ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

Informatik
& Security

fh
st.pölten

Sicherheitsanalyse eines Einsteiger-Saugroboter Modells

Welche Sicherheitsschwachstellen weist der Ecovacs Deebot Slim 2 auf?

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

eingereicht von

Alexander Jäger, BSc.

im Rahmen des
Studiengangs MIS19 an der Fachhochschule St. Pölten

Betreuung
Betreuer/Betreuerin: Dipl.-Ing. Dr. Robert Luh, BSc

Wien, 28.05.2023

(Unterschrift) Alexander Jäger

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Wien, 28.05.2023

(Unterschrift Alexander Jäger)

Inhaltsverzeichnis

Inhalt

Zusammenfassung

IoT-Geräte werden auf dem Markt immer beliebter. Aufgrund der raschen technologischen Weiterentwicklung und der wachsenden Nachfrage sind IoT-Geräte meist anfälliger für Angriffe im Vergleich zu anderen Anwendungen. Komplette Smart-Home Systeme werden immer mehr zur Realität und dadurch wird auch die Sicherheit dieser Geräte immer wichtiger. Saugroboter sind ein IoT Produkt, deren Anzahl und Komplexität im Privatsektor in den letzten Jahren besonders stark zugenommen hat [1]. Sie vereinfachen einerseits den Alltag und sind andererseits dank einer langlebigen Dauer auch aus finanzieller Sicht eine beliebte Investition. Es ist jedoch genau diese Langlebigkeit, die auch ihre Tücken birgt, da der Saugroboter für zumeist mehrere Jahre in einem Haushalt gebraucht wird und dabei ein Bestandteil des Heimnetzwerkes ausmacht. Fraglich ist es daher, inwieweit ein Verbraucher das nötige Wissen hat, um zu erkennen, wann der Saugroboter eine potenzielle Gefahrenquelle darstellt und für den Fall, dass sich dies sogar bewahrheitet, inwieweit die Hersteller solcher Geräte dies versuchen zu verhindern.

In dieser Arbeit wird versucht den Ablauf eines Sicherheitstests bei IoT-Geräten vorzuschlagen und diese Testmethoden an dem Saugroboter Deebot Slim 2 von Ecovacs umzusetzen. Um potenzielle Angriffsziele vorab einzuschätzen, wurden die DREAD und STRIDE Bedrohungsmodelle angewendet. Anschließend wurden basierend auf diesen Bedrohungsmodellen Tests auf dem System durchgeführt, um die Hardwaresicherheit des Saugroboters zu evaluieren.

Abstract

IoT devices are becoming increasingly popular in the market. Due to the rapid technological development and the growing demand on the market, they are usually more vulnerable to attacks compared to other applications. A complete smart home is becoming more and more of a reality and as a result, the security of these devices is also becoming more important. Robot vacuum cleaners are an IoT product whose number and complexity in the private sector has increased particularly strongly in recent years [1]. On the one hand, they simplify everyday life and, on the other, they are also a popular investment from a financial perspective thanks to their longevity. However, it is precisely this longevity that also has its pitfalls since the robot vacuum cleaner is usually used for several years in a household and thereby constitutes a component of the home network. It is therefore questionable to what extent a consumer has the necessary knowledge to recognise when the robot vacuum cleaner represents a potential source of danger and, if this even proves to be true, to what extent the manufacturers of such devices try to prevent this.

This paper attempts to propose a security test procedure for IoT devices and implement these test methods on the Deebot Slim 2 robot vacuum cleaner from Ecovacs. To assess potential attack targets in advance, the DREAD and STRIDE threat models were applied. Then, based on these threat models, tests were conducted on the system to evaluate the hardware security of the robot vacuum cleaner.

1. **Einleitung**

Die Heimautomatisierung und die Zahl der angeschlossenen Geräte in unseren Häusern sind in den letzten Jahren explodiert. Vor allem die Zahl der Geräte aus dem Internet der Dinge (engl. Internet of Things - IoT) hat drastisch zugenommen. Es wird vorhergesagt, dass bis 2027-2028 mit einem Zuwachs um 22 Prozent zu rechnen ist und, somit bis zu 40 Milliarden Geräte in Verwendung sein werden [1]. Viele der IoT-Geräte sind mit dem Internet verbunden, und dieser Anteil wird mit dem Aufkommen der 5G-Technologie sicherlich noch steigen. Diese Geräte können zwar erstaunliche Dinge sein, von intelligenten Lautsprechern bis hin zu vernetzten Kühlschränken, jedoch sind sie nicht für ihre Sicherheit bekannt.

Die mangelnde Sicherheit von IoT-Geräten ist zwar in der IT-Sicherheitsgemeinschaft bekannt, aber die durchschnittlichen, nicht technikaffinen Verbraucher sind sich dieser Sicherheitsaspekte vielleicht nicht so sehr bewusst, wenn sie ein IoT-Gerät in ihr Zuhause bringen. Es besteht auch die Gefahr, dass diese IoT-Geräte schon mit Hardware ausgeliefert werden, die bereits manipuliert wurde. So könnte ein Gerät abgefangen und mit Schadsoftware ausgestattet werden, bevor es an den Konsumenten gelangt.

Aber warum ist IoT Sicherheit auch im industriellen Bereich wichtig? IoT-Geräte können nicht nur für Bequemlichkeiten wie Saugroboter verwendet werden, sondern auch zum Beispiel für Sensoren, die Gesundheitsdaten an den Hausarzt senden, um frühzeitig Herzinfarkte zu erkennen, oder um Waren und Güter mit einem selbstfahrenden Auto zu transportieren. Die hohe Anzahl an IoT-Geräten in Privathaushalten stellt auch eine Gefahr für die Industrie dar. Diese Geräte, die oft eine schlechte Security haben, können übernommen und für Angriffe auf Unternehmensstrukturen verwendet werden, wie am Beispiel des Mirai Botnet aus dem Jahr 2016, das sich vorwiegend an IoT-Geräten vergriffen hat [2].

Die meisten Privatpersonen akzeptieren das Risiko, das mit IoT-Geräten einherkommt oder haben nicht die Kenntnisse, sich mit solchen Problemen auseinander zu setzen. Durch die Anbindung an Cloud-Infrastrukturen wurde ein weiterer Angriffsvektor miteinbezogen. Die IoT Welt ist geschaffen aus Geräten, die wenig Computerleistung, Arbeitsspeicher und Festplattenspeicher haben, aber sehr oft spezielle Funktionen bedienen, wie spezifische Protokolle. Aufgrund dessen bieten diese Geräte eine besondere Angriffsfläche im Vergleich zu herkömmlichen Geräten. Um die verschiedenen Angriffsvektoren zu identifizieren, werden die Threat Models STRIDE und DREAD angewendet. Der Bauweise verschuldet, werden IoT-Geräte oft nicht mit der notwendigen Sicherheitssoftware, wie Software Whitelisting oder Network Access Control (NAC), ausgestattet und die meisten Hersteller liefern keine bis wenig Security Updates [3].

Aus den oben genannten Gründen widmet sich diese Arbeit einem IoT Gerät, welches neben intelligenten Steckdosen in den letzten Jahren seine Anzahl stark in privaten Sektor erhöhen konnte, [1] einem Saugroboter. Im Zuge dieser Arbeit wird eine Sicherheitsanalyse des Deebot Slim 2 von Ecovacs, einem Staubsaugroboter aus dem Einsteigersegment, gemacht und die Frage „Welche

Sicherheitsschwachstellen weist der Deebot Slim 2 auf?“ beantwortet. Auf dieser Frage aufbauend ergibt sich die Hypothese, dass der Deebot Slim 2 Sicherheitsschwachstellen hat. Durch seinen günstigen Preis ist er in vielen Haushalten im Einsatz, was ihn zu einem interessanten Ziel macht. Der Saugroboter ist sowohl über zwei unterschiedliche Apps, „Deebot 2017“ und „Ecovacs Home“, wie auch über eine Fernbedienung bedienbar. Da die Analyse von IoT-Geräten andere Methoden benötigt, soll ein Leitfaden für die Analyse von IoT-Geräten erstellt werden, der später auf andere Geräte umgemünzt werden kann. Am Ende dieser Arbeit werden die gefundenen Schwachstellen sowie Beispiele für weitere Vorgehensweisen und Gegenmaßnahmen besprochen.
Neben der Hauptfrage ergeben sich folgende weitere Fragen

- Welche Methodiken müssen für einen guten Sicherheitstest bei IoT-Geräten angewendet werden?
- Welche Hardware- und Softwaregegenmaßnahmen wurden bereits getroffen?
- Welche Sicherheitsmaßnahmen können getroffen werden um Schwachstellen zu mitigieren?

Um die oben angeführten Fragen beantworten zu können, wurde der Deebot Slim 2 wie auch dessen Applikation analysiert. In weiterer Folge wird ein Überblick der Gliederung dieser Arbeit sowie den darin beinhalteten Arbeitsschritten gegeben.

1.2 Gliederung

Dieses Dokument ist in mehrere Teile unterteilt, beginnend mit einer Einführung in das Thema (1. Einleitung), seine allgemeinen Probleme und bereits geleistete Arbeiten (2. Stand der Forschung) dazu. Weiters werden in 3. Grundlagen erklärt, die für das Verständnis dieser Arbeit benötigt werden. Im Kapitel 4. Methodik werden die generelle Herangehensweise und die verwendeten Methodiken beschrieben. Vor dem Start des Reverse Engineerings wird in 4.2 Threat Model ein Threat Model erstellt, in dem aufgrund der zur Verfügung stehenden Komponenten und Einsatzzwecke des Gerätes, Schwachpunkte klassifiziert und priorisiert werden. Bei der Osint Analyse in 4.3 OSINT Analyse werden öffentlich verfügbare Ressourcen verwendet, um bereits identifizierte Schwachstellen zu finden oder Informationen, welche diese Identifizierung erleichtern. In den darauffolgenden zwei Kapiteln wird die App des Deebot Slim 2, „Ecovacs Home“, und anschließend der Saugroboter selbst reverse-engineered und wird versucht, lokale Zugriffspunkte zu finden. Am Ende der beiden Kapitel wird es einen Abschnitt 6. Diskussion geben in dem die wichtigsten Ergebnisse und dessen Bedeutung zusammengefasst werden. Das letzte Kapitel 7. Conclusio schließt die Arbeit ab und gibt einen Überblick über die gewonnenen Erkenntnisse dieser Arbeit. Noch zu leistende Arbeit wird im Kapitel 8. Weiterführende Arbeiten erwähnt und noch offene weiterführende Fragestellungen werden diskutiert.

2. Stand der Forschung

Vor der Durchführung der Sicherheitsanalyse, wurde eine generelle Literaturrecherche zu dem Thema gemacht, um ein Bild von der aktuellen Lage der existierenden Ressourcen und Werkzeuge zu bekommen. In der Studie [4] aus dem Jahr 2017 wurde nicht versucht ein spezielles Gerät zu exploiten, sondern so viele IoT-Geräte wie möglich auszunutzen. Dafür wurden hauptsächlich Tools wie Shodan [5] und Nestle verwendet. Shodan scannt regelmäßig das offene Internet für verfügbare offene Ports, die beispielsweise von einem IoT-Gerät geöffnet wurden. Es konnte aufgezeigt werden, dass sehr viele IoT-Geräte einfach zu kompromittieren und sensible Userdaten gefährdet sind.

Es gab bereits einige Saugroboter, die untersucht wurden, wie zum Beispiel der Dongguan Diqee 360 [6], der Xiomi Roboter [7] und der Ironpie M6 [8]. In all diesen Arbeiten konnten Sicherheitsschwachstellen von allen Schweregraden erkannt werden. Besonders beim Ironpie M6 konnten einige sehr hohe CVSS-Schwachstellen gefunden werden. Die hauptsächlichen Problemquellen waren dabei die Android App und das von dem Saugroboter verwendete MQTT-Protokoll. In den Arbeiten wurden keine gesamten Analysen aller Komponenten der Saugroboter gemacht, sondern meist ein Teil wie zum Beispiel Hardwareschwachstellen behandelt.

Im Bereich Android App Analyse konnten sehr viele Quellen gefunden werden [9] [10]. Dadurch, dass Android Geräte sehr leicht zu rooten sind, können Android Apps einfacher getestet werden. Die Hersteller versuchen dem mit unterschiedlichen Detection- oder Prevention-Methoden entgegenzuwirken. So werden für Emulatoren oft Checks eingebaut, um das Verhalten der App bei der Erkennung zu verändern oder Netzwerkverkehr nur über von der App genehmigten Zertifikaten durchzuführen [11]. In [3] und [12] wurde ein Guide für IoT Analysen gegeben.

Auch konnte einiges zum Thema IoT Netzwerkanalyse, Protokollanalyse und Cloudhacking gefunden werden. In [13] wurde der Netzwerkverkehr zwischen Roboter und Cloud analysiert, um anschließend mehre Angriffsversuche sowohl gegen Cloud als auch Roboter durchzuführen. In Zuge dessen wurden einige Schwachstellen gefunden. Protokollanalysen von IoT Protokollen wurden in [14] durchgeführt. In der Arbeit wurde das MQTT-Protokoll analysiert und festgestellt, dass die Authentifizierung optional und Verschlüsselung nicht standartmäßig aktiviert ist.

Im Bereich Hardware Hacking war es etwas schwieriger Quellen zu finden. Die meisten dieser Quellen bestanden aus Blogs wie [15] [16] [17] oder Büchern, die sehr praktisch orientiert waren. Im Großteil der Quellen werden Angriffe über UART, JTAG oder SPI beschrieben und durchgeführt. In [15] wurde die Zerlegung, das Analysieren des Boards und der Angriff der Schnittstellen aufgezeigt. Das UART Interface wurde in [15] [16] ausgenutzt, um die Firmware vom Gerät zu extrahieren. Wie es mit Flash Chips möglich ist die Firmware zu extrahieren wurde in [17] an dem Beispiel von zwei Geräten aus CTF Challenges aufgezeigt. In beiden Fällen konnte die Firmware erfolgreich entnommen werden.

3. Grundlagen

3.1 IoT

IoT steht für Internet of Things und ist meist ein geschlossenes Gerät ohne Bildschirm, das Zugang zum Internet benötigt. Solche Geräte haben in letzter Zeit an Bedeutung gewonnen und ihre Zahl ist rapide gestiegen. IoT zielt darauf ab, alle Geräte mit dem Internet zu verbinden und so den Prozess zu vereinfachen, Daten zu sammeln und zu verarbeiten, die von verschiedenen Geräten wie Sensoren gesammelt werden. Ein weiterer Grund für den Anschluss von Geräten an das Internet, ist die Möglichkeit, einige Geräte aus der Ferne zu steuern. Die Reichweite von „intelligenten" Geräten wird ständig erweitert und es werden laufend neue Geräte für immer mehr Märkte entwickelt, wie zum Beispiel intelligente Kühlschränke, intelligente Staubsauger etc. Dies hat jedoch seinen Preis, da einige Geräte persönliche und in manchen Fällen private Daten benötigen, um wie vorgesehen zu funktionieren. Somit sind diese Daten für Dritte interessant und könnten von jemanden erlangt werden, wenn die Sicherheit des Geräts unzureichend ist [18]. Sicherheitsanalysen der Geräte können dabei helfen, diese Geräte in ihrer Sicherheit zu verbessern.

3.2 Man-in-the-Middle

Ein Man-In-The-Middle (MITM)-Angriff [19] ist eine wichtige Angriffstechnik beim computerbasierten Hacking, bei dem der Angreifer die Kommunikation eines Opfers abfängt. Eine der bekanntesten MITM-Methoden ist das Sniffing. Der Angreifer versucht dabei, die gesendeten Informationen zu sammeln und/oder zu verändern, wobei er sich dabei als der Empfänger ausgibt oder die Informationen zur späteren Verwendung speichert. Da es sich bei Wi-Fi-Verbindungen um indirekte Verbindungen handelt, sendet der Client seine Daten drahtlos und ohne Garantie, dass sie auch empfangen werden. Da die Daten über die Luft übertragen werden, kann jedes Gerät in der Nähe die drahtlosen Daten technisch abfangen. Daher sind MITM-Angriffe besonders effektiv, wenn sie auf die drahtlose Kommunikation abzielen [20].

3.3 Physische Debug Schnittstellen

Physische Debug Ports wie zum Beispiel UART, JTAG und SWD sind Diagnoseschnittstellen auf Chipebene mit denen in einem integrierten Schaltkreis, Konfiguration, Fehlersuche und Systemprogrammierungen durchgeführt werden können. Diese Schnittstellen sind für die Entwicklung von IoT-Geräten von großer Bedeutung. Im Endprodukt werden solche Schnittstellen aus Sicherheitsgründen deaktiviert, um Reverse Engineering und Firmware Extraktion zu erschweren. Eine Deaktivierung ist auch im Sinne der Spionage notwendig.

3.3.1 UART

Universal Asynchronous Receiver / Transmitter ist eine serielle Schnittstelle, wo immer ein Bit nach dem anderen übertragen wird. UART besteht aus 4 Pins, dem Ground-Pin, VCC-Pin (3,3 oder 5 Volt), TX-Pin und RX-Pin. Um mit einer UART-Schnittstelle kommunizieren zu können, werden Transmitter (TX) und Receiver (RX) Pins benötigt. Weiters gibt es eine Transaktionsrate, welche mit einem Logic Analyser berechnet werden kann [3]. Programmierer wie der USB Programmer CH431a [21] oder Bus Pirate[22] [23] können für das Debuggen verwendet werden.

3.3.2 JTAG

Joint Test Action Group, kurz JTAG, ist eine Schnittstelle für das Debugging eines integrierten Schaltkreises. Die Schnittstelle verbindet sich zu dem Test Access Port (TAP), der aus fünf Datenleitungen besteht. Data Input, Output, Clock und Mode Select sind dabei essenziell. Optional kann auch der Reset-Pin verwendet werden. JTAG muss von dem SoC unterstützt werden [12].

3.3.3 SWD

Der Serial Wire Debug Port ist ein alternativer Port zu JTAG, der das Debugging von PCB Boards möglich macht und ist ein für die ARM Architektur spezifisches Protokoll [3]. Oftmals werden diese Ports noch bei unfertigen PCB Boards eingesetzt, um diese zu testen. Die Schnittstelle hat die bidirektionale Leitungen SWD-Input/Output (SWDIO) und SWD-Clock (SWDCLK). Weiters gibt es die unidirektionale Leitung SWD-Output (SWDO) [12]. Für den Zugriff auf die Debug Schnittstelle können Chips verwendet werden, wie der Bus Blaster [24], der Segger J-Link [25] oder der sehr günstige ST-Link v2.

3.4 SPI-Chip

Das Serial Peripheral Interface ist ein Protokoll, um den Austausch von Daten zwischen dem Mikrokontroller und anderer Peripherie zu ermöglichen [26]. Es wird meist an Orten verwendet, wo die Lese- und Schreibgeschwindigkeit wichtig ist, wie zum Beispiel bei Bildschirmen, SD Karten Lesern oder Ethernet Schnittstellen [3]. Mit dem Debug Programmer CH431A und einem SOIC Clip kann der Speicher eines SPI-Chips ausgelesen werden.

3.5 Reverse Engineering

Reverse Engineering ist ein Prozess, bei dem versucht wird, die Aufgabe eines Gerätes oder einer Software zu rekonstruieren. Beim Reverse Engineering eines Gerätes wird die Hardware und deren Bausteine untersucht. Beim Software Reverse Engineering wird der Code der Software analysiert. Bei

beiden Verfahren sollen die Komponenten in einer Art höheren Abstraktion dargestellt werden, um mit der Darstellung ein einfacheres Verständnis für den Aufbau dieser Komponenten zu bekommen [27].

