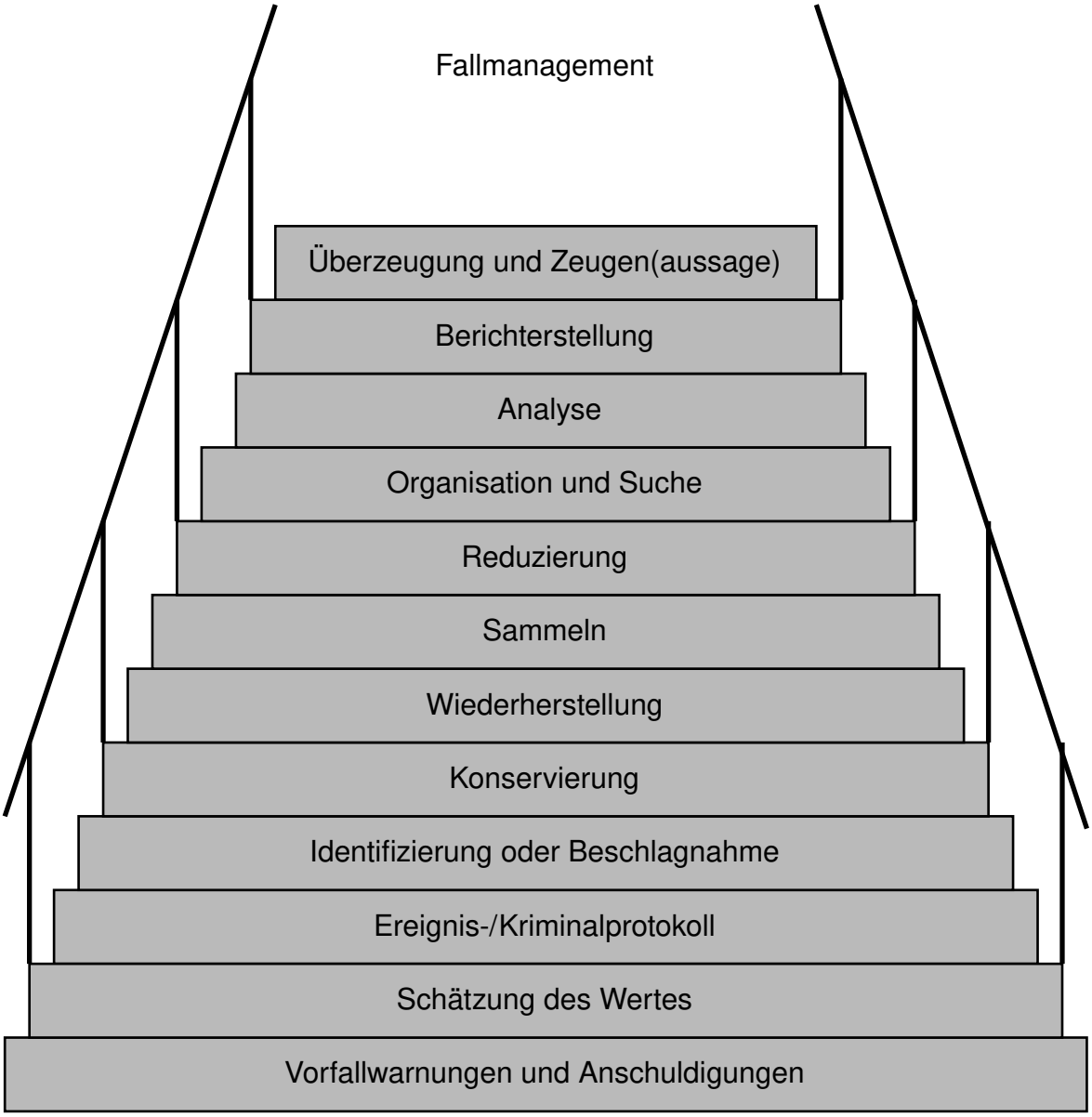

Abbildung 2.6: Casey-Modell nach [1, S. 26]

Das letzte Modell, auf das kurz eingegangen werden soll, ist das Investigation-Process-Modell nach Casey. Dieses Zwölf-Phasen-Modell stellt den Ablauf einer forensischen Untersuchung als Stufenprozess dar. Das Modell verbindet die wesentlichen Schritte der forensischen Untersuchung mit den Methoden und Techniken der Polizeiwissenschaft und der Kriminalistik. Der detaillierte Aufbau des Modells kann der Abbildung 2.6 entnommen werden. [1, S. 26f.]

2.3.2 Praxisnahes Vorgehen bei der Datenanalyse in IoT-Systemen

Für die Datenanalyse in IoT-Umgebungen gibt es kein eigenes standardisiertes Vorgehensmodell. Daher soll in diesem Abschnitt eine von Servida und Casey entwickelte Schrittfolge für das praktische Vorgehen beim Analyseprozess dargestellt werden. Dieses beschreiben sie in dem Paper „IoT forensic challenges and opportunities for digital traces“ von 2019. Sie stellen darin die besondere Bedeutung der Vorbereitung heraus. Die Daten von IoT-Geräten sind schwer zugänglich und flüchtig. Des Weiteren sind Netzwerk- sowie Speicherquellen stark verteilt und viele Daten gehen beim Neustart solcher Geräte verloren.

Der erste Schritt der forensischen Analyse nach Servida und Casey ist die Vorabrecherche. Diese zielt darauf ab, Informationen zum Gerät, zu verwendeten Apps, bekannten Schwachstellen und evtl. Möglichkeiten zum Root-Zugriff auf das Gerät zu erlangen. Dazu erfolgt eine Literatur- und Internetrecherche. [13, S. S23]

Danach folgt die Einrichtung der Testumgebung. Hierbei wird eine kontrollierte Umgebung geschaffen. Das Netzwerk muss so konfiguriert werden, dass es möglich ist, alle vom IoT-Gerät gesendeten und empfangenen Daten passiv zu sammeln. Außerdem müssen die Netzwerkkonfigurationen so angepasst werden, dass MITM-Angriffe unterstützt werden, um mögliche Angriffe auf das Gerät zu testen. [13, S. S23]

Anschließend erfolgen die unterschiedlichen Analysen. Zunächst muss hierbei eine Netzwerkanalyse durchgeführt werden. Sie umfasst die Analyse aller ein- und ausgehenden Daten. Dazu gehört auch, welche Protokolle verwendet werden, ob die Informationen lesbar übermittelt werden oder ob die Daten verschlüsselt sind. Des Weiteren wird untersucht, ob eine Anfälligkeit für MITM-Angriffe besteht. Zudem wird eine Untersuchung in Bezug auf offene Einstiegspunkte, etwa Ports und aktive Dienste, durchgeführt. [13, S. S23]

Nach der Analyse des Netzwerkes folgt die Appanalyse. Die forensische Untersuchung der mobilen App muss manuell erfolgen, weil die meisten Forensikprogramme keine Möglichkeit zur spezifischen Appanalyse bieten. [13, S. S23]

Darauf folgt die Schwachstellenanalyse. In diesem Schritt wird das IoT-Gerät gezielt auf bekannte und potenzielle Schwachstellen überprüft. Dazu zählen unter anderem unverschlüsselte Log-Dateien, offene Ports, die Nutzung unsicherer Dienste sowie Standard-Passwörter. Ziel ist es, Einfallstore zu erkennen, über die ein Angreifer Zugriff auf das Gerät oder seine Daten erlangen könnte. [13, S. S23f.]

Danach erfolgt die physische Analyse. In Fällen, in denen weder über das Netzwerk noch über Apps ausreichender Zugriff auf das Gerät möglich ist, wird versucht, direkt auf den Gerätespeicher zuzugreifen. Dies kann über serielle Schnittstellen wie UART oder JTAG erfolgen. Falls notwendig, kann auch auf sogenannte Chip-Off-Techniken zurückgegriffen werden, um den internen Speicher auszulesen und so wertvolle forensische Daten zu erhalten. [13, S. S24]

Ein weiterer Schritt im Analyseprozess kann die Auswertung von in der Cloud gespeicherten Daten sein. Viele IoT-Geräte synchronisieren regelmäßig Daten mit einem Cloud-Dienst, etwa um Nutzern über Apps den Fernzugriff zu ermöglichen oder Backups zu speichern. Diese Informationen können über den Anbieter selbst oder mit Hilfe von Cloud-Identifikatoren aus dem Smartphone des Nutzers erlangt werden. In der Regel ist hierfür jedoch eine rechtliche Genehmigung erforderlich. Die Analyse dieser Cloud-Daten kann wichtige ergänzende Informationen liefern, etwa zu Nutzerverhalten, Gerätezuständen oder zeitlichen Abläufen. [13, S. S23f.]

3 Methodiken

In diesem Kapitel wird das Vorgehen bei der Identifikation, Extraktion und Analyse der digitalen Spuren, die während der Nutzung der Bosch Eyes Innenkamera II und der Bosch Smart Camera App anfallen, dargestellt. Die Untersuchung wird auf Grundlage der von Servida und Casey vorgeschlagenen Schrittfolge durchgeführt. Die einzelnen Analysen werden nach dem pSAP-Modell vorgenommen. Die Schrittfolge von Servida und Casey wird jedoch aufgrund des in dieser Arbeit gesetzten Schwerpunkts nur auf die Vorabrecherche, den Aufbau der Testumgebung, die Netzwerkanalyse und die Appanalyse beschränkt. Das heißt, im Rahmen dieser Arbeit wird keine Cloudanalyse und keine physische Analyse durchgeführt. Aufgrund der ausgewählten Tools erfolgt eine Sicherheitsanalyse im Rahmen der Appanalyse.

3.1 Aufbau der Testumgebung

Um die Bosch Smart Camera App testen zu können, muss eine kontrollierte Umgebung geschaffen werden. Dafür wird ein separates Netzwerk eingerichtet, in das eine Reihe von Smart Home Geräten eingebunden ist. Das Netzwerk wird so konfiguriert, dass alle Geräte immer dieselbe IP-Adresse erhalten.

Als Netzwerkschnittstelle nach außen dient ein *OPNsense*-Gerät. Es ist über eine Switch mit der Fritzbox verbunden. Die Fritzbox dient nur als Access Point (AP). Das heißt, sie übernimmt keine Routing- und Firewall-Funktionen, sondern verteilt lediglich das WLAN und gegebenenfalls die LAN-Ports. Die Fritzbox leitet also keine Daten zwischen verschiedenen Netzwerken weiter und nimmt auch keine Filterung des Datenverkehrs vor. [23, S. 9] Das *OPNsense*-Gerät sorgt als zentrales Element für die Sicherheit des Datenverkehrs. *OPNsense* ermöglicht eine einheitliche und effiziente Überwachung und Einrichtung der verschiedenen Sicherheitskomponenten. Es fungiert als Router und übernimmt die Firewall-Funktionen. Dazu gehören unter anderem das Erstellen von Firewall-Regeln und die Erkennung und gegebenenfalls die Abwehr von Angriffen auf das Netzwerk.

Mit dem so bereitgestellten WLAN-Netzwerk ist ein gerootetes Smartphone Google Pixel 7a verbunden. Dieses verwendet das Betriebssystem Android 14. Auf diesem Gerät ist die Bosch Smart Camera App Version 2.8.0 installiert. Außerdem befindet sich eine nach Anleitung eingerichtete Bosch Eyes Innenkamera II im Netzwerk, die mit der App verbunden ist. Zudem ist ein Dell-Laptop mit dem Operating System (OS) Kali Linux Teil des Netzes. Auch steht zur Analyse ein ASUS-Laptop mit einem Betriebssystem Windows 11 Pro zur Verfügung. Er wird ausschließlich für die dynamische Analyse ins Netzwerk eingebunden. Auf diesem Laptop ist die Software, die zur Sicherung und Analyse der App-Daten benötigt wird, installiert. Außerdem wurde die Software zur Analyse der aufgezeichneten Pakete auf diesem Laptop installiert.

3.2 Netzwerkanalyse

In diesem Abschnitt soll nun die strukturierte Vorgehensweise zur Analyse des Netzwerkverkehrs der Bosch Smart Camera App dargestellt werden. Wie in Kapitel 2.2.2 beschrieben, umfasst die Netzwerkanalyse einen Portscan, das passive Mitschneiden der Datenpakete, die Analyse der aufgezeichneten Pakete sowie einen MITM-Angriff, um verschlüsselte Datenpakete lesbar zu machen. Für diese Aufgaben gibt es eine Vielzahl unterschiedlicher Tools. Im Rahmen dieser Arbeit werden *Nmap*, *OPNsense*, *Wireshark* und *Burp Suite* eingesetzt. Diese sollen im Folgenden vorgestellt werden.

3.2.1 Nmap als Werkzeug in der forensischen Untersuchung von IoT-Systemen

Für die forensische Untersuchung von IoT-Umgebungen ist es, wie bereits beschrieben, von elementarer Bedeutung, eine systematische Analyse der Netzwerkkommunikation durchzuführen. Dafür muss zuerst das verwendete Netzwerk systematisch untersucht werden, um die Geräte innerhalb des Netzes zu identifizieren, offene Kommunikationskanäle zu finden und potenzielle Angriffsvektoren zu erkennen. Es existiert eine Vielzahl von Netzwerk-Scannern. Der vermutlich bekannteste und am universellsten einsetzbare ist *Nmap*. Aufgrund dieser universellen Einsetzbarkeit und der Anerkanntheit ist die Nutzung von *Nmap* in der Version 7.94 sinnvoll. Dieses Werkzeug soll nun im folgenden näher beschrieben werden. [12, S. 116]

Das Open-Source-Tool ist ein sowohl in der Forensik als auch in der Sicherheitsanalyse anerkanntes Werkzeug. Durch verschiedene Scanverfahren ermöglicht es eine umfassende Netzwerkanalyse. Die wichtigsten Scanverfahren für die Netzwerkanalyse sind der sogenannte Stealth-Scan, der Dienst-Scan und die Betriebssystemerkennung. [12, S. 116-119]

Der Stealth-Scan wird mit der Option *-sS* ausgelöst und wird auch als Transmission Control Protocol (TCP)-synchronize (SYN)-Scan bezeichnet. Dieser dient zur diskreten Erkennung offener TCP-Ports, ohne dass er dafür eine vollständige TCP-Verbindung herstellt. [12, S. 116-119]

Der Dienst-Scan wird mit der Option *-sV* aufgerufen. Mit diesem Scan versucht *Nmap*, offene Ports zu finden sowie zu erkennen, welche Dienste an den Ports aktiviert sind und welche Version der Dienste genutzt wird. [12, S. 116-119]

Die Betriebssystemerkennung wird mit der Option *-O* gestartet. Dabei versucht *Nmap* durch einen TCP/IP-Fingerprintvergleich das Betriebssystem herauszufinden. Dies kann in der IoT-Forensik helfen, um zum Beispiel Embedded-Linux-Systeme oder proprietäre Betriebssysteme zu erkennen, die von IoT-Geräten genutzt werden. So können diese Geräte im Netz identifiziert werden. [12, S. 116-119, 13]

Im Rahmen der Analyse der Bosch Eyes Innenkamera II und Bosch Smart Camera App kann *Nmap* für folgende Punkte genutzt werden:

- Lokalisierung der Kamera im lokalen Netzwerk
- Erkennung verwendeter Protokolle (denkbar wären zum Beispiel: HTTPS oder MQTT) [14, S. 50, 343–345]
- Sichtbarmachung potenzieller Schwachstellen

Hierfür müssen ein vollständiger Portscan (*-p-*) und ein Dienst-Scan erfolgen, um unerwartete oder veraltete Netzwerkdienste zu erkennen. Außerdem können so Hinweise auf Fehlkonfigurationen oder unsichere Firmwareversionen gefunden werden. [12, S. 116-119, 15, S. 207]

Zusätzlich zu den hier beschriebenen Scanverfahren verfügt *Nmap* über eine sogenannte Nmap Scripting Engine (NSE). Mit dieser lassen sich spezifische Dienste tiefgehender analysieren. Diese Funktionen sind besonders nützlich, wenn Geräte ohne offizielle Dokumentation untersucht werden sollen. Die NSE erweitert die Basisfunktionalität von *Nmap* durch in Lua geschriebene Skripte, die automatisierte Aufgaben wie das Erkennen von Schwachstellen, das Ausnutzen von Fehlkonfigurationen oder die detaillierte Informationsbeschaffung über Netzwerkdienste ermöglichen. [24]

Abschließend lässt sich also sagen, dass *Nmap* ein wichtiges Werkzeug für forensische Untersuchungen ist.

Es gibt jedoch eine Reihe von Alternativen zu *Nmap*. Diese bringen andere Vorteile mit sich. So sind diese Tools zum Beispiel auf verschiedene Netzwerkprotokolle oder -verfahren optimiert und arbeiten besonders schnell, unauffällig oder ähnliches. Hier sollen nur einige Bespiele für solche optimierten Programme genannt werden, ohne einen Anspruch auf Vollständigkeit zu erheben. So bietet *fping* eine schnellere Form des *ping*, was Vorteile beim Testen der Erreichbarkeit vieler IP-Adressen mit sich bringt. *MASSCAN* ist hingegen ein besonders schneller Netzwerkscanner, der aber nur ausgewählte Ports testet. [12, S.116-119, 14, S. 225, 236ff.]

3.2.2 OPNsense zur Sicherung der Netzwerkpakete

Bei versendeten Netzwerkpaketen handelt es sich um flüchtige Daten. Daher ist es von elementarer Bedeutung, diese unmittelbar bei der Übermittlung aufzuzeichnen, um sie später analysieren zu können. Hierfür gibt es verschiedene Werkzeuge wie zum Beispiel *tcpdump* oder *Wireshark*. Im Rahmen dieser Arbeit wird jedoch *OPNsense* genutzt. Dieses verwendet *tcpdump* im Hintergrund. Dadurch ist eine Aufzeichnung der Datenpakete möglich, ohne ein zusätzliches Programm auf dem Dell-Analyselaptop installieren zu müssen. Die aufgezeichneten Pakete können in packet capture (pcap)-Dateien gespeichert und mit *Wireshark* analysiert werden. [25]

3.2.3 Wireshark zur Analyse des Netzwerkverkehrs in IoT-Systemen

Auf Grundlage des mit *Nmap* gewonnenen Überblicks über das Netzwerk, die offenen Ports und die genutzten Dienste der Bosch Eyes Innenkamera II kann eine genauere Analyse der Kommunikation zwischen Kamera, Cloud und App erfolgen. Auch hierfür gibt es ei-

ne Reihe von unterschiedlichen Werkzeugen. Neben *Wireshark* existieren zum Beispiel die Kommandozeilen-Tools *tcpdump* und *ngrep*. Diese nutzen beide intern die pcap-Bibiliothek, um Netzwerkpakete auszulesen und zu filtern. [12, S. 141-143]

Im Rahmen dieser Arbeit wird sich jedoch für *Wireshark 4.4.6* entschieden. Dies ist ein weit verbreitetes Open-Source-Analyseprogramm für Netzwerkpakete. *Wireshark* ermöglicht es, den Netzwerkverkehr in Echtzeit zu erfassen und detailliert zu analysieren. Zudem können auch Pakete analysiert werden, die von anderen Anwendungen oder *Wireshark* selbst im Vorfeld aufgezeichnet und gespeichert wurden. Für die Untersuchung bietet *Wireshark* zum einen eine Kommandozeilenoberfläche (Command Line Interface (CLI)) und zum anderen eine grafische Benutzeroberfläche (Graphical User Interface (GUI)). Mit *Wireshark* ist es möglich, die aufgezeichneten Pakete schon während des Mitschnitts flexibel zu filtern. Diese Filter bezeichnet man als Capture-Filter. Die erstellten Mitschnitte können auch mit sogenannten Display-Filtern weiter eingegrenzt werden, ohne den eigentlichen Mitschnitt zu verändern. [12, S.135-140, 26, S. 1f., 9f.]

In der IoT-Forensik kann mit Hilfe von *Wireshark* der Netzwerkverkehr zwischen IoT-Geräten, der App und der Cloud aufgezeichnet werden. Dabei erkennt *Wireshark* auch, zu welchem Kommunikationsprotokoll das aufgezeichnete Paket gehört. Dies kann Aufschluss darüber geben, zu welchem Zweck das Paket übertragen wurde und welche Art von Daten es enthalten könnte. Des Weiteren ist es durch die Filter- und Suchfunktionen von *Wireshark* möglich, ungewöhnliche Kommunikationsmuster, unerwartete Verbindungen zu Servern oder verdächtige Datenübertragungen zu erkennen. Außerdem kann durch die zeitliche Abfolge der erfassten Pakete eine Rekonstruktion der Interaktionen zwischen Kamera, App und anderen Netzwerkkomponenten erfolgen. Weiterhin bietet *Wireshark* die Möglichkeit, verschlüsselte Kommunikation zu entschlüsseln. Hierfür muss jedoch der private Schlüssel oder der Transport Layer Security (TLS)-Sitzungsschlüssel bekannt sein. Sollte dies der Fall sein, wird ein tieferer Einblick in die übertragenen Daten möglich. [26]

Wireshark ist somit ein unverzichtbares Werkzeug, um die Netzwerkebene der Kommunikation in IoT-Systemen zu verstehen und forensisch zu untersuchen. Die gewonnenen Erkenntnisse können entscheidend für die Rekonstruktion von Ereignissen und die Identifizierung potenzieller Sicherheitsvorfälle sein.

3.2.4 MITM-Angriff mit Burp Suite

Ein zentrales Problem bei der Datenübertragung von IoT-Geräten ist, dass diese zunehmend Protokolle nutzen, bei denen die Daten verschlüsselt übertragen werden. Aber auch diese Daten können forensisch wichtige Informationen enthalten. Um an diese zu gelangen, muss also eine Möglichkeit gefunden werden, trotzdem Zugriff auf die unverschlüsselten Daten zu erhalten. Hierfür werden spezialisierte Werkzeuge eingesetzt, die eine Analyse und Manipulation des Netzwerkverkehrs ermöglichen. Eines der etabliertesten Werkzeuge in diesem Bereich ist *Burp Suite*. Dieses ursprünglich für die Sicherheitsanalyse von Webanwendungen entwickelte Tool wird heute auch in der IoT-Forensik eingesetzt. [13, S. S23, 12, S. 199-205]

Burp Suite verfügt über eine Vielzahl von Funktionen. Dazu gehören Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer und Extensions. Im Reiter Dashboard können unter anderem verschiedene Scans eingerichtet werden. Dies ermöglicht eine automatische Analyse von Webapplikationen. Im Reiter Target befindet sich eine Auflistung der besuchten Uniform Resource Locator (URL)s. Target kann außerdem genutzt werden, um einen sogenannten Scope auf die Adresse einer Webseite zu legen. Dies sorgt dafür, dass diese Seite als Ziel der Analyse spezifiziert wird. Es können auch Webseiten ausgeschlossen werden. Mit Proxy können Request-Anfragen abgefangen und modifiziert werden. Er bietet auch eine Übersicht über die Historie aller verarbeiteten Request-Anfragen. Der Intruder ermöglicht es, Angriffe zu automatisieren. Dafür stehen verschiedene Konfigurationen zur Verfügung. Der Repeater dient dazu, einzelne Anfragen erneut zu senden. Diese können im Original oder in modifizierter Form versendet werden. Der Collaborator ermöglicht es, zu testen, ob eine Anwendung dazu bewegt werden kann, mit einem externen Dienst zu interagieren. Der Sequencer erlaubt es, die Qualität der Zufälligkeit aller Token zu testen. Dafür führt er mehrere Zufallstests gegen Stichproben durch. Der Decoder wird genutzt, um Zeichenketten in unterschiedlichen Formaten zu kodieren oder zu dekodieren. Mit dem Comparer ist es möglich, zwei Eingaben zu vergleichen. Der Logger zeichnet den gesamten HTTP-Verkehr in Echtzeit auf. Mit dem Organizer kann man Kopien von HTTP-Nachrichten speichern und kommentieren. Im Extensions-Tab ist es möglich, *Burp Suite* einfach zu erweitern und so an die Bedürfnisse des Nutzers anzupassen.

Dies zeigt nur einen Teil der Möglichkeiten auf, die *Burp Suite* dem Nutzer zur Analyse von Webapplikationen bietet. Die Community Edition ist eine kostenlose Version von *Burp Suite* mit eingeschränkter Funktionalität. Für diese Arbeit sind die Funktionen der kostenlosen Version 2025.2.4 jedoch ausreichend. Sie bietet die Funktion eines HTTP(S)-/WebSocket-Proxy, die Historie und die grundlegenden Werkzeuge. Zu diesen gehören Repeater, Decoder, Sequencer und Comparer. Des Weiteren steht eine Demoversion des Intruders zur Verfügung. [12, S. 199-205, 27]

Für diese Arbeit ist die Proxy-Funktion die wichtigste. Sie erlaubt es, den Datenverkehr zwischen einer Anwendung und einem Server abzufangen, zu analysieren und gegebenenfalls zu manipulieren. *Burp Suite* stellt dabei einen MITM-Proxy dar. Die Proxyfunktion ermöglicht es *Burp Suite*, eigene Zertifikate zu erstellen. Diese können installiert werden, um so auch verschlüsselte Inhalte sichtbar zu machen. Im IoT-Kontext bedeutet dies, die Zielapp muss auf einem Clientgerät betrieben werden, das dem *Burp-Suite*-Proxy vertraut. Das erfordert eine systemweite Installation des *Burp-Suite*-Zertifikats auf dem Android-Gerät. Hierfür benötigt man Rootzugriff auf das Gerät. [12, S.199-205, 15, S. 210-222]

Es gibt auch eine Reihe von anderen Programmen, mit deren Hilfe auf die unverschlüsselten Inhalte des Datenverkehrs zugegriffen werden kann. An dieser Stelle sollen nur *Fiddler* und *HTTP Toolkit* genannt werden. Da es sich bei *Burp Suite* jedoch um ein in der Sicherheitsanalyse von Webanwendungen und der IoT-Forensik anerkanntes Werkzeug handelt, wird dieses im Rahmen dieser Arbeit genutzt.

3.3 Appanalyse

In diesem Kapitel soll das strukturierte Vorgehen bei der Appanalyse dargestellt werden. Hierfür müssen einige Vorbereitungen getroffen werden. So wird die Bosch Smart Camera App in der Version 2.8.0 auf einem gerooteten Smartphone mit der Androidversion 14 und einem gerooteten AVD mit Androidversion 11 installiert. Eine detalliertere Darstellung der Vorbereitungen wird im Kapitel 6 beschrieben.

Von diesen Geräten werden die App und die dazugehörigen Daten logisch gesichert. Bei einer logischen Sicherung wird das Gerät über Universal Serial Bus (USB)-Kabel oder andere standardisierte Schnittstellen mit forensischer Hardware bzw. einer forensischen Workstation verbunden. Alternativ zu kabelgebundenen Verbindungen können auch Verbindungen über WLAN, Infrarot oder Bluetooth erfolgen. Nachdem eine Verbindung besteht, wird ein Befehl an das Gerät gesendet. Dieser wird vom Prozessor des Geräts interpretiert. Der Prozessor ruft dann die abgerufenen Daten aus dem Speicher und sendet sie an die forensische Hardware bzw. eine forensische Workstation. Da nur die Daten mit Appbezug zur Analyse extrahiert werden sollen ist diese Methode der Sicherung ausreichend. Das heißt für die Analyse ist keine physische Extraktion aller Daten des Geräts notwendig. Eine solche physische Sicherung bezeichnet man auch als *Hex Dump*. Dieser stellt eine exakte Kopie des Geräts in Binärform dar. Gleichzeitig wäre eine manuelle Extraktion der Daten nicht ausreichend, da dabei nicht alle Daten mit Appbezug extrahiert werden können. Für eine logische Extraktion der Daten gibt es eine ganze Reihe von Werkzeugen. Das wohl bekannteste ist die Android Debug Bridge (ADB). Es gibt aber auch noch andere Tools wie zum Beispiel *MyPhoneExplorer*. [16, S.21-24, 222–224]

Nach der Sicherung wird die App getrennt von den restlichen Daten mit Anwendungsbezug analysiert. Bei der Untersuchung von Anwendungen gibt es zwei generelle Herangehensweisen. Die dynamische und die statische Analyse. Je nach Literatur werden diese beiden großen Analysearten noch in jeweils zwei kleinere Methoden aufgeteilt. Diese werden dann als einfache und fortgeschrittene dynamische bzw. statische Analyse bezeichnet. In dieser Arbeit wird sich jedoch auf die Verwendung der Begriffe dynamische Analyse und statische Analyse beschränkt. [28, S. 22f.]

Unter statischer Analyse versteht man die Untersuchung des Quellcodes der Anwendung. Dafür ist es nicht notwendig, die Anwendung auszuführen. Ziel dieser Untersuchung ist es, herauszufinden, wie sich das Programm verhält, um so Schwachstellen oder andere forensisch relevante Informationen zu erhalten. Für diese statische Analyse muss die Android Package Kit (APK)-Datei zuerst dekompiliert werden. Dies kann entweder mit entsprechenden Tools erfolgen oder man ändert die Dateiendung von .apk auf .zip und extrahiert die Daten. Danach ist es möglich mit Tools wie *dex2jar* die Dex-Dateien in Java-Code umzuwandeln. Dieser kann dann untersucht werden. So können zum Beispiel Informationen zu Schwachstellen, zur Art und Weise der Datenspeicherung und zu vorhandenen Berechtigungen erlangt werden. [28, S. 22f. 15, S.188-209]

Unter dynamischer Analyse versteht man die Untersuchung der Anwendung während der Laufzeit. Hierfür muss eine kontrollierte Umgebung geschaffen werden, um Schäden am eigenen System oder Netzwerk zu verhindern. Ziel der dynamischen Analyse ist es, heraus-

zufinden, wie sich eine Anwendung wirklich bei der Ausführung verhält. Das heißt, es wird analysiert, welche Interaktionen mit Betriebssystem, Netzwerk und Daten stattfinden. So kann ungewöhnliches Verhalten erkannt und Schwachstellen aufgedeckt werden. In der Praxis wird dafür häufig eine virtuelle Maschine, ein Emulator oder eine sogenannte Sandbox verwendet, um die App isoliert zu betreiben. Innerhalb dieser Umgebung lassen sich dann alle Aktivitäten der App genau beobachten und aufzeichnen. Dazu zählen unter anderem Dateioperationen, Netzwerkverbindungen, Zugriffe auf den Gerätespeicher sowie Systemaufrufe. Besonders wichtig ist dabei die Überwachung des Netzwerkverkehrs. Es wird überprüft, welche Daten die App sendet oder empfängt, zum Beispiel ob sensible Informationen wie Passwörter, Standortdaten oder persönliche Nachrichten übertragen werden. Auch das Verhalten gegenüber anderen Apps oder Systemdiensten wird mit einbezogen. Ein weiterer Bestandteil der Analyse ist die Beobachtung der Ressourcennutzung. Hierbei werden Kennzahlen wie Central Processing Unit (CPU)-Auslastung, Speicherverbrauch und Batterienutzung erfasst, um die Effizienz und das Ressourcenverhalten der App zu bewerten. Zusätzlich kann eine sogenannte Datenflussanalyse durchgeführt werden. Diese untersucht, wie Daten innerhalb der App verarbeitet und weitergegeben werden. Auf diese Weise kann sichergestellt werden, dass sensible Informationen nicht unerlaubt verwendet oder weitergeleitet werden. Durch die Kombination all dieser Beobachtungen bietet die dynamische Analyse einen tiefen Einblick in das tatsächliche Verhalten einer App und ist somit ein wertvolles Werkzeug zur Sicherheitsbewertung und forensischen Untersuchung mobiler Anwendungen. [28, S. 22f., 61ff., 75–80]

Neben der statischen und dynamischen Analyse stellt die Untersuchung von App-Dateien eine weitere effektive Methode dar, um relevante Informationen über mobile Anwendungen zu gewinnen. Dabei werden insbesondere Dateien betrachtet, die von der App während der Laufzeit erzeugt oder benötigt werden. Der erste Schritt besteht darin, die relevanten Dateien auf dem Gerät zu lokalisieren. Anschließend werden diese extrahiert und mit geeigneten Tools analysiert. Hierfür kommen spezialisierte Werkzeuge zum Einsatz. Die Auswahl der passenden Tools hängt stark vom Dateityp und dem verwendeten Betriebssystem ab. Die in dieser Arbeit verwendeten Tools und Alternativen werden in Kapitel 3.3.2 beschrieben.

In vielen Fällen ist der Zugriff auf geschützte Verzeichnisse notwendig, beispielsweise bei Android unter */data/data/<App-Paketname>* oder bei iOS unter */var/mobile/Containers/Data/*. Um diese Verzeichnisse einsehen zu können, ist es häufig erforderlich, das Gerät zu rooten oder einen Jailbreak durchzuführen. Mit dem Rooten von Android-Geräten erlangt man Root-Rechte auf dem Gerät. Das heißt, man hat administrativen Zugriff auf das System. Beim Jailbreaking handelt es sich um das Äquivalent zum Rooten von Android-Geräten unter iOS. [16, S. 31, 42, 49f., 215–221, 224–227, 251–277] Zu den für die Analyse besonders interessanten Dateitypen zählen:

- Datenbanken, die Informationen über Benutzerkonten, App-Verläufe oder Einstellungen enthalten
- Caches und temporäre Dateien, in denen Nutzungsdaten gespeichert sein können
- Konfigurationsdateien wie Extensible Markup Language (XML)- oder JavaScript Object Notation (JSON)-Dateien, die App-Einstellungen oder Parameter enthalten
- Logdateien, die Hinweise auf Benutzeraktivitäten, Fehler und Kommunikationsprozesse liefern können [16, S. 31, 42, 49f., 215–221, 224–227, 251–277]

Durch diese Dateianalyse lassen sich viele zusätzliche Informationen gewinnen, die durch eine reine dynamische Beobachtung möglicherweise verborgen bleiben. Die Methode ist daher ein wertvoller Bestandteil bei der forensischen Untersuchung mobiler Anwendungen und bietet wichtige Erkenntnisse über die interne Funktionsweise der App.

3.3.1 ADB für den direkten Gerätezugriff und die Artefaktsicherung

Die ADB ist ein wichtiges Werkzeug für Entwickler und Systemanalysten im Umgang mit Android-Geräten. Sie dient als Schnittstelle zwischen einem Host-Rechner und einem Android-System, wobei die Verbindung über USB oder das TCP/IP-Protokoll aufgebaut werden kann. ADB basiert auf einer klassischen Client-Server-Architektur: Der Client nimmt Benutzerbefehle entgegen, der Server fungiert als Vermittler und koordiniert die Kommunikation, während der Android Debug Bridge Daemon (adbd) direkt auf dem Gerät läuft und dort die Anweisungen ausführt. [29, S. 125-136]

Die ADB-Kommunikation erfolgt über den standardisierten TCP-Port 5037, über den der Server Verbindungen von Clients entgegennimmt und Anfragen an den Daemon weiterleitet. Dieses Zusammenspiel ermöglicht eine Vielzahl von Funktionen, von der Fehlersuche, dem Debugging, über den Zugriff auf Dateisysteme bis hin zu kompletten System-Backups. Zu den häufig genutzten Befehlen zählen etwa *adb devices* zur Auflistung aller erkannten Geräte oder *adb shell*, mit dem direkt auf die Kommandozeile des Android-Systems zugegriffen werden kann. [29, S. 125-136]

Ein zentrales Merkmal von ADB ist das strukturiert aufgebaute Kommunikationsprotokoll, das sowohl einfache als auch komplexe Befehlsabläufe abbilden kann. Je nach Befehl werden Daten entweder direkt vom Server verarbeitet oder über den Daemon mit dem Gerät ausgetauscht. Die Befehle werden dabei in genau definierten Nachrichtenformaten mit Header-Informationen und Nutzdaten, den sogenannten Payloads, übermittelt. [29, S. 125-136]

Für die Nutzung von ADB ist es notwendig, das Android Software Development Kit (SDK) auf dem Host-System zu installieren und auf dem Android-Gerät das USB-Debugging zu aktivieren. Bei bestimmten Betriebssystemen wie Linux müssen zusätzlich entsprechende userspace /dev (udev)-Regeln eingerichtet werden, um den Zugriff auf die USB-Schnittstelle zu ermöglichen. Ist die Verbindung einmal hergestellt, bietet ADB sowohl für Entwickler in der Entwicklungsphase als auch für Analysten bei der systematischen Auswertung von App-Daten eine schnelle und flexible Möglichkeit, mit dem Android-System zu arbeiten. [29, S. 125-136]

Einige ADB-Befehle, wie zum Beispiel *adb pull*, benötigen Root-Rechte. Es kann aber dazu kommen, dass trotz vorhandener Root-Rechte die Fehlermeldung *Permission denied* zurückgegeben wird. Ursächlich für diese Fehlermeldung kann einer der Sicherheitsmechanismen von Android sein. Bei diesem handelt es sich um Security-Enhanced Linux (SELinux). Der mit Android 4.3 eingeführte Sicherheitsmechanismus ist seit Android 5.0 durchgesetzt. Seitdem nutzt Android statt einem klassischen Discretionary Access Control (DAC) das Prinzip des Mandatory Access Control (MAC)s. Bei der Nutzung von DAC fragt die Anwendung den Nutzer nach Berechtigungen. Der Benutzer verweigert oder gewährt diese. Im Gegensatz dazu ist es bei MAC so, dass, auch wenn ein Nutzer über Root-Rechte verfügt, die durch SELinux

definierten Sicherheitsrichtlinien weiter gelten. Damit wird jede Aktion, die nicht ausdrücklich erlaubt ist, blockiert. SELinux verhindert so, dass kompromittierte Prozesse mit Root-Rechten auf sicherheitskritische Systembereiche zugreifen können oder Apps manipulieren können. Dieser Modus wird auch als Enforcing Mode bezeichnet und ist meist standardmäßig aktiviert. Er blockiert aktiv alle nicht erlaubten Prozesse und protokolliert dies. Es ist möglich, SELinux in einen anderen Modus, den Permissive Mode, umzuschalten. In diesem Modus werden die nicht durch die Sicherheitsrichtlinien zugelassenen Aktionen nur protokolliert und nicht blockiert. Das erlaubt dem Nutzer, auch solche Aktivitäten auszuführen. [29, S. 125-136, 16, S. 170, 224–227]

So kann dann mit ADB eine Vielzahl von Daten, die von Android-Anwendungen genutzt und gespeichert werden, gesammelt und extrahiert werden. Apps speichern Anwendungsdateien, Benutzerdaten, temporäre Dateien und Multimedia- bzw. Benutzerinhalte. [16, S. 224-227] Dazu gehören Anwendungsdaten wie APKs. Diese Dateien enthalten die ausführbare Anwendung, die aus Code, Ressourcen und Manifestinformationen besteht. Eine App kann auch aus mehreren APKs aufgebaut sein. Dies wird dann auch als Split-APK bezeichnet. Benutzerdaten können unter anderem in SQLite-Datenbanken gespeichert sein. Diese enthalten strukturierte Informationen in Form von Datenbanken. Ebenso können XML-Dateien Nutzereinstellungen und Sitzungsdaten beinhalten. Auch Logdateien und Webview-Daten enthalten Benutzerdaten. Zu Multimedia- und Benutzerinhalten gehören Bilder, Videos, Tonaufnahmen oder verschiedene Dokumente, die von der App heruntergeladen oder erzeugt werden. Temporäre Dateien sind beispielsweise Cache-Dateien und Session-Daten. Dies soll nur einen Überblick über die Vielzahl der von Apps gespeicherten Daten liefern, ohne einen Anspruch auf Vollständigkeit zu haben. [16, S.224-227, 288–303]

Die Daten, die in Apps anfallen, werden an unterschiedlichen Orten gespeichert. Die beiden primären Speicherorte sind */data/app/PackageName* und */data/data/PackageName*. Das Verzeichnis */data/app/PackageName* enthält die eigentliche Anwendung. */data/data/PackageName* enthält die App-spezifischen Daten wie SQLite-Datenbanken, SharedPreferences und temporäre Daten. Es können sich auch Daten mit App-Bezug an externen Speicherorten wie Secure-Digital (SD)-Karten befinden. Auch diese müssen bei einer forensischen Untersuchung berücksichtigt werden. [16, S. 224-227, 241f., 289]

3.3.2 Tools zur Untersuchung der gesicherten Daten

Wie bereits beschrieben, können bei der Analyse der gesicherten Daten eine Vielzahl unterschiedlicher Dateiformate auftreten. Um diese öffnen und analysieren zu können, werden unterschiedliche Programme und Tools benötigt. In diesem Abschnitt sollen die in dieser Arbeit verwendeten Werkzeuge näher beschrieben werden.

Häufig werden Datenbanken zur Speicherung von Daten genutzt. Zur Auswertung solcher Datenbanken stehen eine Reihe verschiedener Tools zur Verfügung. Dazu gehören unter anderem *SQLiteStudio*, *DB Browser for SQLite* und *SQLiteSpy*. In der vorliegenden Arbeit wird der *DB Browser for SQLite* in der Version 3.13.1 eingesetzt. Dabei handelt es sich um ein weit verbreitetes, intuitiv bedienbares Open-Source-Tool. Es ist plattformübergreifend nutzbar und bietet eine übersichtliche, benutzerfreundliche grafische Oberfläche. [16, S. 104f. 30] Die

meisten zu analysierenden Dateien können im Texteditor geöffnet und analysiert werden. Alternativ zum in Windows standardmäßig enthaltenen *Editor* können auch andere Tools wie beispielsweise *Notepad++* genutzt werden. Dies bringt jedoch keine entscheidenden Vorteile. Aus diesem Grund wird in dieser Arbeit darauf verzichtet, ein zusätzliches Programm für diese Aufgabe zu installieren. Für die Analyse binär codierter Dateien können Hex-Editoren verwendet werden. Die hier verwendete Software *HxD* in der Version 2.5.0.0 ist nur ein Beispiel für Hex-Editoren. Andere Hex-Editoren sind unter anderem *wxHexEditor* und *WinHex*. [17, S. 378] *HxD* ist ein leistungsfähiges, kostenloses Werkzeug zur Bearbeitung von Rohdaten auf Datei-, Speicher- und Datenträgerebene. Es erlaubt Dateien beliebiger Größe in hexadezimaler und American Standard Code for Information Interchange (ASCII)-Darstellung einzusehen und zu bearbeiten. In der Forensik kann *HxD* genutzt werden, um Binärdateien auf mögliche Klartextinformationen oder relevante Datenmuster zu untersuchen. Das Programm bietet eine übersichtliche Benutzeroberfläche und unterstützt verschiedene Zeichensätze und Funktionen. Zu den Funktionen, die eine Analyse erleichtern, gehören eine Suchfunktion, Lesezeichen und eine Änderungsverfolgung. Das Programm bietet außerdem die Möglichkeit, Prüfsummen zu berechnen und bearbeitete Daten in verschiedenen Formaten zu speichern. [31]

Für die Untersuchung der Protocol Buffer (Protobuf)-Dateien, die mit *Perfetto* erzeugt wurden, wird das Werkzeug *Perfetto UI* genutzt. Dieses kann über https://ui.perfetto.dev/ erreicht werden und ermöglicht es, die aufgezeichneten Traces zu analysieren. Wenn das Tool einmal im Browser geöffnet ist, arbeitet es lokal und es wird keine Verbindung zum Server benötigt. Neben *Perfetto UI* bietet *Perfetto* ein CLI-Tool zur Analyse der Dateien an. [32]

Abschließend soll nun kurz auf die Umrechnung der Unix-Zeitstempel eingegangen werden. Hierfür gibt es eine Reihe unterschiedlicher Online-Konverter. Außerdem bieten viele Programmiersprachen Module, Klassen oder Funktionen zur Umrechnung von Unix-Zeitstempeln. In dieser Arbeit wird ein Online-Konverter genutzt. Dieser ist unter https://www.unixtimestamp.com/de/ erreichbar.

3.3.3 Schwachstellenanalyse und Appanalyse mit MobSF

Die Schwachstellen- und Appanalyse der Bosch Smart Camera App kann, wie bereits beschrieben, manuell durchgeführt werden. Dies benötigt jedoch ein fortgeschrittenes Fachwissen und ist sehr zeitintensiv [12, S. 961f.]. Aus diesem Grund wird im Rahmen dieser Arbeit ein Tool eingesetzt, das diesen Prozess automatisiert und einen Überblick liefert. Bei diesem Tool handelt es sich um *MobSF*. Als Alternativen wären auch *Appknox* und *Quokka* möglich.

MobSF ist ein leistungsstarkes und vielseitiges Open-Source-Tool für Sicherheitsanalysen von mobilen Anwendungen. Das Tool unterstützt die Analyse von mobilen Applikationen auf verschiedenen Plattformen wie Android, iOS und Windows Mobile. Es ist in der Lage, verschiedene Dateiformate wie zum Beispiel APK, iOS App Package (IPA) und APPX (Paketdateiformat für die Verteilung und Installation einer Anwendung) zu analysieren. Der große Vorteil von *MobSF* im Vergleich zu einer Reihe ähnlicher Software liegt darin, dass es nicht nur statische oder dynamische Analyse ermöglicht, sondern beide Varianten unterstützt. Damit bietet es die Möglichkeit, die Sicherheit einer App ganzheitlich zu bewerten. [33, 34, 35]

Bei der statischen Analyse wird der Quellcode auf Schwachstellen untersucht und die Struktur der App analysiert, ohne diese auszuführen. Dadurch lassen sich kritische Sicherheitslücken wie unsichere Berechtigungen, unsichere Kryptografiealgorithmen sowie unzureichende Sicherheitsprotokolle erkennen. Zu Beginn der Analyse dekompiliert *MobSF* die zu untersuchende APK, um den Inhalt und den Aufbau der Anwendung analysieren zu können. [33, 34]

Bei der Permission-Analyse bewertet *MobSF* kritische Berechtigung hinsichtlich möglicher Risiken. Als kritische Berechtigungen einer App werden hierbei zum Beispiel Zugriff auf die Kamera, das Mikrofon, den Standort und Kontakte angesehen. Kritisch sind beispielsweise unautorisierte Aufzeichnungen und die Sammlung von Nutzerdaten. *MobSF* kategorisiert die Risiken, um potenzielle Gefahren bewerten zu können. [33, 34]

Eine weitere zentrale Funktion der statischen Analyse ist die Kryptografieanalyse. Bei dieser sollen Schwächen in der Implementierung der kryptografischen Verfahren aufgedeckt werden. Dabei werden unsichere kryptografische Algorithmen erkannt. Dazu gehören zum Beispiel Message-Digest Algorithm 5 (MD5) oder Secure Hash Algorithm (SHA)-1. Des Weiteren sucht das Tool nach Secure Sockets Layer (SSL)/TLS-Schwachstellen. Dazu gehören unter anderem unverschlüsselte HTTP-Links oder Zertifikatsprüfungsumgehungen. [33, 34]

Bei der statischen Analyse führt das Tool außerdem eine Codeanalyse und Reverse Engineering durch. Hierbei wird der Bytecode der Anwendung in einen Java-Code umgewandelt. So ist es dem Anwender möglich, potenziell gefährliche Funktionen zu erkennen. Dazu zählen das Abgreifen von Anrufdaten, Browserhistorie oder Short Message Service (SMS)-Nachrichten. [33, 34]

Im Rahmen der statischen Analyse wird zudem ein Domain Malware Check durchgeführt. Dabei werden die in der App festgestellten Domains mit den Einträgen in der Malware-Datenbank abgeglichen. Damit wird ermöglicht, dass schädliche Netzwerkanfragen frühzeitig erkannt und Bedrohungen abgewehrt werden können. [33, 34]

Alle gefundenen Schwachstellen bewertet *MobSF* anhand des Common Vulnerability Scoring System (CVSS). Bei diesem handelt es sich um eine etablierte Sicherheitsmetrik, die eine numerische Bewertung der Schwachstellen liefert. Damit ist eine eindeutige Priorisierung der Probleme möglich. [33]

Neben der statischen Analyse bietet *MobSF* die Möglichkeit einer dynamischen Analyse. Bei der dynamischen Analyse wird das Verhalten der Anwendung während der Laufzeit analysiert. Hierfür muss die App in einer isolierten Umgebung, wie zum Beispiel in einem Emulator oder in einer Sandbox, ausgeführt werden. Auf diese Weise wird das Host-System vor potenziellen Schäden geschützt. *MobSF* unterstützt dafür verschiedene gerootete Android Virtuelle Maschine (VM)s und gejailbreakte iOS VMs. Empfohlen wird für die Analyse von Androidanwendungen der Android Emulator *Genymotion*. Beim Aufsetzen der *Genymotion* Android VM schlägt die Dokumentation von *MobSF* vor, eine Android-Version 7.0 oder höher zu verwenden. Durch das Ausführen der App in einer isolierten Umgebung ist es möglich,

Schwachstellen zu erkennen, die erst im laufenden Betrieb deutlich werden. Dazu gehören unter anderem ungesicherte Datenübertragungen, schädliche Application Programming Interface (API)-Aufrufe und bösartige Aktivitäten. [33, 34, 36, 37]

Um eine dynamische Analyse einer App durchzuführen, wird die zu analysierende Anwendung, wie bereits erwähnt, in einer sicheren Umgebung gestartet. *MobSF* überwacht während der Ausführung mehrere sicherheitsrelevante Parameter. [33, 34]

Dazu gehört unter anderem eine Netzwerkverkehrsanalyse, bei der der gesamte ein- und ausgehende Datenverkehr aufgezeichnet wird. Damit ist es *MobSF* möglich, ungesicherte Verbindungen wie zum Beispiel HTTP-Anfragen zu erkennen. Außerdem kann so geprüft werden, ob sensible Daten unautorisiert übertragen werden. Zusätzlich wird der aufgezeichnete Datenverkehr dahingehend untersucht, ob bekannte Maleware-Server angefragt werden. [33, 34]

Neben der Netzwerkverkehrsanalyse werden auch Dateioperationen analysiert. Hierfür wird überwacht, welche Dateien die App liest, erstellt oder verändert. Dadurch können potenziell gefährliche Aktivitäten wie beispielsweise die unautorisierte Sammlung von Nutzerdaten oder Dateisystemveränderungen erkannt werden. [33, 34]

Weiterhin zeichnet *MobSF* die genutzten Berechtigungen und die API-Aufrufe auf, um auffällige Zugriffe auf sensible Komponenten wie Kamera, Mikrofon oder auch Standorterkennung zu identifizieren. Des Weiteren wird bei der dynamischen Analyse überprüft, ob die Anwendung ein bekanntes Malware-Verhalten aufweist. Dazu gehören zum Beispiel das Nachladen schädlicher Inhalte oder die Erstellung von sogenannten Backdoors (Hintertüren). [33, 34]

MobSF überprüft außerdem, wie mit verschlüsselten Daten umgegangen wird. So ist die Erkennung ungesicherter Datenübertragungen möglich. Auch werden die System-Logs analysiert, um sicherheitsrelevante Ereignisse wie Abstürze, Fehler und andere auffällige Ereignisse zu erkennen und zu bewerten. [33, 34]

MobSF erlaubt zudem, eine erweiterte Sicherheitsprüfung durchzuführen. Bei dieser werden gezielt Angriffe simuliert. So kann unter anderem eine SSL/TLS-Bypass-Prüfung durchgeführt werden. Dadurch können Sicherheitslücken in der Implementierung der Verschlüsselung erkannt werden. [33, 34]

MobSF verknüpft die Vorteile der statischen und dynamischen Analyse und verfolgt somit einen hybriden Ansatz. Da auf diese Weise die Vorteile beider Methoden ausgenutzt werden können, ist mit *MobSF* eine umfassende Bewertung der Sicherheit mobiler Anwendungen möglich. Durch die statische Analyse erhält man eine detaillierte Übersicht über die Anwendungsstruktur, anhand derer man frühzeitig Schwachstellen erkennen kann. Die dynamische Analyse liefert Erkenntnisse über das tatsächliche Verhalten der Anwendung während der Laufzeit. So lassen sich auch verschleierte Codeabschnitte oder Sicherheitsrisiken, die erst während der Ausführung der App erkennbar sind, identifizieren. [33]

Ein weiterer Vorteil von *MobSF* ist die einfache Bedienbarkeit und das übersichtliche Design des Tools. Dadurch ist es sowohl für Einsteiger als auch für Experten geeignet. Das Tool verfügt über eine grafische Benutzeroberfläche (GUI). Über diese können Nutzer einfach Anwendungen hochladen und erhalten einen Bericht über deren Sicherheit. [33]

4 Vorabrecherche

In diesem Kapitel werden die Ergebnisse der Vorabrecherche vorgestellt. Hierfür wird eine Internetrecherche durchgeführt. Der Schwerpunkt dieser Recherche sind die unterschiedlichen Publikationen des Herstellers. Diese werden analysiert, um Informationen über die Bosch Eyes Innenkamera II und die Bosch Smart Camera App zu erhalten. Zusätzlich werden auch aktuell bekannte kritische Sicherheitslücken ausfindig gemacht.

4.1 Die Bosch Eyes Innenkamera II und die Bosch Smart Camera App

Die Bosch Eyes Innenkamera II ist eine IP-fähige Kamera für den Innenbereich, die mit Hilfe der Bosch Smart Camera App gesteuert und konfiguriert wird. Sie ist laut Hersteller ausschließlich für die Überwachung privater Wohnungen und ähnlicher Räume gedacht und soll einen Beitrag leisten, das Sicherheitsgefühl des Nutzers zu erhöhen. [38, 39, 40]

Die Kamera bietet mehrere Funktionen. Sie erfasst Bewegungen und Geräusche. In der App kann definiert werden, ab welcher Intensität diese als relevant bewertet werden. Dabei kann durch Videoanalyse erkannt werden, ob es sich um die Bewegung eines Menschen handelt. Durch eine integrierte Infrarot-Leuchte ist es möglich, das Gerät auch nachts zu benutzen. In der App können Bewegungszonen definiert werden. Die Kamera erkennt dann nur Ereignisse in den ausgewählten Bereichen als relevant. Durch das Einfahren der Kamera kann die Überwachung des Raumes unterbunden werden. Das Einfahren des Kamerakopfes erfolgt durch einen Berührungssensor am Gerät oder auch durch die App. [38, 39, 40, 41]

Bei relevanten Ereignissen werden 15-sekündige Clips erstellt. Bis zu 100 solcher Aufzeichnungen können in der Cloud sieben Tage lang kostenlos gespeichert werden. Wenn mehr als 100 Clips gespeichert werden sollen, kann ein zusätzliches Abonnement (Abo) abgeschlossen werden. Ansonsten werden automatisch die ältesten Aufzeichnungen überschrieben. Durch das Cloud+Abo ist es dem Nutzer möglich, 30 Tage lang bis zu 400 Clips mit einer Länge von bis zu 60 Sekunden in der Cloud zu speichern. In der App können bis zu 25 Clips als Favoriten gekennzeichnet und so dauerhaft gespeichert werden. Bei relevanten Ereignissen wird zusätzlich zur Videoaufzeichnung eine Push-Benachrichtigung an die Geräte gesendet, die durch die App mit der Kamera verbunden sind. Zudem bietet die App die Möglichkeit, ein Sirenensignal auszulösen. [38, 40, 41, 42]

Neben der Speicherung von Clips bei relevanten Ereignissen ist es auch möglich, die Kamera zur Live-Überwachung des Wohnraums zu nutzen. Der Live-Video-Stream kann in der Bosch Smart Camera App aufgerufen werden. In der App wählt man diesen in der Übersicht über alle verbundenen Kameras aus. Die App bietet auch die Möglichkeit zur Live-Kommunikation über die Kamera. Zudem ist es möglich, Aufnahmezeitpläne zu erstellen und die aufgenommenen Clips zu verwalten. Des Weiteren wird sie auch zur Updateverwaltung genutzt. [38, 40]

Zusätzlich können smarte Kameras mit der Bosch Smart Home App verbunden werden. Dadurch werden Automatisierungen und weitere intelligente Szenarien zusammen mit anderen Smart Home Geräten möglich. [40]

Des Weiteren lässt sich keine Dokumentation der API der Kamera finden, die die technischen Schnittstellen und verfügbaren Steuerungsfunktionen im Detail beschreibt. Stattdessen steht lediglich die offizielle OpenAPI-Dokumentation für die Nutzung über den Bosch Smart Home Controller zur Verfügung, während eine direkte oder eigenständige Ansteuerung der Kamera ohne den Controller weder dokumentiert noch vorgesehen ist. [43]

Die Bosch-Gruppe nutzt zur Anmeldung bei ihren Diensten die zentrale Identitätsplattform *SingleKey ID*. Über diese ist die Anmeldung bei verschiedenen Anwendungen und Diensten mit nur einem Benutzernamen und einem Passwort möglich. Auch die Anmeldung bei der Smart Home App und der Smart Camera App erfolgt über SingleKey ID. Damit ist die Nutzung der Apps nur bei bestehender Internetverbindung möglich. [44]

Zusätzlich zur Recherche zur allgemeinen Funktionsweise der Kamera und der App werden auch Informationen über die Datenverarbeitung, Datenerfassung und Sicherheitsmaßnahmen gesammelt. Dabei fällt auf, dass die Bosch Eyes Innenkamera II nicht über einen verbauten Speicher oder die Möglichkeit, eine Speicherkarte zu integrieren, verfügt. Alle Video- und Tonaufnahmen werden in der Cloud gespeichert. Die Übermittlung und Speicherung erfolgt verschlüsselt. [40]

Auch Nutzerdaten werden nicht auf der Kamera selbst, sondern in der App oder auf den Systemen von Bosch gespeichert. Nutzerdaten sind personenbezogene Daten. Zur Nutzung der Bosch Smart Camera App ist die Angabe solcher Daten notwendig. [45] Folgende benutzerbezogene Daten werden laut Datenschutzhinweisen der Bosch Smart Camera App erfasst:

- „Merkmale zur Identifikation als Nutzer (Benutzerkennung, Kennung, IP-Adresse)
- Informationen zu Beginn und Ende sowie des Umfangs der Nutzung der Bosch Smart Camera App
- Gerätekennungen
- Standortinformationen
- Kontaktdaten (z. B. zur Eventbenachrichtigung)
- Einstellungen der App bzgl. unserer angebotenen Services
- Technische Informationen zum Abgleich und zur Bereitstellung von aktueller Uhrzeit und Updates Ihres Systems durch unsere Server
- Systemdaten z.B. angeschlossene Kameras und Zubehör, Seriennummer, Softwareversionen der einzelnen Komponenten
- Umstände der Bilderzeugung (z. B. Lichtverhältnisse, Datum, Uhrzeit)
- Zustandsdaten des Systems z. B. Systemzeit, Fehlermeldungen
- Ereignishistorie des Video Systems
- Typ des verwendeten Endgerätes z.B. Smartphone bzw. Tablet-PC, Hersteller, OS-Version des Endgeräts

- Nutzungsdaten der Applikation z.B. Nutzungshäufigkeit, registrierte Abstürze, Fehler der Applikation “ [45, S. 3]

4.2 Potenzielle Schwachstellen der Bosch Smart Camera App und der Bosch Eyes Innenkamera II

Bei der Internetrecherche zu potenziellen Schwachstellen der Bosch Smart Camera App und der Bosch Eyes Innenkamera II fällt auf, dass zum Zeitpunkt der Recherche keine aktuellen Meldungen zu kritischen Sicherheitslücken existieren (Stand: 11.04.2025). Frühere Vorfälle können aber auch Risikobereiche aufzeigen. So betrifft die Schwachstelle Common Vulnerabilities and Exposures (CVE)-2019-7729 alle App-Versionen für Android der Bosch Smart Camera App vor der Version 1.3.1. Dort gibt es ein Problem mit unsicheren Berechtigungen. Dies ermöglicht potenziell den unbefugten Zugriff auf zwischengespeicherte Videoclips. Wichtig ist hierbei zu betonen, dass diese Meldung nur ältere App-Versionen betrifft. [46]

Die Schwachstelle CVE-2020-6781 betrifft die Bosch Smart Home System App für iOS. So war es potenziell möglich, mit einem MITM-Angriff unbefugt Zugriff auf Videoinhalte zu erlangen. Wichtig für die Einordnung ist jedoch, dass es sich bei der Bosch Smart Home System App lediglich um eine App vom selben Entwickler handelt. [47]

Auch Bosch-Anwendungen sind nicht immun gegen Sicherheitslücken. Bosch veröffentlicht in der Regel Sicherheitsupdates zur Behebung gefundener Schwachstellen. Die aufgezählten Sicherheitslücken sind daher in den aktuellen Versionen geschlossen. Für die Benutzer jedoch bleibt die regelmäßige Installation der Software- und Firmware-Updates wichtig, um gegen neu erkannte Risiken abgesichert zu sein. Außerdem sollte unbedingt auf die Verwendung eines sicheren Passwortes und die Beachtung der allgemeinen Sicherheitspraktiken bei der Benutzung vernetzter Geräte geachtet werden.

5 Untersuchung und Analyse des Netzwerkverkehrs

In diesem Kapitel soll beschrieben werden, wie bei der Netzwerkverkehrsanalyse konkret vorgegangen wird. Teile dieser Analyse sind, wie bereits im Kapitel 3.2 „Methodiken“ beschrieben, ein Port- und Netzwerkscan mit *Nmap*, die Untersuchung von Paketmitschnitten mit *Wireshark* und ein MITM-Angriff, der mit dem Tool *Burp Suite* durchgeführt werden kann. Auch bei der Netzwerkverkehrsanalyse müssen eine Reihe von Vorbereitungen getroffen werden. Dazu gehört der Aufbau eines gesichertes Testnetzwerks. Eine Beschreibung dieses Netzwerk enthält das Kapitel 3.1. Des Weiteren erfolgt eine Planung der Analyseszenarien, anhand welcher das Verhalten der Bosch Smart Camera App und der Bosch Eyes Innenkamera II analysiert wird.

5.1 Netzwerk- und Portscan

Der Netzwerk- und Portscan dient dazu, einen Überblick über das Netzwerk und die darin eingebundenen Geräte zu erhalten. Um mit dem Tool *Nmap* einen Netzwerkscan durchführen zu können, ist es notwendig, die Subnetzmaske zu bestimmen. Dies erfolgt mit Hilfe des im Testnetzwerk integrierten Kali-Linux-Analyselaptops.

Nach Eingabe des Befehls *ip a* im Terminal erhält man eine Anzeige aller Netzwerkschnittstellen und deren IP-Adressen sowie deren Classless Inter-Domain Routing (CIDR)-Präfixe. Diese sind jeweils von der IP-Adresse durch einen Schrägstrich getrennt angegeben. Hierbei ist vor allem die WLAN-Schnittstelle *wlan0* für die Analyse relevant. Bei dieser findet sich, wie bei den anderen Schnittstellen auch, eine Zeile, die mit *inet* beginnt. *inet* steht dabei für Internet Protocol Version 4 (IPv4)-Adressen. Darauf folgt die IP-Adresse des Laptops kombiniert mit dem CIDR-Präfix. Im konkreten Fall ist dem Laptop die IP-Adresse 10.0.1.121 zugewiesen. Der Präfix lautet */24*. Er gibt an, wie viele Bits der IP-Adresse für den Netzwerkteil reserviert sind. In diesem Fall sind es 24 Bit. Nach der Bestimmung des Subnetzes kann mit dem in Quelltext 5.1 dargestellten Befehl ein Ping-Scan über das gesamte Subnetz erfolgen. Hierdurch erhält man eine Liste aller aktiven im Netzwerk integrierten Geräte. Die Liste liefert eine Übersicht über deren IP-Adressen, Media Access Control-Adressen (MAC-Adressen) und Herstellerinformationen, soweit diese erkennbar sind. So erhält man einen guten Überblick über die eingeschalteten und erreichbaren Geräte im Netzwerk.

```
nmap -sn 10.0.1.121/24
```

Quelltext 5.1: Ping-Scan über das gesamte Subnetz

Der Ping-Scan über das gesamte Subnetz ist ein wichtiger Vorbereitungsschritt für den geplanten Portscan der Bosch Eyes Innenkamera II. Um diese im Netzwerk besser identifizieren zu können, wird der Befehl in Quelltext 5.2 ausgeführt.

```
nmap -sn 10.0.1.121/24 | grep -i "64:DA:A0:20:BB:07" -B 2
```

Quelltext 5.2: Bestimmung der IP-Adresse der Bosch Eyes Innenkamera II

Hierbei wird die Ausgabe des Ping- Befehls durch die Pipe (|) an den darauf folgenden Befehl direkt weitergeleitet. Dieser ist im konkreten Fall ein *grep*-Befehl. Der Befehl beinhaltet den Parameter *-i*. Dieser gibt an, dass Groß- und Kleinschreibung bei der Suche nicht berücksichtigt werden. Darauf folgt die Zeichenkette, nach der gesucht werden soll. In diesem Fall handelt es sich um die Media Access Control-Adresse (MAC-Adresse) der Kamera, die auf der Unterseite des Geräts zu finden ist. Außerdem wird die Ausgabe des *grep*-Befehls mit *-B 2* angepasst. Dabei steht *-B* dafür, dass Zeilen vor dem Treffer ausgegeben werden sollen. Die Ziffer *2* gibt an, wie viele Zeilen ausgegeben werden. Damit wird in der Ausgabe von *Nmap* die angegebene MAC-Adresse unabhängig von ihrer Groß- und Kleinschreibung gesucht. Durch die Auswahl der Parameter werden zusätzlich zur MAC-Adresse auch die anderen Angaben zur Kamera ausgegeben, die in der Ping-Scan-Liste enthalten sind. So findet man unter anderem die IP-Adresse der Kamera. Diese lautet 10.0.1.131 und wird für den Portscan benötigt.

Für den Portscan wird der in Quelltext 5.3 dargestellte Befehl ausgeführt. Mit dem Befehl *sudo* führt *Nmap* den nachfolgenden Scan mit Administratorrechten durch. Dies ist notwendig, weil einige der Parameter nur mit diesen Rechten funktionieren. Der Parameter *-p-* gibt an, dass der Scan alle Ports erfasst. Durch *-sV* und *-O* wird versucht, die Version der Dienste an offenen Ports und die Betriebssystemversion des Gerätes mit der IP-Adresse 10.0.1.131 herauszufinden. Bei diesem Gerät handelt es sich um die Kamera.

```
sudo nmap -p- -sV -O 10.0.1.131
```

Quelltext 5.3: Portscan Bosch Eyes Innenkamera II

Mit den oben beschriebenen Befehlen wird ein erster Überblick über die offenen Ports und die bereitgestellten Dienste gewonnen. Danach erfolgt eine detaillierte Analyse des einzigen verfügbaren Dienstes auf Port 443. Hierfür wird erneut ein strukturiertes Vorgehen genutzt. Zunächst kann versucht werden, die Webseite in einem Webbrowser zu öffnen. So kann sich potenziell ein Überblick über den angebotenen Dienst verschafft werden. Des Weiteren können mit dem Befehl in Quelltext 5.4 Informationen über den TLS-Handshake, Zertifikate und unterstützte Verschlüsselungsverfahren erlangt werden.

```
openssl s_client -connect 10.0.1.131:443
```

Quelltext 5.4: Verbindung zu Port 443 mittels OpenSSL

Um an diese Informationen zu gelangen, wird das *OpenSSL*-Tool 3.5.0 genutzt. Es baut eine Verbindung zur IP-Adresse 10.0.1.131 über Port 443 auf. Der Parameter *s_client* sorgt dafür, dass *OpenSSL* einen Client simuliert, der sich über *-connect* mit der entsprechenden IP- und Portkombination verbindet.