3.6 Codeanalyse

Bei der Codeanalyse werden zwei Methoden angewendet. Bei der statischen Analyse wird der Programmcode nicht ausgeführt, sondern es wird der Quellcode oder eine Form des Objektcodes analysiert. Die zweite Methode ist die dynamische Analyse, bei der das Verhalten im ausgeführten Zustand analysiert wird. Aufgrund des enormen Aufwandes der beiden Analysemethoden gibt es unterstützende Werkzeuge, die diese Untersuchungen automatisiert durchführen können. Anschließend müssen diese Ergebnisse in einem „Code-Review“ interpretiert und überprüft werden [27]. Infolgedessen, dass diese Werkzeuge nicht immer fehlerfrei, alle neuen Erkenntnisse überprüfen oder nicht für diese Produkte abgestimmt sind, müssen einige Tätigkeiten weiterhin von „menschlicher Hand“ durchgeführt werden.

3.7 Threat Modeling

Da IT-Security immer mehr an Bedeutung gewonnen hat, wurden viele verschiedene Modelle definiert und erstellt, um die Sicherheit von Produkten zu analysieren. Dies wird als Threat Modeling bezeichnet und ist die systematische Identifizierung und Priorisierung, basierend auf ihrem Schweregrad, bestimmter Probleme, um Risiken innerhalb eines Produktes zu reduzieren [28]. Es kann mit manuell oder automatisierten Tools durchgeführt werden. Beim Threat Modeling gibt es software-, asset- oder angreifer-zentrierte Ansätze, die von verschiedenen Tools präferiert werden [12]. So wird beim software-zentrierten Threat Modeling die Anfälligkeit der einzelnen Anwendungskomponenten priorisiert, während einerseits beim asset-zentrierten Ansatz zuerst die für das System wichtigen Daten identifiziert und andererseits beim angreifer-zentrierten Ansatz die potenziellen Angreifer identifiziert werden. Einige manuelle Ansätze wären die Verwendung von STRIDE[29], PASTA[30], Trike[31] oder OCTAVE[32]. Einige automatisierte Ansätze sind die Verwendung von VAST[33] oder TVA-Tool[34].

3.7.1 Stride

Stride fokussiert sich auf die Erkennung von Schwachstellen in der Technologie. Das Modell wurde von Praerit Garg und Loren Kohnfelder mit Microsoft entwickelt. Folgende Bedrohungsklassen wurden dafür definiert:

- **S**poofing (Identitätsverschleierung): Wenn ein Akteur die Rolle eines Systems, Benutzers oder von Komponenten vorspielt.
- **T**ampering (Manipulation): Wenn die Integrität von Daten oder eines Systems verletzt werden.

- **R**epudiation (Verleugnung): Aktionen, die getan, abgelehnt oder verworfen werden.
- **I**nformation disclosure (Verletzung der Privatsphäre): Die Vertraulichkeit der Systemdaten ist verletzt.
- **D**enial of service (Verweigerung des Dienstes): Wenn die Verfügbarkeit eines Systems angegriffen wird.
- **E**levation of privilege (Rechteausweitung): Wenn Benutzer oder Komponenten Rechte erlangen können, welche sie nicht erreichen sollten.

Diese Bedrohungsklassen werden auf die vorab definierten Assets mit ihren Use Cases angewendet [29].

3.7.2 Dread

Eine Bedrohung braucht eine Auswirkung, um eine Gefahr darzustellen. Um die genaue Auswirkung der gefundenen Bedrohungen festzustellen, kann ein Bedrohungsbewertungsmodell, wie Dread [35], verwendet werden. Es besteht aus fünf verschiedenen Faktoren, welche einzeln mit Zahlen zwischen 0 und 10 bewertet und anschließend summiert werden. Die errechnete Zahl kann für die Risikobewertung verwendet werden, um in weiterer Folge festzustellen, welche Bedrohungen schneller behoben werden sollten. Diese fünf Faktoren sind [3]:

- **D**amage (Schaden): Wie viel Schaden wird durch das Ausnutzen dieser Schwachstelle verursacht?
- **R**eproducibility (Reproduzierbarkeit): Wie einfach ist es, die Ausnutzung zu reproduzieren?
- **E**xploitability (Ausbeutbarkeit): Wie einfach kann die Schwachstelle ausgenützt werden?
- **A**ffected Users (Betroffene Benutzer): Wie viele Benutzer wären davon betroffen?
- **D**iscoverability (Entdeckbarkeit): Wie einfach ist es, die Schwachstelle zu identifizieren?

4. Methodik

4.1 Flow Charts

Die in Abbildung 1: Methoden App Analyse, Abbildung 2: Methoden Netzwerk IoT, Abbildung 3: Methoden Hardware Analyse gezeigten Methoden sind aus dem Stand der derzeitigen Literatur abgeleitet. Die Applikations-, Netzwerk- und Hardwaremethoden können je nach IoT-Gerät unterschiedlich sein und in beliebiger Reihenfolge angewendet werden. Zum Beispiel könnte eine Netzwerkanalyse des Saugroboters bereits während der Applikationsanalyse Sinn machen. Sollte bereits Firmware vorhanden sein oder mehrere physische Geräte für die Hardwareanalyse verfügbar sein, um die Hardware zu extrahieren, könnte eine Firmwareanalyse vorab zu Erkenntnissen für den weiteren Verlauf der anderen Analysen führen. Aufbauend auf diesem Prinzip gründet sich die Reihenfolge der angewendeten Methoden.

Die Flow Charts sollen als Leitfaden für die Analyse anderer IoT-Geräte dienen und aufbauend auf ihnen werden in diesem Kapitel Methodiken für den Leitfaden beschrieben. Erklärungen zu den Flow Charts werden im Kapitel 3. Grundlagen und 4. Methodik gegeben und in 6. Diskussion dieser Arbeit besprochen.

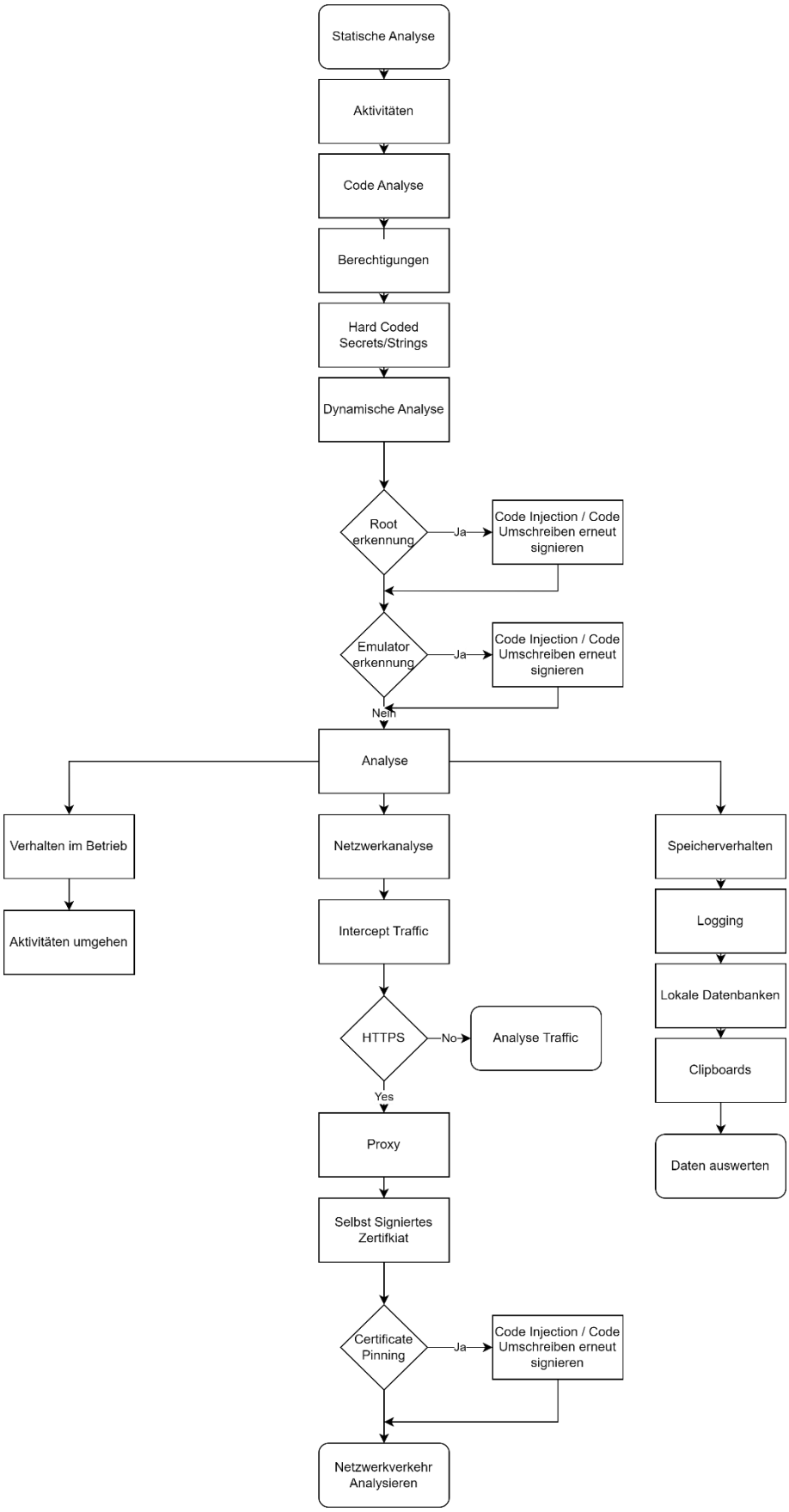


Abbildung 1: Methoden App Analyse

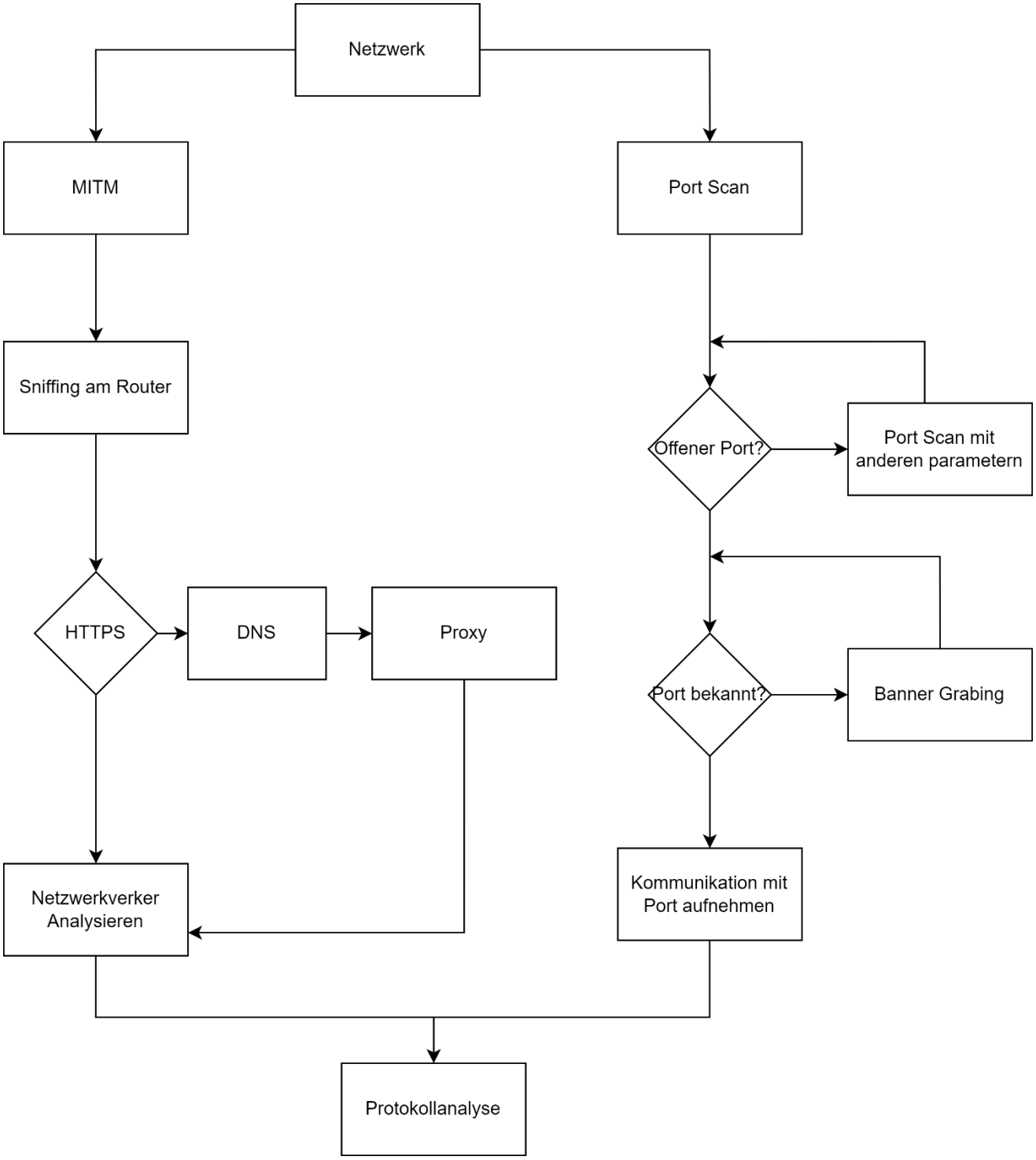


Abbildung 2: Methoden Netzwerk IoT

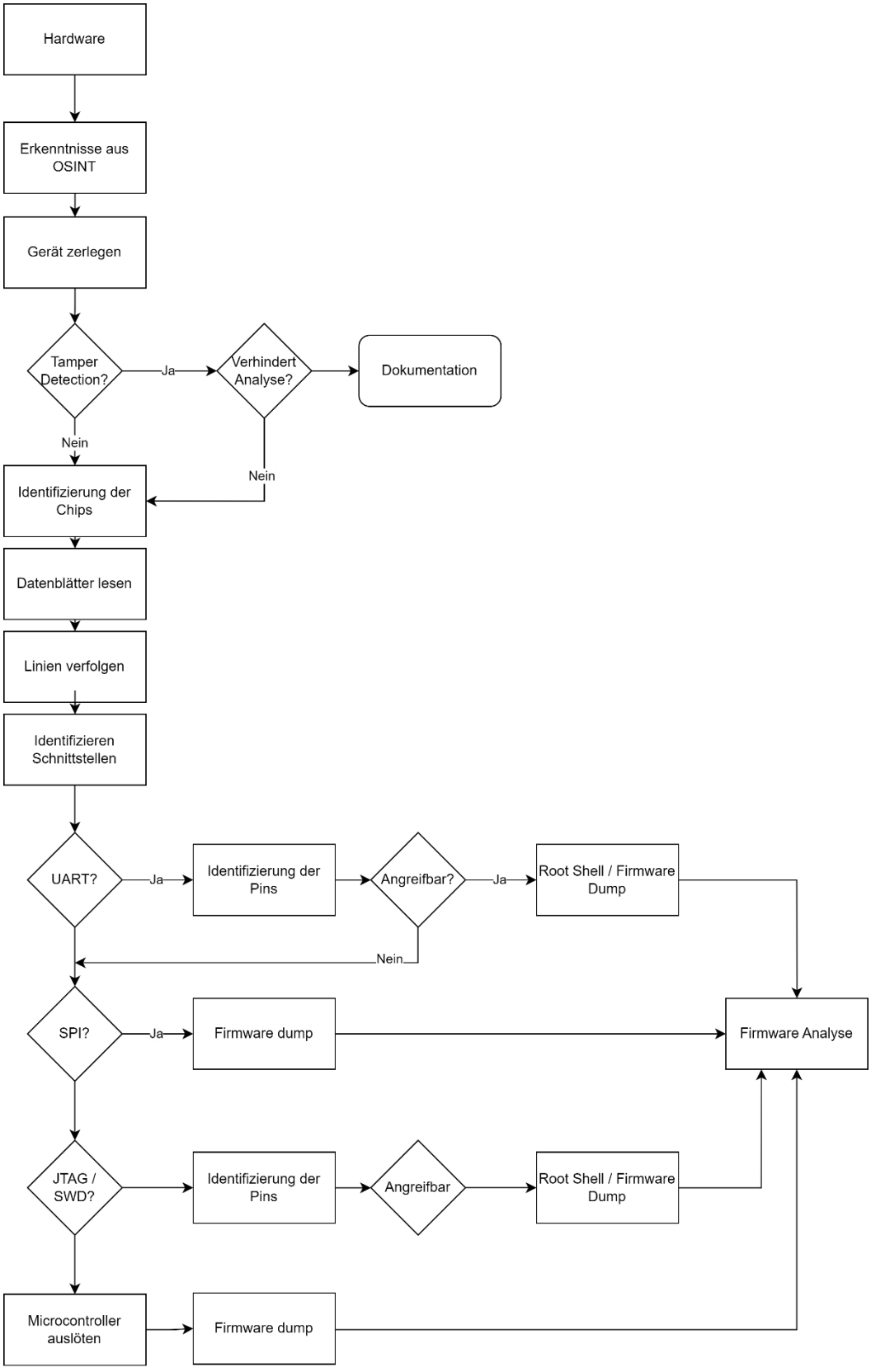


Abbildung 3: Methoden Hardware Analyse

4.2 Threat Model

In diesem Projekt werden die manuellen Methoden STRIDE und DREAD aufgrund ihrer Einfachheit und Effizienz bei der Identifizierung von Bedrohungen in kleineren IoT-Ökosystemen angewendet [3].

Threat Modeling wird in dieser Arbeit in folgende vier Teile unterteilt [36]:

- Angreifbare Komponenten feststellen
- Use Cases dieser Komponenten definieren
- STRIDE Modell anwenden
- DREAD Modell anwenden

4.3 OSINT Analyse

In dieser Arbeit wird Open-Source Intelligence (OSINT) dazu verwendet, die spätere Analyse zu unterstützen oder zu erleichtern. Dazu wären folgende Informationen für den späteren Verlauf der Analyse interessant:

- Bereits gefundene Schwachstellen in der Applikation oder im Saugroboter
- Aufbau und/oder Zerlegungsvorgang des Saugroboters
- Schnittstellen
- Verbaute Komponenten
- Öffentliche Passwortlisten oder kryptographische Schlüssel

Dafür wurden Suchmaschinen wie Google oder baidu [37] verwendet, da Ecovacs ein chinesischer Saugroboter Hersteller ist [38]. Weitere Quellen waren:

- Herstellerdownloads
- Wissenschaftliche Arbeiten
- Öffentliche Code Repositorien (Pastebin)
- Communities (Reddit)

4.4 Applikation

Applikationen sind für die Steuerung von IoT-Geräten meist unverzichtbar. Zumindest für die Ersteinrichtung der Geräte und um den vollen Leistungsumfang nutzen zu können, wird ein Smartphone mit der zugehörigen Applikation benötigt. Eine allgemeine Sicherheitsanalyse der App sollte bei einem Penetrationstest eines IoT-Gerätes enthalten sein [12]. Als grober Leitfaden kann dafür der „OWASP Mobile Application Security Verification Standard MASVS“ [39] verwendet werden. Zumal dieser Leitfaden für alle Applikationen und nicht speziell für IoT-Geräte geschrieben wurde, sollte dieser Leitfaden an die IoT Welt angepasst sein. Zum Beispiel wird im MASVS Bluetooth Pairing, welches für IoT-Geräte häufig verwendet wird, nicht thematisiert. Die in diesem Kapitel beschriebenen Methodiken wurden aus den Erkenntnissen von Abbildung 1: Methoden App Analyse entnommen.