Des Weiteren sendet das Kommandozeilentool *curl* auf den Befehl in Quelltext 5.5 eine HTTP/HTTPS-Anfrage an den Server.

```
curl -k -v https://10.0.1.131
```

Quelltext 5.5: Analyse der HTTPS-Verbindung mit curl

Diese ermöglicht eine nähere Analyse des HTTPS-Headers und Bodys. Die Parameter *-k* und *-v* beeinflussen die Anfrage bzw. die Ausgabe des Befehls. Durch den Parameter *-k* ist es möglich, SSL/TLS-Zertifikatswarnungen zu ignorieren. Damit wird verhindert, dass Verbindungen zum Server abgelehnt werden, wenn dieser selbstsignierte Zertifikate oder Zertifikate von nicht vertrauenswürdigen Zertifizierungsstellen verwendet. Mit dem Parameter *-v* wird *curl* im „Verbose-Mode“ genutzt. Damit werden alle Details der Kommunikation ausgegeben. Hierzu gehören der Verbindungsversuch an sich, genaue Informationen zum SSL/TLS-Handshake sowie alle gesendeten und empfangenen HTTP-Header. Darüber hinaus gibt der Verbose-Modus auch Rückmeldungen zu Weiterleitungen, Fehlermeldungen oder Verbindungsabbrüchen aus.

Neben dieser ersten Analyse der Hauptseite kann mit den Tools *Fuzz Faster U Fool (ffuf)* oder *Directory Buster (dirb)* nach versteckten Weboberflächen gesucht werden. Es gibt noch eine Reihe anderer solcher Tools, die durch einen Wörterbuch-Test versteckte Dateien und Verzeichnisse suchen. Im Rahmen dieser Arbeit werden jedoch die Tools *dirb* und *ffuf* genutzt. Bei einem solchen Wörterbuch-Test werden systematisch mögliche URLs anhand einer Wortliste mit häufigen Pfaden ausprobiert, um versteckte Seiten, Administratorbereiche oder API-Endpunkte auf dem Webserver zu entdecken. Dazu wird zunächst mit dem Befehl im Quelltext 5.6 nach entsprechenden Dateien und Verzeichnissen gesucht.

```
dirb https://10.0.1.131
```

Quelltext 5.6: Suche versteckten Weboberflächen mit dirb

Dieser Befehl nutzt die Standardwortliste `/usr/share/dirb/wordlists/common.txt`. Des Weiteren wurde mit dem Befehl in Quelltext 5.7 ein erneuter Wörterbuch-Test mit der Wortliste */home/kali/Documents/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt* durchgeführt.

```
ffuf -w /home/kali/Documents/SecLists/Discovery/Web-Content/directory-list-2.3-
    medium.txt -u https://10.0.1.131:443/FUZZ
```

Quelltext 5.7: Suche versteckten Weboberflächen mit ffuf

Der Befehl ist folgendermaßen aufgebaut: ffuf gibt das Tool an. Mit dem Parameter *-w* wird die Wortliste übergeben. Bei dieser handelt es sich um *directory-list-2.3-medium.txt*. Mit *-u* wird dann die Ziel-URL angegeben. *FUZZ* gibt in der URL den Platzhalter für die Einträge der Wortliste an. Neben der Wortliste *directory-list-2.3-medium.txt* werden auch weitere Wortlisten genutzt. Eine Übersicht aller in dieser Arbeit genutzten Listen und der gewonnenen Ergebnisse findet sich in Kapitel 5.5.1

Zur weiteren Analyse des Webservers wird der *Nmap*-Befehl in Quelltext 5.8 genutzt.

```
nmap -sV -p 443 --script "http-*" 10.0.1.131
```

Quelltext 5.8: Analyse des HTTPS-Dienstes mit Nmap und HTTP-bezogenen NSE-Skripten

Dieser ermöglicht eine detaillierte Service- und Sicherheitserkennung. Hierbei wird die bereits beschriebene Erkennung der genauen Version des Dienstes auf den Port 443 beschränkt. Zusätzlich werden mit dem Parameter *script "http-*"* alle NSE-Skripte, die mit *„http-“* beginnen, ausgeführt.

5.2 Sicherung der Datenpakete

In diesem Abschnitt soll die Sicherung der Datenpakete beschrieben werden. Für die Sicherung dieser Datenpakete wird der *OPNsense*-Router genutzt. *OPNsense* bietet eine Weboberfläche, die ein Aufzeichnen der Netzwerkpakete als Teil der Diagnosefunktionen ermöglicht.

Die Analyse des Netzwerkverkehrs erfolgt getrennt für die Bosch Eyes Innenkamera II und das Google Pixel 7a. Dabei wird die Sicherung der Datenpakete in verschiedenen Nutzungsszenarien durchgeführt. Diese sollen im Folgenden näher beschrieben werden.

Im ersten Szenario wird untersucht, welche Daten im Rahmen der Anmeldung mit SingleKey ID übermittelt werden und welche Verbindungen dafür genutzt werden. Hierfür erfolgt in der App die Eingabe der E-Mail-Adresse und des Passwortes.

Beim zweiten Szenario erfolgt die Sicherung des Netzwerkverkehrs während der Fernsteuerung der Kamera über die Bosch Smart Camera App. Hierfür wird die Kamera über die App eingeschaltet.

Das dritte Szenario ist die Untersuchung der Daten und Verbindungen bei der Nutzung des Livestreams. Hierbei erfolgt die Sicherung der Pakete beim Start sowie während des Betriebs und beim Beenden des Livestreams über die Bosch Smart Camera App.

Im folgenden Szenario wird untersucht, ob es bei der Erstellung und Speicherung eines Screenshots über die App zum Aufbau von Verbindungen oder zu Datenübertragungen über das Netzwerk kommt.

Des Weiteren wird untersucht, ob bei der Umstellung der Kamerakonfigurationen in der App Datenübertragungen stattfinden. Hierfür wird die Bewegungserkennung über die App deaktiviert.

Im letzten Szenario wird eine Meldung der Kamera durch Bewegung erzeugt. Ziel ist es, zu untersuchen wie diese Meldung an das Smartphone versendet wird.

Bei der Sicherung der Datenpakete wird darauf verzichtet, andere Filter als den IP-Adressen-Filter zu setzen, um nicht zufällig relevanten Datenverkehr auszufiltern. Die Aufzeichnung der Datenpakete erfolgt dabei, wie bereits beschrieben, für Kamera und Smartphone getrennt. Um den Datenverkehr nach den IP-Adressen filtern zu können, müssen diese zuerst bestimmt werden. Die Bestimmung der IP-Adresse der Kamera ist bereits in Kapitel 5.1 beschrieben. Die IP-Adresse des Google Pixel 7a wird über die Einstellungen des Handys festgestellt. Zusätzlich muss bei der Erstellung der Sicherung darauf geachtet werden, dass beim Parameter *Count 0* eingestellt ist. Bei dieser Auswahl werden alle Pakete, unabhängig von der Anzahl und unabhängig davon, ob es sich um gesendete oder empfangene Pakete handelt, aufgezeichnet.

5.3 Analyse der gesicherten Datenpakete mit Wireshark

Die Analyse der aufgezeichneten Netzwerkdaten erfolgt mit Hilfe der Software *Wireshark*. Ziel ist es, den Netzwerkverkehr im Rahmen verschiedener Nutzungsszenarien der Bosch Smart Camera App zu untersuchen. Hierbei liegt der Fokus insbesondere auf der Identifikation von Kommunikationspartnern, den verwendeten Protokollen sowie dem Zweck einzelner Verbindungen, etwa zur Authentifizierung oder zur Übertragung von Medieninhalten.

Die Analyse der pcap-Dateien erfolgt nach dem folgenden strukturierten Vorgehen, dass sich in den unterschiedlichen Szenarien nicht wesentlich unterscheidet. Zunächst wird sich mit der Statistik-Funktion „Verbindungen“ eine Übersicht über aktive Verbindungen, beteiligte IP-Adressen, die übertragenen Datenmengen und die verwendeten Protokolle verschafft. Im Anschluss daran erfolgt eine gezielte Filterung nach bekannten IP-Adressen von Diensten, die im Rahmen der Anmeldung oder Nutzung der App kontaktiert wurden. Die entsprechenden IP-Adressen werden zuvor mittels DNS-Analyse (Filter: *dns.flags.response == 1*) aus den pcap-Dateien extrahiert und mit öffentlichen *Whois*- und DNS-Reverse-Diensten verifiziert.

Ergänzend wird die Funktion „Follow TCP Stream“ bzw. „Follow QUIC Stream“ verwendet, um die gesamte Kommunikation zwischen den Clients und dem jeweiligen Dienst im Zusammenhang nachvollziehen zu können. Zur besseren zeitlichen Einordnung der Ereignisse wird auch auf die „I/O Graphs“ zurückgegriffen, um die Intensität der Kommunikation über die Zeit visuell darzustellen.

Besonderes Augenmerk gilt dem verschlüsselten Datenverkehr über TLS und Quick UDP Internet Connections (QUIC). Obwohl die übertragenen Inhalte nicht einsehbar sind, liefern Metadaten, insbesondere Server Name Indication (SNI)-Einträge im „Client Hello“ sowie Muster im Verbindungsaufbau und das Antwortverhalten der Server wichtige Hinweise auf die zugrunde liegenden Abläufe der Datenübertragung und die Funktion der beteiligten Server.

Die Kombination aus DNS-basierter Vorauswahl, IP-Filterung, Protokollinspektion und Strömungsverfolgung ermöglicht eine systematische und nachvollziehbare Untersuchung der Netzwerkkommunikation bei der Benutzeranmeldung und der Nutzung der Bosch Smart Camera App.

5.4 MITM-Angriff

Zur Durchführung des MITM-Angriffs wird ein kontrolliertes Testumfeld aufgebaut, bestehend aus einem emulierten Android-Gerät und einem lokal installierten Proxy-Server. Als Testgerät dient ein über *Genymotion* emuliertes Google Pixel 5 mit Android 11, auf dem die Bosch Smart Camera App direkt aus dem Google Play Store installiert wird. Nähere Informationen zur Einrichtung eines emulierten Geräts finden sich in Kapitel 6.1. Für das Abfangen und Analysieren des Datenverkehrs kommt die *Burp Suite Community Edition* in der Version 2025.4.5 zum Einsatz. Der Proxy-Listener wird auf allen verfügbaren Netzwerkschnittstellen mit Port 8080 im unsichtbaren Modus („Invisible mode“) konfiguriert.

Um HTTPS-Traffic korrekt entschlüsseln zu können, wird das von *Burp Suite* generierte Certificate Authority (CA)-Zertifikat im *.cer*-Format per Drag-and-Drop auf das emulierte Gerät übertragen und dort als Benutzerzertifikat installiert. Die Proxy-Konfiguration wird anschließend über die ADB vorgenommen. Zunächst wird das emulierte Gerät per *adb connect* verbunden, dann der globale HTTP-Proxy auf *localhost:8080* gesetzt. Zusätzlich wird mit *adb reverse* eine Portweiterleitung konfiguriert, um sicherzustellen, dass Verbindungen korrekt über den lokalen Proxy geleitet werden. Die erfolgreiche Konfiguration wird durch Ausführen des Befehls in Quelltext 5.9 überprüft.

```
adb shell settings get global http_proxy
```

Quelltext 5.9: Überprüfen der MITM-Proxy Konfigurationen

Neben der systemweiten ADB-Konfiguration bietet Android die Möglichkeit, Proxy-Einstellungen direkt innerhalb der WLAN-Verbindung vorzunehmen. Dies ist über die erweiterten Netzwerkeinstellungen möglich. Dort kann unter dem Abschnitt „Proxy“ die Einstellung von „None“ auf „Manual“ geändert werden. Daraufhin lassen sich Hostname und Port manuell eintragen. Nach dem Speichern dieser Einstellungen wird der gesamte HTTP/HTTPS-Verkehr über die angegebene Proxy-Adresse geleitet, sofern die App keine eigene Proxy-Konfiguration ignoriert oder umgeht. Diese Methode eignet sich besonders für Testumgebungen ohne Root-Zugriff oder ADB-Interface. Sie bietet eine einfache Möglichkeit zur Einbindung eines Proxys, ohne auf ADB-Befehle zurückgreifen zu müssen. Zudem ist dieses Vorgehen insbesondere dann hilfreich, wenn manuelle Netzwerkverbindungen mit fest definierten Parametern getestet werden sollen oder wenn keine Konsole zur Verfügung steht.

Als Funktionstest der Proxy-Verbindung wird eine einfache Google-Suche im integrierten Browser durchgeführt. Dabei wurde der Begriff „hallo welt“ eingegeben. Erscheinen daraufhin in *Burp Suite* mehrere GET-Anfragen an www.google.com, bestätigt dies die erfolgreiche Einbindung des Proxy-Servers.

Im nächsten Schritt wird die Bosch Smart Camera App gestartet. Nach dem Initialisieren der App wird der Login-Button „Log in with SingleKey ID“ ausgewählt, um einen Authentifizierungsprozess zu starten. Parallel dazu wird in *Burp Suite* überprüft, ob sich der Datenverkehr abfangen lässt.

5.5 Ergebnisse der Analyse des Netzwerkverkehrs

Dieses Kapitel enthält die relevanten Ergebnisse der forensischen Analyse des Netzwerkverkehrs. Es werden die Resultate aus dem durchgeführten Port- und Netzwerkscan sowie die Analyse der aufgezeichneten Datenpakete und die Erkenntnisse aus dem durchgeführten MITM-Angriff vorgestellt. Im Fokus stehen hierbei die Untersuchung und Identifizierung von sicherheitskritischen Schwachstellen sowie die Gewinnung von potenziell forensisch relevanten Informationen.

5.5.1 Ergebnisse Portscan und Netzwerkscan

In diesem Abschnitt sollen nun die Ergebnisse des Netzwerk- und des Portscans dargestellt werden. Diese zeigen, dass eine Vielzahl unterschiedlicher Geräte im Subnetz vorhanden sind. Interessant für die Untersuchung ist jedoch ausschließlich die Kamera. Sie besitzt die IP-Adresse 10.0.1.131 und die MAC-Adresse 64:DA:A0:20:BB:07 . Die MAC-Adresse kann der Robert Bosch Smart Home GmbH zugeordnet werden. Des Weiteren antworten nur von drei Ports auf den Portscan. Dies sind die Ports 22, 2345 und 443. Über den Port 22 wird ein SSH-Dienst angeboten. Dieser ist jedoch geschlossen. Auch Port 2345 ist geschlossen. Der Dienst, der auf Port 2345 angeboten wird, konnte jedoch nicht erkannt werden. Dies ist ein Hinweis auf einen proprietären Dienst. Das heißt, der Dienst könnte vom Unternehmen bzw. Hersteller selbst entwickelt worden sein. Der einzige Port, der als offen erkannt wird, ist Port 443. Über diesen wird der HTTPS-Dienst angeboten. Die Betriebssystemerkennung gibt mehrere Linux-Kernel-Versionen zurück. Dies weist darauf hin, dass das Betriebssystem auf Linux basiert.

Die nähere Untersuchung des HTTPS-Dienstes zeigt, dass das Gerät Transport Layer Security Version 1.3 (TLSv1.3) nutzt. Es wird außerdem der Cipher Suite *TLS_AES_256_GCM_SHA384* für die TLS-Verbindung verwendet. Dieser Cipher Suite gibt an, wie genau Daten zwischen Client und Server verschlüsselt werden. TLS steht hierbei für das verwendete Protokoll. *AES_256* gibt den verwendeten Verschlüsselungsalgorithmus an. In diesem Fall ist dieser Advanced Encryption Standard (AES) mit einer Schlüssellänge von 256 Bit. Galios/Counter Mode (GCM) gibt den Modus der AES-Verschlüsselung an. SHA384 ist der verwendete Hashing-Algorithmus für die Datenintegritätsprüfung. Dies zeigt, dass moderne Verschlüsselungsverfahren genutzt werden. Das vom Server genutzte Zertifikat ist selbstsigniert. Dies ist typisch für eingebettete Geräte. Als Aussteller des Zertifikats ist die Robert Bosch Smart Home GmbH angegeben.

Der Webserver antwortet technisch korrekt auf HTTPS-Anfragen, gibt jedoch auf dem Wurzelpfad nur die Meldung „Object not available on this Webserver“ zurück. Daraus ergibt sich, dass der Server nicht über eine klassische Startseite oder Benutzeroberfläche verfügt. Dies kann sowohl mit einem Webbrowser als auch mit dem Kommandozeilentool *curl* beobachtet werden. Auch die Ergebnisse der Scan-Tools *dirb*, *ffuf* und *Nmap* bestätigen dies. Die weiterführenden Skriptscans mittels *Nmap* liefern ebenfalls keine Hinweise auf bekannte Verzeichnisse, Loginseiten oder serverseitige Anwendungen. In der nachfolgenden Tabelle 5.1 sind die bei der Untersuchung des Dienstes mit *dirb* und *ffuf* genutzten Wortlisten mit ih-

ren Verzeichnissen und den Ergebnissen der Tests dargestellt. Bis zur Abgabe dieser Arbeit war es nicht möglich, den Scan mit allen Listen vollständig durchlaufen zu lassen. Die nicht vollständig durchgelaufenden Listen sind in der Tabelle 5.1 mit einem * markiert.

Tabelle 5.1: Ergebnisse der Wörterbuch-Tests mit dirb und ffuf

Listenname	Verzeichnis	Ergebnis
big.txt *	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer
common.txt *	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer
common.txt	`/usr/share/dirb/wordlists`	Kein Treffer
common_directories.txt	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer
common-api-endpoints-mazen160.txt	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer
directory-list-2.3-big.txt *	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer
directory-list-2.3-medium.txt	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer
SOAP-functions.txt	`/home/kali/Documents/SecLists/Discovery/Web-Content/Web-Services`	Kein Treffer
spring-root.txt *	`/home/kali/Documents/SecLists/Discovery/Web-Content`	Kein Treffer

5.5.2 Ergebnisse Analyse der Datenpakete

In diesem Kapitel werden die Ergebnisse der durchgeführten Analyse des Netzwerkverkehrs der Bosch Smart Camera App und der Bosch Eyes Innenkamera II vorgestellt. Dabei wird auf die Ergebnisse der durchgeführten Szenarien einzeln eingegangen und dargestellt, welche Informationen aus diesen entnommen werden können.

Im ersten Szenario wird die Bosch Smart Camera App gestartet und sich mit Hilfe von Single-Key ID angemeldet. Bei der Analyse des dabei aufgezeichneten Datenverkehrs fällt auf, dass die Bosch Smart Camera App einen relativ komplexen Authentifizierungsprozess über den „SingleKey ID“-Dienst nutzt. Hierfür baut die App Verbindungen zu einer Vielzahl von Servern auf. In der Tabelle 5.2 sind die Server, zu denen im Rahmen des Anmeldungsprozesses Verbindungen eingegangen werden, mit Domainnamen und IP-Adressen dargestellt.

Tabelle 5.2: Zur Anmeldung mit SingleKey ID benötigte Server

Domainname	**IP-Adresse**
play-fe.googleapis.com	142.250.181.206
cryptauthdevicesync.googleapis.com	142.250.181.202, 172.217.16.74, 142.251.209.138
playgateway-pa.googleapis.com	142.250.181.202, 172.217.16.74, 142.251.209.138, 172.217.19.74
smarthome.authz.bosch.com	108.142.53.151
singlekey-id.com	20.33.66.163
cdn.singlekey-id.com	13.107.253.45
content-autofill.googleapis.com	142.251.209.138, 172.217.19.74, 142.250.181.202
js.hcaptcha.com	104.19.230.21, 104.19.229.21
newassets.hcaptcha.com	104.19.230.21, 104.19.229.21
api.hcaptcha.com	104.19.229.21, 104.19.230.21
google-ohttp-relay-safebrowsing.fastly-edge.com	146.75.121.91
proxy-19.live.cbs.boschsecurity.com	3.65.16.105
29.mcg.escrypt.com (alias für mcg.escrypt.com)	139.15.237.48
residential.cbs.boschsecurity.com	52.28.39.3

Die Analyse des Netzwerkverkehrs zeigt, dass verschiedenen Domains dieselbe IP-Adresse zugeordnet wird. Dies weist darauf hin, dass die Dienste auf einem gemeinsamen Server gehostet werden. Das bezeichnet man auch als Host-Sharing. So können zum Beispiel die Domains cryptauthdevicesync.googleapis.com, playgateway-pa.googleapis.com und content-autofill.googleapis.com alle die IP-Adresse 142.250.181.202 nutzen. Dies ist für Google-Dienste typisch, da Google viele seiner Dienste auf derselben Serverinfrastruktur anbietet. Neben dem Host-Sharing erschwert die konsequente Verschlüsselung des Datenverkehrs die Analyse erheblich.

Die Datenübertragung erfolgt konsequent über HTTP/2 und HTTP/3. *Wireshark* erkennt hierbei Transport Layer Security Version 1.2 (TLSv1.2), TLSv1.3 und QUIC als Transportprotokolle. Bei diesen Protokollen werden die Daten konsequent verschlüsselt übertragen. Daher muss sich bei der Analyse ausschließlich auf verwendete Protokolle, Zeitstempel, IP-Adressen, Servernamen und Kommunikationsmuster gestützt werden. Auf Grundlage der Ergebnisse dieser Analyse können die aufgebauten Verbindungen drei Kategorien zugeordnet werden.

Die erste Gruppe der Verbindungen dient in der Kommunikation zur Vorbereitung und Sicherheitsüberprüfung des Systems. Folgende Google- und Sicherheitsdienste können der Gruppe zugeordnet werden:

- cryptauthdevicesync.googleapis.com, playgateway-pa.googleapis.com und play-fe.googleapis.com. Diese nutzen QUIC- und TLS-Verbindungen und dienen zur Überprüfung der App- und Geräteintegrität sowie deren Aktualität.
- content-autofill.googleapis.com dient der Initialisierung der Formular-Autovervollständigung. content-autofill.googleapis.com nutzt QUIC- und TLSv1.3-Verbindungen.
- js.hcaptcha, newassets.hcaptcha.com und api.hcaptcha.com nutzen QUIC- sowie TLSv1.3-Verbindungen. Diese Dienste dienen der Integration einer Bot-Erkennung mit Human Completely Automated Public Turing test to tell Computers and Humans Apart (hCAPTCHA).
- google-ohttp-relay-safebrowsing.fastly-edge.com kommuniziert ebenfalls über QUIC- und TLSv1.3. Dieser Dienst dient zur Nutzung von Google Safe Browsing über Oblivios HTTP (OHTTP). OHTTP ermöglicht hierbei eine Kommunikation zwischen Server und Client, ohne dass diese vollständige Informationen über einander erhalten.

Der zweiten Gruppe werden die Verbindungen zugeordnet, die den eigentlichen Authentifizierungsprozess mit SingleKey ID bzw. das Laden der Login-Komponenten ermöglichen. Hierzu gehören die Verbindungen zu folgenden drei Domains:

- cdn.singlekey-id.com gehört wahrscheinlich zum Content Delivery Network (CDN) und wird über TLSv1.3 kontaktiert, um Ressourcen nachzuladen.
- Über singlekey-id.com erfolgt der eigentliche Login per HTTPS (TLSv1.3). Das Kommunikationsmuster lässt darauf schließen, dass die App einen Autorisierungscode erhält, der für den weiteren Anmeldeprozess benötigt wird.
- Bei smarthome.authz.bosch.com handelt es sich vermutlich um den boscheigenen Autorisierungsserver. Dort müsste der von singlekey-id.com erhaltene Autorisierungscode gegen Access Token und einen Identification (ID)-Token getauscht werden. Die Token stellen hierbei kryptografisch signierte Nachweise für die erfolgreiche Anmeldung dar.

Die Verbindungen, die in der dritten Gruppe zusammengefasst werden, treten erst nach der erfolgreichen Anmeldung mit SingleKey ID auf. Sie dienen der Kommunikation mit den Backend- und Geräte-Diensten. Diese Gruppe beinhaltet:

- proxy-19.live.cbs.boschsecurity.com ist der Server, der mit der Kamera kommuniziert.
- residential.cbs.boschsecurity.com ist der Server, mit dem die App kommuniziert. Via TLSv1.2 wird eine Verbindung aufgebaut, die nicht beendet wird.
- 29.mcg.escrypt.com ist eine Einbindung eines Public Key Infrastructure (PKI)-Dienstleisters und dient vermutlich der Geräteverifikation.

Im zweiten Szenario wird das Verhalten bei der Fernsteuerung der Kamera über die Bosch Smart Camera App untersucht. Dabei kommunizieren sowohl die Kamera (IP: 10.0.1.131) als auch die App auf dem Smartphone (IP: 10.0.1.112) mit Servern der Bosch-Cloud. Die Kommunikation erfolgt verschlüsselt über TLSv1.2 auf TCP-Port 443. Port 443 ist der klassische Port für HTTPS-Kommunikation.

In den Mitschnitten können keine Pakete festgestellt werden, die auf einen TLS-Handshake hinweisen. Stattdessen werden in regelmäßigen Abständen TCP-Acknowledgement (ACK)-Pakete versendet. Auch kann festgestellt werden, dass sowohl bei der Kamera als auch beim Smartphone eine andauernde TLS-Kommunikation stattfindet, wobei sogenannte *Applicaton Data* verschlüsselt versendet werden. Dies weist darauf hin, dass eine bereits bestehende Verbindung zur Datenübermittlung genutzt wird.

Die Kamera kommuniziert mit dem Amazon Web Services (AWS)-Server mit der IP-Adresse 3.65.16.105, das Smartphone mit dem Server 52.28.39.3. Die Ergebnisse der Analyse zeigen, dass das Smartphone überwiegend Daten an den Server sendet, die Kamera dagegen überwiegend Daten vom Server empfängt. Dies weist darauf hin, dass die App Befehle über den Server an die Kamera weiterleitet. Zusätzlich können im Datenverkehr der Kamera regelmäßige DNS-Anfragen zur Namensauflösung der Domain dm140s.cbs.boschsecurity festgestellt werden. Die Kamera erhält jedoch ausschließlich Start of Authority (SOA)-Antworten ohne Internet Protocol Version 6 (IPv6)-Adressen. Die Kommunikation erfolgt daraufhin vollständig über IPv4. Des Weiteren kann beim Datenverkehr der Kamera festgestellt werden, dass diese mehrfach bestehende Verbindungen beendet und neue initiiert. Zudem stellt die Kamera eine Network Time Protocol (NTP)-Anfrage an einen Server mit der IP-Adresse 18.134.134.61. Diese dient zur Zeitsynchronisation. Eine direkte Kommunikation zwischen Kamera und App innerhalb des lokalen Netzwerks findet nicht statt. Die gesamte Steuerung läuft zentral über die Bosch-Cloud.

In einem weiteren Untersuchungsszenario wird das Zusammenspiel zwischen der Kamera und der mobilen App während eines aktiven Livestreams analysiert. Die Netzwerkdaten zeigen, dass sowohl das Smartphone mit der IP-Adresse 10.0.1.112 als auch die Kamera mit der IP-Adresse 10.0.1.131 unabhängig voneinander und ausschließlich über das Internet mit verschiedenen Servern in der Bosch-Cloud kommunizieren. Zu keinem Zeitpunkt erfolgt eine direkte Kommunikation zwischen der Kamera und dem Smartphone.

Die App stellt unmittelbar nach Start des Livestreams mehrere parallele TLSv1.2-Verbindungen zum Server 52.28.39.3 über unterschiedliche Ports her. Es erfolgt kein TLS-Handshake. Das bedeutet, es werden bereits bestehende TLS-Verbindungen genutzt. Im Datenverkehr werden außerdem *TCP-Retransmissions* und *Duplicate ACKs* festgestellt. Diese dienen zur Vermeidung von Paketverlusten bei fortlaufenden Übertragungen. Des Weiteren können verschlüsselte *TLS-Alerts* und *TCP-Resets* identifiziert werden, die typisch für das Schließen einer Session sind. Die parallele Übertragung und der Ausgleich von Paketverlusten sowie das Beenden einer Session in dieser Form ist ein typisches Verhalten von fortlaufenden Datenübertragungen.

Zeitgleich nutzt die Kamera die bereits bestehenden TLS-Verbindungen über unterschiedliche Ports, um Daten verschlüsselt an den Server mit der IP-Adresse 3.65.16.105 zu übermitteln. Weiterhin können mehrere DNS-Anfragen identifiziert werden, die es ermöglichen, der IP-Adresse des Servers die Domain dm140s.cbs.boschsecurity.com zuzuordnen. Außerdem kann dem verwendeten NTP-Dienst time.aws.com die IP-Adresse 35.176.149.124 zugeordnet werden. Abschließend werden von der Kamera alle Sessions mit *TLS-Alerts* und Finish (Fin)-ACK-Sequenzen beendet.

Im vierten Szenario wird das Kommunikationsverhalten der Bosch-Kamera bei der Auslösung eines Screenshots über die zugehörige mobile App untersucht. Die Analyse zeigt, dass sowohl das Endgerät der Benutzerin bzw. des Benutzers (IP: 10.0.1.112) als auch die Kamera (IP: 10.0.1.131) über das Internet mit der Bosch-Cloud kommunizieren. Eine direkte Kommunikation im lokalen Netzwerk kann auch in diesem Fall nicht beobachtet werden.

Die App nutzt mehrere TLSv1.2-Verbindungen zum Cloud-Server mit der IP-Adresse 52.28.39.3. Diese Verbindungen erfolgen über unterschiedliche lokale Quellports, was auf die gleichzeitige Nutzung mehrerer TLS-Sitzungen zur parallelen Verarbeitung von Steuerbefehlen und Datenströmen hinweist. Des Weiteren erfolgen Verbindungen mit den Google-Diensten www.googleapis.com (IP-Adresse 172.217.16.74) und subscriptionsfirstparty-pa.googleapis.com, der dieselbe IP-Adresse nutzt. Der Dienst www.googleapis.com verwendet eine TLSv1.3-Verbindung, subscriptionsfirstparty-pa.googleapis.com hingegen eine TLSv1.2-Verbindung. Hierbei fällt auf, dass eine große Menge an Daten an subscriptionsfirstparty-pa.googleapis.com gesendet wird. Dies spricht dafür, dass ein Screenshot erstellt und an einen entsprechenden Google-Cloud-Server zur Speicherung weitergeleitet wird.

Auch bei der Aufnahme eines Screenshots wird der Datenverkehr der Kamera untersucht. Dabei kann ein ähnliches Verhalten wie bei der Nutzung des Livestreams beobachtet werden. Aus diesem Grund wird an dieser Stelle auf eine nähere Beschreibung verzichtet.

In nächsten Szenario wird das Verhalten der Kamera bei der Änderung von Einstellungen über die Bosch-App untersucht. Dabei kommunizieren sowohl die Kamera als auch die App auf dem Mobilgerät mit verschiedenen Bosch-Cloud-Servern. Die App nutzt dafür bestehende TLS-Verbindungen über unterschiedliche Ports. Ähnlich wie bei der Übertragung des Livestreams werden Paketverluste durch *TCP-Retransmissions* und *Duplicate ACKs kompensiert*. Die Verbindungen werden auch hier sauber mit TLS-Alerts und Fin-ACK-Sequenzen getrennt.

Kurz darauf ist eine Reaktion der Kamera erkennbar, die selbst eine TLS-verschlüsselte Verbindung zu 3.65.16.105 nutzt. Bei diesem Server handelt es sich ebenfalls um einen Bosch-Cloud-Server. Auch hier findet ein Austausch von *Application Data* statt. Die zeitliche Abfolge legt nahe, dass die vorgenommenen Einstellungen über die Bosch-Cloud an die Kamera weitergeleitet und dort unmittelbar umgesetzt werden. Eine direkte Kommunikation zwischen App und Kamera innerhalb des lokalen Netzwerks ist auch in diesem Fall nicht feststellbar. Die gesamte Übertragung der Steuerdaten verläuft verschlüsselt und vollständig über die Cloud-Infrastruktur von Bosch.

Im sechsten Szenario wird das Verhalten bei einer Alarmmeldung durch die Kamera analysiert. Beim Netzwerkverkehr der Kamera kann erneut kein Aufbau einer neuen TLS-Verbindung festgestellt werden. Die Kamera nutzt also, wie in den vorherigen Szenarien, eine bereits bestehende TLS-Verbindung zum Server mit der IP-Adresse 3.65.16.105. Auch in diesem Szenario stellt die Kamera regelmäßige DNS-Anfragen und erhält keine IPv6-Adresse. Ab einem bestimmten Zeitpunkt kann im Netzwerkverkehr festgestellt werden, dass auffällig große TLS-Pakete versendet werden. Dies ist ein Hinweis auf ein Alarmereignis und die Übertragung einer Aufzeichnung.

Bei der Analyse des Mitschnitts des Netzwerkverkehrs des Smartphones im Moment der Alarmmeldung wird festgestellt, dass mehrere verschlüsselte Verbindungen zu einem Cloudserver hergestellt werden. Hierfür werden mehrere parallele Kommunikationskanäle durch TLSv1.2-Handshakes auf unterschiedlichen Ports geöffnet. Die Kommunikation erfolgt dabei wieder mit dem Server mit der IP-Adresse 52.28.39.3 über Port 443. Bei diesem Server handelt es sich um den bereits bekannten AWS-Server, der über das SNI-Feld der Bosch-Infrastruktur zugeordnet werden kann. In diesem Feld ist der Domainname residential.cbs.boschsecurity.com hinterlegt.

Nach dem erfolgreichen Aufbau der Verbindung erfolgt die verschlüsselte Übertragung von Nutzdaten, was auf den Empfang und die Verarbeitung der Alarmmeldung hinweist. Zusätzlich fällt eine Kommunikation mit der Google-IP-Adresse 142.250.153.188 über Port 5228 auf. Bei dieser Adresse handelt es sich um eine bekannte IP-Adresse für Firebase Cloud Messaging. Das ist ein Hinweis darauf, dass zusätzlich zur Kommunikation mit der Cloud auch die Erstellung einer Push-Benachrichtigung erfolgt. Ein solches Verhalten ist typisch für die Meldung eines Alarmereignisses. Da die Daten aber vollständig verschlüsselt versendet werden, ist es nicht möglich, dies eindeutig zu belegen.

5.5.3 Ergebnisse MITM-Angriff

Die eingangs durchgeführte Google-Suche bestätigt die korrekte Konfiguration des MITM-Proxys. Dies ist in Abbildung 5.1 zu sehen. Der gesamte Datenverkehr zur Google-Suchanfrage wird sichtbar in der HTTP-Historie von *Burp Suite* protokolliert. Dies belegt eindeutig, dass sowohl die Proxy-Weiterleitung als auch das eingesetzte CA-Zertifikat funktionieren.

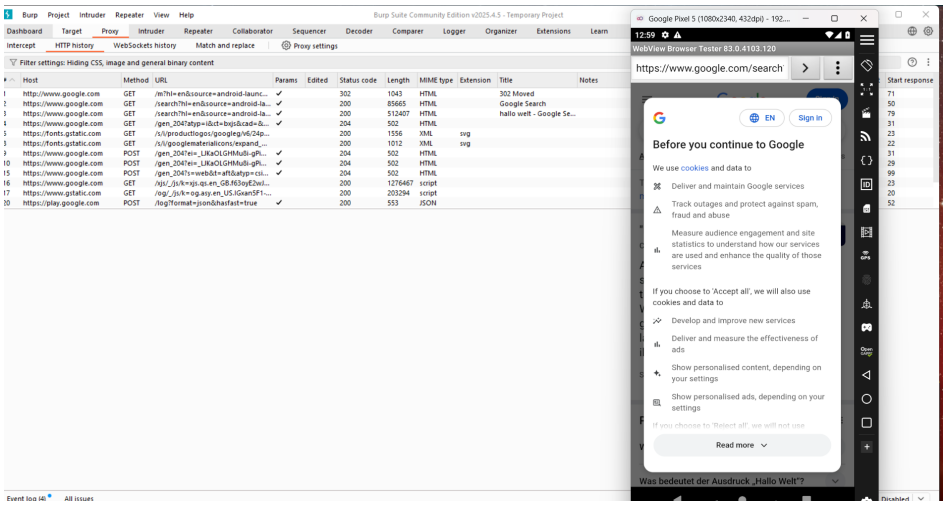

Abbildung 5.1: Test der Burp Suite Proxy Konfiguration

Anders verhält es sich beim Versuch, die Bosch Smart Camera App zu analysieren. Während das Starten der App selbst keine sicherheitsrelevanten Verbindungen offenbart, führt der Versuch der Anmeldung über den SingleKey-ID-Login zu einer Fehlermeldung im Emulator. Statt eines erfolgreichen Handshakes oder Authentifizierungsversuchs erscheint ein Dialog mit der Meldung „Untrusted connection“. Als Grund wird ein nicht vertrauenswürdiges Zertifikat genannt, das von „PortSwigger CA“ ausgestellt und für die Domain smarthome.authz.bosch.com verwendet wird. Diese Fehlermeldung ist in der nachfolgenden Abbildung 5.2 zu sehen.

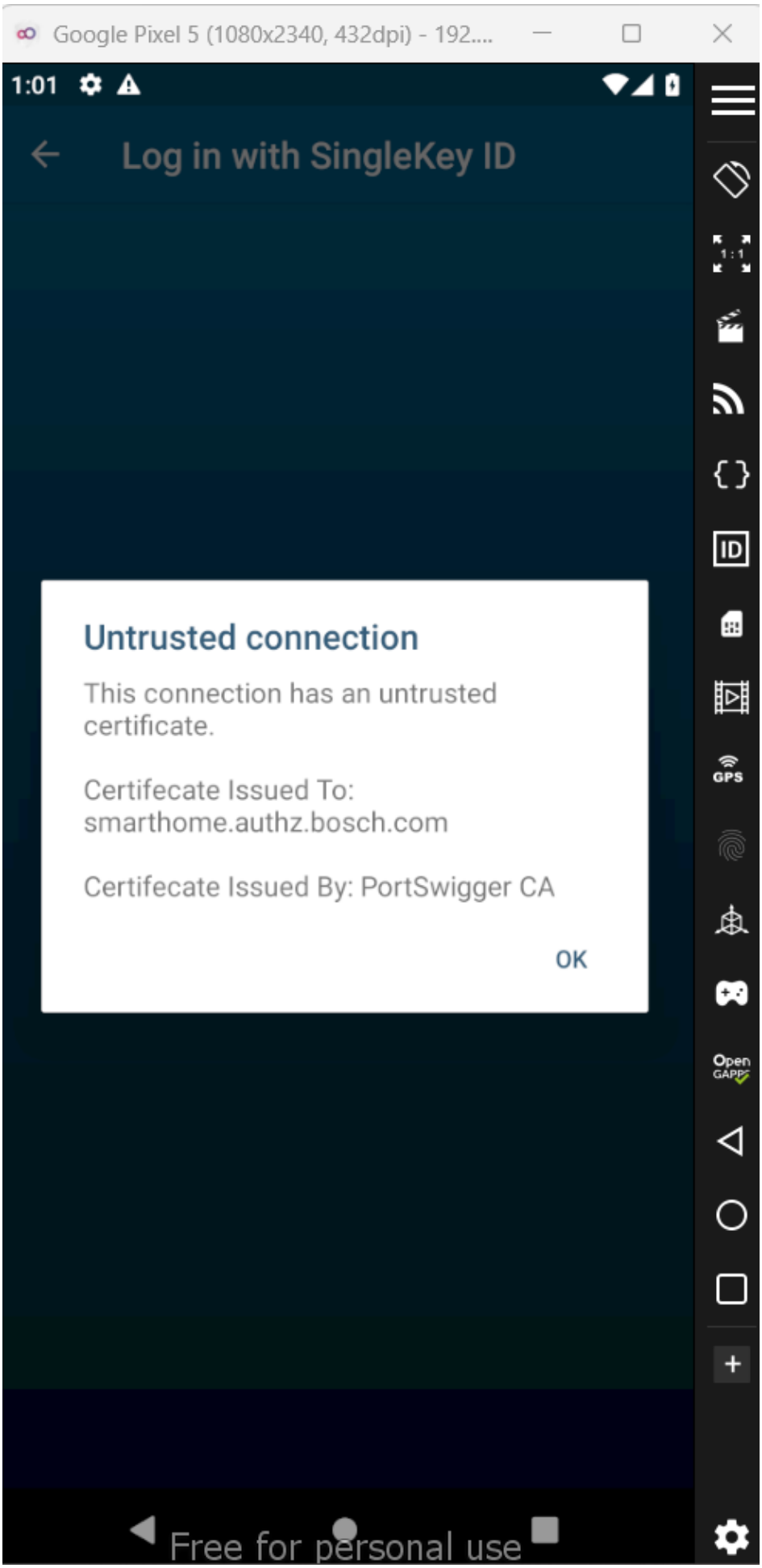

Abbildung 5.2: Bildschirm des AVDs nach dem Anmeldeversuch

In *Burp Suite* sind zu diesem Zeitpunkt keine Anfragen sichtbar. Dies deutet stark darauf hin, dass die App ein Zertifikat-Pinning implementiert. Da das von *Burp Suite* verwendete Zertifikat nicht dem ursprünglich vorgesehenen Serverzertifikat entspricht, wird der Handshake von der App aktiv unterbunden.

Insgesamt zeigt dieses Ergebnis, dass die Bosch Smart Camera App durch einen effektiven Mechanismus vor MITM-Angriffen geschützt ist.

6 Forensische Analyse der Bosch Smart Camera App

In diesem Kapitel werden das Vorgehen und die Ergebnisse der detaillierten Analyse der Bosch Smart Camera App dargestellt.

6.1 Vorbereitung der forensischen Analyse der Bosch Smart Camera App

Wie bereits im Methodikenteil beschrieben, müssen einige Vorbereitungen getroffen werden, um die App und ihre zugehörigen Daten zu analysieren. Dazu gehört die Bereitstellung und Installation der notwendigen Hard- und Software. So muss ein gerootetes Android-Smartphone zur Verfügung stehen. Bei diesem muss USB-Debugging im Entwicklermodus aktiviert werden. Um den Entwicklermodus zu aktivieren, muss in den Einstellungen zum Bereich „Über das Telefon“ oder „Über das Gerät“ navigiert werden. Dort wird siebenmal in Folge auf die Build-Nummer des Smartphones getippt. Des Weiteren muss auf dem Windows-Analyse-Laptop das ADB-Tool installiert werden. Dies erfolgt, indem man über die offizielle Android-Entwicklerwebseite die *SDK Plattform Tools* herunterlädt und installiert. Damit ADB systemweit in Windows genutzt werden kann, muss der Pfad zum ADB-Verzeichnis als Umgebungsvariable hinzugefügt werden. Damit können ADB-Befehle in jedem Verzeichnis ausgeführt werden, ohne den vollständigen Pfad zum ADB-Tool angeben zu müssen. Nach Schaffung dieser Voraussetzungen kann das Smartphone über USB mit dem Laptop verbunden werden. Auf diesem Weg ist die logische Sicherung der App und der dazugehörigen Daten möglich. Dies wird in Kapitel 6.2 näher beschrieben.

Neben der Analyse der Dateien, die auf der physischen Hardware durch die App gespeichert sind, wird untersucht, welche Daten die App bei der Ausführung in einer virtuellen Umgebung ablegt. Hierfür wird der Android Emulator *Genymotion* in der Version 3.9.0 genutzt. Dieser Emulator basiert auf der Virtualisierungssoftware *VirtualBox* und funktioniert ausschließlich mit einer *VirtualBox*-Version bis zur Version 7.0.26. Auch das mit *Genymotion* erzeugte AVD muss für die Analyse gerootet sein. Dies ist notwendig, damit die Daten extrahiert werden können und die dynamische Analyse mit *MobSF* durchgeführt werden kann. Ein emuliertes Gerät kann mit *Genymotion* bei der Erzeugung durch das Setzen eines Hakens ganz einfach gerootet werden. Auch die sonstige Einrichtung des AVDs kann einfach über das intuitiv bedienbare und nutzerfreundliche GUI von *Genymotion* erfolgen. Für die Funktionsfähigkeit der Bosch Smart Camera App sind der Google Play Store und die Google-Play-Dienste notwendig. Diese können nach dem Start des AVDs über *Open GApps* installiert werden. So kann auf dem AVD auch die Bosch Smart Camera App, wie bei einem physischen Gerät, aus dem Play Store heruntergeladen werden.

Bei der Einrichtung des AVDs muss zudem darauf geachtet werden, dass *MobSF* nur Android-Versionen zwischen Android 4.1 und Android 11 unterstützt. Die aktuellsten Funktionen von *MobSF* stehen jedoch erst ab Android Version 5.0 zur Verfügung. Ab dieser Version kann von

MobSF auch das Toolkit *Frida* eingesetzt werden. Daher empfiehlt sich für die Einrichtung des AVDs die Nutzung von Android 5.0 oder höher. Das Toolkit *Frida* ermöglicht es, Android-Anwendungen in Echtzeit auf Fehler zu untersuchen, zu manipulieren und zu analysieren. Dies wird durch die Injektion von Skripten in die laufende App ermöglicht und bringt große Vorteile bei der Untersuchung der Anwendung. In dieser Arbeit soll die Bosch Smart Camera App 2.8.0 untersucht werden. Diese ist erst ab Android 8.0 im Play Store verfügbar. Das bedeutet, für die Auswahl der Betriebssystemversion steht nur noch ein Bereich zwischen Android 8.0 und Android 11.0 zur Verfügung. Im Rahmen der vorliegenden Arbeit wird die Version Android 11.0 verwendet, da dies die aktuellste der nutzbaren Betriebssystemversionen ist. Diese Android-Version wird außerdem von der kostenlosen *Genymotion*-Version automatisch bei der Erzeugung gerootet.

Für die spätere Analyse der Daten wird die Software *MobSF* benötigt. Um die Installation und die Verwaltung zu vereinfachen, wird eine *Docker*-Umgebung für *MobSF* genutzt. Hierfür wird *Docker* auf dem Windows-Laptop installiert. Damit ist es möglich, *MobSF* plattformunabhängig in einem isolierten Container zu benutzen. Zur Konfiguration des *Docker*-Containers ist außerdem nur das Herunterladen des *MobSF*-Images notwendig. Danach kann der *Docker*-Container einfach über das Terminal gestartet werden. Dabei müssen nur die Ports für den Zugriff auf die Weboberfläche angegeben werden. Der Befehl für die Installation ist in Quelltext 6.1 dargestellt.

```
docker pull opensecurity/mobile-security-framework-mobsf:latest
```

Quelltext 6.1: Herunterladen des *MobSF-Docker*-Images

Der Befehl in Quelltext 6.2 zeigt, wie der Container gestartet werden kann.

```
docker run -it --rm \
-p 8000:8000 \
opensecurity/mobile-security-framework-mobsf:latest
```

Quelltext 6.2: Starten des *MobSF-Docker*-Containers

Dieser Befehl startet *MobSF* in einer *Docker*-Umgebung und ermöglicht durch die verwendeten Optionen die Interaktion mit dem Container über die Kommandozeile. Die Option *-it* setzt sich aus zwei Teilen zusammen: *-i* (interaktiv) und *-t* (Terminal). Sie stellt sicher, dass eine aktive Terminal-Sitzung mit dem Container besteht. Dadurch können Ausgaben direkt verfolgt und gegebenenfalls Eingaben über das Terminal gemacht werden, was besonders beim Debugging oder bei der Beobachtung des Startprozesses hilfreich ist.

Die Option *–rm* sorgt dafür, dass der Container nach dem Beenden automatisch gelöscht wird. Dies ist vor allem bei temporären Testläufen praktisch, kann aber bei längerfristiger Nutzung oder Analyse weggelassen werden, um den Container nach dem Stoppen wiederverwenden zu können.

Ein zentraler Bestandteil des Befehls ist die Portweiterleitung mit Hilfe der Option *-p*. In diesem Beispiel wird der Port *8000* des Host-Systems an den Port *8000* im Container weitergeleitet. Diese Weiterleitung ermöglicht den Zugriff auf die Weboberfläche von *MobSF* über den Webbrowser. Der externe (Host-)Port kann grundsätzlich frei gewählt werden. Es sollte jedoch darauf geachtet werden, dass kein bereits belegter oder systemkritischer Port verwendet wird.

Nach dem erfolgreichen Start des Containers kann die Weboberfläche von *MobSF* über http://localhost:8000 in einem Browser aufgerufen werden.

6.2 Sicherung der Daten mit ADB

In diesem Abschnitt soll nun beschrieben werden, wie bei der Extraktion der Daten vorgegangen wird. Zur logischen Sicherung wird das *ADB*-Tool verwendet.

Hierfür muss das gerootete Smartphone zuerst per USB mit dem Analyselaptop verbunden werden. Um zu testen, ob die Verbindung zwischen Smartphone und Laptop funktioniert, kann der Befehl *adb devices* genutzt werden. Sollte die Verbindung bestehen, wird das Smartphone in einer Liste angezeigt. Ein Beispiel für eine solche Liste ist in dem nachfolgenden Quelltext 6.3 zu sehen. Wenn die Liste keinen Eintrag enthält, wird kein verbundenes Gerät erkannt.

```
C:\Users\User>adb devices

List of devices attached
3B061JEHN07676  device
```

Quelltext 6.3: Anzeige angeschlossener Geräte mit *ADB*

Wenn die Verbindung zum Gerät funktioniert, erfolgt die Sicherung der App. Hierfür ist es notwendig, in der Verzeichnisstruktur den Pfad zu den APK-Dateien zu finden. Nur über diesen können die Anwendungsdateien gesichert werden. Im folgenden Quelltext 6.4 sind die dafür nötigen Befehle mit ihren Ausgaben zu sehen.

```
C:\Users\User>adb shell
1|lynx:/ # su
lynx:/ # pm list packages -f|grep bosch.cbs
package:/data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH
    -1Rvtu6fs_wi3VQ==/base.apk=com.bosch.cbs.consumer.prod
lynx:/ # pm path com.bosch.cbs.consumer.prod
package:/data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH
    -1Rvtu6fs_wi3VQ==/base.apk
package:/data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH
    -1Rvtu6fs_wi3VQ==/split_config.arm64_v8a.apk
package:/data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH
    -1Rvtu6fs_wi3VQ==/split_config.de.apk
package:/data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH
    -1Rvtu6fs_wi3VQ==/split_config.xxhdpi.apk
```

Quelltext 6.4: Bestimmung des Dateipfads zu den APKs

Der Quelltext 6.4 zeigt, dass eine *ADB*-Shell gestartet wird. Dies ermöglicht den direkten Zugriff auf das Dateisystem des verbundenen Android-Geräts über die Kommandozeile des Computers. Danach werden sich mit dem Kommando *su* Super-User-Rechte verschafft. Diese sind notwendig, um sich frei im System bewegen zu können. Ohne Root-Rechte würde die Ausführung der darauffolgenden Befehle keine Ausgaben zurückgeben. Im nächsten Schritt wird der Befehl *pm list packages -f | grep bosch.cbs* ausgeführt. Dieser listet alle installierten Pakete (*pm list packages*) zusammen mit ihrem Dateipfad (*-f*) auf und filtert die Ausgabe dann mit *grep bosch.cbs*, sodass nur Pakete angezeigt werden, deren Name „bosch.cbs “ enthält. Das Ergebnis zeigt den Pfad der Basis-APK-Datei für das gefundene Paket *com.bosch.cbs.consumer.prod*. Anschließend wird der Befehl *pm path com.bosch.cbs.consumer.prod* verwendet. Dieser Befehl fragt explizit den Dateipfad für das Paket *com.bosch.cbs.consumer.prod* ab. Die Ausgabe zeigt daraufhin die Pfade zu verschiedenen APK-Dateien, die zu dieser Anwendung gehören: die Basis-APK (*base.apk*) sowie Split-APKs für spezifische Architekturen (*arm64_v8a*), Sprachen (*de*) und Bildschirmdichten (*xxhdpi*). Split-APKs sind eine Möglichkeit, die Größe von Anwendungsdownloads zu optimieren, indem nur die für das jeweilige Gerät relevanten Komponenten heruntergeladen werden. Nach der Identifikation des Speicherpfads der APKs, können diese mit *adb pull* gesichert werden. Der allgemeine Aufbau eines Pull-Befehls ist in Quelltext 6.5 zu sehen. Bei diesem wird der Platzhalter <*Quell_Verzeichnis*> durch das bestimmte Verzeichnis */data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH-1Rvtu6fs_wi3VQ==* ersetzt. Der Platzhalter <*Ziel_Verzeichnis*> muss durch den Pfad ersetzt werden, in welchem die Daten auf dem Analyselaptop gespeichert werden sollen.

```
adb pull <Quell_Verzeichnis> <Ziel_Verzeichnis>
```

Quelltext 6.5: Allgemeiner Aufbau eines *adb pull*-Befehls

Nach der Sicherung der App, müssen die dazugehörigen Daten ebenfalls gesichert werden. Hierfür werden, ähnlich wie bei der Sicherung der APK-Dateien, die Verzeichnisse der Daten benötigt. Um diese zu bestimmen, wird erneut eine ADB-Shell gestartet und es werden sich

erneut Super-User-Rechte mit *su* verschafft. Danach erfolgt ein Wechsel ins Stammverzeichnis. So ist es möglich, das gesamte System nach relevanten Daten zu durchsuchen. Um alle Verzeichnisse aufzulisten, die Daten mit Bezug zur Bosch Smart Camera App enthalten, wird der Befehl 6.6 ausgeführt.

```
du | grep bosch.cbs
```

Quelltext 6.6: Filtern nach app-spezifischen Daten der Bosch Smart Camera App

Durch den Befehl *du* erhält man eine Auflistung aller Dateien und Verzeichnisse inklusive ihrer Speicherplatznutzung, ausgehend vom aktuellen Verzeichnis. Dies erfolgt standardmäßig rekursiv. Das heißt, *du* beschränkt sich nicht nur auf die Dateien und direkten Unterverzeichnisse des aktuellen Pfades, sondern erfasst auch alle darunterliegenden Unterverzeichnisebenen und bezieht deren Speicherverbrauch ein. So erhält man eine umfassende Übersicht über die Speicherplatznutzung einer gesamten Verzeichnisstruktur. Durch die Filterung mit *grep bosch.cbs* erhält man nur die Dateien und Verzeichnisse, die tatsächlich einen App-Bezug haben. Dadurch können die Dateipfade, die in Quelltext 6.7 dargestellt sind, herausgefiltert werden. Die Ergänzung der Zeilennummern vor den Pfaden, dient der besseren Nachvollziehbarkeit der folgenden Darstellungen. Die Angaben in Klammern sind zusätzlich eingefügte Bemerkungen.

```
1       ./storage/emulated/0/Android/data/com.bosch.cbs.consumer.prod (leer)

2       ./data_mirror/ref_profiles/com.bosch.cbs.consumer.prod
3       ./data_mirror/cur_profiles/0/com.bosch.cbs.consumer.prod
4       ./data_mirror/data_de/null/0/com.bosch.cbs.consumer.prod
5       ./data_mirror/data_ce/null/0/com.bosch.cbs.consumer.prod

6       ./data/misc/profiles/cur/0/com.bosch.cbs.consumer.prod
7       ./data/misc/profiles/ref/com.bosch.cbs.consumer.prod
8       ./data/app/~~fGVny7piawJIwz5kDTv7jw==/com.bosch.cbs.consumer.prod-TbgU2XH
    -1Rvtu6fs_wi3VQ== (App)
9       ./data/media/0/Android/data/com.bosch.cbs.consumer.prod (leer)
10      ./data/user/0/com.bosch.cbs.consumer.prod
11      ./data/user_de/0/com.bosch.cbs.consumer.prod
12      ./data/data/com.bosch.cbs.consumer.prod

13      ./mnt/pass_through/10/emulated/0/Android/data/com.bosch.cbs.consumer.prod
     (leer)
14      ./mnt/pass_through/0/emulated/0/Android/data/com.bosch.cbs.consumer.prod
    (leer)
15      ./mnt/androidwritable/0/emulated/0/Android/data/com.bosch.cbs.consumer.
    prod (leer)
16      ./mnt/installer/0/emulated/0/Android/data/com.bosch.cbs.consumer.prod (
    leer)
17      ./mnt/user/0/emulated/0/Android/data/com.bosch.cbs.consumer.prod (leer)
```

Quelltext 6.7: Auf dem Google Pixel 7a gefundene Verzeichnispfade mit App-Bezug

Diese Verzeichnisse werden ebenfalls mit *adb pull* gesichert. Dabei kann es durch SELinux dazu kommen, dass es trotz vorhandener Root-Rechte nicht möglich ist, die Daten zu kopieren. Sollte dies der Fall sein, kann mit *getenforce* in der ADB-Shell der aktuelle Modus abgefragt werden. Sollte dabei *Enforcing* zurückgegeben werden, muss mit *setenforce 0* der Modus geändert werden, damit ein Kopieren der Daten möglich ist.

Analog zur Extraktion der Daten von dem gerooteten Google Pixel 7a werden auch die Daten mit App-Bezug vom emulierten Google Pixel 5 gesichert. Im Quelltext 6.8 sind die Speicherpfade angegeben.

```
1       ./data_mirror/cur_profiles/0/com.bosch.cbs.consumer.prod
2       ./data_mirror/data_de/null/0/com.bosch.cbs.consumer.prod
3       ./data_mirror/data_ce/null/0/com.bosch.cbs.consumer.prod

4       ./data/app/~~ZJ5YGCjyXRbGBPUjP4tUTg==/com.bosch.cbs.consumer.prod-
    fggSRZgEbDhYDRE9HqfITQ==
5       ./data/user/0/com.bosch.cbs.consumer.prod
6       ./data/data/com.bosch.cbs.consumer.prod
7       ./data/system/graphicsstats/1747699200000/com.bosch.cbs.consumer.prod
8       ./data/system/graphicsstats/1747872000000/com.bosch.cbs.consumer.prod
9       ./data/user_de/0/com.bosch.cbs.consumer.prod
10      ./data/misc/profiles/ref/com.bosch.cbs.consumer.prod
11      ./data/misc/profiles/cur/0/com.bosch.cbs.consumer.prod
12      ./data/misc/iorapd/com.bosch.cbs.consumer.prod
```

Quelltext 6.8: Ausfindig gemachte Verzeichnispfade mit App-Bezug Google Pixel 5 (AVD)

6.3 Analyse der Bosch Smart Camera App

In diesem Kapitel soll beschrieben werden, wie die Bosch Smart Camera App mit *MobSF* statisch und dynamisch untersucht wird.

6.3.1 Statische Analyse

Mit Hilfe der statischen Analyse mit *MobSF* kann sich ein Überblick über Schwachstellen, potenzielle Sicherheitsrisiken und die Implementierung von Sicherheitsstandards der Bosch Smart Camera App, verschafft werden. Dafür analysiert *MobSF* den Quelltext und die Struktur der App, ohne dass diese ausgeführt werden muss. Nach der Analyse erstellt das Tool einen Bericht. Dieser stellt die Grundlage der Untersuchung der Anwendung dar.

In der vorliegenden Arbeit wird ausschließlich die Anwendungsversion 2.8.0 der Bosch Smart Camera App untersucht. Da es sich bei dieser um eine Split-APK handelt und man bei *MobSF* nur einzelne Dateien untersuchen kann, muss das vom Google Pixel 7a kopierte Verzeichnis komprimiert werden. Dadurch erhält man eine ZIP-Datei. Diese kann dann im entsprechenden Menüpunkt der Weboberfläche hochgeladen werden. Man kann die Dateiendung der ZIP-Datei in *.apks* ändern. Das ist aber nicht zwingend notwendig. Das Hochladen der Datei

funktioniert entweder über den Dateiexplorer oder per Drag and Drop. Durch das Hochladen startet *MobSF* automatisch die Analyse. Genaueres zum Scan, der vom Framework durchgeführt wird, findet sich in Kapitel 3.3.3. Der interaktive Bericht ist in Abbildung 6.1 zu sehen.

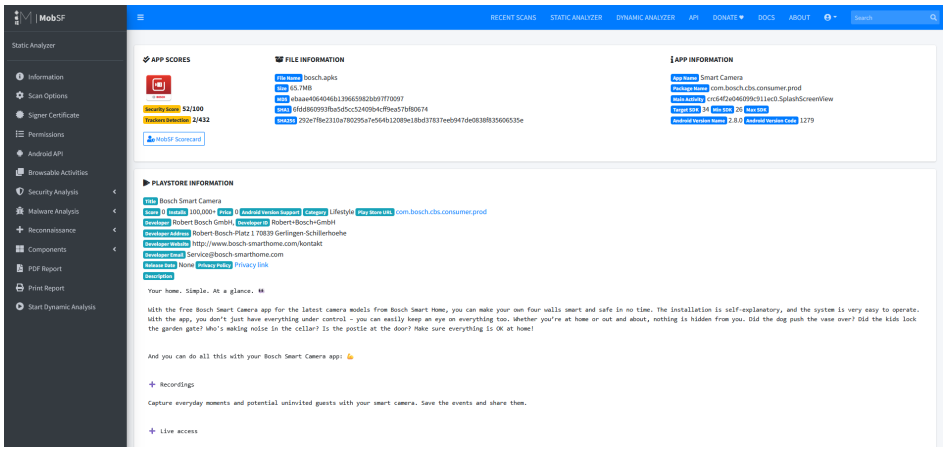

Abbildung 6.1: Ausschnitt aus dem dynamischen Bericht für die statische Analyse von *MobSF*

Der Bericht der statischen Analyse umfasst verschiedene Hauptkategorien. Unter diesen Gesichtspunkten kann die Sicherheit der Bosch Smart Camera App bewertet werden. Zu den Kategorien gehören unter anderem die detaillierte Erfassung von App-Informationen wie Paketname, Version, Ziel- und SDK-Version zur Identifizierung potenzieller Inkompatibilitäten oder Risiken. Weiterhin werden die exportierten Komponenten der App, beispielsweise Aktivitäten, Services und Broadcast Receiver, auf mögliche Sicherheitsprobleme hin untersucht. Dabei wird besonderes Augenmerk auf ungeschützte exportierte Aktivitäten als potenzielle Angriffsvektoren gelegt. Die Analyse der angeforderten Berechtigungen zielt darauf ab, übermäßige oder kritische Berechtigungen wie *READ_EXTERNAL_STORAGE* und *INTERNET* zu identifizieren, die von Schadsoftware missbraucht werden könnten. Die Sicherheit der verwendeten Verschlüsselungsalgorithmen und die korrekte Implementierung kryptografischer Verfahren werden ebenso geprüft wie die sichere Speicherung sensibler Daten. So werden schwächere Algorithmen wie MD5 und SHA-1 erkannt und bewertet. Die Untersuchung der Netzwerkkommunikation beinhaltet die Überprüfung der konsequenten Nutzung von HTTPS und der korrekten Implementierung von Zertifikat-Pinning, da diese ansonsten potenzielle Angriffspunkte darstellen können. Darüber hinaus werden hartkodierte Informationen und unsichere Programmierpraktiken wie Structured Query Language (SQL)-Injektion und Debug-Logs im Quellcode analysiert. Abschließend werden die identifizierten Schwachstellen im Kontext ihrer Wechselwirkungen bewertet, um ihre potenziellen Auswirkungen zu priorisieren.

6.3.2 Dynamische Analyse

Nach dem Abschluss der statischen Analyse erfolgt die dynamische Analyse mit *MobSF*. Dabei wird das Verhalten der App zur Laufzeit untersucht. Für diesen Zweck kommt in dieser Arbeit der „Dynamic Analyzer“ von *MobSF* zum Einsatz. Wie bereits in Kapitel 6.1 beschrieben, wird *MobSF* in einer *Docker*-Umgebung gestartet. Unter Windows benötigt *Docker* die Aktivierung bestimmter Virtualisierungsfunktionen, was jedoch zu einer erheblichen Verlangsamung des

AVDs führt. Die dadurch entstehenden Leistungseinbußen machen das emulierte Gerät für eine dynamische Analyse praktisch unbrauchbar. Aus diesem Grund wird der *MobSF-Docker*-Container auf einem separaten Kali-Linux-Analysegerät betrieben. Um dies umzusetzen, sind zusätzliche Vorbereitungsschritte erforderlich. *Docker* und das ADB-Tool müssen auf dem Kali-Linux-Laptop installiert sein. Zudem müssen sich sowohl der Windows- als auch der Linux-Rechner im selben Netzwerk befinden, da der *Docker*-Container Zugriff auf das emulierte Gerät benötigt. Gleichzeitig soll der Windows-Laptop weiterhin auf die grafische Benutzeroberfläche von *MobSF* zugreifen können. Damit der Zugriff vom Container auf das AVD möglich ist, sind zusätzliche Konfigurationsschritte nötig. Nach dem Start des Emulators auf dem Windows-Rechner wird in einem Terminal der Befehl *adb tcpip 5555* ausgeführt, um den Emulator in den TCP/IP-Modus zu versetzen. Der Bildschirm des gestarteten AVDs ist in Abbildung 6.2 zu sehen. Anschließend kann auf dem Kali-Linux-System eine Verbindung zum Emulator über den Befehl *adb connect 10.0.1.143:5555* hergestellt werden.

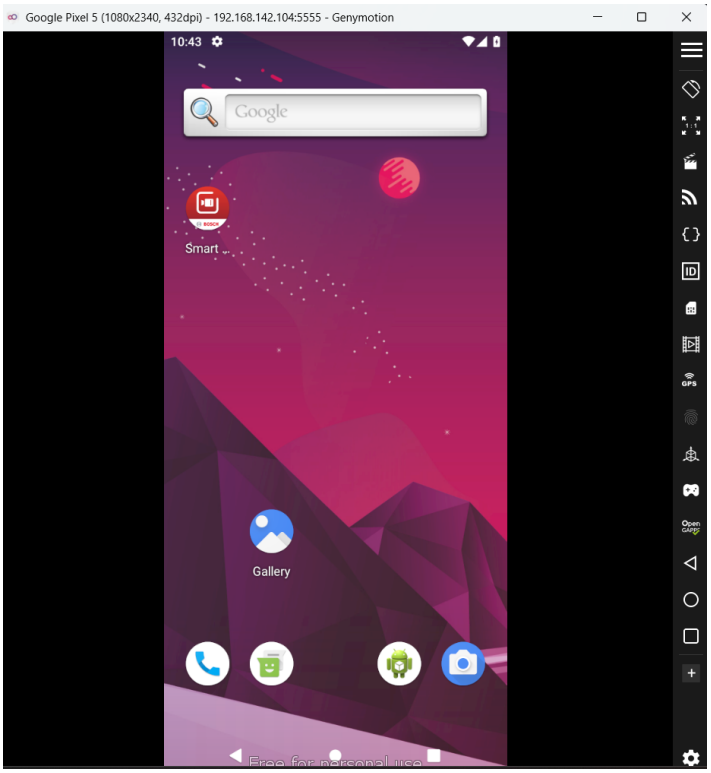

Abbildung 6.2: Bildschirm des gestarteten AVDs

Sobald die Verbindung erfolgreich aufgebaut ist, kann die App in der virtuellen Umgebung getestet und analysiert werden. Dazu wird der im Quelltext 6.9 dargestellte angepasste *Docker*-Befehl auf dem Kali-Linux-Analysegerät ausgeführt.

```
sudo docker run -it --rm \
-p 8000:8000 \
-p 1337:1337 \
-e MOBSF_ANALYZER_IDENTIFIER=10.0.1.143:5555 \
--network host \
opensecurity/mobile-security-framework-mobsf:latest
```

Quelltext 6.9: Startbefehl des MobSF-Docker-Containers für die dynamische Analyse

Dieser *docker run*-Befehl startet einen Container basierend auf dem Image *opensecurity/mobile-security-framework-mobsf* in der neuesten Version. Durch die Option *–rm* wird der Container nach Beendigung automatisch gelöscht. Die grafische Benutzeroberfläche von *MobSF*, die im Container auf Port 8000 läuft, ist über denselben Port auch auf dem Host-Rechner erreichbar. Zusätzlich wird über die Option *-p 1337:1337* der Port 1337 freigegeben. Dieser wird für den HTTP-Proxy verwendet, der eine zentrale Rolle bei der dynamischen Analyse mobiler Anwendungen in *MobSF* spielt. Des Weiteren wird mit *-e MOBSF_ANALYZER_IDENTIFIER=10.0.1.143:5555* die Umgebungsvariable *MOBSF_ANALYZER_IDENTIFIER* im Container auf 10.0.1.143:5555 gesetzt. Dieser Wert dient als Identifikator für die VM, die für die dynamische Analyse verwendet wird. Dieser Identifikator kann zum Beispiel mit dem Befehl *adb devices* auf dem Linux-Laptop bestimmt werden. Durch die Verwendung der Option *–network host* nutzt der Container direkt das Netzwerk des Host-Systems. Dies ist erforderlich, damit der Container ohne Einschränkungen auf den Android-Emulator im selben Netzwerk zugreifen kann, insbesondere um über *ADB* eine Verbindung auf Port 5555 herzustellen. Ohne diese Option wäre der Zugriff aus dem Container auf das externe Gerät nicht möglich, da *Docker* standardmäßig ein isoliertes Netzwerk verwendet. *opensecurity/mobile-security-framework-mobsf:latest* gibt erneut das zu verwendende *Docker*-Image an, wobei *:latest* die aktuellste verfügbare Version herunterlädt und startet. Nachdem die Einrichtung der virtuellen Umgebung und der Start des *Docker*-Containers erfolgt sind, kann die Weboberfläche von *MobSF* über http://localhost:8000 oder http://10.0.1.121:8000 in einem Browser aufgerufen werden. Bei der IP-Adresse 10.0.1.121 handelt es sich um die IP-Adresse des Linux-Laptops. Nach dem Öffnen der Weboberfläche kann nun über den Reiter „Dynamic Analyzer“ eine dynamische Analyse einer Anwendung gestartet werden. Hierfür wird der Button „Start Dynamic Analysis“ hinter der zu analysierenden Anwendung genutzt. Dies ist auch in Abbildung 6.3 zu sehen.