4.4.1 Statische Analyse

Das Ziel der statischen Untersuchung ist es, wichtige Informationen für den weiteren Verlauf der Analyse herauszufinden. Erkenntnisse in der statischen Analyse können seltsames Verhalten, welches beispielsweise durch eine Emulator Detection ausgelöst wird, in der dynamischen Analyse erklärbar machen und diese Detektionsmechanismen vorab erkennen [40]. Dafür wird mit einem automatisierten Tool wie dem Mobile-Security-Framework MobSF [41] eine vorab Untersuchung durchgeführt. Anschließend werden im Zuge eines „Code Reviews“ die wichtigen Ergebnisse der automatischen Analyse interpretiert und auf Basis dieser, weitere Erkenntnisse aus dem Quellcode gezogen. Sollte ein Detektionsmechanismus in die Applikation eingebaut sein, müssen mögliche Umgehungen dieser gefunden werden, damit eine dynamische Analyse möglich ist. Für Android Geräte kann die .APK Datei über die Adroid Debug Bridge[42] erhalten werden. Die meisten Applikationen sind auch öffentlich über Drittanbieter zum Download verfügbar. Bei iOS kann das .IPA Paket über das Programm iMazing [43] von einem emulierten oder physischen Gerät extrahiert werden. Mittels Patching könnte diese Datei für eine dynamische Analyse vorbereitet werden.

Fragen, die in der statischen Analyse für den Deebot Slim 2 beantwortet werden sollten:

- Wie verbindet sich die App mit dem Staubsauger?
- Welche Funktionen können erkannt werden?
- Welche Daten werden über die App abgerufen?
- Gibt es Detektionsmechanismen, die eine dynamische Analyse erschweren könnten? (Root/ Emulator Detection, SSL Pinning)
- Gibt es Datenbanken, die angreifbar sind?

Um diese Fragen beantworten zu können müssen im Applikationscode gesetzte Berechtigungen und nach möglichen „Hard Coded Secrets“ gesucht werden.

4.4.2 Dynamische Analyse

Bei der dynamischen Analyse sollte, aufbauend auf der statischen Analyse, mehr über die App und ihr Zusammenspiel mit dem Saugroboter herausgefunden werden. Dafür müssen eine Netzwerkanalyse, das Speicherverhalten und die Zugriffsrechte geprüft werden. Sollten weitere Funktionen, wie zum Beispiel Bluetooth, verwendet werden, kann auch dieser Verkehr mit der entsprechenden Aktivierung in den Entwickleroptionen mitgeschnitten werden. Die dynamische Analyse könnte vorab automatisiert mit dem Werkzeug Medusa durchgeführt werden [44].

4.4.3 Statisches und Dynamisches Patching

In der statischen Analyse sollten Erkenntnisse gewonnen werden, um eine mögliche Root/Emulator Detection zu erkennen. Durch die Suche von Strings „Emulator“ oder „root“, können Funktionen, wie zum Beispiel showRootStatus() oder checkEmulatorStatus(), vorab erkannt werden und anschließend mittels statischem oder dynamischem Patching umgangen werden [3].

Beim statischen Patching wird der Applikationscode umgeschrieben und dann erneut kompiliert [45]. Ein Werkzeug, das dazu verwendet wird, ist Baksmali[46], ein assembler/disassembler für das dex Format. Im Gegensatz dazu wird beim dynamischen Patching, welches zuverlässiger ist als das statische Patching, der Code bei der Ausführung verändert. Dafür bringt Frida [47] alle nötigen Mittel mit. Mit Frida kann Java Script Code direkt in die App injiziert werden und damit können Bedingungen im Code verändert oder übersprungen werden [9]. Mithilfe des dynamischen Patchings können auch Funktionsaufrufe, die zu einem Certificate Pinning führen, umgangen oder umgeschrieben werden.

4.5 Saugroboter

Die meisten IoT-Geräte bringen sehr viele Funktionen und Sensoren mit sich und daher auch einige Angriffspunkte. Zu diesen Punkten zählen:

- Hardware
- Firmware
- Netzwerk
- Radio

Beim Hardwareteil wird versucht unterschiedliche Debug Schnittstellen oder Speichereinheiten anzugreifen, um dort beispielsweise Firmware oder andere Daten zu bekommen. Mit der extrahierten Firmware können anschließend mehr Angriffsflächen durch deren Analyse im Netzwerkteil gefunden werden oder falls dies auch möglich ist, die Firmware manipuliert werden und erneut auf das Gerät abgelegt werden [48]. Sollte eine Firmwareextraktion nicht über die Hardware möglich sein, kann über das Netzwerk im Zuge eines Updates die neueste Firmware Version abgefangen werden[49]. Bei IoT-Geräten werden Protokolle wie, XMPP, Zigbee oder Z-Wave verwendet. Nicht alle dieser Protokolle sind standardmäßig „sicher“ konfiguriert bzw. überhaupt geschützt [50]. Beim Radio Hacking werden andere Kommunikationsinstrumente, wie Bluetooth, RFID LoRaWAN oder auch Wifi, angegriffen.

4.5.1 Netzwerkverkehr des Gerätes

Die in diesem Kapitel beschriebenen Methodiken wurden aus den Erkenntnissen von Abbildung 2: Methoden Netzwerk IoT entnommen. Für die Analyse des Netzwerkverkehrs muss eine Aufzeichnung durchgeführt werden. Dafür bieten sich je nach Anwendungszweck mehrere Möglichkeiten. Die Aufzeichnung kann direkt auf dem Router mittels dem Werkzeug Wireshark erfolgen. Mit Wireshark können Datenprotokolle angezeigt und analysiert werden [51]. Durch diese Art der Aufzeichnung können Pakete, die verschlüsselt verschickt werden, allerdings nicht im Detail betrachtet werden. Um dies bewerkstelligen zu können, kann ein Proxy, wie Burp Suite [52], verwendet werden. Mit Burp Suite kann ein Zertifikat erstellt werden, das auf dem Gerät, welches überwacht wird, installiert werden muss. Somit kann die Verschlüsselung mittels des Zertifikats am Proxy aufgebrochen und eine Deep Packet Inspection gemacht werden. Diese Methode wird Man-in-the-Middle genannt [53].

4.5.1.1 Offene Ports

Um bestimmte Funktionen erfüllen zu können, haben IoT-Geräte lokal freigeschaltene Ports, über welche die Kommunikation mit dem Gerät möglich ist. Bei manchen Geräten ist diese Kommunikationen nicht geschützt und/oder keine Authentifizierung notwendig. Beispielhafte Ports hierfür wären Telnet und ssh, welche einen direkten Zugriff auf die Geräte zulassen würden. Um Ports identifizieren zu können, kann das Werkzeug Nmap eingesetzt werden. „Nmap ist ein Portscanner zum Scannen und Auswerten von Hosts in einem Rechnernetz.“ [54] Wenn ein offener Port identifiziert

werden kann, aber der Service hinter dem Port nicht erkannt wird, kann Banner Grabbing angewendet werden [55].

4.5.1.2 Banner Grabbing

Banner Grabbing ist der Vorgang in dem bei einem Computersystem in einem Netzwerk der Dienst hinter einem offenen Port bestimmt wird [55]. Es wird zwischen aktiven und passiven Banner Grabbing unterschieden. Unter aktiven Banner Grabbing versteht man, wenn eine Verbindung mit dem Port hergestellt wird, zum Beispiel mit Tools, wie Telnet, Netcat oder wget, und mit verschiedenen Kommandos versucht wird, eine Antwort von dem Port zu erzeugen. Bei diesem Vorgang können, wie in [3] beschrieben, .xml Files als Antwort von dem Port zurückkommen. Beim passiven Banner Grabbing wird der Netzwerkverkehr aufgezeichnet und die vom Port retournierten Pakete werden analysiert [56].
Die einfachste Art herauszufinden, welcher Service sich hinter einem Port versteckt, ist mittels eines nmap Scans. Mit „–version-all“ wird der Port auf bekannte Signaturen geprüft.

4.5.1.3 Denial-of-Service

Der Denial-of-Service Angriff (DoS) ist einer der einfachsten Formen von Netzwerkangriffen. Bei einem DoS Angriff wird im Gegensatz zu den bisher genannten Methoden nicht versucht Informationen zu stehlen, sondern es wird schlicht versucht, den Dienst nicht verfügbar zu machen. Dabei können zwei Formen unterschieden werden, solche, bei denen der Dienst zum Absturz gebracht wird und solche, bei denen der Dienst mit Anfragen überflutet wird, damit er die Warteschlange nicht mehr abarbeiten kann [57].

4.5.2 Protokoll Analyse

Netzwerkprotokolle sind ein häufig übersehenes Gebiet beim Testen von IoT-Geräten. Aufgrund der Tatsache, dass diese Geräte meistens andere Protokolle als gängige Netzwerkgeräte verwenden, um miteinander zu kommunizieren, können Schwachstellen, die innerhalb der Protokolle auftreten, gefunden werden. Eine Schwachstelle in einem Protokoll kann gravierenden Sicherheitsproblemen für das Gerät auslösen, wie am Beispiel des KNOB-Angriffs [58], bei dem eine Schwachstelle im Bluetooth Protokoll gefunden wurde.

Einige IoT-Referenzarchitekturen sind:

- IoT-A
- IEEE Draft Standard for an Architectural Framework for the IoT
- Industrial Internet Reference Architecture (IIRA)

4.5.2.1 Fuzzing

Fuzzing oder auch Fuzz-Testing ist eine Methode, bei der Anwendungen, Betriebssysteme oder Netzwerkprotokolle mit zufälligen Daten überflutet werden, um seltsames Verhalten oder einen Absturz zu erzwingen. Diese Methode wird meist automatisiert ausgeführt [59]. Dafür gibt es beispielsweise Werkzeige wie zfuzz [60], welche speziell für Zigbee abgestimmt wurde. Für memory corruptions kann IOTFuzzer[61] eingesetzt werden.

4.5.2.2 Bluetooth LE

Bluetooth LE oder auch Bluetooth 4.0 sowie neuere Versionen sind Drahtlosübertragungsprotokolle für die Kommunikation zwischen zwei Geräten. Bei der Verbindung von Geräten gibt es Phasen, die erfüllt werden müssen, um einen Datenaustausch zu ermöglichen.

- Sichtbarkeit
- Suche nach anderen Geräten
- Verbindungsanfrage

In der ersten Phase muss ein Gerät für das andere Gerät sichtbar gemacht werden. Anschließend wird auf dem nicht sichtbaren Gerät die Suche nach sichtbaren Geräten gestartet, worauf eine Verbindungsanfrage und Antwort folgen sollten. Seit der Version 4.2 gibt es die Möglichkeit der verschlüsselten Kommunikation mittels „LE Legacy Paring “ oder „LE Secure Connection“. Die beiden Methoden unterscheiden sich in der Generierung und Übertragung der Schlüssel [12]. Je nach gewählter Bluetooth Version und dem gewählten Verbindungstyps kann aktives oder passive Abhören, mit beispielsweise einem Ubertooth One [62], durchgeführt werden.

4.5.2.3 Zigbee

Zigbee ist ein sehr altes Protokoll, welches in der IoT Welt aber dennoch sehr gängig ist. Bei diesem Protokoll können mehrere Geräte miteinander über sogenannte Knotentypen kommunizieren. Dafür müssen diese vorher aber erst dem Netzwerk beigetreten sein. Das Beitreten eines Zigbee Netzes mit einem Endgerät erfolgt mit dem Netzwerk Key, welcher Broadcast Nachrichten innerhalb des Netzwerkes verschlüsselt, dem Line Key, der für Unicast-Nachrichten zwischen zwei Geräten verwendet wird und dem Master Key, der den Austausch der Line-Keys durch ein Symmetric-Key Key Establishment Protocol (SKKE) sichert [12].

Durch die Gegebenheiten des Protokolls und dessen Alter gab es bereits einige Sicherheitstests [59]. Das Zigbee Protokoll ist meist besonders anfällig auf Replay-Angriffe und Angriffe auf den Netzwerkschlüssel, welcher unverschlüsselt zu den Geräten gesendet wird.

4.5.2.4 MQTT

Das Message Queue Telemetry Transport Protocol kurz MQTT ist ein von IBM entwickeltes Protokoll für Machine-to-Machine Kommunikation, das von der Organization for Advanced Structured Information Standards (OASIS) [63] als Open Source Protokoll weiterentwickelt wurde [12]. Es basiert auf einer Publish-Subscribe Architektur, welche aus drei Komponenten besteht [64]:

- MQTT-Broker – koordiniert die Kommunikation zwischen den Clients
- Publisher Client – sendet Daten an den Broker
- Subscribe Client – empfängt Daten vom Broker

Schwachstellen bei MQTT sind der Port des Brokers, standartmäßig 1883 [65], können aber auch die Wiederholungen bestimmter Bits, Stack Overflow Angriffe oder Fuzzing sein [64].

4.5.3 Hardware

Die in diesem Kapitel beschriebenen Methodiken wurden aus den Erkenntnissen von Abbildung 3: Methoden Hardware Analyse entnommen.

4.5.3.1 Zerlegung des Gerätes

Bei der Zerlegung eines Gerätes, welches später analysiert werden soll, muss besonders auf Tamper Prevention Mechanismen geachtet werden. Durch Tamper Prevention, die in besonders sicherheitskritischen Geräten, wie Sicherheitskameras oder Türschlössern, zum Einsatz kommt, kann ein Gerät für die spätere Analyse unbrauchbar gemacht werden.

Es kann zwischen Tamper Prevention, dem erschweren der Manipulation der Hardware, und Temper Detection, dem erkennen von Manipulation, unterschieden werden. Eine Maßnahme kann auch beide Methoden bedienen. Die gängigsten Methoden, zu denen hierbei gegriffen wird, sind:

Siegel und Plomben: Durch Siegel und Plomben kann eine Veränderung nicht verhindert werden. Sie dienen der Erkennung von Eingriffen in die Hardware.

Verklebungen: Oftmals besonders bei Smartphones kommt es zu Verklebungen des Gehäuses. Solche Verklebungen lassen sich meist leicht, aber mit größerem Aufwand lösen. Es können sich Farbkugeln in der Klebespur befinden, um eine Manipulation zu erkenn, da auch diese Methode eher zur Erkennung oder als Garantieschutz, anstatt als Erschwernis, genutzt wird.

Spezialschrauben: Spezialschrauben sind Schrauben, die keinen gängigen Aufsatz besitzen oder sich nicht aufdrehen lassen. Ersteres lässt sich nur mit dem richtigen Werkzeug öffnen. Anders hingegen ist der Fall mit Schrauben, die sich nicht aufdrehen lassen. Diese können durch Aufbohren, Fräsen einer Nut in den Kopf der Schraube oder starker Gewalteinwirkung geöffnet werden.

Verschweißungen: Kunststoffverschweißungen lassen sich kaum zerstörungsfrei öffnen. Eine Manipulation ist somit garantiert zu erkennen. Außerdem benötigt es erhöhten Aufwand, die Verschweißung zu lösen.

Temper Detection Mechanismen: Bei Sicherheitsanlagen werden besondere Temper Detection Mechanismen eingebaut. Diese können zum Beispiel in Form eines Mikroschalters eingebaut sein, welcher bei Öffnung des Geräts eine Funktion im Gerät auslöst. [12]

4.5.3.2 PCB-Board - Chips und sonstige Schnittstellen

4.5.3.2.1 Identifizierung der Chips

Um die Chips des jeweiligen PCB Boards zu identifizieren, müssen die Chips auf dem Board untersucht werden. In den meisten Fällen steht die Nummer des jeweiligen Chips auf der Oberseite.

Bei einigen Geräten können aber zu deren Sicherheit Methoden angewendet werden, um diese Nummern zu entfernen. Zu den Methoden zählen:

- Abschleifen
- Übermalen
- Harzeingießen
- Metallblenden

Übermalte Bezeichnungen können oft sehr leicht mit Aceton wieder freigelegt werden. Auch die Metallblenden sind einfach zu entfernen. Sollte allerdings der Chip abgeschliffen worden sein, ist eine Identifizierung nur mehr über die Messung der Pins zu bewerkstelligen.

4.5.3.2.2 Identifizierung der Pins am Beispiel von UART und SWD

Bei der Identifizierung der Pins gibt es mehrere Möglichkeiten die Kombiniert werden können:

- Verfolgung über die Linien ausgehend vom Mikrokontroller
- Multimeter
- Logic Analyzer

Für die Identifizierung der Pins und Schnittstellen wurden folgende Werkzeuge verwendet:

- Bus Pirate [22]
- ST-Link v2 [66]
- CH341a Programmierer

4.5.3.2.2.1 Multimeter

4.5.3.2.2.1.1 Ground Pin

Mit dem Multimeter kann die Kontinuität geprüft werden. Es kann gegen ein Metallstück, das geerdet ist, oder einen anderen Ground-Pin, falls bereits identifiziert, mit der schwarzen Sonde getestet werden. Wenn ein „Piepsen“ durch das Anhalten der roten Sonde auf einen der Pins durch das Multimeter zu hören ist, kann dieser als Ground-Pin identifiziert werden. Auch mit Resistenzmessung kann der Ground-Pin erkannt werden, da bei dieser Messung gegen einen anderen Ground-Pin oder geerdetes Metallstück ein konstanter Wert von 0 aufscheinen sollte.

4.5.3.2.2.1.2 VCC Pin

Um den VCC-Pin zu ermessen, muss vorab das Multimeter auf Volt Messung umgestellt werden. Dadurch kann gegen einen geerdeten Pin oder Metall gemessen werden. Anschließend sollte das Gerät eingeschalten werden. Um den Pin zu erkennen, sollte das Multimeter 3,3 Volt oder 5 Volt konstant anzeigen.

4.5.3.2.2.1.2 RX- und TX-Pin

Für die RX- und TX-Pins kann das Gerät und das Multimeter in demselben Zustand verweilen, wie bei der VCC-Messung. Der RX-Pin sollte konstant 0 Volt anzeigen. Da dieser auf den Empfang von Daten wartet. Der TX-Pin sollte beim Start eines Gerätes Debug Informationen senden. Daher sollte dieser durch eine Float Voltanzeige erkennbar sein.

4.5.3.2.2.2 Logic Analyzer

Mit dem Logic Analyzer können digitale Signale von und zu den verschiedenen Pins gemessen und interpretiert werden. Er ist ein ausgezeichnetes Werkzeug, um unbekannte Pins auf einem PCB zu identifizieren

4.5.3.2.2.2.1 UART Baud-Rate Kalkulation

Die Baud-Rate ist ein Terminus, der verwendet wird, um die Geschwindigkeit der Datenübertragung zu messen. Sie wird in Element pro Sekunde gemessen [67]. Errechnet wird die Baud-Rate mit der Formel:

$$Baud\ Rate = \frac{Element}{Sekunden} [3]$$

Mit dem Logic Analyzer können die am RX- und TX-Pin ankommenden Daten eingelesen werden. Wenn die Daten eingelesen wurden, kann durch das genauere Analysieren der Breite eines ankommenden Elements die Baud-Rate errechnet werden.

In Abbildung 4 mit 8,333 Mikro Sekunden zu sehen wird die Baud-Rate dann mit:

$$\frac{1}{8{,}333} = 0{,}1200005$$

Die Standard-Rate, die dieser Berechnung am nächsten ist, ist 115200 [3].

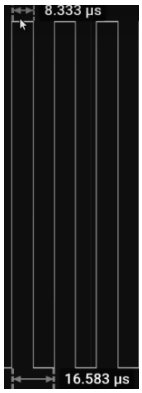


Abbildung 4: Logic Analyzer TX Daten

Automatisch kann die Baud-Rate zum Beispiel von dem Logic Analyzer von Saleae oder mit dem Buspirate und der Verwendung des Baud-Rate Scriptes [68] bestimmt werden [12].

4.7 Firmware Reverse-Engineering

Ein wichtiger Teil bei der Sicherheitsanalyse eines IoT-Gerätes ist die Firmware. Die Firmware ist Software, die meist in einem Flash-Speicher, EPROM oder ROM gespeichert ist und direkt mit der Hardware kommuniziert [3]. Einige Schwachstellen, die dabei entdeckt werden können, werden in [69] aufgelistet. Diese Probleme können meist in anderen Teilen der Analyse ausgenutzt werden oder mehr Informationen dafür liefern. Firmware ist normalerweise in folgenden Formaten anzutreffen:

- Archivformate (Zip, Tar, CPIO),
- Dateisystemformate (SquashFS, JFFS2, YAFFS),
- Binärformate (ELR, PE) [12].