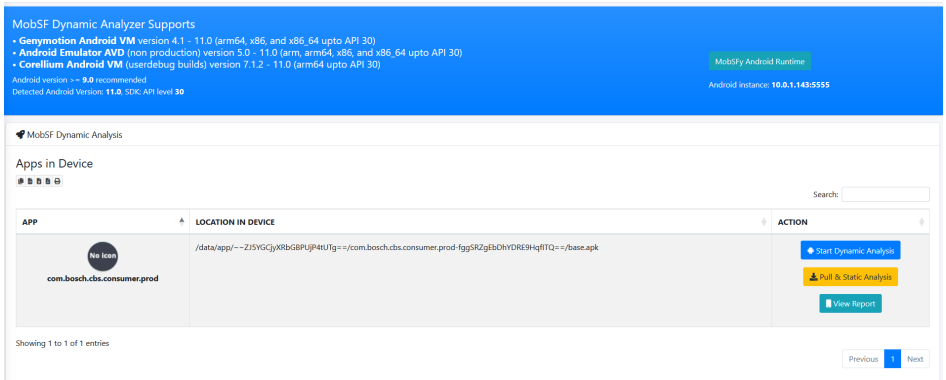

Abbildung 6.3: Ausschnitt „Dynamic Analyzer“ von *MobSF*

Nach dem Start der Analyse wird der Nutzer von *MobSF* automatisch auf eine interaktive Oberfläche weitergeleitet. In dieser stehen ihm eine Reihe verschiedener Funktionen zur Verfügung. Diese können einzeln oder in Kombinationen eingesetzt werden. Der Aufbau der Oberfläche ist in Abbildung 6.4 zu sehen.

Der „Dynamic Analyzer“ bietet verschiedene Analysemöglichkeiten. So kann über „Show Screen“ in Echtzeit auf den Bildschirm der Android-Umgebung zugegriffen werden.

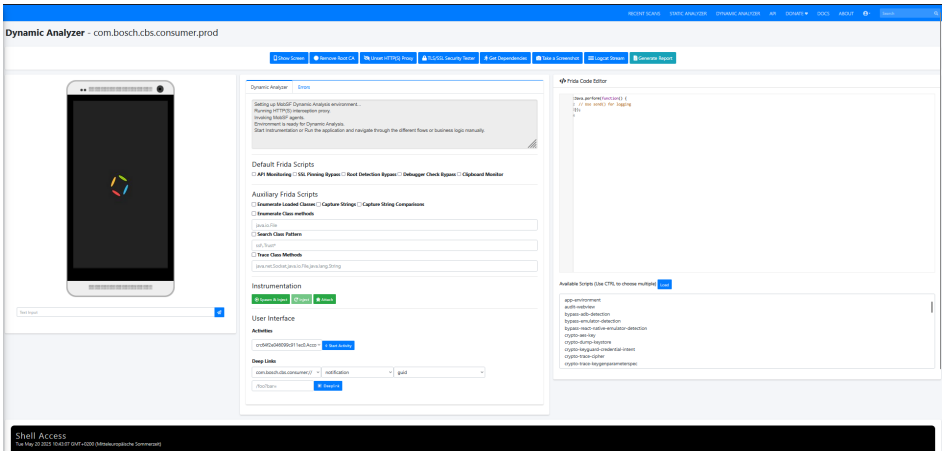

Abbildung 6.4: Benutzeroberfläche „Dynamic Analyzer“ von *MobSF*

„Remove Root CA“ entfernt das von *MobSF* installierte Root-Zertifikat vom Testgerät. Dieses Zertifikat wird von *MobSF* installiert, um den verschlüsselten Netzwerkverkehr der App während der dynamischen Analyse abzufangen und zu untersuchen. Dadurch können Schwächen in SSL/TLS-Implementierungen aufgedeckt werden, indem beispielsweise getestet wird, ob Applikationen gefälschte Zertifikate akzeptieren. Das Entfernen des Root-Zertifikats nach der Analyse ist eine wichtige Sicherheitsmaßnahme. Mit dem Button „Install Root CA“ kann dieses Zertifikat erneut installiert werden.

Mit „Unset HTTP(S) Proxy“ wird der von *MobSF* eingerichtete HTTPS-Proxy entfernt. Dies ermöglicht es, die Applikation auch ohne die Verwendung eines Proxys zu analysieren. Das kann interessant sein, wenn es die Vermutung gibt, dass eine App anders kommuniziert, sobald ein Proxy aktiv ist. Der Proxy kann erneut mit „Set HTTP(S) Proxy“ eingerichtet werden.

„TLS/SSL Security Tester“ bietet die Möglichkeit, die Implementierung von TLS/SSL zu untersuchen. Hierbei wird analysiert, wie die Applikation Zertifikate validiert, welche Verschlüsselungsprotokolle sie unterstützt und ob sie anfällig für bekannte Schwachstellen wie beispielsweise unsichere Cipher Suites oder das Akzeptieren abgelaufener Zertifikate ist. Cipher Suites sind eine Reihe von Algorithmen, die verwendet werden, um eine sichere Netzwerkverbindung über Protokolle wie TLS und dessen Vorgänger SSL auszuhandeln. Sie definieren, wie die Daten verschlüsselt und authentifiziert werden und wie deren Integrität geschützt wird. Ziel ist es, potenzielle MITM-Angriffe aufzudecken und sicherzustellen, dass die Kommunikation der Applikation über sichere Kanäle erfolgt.

Der „Exported Activity Tester“ ermöglicht das gezielte Aufrufen von als „exported“ deklarierten Activities einer Android-App. Exportierte Activities sind Komponenten der App, die von anderen Apps oder sogar direkt über *ADB* gestartet werden können. Durch das gezielte Aufrufen dieser Activities können Sicherheitslücken aufgedeckt werden. So kann beispielsweise das Umgehen von Berechtigungen, das Ausführen unerwarteter Funktionen oder die Preisgabe sensibler Informationen erkannt werden, wenn diese Activities nicht ausreichend vor unbefugtem Zugriff geschützt sind.

Mit „Get Dependencies“ kann getestet werden, welche Bibliotheken und Frameworks in der Android-App verwendet werden. Diese Funktion analysiert die Abhängigkeiten der App und gleicht sie mit bekannten Schwachstellen-Datenbanken ab. Dadurch können veraltete oder unsichere Bibliotheken identifiziert werden, die potenzielle Sicherheitsrisiken für die App darstellen. Die Erkennung der Sicherheitslücken in den Bibliotheken ist relevant, da Angreifer diese ausnutzen könnten.

Die Funktion „Take a Screenshot“ ermöglicht es, einen Screenshot des Bildschirms zu erstellen.

Mit „Logcat Stream“ kann die Echtzeit-Ausgabe des Android-Systemprotokolls (Logcat) eingesehen werden. Dieses Protokoll enthält detaillierte Informationen über das Verhalten der App und des zugrundeliegenden Android-Systems. Dies schließt API-Aufrufe, Fehlermeldungen, Debugging-Ausgaben und potenziell auch sensible Daten, die im Klartext protokolliert werden, ein. Durch die Überwachung des Logcat-Streams während der Ausführung der App können verdächtige Aktivitäten, Datenlecks oder unsichere Interaktionen in Echtzeit analysiert werden.

Der „Dynamic Analyzer“ bietet noch weitere Möglichkeiten zur Analyse der Anwendung. Dazu gehört die Verwendung von Frida-Skripten, die ausgewählt und geladen werden können. Diese stammen aus externen Datenquellen und werden von *MobSF* bereitgestellt.

In dieser Arbeit wird die App nach den folgenden Gesichtspunkten untersucht:

- Lokale Datenextraktion
- Netzwerkverkehr und API-Interaktionen
- User Interface (UI)- und Benutzerinteraktionen
- Kryptografie und Schlüsselverwaltung
- Umgehen von Sicherheitsmechanismen

Hierfür wird für jedes dieser Szenarien ein eigener Bericht erstellt. Es finden jedes Mal dieselben Interaktionen statt. Abhängig vom gesetzten Schwerpunkt erfolgte jedoch die Nutzung unterschiedlicher Frida-Skripte. Zur Bewertung der Ergebnisse wird überwiegend der automatisch erzeugte Bericht verwendet. Des Weiteren werden in die Bewertungen der API-Monitor-Stream, der Data Dump, der Frida-Out-Stream, der Logcat-Stream, der Web-Traffic-Stream sowie Applikationsdaten mit einbezogen.

Folgende Interaktionen werden mit der App durchgeführt, um Daten zu generieren:

1. Starten der App und Anmelden mit SingleKey ID
2. Einschalten der Kamera über die App
3. Öffnen und Beenden des Livestreams
4. Screenshot erstellen und speichern
5. Konfiguration der Kamera verändern
6. Kamera über die App ausschalten
7. App schließen

Im Nachfolgenden soll nun das Vorgehen bei den einzelnen Analyseszenarien vorgestellt werden.

Bei der lokalen Datenextraktion gilt es herausfinden, wo die Bosch Smart Camera App Dateien, Einstellungen oder Datenbanken speichert und welche forensisch relevanten Informationen sich dort befinden. Hierzu wird zunächst der Logcat-Stream aktiviert. Dies dient dazu, API-Aufrufe und Datenbankoperationen im Nachhinein zu identifizieren und zu bewerten. Danach werden die Frida-Skripte *trace_shared_preferences*, *trace_file*, *crypto-trace-cipher* und *dump-intent* aktiviert. *trace_shared_preferences* macht es möglich, Zugriffe auf die SharedPreferences zu analysieren. Dadurch ist es potenziell möglich, Klartextdaten wie Token, Konfigurationseinstellungen oder Benutzerdetails zu finden. Das Skript *trace-file* dient dazu, Dateioperationen der App zu erfassen. Das Skript *crypto-trace-cipher* wird verwendet, um die Nutzung von Verschlüsselungsalgorithmen durch die App zu verfolgen und so potenziell sensible Daten, die verschlüsselt gespeichert werden, zu identifizieren. Schließlich dient das Skript *dump-intent* dazu, die Kommunikation zwischen verschiedenen Komponenten der App zu überwachen, um so möglicherweise sensible Daten zu entdecken, die zwischen diesen Komponenten ausgetauscht werden. Durch die Kombination dieser Skripte mit der „Take a Screenshot“-Funktion von *MobSF* werden aus der Life-Analyse sofort erkennbare Informationen dokumentiert.

Bei der Analyse des Netzwerkverkehrs und der API-Interaktion ist es das Ziel, API-Requests und deren Inhalte sichtbar zu machen, um forensisch relevante Daten zu identifizieren, die über das Netzwerk übertragen werden. Hierfür wird erneut der Logcat-Stream aktiviert, damit HTTP/HTTPS-Anfragen und -Antworten analysiert werden können. Des Weiteren muss das Skript *ssl-pinning-bypass* aktiviert werden, um SSL-Pinning zu umgehen. SSL-Pinning ist eine Sicherheitsmaßnahme, die Man-in-the-Middle-Angriffe verhindert, indem die Applikation während der SSL-Verbindung zusätzlich prüft, ob das Zertifikat mit einem hinterlegten Zertifikat übereinstimmt. Das Skript *ssl-pinning-bypass* deaktiviert diese Prüfung, sodass der Datenverkehr über einen MITM-Proxy analysiert werden kann. Die Konfiguration eines solchen Proxys ist mit *MobSF* möglich. Des Weiteren wird das Frida-Skript *crypto-aes-key* angewendet. Dieses ermöglicht potenziell verschlüsselte Payloads zu entschlüsseln und so deren Inhalte zu analysieren. Mit dem *trace-file*-Skript können auch sensible Daten im Netzwerk verfolgt und analysiert werden. Zudem kann nachverfolgt werden, ob eine lokale Speicherung dieser Daten erfolgt. Erneut wird das *dump-intent*-Skript genutzt. Dieses Skript dient dazu, die API-Aufrufe zwischen den App-Komponenten zu überwachen. Weiterhin wird der integrierte „TLS/SSL Tester“ von *MobSF* in Kombination mit den Skripten verwendet. Damit ist es möglich, Schwachstellen im TLS/SSL-Stack der App zu identifizieren. Das heißt, es werden TLS/SSL-Sicherheitsprüfungen wie der Cleartext-Traffic-Test, eine TLS-Konfigurationsprüfung und SSL-Pinning-Tests durchgeführt. Diese Tests werden vom *MobSF*-Tool als „Cleartext Traffic Test“, „TLS Misconfiguration Test“, „TLS Pinning/Certificate Bypass Test“ und „TLS Pinning/-Certificate Transparency Test“ bezeichnet.

Das nächste Szenario setzt sich mit den UI- und Benutzerinteraktionen auseinander. Ziel hierbei ist es herauszufinden, welche Benutzerinteraktionen in der Bosch Smart Camera App stattfinden und welche potenziell forensisch relevanten Daten in dem UI angezeigt werden. Hierfür wurde unter anderem die Funktion „Take a Screenshot“ von *MobSF* genutzt. Zusätzlich werden auch hier wieder Frida-Skripte eingesetzt. Das Skript *ui-flag-secure-bypass* ermöglicht

es, Blockierungsversuche von Screenshots zu umgehen. Des Weiteren wird das Skript *app-enviroment* genutzt, um Textinhalte auf dem Bildschirm abzufangen. Kombiniert werden diese Skripte und die Funktion „Take a Screenshot“ mit „Logcat Stream“ und „Get Dependancies“, um im Nachgang relevante Informationen zu sammeln und zu analysieren.

Im vierten Szenario soll analysiert werden, welche kryptografischen Algorithmen und Schlüsselverwaltungspraktiken die Bosch Smart Camera App verwendet. Hierfür werden eine Reihe verschiedener Frida-Skripte genutzt. Mit dem Skript *crypto-dump-keystore* ist es möglich, auf die Schlüssel im Keystore zuzugreifen. Zudem kann mit *crypto-trace-cipher* überwacht werden, welche Verschlüsselungs- und Entschlüsselungsoperationen durchgeführt werden. Zusätzlich können mit *crypto-aes-key* Schlüssel zur Dekodierung sensibler Daten extrahiert werden. Mit dem Skript *crypto-trace-keystore* können auch Zugriffe auf den Android Keystore nachverfolgt werden. Neben diesen Skripten werden auch wieder der „TLS/SSL-Tester“ und das *ssl-pinning-bypass*-Skript verwendet.

Beim fünften Szenario wird analysiert, welche Sicherheitsmechanismen es gibt und ob diese umgangen werden können. Auch hierfür wurden wieder eine Reihe von Frida-Skripten genutzt. Mit dem Skript *bypass-adb-detection* kann das ADB-Debugging aktiviert werden. Damit ist es möglich, die App anschließend automatisch zu untersuchen. Des Weiteren wurde das Skript *bypass_emulator_detection* genutzt. Dieses Skript ermöglicht es zu verhindern, dass ein Emulator als solcher erkannt wird. Damit findet dann auch keine Anpassung des Verhaltens statt. Um verschiedene Skripte und Mechanismen testen zu können, ohne dass es zu Abstürzen kommt oder Schutzmechanismen greifen, kann das Skript *stop-app-terminate* verwendet werden. Mit diesem Skript werden Abstürze oder Schutzmechanismen unterbunden. Auch in diesem Szenario wurde der Logcat-Stream aktiviert, um Debugging-relevante Informationen wie Fehlermeldungen oder Anti-Debugging-Aktionen zu erfassen.

6.4 Untersuchung der Appdaten

An dieser Stelle wird das Vorgehen bei der Analyse der Appdaten beschrieben. Hierfür muss sich zunächst ein Überblick über die gesicherten Verzeichnisse verschafft werden. Danach ist es möglich, die in den Verzeichnissen enthaltenen Dateien zu analysieren und so potenziell forensisch relevante Informationen zu gewinnen. Die Sicherung erfolgt, wie bereits in Kapitel 6.2 beschrieben, sowohl bei dem emulierten Google Pixel 5 als auch beim gerooteten Google Pixel 7a. In den Quelltexten 6.7 und 6.8 sind die gesicherten Verzeichnisse dargestellt. Die generelle Verzeichnisstruktur bei dem AVD unterscheidet sich nicht wesentlich von der beim Google Pixel 7a. Es wurden jedoch einige Unterschiede festgestellt. Das Vorgehen bei der Datenanalyse ist dennoch identisch. Die Unterschiede in der Verzeichnisstruktur der Daten mit App-Bezug werden an den entsprechenden Stellen aufgezeigt. Zur besseren Nachvollziehbarkeit der Beschreibungen sind die Pfade nummeriert. Auf diese Nummerierung wird nun zur eindeutigen Beschreibung zurückgegriffen. Wenn nicht explizit erwähnt wird, dass sich die Pfadnummer auf das Google Pixel 5 bezieht, ist damit immer die Nummerierung des physischen Geräts gemeint (siehe Quelltext 6.7).

Der Pfad 1 befindet sich im Bereich des Dateisystems, der dem Nutzer zugänglich ist. Solche Verzeichnisse werden auch als Benutzerverzeichnisse bezeichnet. In diesen kann die Anwendung medienbezogene Inhalte speichern. Hier könnten forensisch relevante Daten wie Konfigurationsdateien, gespeicherte Screenshots bzw. Videos zu finden sein. Im emulierten Gerät konnte im *storage*-Verzeichnis kein Unterverzeichnis mit App-Bezug festgestellt werden. Daher kann nach der Filterung der Ausgaben kein Verzeichnispfad im *storage*-Verzeichnis festgestellt werden. Deshalb ist dieser in Quelltext 6.8 nicht enthalten.

Bei den Punkten 13 bis 17 handelt es sich um verschiedene Mountpoints. Diese werden für unterschiedliche Zugriffsarten benötigt. Beispiele für diese Zugriffsarten sind Prozesse der Installation und Aktualisierung, Userzugriffe oder Schreibzugriffe. Die Mountpoints zeigen alle auf den gleichen Speicherbereich. Bei diesem handelt es sich um Verzeichnis 1. Der Inhalt des Verzeichnisses 1 und der Mountpoints ist identisch. Beim emulierten Gerät ist das Verzeichnis *mnt* vorhanden. Es konnten jedoch auch hier keine Unterverzeichnisse mit App-Bezug festgestellt werden. Daher enthält der Quelltext 6.8 dieses Verzeichnis nach der Filterung nicht.

Die Pfade 2 bis 5 liegen im `./data_mirror`-Verzeichnis. Sie sind temporäre Spiegelungen des Datenbereichs im RAM. Sie dienen dazu, die Sicherheit bei der App-Ausführung durch Isolierung zu erhöhen. So wird verhindert, dass andere Applikationen versehentlich oder absichtlich auf App-Daten zugreifen können.

Bei den Pfaden 2 und 3 handelt es sich um Spiegelungen der Laufzeit- und Referenzprofile (Pfade 6 und 7 beim emulierten Gerät). Sie dienen der Optimierung der App-Ausführung. In diesen Pfaden werden Daten darüber gespeichert, welche Methoden und Codepfade häufig genutzt werden. Mit diesen Daten kann die App nach einem Reboot schneller starten. Die Daten der Pfade 2 und 3 sind dem Benutzer nicht direkt zugänglich und enthalten keine Nutzerdaten, sind jedoch relevant für die Performance der App. Eine ähnliche Struktur findet sich auch beim emulierten Gerät. Hierbei ist jedoch auffällig, dass keine Kopie des Referenzprofils in `data_mirror` zu finden ist.

Pfad 4 ist eine sichere Spiegelung von Pfad 11. Der Pfad 5 ist eine Kopie von Pfad 10 und Pfad 12. Die Pfade 4 und 5 dienen der Trennung von geräteverschlüsselten und benutzerverschlüsselten App-Daten. Der Unterordner *data_de* speichert Daten, die direkt nach dem Gerätestart verfügbar sind, noch bevor sich ein Benutzer authentifiziert hat. Dies können zum Beispiel Cache-Dateien oder Systemlogs sein. Der Unterordner *data_ce* enthält dagegen benutzerverschlüsselte Daten, die erst nach erfolgreicher Entsperrung des Geräts (zum Beispiel per Persönliche Identifikationsnummer (PIN) oder biometrisch) zugänglich sind. Hier können sicherheitskritische Informationen wie Sitzungsschlüssel, Passwörter oder App-Konfigurationen abgelegt sein. Auch diese Struktur findet sich beim emulierten Gerät.

Das zentrale Verzeichnis für Benutzerdaten ist *./data*. Es liegt auf der `/data`-Partition, die bei jedem Gerät vorhanden ist und vom System beim Boot gemountet wird. Es enthält Daten wie Appdaten, Benutzereinstellungen, Verschlüsselungsdaten, temporäre Dateien und systeminterne Profile sowie Metadaten.

In den Verzeichnispfaden 6 und 7 befinden sich die Optimierungsdaten der App. Diese haben für forensische Untersuchungen in der Regel keine Relevanz. Auch diese Verzeichnispfade finden sich beim emulierten Gerät. Das Unterverzeichnis `data/misc` enthält hier zusätzlich ein Unterverzeichnis *iorapd* (Quelltext 6.8 Pfad 12). Dies enthält Dateien, die die Appstartzeit optimieren sollen.

Pfad 8 der Verzeichnisstruktur des physischen Geräts (Quelltext 6.7) enthält die installierten APK-Dateien. Dort befinden sich also der Code und die Ressourcen der App. Auch für diesen Verzeichnispfad findet sich eine Entsprechung auf dem emulierten Gerät.

Bei Pfad 9 handelt es sich um ein Verzeichnis, in dem Medien im appspezifischen Teil des Dateisystems gespeichert werden können. Dieser ist jedoch leer. Im emulierten Gerät findet sich im Verzeichnis *data* das Unterverzeichnis *media*. Es konnten jedoch keine Ordner mit App-Bezug festgestellt werden. Aus diesem Grund enthält der Quelltext 6.8 keinen entsprechenden Verzeichnispfad.

Der Pfad 10 beinhaltet die Appdaten des Hauptnutzers. Das klassische Appdatenverzeichnis `./data/data` ist heute ein symbolischer Link auf dieses Verzeichnis. Damit weisen Pfad 10 und 12 auf denselben Speicher und enthalten die Appdaten, die erst nach der Entsperrung des Geräts zugänglich sind. Im Gegensatz dazu enthält Pfad 11 die Daten der App, welche bereits vor der Entsperrung zugänglich sind. Für diese Verzeichnispfade finden sich auf dem emulierten Gerät Entsprechungen. Auf dem emulierten Gerät konnten außerdem zwei Pfade festgestellt werden, die Teil des Android-Systems sind und Grafikleistungsstatistiken enthalten. Diese Pfade sind in Quelltext 6.8 als 7 und 8 nummeriert.

Wie oben beschrieben, enthalten einige Verzeichnisse Spiegelungen anderer Verzeichnisse. Deshalb ist deren Inhalt identisch. So enthalten Verzeichnis 5, 10 und 12 dieselben Unterverzeichnisse. Gleiches gilt auch für die Entsprechungen bei dem AVD. Diese Unterverzeichnisse sollen im Folgenden näher erläutert werden. Dabei handelt es sich um:

- *app_textures*
- *app_webview*
- *database*
- *files*
- *no_backup* (ist auf dem emulierten Gerät nicht vorhanden)
- *shared_prefs*

Die Unterverzeichnisse *app_textures* und *no_backup* sind leer. Aus diesem Grund haben sie für die forensische Untersuchung keine weitere Relevanz.

Das Verzeichnis *app_webview* enthält Informationen der eingebetteten Webansichten. Diese können mit einem Texteditor und dem *DB Browser for SQLite* relativ einfach gewonnen werden. So lassen sich dort beispielsweise Webdaten, Cookies und gespeicherte Sitzungsdaten finden.

Im Verzeichnis *database* befinden sich in der Regel SQLite-Datenbanken. Aus diesen können Informationen wie Benutzeranmeldedaten, API-Schlüssel und Token gewonnen werden. Diese Datenbanken können mit dem *DB Browser for SQLite* geöffnet und untersucht werden. Im Verzeichnis *database* ist eine Vielzahl von Datenbanken angelegt, die jedoch alle keine Daten enthalten.

Im *files*-Verzeichnis werden appspezifische Daten wie Logs, temporäre Dateien und gespeicherte Daten abgelegt. Im konkreten Fall ist die Analyse der hier gespeicherten JSON-Dateien von zentraler Bedeutung. Diese wurden mit *Microsoft Visual Studio 22* geöffnet und analysiert. Aus den anderen Dateien und Verzeichnissen konnten keine Informationen gewonnen werden.

In *no_backup* sind Daten gespeichert, die nicht ins Backup aufgenommen werden. Bei der Analyse mit *DB Browser for SQLite* wurde in diesem Ordner eine Datenbank festgestellt, die jedoch keine Informationen enthält.

Im Verzeichnis *shared_prefs* befinden sich eine ganze Reihe von XML-Dateien. Diese enthalten App-Einstellungen und Konfigurationsdaten. In diesen Dateien können einige forensisch relevante Daten enthalten sein, wie beispielsweise Passwörter, Benutzerdaten und Token. Zur Analyse wurde hier erneut der Texteditor eingesetzt.

Auch in den Verzeichnissen 4 und 11 wurden folgende identische Unterverzeichnisse festgestellt:

- *cache*
- *code_cache*
- *database*
- *files*
- *no_backup*
- *shared_prefs*.

Die entsprechenden Verzeichnisse auf dem emulierten Gerät weisen jedoch eine andere Struktur auf. Diese ist im Folgenden dargestellt:

- *app_phenotype_hermetic*
- *cache*
- *code_cache*
- *shared_prefs*.

Im Google Pixel 7a enthält nur das Unterverzeichnis *cache* eine kompilierte Grafikroutine. Solche Dateien haben meist keine forensische Relevanz. Alle anderen Ordner sind leer.
Im emulierten Gerät sind alle enthaltenen Unterverzeichnisse leer.

Die Verzeichnisse 2 und 7 sind, wie bereits beschrieben, ebenfalls identisch und beide leer. Die Entsprechung im emulierten Gerät enthält im Gegensatz dazu eine Datei *primary.prof*.

Die Verzeichnisse 3 und 6 sind ebenfalls identisch und enthalten eine Datei *primary.prof*. Bei dieser handelt es sich um eine Konfigurations- oder Profildatei. Mit Hilfe von *Android Studio* und dem Hexeditor *HxD* kann versucht werden, die Datei zu analysieren. Im konkreten Fall sind jedoch keine relevanten Informationen erkennbar. Diesen Verzeichnissen entsprechen die Pfade 1 und 11 im emulierten Gerät (Quelltext 6.8). Sie weisen den gleichen Inhalt auf.

6.5 Ergebnisse der Appanalyse

In diesem Abschnitt werden die zentralen Ergebnisse der forensischen Analyse der Bosch Smart Camera App zusammengefasst. Dabei werden die Resultate aus der statischen und der dynamischen Analyse sowie aus der Datenextraktion mit ADB detailliert dargestellt. Der Fokus der Untersuchung und Bewertung liegt auf der Identifikation sicherheitskritischer Schwachstellen, der Bewertung der Anwendungsdaten und der Untersuchung potenziell forensisch relevanter Informationen.

6.5.1 Ergebnisse der statischen Analyse der Bosch Smart Camera App 2.8.0

Der *MobSF*-Bericht zur statischen Analyse wird hinsichtlich sicherheitsrelevanter und forensisch bedeutsamer Informationen ausgewertet.

Zuerst werden die von der App überlieferten Playstoreinformationen untersucht. Sie geben wichtige Hinweise zur Herkunft und Authentizität der Anwendung und des Entwicklers. Zudem helfen sie, potenzielle Sicherheitsrisiken durch Klone oder gefälschte Versionen zu erkennen. Der Paketname der App lautet: *com.bosch.cbs.consumer.prod*. Er stellt einen eindeutigen Identifikator der Anwendung im Google Playstore dar. Als Entwickler der App wird die Robert Bosch GmbH angegeben. Die Playstore-Seite enthält offizielle Kontaktinformationen. Zu diesen gehören eine E-Mail-Adresse und eine Webseite, die auf den Entwickler verweist.

Neben diesen Informationen stellt auch das Zertifikat der App einen Nachweis der Herkunft und Authentizität der Anwendung dar. Auch dieses Zertifikat kann der Robert Bosch GmbH zugeordnet werden. Die Analyse der App-Signatur zeigt, dass diese mit dem Algorithmus *SHA1withRSA* signiert ist. Das heißt, zum Hashen der Daten wird der Algorithmus SHA-1 verwendet. Dieser gilt seit mehreren Jahren als kryptografisch unsicher. Er ist anfällig für Kollisionen. Das bedeutet, es ist möglich, zwei Eingaben in die Hashfunktion zu finden, die dieselbe Ausgabe erzeugen. Dieser Umstand kann dazu führen, dass manipulierte oder gefälschte APK-Dateien als gültig signiert erkannt werden. Der zweite Teil des Algorithmus ist Rivest–Shamir–Adleman (RSA). Dieses Verfahren dient zur Erzeugung und Überprüfung der digitalen Signatur. Es handelt sich hierbei um ein weit verbreitetes und sicheres Public-Key-Kryptosystem. Es wird genutzt, um den mit SHA-1 erzeugten Hashwert der Daten mit einem privaten Schlüssel zu verschlüsseln und so die Signatur zu erzeugen.

Ein zentraler Bestandteil der Analyse ist die Untersuchung der von der App angeforderten Berechtigungen. Insgesamt erkennt *MobSF* über 40 Berechtigungen, darunter einige mit erhöhtem Sicherheitsrisiko. *MobSF* identifizierte die Netzwerkberechtigungen

android.permission.INTERNET und *ACCESS_NETWORK_STATE*. Diese ermöglichen der Anwendung das Herstellen und Überwachen von Netzwerkverbindungen. Die beiden Berechtigungen sind häufige Einstiegspunkte für Datenexfiltration durch schadhafte Anwendungen.

Die App nutzt die Berechtigung *BLUETOOTH_CONNECT*. Mit dieser ist es der App möglich, sich mit den gekoppelten Bluetooth-Geräten zu verbinden. Dies kann von Schadsoftware dazu genutzt werden, Daten zu stehlen oder gekoppelte Geräte fernzusteuern. Des Weiteren verschafft sich die Anwendung mit den Speicherberechtigungen *READ_EXTERNAL_STORAGE* und *WRITE_EXTERNAL_STORAGE* Zugriff auf externe Speicherbereiche. Im externen Speicher können sich dabei potenziell sensible Nutzerdaten, wie Bilder und Dokumente befinden. Aus diesem Grund nutzt Malware diese beiden Berechtigungen häufig, um Daten zu stehlen oder Daten persistent zu hinterlegen.

Mit den Berechtigungen *ACCESS_COARSE_LOCATION* und *ACCESS_FINE_LOCATION* erlangt die App zudem Zugriff auf den Standort des Nutzers. *ACCESS_COARSE_LOCATION* ermöglicht dabei die ungefähre Standortbestimmung über WLAN-Netze und die Mobilfunkmasten. *ACCESS_FINE_LOCATION* nutzt unter anderem Global Positioning System (GPS) für eine genaue Standortbestimmung. Mit diesen Berechtigungen wäre es einer Malware möglich, zum Beispiel ein Bewegungsprofil des Nutzers zu erstellen oder dessen genauen Standort zu bestimmen.

Weiterhin können die Berechtigungen *GET_ACCOUNTS*, *READ_CONTACTS* und *READ_PROFILE* festgestellt werden. Mit *GET_ACCOUNTS* ist es möglich, Zugriff auf die Liste von Benutzerkonten, die auf dem Mobilgerät gespeichert sind, zu erhalten. *READ_PROFILE* gewährt Zugriff auf das persönliche Nutzerprofil des Geräts. Kombiniert man diese beiden Berechtigungen, erhält man ein vollständiges Profil des Nutzers. Verbindet man dies noch mit Kontaktinformationen aus *READ_CONTACTS*, hätte ein Angreifer ein vollständiges Profil des Nutzers mit seinen Kontakten.

Des Weiteren fallen im Bericht auch die Berechtigungen *RECORD_AUDIO* und *CAMERA* auf. Diese ermöglichen den Zugriff auf Mikrofon und Kamera, was aus forensischer Sicht besonders kritisch zu bewerten ist, da hiermit potenziell vertrauliche Informationen abgefangen werden können.

Die systeminterne Berechtigung *RECEIVE_BOOT_COMPLETED* erlaubt das Starten der App direkt nach dem Booten des Geräts. Dies ist eine typische Berechtigung für persistente Hintergrundaktivitäten. *WAKE_LOCK* verhindert das Einschlafen des Geräts, was bei dauerhaften Hintergrundprozessen genutzt wird. Zusammen mit der Berechtigung *POST_NOTIFICATIONS*, die es der Anwendung erlaubt Benachrichtigungen an den Nutzer zu senden, könnte die App theoretisch dauerhaft aktiv bleiben und verdeckt Aktionen durchführen. Dies ist möglich, da mit *POST_NOTIFICATIONS* nicht nur Benachrichtigungen geschickt werden, sondern auch stille Kommandos ausgeführt werden können. Zu solchen stillen Kommandos gehören zum Beispiel das Aktualisieren von App-Inhalten, das Synchronisieren von Daten, das Starten von Hintergrunddiensten, das Aktivieren oder das Deaktivieren von Features und das Auslösen bestimmter Aktionen. Dabei wird im Hintergrund, ohne dass der Nutzer es bemerkt, ein Code ausgeführt. *MobSF* identifiziert insgesamt 14 von 25 sogenannten „Malware Permissions“, also Berechtigungen, die erfahrungsgemäß häufig in Schadsoftware eingesetzt werden.

Bei der *APKID-Analyse* der App zeigen sich mehrere Hinweise darauf, dass Mechanismen verwendet werden, die die Analyse der Anwendung erschweren sollen. Solche Techniken finden sich oft bei Malware oder verschleierter Software und stellen für forensische Untersuchungen eine Herausforderung dar. *MobSF* stellt bei der *APKID-Analyse* unter anderem fest, dass die App über Mechanismen verfügt, die speziell darauf ausgelegt sind, VMs zu erkennen und deren Ausführung zu blockieren. Dies soll das Testen und Debuggen in kontrollierten Umgebungen verhindern. Solche Anti-VM-Techniken können ein starkes Indiz für Malware sein, da diese versuchen, automatisierte Sicherheitsprüfungen zu umgehen. Weiterhin nutzt die App Anti-Debugging-Techniken. Diese erschweren die dynamische Analyse. Solche Mechanismen sollen Debugger blockieren oder falsche Daten liefern und damit die Analyse erschweren. Man findet diese Techniken vor allem bei Software, die versucht, ihre internen Prozesse vor einer externen Untersuchung zu schützen. Dies wird sowohl von schädlichen Anwendungen als auch von manipulierten Anwendungen genutzt. Des Weiteren kann festgestellt werden, dass der *Compiler r8 ohne Marker* zum Kompilieren verwendet wurde. Dies deutet darauf hin, dass der Quellcode vor einer einfachen Analyse geschützt werden soll. Es gilt aber anzumerken, dass nicht nur schädliche Software Anti-VM- und Anti-Debugging-Mechanismen nutzt.

Bei der Analyse der Manifest-Datei der Anwendung sind mehrere sicherheitskritische Punkte aufgefallen. So ist die App auf Geräten ab *Android 8.0* (API level 26) installierbar. Diese Version gilt mittlerweile als veraltet und weist eine Reihe von Sicherheitslücken auf. Die Ausführung der App in Versionen unter *Android 10* bringt ein erhöhtes Risiko für Angriffe wie Privilegieneskalation und Datenmanipulation mit sich.

Des Weiteren kann festgestellt werden, dass in der Manifest-Datei der Parameter *android:allowBackup="true"* gesetzt ist. Dies erlaubt es dem System, ein Backup der App-Daten zu erstellen. Das ist sowohl über die Google-Backup-Funktion als auch über die ADB möglich. Bei aktivierter Entwickleroption und angeschlossenem Gerät kann so ein vollständiges App-Backup inklusive sensibler Daten wie Konfigurationsdateien, Datenbanken oder Authentifizierungsinformationen erstellt werden, ohne dass Root-Rechte erforderlich sind. Dieser Umstand stellt insbesondere im Kontext einer sicherheitskritischen Anwendung wie einer Smart-Home- oder Kamera-App ein erhöhtes Risiko dar.

Neben diesen beiden Punkten wird festgestellt, dass mehrere sogenannte Broadcast Receiver genutzt werden. Broadcast Receiver sind Android-Komponenten, die auf Systemereignisse oder App-spezifische Signale reagieren und dann Codes ausführen. Einige dieser Broadcast Receiver sind so konfiguriert, dass man ohne ausreichenden Schutz auf sie zugreifen kann. Das ist dadurch möglich, dass *android:exported=true* gesetzt ist. Dies ermöglicht es anderen Apps oder Prozessen auf dem Gerät, diese Receiver anzusprechen. Dadurch kann es potenziell zu unautorisierten Zugriffen und Datenmanipulation kommen.

In der Manifest-Datei der App kann außerdem ein Intent-Filter mit ungewöhnlich hoher Priorität (*android:priority="996"*) identifiziert werden. Durch die hohe Priorität wird sichergestellt, dass die App bei bestimmten Ereignissen vor anderen installierten Anwendungen reagiert. Diese Technik kann dazu verwendet werden, systemweite Broadcasts abzufangen oder zu manipulieren. Derartige Konfigurationen sind insbesondere bei schadhaften Apps bekannt, die gezielt andere Anwendungen übersteuern oder unterdrücken möchten.

Bei der Codeanalyse werden mehrere kritische Schwachstellen erkannt. So enthalten die Dateien hartkodierte sensible Informationen wie API-Schlüssel und Token. Die App nutzt außerdem direkte SQL-Abfragen. Dies ermöglicht potenziell SQL-Injection-Angriffe. Durch manipulierte Eingaben kann es so möglicherweise zu Datenmanipulation und Datenverlust kommen. Des Weiteren wird festgestellt, dass das Debugging aktiviert ist. Das heißt, *BuildConfig.DEBUG = true* ist gesetzt. Dadurch befindet sich die App in einem Entwicklungsmodus, bei dem Debugging-Funktionen wie Log-Ausgaben, Entwicklerzugänge oder die Anbindung externer Debugger aktiv sind. Dies kann dazu führen, dass sensible Informationen im Log ausgegeben werden und Schutzmechanismen wie Code-Verschleierung oder Sicherheitsprüfungen deaktiviert sind. Das macht die App deutlich anfälliger für Reverse Engineering und Angriffe. In diesem Zusammenhang ist besonders kritisch, dass Klassen wie *AppCenterLog* und *ErrorLogHelper* aktiv Informationen in Logfiles schreiben. Laut dem Mobile Application Security Verification Standard (MASVS) vom Open Web Application Security Project (OWASP) gelten solche Log-Ausgaben als potenziell unsicher, da sie, insbesondere im Debug-Modus, vertrauliche Daten wie Token, API-Endpunkte oder Nutzerdetails preisgeben können. Zusätzlich fällt bei der Codeanalyse auf, dass die App einen unsicheren Zufallszahlengenerator in der Klasse *HttpClientRetryer* benutzt. Auch lässt sich die Implementierung eines SSL-Zertifikat-Pinning erkennen. Dies erschwert MITM-Angriffe erheblich.

Bei der statischen Analyse wird weiterhin eine E-Mail-Adresse identifiziert. Diese lautet: service@bosch-smarthome.com. Bei dieser handelt es sich um die offizielle E-Mail-Adresse von Bosch, die bei den Playstore-Informationen als E-Mail-Adresse des Entwicklers angegeben ist.

Die App kommuniziert mit mehreren Drittanbieterdiensten, u.a.:

- bosch-smart-cameras.firebaseio.com (Firebase)
- in.appcenter.ms (Microsoft App Center)
- boschcx.qualtrics.com (Umfragen)
- dashboard.videoexpertsgroup.com (Medieninfrastruktur)

Diese Domains befinden sich überwiegend in den USA und in Deutschland. Abschließend werden bei der Analyse der App mit *MobSF* zwei Tracker identifiziert, die in der App implementiert sind. Bei diesen handelt es sich um *Google Firebase Analytics* und *Microsoft Visual Studio App Center Crashes*. *Google Firebase Analytics* dient dazu, das Nutzerverhalten zu analysieren und Muster zu erkennen. *Microsoft Visual Studio App Center Crashes* erstellt automatisch einen Absturzbericht, der auf dem Gerät gespeichert und bei einem Neustart der App an den Entwickler gesendet wird. [48, 49]

6.5.2 Ergebnisse der dynamischen Analyse der Bosch Smart Camera App 2.8.0

Die dynamische Analyse der App mit *MobSF* liefert aufschlussreiche Erkenntnisse über die Netzwerkkommunikation, den Umgang mit Benutzerdaten, die Speicherung in Datenbanken, WebView-Interaktionen sowie eingesetzte Verschlüsselungstechniken. Diese Erkenntnisse sind zentral, um das Laufzeitverhalten der App besser zu verstehen und potenziell relevante

Informationen für weiterführende Untersuchungen zu identifizieren. Im Folgenden werden die wichtigsten Ergebnisse der einzelnen Test-Szenarien sowie deren Interpretation dargestellt.

Im Rahmen des ersten Analyseszenarios wird der Fokus auf die forensisch relevante lokale Datenspeicherung und die potenzielle Extraktion der Daten auf dem Gerät gelegt. Die Kombination aus statischer und dynamischer Analyse, Frida-Hooks, API-Monitoring sowie manueller Auswertung der Dateisystemstruktur führt zu umfassenden Erkenntnisse über das Datenverhalten der Bosch Smart Camera App.

Zunächst offenbart die Dateisystemanalyse, dass mehrere Datenbanken im App-Verzeichnis zwar initialisiert werden, jedoch zum Zeitpunkt der Analyse leer bleiben, so zum Beispiel *Database.db*, *WebData*, *QuotaManager*, *google_app_measurement_local.db* und *WorkSpec*. Dennoch lässt allein deren Existenz Rückschlüsse auf die Architektur der App zu: Diese Datenbanken dienen vermutlich der Zwischenpufferung von Telemetrie- oder Konfigurationsdaten, die jedoch im laufenden Betrieb dynamisch nachgeladen und bei Bedarf synchronisiert werden.

Die Datei *androidx.work.workdb* enthält hingegen bereits systemrelevante Einträge wie zum Beispiel eine mit einem Zeitstempel versehene Preference-Zeile (*last_force_stop_ms*) sowie strukturierte, aber unbefüllte Tabellen zur Verwaltung geplanter Hintergrundprozesse (*WorkSpec*, *WorkName*, *SystemIdInfo*). Diese Struktur deutet auf eine Nutzung des *Android WorkManager* hin, mit dem geplante Tasks wie zum Beispiel Datenübertragungen oder Sync-Aufgaben gesteuert werden.

Signifikante Funde zeigen sich auch bei der Auswertung der *SharedPreferences*-Dateien. So können sensible Konfigurationsdaten wie die Installations-ID, Push-Token, ein verschlüsselter AES-Schlüssel (*NMFoundation-AES*) sowie Flags zur Nutzerinteraktion (*isAppRatingEnabled*, *AppFreshlyInstalled*) extrahiert werden. Besonders hervorzuheben ist auch der gespeicherte letzte Login unter der Adresse *rexxxxxxxxxxxxxxxxxxxxxx@gmail.com* (Angabe verfremdet), der als direkt personenbezogen gewertet werden muss. Weitere Informationen zu diesen XML-Dateien enthält auch Kapitel 6.5.3.

Während der dynamischen Analyse mit *Frida* können über das API-Monitoring mehrfach Zugriffe auf die *SharedPreferences* beobachtet werden. Abgefragte Schlüssel wie *measurement_enabled*, *app_backgrounded*, *allowedNetworkRequests* oder *RefreshTokenKeyData* zeugen von Telemetrie-Tracking und Zustandsspeicherung. Die Messdaten deuten auf eine lokale Vorhaltung der Analytics-Daten hin, die später synchronisiert werden. Dies wurde auch durch POST-Anfragen an https://in.appcenter.ms/logs bestätigt.

Darüber hinaus werden durch das *dump-intent*-Skript verschiedene Intents zwischen App-Komponenten protokolliert, so zum Beispiel zwischen *UserOnboardingView* und *OAuth2LoginView*. Die übergebenen Extras enthalten strukturierte ViewModel-Daten (*MvxLaunchData*). Diese stellen potenzielle Quellen für das Extrahieren von Nutzernavigation, UI-Pfaden und Authentifizierungsflüssen dar.

Einige temporäre .dex-Dateien werden ebenfalls bei App-Start im *cache*-Verzeichnis erstellt, wie zum Beispiel */frida780357356382432642.dex*, was auf erfolgreiche Hook-Injektionen mit *Frida* hinweist. Zusätzlich werden systemnahe Verzeichnisse wie */system/lib64*, */system/vendor/lib64* oder */data/local/tmp* referenziert und verwendet, insbesondere zum Laden der Frida-Server-Binaries und zur Manipulation nativer Bibliotheken.

Der Frida-Out-Stream liefert ergänzend Erkenntnisse über lesende Dateizugriffe wie zum Beispiel auf */WebViewChromiumPrefs.xml*. Dies lässt auf persistente *WebView*-Nutzung schließen, etwa zur Darstellung dynamischer Inhalte oder Remote-UI-Komponenten, bei denen auch lokal gespeicherte Konfigurationsdaten zum Einsatz kommen. Des Weiteren kann bei der Screen-Shot Erstellung das Anlegen eines Unterverzeichnisses in */data/user/0/com.bosch.cbs.consumer.prod/files* festgestellt werden. Das Unterverzeichnis hat die Bezeichnung *temp* und enthält nach der Erstellung des Screen-Shots eine Joint Photographic Experts Group (JPEG)-Datei die als Bezeichnung eine Universally Unique Identifier (UUID) erhält. Im konkreten Fall ist die Datei *2d5d8080-a4a1-40fd-a77d-3b71798b64fb.jpg* .

Abschließend bleibt festzuhalten, dass auch wenn manche Datenbanken zur Analysezeit leer sind, durch das Zusammenspiel von Logcat-Überwachung, Intent-Dumping , API-Monitoring und Frida-Out-Stream ein detailliertes Bild des lokalen Datenverhaltens gezeichnet werden kann. Die App speichert sowohl Konfigurations- als auch Telemetriedaten lokal, nutzt temporäre Dateierstellung und legt Benutzerpräferenzen sowie Identifikatoren in *SharedPreferences* ab. Diese Speicherorte sind insbesondere für forensische Analysen und Datenschutzbewertungen von zentraler Bedeutung.

Im zweiten Szenario können relevante Erkenntnisse über den Netzwerkverkehr und die API-Interaktionen der Bosch Smart Camera App gewonnen werden. Zunächst zeigt sich, dass beim Start der App erneut temporäre .dex-Dateien im app-spezifischen Cache-Verzeichnis erstellt werden, wie zum Beispiel `frida136851101104075872.dex`. Diese Artefakte deuten erneut auf eine erfolgreiche Code-Injektion durch Frida-Skripte hin, etwa zur Umgehung von SSL-Zertifikat-Pinning, Debugger-Checks und Root-Erkennung. Auch systemnahe Verzeichnisse wie `/system/lib64` oder `/system/vendor/lib64` werden im Rahmen der Frida-Hooks genutzt. Dies weist auf tiefergehende Interaktionen mit der Android-Systemebene hin.

Die Anwendung zeigt in mehreren Fällen Verbindungsversuche zu bekannten Cloud- und Authentifizierungsdiensten. Dazu zählen unter anderem Server von hcaptcha.com, android.googleapis.com, in.appcenter.ms und singlekey-id.com. Die TLS/SSL-Analyse mit *MobSF* ergibt, dass die App keine klartextbasierten HTTP-Verbindungen nutzt und grundlegende TLS-Konfigurationen korrekt implementiert sind. Obwohl *MobSF* keine aktiven SSL-Zertifikat-Pinning-Mechanismen erkennen konnte, zeigten manuelle Tests während der Analyse, dass Zertifikatsfehler durch die App erkannt und die Verbindung verweigert wird. Dies weist eindeutig auf eine aktives, aber nicht standardisiert implementiertes SSL-Zertifikat-Pinning hin.

Darüber hinaus können durch das *dump-intent*-Skript mehrfach Intents zwischen internen Komponenten der App abgefangen werden, etwa bei der Navigation zwischen *SplashView*, *UserOnboardingView* und *OAuth2LoginView*. Die übergebenen Metadaten (*MvxLaunchData*) belegen, dass die App ein komponentenbasiertes Model View ViewModel (MVVM)-

Navigationsmodell verwendet. Dabei werden sicherheitsrelevante Informationen über die geladenen ViewModel-Typen und deren Parameter mitgeführt. Der Zugriff auf die *OAuth2LoginView* legt zudem nahe, dass ein Authentifizierungsprozess durchlaufen wird.

Es ist möglich, das SSL-Zertifikat-Pinning der App erfolgreich zu umgehen und verschiedene Netzwerkverbindungen über den Proxy zu beobachten. Ein Teil des Datenverkehrs kann mit Hilfe des „HTTP(S) Traffic"-Logs und des „HTTPTools" genauer analysiert werden. Dabei ist jedoch eine Reihe von Servern, die bei der Analyse des Netzwerkverkehrs mit Wireshark identifiziert wurden, nicht sichtbar. Insgesamt kann nur von neun Servern der Datenverkehr erfasst werden. Diese sind:

- https://singlekey-id.com
- https://in.appcenter.ms
- https://cdn.singlekey-id.com
- https://js.hcaptcha.com
- https://newassets.hcaptcha.com
- https://api.hcaptcha.com
- https://smarthome.authz.bosch.com
- https://prod-sys01-adaptable-theme.authz.bosch.com
- https://36.mcg.escrypt.com

Da die App bei der Analyse des Netzwerkverkehrs in Kapitel 5.5.2 bereits gestartet war, konnte die Kommunikation mit dem *Microsoft App Center* (https://in.appcenter.ms) dort nicht erfasst werden. Bei der dynamischen Analyse ist der Appstart jedoch Teil der Untersuchung. Deshalb kann dieser Datenverkehr nun aufgezeichnet werden. Bei https://36.mcg.escrypt.com handelt es sich um eine Subdomaine des Dienstes mcg.escrypt.com, der auch bei der Analyse des Netzwerkverkehrs mit *Wireshark* erkennbar ist.

Der gesamte mit Hilfe des „HTTP(S)-Proxys" aufgezeichnet Datenverkehr steht mit dem Appstart- und Anmeldeprozess in Verbindung. So können auch nur Aussagen über diesen getroffen werden. Durch den Einsatz des „HTTP(S)-Proxys" von *MobSF* ist es möglich, die Payloads und Header unverschlüsselt einzusehen. Damit ist es möglich, einige bisher unbekannte relevante Daten zu sammeln und Annahmen aus Kapitel 5.5.2 zu belegen.

Aus den Headern lassen sich hierbei einige forensisch relevante Informationen zu den Servern mit denen die App kommuniziert gewinnen. So sieht man zum Beispiel im Anhang A.3 den Zeitstempel der Kommunikation, die Cookies, die der Server setzt und die Konfigurationen der Sicherheitsmechanismen. Die Sicherheitsmechanismen, die im konkreten Fall angegeben sind, sollen vor allem gängige Webangriffe wie MITM, Cross-Site-Scripting (XSS), Clickjacking oder Cross-Site-Request-Forgery (CSRF) verhindern. Beim XSS versucht ein Angreifer, schädlichen JavaScript-Code in eine Webseite einzuschleusen, der im Browser eines anderen Nutzers ausgeführt wird. So kann zum Beispiel versucht werden, Session-Cookies zu stehlen. Dies wird von dem Server unter anderem durch das Setzen des Cookies *httponly* verhindert. Beim Clickjacking wird versucht, bei einer bösartigen Seite die echte Seite in einem unsichtbaren Frame einzubinden und so den Nutzer zu täuschen, damit er unbeabsichtigt auf Buttons klickt. Dies wird vom Server durch die Einstellung von *x-frame-options: SAMEORIGIN* verhindert. Beim CSRF versucht der Angreifer einen eingeloggten Nutzer dazu zu bringen,

dass er ungewollte Aktionen auf einer Webseite auszuführt. Im Header lässt sich zum Schutz vor solchen Angriffen vor allem das Cookie-Attribut *SameSite=Lax* identifizieren, weil es das automatische Mitsenden von Cookies bei fremden Seiten blockiert. Zusätzlich helfen *Secure*, *HttpOnly* und *no-store*, die Session-Integrität abzusichern. [20, S. 160f. 50, S. 101, 106]

Beim Appstart wird, wie bereits erwähnt, eine HTTPS-POST-Anfrage an *https://in.appcenter.ms/logs?api-version=1.0.0* gesendet. Dabei wird eine Reihe von App- und Geräteinformationen im JSON-Format an den Server übermittelt. Im Anhang A.1 befindet sich der „HTTP(S) Traffic“-Log-Ausschnitt der entsprechenden POST-Anfrage und die Antwort des Servers. Dort finden sich forensisch relevante Informationen, darunter das App-Secret. Dieses wird vom Microsoft App Center genutzt, um eine App eindeutig zu identifizieren, die Kommunikation zwischen App und Server zu authentifizieren und vor Manipulation zu schützen. Es hat eine ähnliche Funktion wie ein API-Schlüssel. Zudem werden weitere Informationen zum Gerät und zur App an den Server übertragen.

Durch das Sichtbarmachen des Datenverkehrs ist es zudem möglich, einen tieferen Einblick in den Authentifizierungsprozess mit https://singlekey-id.com und https://smarthome.authz.bosch.com zu gewinnen. Damit kann man den Anmeldeprozess genauer nachvollziehen. Während des Anmeldeprozesses sind sämtliche Daten im Klartext einzusehen. Dem Datenverkehr können so verschiedene Token, Cookies und Anmeldedaten (E-Mail-Adresse und Passwort) entnommen werden. Im Anhang A.3 und Anhang A.2 sind die Auszüge aus dem „HTTP(S) Traffic“-Log mit dem Passwort und der E-Mail-Adresse zu sehen. Da dieses Benutzerkonto weiterhin genutzt werden soll, werden die genauen Angaben verfremdet. Auch die Ressourcen, die von den Servern bereitgestellt werden, sind im Klartext einzusehen. Dazu gehören Hypertext Markup Language (HTML)-Dateien, die die Struktur und Inhalte der entsprechenden Seiten definieren. Weiterhin werden verschiedene JavaScripte sowie JavaScript-Modul-Dateien erkannt. Diese ermöglichen es, Webseiten dynamisch zu gestalten und auf Benutzerinteraktionen zu reagieren. Auch die Übermittlung von unterschiedlichen Bilddateien in verschiedenen Formaten kann im Klartext nachvollzogen werden. Des Weiteren sind eine Reihe von Web Open Font Format 2.0 (WOFF2)-Dateien im Datenverkehr erkennbar. Diese enthalten komprimierte Schriftarten und sind speziell für Web-Anwendungen gedacht. Zudem finden sich Cascading Style Sheets (CSS)-Dateien. Diese sind für das Layout von Webseiten verantwortlich und werden benötigt, um dem Nutzer eine grafische Oberfläche zu bieten und Inhalte anzuzeigen.

Die Bosch Smart Camera App nutzt für die Anmeldung einen Open Authorization (OAuth) 2.0 Authorization Code Flow mit Proof Key for Code Exchange (PKCE). Die Authentifizierung erfolgt dabei über SingleKey ID. Dieser Dienst dient ausschließlich als Identitätsanbieter. Für die Zugriffsvergabe nutzt die Bosch Smart Camera App *Keycloak* als Autorisierungsserver. Der Autorisierungsserver ist hierbei über die Domaine https://smarthome.authz.bosch.com und SingleKey ID über https://singlekey-id.com erreichbar. Der eigentliche Anmeldeprozess läuft wie folgt ab: Nach einer initialen Authentifizierungsanfrage an den Authentifizierungsserver wird der Nutzer zum Identitätsanbieter SingleKey ID weitergeleitet. Dort meldet sich der Nutzer mit seinen Anmeldedaten an und erhält einen Autorisationscode. Dieser wird zusammen mit dem *code_verifier* an den Autorisierungsserver gesendet. Der Autorisierungsserver überprüft dann die übersendeten Daten auf Richtigkeit und erstellt nach erfolgreicher Prüfung einen JSON Web Token (JWT)-Access-Token. Dieser besteht aus drei Teilen, die durch einen

Punkt getrennt sind und bestätigt die Identität des Nutzers. Die Token werden von Bosch als Cookies gespeichert. JWT-Token können unter anderem mit dem Onlinetool JWT Debugger, das unter https://jwt.io/ erreichbar ist, decodiert werden. Dies liefert genaue Informationen über die Identität des Nutzers und Eigenschaften des Tokens wie zum Beispiel Erstellungszeit und Ablaufzeit.

Neben dieser Anmeldung, die sowohl im „HTTP(S) Traffic"-Log als auch im „HTTPTools" enthalten ist, sind im „HTTPTools" mehrere GET-Anfragen erkennbar. Diese Anfragen dienen der Überprüfung der Gültigkeit eines Zertifikats anhand seiner ID. Bei der Analyse des vom Proxy aufgezeichneten Datenverkehrs sind jedoch keine Antworten auf diese Anfragen feststellbar.

Während des Anmeldeprozesses kann im Klartext nachvollzogen werden, wie mit Hilfe von hCAPTCHA ein Bot-Schutz erfolgt. Für diesen werden von den Domains https://js.hcaptcha.com und https://newassets.hcaptcha.com mit Hilfe von GET-Anfragen verschiedene Ressourcen bezogen. Zudem werden durch POST-Anfragen Daten an https://api.hcaptcha.com gesendet.

Die beiden Domaines https://cdn.singlekey-id.com und https://prod-sys01-adaptable-theme.authz.bosch.com liefern ausschließlich statische Ressourcen, die vollständig im Klartext einzusehen sind.

Auch das *crypto-aes-key*-Skript wird ausgeführt, wobei in den analysierten Datenströmen sensible Klartextinformationen identifiziert werden. Die Verwendung kryptografischer Klassen oder Methoden wie *Cipher* und *MessageDigest* kann ebenfalls nachvollzogen werden.

Die Analyse der UI- und Benutzerinteraktionen zeigt, dass über das UI sowohl direkte Nutzerdaten wie Anzeigenamen und Gerätezustände als auch Verhaltensdaten, unter anderem Navigationspfade und Nutzereingaben, gewonnen werden können. Die Kombination aus Screenshots, Textinhalten und aufgezeichneten App-Intents ermöglicht eine detaillierte Rekonstruktion des Nutzerverhaltens innerhalb der App. Insbesondere durch das Entfernen des Screenshot-Schutzes können sensible Bereiche dokumentiert werden, die sonst nicht zugänglich wären.

Im Rahmen des vierten Szenarios wird die Bosch Smart Camera App einer dynamischen Analyse unterzogen, um die verwendeten kryptografischen Algorithmen sowie die Schlüsselverwaltungspraktiken zu untersuchen. Hierfür kommen verschiedene Frida-Skripte zum Einsatz, darunter *crypto-dump-keystore*, *crypto-trace-cipher*, *crypto-aes-key* und *crypto-trace-keystore*. Ergänzt wird die Analyse durch ein API-Monitoring sowie ein SSL-Zertifikat-Pinning-Umgehungsskript.

Die Analyse zeigt, dass die App sowohl symmetrische als auch asymmetrische Verschlüsselung verwendet. Für die symmetrische Verschlüsselung wird der Algorithmus AES im Cipher Block Chaining (CBC)-Modus mit *PKCS5Padding* eingesetzt. Dieser Modus verarbeitet Daten blockweise, wobei jeder Klartextblock vor der Verschlüsselung mit dem vorherigen Chiffreblock verknüpft wird. Dies verhindert Wiederholungen im Chiffretext und erhöht die Sicherheit, sofern ein Initialisierungsvektor (IV) verwendet wird. In der App wird der IV mit Hilfe

von *SecureRandom* dynamisch generiert, was eine sichere und zufällige Erzeugung gewährleistet. Das Padding erfolgt nach dem PKCS#5-Standard, welcher Daten auf die Blockgröße auffüllt, um vollständige AES-Blöcke zu gewährleisten. [14, S. 137-139, 51, S. 10f.]

Zur asymmetrischen Verschlüsselung wird der Algorithmus RSA mit dem Modus Electronic Codebook (ECB) und PKCS1Padding verwendet. RSA dient hier vorrangig zur Verschlüsselung kleiner Datenmengen, typischerweise von Schlüsseln oder Token. Der verwendete Padding-Mechanismus (PKCS#1 v1.5) ist weit verbreitet, gilt jedoch mittlerweile als veraltet, da er gegen bestimmte Angriffe, wie zum Beispiel Chosen-Ciphertext-Angriffe, anfällig sein kann. Eine modernere Alternative wäre hier RSA mit Optimal Asymmetric Encryption Padding (OAEP). [14, S. 162f. 51, S. 23-30]

Hinsichtlich der Schlüsselverwaltung nutzt die App den Android Keystore, wie durch wiederholte Zugriffe auf den Alias *NMFoundation-RSA* deutlich wird. Der private RSA-Schlüssel wird im Keystore gehalten und nicht exportiert, was eine sichere Praxis darstellt, da das Schlüsselmaterial hardwarebasiert geschützt wird. Allerdings werden die AES-Schlüssel nicht im Keystore gespeichert, sondern mit Hilfe von *SecretKeySpec* im Arbeitsspeicher verarbeitet. Dies stellt ein potenzielles Risiko dar, da der Schlüssel theoretisch durch Speicheranalysen extrahiert werden könnte.

Zusätzlich zeigt die Analyse, dass die App zwar verschiedene Sicherheitsmechanismen wie SSL-Zertifikat-Pinning und Anti-Debugging-Checks implementiert hat, diese jedoch durch *Frida* erfolgreich umgangen werden können. Das SSL-Zertifikat-Pinning kann unter anderem durch das Skript *ssl-pinning-bypass* neutralisiert werden.

Des Weiteren werden TLS/SSL-Sicherheitstests durchgeführt, um die Netzwerksicherheit der App zu analysieren. Die App verwendet ausschließlich verschlüsselte Verbindungen und zeigt keine TLS-Misskonfigurationen. Besonders relevant ist, dass die App vermutlich über eine eigene Implementierung von SSL-Zertifikat-Pinning verfügt, welche allerdings von *MobSF* nicht automatisch erkannt wird, vermutlich weil sie nicht auf einem bekannten Framework basiert. Zwei Beispiele für solche Frameworks wären *okhttp.CertificatePinner* oder *TrustKit*. In der Praxis blockiert die App tatsächlich Netzwerkverbindungen, wenn Zertifikate manipuliert sind, sofern kein Frida-Bypass-Skript verwendet wird. Dies zeigt, dass das SSL-Zertifikat-Pinning funktional, jedoch nicht gegen Frida-Skripte geschützt ist. Ein effektiver Schutz gegen Hooking-Frameworks wäre notwendig, um dieses Sicherheitsmerkmal robust zu machen.

Im fünften Analyseszenario wird versucht, die implementierten Sicherheitsmechanismen zu umgehen. Durch die eingesetzten Frida-Skripte können sämtliche getesteten Sicherheitsmechanismen wie SSL-Zertifikat-Pinning, Debugger-Erkennung und Emulator-Erkennung erfolgreich umgangen werden. Es ist jedoch nicht möglich, dadurch weitere zusätzliche Informationen zu gewinnen.

6.5.3 Ergebnisse der Untersuchung der Daten mit App-Bezug

Die Inhalte der Verzeichnisse des emulierten Google Pixel 5 und des Google Pixel 7a unterscheiden sich in den forensisch relevanten Bereichen nicht wesentlich. Es existieren jedoch kleine Unterschiede. Diese werden an entsprechender Stelle aufgezeigt. Wie bereits dargestellt, haben die nicht angelegten Pfade im *storage*- und *mnt*-Verzeichnis bei der forensischen Untersuchung keine Relevanz. Deshalb wird sich bei der Darstellung der Ergebnisse der Untersuchung der Daten mit App-Bezug auf die Daten des Google Pixel 7a beschränkt und die Unterschiede zum emulierten Gerät in den anderen Verzeichnissen nur kurz aufgezeigt. Des Weiteren wird ausschließlich auf die Verzeichnisse und Dateien eingegangen, die tatsächlich forensisch relevante Informationen enthalten.

Zunächst wird das *app_webview*-Verzeichnis untersucht. Das Hauptaugenmerk liegt dabei auf den Log-Dateien im *Default*-Verzeichnis sowie den Dateien *last-exit-info* und *pref_store*. Hierbei kann festgestellt werden, dass auf dem AVD keine Datei *last-exit-info* existiert. Im *Default*-Verzeichnis können mit Hilfe des *DB Browsers for SQLite* sicherheitsrelevante Informationen aus der *Cookies*-Datei gewonnen werden. In dieser Datei sind Cookies gespeichert. Die Cookies *AUTH_SESSION_ID*, *KEYCLOAK_IDENTITY*, *AF*, *KEYCLOAK_SESSION* und *ID* stehen mit den Domains singlekey-id.com und smarthome.authz.bosch.com in Verbindung und belegen, dass eine Authentifizierung über diese Dienste erfolgte. Auch in den Log-Dateien in `Default/LocalStorage` finden sich Hinweise zur Verwendung von singlekey-id.com zur Authentifizierung.

Auch in der Datei *last-exit-info* können relevante Informationen festgestellt werden. Diese Datei beinhaltet Exit-Informationen. Sie beschreiben den Zustand der App zum Zeitpunkt des Schließens. Der laufende Prozess hatte dabei die Prozess-ID *18388*. Die Datei enthält den Zeitstempel als Unix-Zeitstempel in Millisekunden. Dieser ist *1744703900178*. Das bedeutet, die App wurde am *15.04.2025* um *9:58:20* Uhr geschlossen. Außerdem wird der Status der App als Status *0* angegeben. Dies deutet auf ein reguläres Beenden der App hin.

Die Datei `pref_store` enthält Angaben zu Geräteinfos, WebView-Sicherheitseinstellungen, Informationen zur Telemetrie von Nutzung und Stabilität der App, den Zeitpunkt der Appinstallation, eine anonyme Benutzer-ID, unverschickte Logdaten und weitere Informationen. Der Eintrag `user_experience_metrics.client_id2`, enthält eine eindeutige UUID (*6567xxxx-xxxx-xxxx-xxxx-xxxxxxxx00fe*; verfremdet). Diese erlaubt es, eine App-Instanz oder ein Nutzungsprofil eindeutig zu identifizieren. Die Zeitstempel `client_id_timestamp`, `last_seen.BrowserMetrics` und `installation_date2` (Unix: *1730202650*) erlauben es, eindeutig zu rekonstruieren, wann die App installiert und genutzt wurde.