Diese speziellen Dateisystemformate werden oft bei IoT-Geräten aufgrund von Ressourcenknappheit verwendet. Die einfachsten Methoden, um an die Firmware eines Gerätes zu gelangen, sind bei der OSINT-Analyse, durch Extrahieren auf der Hardware oder das Mitlesen eines Firmware Updates des Gerätes. Wenn eine Firmwaredatei erfolgreich erhalten werden konnte, muss diese zunächst entpackt werden.

4.7.1 Entpacken der Firmware

Vor der Analyse sollte der Entropiewert der Firmware überprüft werden, um eine Verschlüsslung und/oder Komprimierung der Datei feststellen zu können. Das Werkzeug binwalk [70] kann dabei mit der Option -E einen Entropie Check durchführen und mit der Option -e das Dateisystem extrahieren. Mit -M können Dateitypen durch ihre Muster automatisch erkannt werden. Sollten mit binwalk interessante komprimierte Daten gefunden werden, können diese mit dd [71] und dem richtigen Offset extrahiert werden, um sie anschließend mit der richtigen Archivierungssoftware zu entpacken.

4.7.2 Statische Analyse

Bei der statischen Analyse wird versucht, in der entpackten Firmware Schwachstellen in Konfigurationsdateien, Applikationscode oder Binärdateien zu identifizieren. Von besonderem Interesse sind hierbei:

- Zugangsdaten für Telnet, SSH oder Weboberflächen
- /etc/passwd und /etc/shadow
- SSL-Zertifikate
- API- oder SSH-Schlüssel [12]

Vorab kann mit firmwalker [72] eine oberflächliche automatisierte Analyse gemacht werden. Meistens können dabei bereits einige interessante Erkenntnisse gezogen werden. Mit find [73] und und grep [74] kann das Dateisystem durchsucht werden. Ein Hash aus einer Datei könnte mit Hashcat [75] wiederhergestellt werden. Mit Tools wie IDAPro [76] oder radare [77] können Schlüssel aus Verschlüsselungsmethoden ausgelesen werden [12].

4.7.3 Emulation

Um eine dynamische Analyse der Firmware durchzuführen, sollte die Firmware erst emuliert werden. Für die Emulation können Werkzeuge wie QEMU [78] oder auch Emulatoren mit automatisierten dynamischen Analysen wie Firmware Analysis Toolkit [79] oder FIRMADYNE [80] verwendet werden.

4.7.4 Dynamische Analyse

Bei der dynamischen Analyse können bei erfolgreicher Emulation alle Funktionen des Gerätes getestet werden. Dateien, die erst nach dem Startvorgang erstellt werden, sollten untersucht werden. Der Netzwerkverkehr und offene Ports können mit netstat [81] direkt auf dem Gerät überprüft werden. Die Firmware kann auch manipuliert und anschließend auf das Zielgerät mit einer Backdoor entweder durch Zugriff auf die Hardware oder durch den Updatemechanismus aufgespielt werden [3].

5. Ergebnisse

5.1 Threat Model

5.1.1 Komponenten

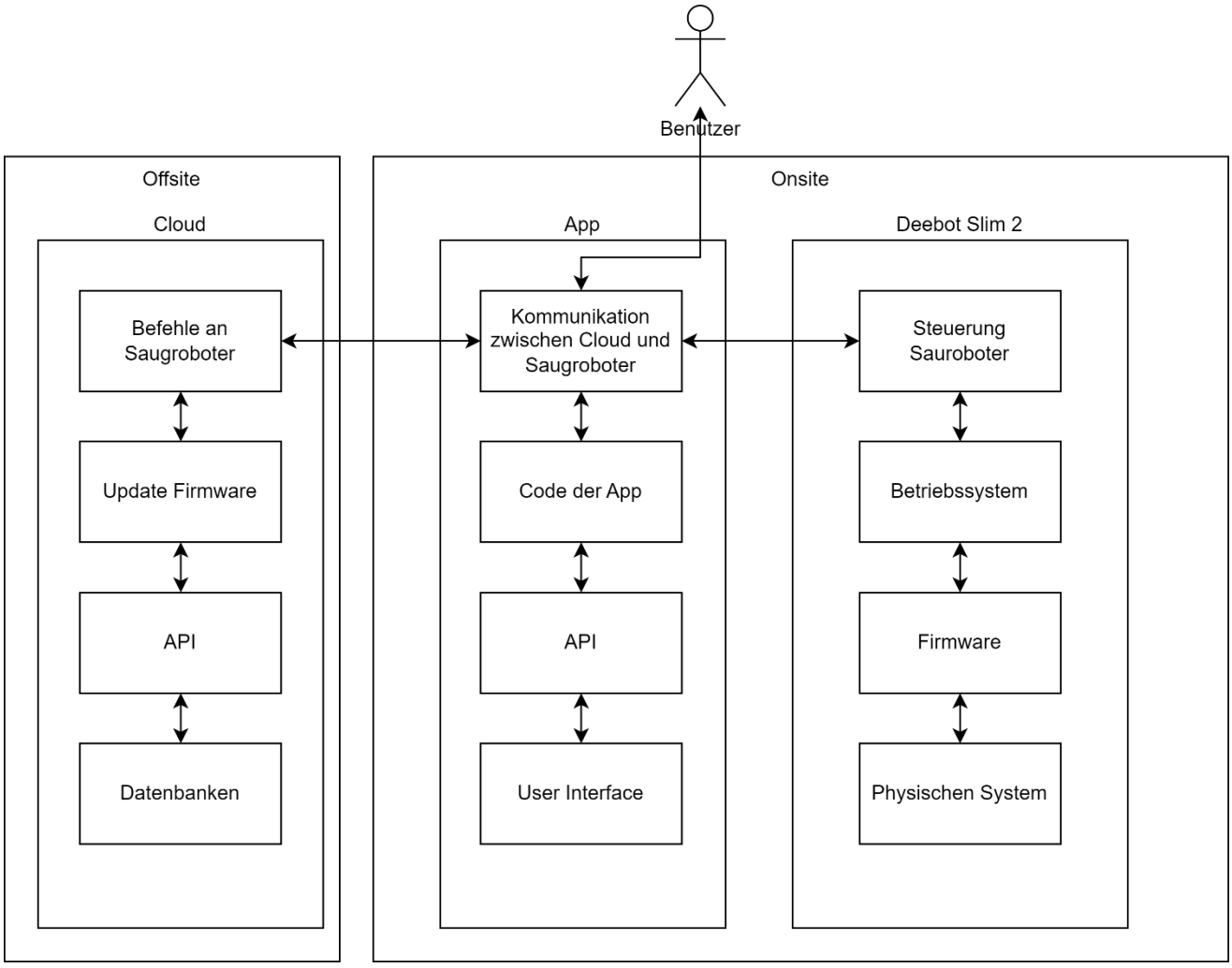


Abbildung 5: Komponenten

In Abbildung 5 sind die Assets für die volle Funktionsweise des Saugroboters abgebildet. Der Saugroboter enthält die Hardware (Steuerung), das Betriebssystem, die Firmware und die physische Ebene. Befehle werden von der Cloud über die App an den Saugroboter weitergegeben. Wichtige Updateaufrufe aus Datenbanken kommen aus der Cloud und werden über die App an den Saugroboter weitergegeben.

5.1.2 Use Cases

Use Case 1: Den Saugroboter mit der App verbinden.

1 Evocas Mobile App herunterladen
2 Einen Account mit E-Mail und Passwort registrieren
3 Mit den Zugangsdaten einloggen
4 Den Deebot Slim 2 im Menü auswählen
5 Beim Deebot Slim 2 zehn Sekunden auf den WIFI Button drücken
6 Verbindung in der App initialisieren

Bei den Use Cases 2 bis 4 werden die Schritte 1-6 aus Use Case 1 wiederholt.

Use Case 2: Saugen, Wischen, Laden oder selbst mit den Pfeiltasten in der App Steuern

1 Deebot Slim 2 im Menü auswählen
2 Saugen, Wischen, Laden oder selbst Steuern auswählen

Use Case 3: Manuel ein Firmware Update auslösen

1 In der App unter „Mehr – Deebot Info – Firmware Version“
2 Update auswählen

Use Case 4: Passwort ändern

1 In den Einstellungen der App auf „Kennwort ändern“
2 Altes und Neues Kennwort eingeben

Use Case 5: Passwort zurücksetzen

1 Beim Login auf Passwort zurücksetzen
2 E-Mail Adresse eingeben
3 Bestätigungslink in der E-Mail folgen
4 Neues Passwort eingeben

Use Case 6: Saugen, Wischen, Laden oder Steuern über die Fernbedienung

1 Über die Fernbedienung den jeweiligen Modus auswählen

Use Case 7: Wifi zurücksetzen

1 Den Auto Knopf gedrückt halten, bis ein Ton zu hören ist
2 Das WIFI Led sollte in regelmäßigen Abständen aufleuchten
3 Schritte 1-6 aus Use Case 1

Use Case 8: Intelligenten Zeitplan erstellen

1 Den „Auto“ Knopf zweimal Drücken zu der geplanten Zeit

5.1.3 Stride

Bei der aus Tabelle 1 angewendeten Stride Methode wurden verschiedene Angriffsszenarien für den Deebot Slim 2 und die Ecovacs Applikation überlegt. Es wurde versucht, aus jeder in Stride definierten Kategorie ein Beispiel zu finden. Um Szenarien zu finden, wurden die bereits in dem vorherigen Kapitel definierten Komponenten und Use Cases eingesetzt. Zusätzlich wurden Gegenmaßnahmen für die Angriffsvektoren definiert.

Tabelle 1: STRIDE

Nummer	Threat Beschreibung	Threat Ziel	Angriffstechnik	Gegenmaßnahmen
1 Spoofing	Angreifer kann Netzwerkverkehr mitlesen	App	MITM	Einsatz von SSL oder TLS und Nonces.
2 Information Disclosure	Persönliche Daten können ausgelesen werden	App	Unsichere Datenspeicherung	Verschlüsselte Abspeicherung der Daten
3 Spoofing	Angreifer bekommt Zugriff auf die Firmware durch ein Update	Saugroboter	DNS und Arp Spoofing des Saugroboters	Einsatz von Verschlüsselung und einer Root CA
4 Tampering	Root Zugang durch Hardware	Saugroboter	Angriffe auf die Hardware-Integrität	Eine Sicherung, die bei Zerlegung das PCB Board zerstören würde
5 Tampering	Manipulation Firmware	Saugroboter	Firmware Reverse Engineering	Firmware nicht zugänglich machen
6 Information Disclosure	Zugriff auf persönliche Daten die auf der Hardware gespeichert sind	Saugroboter	Angriffe auf die Hardware (UART,JTAG etc.)	Tamper Protection
7 Denial of Service	Hohe Anzahl an Kommandos an den Saugroboter schicken	Saugroboter	Denial of Service (Dos)	Keine Zugangspunkte offen lassen
8 Elevation of Priviledge	Root Zugang durch einen offenen Port	Saugroboter	Port Scanning und Banner Grabbing	Keine offenen ungeschützten Ports

5.1.4 Dread

Der letzte Schritt des Threat Modelings ist, die aufgedeckten Schwachstellen zu priorisieren und zu evaluieren. In Tabelle 2: DREAD wurden die aus Tabelle 1: STRIDE gefunden Angriffsszenarien auf einer Skala von eins bis zehn bewertet. Ein höherer Wert bedeutet eine größere Auswirkung auf die Sicherheit des Saugroboters.

Tabelle 2: DREAD

Threat	D	R	E	A	D	Gesamt
Angreifer kann Netzwerkverkehr mitlesen	7	7	7	3	7	31
Persönliche Daten können ausgelesen werden	5	4	5	3	6	23
Angreifer bekommt Zugriff auf Firmware durch ein Update	9	7	4	5	5	30
Root Zugang durch Hardware	10	2	2	2	2	18
Manipulation Firmware	10	3	3	3	3	22
Zugriff auf Persönliche Daten die auf der Hardware gespeichert sind	5	3	3	8	3	22
Denial of Service: Hohe Anzahl an Kommandos an den Saugroboter schicken	3	10	9	6	10	38
Root Zugang durch einen Offenen Port	10	6	4	3	4	27

5.2 Flow Charts

Aufbauend auf dem Threat Model und den Kapitel 4.1 Flow Charts ergeben sich für den Deebot Slim 2 die Flow Charts aus Abbildung 6: Applikationsanalyse Deebot Slim 2, Abbildung 7: Netzwerkanalyse Deebot Slim 2, Abbildung 8: Hardware Analyse Deebot Slim 2.

Weil für die Analyse nur ein Deebot Slim 2 zur Verfügung stand und nach der Hardwareanalyse eine einwandfreie Funktion des Saugroboters möglicherweise nicht mehr gegeben wäre, mussten vor der Hardwareanalyse die Applikationsanalyse und der Netzwerkverkehr des Saugroboters behandelt werden.