Im Verzeichnis `files` befinden sich die Dateien *cameras.json* und `PersistedInstallation.W0RFRkFVTFRd+MTo0MDQ2MzA0MjQ0MDU6YW5kcm9pZDo5ZTViNmI1OGU0YzcwMDc1.json`, die forensisch relevante Informationen enthalten. Diese sind in den nachfolgenden Tabellen dargestellt.

Die Tabelle 6.1 zeigt die forensisch relevanten Informationen aus der Datei *camaeras.json*. Diese enthält Geräteinformationen und Informationen zu Kameraeinstellungen.

Tabelle 6.1: Forensisch relevante Informationen aus der Datei *cameras.json*

Eindeutige Kamera-ID	9DE5xxxx-xxxx-xxxx-xxxx-xxxxxxxx704B
Art der Kamera	HOME_Eyes_Indoor
Verwendete Zeitzone	Europe/Berlin
Aktivitätszustand der Kamera	privacyMode:"0FF"; Kamera aktiv
Besitzername	null
Kamera geteilt mit mir :	null; wurde nicht geteilt
Kamera geteilt mit Freunden	[]; leere Liste; wurde mit niemandem geteilt
Benachrichtigungen werden ausgelöst durch:	Bewegungen, Personen, Audio, Störungen
Verbindungsstatus	ONLINE

In Tabelle 6.2 sind die Informationen aus *PersistedInstallation.W0RFRkFVTFRd+MTo0MDQ2MzA0MjQ0MDU6YW5kcm9pZDo5ZTViNmI1OGU0YzcwMDc1.json* dargestellt. Diese Datei enthält wichtige Firebase-Informationen.

Tabelle 6.2: Forensisch relevante Informationen aus der Datei *PersistedInstallation.W0RFRkFVTFRd+MTo0MDQ2MzA0MjQ0MDU6YW5kcm9pZDo5ZTViNmI1OGU0YzcwMDc1.json*

Firebase Installations-ID	f9y6...ziiU
Status der Firebase Installation	3; erfolgreich eingerichtet und registriert
Firebase-Authentifikationstoken	eyJh...VCJ9.eyJh...wNX0.AB2L...R83w
Firebase-Refresh-Token	3_AS3...vChQ
Unix-Zeitstempel der Erstellung des Authentifizierungstokens	1744613226; 14.04.2025,8:47:06 Uhr Greenwich Mean Time (GMT)+0200
Gültigkeitsdauer des Authentifizierungstokens in Sekunden	604800

Abschließend wird das Verzeichnis *shared_prefs* analysiert. Aus den dort enthaltenen XML-Dateien können eine Reihe relevanter Informationen extrahiert werden. So enthält *AppCenter.xml* die Installations-ID sowie die Start- und Endzeiten der einzelnen App-Sitzungen als Unix-Zeitstempel. Außerdem wird ein Limit für die Größe der Crash-Speicher-Daten von 20 MB festgelegt. Aus der Datei *AwOriginVisitLoggerPrefs.xml* geht hervor, dass die Webseite am 15. April 2025 besucht wurde. Diese Datei existiert jedoch bei dem emulierten Gerät nicht.

Aus der Datei *com.bosch.cbs.consumer.prod_preferences.xml* lassen sich Informationen über die Konfiguration der App sowie Telemetrie- und Nutzermetriken der App gewinnen. Sie enthält außerdem Informationen zu App-Bewertungen. Der Umfang der auf dem physischen und dem emulierten Gerät gespeicherten Daten sowie deren Inhalt unterscheiden sich in einigen Punkten. Auf beiden Geräten kann man einsehen, welche Version der App gespeichert ist. Aktuell ist das die Version *2.8.0*. Auf dem physischen Gerät war die zuerst installierte Version die Version *2.6.0*. Die App wurde nicht neu installiert. Auf dem emulierten Gerät ist die Version *2.8.0* die zuerst installierte Version, was darauf zurückzuführen ist, dass die VM nur für diese Arbeit eingerichtet wurde. Deshalb ist die App auch als frisch installiert in der Datei gekennzeichnet. Beim Google Pixel 7a sind in der Datei mehr

Informationen zu Telemetrie- und Nutzermetriken hinterlegt. Außerdem enthält die Datei zusätzliche Informationen zur App-Bewertung. Das emulierte Gerät verfügt in diesen Bereichen über weniger Informationen. Jedoch sind in der Datei persönliche Informationen wie *rexxxxxxxxxxxxxxxxxxxxxx@gmail.com* (Angabe verfremdet) hinterlegt. Bei dieser E-Mail-Adresse handelt es sich um die E-Mail-Adresse, die bei der letzten Anmeldung verwendet wurde. Auch eine Migration auf längere RSA-Schlüssel geht aus der Datei hervor. Zum Sicherungszeitpunkt sind außerdem keine Live-Features wie Streams oder Alarme genutzt worden.

Aus der Datei *com.google.android.gms.appid.xml* kann der Firebase Push Token, der verwendet wird, um Nachrichten an das Gerät zu schicken, extrahiert werden. Des Weiteren geht aus der Datei hervor, dass die Erstregistrierung bei Firebase Cloud Messaging am *29.10.2024* um *12:50:24* Uhr GMT+0100 stattfand. Der aktuelle Token wurde am *14.04.2025* um *8:24:09* GMT+0200 erstellt.

Die Datei *measurement.prefs.xml* enthält Informationen zum Betriebssystem und zur Konfiguration der App in Bezug auf Tracking und Datenschutz. So beinhaltet die Datei den Zeitpunkt des ersten Appstarts. Dies war der *29. Oktober 2024 12:50:34* GMT+0100. Der Status des Health-Monitorings, des Datenmessungsservices und von Global Analytics ist ebenfalls in der Datei hinterlegt. Health-Monitoring wurde am *15.04.2025 09:26:16* GMT+0200 gestartet. Der Datenmessungsservice von Firebase Analytics wird genutzt. Global Analytics ist deaktiviert. Auch findet man in dieser Datei die genutzte Version des Betriebssystems- in diesem Fall Android *14*.

Die Datei *com.google.firebase.messaging.xml* enthält die Information, dass das Gerät Push-Nachrichten empfangen kann und dass verpasste bzw. verzögerte Benachrichtigungen auch später abgerufen werden können.

Die Datei *FirebaseHeartBeatW0RFRkFVTFRd+MTo0MDQ2MzA0MjQ0MDU6YW5kcm9pZDo5ZTViNmI1OGU0YzcwMDc1.xml* beinhaltet Informationen zum Gerät, zu verwendeten Firebase-Modulen und Android-Daten. Zudem sind dort eine Liste von Tagen, an denen die Module genutzt wurden, das Datum der letzten Nutzung sowie der Zeitpunkt des letzten Firebase Heartbeats hinterlegt. Ein solcher Firebase Heartbeat ist eine automatisch ausgelöste, regelmäßige technische Rückmeldung an Firebase- bzw. Google-Dienste. Diese kann somit nicht vom Nutzer unterbunden werden. Die Meldung erfolgte im konkreten Fall am *14. April 2025* um *08:47:05* GMT+0200.

Aus der Datei *WebViewChromiumPrefs.xml* geht hervor, dass die App WebView-Komponenten benutzt, und dafür einen separaten Ressourcenkontext verwendet.

Mit *shared_prefs* kann sich also ein Überblick über die Installation, Konfiguration und Nutzung der Bosch Smart Camera App verschafft werden. Bei allen Dateien, bei denen nicht genauer auf das Google Pixel 5 eingegangen wurde, entsprechen die forensisch relevanten Inhalte denen des Google Pixel 7. Abweichungen gibt es in Zeitstempeln, bei konkreten Token und der Angabe der Betriebssystemversionen. Dies ist darauf zurückzuführen, dass es sich um unterschiedliche Geräte mit unterschiedlichen Betriebssystemen und Nutzungszeitpunkten handelt.

Auch aus den Verzeichnissen, die nur auf dem emulierten Gerät vorhanden sind, können einige forensisch relevante Informationen gewonnen werden. So belegen die Pfade 7 und 8 (Quelltext 6.8), dass die App am *20.05.2025* und am *22.05.2025* aktiv war und Grafikoptionen ausgeführt hat. Dies geht aus den im Pfad enthaltenen Unix-Zeitstempeln *1747699200000* und *1747872000000* hervor. Diese Zeitstempel geben die bereits erwähnten Tage jeweils um *0.00* Uhr GMT+0 an. Beide Verzeichnisse weisen eine identische Unterverzeichnisstruktur auf. Die Verzeichnisse enthalten jeweils einen Unterordner *com.bosch.cbs.consumer.prod*. Der Verzeichnisname entspricht der Bezeichnung der App. In diesem Ordner befindet sich erneut ein Unterordner mit der Bezeichnung *1279*. Bei dieser Bezeichnung handelt es sich um das App Build der Bosch Smart Camera App. Dies geht aus den Erkenntnissen der dynamischen Analyse der App hervor (siehe auch Anhang A.1). In diesem Verzeichnis befindet sich eine Datei mit dem Namen *total*. Nähere Untersuchungen der Datei mit dem Hex-Editor zeigen das es sich vermutlich um eine Datei in einem binäre Protobuf-Format handelt. Da aber eine Recherche keine weiteren Informationen zum Aufbau der Datei ergab, können keine zusätzlichen Informationen aus der Datei gewonnen werden.

Das Verzeichnis 12 (Quelltext 6.8) enthält erneut ein Unterverzeichnis *1279*. In diesem befindet sich ein Unterverzeichnis, in dem ein Ordner mit der Bezeichnung `raw_traces` enthalten ist. Dieser beinhaltet eine Reihe von Dateien, deren Bezeichnung jeweils gleich aufgebaut ist. Der Name setzt sich aus dem Unix-Zeitstempel in Nanosekunden und dem Teil `.perfetto_trace.pb` zusammen. Die Dateiendung `.pb` weist auf binäre Dateien im Protobuf-Format hin. Im konkreten Fall wurden diese, wie sich aus dem Dateinamen ableiten lässt, vermutlich mit dem Open-Source-Profiling-Tool *Perfetto* von Google erstellt. Dieses dient zur Analyse von Performance-Traces in Android-, Linux- und anderen Systemen. Die Dateien werden mit dem Tool *Perfetto UI* geöffnet. Mit diesem Tool kann analysiert werden, welche Systemaufrufe, Dateizugriffe und Input/Output (I/O)-Vorgänge beim Appstart stattfinden.

7 Zusammenfassung der Ergebnisse und forensische Bewertung

In diesem Kapitel werden die wesentlichen Ergebnisse der durchgeführten Analysen zusammengefasst, die bei der Untersuchung der Bosch Eyes Innenkamera II und der Bosch Smart Camera App festgestellt wurden. Des Weiteren erfolgt in diesem Kapitel eine Einschätzung der forensischen Relevanz der gewonnenen Daten und Informationen.

Die Netzwerkanalyse der Bosch Eyes Innenkamera II und der zugehörigen Bosch Smart Camera App beinhaltet sowohl Netzwerk- und Portscans als auch Paketmitschnitte und einen MITM-Angriff. Der Netzwerk- und Portscan identifiziert die IP-Adresse 10.0.1.131 und die MAC-Adresse 64:DA:A0:20:BB:07 der Kamera, welche eindeutig der Robert Bosch Smart Home GmbH zugeordnet werden können. Nur ein einziger Port, der Port 443, ist offen und bietet HTTPS-Kommunikation unter Nutzung des modernen Verschlüsselungsverfahrens TLSv1.3 mit der Cipher Suite *TLS_AES_256_GCM_SHA384* an. Die anderen Ports 22 (SSH-Dienst) und 2345 (vermutlich proprietärer Dienst) sind geschlossen. Der Webserver zeigt keine Benutzeroberfläche, sondern reagierte lediglich mit der Meldung „Object not available on this Webserver “.

Die Analyse des Netzwerkverkehrs zeigt, dass sowohl App als auch Kamera ausschließlich über externe Bosch-Cloud-Server kommunizieren. Eine direkte Verbindung innerhalb des lokalen Netzwerks kann in keinem der sechs untersuchten Nutzungsszenarien festgestellt werden. Die Steuerung der Kamera erfolgt ausschließlich über TLS-verschlüsselte Verbindungen zu Endpunkten, die der Robert Bosch Smart Home GmbH zugeordnet werden können (z.B. *proxy-19.live.cbs.boschsecurity.com* und *residential.cbs.boschsecurity.com*). Die Kamera und die App nutzen hierfür vorrangig bestehende TLS-Verbindungen, wodurch klassische Handshake-Pakete ausbleiben und Rückschlüsse auf den Inhalt der Kommunikation nur durch Zeitstempel und Transferverhalten möglich sind. Bestätigt wird auch der Einsatz von Google-Diensten und anderen Drittanbieterdiensten (z.B. hcaptcha.com, firebase.googleapis.com).

Bei der Untersuchung der aufgezeichneten Datenpakete zeigen sich in den verschiedenen Analyseszenarien typische Kommunikationsmuster: So erzeugt zum Beispiel ein Livestream parallele TLS-Verbindungen mit hoher Datentransferrate. Beim Upload des Screenshots in die Google-Cloud kann eine auffällig große Uploaddatenmenge festgestellt werden. Alarmereignisse werden durch auffällige TLS-Datenpakete der Kamera sowie parallele Kommunikationskanäle der App mit den Cloudservern begleitet. Alle Funktionen nutzen die Bosch-Infrastruktur. Des Weiteren belegen DNS-Anfragen, Zeitsynchronisation (NTP) und regelmäßige TLS-ACKs die andauernde Kommunikation.

Ein gezielter MITM-Angriff mittels *Burp Suite* ist nicht erfolgreich durchführbar. Zwar kann der Datenverkehr bei einer Google-Suche abgefangen werden, bei der Anmeldung in der Bosch Smart Camera App ist dies jedoch nicht möglich. Hier wird das Burp-Zertifikat explizit als nicht

vertrauenswürdig abgelehnt. Die Fehlermeldung „Untrusted connection“ deutet klar auf die Implementierung eines Zertifikat-Pinnings hin, wodurch der Verbindungsaufbau blockiert wird.

Aus den Ergebnissen der Netzwerkanalyse lassen sich also eine Reihe von forensisch relevanten Informationen gewinnen. Die forensische Relevanz wird in den folgenden Absätzen näher bewertet.

Die eindeutige IP- und MAC-Adresse ermöglichen eine eindeutige Identifikation der Kamera. Des Weiteren ist es durch die MAC-Adresse möglich, der Kamera die Bosch Smart Home GmbH als Hersteller zuzuordnen. Diese Informationen können wichtige Spuren in einem Ermittlungsverfahren darstellen. Sie können sowohl für die Netzwerkrekonstruktion als auch zur eindeutigen Identifikation des physischen Geräts genutzt werden.

Der vollständige Verzicht auf lokale Kommunikation und die ausschließliche Nutzung verschlüsselter Verbindungen über Bosch-Cloudserver liefern wichtige Informationen zur Architektur und zum Betriebsmodus des Systems. Forensisch relevant ist insbesondere die Trennung der Kommunikationskanäle zwischen App und Cloud sowie zwischen Cloud und Kamera. Durch die Analyse der Datenpakete lassen sich potenzielle Ereignisse wie Alarmmeldungen, Fernsteuerungsaktionen und Konfigurationsänderungen anhand von charakteristischen und wiederkehrenden Mustern im Netzwerkverkehr erkennen. Wichtige Indikatoren für die Erkennung der Ereignisse sind große TLS-Pakete, parallele Verbindungen sowie DNS/NTP-Anfragen und -Antworten. Dies ermöglicht zusammen mit den Zeitstempeln eine präzise zeitliche Einordnung von Nutzer- und Geräteaktionen.

Eine weitere forensisch relevante Information ist die Ablehnung des Burp-Zertifikats im Rahmen des simulierten MITM-Angriffs. Sie belegt die korrekte Implementierung eines Zertifikat-Pinnings als wirksamen Schutz gegen MITM-Angriffe.

Auch die Erkennung der eingebundenen Drittanbieter-Dienste ist forensisch relevant, da so nachvollzogen werden kann, wohin Nutzerdaten übertragen und wo sie gespeichert werden. Dies ermöglicht es, ausgelagerte Beweismittelquellen zu identifizieren. Auch können Kenntnisse über die eingebundenen Dienste für datenschutzrechtliche Bewertungen von Bedeutung sein.

Zusammenfassend kann zur forensischen Relevanz der Ergebnisse der Netzwerkanalyse festgestellt werden, dass trotz vollständiger Verschlüsselung des Datenverkehrs wichtige Indizien für forensische Untersuchungen gesammelt werden können. Dabei sind vor allem die Metadaten wie Zeitstempel, Quell-/Zieladressen, Protokolleigenschaften und Kommunikationsdichte wichtige Informationsquellen. Diese ermöglichen Aussagen zur Kommunikation zwischen den Cloud-Servern, der Kamera und der App.

Die statische Analyse der Bosch Smart Camera App in der Version 2.8.0 mit Hilfe von *MobSF* offenbarte zahlreiche sicherheitsrelevante und forensisch bedeutsame Erkenntnisse. Zunächst können grundlegende Informationen zur Herkunft der App über den Playstore ermittelt werden, darunter der Paketname, die Robert Bosch GmbH als Entwickler sowie die digitale

Signatur. Dabei fällt auf, dass der verwendete Signaturalgorithmus *SHA1withRSA* potenziell unsicher ist, da SHA-1 als anfällig für Kollisionen gilt. Dies könnte die Integrität der APK verletzen.

Ein wesentlicher Bestandteil der Analyse ist die Auswertung der App-Berechtigungen. MobSF identifiziert über 40 Berechtigungen, darunter zahlreiche mit erhöhtem Risiko. Besonders kritisch sind Berechtigungen zum Zugriff auf Mikrofon, Kamera, Standortdaten, externe Speicherbereiche sowie persönliche Daten wie Kontakte und Nutzerinformationen. Zusätzlich nutzt die App Berechtigungen zur dauerhaften Hintergrundaktivität und zum Start bei Systemboot, was Hinweise auf eine hohe Persistenz liefert. Laut *MobSF* sind 14 von 25 Berechtigungen zu finden, die typischerweise bei Malware auftreten.

Weiterhin zeigt sich, dass die App Anti-VM- und Anti-Debugging-Techniken sowie Techniken der Code-Verschleierung, zum Beispiel r8-Compiler ohne Marker, nutzt. Diese Methoden erschweren sowohl die Sicherheitsprüfung als auch forensische Untersuchungen erheblich. Zudem gibt es auch Hinweise auf SSL-Zertifikat-Pinning.

Die Manifest-Datei zeigt mehrere sicherheitsrelevante Konfigurationen. Die App ist ab Android 8.0 lauffähig, was potenziell auch ältere, anfälligere Systeme betrifft. Der Parameter *allowBackup="true"* ermöglicht zudem ungesicherte Datenextraktion über ADB-Backup, was bei sicherheitsrelevanten Apps problematisch ist. Auch werden exportierte Broadcast Receiver ohne ausreichenden Zugriffsschutz verwendet. In Verbindung mit einem hoch priorisierten Intent-Filter, der die App bei bestimmten Ereignissen bevorzugt ausführt, ist dies als sicherheitskritisch zu bewerten.

Die Codeanalyse offenbart mehrere Schwachstellen. Neben hartkodierten API-Schlüsseln und Token werden ungesicherte SQL-Abfragen, ein aktivierter Debug-Modus und potenziell unsichere Logging-Funktionen identifiziert. Besonders kritisch ist die Nutzung eines unsicheren Zufallszahlengenerators sowie die Protokollierung sensibler Daten durch Logging-Klassen. Gleichzeitig weist die App eine Kommunikation mit mehreren Drittanbieterdiensten, darunter Firebase, Microsoft App Center und Qualtrics auf. Außerdem sind zwei Tracking-Module integriert, die zur Analyse des Nutzerverhaltens und zur Fehlerberichterstattung dienen.

In forensischen Untersuchungen lassen sich die durch die statische Analyse gewonnenen Ergebnisse wie folgt nutzen. Die Analyse von Paketname, Signatur der App und Zertifikatsinformationen bietet die Möglichkeit zur eindeutigen Identifikation der App und ihrer Authentizität sowie Hinweise zur Herkunft der Anwendung. Es ist außerdem möglich, unveränderte Installationen von manipulierten Versionen zu unterscheiden.

Durch die Analyse der Berechtigungen erhält man einen Einblick in die Zugriffsmöglichkeiten der App. So können potenzielle Missbrauchsmöglichkeiten eines Angreifers identifiziert werden. Dies kann einen Einstieg für weitere Analysen bieten.

Auch die Nutzung von Anti-VM-, Anti-Debugging-Maßnahmen, Zertifikat-Pinning und Code-Verschleierung sind wichtige Informationen. So beweisen diese eine bewusste Absicherung gegen externe Analysen. Dies muss bei der Auswahl der forensischen Untersuchungsme-

thodiken berücksichtigt werden. So können gängige Tools wie Emulatoren und MITM-Proxys ohne Modifikationen kaum genutzt werden. Dies muss bei der Beweissicherung und der darauffolgenden Untersuchung zwingend beachtet werden.

Des Weiteren sind die identifizierten Schwachstellen von forensischer Bedeutung. So stellen die veraltete Signaturmethode, die hartcodierten Informationen, die ungesicherten SQL-Abfragen und der aktivierte Debug-Modus potenzielle Angriffsvektoren dar.

Neben diesen Schwachstellen zeigt die aktivierte Backup-Funktion, dass sensible Daten auch ohne Rootzugriff gesichert werden können.

Auch bei der statischen Analyse der App fällt die Kommunikation mit verschiedenen Drittanbietern auf. Deren forensische Bedeutung wurde bereits erläutert.

Die dynamische Analyse der Bosch Smart Camera App 2.8.0 ermöglicht mit Hilfe von *MobSF*, Frida-Hooks, API-Monitoring und Logcat-Überwachung einen tiefen Einblick in das Laufzeitverhalten der App. Im Fokus stehen die Verarbeitung von Nutzerdaten, die Netzwerkkommunikation und API-Interaktionen, die UI- und Benutzerinteraktionen, die Kryptografie und Schlüsselverwaltung sowie die Umgehung von Sicherheitsmechanismen.

Zunächst wird deutlich, dass die App trotz leerer Datenbanktabellen bereits initiale Strukturen für geplante Hintergrundprozesse anlegt, insbesondere über den Android WorkManager. In den *SharedPreferences* werden zahlreiche sensible Konfigurationsdaten gefunden, darunter Installations-IDs, verschlüsselte AES-Schlüssel, Push-Token und sogar ein gespeicherter Login mit personenbezogener E-Mail-Adresse. Die API-Überwachung zeigt, dass Telemetrie- und Analysedaten lokal gespeichert und später an Server wie *in.appcenter.ms* übermittelt werden.

Die Beobachtung interner App-Intents offenbart ein komponentenbasiertes MVVM-Modell. Dabei lassen sich Navigationspfade und Authentifizierungsprozesse rekonstruieren. Die gezielte Nutzung von *Frida* ermöglicht zudem die Umgehung von SSL-Zertifikat-Pinning, Debugger-Erkennung und Emulator-Prüfungen. Temporäre .dex-Dateien im Cache-Verzeichnis bestätigen erfolgreiche Code-Injektionen, unter anderem zur Analyse von Dateizugriffen auf lokale Konfigurationsdateien.

Im Rahmen der Netzwerkanalyse werden zahlreiche verschlüsselte Verbindungen zu bekannten Drittanbieter- und Authentifizierungsdiensten dokumentiert. Die App kommuniziert mit mehreren Servern. Dazu gehören unter anderem SingleKey ID, hCAPTCHA, Microsoft App Center und Escrypt. Besonders aufschlussreich ist die detaillierte Nachverfolgung des OAuth2-Anmeldeprozesses mit PKCE, bei dem JSON-Daten, HTML-Seiten, JavaScript-Dateien und sogar Token und Anmeldedaten im Klartext über den Proxy sichtbar gemacht werden können. Es kann jedoch ausschließlich der Datenverkehr des App-Start- und des Anmeldeprozesses sichtbar gemacht werden.

Die Analyse der Verschlüsselungstechniken zeigt, dass die App sowohl symmetrische Verschlüsselung in Form von AES im CBC-Modus mit PKCS5Padding als auch asymmetrische Verschlüsselung durch RSA mit PKCS1Padding nutzt. Während der RSA-Schlüssel sicher im Android Keystore gespeichert wird, verbleibt der AES-Schlüssel im Arbeitsspeicher. Die Aufbe-

wahrung des Schlüssels im Arbeitsspeicher stellt ein potenzielles Risiko bei Speicherangriffen dar. Es werden zwar verschlüsselte TLS-Verbindungen korrekt implementiert, dennoch ist es möglich, das funktionale SSL-Zertifikat-Pinning mit *Frida* zu umgehen.

Insgesamt kann durch die dynamische Analyse der App ein umfassendes Bild der Datenflüsse, Speicherorte und Sicherheitsmechanismen der App gewonnen werden einschließlich der UI-Interaktionen, Authentifizierungsdetails und Serverkommunikation.

Für forensische Untersuchungen bieten die Ergebnisse der dynamischen Analyse die folgenden Möglichkeiten.

Die Beobachtung interner App-Intents, ViewModel-Wechsel und UI-Zustände ermöglicht eine präzise Nachverfolgung von Nutzerinteraktionen, etwa der Anmeldung, Navigation oder Konfiguration. In Kombination mit entfernten Schutzmechanismen wie Screenshot-Sperren lassen sich selbst geschützte Inhalte dokumentieren. Damit ist es möglich, Nutzungsverhalten detailliert zu rekonstruieren.

Die Identifikation konkreter Domains und der Klartextdatenverkehr bei Anmeldeprozessen bieten wertvolle Informationen über den Authentifizierungsprozess, Serverrollen, Datenweitergabe und Schnittstellen zu Drittanbietern.

Die Verwendung gängiger Verschlüsselungsverfahren mit veralteten Padding-Methoden (RSA mit PKCS1Padding) sowie die unsichere Verarbeitung von AES-Schlüsseln im Arbeitsspeicher sind aus sicherheitstechnischer wie auch forensischer Perspektive relevant. Sie könnten unter bestimmten Bedingungen eine Extraktion durch Speicheranalyse ermöglichen.

Die implementierten Schutzmechanismen, wie SSL-Zertifikat-Pinning, Debugger- und Emulator-Erkennung zeigen, dass die App bewusst versucht, Analyse und Manipulation zu verhindern. Die erfolgreiche Umgehung dieser Maßnahmen durch *Frida* dokumentiert jedoch ihre technischen Grenzen. Dies hat direkte Relevanz für die Auswahl forensischer Werkzeuge und Methoden.

Auch können durch die dynamische Analyse zentrale Speicherorte und Artefakte identifiziert werden. Zudem kann ermittelt werden, wie die App auf diese zugreift.

Im Rahmen der forensischen Analyse der Bosch Smart Camera App 2.8.0 werden appbezogene Daten auf einem physischen Google Pixel 7a sowie einem emulierten Google Pixel 5 systematisch untersucht. Die Analysen konzentriern sich auf spezifische Verzeichnisse und Dateien mit forensischer Relevanz.

Im *app_webview*-Verzeichnis können Cookies mit direktem Bezug zu Authentifizierungsprozessen über singlekey-id.com und smarthome.authz.bosch.com identifiziert werden. Die *last-exit-info*-Datei dokumentiert detailliert den Zeitpunkt und Zustand beim Schließen der App, während *pref_store* unter anderem eine UUID, Telemetrie-Informationen, Installationszeitpunkte sowie Nutzerkennungen enthält. Die gespeicherten Zeitstempel ermöglichen eine genaue zeitliche Rekonstruktion der App-Nutzung.

Im *files*-Verzeichnis enthält die Datei *cameras.json* Informationen zur installierten Kamera (z.B. Kamera-ID, Typ, Status, Auslöser für Benachrichtigungen, Zeitzone). Die Datei *PersistedInstallation.W0RFRkFVTFRd+MTo0MDQ2MzA0MjQ0MDU6YW5kcm9pZDo5ZTViNmI1OGU0YzcwMDc1.json* liefert eine Firebase-Installations-ID, Authentifizierungs- und Refresh-Token sowie deren Gültigkeitsdauer und Erstellungszeitpunkt.

Im *shared_prefs*-Verzeichnis werden zahlreiche XML-Dateien mit weiterführenden Informationen entdeckt. Dazu zählen Telemetriedaten, Sitzungszeiten, Gerätezustände, App-Bewertungen, Firebase-Konfigurationen und Push-Token. Auch die Erfassung des ersten Appstarts, der Firebase Heartbeats sowie installierter Firebase-Module kann belegt werden. Besonders hervorzuheben ist die Speicherung der beim letzten Login verwendeten E-Mail-Adresse auf dem emulierten Gerät. Unterschiede zwischen physischem und emuliertem Gerät betreffen hauptsächlich die Installationshistorie und den Umfang der gespeicherten Nutzungsdaten sowie Zeitstempel.

Zusätzliche Erkenntnisse ergeben sich aus temporären Verzeichnissen auf dem emulierten Gerät. Dort können Zeitstempel aus der Verzeichnisstruktur extrahiert werden, die konkrete Nutzungszeitpunkte der App am 20. und 22. Mai 2025 belegen. Die enthaltenen Dateien, unter anderem im .pb-Format, lassen auf die Nutzung von Google *Perfetto* zur Aufzeichnung von System- und Performance-Traces schließen. Einige dieser Dateien können analysiert und zur Rekonstruktion von Datei- und Speicherzugriffen herangezogen werden.

Die untersuchten Dateien und Verzeichnisse liefern eine Vielzahl an forensisch verwertbaren Artefakten. Durch die gefundenen Cookies und Logdateien ist zweifelsfrei belegbar, dass der Authentifizierungsprozess über feste Dienste erfolgt. Des Weiteren kann auf Grundlage der in den Dateien aufgefundenen Zeitstempel eine zeitliche Einordnung von Sitzungen, Logins, Appnutzung und Appinstallation erfolgen. Zudem können eine Reihe personenbezogener Daten extrahiert werden. Dazu gehören die E-Mail-Adresse, die zur letzten Anmeldung benutzt wurde, eindeutige UUIDs und Token. Diese personenbezogenen Daten ermöglichen es, die App einem konkreten Gerät oder einer Person zuzuordnen und sind damit von zentraler forensischer Bedeutung.

Die enthaltenen Zeitstempel ermöglichen die präzise zeitliche Einordnung von Sitzungen, Logins und App-Nutzung. Auch die gesammelten Informationen über Kameraaktivitäten, Benachrichtigungsauslöser, App-Konfigurationen und Telemetrieeinstellungen ermöglichen eine detaillierte Rekonstruktion der Gerätezustände und Nutzungsintensität. Diese Informationen sind zum Beispiel relevant, wenn ein Zeitabgleich bei Ereignissen oder Alibiprüfungen durchgeführt werden soll.

8 Herausforderungen und Grenzen bei der Analyse

Durch die Untersuchung der Bosch Smart Camera App 2.8.0 und der Bosch Eyes Innenkamera II können, wie in dieser Arbeit aufgezeigt wurde, eine ganze Reihe von forensisch relevanten Informationen gewonnen werden. Bei der Untersuchung zeigen sich jedoch auch zahlreiche Probleme und Herausforderungen. Auf diese soll nun in diesem Kapitel eingegangen werden.

Im Rahmen der Netzwerkanalyse stellt sich die Verschlüsselung des Datenverkehrs als zentrales Problem dar. Zudem ist es durch das verwendete Zertifikat-Pinning der App nicht möglich, diese mit dem Burp-Suite-MITM-Proxy zu umgehen. Auch bei der Kamera kann die Verschlüsselung des Datenverkehrs nicht umgangen werden, da keine Möglichkeit gefunden wurde, das dafür notwendige Zertifikat zu installieren. Dadurch sind die Möglichkeiten der Analyse des Datenflusses und dessen Inhalts stark beschränkt. Ein weiteres Problem stellt sich bei der Analyse des von der Kamera auf Port 443 angebotenen HTTPS-Dienstes heraus. Dort können ausschließlich Informationen zu Zertifikaten und zur Verschlüsselung erhalten werden, da die Wörterbuchtests mit *ffuf* und *dirb* keine Ergebnisse brachten. Dadurch kann der Dienst nicht weiter untersucht werden.

Ein weiteres bedeutendes Problem ist, dass die Firmware der Kamera nicht untersucht werden konnte, da die Kamera nicht über eine von außen zugänglicher Schnittstelle verfügt und nicht beschädigt werden durfte.

Auch bei der Appanalyse zeigen sich einige Probleme. So ist vor allem die dynamische Analyse der App von Herausforderungen geprägt. Diese entstehen hauptsächlich durch die Verwendung von *Docker*, *Genymotion* und *MobSF* unter Windows. Die Probleme können, wie bereits beschrieben, zum größten Teil durch die Verwendung von zwei Laptops im selben Netzwerk umgangen werden. Ein weiteres Problem ist, dass die mit *Genymotion* eingerichtete VM plötzlich trotz unveränderter Netzwerkeinstellungen die Internetverbindung verlieren kann. Dieses Problem tritt bei *Genymotion* unter Windows häufig auf und kann am einfachsten durch die Einrichtung einer neuen VM behoben werden. Bei der Analyse des mit dem in *MobSF* integrierten „HTTP(s)-Proxys“ abgefangenen Datenverkehrs kann nur die Kommunikation sichtbar gemacht werden, die mit dem Appstart- bzw. Anmeldeprozess in Verbindung steht. Damit ist es nicht möglich, die Kommunikation mit den anderen Servern genauer zu analysieren.

Aber auch bei der Analyse der Daten, die mit der ADB extrahiert wurden, treten Probleme auf. So wird zunächst trotz Root-Rechten das Herunterladen der Daten verhindert. Dies kann nach dem Umschalten des SELinux-Modus umgangen werden. Auch die Analyse der Daten hält einige Herausforderungen bereit. So erweist es sich zunächst als schwierig festzustellen, an welchen Stellen Daten mit forensischer Relevanz gespeichert sind. Dieses Problem entsteht dadurch, dass im Großteil der Fachliteratur die neuere Verzeichnisstruktur von Android noch nicht beschrieben ist. Auch die Vielzahl an unterschiedlichen Dateiformaten stellt eine

Herausforderung bei der Analyse dar. Eine weitere Schwierigkeit ist, dass viele Verzeichnisse und Dateien zum Zeitpunkt der Analyse keine Daten enthalten und so nur Vermutungen zu deren Nutzung getroffen werden können.