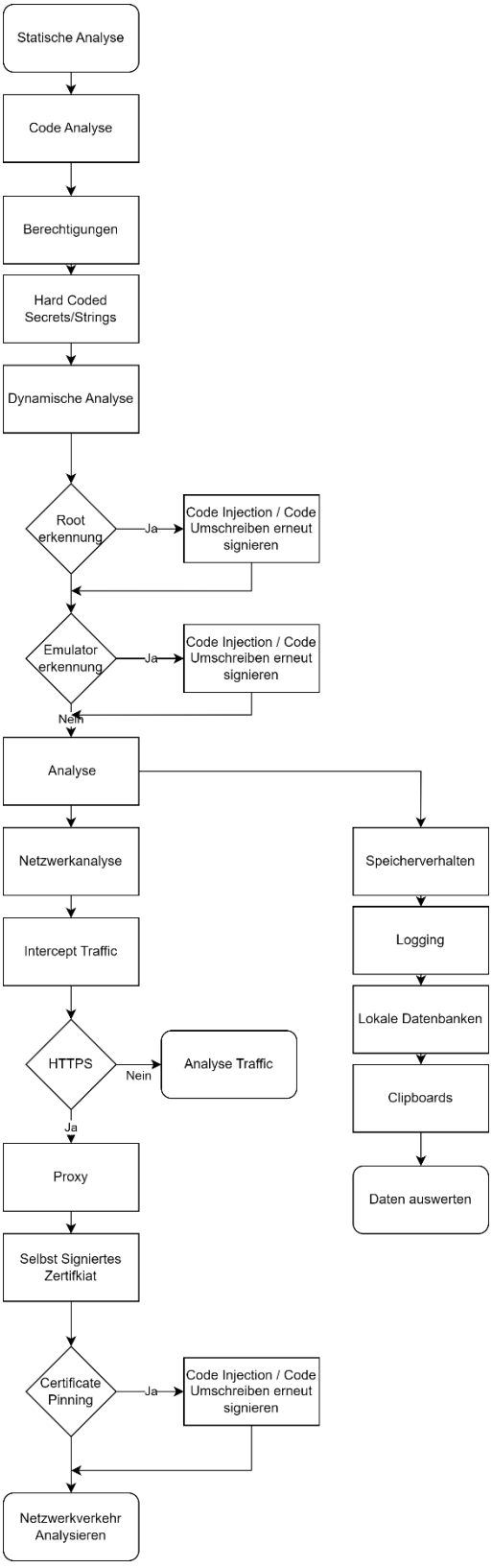


Abbildung 6: Applikationsanalyse Deebot Slim 2

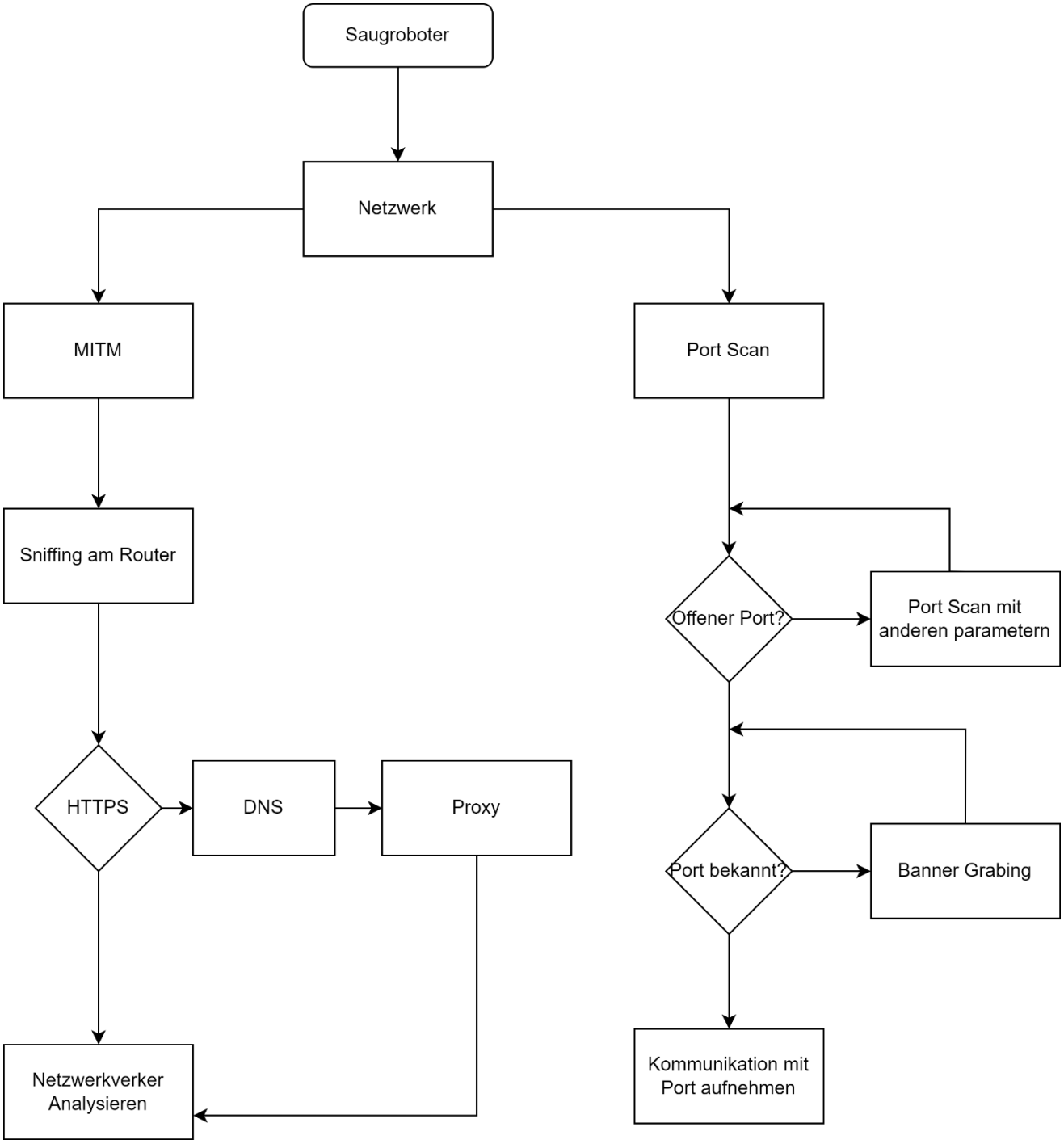


Abbildung 7: Netzwerkanalyse Deebot Slim 2

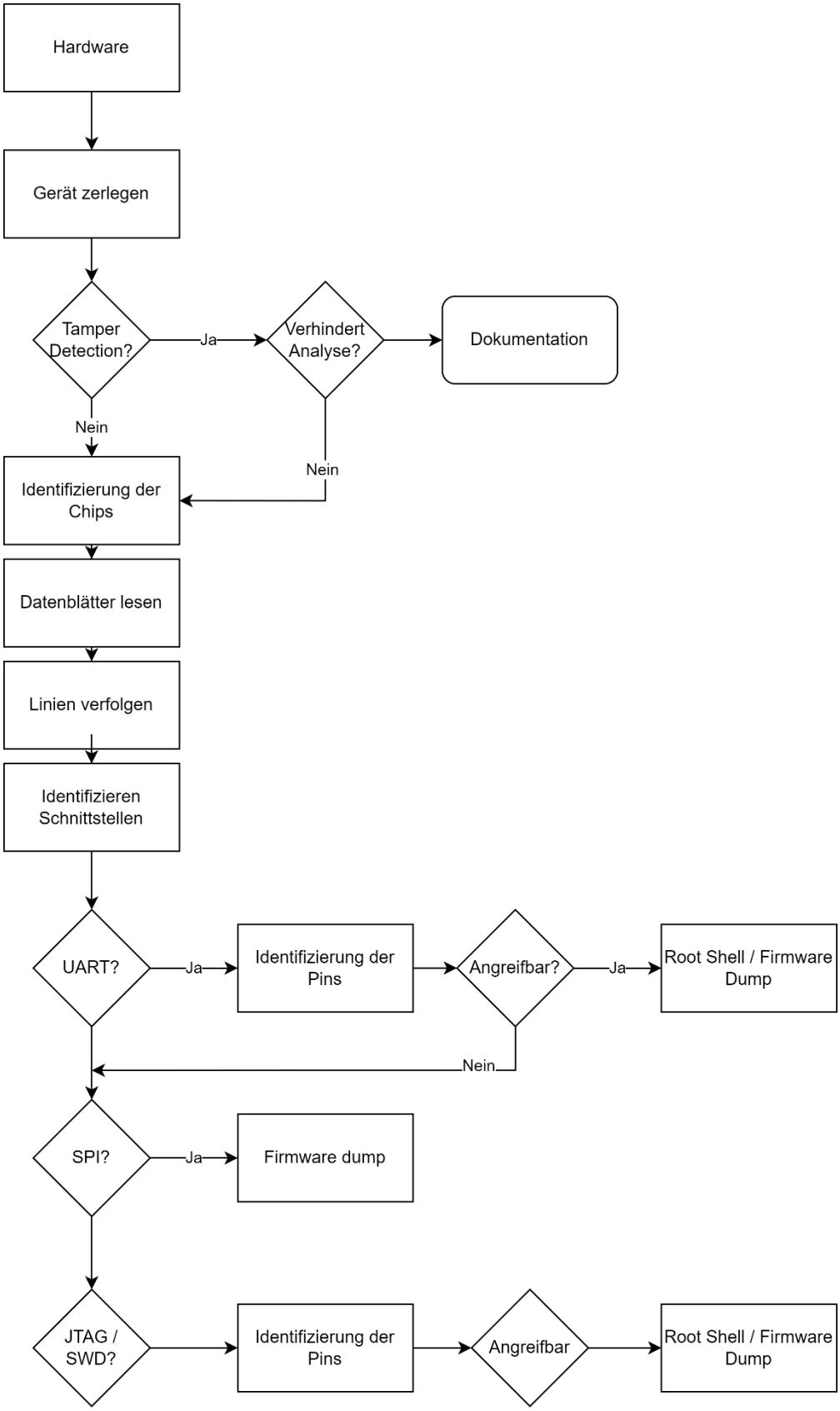


Abbildung 8: Hardware Analyse Deebot Slim 2

5.3 Osint Analyse

Bei der OSINT Analyse konnte über die Zerlegung des Gerätes eine Anleitung gefunden werden, in welcher der Deebot Slim 2 im Reparaturvorgang zerlegt wurde und auf den Bildern mögliche physische Sicherheiten erkannt werden können [82]. In [83] wurden die Protokolle von mehreren Deebot Saugrobotern untersucht. Der Deebot Slim 2 wurde dabei jedoch nicht untersucht. Sollte eines der Protokolle in der späteren Analyse entdeckt werden, kann dies als Hilfestellung verwendet werden. Es konnten alle Datenblätter zu den verbauten Komponenten gefunden werden. Unter Anwendung von „Google Hacking“ und der chinesischen Suchmaschine baidu [37] konnte keine Firmware Datei gefunden werden. Der Deebot Slim 2 wurde bis jetzt in keiner wissenschaftlichen Arbeit oder einem öffentlichen Blog erwähnt. Für die Analyse der Applikation konnte die APK-Datei von der folgenden Seite heruntergeladen werden [84].

5.4 Applikation

5.4.1 Statische Analyse

Um die aus dem Kapitel „Methodik 4.4.1 Statische Analyse“ genannten Fragen beantworten zu können müssen folgende Komponenten der App genauer untersucht werden:

- Berechtigungen
- Hard Coded Secrets
- Applikationscode

Es wurde vorab mittels MobSF [41] die Analyse automatisiert durchgeführt.

5.4.1.1 Berechtigungen

Die App benötigt einige erhöhte Rechte. Das Lesen und Schreiben von externen Storage GPS Locations, mount und unmount Dateisystemen und Telefonnummern, Serialnummern und Rufnummern. Außerdem werden Daten in einem Clipboard abgelegt und Dateien so abgespeichert, dass sie für andere Apps erreichbar wären. Bei der dynamischen Analyse könnte ein Backup gemacht werden, um dann diese abgelegten Daten zu überprüfen.

5.4.1.2 Hard Coded Secrets

Es konnten keine Hard Coded Secrets in der App identifiziert werden. Es gab einige Hard Coded http URLs im Code. Interessant war zum Beispiel folgender XMPP Client, der im Source Code zu finden ist:

```
private static final String ENTITY_NODE =
"http://www.igniterealtime.org/projects/smack";
```

5.4.1.3 Code Analyse

Im Code konnten Anzeichen für eine Root oder Emulator Detection gefunden werden. Als Datenbank wurde Firebase entdeckt. Manche Firebase Datenbanken sind schlecht konfiguriert. Ein Testen der Datenbank ist im Zuge der Arbeit ohne rechtliche Erlaubnis nicht möglich. Im AndroidManifest.xml wurden einige Aktivitäten festgestellt, welche den „launchmode=singleTask“ gesetzt haben. Diese Konfiguration kann zu einem Task Hijacking durch andere Malicious Apps führen [85].

5.4.1.3.1 Root Detection

Es wurde eine Root Detection in der App gefunden. Es wird überprüft, ob die Datei Superuser.apk, welche nur auf einem gerooteten Android-Gerät zur Verfügung stehen sollte, existiert.

```
private static boolean c() {
        try {
            if (new File("/system/app/Superuser.apk").exists()) {
                return true;
```

Somit könnte bei Existenz dieser Datei die App eine dynamische Analyse erschweren oder ihr eigenes Verhalten verändern. Aus diesem Grund müssen später in der dynamischen Analyse Gegenmaßnahmen hierfür ergriffen werden. Für die Root Detection Bypass wurde das Script aus [86] verwendet.

5.4.1.3.2 Emulator Detection

Mit dem Tool MobSF konnten mehrere Anzeichen für eine Emulator Detection in der App erkannt werden. Es werden einige Funktionen in unterschiedlichen Files aufgerufen, die einen Emulator erkennen sollen.

- Build.FINGERPRINT check
- Build.MANUFACTURER check
- Build.HARDWARE check
- Build.BOARD check
- possible Build.SERIAL check
- Build.TAGS check
- SIM operator check
- network operator name check
- subscriber ID check

Infolgedessen, dass ein Android Gerät für die dynamische Analyse verwendet wird, werden keine Gegenmaßnahmen für diese Checks ergriffen. Für die Emulator Detection könnte jedoch das bestehende Script für Frida um den Bypass für die Emulator Detektionsmechanismen erweitert werden [9].

5.4.1.3.3 SSL Konfiguration

Es wurde ein Trust Manager aufgesetzt, um keinen selbstsignierten Zertifikaten zu vertrauen. Aus dem „keystore“ werden die Zugangsdaten geladen, um sie über den Trust Manager zu übertragen.

```
public EasyX509TrustManager(KeyStore keyStore) throws
NoSuchAlgorithmException, KeyStoreException {
        this.standardTrustManager = null;
        TrustManagerFactory trustManagerFactory =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm())
;
```

```
        trustManagerFactory.init(keyStore);
        TrustManager[] trustManagers =
trustManagerFactory.getTrustManagers();
        if (trustManagers.length == 0) {
            throw new NoSuchAlgorithmException("no trust manager found");
        }
        this.standardTrustManager = (X509TrustManager) trustManagers[0];
    }
```

Mehrmals konnten im Code Misskonfigurationen entdeckt werden, welche die App anfällig für MTIM-Angriffe machen könnte. In dem Beispiel wurde der setHostnameVerifier=ALLOW_ALL_HOSTNAME_VERIFIER gesetzt [87].

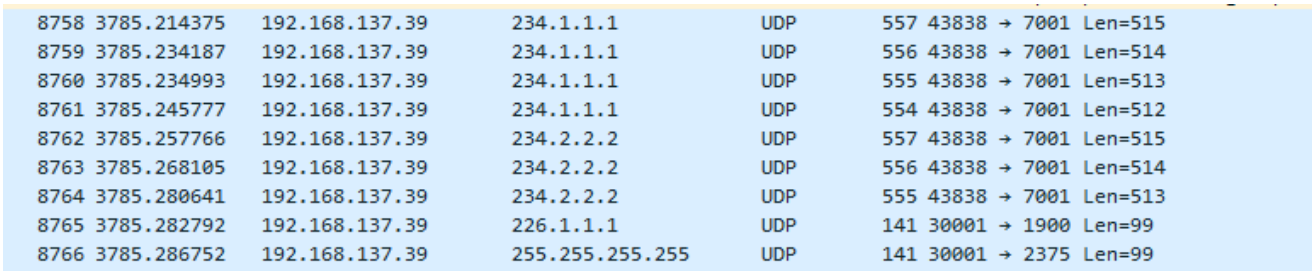
```
SSLSocketFactory.getSocketFactory();

sSLSocketFactory.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_V
ERIFIER);
```

5.4.2 Dynamische Analyse

Folgende Punkte wurden bei der dynamischen Analyse betrachtet:

- Netzwerkverkehr
- Speicherverhalten
- Systemfunktionsaufrufe

5.4.2.1 Netzwerkanalyse der App



8758	3785.214375	192.168.137.39	234.1.1.1	UDP	557 43838 → 7001 Len=515
8759	3785.234187	192.168.137.39	234.1.1.1	UDP	556 43838 → 7001 Len=514
8760	3785.234993	192.168.137.39	234.1.1.1	UDP	555 43838 → 7001 Len=513
8761	3785.245777	192.168.137.39	234.1.1.1	UDP	554 43838 → 7001 Len=512
8762	3785.257766	192.168.137.39	234.2.2.2	UDP	557 43838 → 7001 Len=515
8763	3785.268105	192.168.137.39	234.2.2.2	UDP	556 43838 → 7001 Len=514
8764	3785.280641	192.168.137.39	234.2.2.2	UDP	555 43838 → 7001 Len=513
8765	3785.282792	192.168.137.39	226.1.1.1	UDP	141 30001 → 1900 Len=99
8766	3785.286752	192.168.137.39	255.255.255.255	UDP	141 30001 → 2375 Len=99

Abbildung 9: Wireshark

Die erste Verbindung der App mit dem Gerät geschieht über UDP-Multicast. 192.168.137.39 ist in Abbildung 9 das Smartphone und sendet UDP-Pakete auf Multicast Adressen aus. Zu sehen ist der Datenaustausch von der Cloud über die Applikation an den Saugroboter.

Abbildung 10: Packet Capture

In Abbildung 10 wurde versucht, den Traffic mit einem einfachen Packet Capture [88] mitzulesen. Nach der Konfiguration des Proxys wurden Logins in der App gemacht. Sobald der Caputure aktiviert

wurde, konnte keine Verbindung der App mit dem Server hergestellt werden. Das durch den MITM Proxy bereitgestellte Zertifikat wurde nicht akzeptiert. Die App, wie auch bereits in der statischen Analyse festgestellt, ist durch Certificate Pinning geschützt.

5.4.2.1.1 Certificate Pining

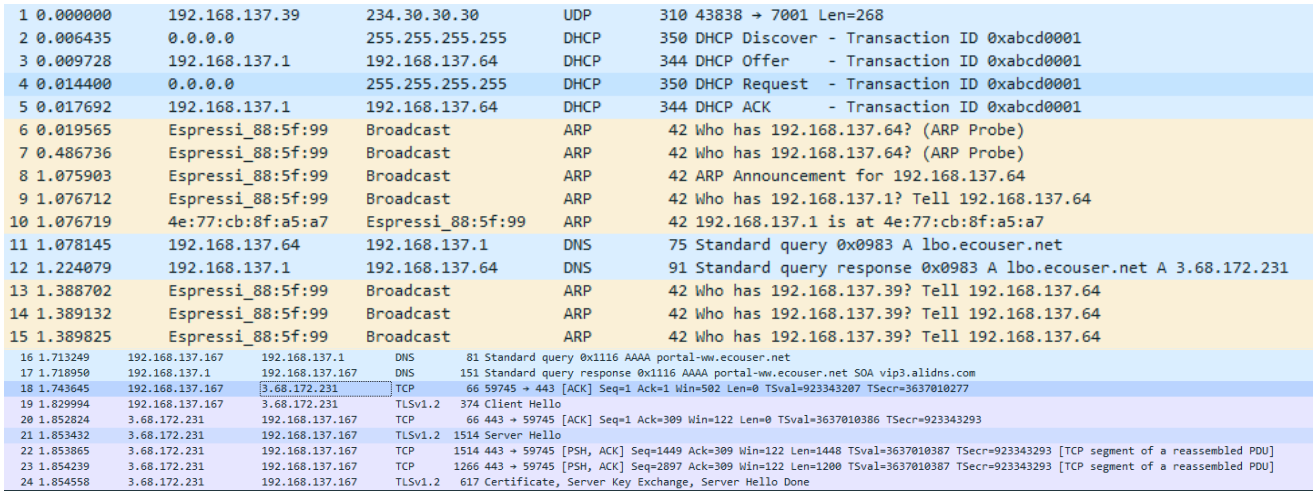
Ein altes Android Smartphone wurde mit LineageosOS [89] ausgestattet und gerooted und via adb [42] die notwendigen Werkzeuge darauf installiert. Auf dem Smartphone wurde Frida Server [47] und http Toolkit [90] installiert. Dadurch, dass das Telefon in einem gerooteten Zustand ist, konnte ein Zertifikat für den Man-in-the-Middle Proxy in den System Trust Store hinterlegt werden. Es ist nicht ausreichend, das Zertifikat als User zu hinterlegen. Nach der Eintragung des Proxys am Telefon waren Verbindungen der Ecovacs App nicht mehr möglich, weil die App mit Zertifikats Pinning ausgestattet ist. Um diese zu umgehen, wurden durch die statische und dynamische Analyse die Checks der App herausgefunden und diese mittels eines inject im Appendix angehängten Java Frida Script umgangen. Das bereits bestehende Javascipt aus [11] wurde verwendet und um die nötigen Bypasses erweitert. Es wurde erfolglos versucht den App Code umzuschreiben und erneut zu signieren, um das Certificate Pinning zu umgehen [91].

5.4.2.2 Speicherverhalten

Oft speichern Android Applikationen Geheimnisse in ihrem Verzeichnis, sobald sie ausgeführt werden. In einem nicht gerooteten Gerät wären diese Informationen nicht zugänglich. Es wurde mittels adb [42] eine komplette Sicherung von dem Gerät gezogen und diese mit dem Android backup extractor [92], der für die adb Sicherungsfunktion optimal genutzt werden kann, entpackt. Sowohl im Applikationsverzeichnis als auch in der Datenbank wurden keine Geheimnisse gefunden.

5.5 Saugroboter

5.5.1 Netzwerkverkehr des Saugroboters



1	0.000000	192.168.137.39	234.30.30.30	UDP	310 43838 → 7001 Len=268
2	0.006435	0.0.0.0	255.255.255.255	DHCP	350 DHCP Discover - Transaction ID 0xabcd0001
3	0.009728	192.168.137.1	192.168.137.64	DHCP	344 DHCP Offer - Transaction ID 0xabcd0001
4	0.014400	0.0.0.0	255.255.255.255	DHCP	350 DHCP Request - Transaction ID 0xabcd0001
5	0.017692	192.168.137.1	192.168.137.64	DHCP	344 DHCP ACK - Transaction ID 0xabcd0001
6	0.019565	Espressi_88:5f:99	Broadcast	ARP	42 Who has 192.168.137.64? (ARP Probe)
7	0.486736	Espressi_88:5f:99	Broadcast	ARP	42 Who has 192.168.137.64? (ARP Probe)
8	1.075903	Espressi_88:5f:99	Broadcast	ARP	42 ARP Announcement for 192.168.137.64
9	1.076712	Espressi_88:5f:99	Broadcast	ARP	42 Who has 192.168.137.1? Tell 192.168.137.64
10	1.076719	4e:77:cb:8f:a5:a7	Espressi_88:5f:99	ARP	42 192.168.137.1 is at 4e:77:cb:8f:a5:a7
11	1.078145	192.168.137.64	192.168.137.1	DNS	75 Standard query 0x0983 A lbo.ecouser.net
12	1.224079	192.168.137.1	192.168.137.64	DNS	91 Standard query response 0x0983 A lbo.ecouser.net A 3.68.172.231
13	1.388702	Espressi_88:5f:99	Broadcast	ARP	42 Who has 192.168.137.39? Tell 192.168.137.64
14	1.389132	Espressi_88:5f:99	Broadcast	ARP	42 Who has 192.168.137.39? Tell 192.168.137.64
15	1.389825	Espressi_88:5f:99	Broadcast	ARP	42 Who has 192.168.137.39? Tell 192.168.137.64
16	1.713249	192.168.137.167	192.168.137.1	DNS	81 Standard query 0x1116 AAAA portal-ww.ecouser.net
17	1.718950	192.168.137.1	192.168.137.167	DNS	151 Standard query response 0x1116 AAAA portal-ww.ecouser.net SOA vip3.alidns.com
18	1.743645	192.168.137.167	3.68.172.231	TCP	66 59745 → 443 [ACK] Seq=1 Ack=1 Win=502 Len=0 TSval=923343207 TSecr=3637010277
19	1.829994	192.168.137.167	3.68.172.231	TLSv1.2	374 Client Hello
20	1.852824	3.68.172.231	192.168.137.167	TCP	66 443 → 59745 [ACK] Seq=1 Ack=309 Win=122 Len=0 TSval=3637010386 TSecr=923343293
21	1.853432	3.68.172.231	192.168.137.167	TLSv1.2	1514 Server Hello
22	1.853865	3.68.172.231	192.168.137.167	TCP	1514 443 → 59745 [PSH, ACK] Seq=1449 Ack=309 Win=122 Len=1448 TSval=3637010387 TSecr=923343293 [TCP segment of a reassembled PDU]
23	1.854239	3.68.172.231	192.168.137.167	TCP	1266 443 → 59745 [PSH, ACK] Seq=2897 Ack=309 Win=122 Len=1200 TSval=3637010387 TSecr=923343293 [TCP segment of a reassembled PDU]
24	1.854558	3.68.172.231	192.168.137.167	TLSv1.2	617 Certificate, Server Key Exchange, Server Hello Done

Abbildung 11: Netzwerkverbindungen Saugroboter

Die initiale Einrichtung, wie in Abbildung 11: Netzwerkverbindungen Saugroboter zu sehen, passiert über Port 7001 Multicast UDP Adressen.

Nach erfolgreicher Einbindung in das lokale Netzwerk schickt das IoT Gerät DNS Abfragen für die IP 3.68.172.231 und macht daraufhin einen TLSv1.2 Key Exchange. Um eine korrekte TLS Implementierung des Gerätes zu testen, können die DNS Anfragen gespoofed werden und damit der Proxy mit dem manipulierten Zertifikat als der richtige Server validiert werden. Somit versucht der Saugroboter anschließend seine Verbindung zur Domain über den Proxy zu machen.

5.5.1.1 Man in the Middle

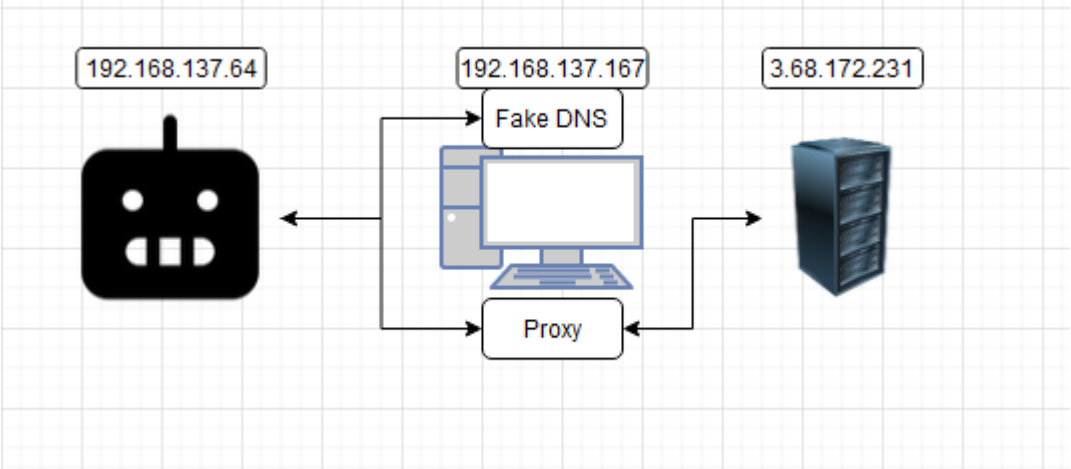


Abbildung 12: Setup Man-in-the-middle Saugroboter

Der Roboter schickt eine DNS Anfrage an den fake DNS Server für die Domain portal-ww-ecouser.net. Darauf wird der DNS Server mit der lokalen IP vom Computer antworten. Für den DNS Server wurde das Script aus [93] verwendet. Aufgrund der Tatsache, dass der PC auch als Router verwendet wird, können für zusätzliche Analysen alle Pakete mit Wireshark aufgenommen werden. Weitere Anfragen können dann direkt über den Proxy, in diesem Fall Burp, abgehandelt werden. Dafür muss Burp im „Invisible Proxying“ Modus konfiguriert werden [94]. Weiters wird in Burp ein Zertifikat mit einem Eintrag der Domain „portal-ww-ecouser.net“ erstellt [95]. Nach abschließender Konfiguration wurde mit dem Computer ein lokaler Hotspot erstellt und der Saugroboter mit diesem Netzwerk verbunden.

Es wurden keine Requests in Burp aufgezeichnet, was zur Schlussfolgerung führt, dass die TLS-Verifizierung korrekt in dem Gerät umgesetzt wurde. Dies kann auch über die aufgezeichneten Paket, wie in Abbildung 13 in Wireshark verifiziert werden.

```
TLSv1.2   1007 Server Hello, Certificate, Server Key Exchange, Server Hello Done
TLSv1.2     73 Alert (Level: Fatal, Description: Handshake Failure)
```

Abbildung 13: Handshake Failure

Damit kann gesagt werden, dass eine Liste mit vertrauten Root CAs in dem Gerät verfügbar ist. Diese Root CAs könnten, wenn eine Firmwareextraktion über die Hardware erfolgreich ist, später um das Zertifikat erweitert werden.

```
HTTP    195 GET /products/wukong/class/123/firmware/latest.json HTTP/1.0
TCP      54 8005 → 1121 [ACK] Seq=1 Ack=142 Win=62586 Len=0
HTTP    250 HTTP/1.1 404 Not Found  (text/plain)
```

Abbildung 14: HTTP Firmware Download

Nach mehreren fehlgeschlagenen Handshakes hat der Saugroboter versucht eine Verbindung über HTTP zu der Domain aufzubauen, um vermutlich eine neue Firmware Version herunterzuladen. Leider war die Antwort des Servers ein 404 Not Found. Vermutlich wurde die Firmware auf dem Gerät so lange nicht aktualisiert, dass es dieses Updateverzeichnis auf dem Server nicht mehr gibt und auch keine Umleitung zu einer neuen Ressource verfügbar ist.

5.5.1.2 Offene Ports

Mittels Nmap Scan konnte auf dem Saugroboter ein offener Port gefunden werden.

```
nmap -p- 192.168.137.177
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-19 22:34 GMT
Nmap scan report for ESP_885F99.mshome.net (192.168.137.177)
Host is up (0.31s latency).

PORT     STATE SERVICE VERSION
8111/tcp open  unknown

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 394.76 seconds
```

Der Port 8111 ist offiziell keinem Service zugewiesen und konnte auch von Nmap nicht zugeteilt werden. Für die Identifizierung des Services wurde Banner Grabbing angewendet.

5.5.1.3 Banner Grabbing

5.5.1.3.1 Aktive Methode

Beim Deebot Slim 2 war es möglich, sich via nc und Telnet auf dem Port zu verbinden. Keine Eingabe führte zu einer Antwort des Ports. Es wurden Scans mit Nmap durchgeführt und es konnte kein Port ermittelt werden.

```
nmap -p 8111 -sV --version-all 192.168.137.177
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-20 01:12 GMT
Nmap scan report for ESP_885F99.mshome.net (192.168.137.177)
Host is up (0.25s latency).

PORT     STATE SERVICE VERSION
8111/tcp open  unknown
```

```
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 394.04 seconds
```

```
nmap -sV --version-intensity 9 -p 8111 192.168.137.177
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-19 22:34 GMT
Nmap scan report for ESP_885F99.mshome.net (192.168.137.177)
Host is up (0.31s latency).

PORT     STATE SERVICE VERSION
8111/tcp open  unknown

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 394.76 seconds
```

5.5.1.3.2 Passive Methode

Während der Durchführung der aktiven Methode wurde der Netzwerkverkehr mit Wireshark mitgeschnitten. Anschließend wurden die Pakete analysiert und es konnten keine Informationen über den Service, der hinter dem Port läuft, gefunden werden.

5.5.1.4 Denial of Service

Der offene Port 8111 konnte mit einem TCP SYN Flood Angriff dem Nutzer die Steuerung des Saugroboters über die Applikation unmöglich machen [96].

5.5.2 Zerlegen des Geräts

Vorab gab es einige Begutachtungen und Vorbereitungen, die vor dem Auseinandernehmen des Geräts gemacht werden mussten, denn im Zuge dessen können einige Erschwernisse auftreten. Oft wird von den Herstellern bereits am Gehäuse versucht, Veränderungen oder Analysen des Gerätes zu erschweren. Auch die Verpackung des Gerätes kann schon wichtige Informationen für die Zerlegung des Geräts oder sicherheitskritische Daten enthalten.

Zerlegung: Die Zerlegung wurde durch keinerlei Hindernisse erschwert. Es waren alles „Click“ Mechanismen, die mit dem richtigen Werkzeug einfach zu öffnen waren. Es gab keine Verklebungen. Zuerst wurden die sieben Schrauben an der Unterseite des Roboters gelöst. Anschließend wurde mit Plättchen und Hebelwerkzeug die Unterseite des Roboters vorsichtig aus dem „Click“ Mechanismus gelöst. Nach der Entfernung war bereits das PCB sichtbar. Es waren keine Temper Detection Mechanismen erkennbar. Das PCB war an mehreren Sensoren und für Saugroboter essenziellen mechanischen Steuerungen angebunden. Die Beschriftungen auf den Chips waren gut zu erkennen. Die Nummer des Mikrokontrollers wurde mit einem schwarzen Lackstift übermalt, welcher sich mit Aceton sehr einfach entfernen ließ. Sonst wurden keine Lacke oder sonstigen Methoden verwendet, um die Hardwareanalyse zu erschweren. Das komplette PCB war frei zugänglich. Es wurden lediglich am WLAN Controller Metallblenden verbaut.

Abbildung 15: Deebot Slim 2

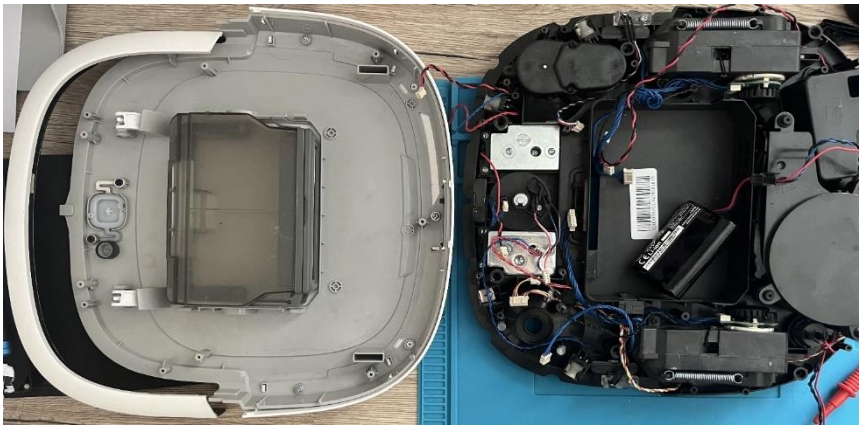


Abbildung 16: Zerlegter Deebot Slim 2

5.5.3 PCB-Board - Chips und sonstige Schnittstellen

Für das Reverse Engineering des Saugroboters müssen alle verbauten Chips und Schnittstellen am Saugroboter identifiziert werden [97]. Um Angreifbare Schnittstellen leichter identifizieren zu können sollten die Linien ausgehend vom Mikrokontroller verfolgt werden [98]. Ein Pinout Diagramm des Mikrokontrollers wie in Abbildung 20: Pinout KL26 zu sehen sollte dem Datenblatt beiliegen.

Einige Pins sind auf der Rückseite beschriftet mit TP**. Diese Test Pins werden meist für automatisierte Tests in den Fabriken verwendet [3]. Unter Umständen werden nach diesen Tests vereinzelt die Traces entfernt, um ein mögliches ungeschütztes Debug Interface zu deaktivieren. Eine ungeschützte Schnittstelle mit bereits angelöteten Pins wäre ein optimales Szenario.

5.5.3.1 Identifizierung der Chips

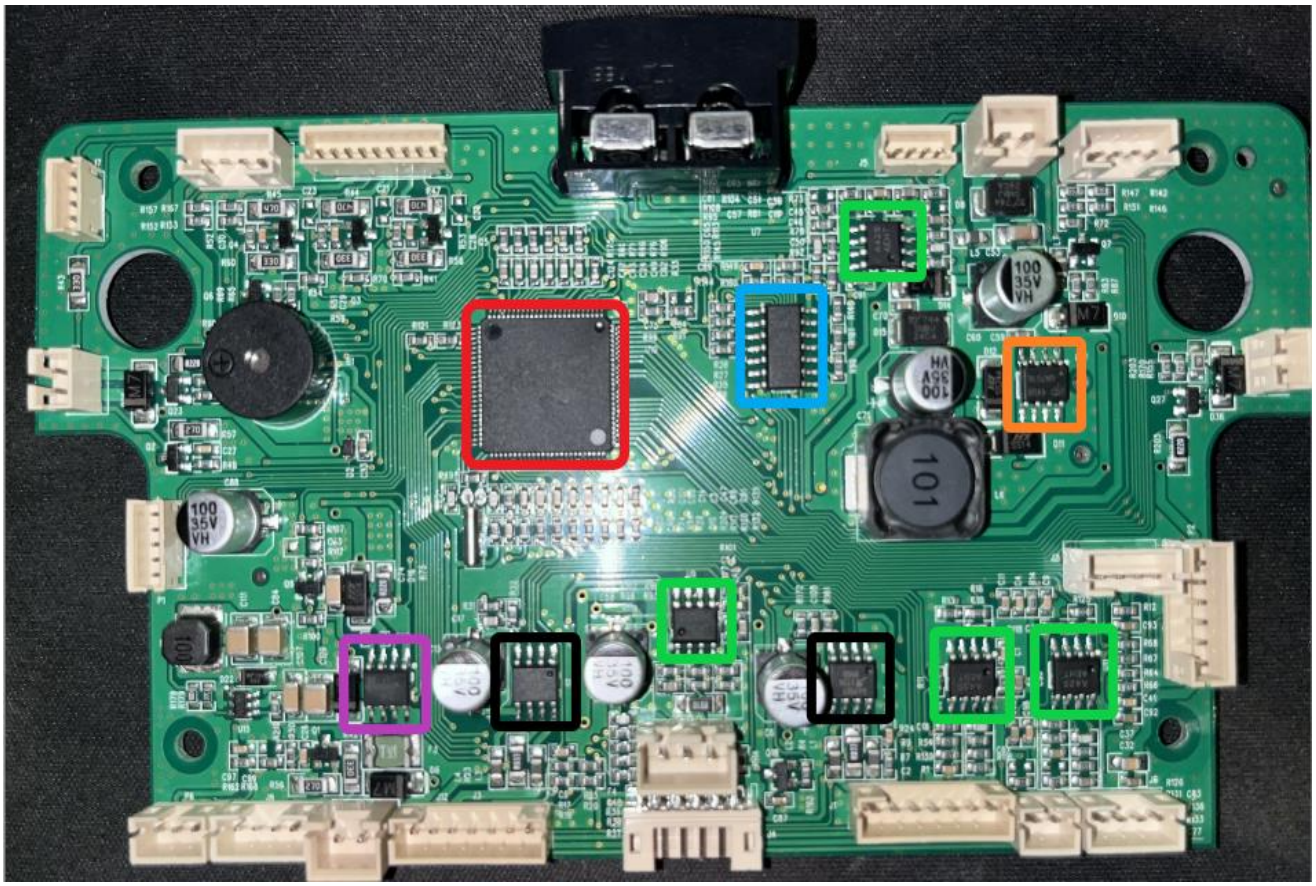


Abbildung 17: Vorderseite PCB

In Abbildung 17: Vorderseite PCB, rot umrandet, ist der mkl26z128vll4 zu sehen. Ein ARM® Cortex®-M0+ Mikrokontroller (MCU) aus der KL26 Produkt Familie [99]. Der MCU hat 128 KB Flash Memory, 16 KB statischen Arbeitsspeicher (SRAM) und 64 B Cache Speicher eingebaut.

Laut Datenblatt kann der Mikrokontroller die Kommunikationsschnittstellen I2C, I2S, SPI, UART, USB bedienen. Als debug Hilfe wird ein 2 Pin SWD Debug Port angegeben.

Als Security und Integrität Modul wird ein Watchdog Timer eingesetzt, welcher regelmäßig Kommunikation vom Betriebssystem erwartet. Sollte diese Kommunikation nicht stattfinden, wird von dem Watchdog eine Systemzurücksetzung durchgeführt.

Laut [100] sind eine Read Out Protection und weitere Sicherheitsmechanismen auf dem MCU vorhanden. Damit wurde ein direktes Auslesen des Flash Speichers auf den Schnittstellen abgesichert.

Weitere Komponenten sind der A4950T, ein Full-Bridge DMOS PWM Motor Driver [101], der sich links oben, in schwarz umrandet, am PCB befindet. In violett ist ein Metalloxid-Halbleiter-Feldeffekttransistoren (MOSFET), der C3025ld [102], verbaut. Der orange gekennzeichnete Chip ist der GA7S16, für den kein Datenblatt zu finden war. Der 74hc14d, in blau umrandet, ist ein CMOS hex inverter mit Schmitt-trigger inputs [103].

Ein Complementary Metal-Oxide-Semiconductor (CMOS) wird dazu verwendet BIOS-Parameter des Mainboards zu speichern [104]. Dadurch können, selbst wenn das Gerät über längeren Zeit keine Stromzufuhr hat, die Daten, welche für die Konfiguration des Gerätes und seiner Hardware benötigt werden, gesichert werden. Wenn die Energiezufuhr und der Kontakt zwischen CMOS und Mikrokontroller unterbrochen werden, wird die Konfiguration und mögliche hinterlegte Passwörter nach einer Zeit zurückgesetzt. [105]

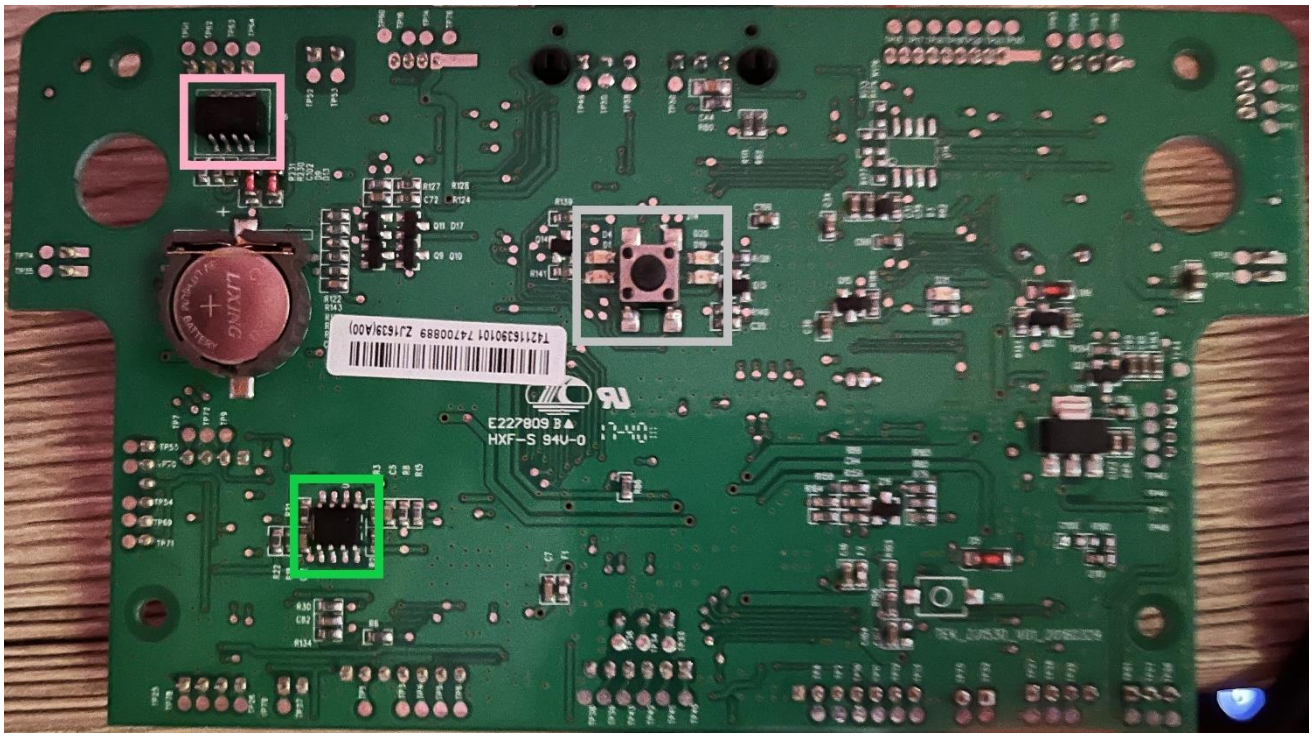


Abbildung 18: Rückseite Leiterplatte

Auf der Rückseite in Abbildung 18: Rückseite Leiterplatte, in rosa, wurde der R8010 eine „Real Time Clock“, der über I2C angesteuert werden kann, identifiziert. [106] In grün, wie auch bereits auf der Vorderseite der Leiterplatte, ist eine A42s ICs verbaut.