Zusammenfassend lässt sich feststellen, dass die Analyse von IoT-Geräten und den dazugehörigen Anwendungen technisch anspruchsvoll ist und eine Reihe von Herausforderungen bereithält. Als die drei wesentlichen Probleme stellten sich dabei die Verschlüsselung, die Identifikation von relevanten Informationen und deren Interpretation heraus. Trotz dieser Probleme können einige wichtige Erkenntnisse über die Funktionsweise der Kamera und der App gewonnen werden. Auch gelingt es, Speicherorte von relevanten Daten zu identifizieren.

9 Zusammenfassung und Ausblick

In diesem abschließenden Kapitel werden die zentralen Erkenntnisse aus der Arbeit noch einmal zusammengefasst und ein Ausblick für zukünftige Forschungsansätze geliefert.

9.1 Kurzzusammenfassung der relevanten Ergebnisse

Die forensische Analyse der Bosch Smart Camera App und der Bosch Eyes Innenkamera II offenbart ein komplexes Zusammenspiel aus sicherheitsrelevanter Netzwerkkommunikation, Datenspeicherung und Schutzmechanismen gegen Angriffe. Die Untersuchung stützt sich auf Netzwerkanalyse, statische und dynamische Appanalyse sowie gezielte forensische Extraktion von appbezogenen Daten unter Laborbedingungen.

Die Netzwerkanalyse der Kommunikation der App und der Bosch Eyes Innenkamera II zeigt, dass diese vollständig cloudbasiert ist. Das heißt, sämtliche Kommunikation wie Livestreaming, Übermittlung von Steuerbefehlen und Alarmmeldungen sowie die Speicherung von Screenshots erfolgen über Bosch-Server und nicht im lokalen Netzwerk. So kann keine direkte Kommunikation zwischen Kamera und App beobachtet werden. Der Datenverkehr ist durchgängig verschlüsselt und nutzt zur Übertragung TLSv1.2, TLSv1.3 und QUIC. Dadurch können die Nutzdaten nicht eingesehen werden. Jedoch ist es durch die Analyse der Metadaten und der Kommunikationsmuster möglich, wichtige Erkenntnisse zu gewinnen. Dazu gehören unter anderem Zieladressen und IP-Adressen. So ist es auch möglich, eine Reihe von Drittanbieter-Diensten wie Firebase und hCAPTCHA zu identifizieren.

Die Analyse der App zeigt des Weiteren, dass mehrere Schutzmaßnahmen implementiert sind. Diese erschweren nicht nur Angriffe, sondern auch forensische Analysen. So kann ein SSL-Zertifikat-Pinning erkannt werden. Ein solches Zertifikat-Pinning verhindert MITM-Angriffe. Dieser Sicherheitsmechanismus kann jedoch mit Hilfe von *Frida* umgangen werden, wodurch Teile des Datenverkehrs im Klartext einsehbar sind. Neben dem SSL-Zertifikat-Pinning können auch Anti-Debugging- und Anti-VM-Techniken sowie Code-Verschleierungstechniken wie beispielsweise r8-Compiler ohne Marker festgestellt werden. Diese Schutzmechanismen sind typisch für sicherheitskritische oder potenziell datenschutzsensible Anwendungen. Auch diese können mit *Frida* umgangen werden.

Bei der Appanalyse können außerdem eine Reihe von Schwachstellen identifiziert werden. Dazu gehören unter anderem, dass das Debugging der App aktiviert ist. Zudem wird der veraltete Hashing-Algorithmus SHA-1 verwendet, was aus kryptografischer Sicht kritisch zu bewerten ist. Die identifizierten hartkodierten API-Schlüssel und Token ermöglichen potenziell unbefugten Zugriff auf Backend-Systeme. Des Weiteren können ungesicherte SQL-Abfragen erkannt werden, was das Risiko von SQL-Injection Angriffen erhöht. Dies sind nur einige Beispiele für erkannte Schwachstellen.

Auch zur Kryptografie und Schlüsselverwaltung können in der dynamischen Analyse Informationen gewonnen werden. So nutzt die App moderne Verschlüsselungstechniken. Erkannt werden mit Hilfe von *MobSF* die Nutzung von AES im CBC-Modus mit PKCS5Padding

und dynamischer IV-Generierung sowie RSA im veralteten Modus ECB mit dem veralteten PKCS1Padding. Die Speicherung des privaten RSA-Schlüssels erfolgt im Android Keystore. Dies stellt einen starken Schutz des Schlüssels dar. Allerdings verbleiben AES-Schlüssel im RAM und sind so potenziell extrahierbar.

Neben der Netzwerkanalyse und der Appanlyse bringt auch die Analyse der mit ADB extrahierten Daten mit App-Bezug zahlreiche forensisch relevante Erkenntnisse mit sich. So können in *pref_store* Authentifizierungsdaten wie Token und E-Mail-Adresse, Konfigurationsdaten, Zeitstempel und Telemetrieeinstellungen gewonnen werden. Auch Sessioninformationen sind dort zu finden. Die Datei *cameras.json* enthält Details zur Kamerakonfiguration wie Status, Modus, Zeitzone und Benachrichtigungseinstellungen. Diese ermöglichen eine Rekonstruktion der Nutzung der Kamera. In *pref_store* und *Cookies* können eine eindeutige ID, der letzte Nutzungszeitpunkt und Verweise auf Authentifizierungsprozesse über SingleKey ID gefunden werden. Firebase-Installationsdaten, Push-Token, Authentifizierungstoken und Sessioninformationen können ebenfalls extrahiert werden.

Die Analysen zeigen, dass sich aus den extrahierten Daten Abläufe von Nutzer- und Geräteaktivitäten rekonstruieren lassen. Besonders die Untersuchung von Nutzerprofilen, Kommunikationsmustern und potenziellen Schwachstellen liefert wertvolle Anhaltspunkte zur Einschätzung von Sicherheitsrisiken und zur Entwicklung geeigneter Gegenmaßnahmen. Durch die Kombination verschiedener Analysemethoden können nicht nur vorhandene Schwachstellen identifiziert werden, sondern auch die Komplexität der Datenverarbeitung in der Bosch Eyes Innenkamera II und in der Bosch Smart Camera App deutlich gemacht werden.

9.2 Ausblick

Die Untersuchung der Bosch Smart Camera App und der Bosch Eyes Innenkamera II beschreibt exemplarisch, wie vernetzte Geräte mit sensiblen Nutzerdaten umgehen. So zeigt sich, wie bei anderen IoT-Geräten auch, dass bei Nutzung der App und der Kamera eine Vielzahl digitaler Spuren entsteht. Diese können in forensischen Untersuchungen bedeutende Hinweise liefern. In dieser Arbeit wurde dargelegt, welche spezifischen Herausforderungen die forensische Untersuchung eines solchen IoT-Gerätes mit sich bringt. Zudem wurde herausgestellt, wie stark die IP-fähige Bosch Eyes Innenkamera II, die Bosch-Cloud-Infrastruktur, die App und eine Vielzahl verschiedener Drittanbieter-Dienste miteinander vernetzt sind.

Aus diesen Erkenntnissen ergeben sich für künftige Arbeiten mehrere aussichtsreiche Ansätze, um die Gewinnung forensisch relevanter Daten gezielt zu erweitern und zu verbessern.

Ein Ansatz ist, in der Bosch Smart Camera App eine Verbindung zur Bosch Smart Home App und zu einem Bosch Smart Home Controller herzustellen und darüber Partnergeräte wie beispielsweise Lichtsysteme oder Sprachassistenten zu integrieren. Die Analyse dieser Verbindungen und der damit verbundenen Datenflüsse könnte wertvolle Erkenntnisse über zusätzliche Kommunikationskanäle und Schnittstellen liefern. Forensische Untersuchungen könnten sich darauf konzentrieren, die Authentifizierungsvorgänge und die übermittelten Steuerungsbefehle zu untersuchen, um potenzielle Schwachstellen zu identifizieren und bis-

lang unzugängliche oder versteckte Datenquellen zu erschließen. Diese Integration stellt somit ein relevantes Ziel für weiterführende Analysen dar, um die Gesamtsicherheitsarchitektur des vernetzten Systems besser zu verstehen.

Darüber hinaus eröffnet die Untersuchung der eingesetzten Drittanbieter-Software ein weiteres Feld. Die Bosch Eyes Innenkamera II und die Bosch Smart Camera App nutzen zahlreiche externe Bibliotheken und Dienste, die potenziell bekannte Schwachstellen enthalten können. Zukünftige Arbeiten könnten gezielt versuchen, solche Sicherheitslücken zu identifizieren und auszunutzen, um auf verschlüsselte oder versteckte Daten zuzugreifen. Dies setzt jedoch fundierte Kenntnisse der verwendeten Software sowie eine sorgfältige ethische und rechtliche Bewertung voraus.

Auch eine vertiefte Speicheranalyse stellt einen vielversprechenden Ansatz dar. Physische Extraktionsmethoden wie JTAG oder Chip-Off können es ermöglichen, Daten auszulesen, die im regulären Betrieb des Geräts nicht zugänglich sind, insbesondere verschlüsselte oder gelöschte Informationen. Dabei könnte auch der Versuch unternommen werden, eigene Zertifikate auf dem Gerät zu installieren, um den verschlüsselten Datenverkehr zu entschlüsseln und die Kommunikation detaillierter zu analysieren. Diese Möglichkeit konnte in der vorliegenden Arbeit aufgrund technischer Einschränkungen noch nicht umgesetzt werden.

Zudem könnte die Netzwerkanalyse weiter verfeinert werden. Insbesondere die Entwicklung neuer Methoden zum Umgang mit modernen Verschlüsselungsprotokollen wie TLSv1.3 sowie der Einsatz spezialisierter Hard- oder Software zur Entschlüsselung des Netzwerkverkehrs könnten helfen, bislang unzugängliche Daten aus der Kommunikation mit externen Servern zu extrahieren.

Ein weiterer Ansatz besteht in der Analyse der Firmware der Bosch Eyes Innenkamera II innerhalb einer virtuellen Umgebung, vergleichbar mit der Vorgehensweise bei der Bosch Smart Camera App in dieser Arbeit. Dazu wären zunächst eine korrekte Extraktion und eine funktionale Emulation der Firmware notwendig. Eine solche Analyse könnte wertvolle Erkenntnisse über interne Prozesse, Datenflüsse sowie Systemabhängigkeiten liefern und bisher schwer zugängliche Schwachstellen oder Angriffspunkte offenlegen.

Insgesamt eröffnen sowohl neue technische Ansätze als auch die Weiterentwicklung bestehender Methoden zahlreiche Möglichkeiten für künftige Arbeiten. Diese könnten nicht nur zusätzliche forensisch relevante Daten liefern, sondern auch das Verständnis der Sicherheitsarchitektur der Bosch Eyes Innenkamera II und der zugehörigen App weiter vertiefen.

Anhang A: Auszüge aus dem „HTTP(S) Traffic -Log“

```
========
REQUEST
========
POST https://in.appcenter.ms/logs?api-version=1.0.0 HTTP/1.1
Install-ID: fb36a517-5e17-43ec-8858-4173cd9ca7d8
App-Secret: f91cxxxx-xxxx-xxxx-xxxx-xxxxxxxxef6f
Content-Type: application/json
Content-Length: 604
User-Agent: Dalvik/2.1.0 (Linux; U; Android 11; Pixel 5 Build/RQ1A.210105.003)
Host: in.appcenter.ms
Connection: Keep-Alive
Accept-Encoding: gzip

b'{"logs":[{"type":"startService","timestamp":"2025-06-03T07:28:58.520Z","device
    ":{"wrapperSdkVersion":"5.0.3","wrapperSdkName":"appcenter.xamarin","
    wrapperRuntimeVersion":"34.0.1.137","sdkName":"appcenter.android","sdkVersion
    ":"5.0.3","model":"Pixel␣5","oemName":"Genymobile","osName":"Android","
    osVersion":"11","osBuild":"RQ1A.210105.003","osApiLevel":30,"locale":"en␣US
    ","timeZoneOffset":120,"screenSize":"1080x2210","appVersion":"2.8.0","
    carrierName":"Android","carrierCountry":"us","appBuild":"1279","appNamespace
    ":"com.bosch.cbs.consumer.prod"},"services":["Crashes"],"
    isOneCollectorEnabled":false}]}'
=========
RESPONSE
=========
HTTP/1.1 200 OK
Date: Tue, 03 Jun 2025 07:29:02 GMT
Content-Type: application/json
Content-Length: 138
Connection: keep-alive
Server: Kestrel

b'{"status":"AllLogsThrottled","validDiagnosticsIds":[],"throttledDiagnosticsIds
    ":[],"correlationId":"4ff204b6-b687-4ac3-ab80-aadfda485586"}'
```

Quelltext A.1: POST-Anfrage und Antwort von https://in.appcenter.ms(App-Secret verfremdet)

```
========
REQUEST
========
POST https://singlekey-id.com/en-us/login/password?returnUrl=%2Fauth%2Fconnect%2
    Fauthorize%2Fcallback%3Fscope%3Dopenid%2520email%2520profile%2520
    offline_access%26state%3DZMrl1WgDMI7dus_ZYtayjvknbS_VfwM-VPzvpCoYhSE.
    QUrrKOLx4zo.Xr90sVi3THi39yfwpJS9sw%26response_type%3Dcode%26client_id%3
    DD4xxxxxx-xxxx-xxxx-xxxx-xxxxxxxx0991%26redirect_uri%3Dhttps%253A%252F%252
    Fsmarthome.authz.bosch.com%252Fauth%252Frealms%252Fhome_auth_provider%252
    Fbroker%252Fskid-p%252Fendpoint%26code_challenge%3
    DAOJ9zFVU3_SiyWGQOWOp5quGFTuIN_Qa5M52BuvtW-0%26code_challenge_method%3DS256
    %26nonce%3Dw94srBcFEThlV2_Lt-xo0A&f=YHpm HTTP/2.0
content-length: 231
hx-request: true
hx-current-url: https://singlekey-id.com/en-us/login/password?returnUrl=%2Fauth%2
    Fconnect%2Fauthorize%2Fcallback%3Fscope%3Dopenid%2520email%2520profile%2520
    offline_access%26state%3DZMrl1WgDMI7dus_ZYtayjvknbS_VfwM-VPzvpCoYhSE.
    QUrrKOLx4zo.Xr90sVi3THi39yfwpJS9sw%26response_type%3Dcode%26client_id%3
    DD4xxxxxx-xxxx-xxxx-xxxx-xxxxxxxx0991%26redirect_uri%3Dhttps%253A%252F%252
    Fsmarthome.authz.bosch.com%252Fauth%252Frealms%252Fhome_auth_provider%252
    Fbroker%252Fskid-p%252Fendpoint%26code_challenge%3
    DAOJ9zFVU3_SiyWGQOWOp5quGFTuIN_Qa5M52BuvtW-0%26code_challenge_method%3DS256
    %26nonce%3Dw94srBcFEThlV2_Lt-xo0A&f=YHpm
hx-target: body
user-agent: Mozilla/5.0 (Linux; Android 11; Pixel 5 Build/RQ1A.210105.003; wv)
    AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/83.0.4103.120
    Mobile Safari/537.36
hx-boosted: true
content-type: application/x-www-form-urlencoded
accept: */*
origin: https://singlekey-id.com
x-requested-with: com.bosch.cbs.consumer.prod
sec-fetch-site: same-origin
sec-fetch-mode: cors
sec-fetch-dest: empty
referer: https://singlekey-id.com/en-us/login/password?returnUrl=%2Fauth%2
    Fconnect%2Fauthorize%2Fcallback%3Fscope%3Dopenid%2520email%2520profile%2520
    offline_access%26state%3DZMrl1WgDMI7dus_ZYtayjvknbS_VfwM-VPzvpCoYhSE.
    QUrrKOLx4zo.Xr90sVi3THi39yfwpJS9sw%26response_type%3Dcode%26client_id%3
    DD4xxxxxx-xxxx-xxxx-xxxx-xxxxxxxx0991%26redirect_uri%3Dhttps%253A%252F%252
    Fsmarthome.authz.bosch.com%252Fauth%252Frealms%252Fhome_auth_provider%252
    Fbroker%252Fskid-p%252Fendpoint%26code_challenge%3
    DAOJ9zFVU3_SiyWGQOWOp5quGFTuIN_Qa5M52BuvtW-0%26code_challenge_method%3DS256
    %26nonce%3Dw94srBcFEThlV2_Lt-xo0A&f=YHpm
accept-encoding: gzip, deflate
accept-language: en-US,en;q=0.9
cookie: AF=CfDJ...h0CU, __Host-FYHpm=CfDJ...D50g


b'Password=Axxxxxxxxxxxxxxxxxxxe&RememberMe=false&__RequestVerificationToken=
    CfDJ8D999u_NU4dGhTI0t3Li-DEcJQWwJbrlFLI1rib3eF_b3TwYYGti6QAVn7aSV0vmsOdn1u_f-
    cLjveaJXKS1RTDT1T1WNBOUfx5Vpy6sDCTNoAgl3flr8P_8f7OxCatYKkbjv5Xgvpt1
```

```
hygS9lNxRS4'
```

Quelltext A.2: POST-Anfrage mit sichtbarem Passwort im Klartext (Passwort, Client-ID und Cookies verfremdet)

```
=========
RESPONSE
=========
HTTP/2.0 200
date: Tue, 17 Jun 2025 05:09:43 GMT
content-type: text/html; charset=utf-8
vary: Accept-Encoding
cache-control: no-store
set-cookie: .AspNetCore.Mvc.CookieTempDataProvider=; expires=Thu, 01 Jan 1970
    00:00:00 GMT; path=/; secure; samesite=lax; httponly, __Host-FYHpm=CfDJ...
    D50g; max-age=1800; path=/; secure; samesite=lax; httponly
request-context: appId=cid-v1:1e51xxxx-xxxx-xxxx-xxxx-xxxxxxxx9d18
content-security-policy: base-uri 'self';frame-ancestors 'self';upgrade-insecure-
    requests;default-src 'none';form-action 'self';connect-src 'self' https://
    hcaptcha.com https://*.hcaptcha.com;img-src 'self' https://cdn.singlekey-id.
    com;font-src 'self' https://cdn.singlekey-id.com;style-src 'self' https://
    hcaptcha.com https://*.hcaptcha.com 'nonce-9Vb8PO7ebfh_Lgj3adIPTjjC' https://
    cdn.singlekey-id.com;script-src 'self' https://hcaptcha.com https://*.
    hcaptcha.com 'nonce-9Vb8PO7ebfh_Lgj3adIPTjjC' https://cdn.singlekey-id.com;
    frame-src 'self' https://hcaptcha.com https://*.hcaptcha.com
strict-transport-security: max-age=31536000
content-encoding: gzip
x-content-type-options: nosniff
x-xss-protection: 0
x-frame-options: SAMEORIGIN
referrer-policy: strict-origin-when-cross-origin
cross-origin-opener-policy: same-origin-allow-popups


b'\n\n\n\n\n<div␣class="seamless-content"␣data-hx-boost="true">\n␣␣␣␣<h2␣class="
    seamless-content__header">Log␣in</h2>\n␣␣␣␣<form␣class="form"␣method="post"␣
    data-hx-boost="true">\n␣␣␣␣␣␣␣␣\n<a␣class="seamless-content__username-link␣
    link"␣data-testid="link-back-to-login"␣href="/en-us/login?returnUrl=%2Fauth%2
    Fconnect%2Fauthorize%2Fcallback%3Fscope%3Dopenid%2520email%2520profile%2520
    offline_access%26state%3DZMrl1WgDMI7dus_ZYtayjvknbS_VfwM-VPzvpCoYhSE.
    QUrrKOLx4zo.Xr90sVi3THi39yfwpJS9sw%26response_type%3Dcode%26client_id%3
    DD4xxxxxx-xxxx-xxxx-xxxx-xxxxxxxx0991%26redirect_uri%3Dhttps%253A%252F%252
    Fsmarthome.authz.bosch.com%252Fauth%252Frealms%252Fhome_auth_provider%252
    Fbroker%252Fskid-p%252Fendpoint%26code_challenge%3
    DAOJ9zFVU3_SiyWGQOWOp5quGFTuIN_Qa5M52BuvtW-0%26code_challenge_method%3DS256
    %26nonce%3Dw94srBcFEThlV2_Lt-xo0A&amp;f=YHpm">\n␣␣␣␣<i␣class="icon␣icon--
    arrow-left␣icon--size-s"></i>\n␣␣␣␣<span␣class="break-word">
    rexxxxxxxxxxxxxxxxxxxxxx@gmail.com</span>\n</a>\n␣...
```

Quelltext A.3: Auszug aus der Antwort von https://singlekey-id.com mit E-Mail-Adresse des Nutzers (E-Mail-Adresse, Client-ID und Cookies verfremdet)

Literaturverzeichnis

[1] A. Raab-Düsterhöft, *IT-Forensik: Ein Grundkurs*. Springer Vieweg, 2024. DOI: 10.1007/978-3-662-69090-1.

[2] D. Pawlaszczyk, „Digitaler Tatort, Sicherung und Verfolgung digitaler Spuren“, in *Forensik in der digitalen Welt*, Springer, 2017, S. 113–116. DOI: 10.1007/978-3-662-53801-2_5.

[3] H. Plank und A. Fiedler, „Das „Spuren- und Indizienparadigma“ - Bedeutung innerhalb der kriminalistischen Handlungslehre im Kontext der Cyberkriminalistik und -kriminologie“, in *Handbuch Cyberkriminologie*, Springer, 2023, S. 1–67. DOI: https://doi.org/10.1007/978-3-658-35450-3.

[4] S. Martens, „Anwendung eines Kriterienkatalogs auf kommerzielle und freie Forensik-Werkzeuge“, Abgerufen am 05.04.2025, Bachelorarbeit, Hochschule Mittweida, 2019. Adresse: https://monami.hs-mittweida.de/files/11815/Bachelorarbeit_Sandro_martens.pdf.

[5] M. Führer, „Systematische Übersichtsarbeit für zukünftige Forschungsgegenstände forensischer Auswertungen von Hausautomatisierungen“, Abgerufen am 05.04.2025, Bachelorarbeit, Hochschule Mittweida, 2024. Adresse: https://monami.hs-mittweida.de/files/15251/Bachelorarbeit_Max_F%C3%BChrer_54372_geschwaerzt.pdf.

[6] T. Schuller, „Historische Entwicklung und zukünftige Herausforderungen bei der Identifikation und Wiederherstellung gelöschter Daten in der IT-Forensik“, Abgerufen am 05.04.2025, Bachelorarbeit, Hochschule Mittweida, 2024. Adresse: https://monami.hs-mittweida.de/files/15720/Bachelorarbeit_Thomas_Schuller_55390_26_08_2024.pdf.

[7] W. Honekamp, R. Povalej, H. Rittelmeier, J. Fähndrich, S. Berner und D. Labudde, „Technologiegetriebene Polizeiausbildung im Umgang mit digitalen Spuren“, in *Handbuch Cyberkriminologie*, Springer VS Wiesbaden, 2022, S. 1–30. DOI: https://doi.org/10.1007/978-3-658-35450-3.

[8] R. G. Richter, „Digital-forensische Spuren in Küchengeräten am Beispiel des Thermomix TM6“, Abgerufen am 10.04.2025, Masterarbeit, Hochschule Mittweida – University of Applied Sciences, Mittweida, 2024.

[9] „Leitfaden IT-Forensik“, Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany, Techn. Ber. Version 1.0.1, März 2011, Abgerufen am 07.04.2025. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Leitfaden_IT-Forensik.pdf.

[10] D. Labudde, F. Czerner und M. Spranger, Hrsg., *Forensik in der digitalen Welt: Moderne Methoden der forensischen Fallarbeit in der digitalen und digitalisierten realen Welt*. Berlin, Heidelberg: Springer Vieweg, 2017, ISBN: 978-3-662-53801-2. DOI: 10.1007/978-3-662-53801-2.

[11] A. S. Tanenbaum und T. Austin, *Rechnerarchitektur. Von der digitalen Logik zum Parallelrechner*, 6.,aktualisierte Auflage. Hallbergmoos: Pearson Deutschland GmbH, 2014, ISBN: 978-3-86326-687-5.

[12] M. Kofler u. a., *Hacking & Security: Das umfassende Handbuch*, 3.korrigierte Auflage. Bonn: Reihnwerk Verlag, 2019, ISBN: 978-3-8362-4548-7.

[13] F. Servida und E. Casey, „IoT forensic challenges and opportunities for digital traces“, *Digital Investigation*, Jg. 28, S22–S29, 2019, Abgerufen am 10.04.2025. DOI: 10.1016/j.diin.2019.01.012. Adresse: https://doi.org/10.1016/j.diin.2019.01.012.

[14] S. Wendzel, *IT-Sicherheit für TCP/IP- und IoT-Netzwerke: Grundlagen, Konzepte, Protokolle, Härtung*, 2. Auflage. Wiesbaden: Springer Vieweg, 2021, ISBN: 978-3-658-33423-9. DOI: 10.1007/978-3-658-33423-9.

[15] A. Gupta, *The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things*. Berkeley, CA: Apress, 2019, ISBN: 978-1-4842-4300-8. DOI: 10.1007/978-1-4842-4300-8.

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

[17] M. Moreb, *Practical Forensic Analysis of Artifacts on iOS and Android Devices . Investigating Complex Mobile Devices*. Berkeley,CA: Apress, 2022, ISBN: 978-1-4842-8026-3. DOI: https://doi.org/10.1007/978-1-4842-8026-3.

[18] J. Ottmann, J. Pollach, N. Scheler, J. Schneider, C. Rückert und F. Freiling, „Zur Blackbox-Problematik im Bereich Mobilfunkforensik“, *Datenschutz und Datensicherheit – DuD*, Jg. 45, Nr. 8, S. 546–552, 2021, Abgerufen am 10.04.2025. DOI: 10.1007/s11623-021-1396-9.

[19] S. Herchel, M. Steinert, V. Dehne und D. Labudde, „Der forensische Wert von Thumbnails und Cache-Dateien im Ermittlungsverfahren“, in *INFORMATIK 2024*, Bonn: Gesellschaft für Informatik e.V., 2024, S. 309–319, ISBN: 978-3-88579-746-3. DOI: 10.18420/inf2024_21.

[20] C. Eckert, *IT-Sicherheit. Konzepte – Verfahren – Protokolle*, 11. Auflage. Berlin, Boston: De Gruyter Oldenbourg, 2023, ISBN: 9783110985115. DOI: doi:10.1515/9783110985115.

[21] K. Kent, S. Chevalier, T. Grance und H. Dang, „Guide to Integrating Forensic Techniques into Incident Response“, National Institute of Standards und Technology (NIST), Gaithersburg, MD, Techn. Ber. Special Publication 800-86, Aug. 2006, Abgerufen am 07.04.2025. Adresse: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-86.pdf.

[22] EDRM. „Current EDRM Model“. Abgerufen am 02.07.2025. Adresse: https://edrm.net/edrm-model/current/.

[23] W. W. Osterhage, *Sicherheitskonzepte in der mobilen Kommunikation. Drahtlose Kommunikation - Protokolle und Gefahren*, 1. Auflage. Berlin: Springer Vieweg, 2018, ISBN: 978-3-662-57903-9. DOI: https://doi.org/10.1007/978-3-662-57903-9.

[24] NMAP.ORG. „Chapter 9. Nmap Scripting Engine“. Abgerufen am 12.04.2025. Adresse: https://nmap.org/book/nse.html.

[25] Deciso B.V. „Welcome to OPNsense's documentation!“ Abgerufen am 27.05.2025. Adresse: https://docs.opnsense.org/.

[26] N. K. Nainar und A. Panda, *Wireshark for Network Forensics*, 1. Auflage. Apress Berkeley, CA, 2020, ISBN: 978-1-4842-9001-9. DOI: 10.1007/978-1-4842-9001-9.

[27] PortSwigger Ltd. „Burp Suite documentation: desktop editions“. Abgerufen am 20.04.2025. Adresse: https://portswigger.net/burp/documentation/desktop.

[28] M. Sikorski und A. Honig, *Practical Malware Analysis. The Hands-On Guide to Dissecting Malicious Software*. Hallbergmoos: No Starch Press, 2012, ISBN: 1-59327-290-1.

[29] R. Regupathy, *Unboxing Android USB. A hands on approach with real world examples*. Berkeley,CA: Apress, 2014, ISBN: 978-1-4302-6209-1. DOI: https://doi.org/10.1007/978-1-4302-6209-1.

[30] DB Browser for SQLite Project. „DB Browser for SQLite“. Abgerufen am 03.05.2025. Adresse: https://sqlitebrowser.org/.

[31] M. Hörz. „HxD - Freeware Hex-Editor und Disk-Editor“. Abgerufen am 08.06. 2025. Adresse: https://mh-nexus.de/de/hxd/.

[32] Perfetto. „Perfetto - System profiling, app tracing and trace analysis“. Abgerufen am 08.06. 2025. Adresse: https://perfetto.dev/docs/.

[33] S. A. Khan u. a., „An Android Applications Vulnerability Analysis Using MobSF“, in *IEEE International Conference on Engineering and Computing Technologies (ICECT)*, Abgerufen am 13. April 2025, 2024. Adresse: https://ieeexplore.ieee.org/document/10581312.

[34] S. U. Kusreynada und A. S. Barkah, „Android Apps Vulnerability Detection with Static and Dynamic Analysis Approach using MOBSF“, *Journal of Computer Science and Engineering (JCSE)*, Jg. 5, Nr. 1, S. 46–63, 2024, Abgerufen am 13.04.2025. Adresse: http://dx.doi.org/10.36596/jcse.v5i1.789.

[35] Mobile Security Framework (MobSF). „Mobile Security Framework - GitHub Repository“. Abgerufen am 13. April 2025. Adresse: https://github.com/MobSF/Mobile-Security-Framework-MobSF.

[36] Genymobile. „Genymotion Desktop – Android Emulator for Application Testing“. Abgerufen am 13. April 2025. Adresse: https://www.genymotion.com/product-desktop/.

[37] Mobile Security Framework (MobSF). „Dynamic Analyzer Docker Setup – MobSF Documentation“. Abgerufen am 13. April 2025. Adresse: https://mobsf.github.io/docs/#/dynamic_analyzer_docker.

[38] Robert Bosch Smart Home GmbH. „Eyes Innenkamera II“. Abgerufen am 09.04.2025. Adresse: https://www.bosch-smarthome.com/de/de/produkte/kameras/eyes-innenkamera/.

[39] Robert Bosch Smart Home GmbH. „Bosch Eyes Innenkamera II Kurzanleitung“. Abgerufen am 09.04.2025. Adresse: https://tt-smarthome.resource.bosch.com/nrt/web/produkt/innenkamera2/PDF/Anleitung/Eyes_Indoor_DE_20210825.pdf.

[40] Robert Bosch Smart Home GmbH. „Bosch Smart Camera App“. Abgerufen am 09.04.2025. Adresse: https://www.bosch-smarthome.com/de/de/apps/bosch-smart-camera-app/.

[41] Robert Bosch Smart Home GmbH. „Hilfe zur Eyes Innenkamera II“. Abgerufen am 09.04.2025. Adresse: https://www.bosch-smarthome.com/de/de/service/hilfe/hilfe-zum-produkt/hilfe-zur-eyes-innenkamera-2/.

[42] Robert Bosch GmbH Home GmbH. „Cloud+ Videospeicher-Erweiterung“. Abgerufen am 09.04.2025. Adresse: https://www.bosch-smarthome.com/de/de/produkte/abonnements/cloud-plus/.

[43] Robert Bosch Smart Home GmbH. „OpenAPI Documentation. Bosch Smart Home Local API for Main Resources“. Abgerufen am 10.07.2025. Adresse: https://local.apidocs.bosch-smarthome.com/.

[44] Robert Bosch Smart Home GmbH. „SingleKey ID FAQ“. Abgerufen am 23.04.2025. Adresse: https://www.bosch-home.com/de/mybosch/singlekeyid-faq.

[45] Robert Bosch Smart Home GmbH. „Datenschutzhinweis - Bosch Smart Camera App“. Abgerufen am 09.04.2025. Adresse: https://www.bosch-smarthome.com/de/de/privacy-statement/camera/.

[46] National Vulnerability Database. „CVE-2019-7729 Detail“. Abgerufen am 23.04.2025. Adresse: https://nvd.nist.gov/vuln/detail/CVE-2019-7729.

[47] National Vulnerability Database. „CVE-2020-6781 Detail“. Abgerufen am 23.04.2025. Adresse: https://nvd.nist.gov/vuln/detail/CVE-2019-7729.

[48] Google LLC. „Google Analytics“. Abgerufen am 09.06.2025. Adresse: https://firebase.google.com/docs/analytics?hl=de.

[49] Microsoft. „App Center Crashes (Android)“. Abgerufen am 09.06.2025. Adresse: https://learn.microsoft.com/en-us/appcenter/sdk/crashes/android.

[50] S. Rahalkar, *A Complete Guide to Burp Suite. Learn to Detect Application Vulnerabilities*. Apress Berkeley, CA, 2020, ISBN: 978-1-4842-6402-7. DOI: https://doi.org/10.1007/978-1-4842-6402-7.

[51] A. Beutelspacher, J. Schwenk und K.-D. Wolfenstetter, *Moderne Verfahren der Kryptographie. Von RSA zu Zero-Kwowledge und darüber hinaus*, 9. Auflage. Heidelberg: Springer Spektrum Berlin, 2022, ISBN: 978-3-662-65718-8. DOI: https://doi.org/10.1007/978-3-662-65718-8.

Eidesstattliche Erklärung

Hiermit versichere ich – Sandra Lindner – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 18. September 2025	
Ort, Datum	Sandra Lindner, B.Sc.