In grau umrandet ist eine Taste verbaut. Diese Taste ist der Auto Button, wie in

Abbildung 15: Deebot Slim 2 erkannt werden kann. Diese Taste wird für einige Konfigurationen verwendet.

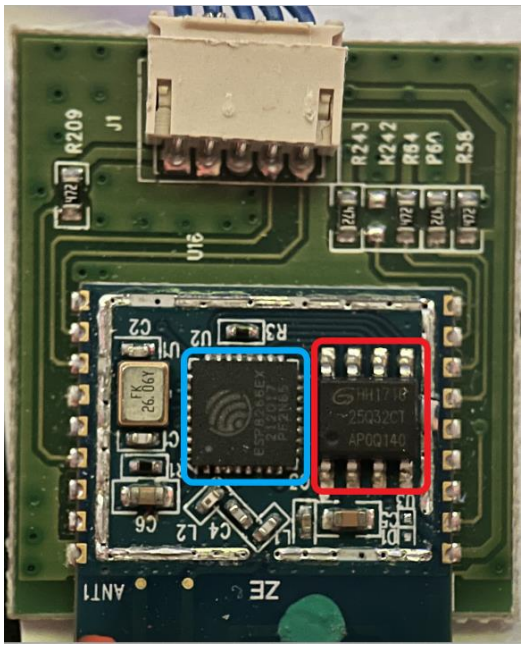


Abbildung 19: WLAN Modul

Neben einen dem rechten Rad war ein externes WLAN Modul verbaut. In blau ist der ESP8266EX zu finden [107]. Direkt daneben, in rot, ist der GD25Q32C [108] eingesetzt, ein SPI Nor Chip, der von dem ESP8266EX als externer Flash Speicher verwendet wird.

5.5.3.2 Identifizierung der Schnittstellen und Pins

Laut Datenblatt des Mikrokontrollers könnte es folgende Schnittstellen geben: I2C, I2S, SPI, UART, USB und SWD. Auf dem PCB konnte keine USB-Schnittstelle gefunden werden.

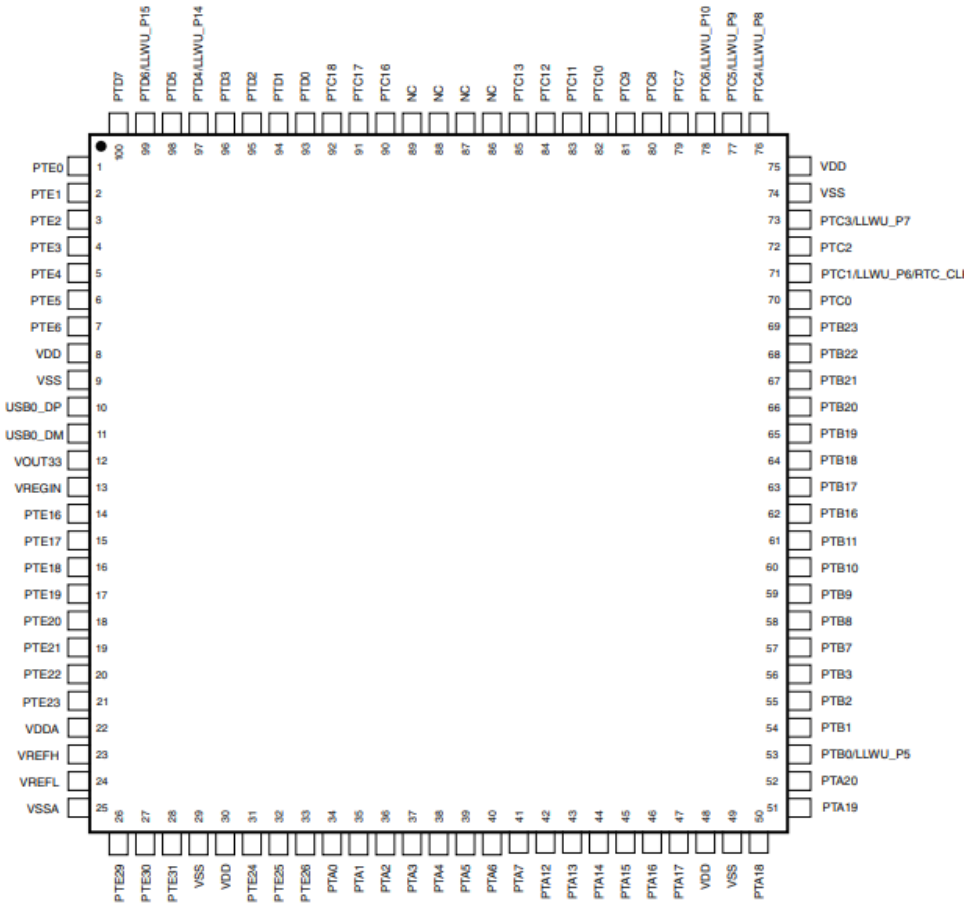


Abbildung 20: Pinout KL26

Mit dem aus dem Datenblatt gefundenen Pinout des KL26, zu sehen in Abbildung 20: Pinout KL26, konnten die Linien auf der Leiterplatte verfolgt werden. Durch diese Analyse konnten Schnittstellen, zu sehen in Abbildung 21: Interessante Schnittstellen identifiziert werden: in blau ein mögliches UART-Interface und in rot, eine SWD oder UART-Schnittstelle. Durch weiteres Identifizieren der Pins konnten diese festgestellt werden.

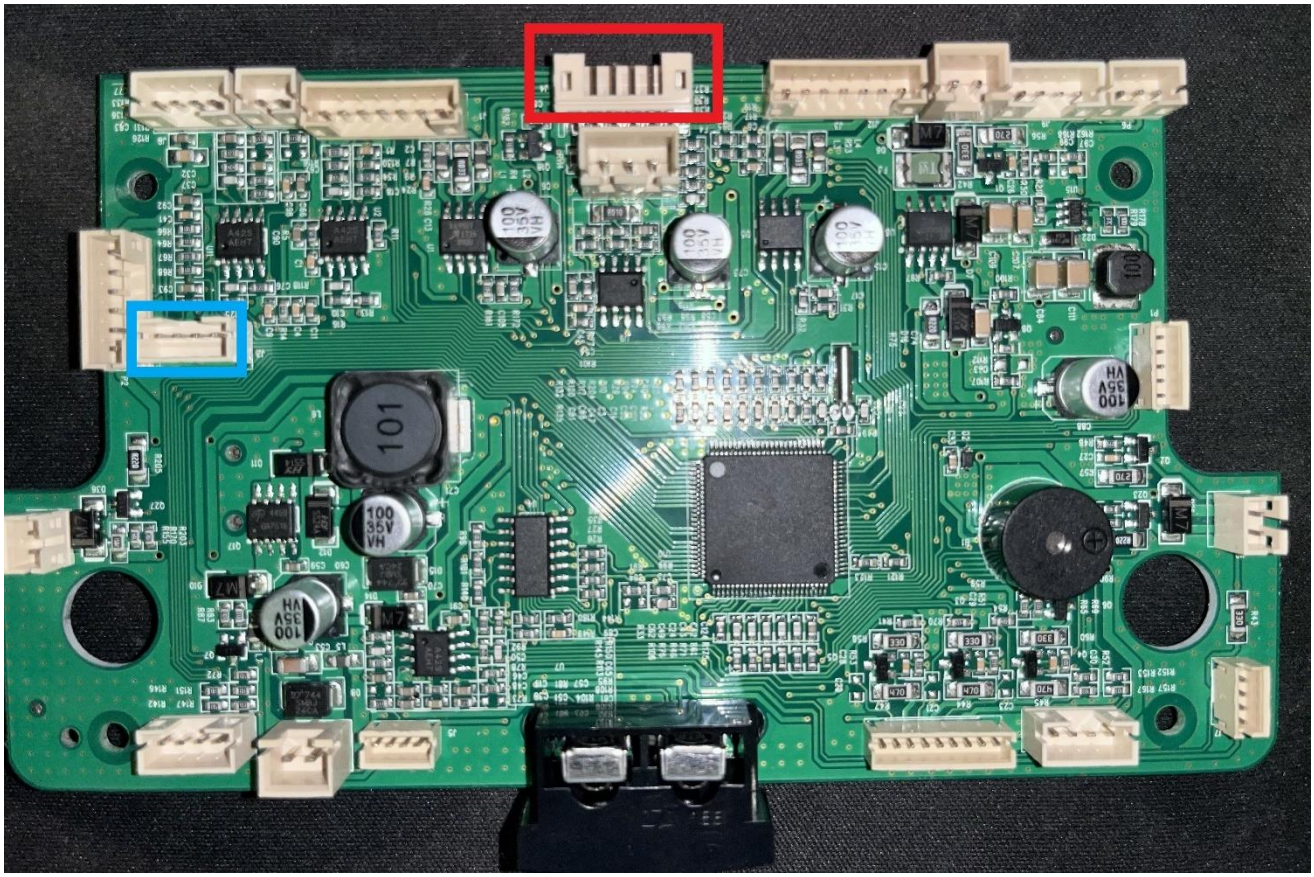


Abbildung 21: Interessante Schnittstellen

Es konnten folgende Pins auf dem in Abbildung 21: Interessante Schnittstellen zu sehenden rot markierten Schnittstelle identifiziert werden.

Nummer des Pins	Beschreibung
1	3,3 Volt
2	SWD_CLK
3	UART_RX
4	UART_TX
5	SWD_IO
6	GND

Tabelle 3: Identifizierte Pins

5.5.3.2.1 SWD- und UART-Pins

Die SWD- und UART-Pins wurden zur Bestätigung anschließend noch mit dem Logic Analyzer geprüft. Es konnte keinerlei Datentransfer auf den UART-Pins festgestellt werden. Die Pins wurden vermutlich vor der Auslieferung deaktiviert oder nie benutzt.

Über die SWD-Pins konnten allerdings, wie in Abbildung 22 zu sehen, oben die SWD_CLK und unten der SWD_IO-Pin identifiziert werden.

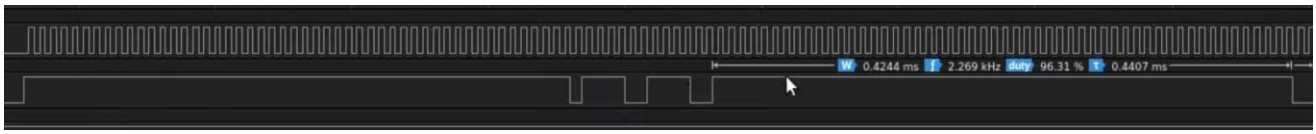


Abbildung 22: Logic Analyzer SWD Pins

5.5.4 Firmware extrahieren - SWD

Über die SWD-Schnittstelle wurde versucht eine Firmwareextraktion zu machen. Der SoC dürfte allerdings, wie in der Ausgabe von Openocd zu sehen, eine Readout Protection haben. Mit einem anderen Debugger wäre es noch möglich den MDM-AP auszulesen, um die gesetzen Security Features zu sehen [99].

```
openocd -f /interface/stlink.cfg  -f target/klx.cfg
Open On-Chip Debugger 0.11.0
Licensed under GNU GPL v2
For bug reports, read
 http://openocd.org/doc/doxygen/bugs.html
Warn : ignoring extra IDs in hla_vid_pid (maximum is 8 pairs)
Info : auto-selecting first available session transport "hla_swd". To
override use 'transport select <transport>'.
Info : The selected transport took over low-level target control. The
results might differ compared to plain JTAG/SWD
Info : add flash_bank kinetis kl25.pflash

!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!! WARNING
!!!!!!!!!!!!!!!!!!!!
 Kinetis MCUs have a MDM-AP dedicated mainly to MCU security related
functions.
 A high level adapter (like a ST-Link) you are currently using cannot
access
 the MDM-AP, so commands like 'mdm mass_erase' are not available in your
 configuration. Also security locked state of the device will not be
reported.
```

```
Be very careful as you can lock the device though there is no way to
unlock
 it without mass erase. Don't set write protection on the first block.
!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!! WARNING
!!!!!!!!!!!!!!!!!!!!

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 1000 kHz
Info : STLINK V2J29S7 (API v2) VID:PID 0483:3748
Info : Target voltage: 3.275762
Error: Unable to set adapter speed
Error: init mode failed (unable to connect to the target)
```

In [99] kann weiters noch Folgendes entnommen werden: “9.7 Debug and security: When flash security is enabled, the debug port capabilities are limited in order to prevent exploitation of secure data. In the secure state, the debugger still has access to the status register and can determine the current security state of the device. In the case of a secure device, the debugger has the capability of only performing a mass erase operation.”

6. Diskussion

Bei der OSINT Analyse konnten einige Informationen, die im späteren Verlauf der Analyse geholfen haben, gefunden werden. Open-Source Intelligenz stellt für die Sicherheit von IoT-Geräten eine große Gefahr dar, weil diese Geräte oft lange verwendet werden aber schlecht mit Sicherheitsupdates gepflegt werden. Bei der Durchsuchung nach der bereits extrahierten Firmware sind wir auf einige Firmwaredateien von anderen Geräten gestoßen. Für den Deebot Slim 2 oder einem ähnlichen Roboter von Deebot waren keine Firmwarefiles verfügbar.

Um die Erstverbindung des Saugroboters mit der Cloud bestens möglich mitlesen zu können, musste vorab die Applikation, welche diese Verbindung mit der Cloud für den Saugroboter ermöglicht, ausgeführt werden. Bei der Analyse der Applikation wurden die Angriffe 1 Spoofing und 2 Information Disclosure, die sich aus Tabelle 1: STRIDE ergeben haben, angewendet. Die Sicherheitsmechanismen der Applikation konnten mit dynamischen Patching größtenteils umgangen werden. Wie im späteren Verlauf der Analyse herausgefunden wurde, verwendet der Saugroboter die Applikation nur für die einmalige Übertragung des WLAN-Passwortes, anschließend kommuniziert der Deebot Slim 2 ausschließlich mit der Cloud direkt. Eine weitere Analyse der Applikation ist nach den gefundenen Informationen nicht weiter wichtig gewesen.

Als der Saugroboter eingerichtet wurde, konnte mit einem Man-in-the-Middle Angriff (3. Spoofing aus Tabelle 1: STRIDE) die verschlüsselte Kommunikation mit der Cloud nicht umgangen werden. Damit konnte eine korrekte Implementierung der Root Zertifikate am Saugroboter gezeigt werden. Allerdings dürfte eine Art Fallback für Firmware Updates eingerichtet worden sein. Nach Stunden der fehlgeschlagenen TLS-Handshakes wurde für das Update ein http request ausgesendet. Die Firmware war in dem vorgesehenen Verzeichnis des Servers nicht mehr auffindbar und es wurde auch kein redirect zu derselben oder neueren Version konfiguriert. Weiters konnte mit einem Port Scan (8. Elevation of Priviledge aus Tabelle 1: STRIDE) ein offener Port auf 8111 gefunden werden. Dieser Port ist keinem standardmäßigen Service zugeteilt. Mit aktiven und passiven Banner Grabbing wurde versucht, diesem Port einem Service zuzuteilen. Eine Identifizierung des Ports hätte zu einem direkten Zugang zum Deebot Slim 2 führen können. Durch einen einfachen SYN-Flood Angriff (7. Denial of Service aus Tabelle 1: STRIDE) konnte der Saugroboter funktionsuntüchtig gemacht werden.

Bei der Analyse der Hardware konnte ein funktionierender SWD-Port identifiziert (4. Tampering und 6. Information Disclosure aus Tabelle 1: STRIDE) werden. Auf dem Port konnten die Pins SWD_CLK und SWD_IO mit dem Logic Analyzer verifiziert werden. Durch die gesetzten Sicherheitsmechanismen auf dem Chip konnte kein Dump der Firmware oder direkter Zugang gemacht werden. Wenn ein Reset-Pin vorhanden wäre, hätte noch ein Angriff, wie in [109] beschrieben, durchgeführt werden können. Ohne dem vorhanden Reset-Pin hätte das Gerät dafür aber im schlimmsten Fall mehrere tausende Male manuell auf- und abgedreht werden müssen. Eine weitere Möglichkeit wäre, wie in [110] gezeigt, ein Ätzen der Oberfläche des SoC Chips gewesen. Anschließend hätten Speicherchips im Inneren über die Kommunikationsverbindungen zur Recheneinheit angegriffen werden können.

Die in Methodik gezeigten Flow Charts und Beschreibungen können in anderen Arbeiten als Leitfaden für die Sicherheitsanalyse eines IoT-Gerätes herangezogen werden. Beispielsweise konnte der in Tabelle 1: STRIDE aufgeführte Angriff 5. Tampering: Firmware Reverse Engineering nicht durchgeführt werden, weil die Firmware im Zuge der Arbeit nicht erfolgreich erhalten werden konnte. Diese Methode wird in Kapitel 4.7 Firmware Reverse-Engineering beschrieben.

7. Conclusio

Wenn lediglich auf die Ergebnisse geblickt wird, könnte argumentiert werden, dass der Deebot Slim 2 ein sicheres Gerät ist. Allerdings wäre durch den Download der Firmware des Saugroboters eine Analyse der Firmware möglich gewesen, was Probleme, wie die Identifikation des offenen Ports, sehr wahrscheinlich gelöst hätte, und womöglich noch größere Sicherheitsprobleme des Deebot Slim 2 aufdecken hätte können. Durch mehr Zeit, Wissen und Ressourcen hätten weitere Tests durchgeführt werden können.

In Abschnitt Methodik wurde mit den Flow Charts und theoretischen Methoden ein Leitfaden für weitere Sicherheitsanalysen von IoT-Geräten gegeben. Im Abschnitt Ergebnisse wurden diese Methoden angewendet und es wurde genauer auf deren Durchführung eingegangen.

Durch die Analyse der Komponenten und Use Cases des Gerätes wurden mit dem Threat Model mögliche Angriffspunkte identifiziert und mögliche Gegenmaßnahmen für die Angriffspunkte definiert.

Es konnte bei der OSINT Analyse keine Firmware für den Deebot Slim 2 oder ein ähnliches Gerät aus der Ecovacsreihe gefunden werden, was darauf schließen lässt, dass Ecovacs bemüht ist, seine Geräte in dieser Hinsicht zu schützen. Es konnten aber einige Informationen über die Zerlegung und mögliche eingesetzte Protokolle zu ähnlichen Modellen gefunden werden.

Durch die Analyse der Applikation konnten wichtige Erkenntnisse für den weiteren Verlauf dieser Arbeit gewonnen werden. Durch das dynamische Patching konnten Teile des Datenverkehrs unverschlüsselt mitgelesen werden. Damit konnte verifiziert werden, dass der Saugroboter die Applikation lediglich für die Initialisierung benötigt hat, was einige Angriffsszenarien für die Analyse der Applikation ausgeschlossen hat.

Bei der Netzwerkanalyse des Saugroboters konnte eine erfolgreiche Implementierung der Zertifikate am Saugroboter nachgewiesen werden. Allerdings konnte nach einer sehr hohen Anzahl an TCP Handshake Misserfolgen ein Fallback auf http festgestellt werden, als der Saugroboter ein Firmware Update durchführen wollte. Die Datei dürfte auf dem Server nicht mehr verfügbar gewesen sein und eine Weiterleitung zu der neuen Datei war nicht verfügbar. Weiters konnte der offene Port 8111 identifiziert werden. Es wurde aktives und passives Banner Grabbing angewendet, um den dahinterliegenden Service zu finden. Mit einem Tcp-Syn-Flood Angriff auf den genannten Port konnte ein Denial-of-Service hervorgerufen werden.

Bei der Zerlegung des Saugroboters konnten keine Sabotage Erkennungsmechanismen gefunden werden. Mit der Identifizierung der wichtigsten Komponenten und deren Datenblätter konnten eine SWD- und UART-Schnittstelle erkannt werden. Über den SWD-Port konnte verifiziert werden, dass Daten darüber gesendet werden. Durch den internen Speicherchip des Mikrokontrollers war ein Auslesen der Firmware nur über diesen möglich. Der Mikrokontroller verfügt über einige Schutzmechanismen, die nur ein Löschen der Firmware ermöglichen. Mit fortschrittlicheren Angriffen hätten diese Sicherheitsmechanismen umgangen werden können.

8. Weiterführende Arbeiten

Weiterführend könnte die Applikation einer genaueren Sicherheitsanalyse unterzogen werden. Zumal die Applikation im Falle des Deebot Slim 2 wenige Funktionen mitgebracht hat, war sie für die Analyse nicht von großer Bedeutung. In der Arbeit wurde nur ein Teil einer vollständigen Applikationsanalyse durchgeführt, wie sie Beispielsweise in [111] beschrieben wird.

Um weitere Informationen oder sogar eine Firmwareversion zu erhalten, könnten mit guten Sprachkenntnissen in Chinesisch über die Suchmaschine baidu, möglicherweise bessere Ergebnisse erzielt werden. Durch das Erraten des Services könnte versucht werden den offenen Port anzugreifen und über diesen die Firmware zu erhalten.

Wie bereits in 6. Diskussion beschrieben wären die beiden Angriffe [109] und [110] eine weitere Möglichkeit, die Firmware oder andere interessante Daten von dem Deebot Slim 2 zu extrahieren.

Sollte eine Firmware Datei gefunden werden, könnte diese, wie in 4.7 Firmware Reverse-Engineering beschrieben, analysiert werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Referenzen

[1] „IoT platforms for Academic R&D: A critical review“. https://www.ericsson.com/en/reports-and-papers/research-papers/iot-platforms-for-academia (zugegriffen 11. Dezember 2022).

[2] „wp-iot-security-february-2017.pdf“. Zugegriffen: 11. Dezember 2022. [Online]. Verfügbar unter: https://www.ericsson.com/49ea71/assets/local/reports-papers/white-papers/wp-iot-security-february-2017.pdf

[3] Paulino Calderon, Evangelos Deirmentzoglou, und Beau Woods, *Practical IoT Hacking - The Definitive Guide to Attacking the Internet of Things*. [Online]. Verfügbar unter: https://practical-iot-hacking.com/

[4] R. Williams, E. McMahon, S. Samtani, M. Patton, und H. Chen, „Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach“, in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, Juli 2017, S. 179–181. doi: 10.1109/ISI.2017.8004904.

[5] „Shodan“, *Shodan*. https://www.shodan.io (zugegriffen 28. Februar 2023).

[6] „Experts disclose dangerous flaws in robotic Dongguan Diqee 360 smart vacuumsSecurity Affairs“. https://securityaffairs.co/74592/hacking/hacking-smart-vacuums.htm (zugegriffen 28. Februar 2023).

[7] „Xiaomi Mi Robot vacuum cleaner hacked“, 4. Januar 2018. https://www.kaspersky.com.au/blog/xiaomi-mi-robot-hacked/19309/ (zugegriffen 28. Februar 2023).

[8] P. Umbelino, „Checkmarx Research: Smart Vacuum Security Flaws May Leave Users Exposed“, *Checkmarx.com*, 26. Februar 2020. https://checkmarx.com/blog/checkmarx-research-smart-vacuum-security-flaws-leave-users-exposed/ (zugegriffen 28. Februar 2023).

[9] T. O. Labs, „Bypassing Root Detection and Emulator Detection in Android Apps using Frida“, *Medium*, 31. März 2021. https://theoffensivelabs.medium.com/bypassing-root-detection-and-emulator-detection-in-android-apps-using-frida-e938109e468c (zugegriffen 21. Februar 2023).

[10] „Reverse engineering & modifying Android apps with JADX & Frida“. https://httptoolkit.com/blog/android-reverse-engineering/ (zugegriffen 28. Februar 2023).

[11] „Frida CodeShare“. https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/ (zugegriffen 20. Februar 2023).

[12] Marcel Mangel und Bicchi Sebastian, *Praktische Einführung in Hardware Hacking*. Mitp. [Online]. Verfügbar unter: https://www.mitp.de/IT-WEB/IT-Sicherheit/Praktische-Einfuehrung-in-Hardware-Hacking.html

[13] F. Ullrich, J. Classen, J. Eger, und M. Hollick, „Vacuums in the Cloud: Analyzing Security in a Hardened IoT Ecosystem“, *https://www.usenix.org/system/files/woot19-paper_ullrich.pdf*, S. 11.

[14] V. Pasknel, „Hacking the IoT with MQTT“, *Medium*, 19. Juli 2017. https://morphuslabs.com/hacking-the-iot-with-mqtt-8edaf0d07b9b (zugegriffen 28. Februar 2023).

[15] „A journey into IoT – Hardware hacking: UART | @Mediaservice.net Technical Blog“. https://techblog.mediaservice.net/2019/03/a-journey-into-iot-hardware-hacking-uart/ (zugegriffen 10. August 2021).

[16] „Accessing and Dumping Firmware Through UART“. https://www.cyberark.com/resources/threat-research-blog/accessing-and-dumping-firmware-through-uart (zugegriffen 7. Juli 2021).
[17] „An Introduction to Hardware Hacking“. https://www.cyberark.com/resources/threat-research-blog/an-introduction-to-hardware-hacking (zugegriffen 23. Juli 2021).
[18] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, und A. Zanella, „IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices“, *IEEE Internet of Things Journal*, Bd. 6, Nr. 5, S. 8182–8201, Okt. 2019, doi: 10.1109/JIOT.2019.2935189. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=8796409&ref=&tag=1
[19] B. Pingle, A. Mairaj, und A. Y. Javaid, „Real-World Man-in-the-Middle (MITM) Attack Implementation Using Open Source Tools for Instructional Use“, in *2018 IEEE International Conference on Electro/Information Technology (EIT)*, Mai 2018, S. 0192–0197. doi: 10.1109/EIT.2018.8500082. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=8500082&ref=
[20] valbrux, „MITM using arpspoof + Burp or mitmproxy on Kali Linux.“, *Valbrux*, 26. November 2017. https://www.valbrux.it/blog/2017/11/26/mitm-using-arpspoof-burp-or-mitmproxy-on-kali-linux/ (zugegriffen 22. Februar 2023).
[21] „CH341A USB serial EEPROM reader under Linux | danman’s blog“. https://blog.danman.eu/ch341a-usb-serial-eeprom-reader-under-linux/ (zugegriffen 7. Juli 2021).
[22] „Bus Pirate UART guide – Dangerous Prototypes“, 14. Mai 2010. http://dangerousprototypes.com/blog/bus-pirate-manual/bus-pirate-uart-guide/ (zugegriffen 22. Februar 2023).
[23] „Bus Pirate v3.6a Hookup Guide - SparkFun Learn“. https://learn.sparkfun.com/tutorials/bus-pirate-v36a-hookup-guide/all (zugegriffen 28. Februar 2023).
[24] „Bus Blaster - DP“. http://dangerousprototypes.com/docs/Bus_Blaster (zugegriffen 16. März 2023).
[25] „J-Link Debug Probes by SEGGER – the Embedded Experts“. https://www.segger.com/products/debug-probes/j-link/ (zugegriffen 16. März 2023).
[26] „Hardware hacking tutorial: Dumping and reversing firmware“. https://ivanorsolic.github.io/post/hardwarehacking1/ (zugegriffen 22. Februar 2023).
[27] Bruce Dang, Alexandre Gazet, und Elias Bachaalany, *Practical Reverse Engineering*. John Wiley & Sons, 2014. [Online]. Verfügbar unter: https://www.oreilly.com/library/view/practical-reverse-engineering/9781118787397/
[28] Adam Shostack, *Threat Modeling*. 2014. [Online]. Verfügbar unter: https://shostack.org/books/threat-modeling-book
[29] jegeib, „Threats - Microsoft Threat Modeling Tool - Azure“, 25. August 2022. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats (zugegriffen 3. März 2023).
[30] „PASTA Threat Modeling - Threat-Modeling.com“, 24. Juli 2022. https://threat-modeling.com/pasta-threat-modeling/ (zugegriffen 3. März 2023).
[31] „Trike Threat Modeling - Threat-Modeling.com“, 4. September 2022. https://threat-modeling.com/trike-threat-modeling/ (zugegriffen 3. März 2023).
[32] EC-Council, „The Benefits of Utilizing the OCTAVE Threat Model“, *Cybersecurity Exchange*, 4. September 2022. https://www.eccouncil.org/cybersecurity-exchange/threat-intelligence/octave-threat-model-benefits/ (zugegriffen 3. März 2023).

[33] threatmodeler, „Threat Modeling Methodologies: What is VAST?“, *ThreatModeler*, 9. Oktober 2018. https://threatmodeler.com/threat-modeling-methodologies-vast/ (zugegriffen 3. März 2023).
[34] R. J. Mejias, M. A. Shepherd, M. Fronmueller, und R. A. Huff, „Using Threat Vulnerability Asset (TVA) Methodology to Identify Cyber Threats and System Vulnerabilities: A Student Field Project Case Study“, [Online]. Verfügbar unter: https://www.csupueblo.edu/hasan-school-of-business/_doc/ccser/using-threat-vulnerability-asset-to-identify-threats-vulnerabilities.pdf
[35] EC-Council, „DREAD Threat Modeling: An Introduction to Qualitative Risk Analysis“, *Cybersecurity Exchange*, 9. März 2022. https://www.eccouncil.org/cybersecurity-exchange/threat-intelligence/dread-threat-modeling-intro/ (zugegriffen 11. April 2023).
[29] R. Achkoudir und Z. Alsaadi, „Ethical Hacking of a Smart Plug“, S. 62. https://kth.diva-portal.org/smash/get/diva2:1536295/FULLTEXT01.pdf
[37] „百度一下，你就知道“. https://www.baidu.com/baidu.html?from=noscript (zugegriffen 15. März 2023).
[38] „ECOVACS Robotics“, *Wikipedia*. 29. Januar 2023. Zugegriffen: 15. März 2023. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=ECOVACS_Robotics&oldid=230333616
[39] „OWASP Mobile Application Security Verification Standard (MASVS)“. OWASP, 10. April 2023. Zugegriffen: 11. April 2023. [Online]. Verfügbar unter: https://github.com/OWASP/owasp-masvs
[34] S. Li, M. Zhang, C. Li, Y. Zhou, K. Wang, und Y. Deng, „Mobile APP Personal Information Security Detection and Analysis“, in *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*, Juni 2021, S. 82–87. doi: 10.1109/ICIS51600.2021.9516873. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=9516873&ref=
[41] „Mobile Security Framework (MobSF)“. Mobile Security Framework, 10. April 2023. Zugegriffen: 10. April 2023. [Online]. Verfügbar unter: https://github.com/MobSF/Mobile-Security-Framework-MobSF
[42] „Android Debug Bridge (adb)“, *Android Developers*. https://developer.android.com/studio/command-line/adb (zugegriffen 20. Februar 2023).
[43] DigiDNA, „iMazing | iPhone-, iPad- & iPod-Manager für Mac & PC“. https://imazing.com/de (zugegriffen 10. April 2023).
[44] +Ch0pin, „Description“. 22. April 2023. Zugegriffen: 22. April 2023. [Online]. Verfügbar unter: https://github.com/Ch0pin/medusa
[45] P. Falcarin, M. Baldi, und D. Mazzocchi, „Software Tampering Detection using AOP and mobile code“, [Online]. Verfügbar unter: https://repository.uel.ac.uk/download/fbd4e77cd0102ea271b226c0821d381dcb989de118a12ae3601cd5414fe9bb5e/81770/Falcarin%2C%20P%20%282004%29%20AOSDSEC.pdf
[46] B. Gruver, „JesusFreke/smali“. 11. April 2023. Zugegriffen: 11. April 2023. [Online]. Verfügbar unter: https://github.com/JesusFreke/smali
[47] „Releases · frida/frida“, *GitHub*. https://github.com/frida/frida/releases (zugegriffen 20. Februar 2023).
[42] A. Adithyan, K. Nagendran, R. Chethana, G. Pandy D., und G. Prashanth K., „Reverse Engineering and Backdooring Router Firmwares“, in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, März 2020, S. 189–193. doi: 10.1109/ICACCS48705.2020.9074317. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=9074317&ref=

[43] C. Sun, R. Xing, Y. Wu, G. Zhou, F. Zheng, und D. Hu, „Design of Over-the-Air Firmware Update and Management for IoT Device with Cloud-based RESTful Web Services“, in *2021 China Automation Congress (CAC)*, Okt. 2021, S. 5081–5085. doi: 10.1109/CAC53003.2021.9727516. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=9727516&ref=
[44] S. Andy, B. Rahardjo, und B. Hanindhito, „Attack scenarios and security analysis of MQTT communication protocol in IoT system“, in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Sep. 2017, S. 1–6. doi: 10.1109/EECSI.2017.8239179. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=8239179&ref=
[51] „Wireshark · Go Deep“, *Wireshark*. https://www.wireshark.org/ (zugegriffen 13. April 2023).
[52] „Burp Suite - Application Security Testing Software“. https://portswigger.net/burp (zugegriffen 13. April 2023).
[53] Chris Sanders, *Practical Packet Analysis 3rd Edition*. William Pollock, 2017. [Online]. Verfügbar unter: https://books.google.at/books?id=kNOaDgAAQBAJ&printsec=frontcover#v=onepage&q&f=false
[54] „Nmap: the Network Mapper - Free Security Scanner“. https://nmap.org/ (zugegriffen 13. April 2023).
[55] Paulino Calderon, *Nmap: Network Exploration and Security Auditing Cookbook - Second Edition*. Packt, 2017. [Online]. Verfügbar unter: https://books.google.at/books/about/Nmap_Network_Exploration_and_Security_Au.html?id=cl3KswEACAAJ&redir_esc=y
[56] „SecurityTrails | What is Banner Grabbing? Best Tools and Techniques Explained“. https://securitytrails.com/blog/banner-grabbing (zugegriffen 28. Februar 2023).
[57] Jon Erickson, *Hacking: The Art of Exploitation*. No Starch Press, 2010. [Online]. Verfügbar unter: https://books.google.at/books/about/Hacking.html?id=P8ijosP6ti4C&redir_esc=y
[58] D. Antonioli, N. O. Tippenhauer, und K. B. Rasmussen, „The {KNOB} is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth {BR/EDR}“, gehalten auf der 28th USENIX Security Symposium (USENIX Security 19), 2019, S. 1047–1061. Zugegriffen: 14. April 2023. [Online]. Verfügbar unter: https://www.usenix.org/conference/usenixsecurity19/presentation/antonioli
[53] M. Ren, X. Ren, H. Feng, J. Ming, und Y. Lei, „Security Analysis of Zigbee Protocol Implementation via Device-agnostic Fuzzing“, *Digital Threats*, Bd. 4, Nr. 1, S. 1–24, März 2023, doi: 10.1145/3551894. https://dl.acm.org/doi/10.1145/3551894
[60] „IoTcube - CSSA“. https://iotcube.net/userguide/manual/zfuzz (zugegriffen 16. April 2023).
[55] J. Chen *u. a.*, „IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing“, in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2018. doi: 10.14722/ndss.2018.23159. https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_01A-1_Chen_paper.pdf
[62] „Ubertooth“. Great Scott Gadgets, 14. April 2023. Zugegriffen: 14. April 2023. [Online]. Verfügbar unter: https://github.com/greatscottgadgets/ubertooth
[63] „OASIS Open“, *OASIS Open*, 12. April 2023. https://www.oasis-open.org/ (zugegriffen 16. April 2023).
[58] G. Casteur *u. a.*, „Fuzzing attacks for vulnerability discovery within MQTT protocol“, in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, Juni 2020, S. 420–

425. doi: 10.1109/IWCMC48107.2020.9148320. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=9148320&ref=
[65] „Service Name and Transport Protocol Port Number Registry“. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=mqtt (zugegriffen 16. April 2023).
[66] „ST-LINK/V2 - ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32 - STMicroelectronics“. https://www.st.com/en/development-tools/st-link-v2.html (zugegriffen 29. Mai 2023).
[66] J. Norhuzaimin und H. H. Maimun, „The design of high speed UART“, in *2005 Asia-Pacific Conference on Applied Electromagnetics*, Dez. 2005, S. 5 pp.-. doi: 10.1109/APACE.2005.1607831. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=1607831&ref=
[68] devttys0, „devttys0/baudrate“. 20. April 2023. Zugegriffen: 22. April 2023. [Online]. Verfügbar unter: https://github.com/devttys0/baudrate/blob/c4561228ac0d872bfcfc5379e0f1711b384ba498/baudrate.py
[68] S. Hemram, G. J. W. Kathrine, G. M. Palmer, und S. E. V. Ewards, „Firmware Vulnerability Detection in Embedded Systems and Internet of Things“, in *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, Nov. 2022, S. 1161–1167. doi: 10.1109/ICAISS55157.2022.10010804. https://ieeexplore-ieee-org/stampPDF/getPDF.jsp?tp=&arnumber=10010804&ref=
[70] „Binwalk“. ReFirm Labs, 21. April 2023. Zugegriffen: 21. April 2023. [Online]. Verfügbar unter: https://github.com/ReFirmLabs/binwalk
[71] „dd(1): convert/copy file - Linux man page“. https://linux.die.net/man/1/dd (zugegriffen 21. April 2023).
[72] „Firmwalker – Craig Smith“. https://craigsmith.net/firmwalker/ (zugegriffen 21. April 2023).
[73] „find(1) - Linux man page“. https://linux.die.net/man/1/find (zugegriffen 21. April 2023).
[74] „grep(1): print lines matching pattern - Linux man page“. https://linux.die.net/man/1/grep (zugegriffen 21. April 2023).
[75] „hashcat/hashcat“. hashcat, 21. April 2023. Zugegriffen: 21. April 2023. [Online]. Verfügbar unter: https://github.com/hashcat/hashcat
[76] „Hex Rays - State-of-the-art binary code analysis solutions“. https://hex-rays.com/ida-pro/ (zugegriffen 21. April 2023).
[77] „Releases · radareorg/radare2“, *GitHub*. https://github.com/radareorg/radare2/releases (zugegriffen 21. April 2023).
[78] „QEMU“. https://www.qemu.org/ (zugegriffen 20. Juni 2019).
[79] „Firmware Analysis Toolkit“. Attify, Inc., 21. April 2023. Zugegriffen: 21. April 2023. [Online]. Verfügbar unter: https://github.com/attify/firmware-analysis-toolkit
[79] D. D. Chen, M. Egele, M. Woo, und D. Brumley, „Towards Automated Dynamic Analysis for Linux-based Embedded Firmware“, in *Proceedings 2016 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2016. doi: 10.14722/ndss.2016.23415. https://www.ndss-symposium.org/wp-content/uploads/2017/09/towards-automated-dynamic-analysis-linux-based-embedded-firmware.pdf
[81] „netstat(8) - Linux man page“. https://linux.die.net/man/8/netstat (zugegriffen 21. April 2023).
[82] Andre, „Repair a Deebot Slim 2“, *Saeber*, 21. Januar 2021. https://saeber.de/?p=4469 (zugegriffen 15. März 2023).

[83] Ligio, „ozmo“. 14. September 2022. Zugegriffen: 16. Februar 2023. [Online]. Verfügbar unter: https://github.com/Ligio/ozmo/blob/8abe128827778f6bab294456fd0ad4e6fdbfb1ec/protocol.md
[84] „ECOVACS HOME APK für Android herunterladen“, *APKPure.com*. https://apkpure.com/de/ecovacs-home/com.eco.global.app (zugegriffen 28. April 2023).
[85] G. link, Facebook, Twitter, Pinterest, Email, und O. Apps, „Android task hijacking using moveTaskToBack() and excludeFromRecents“. http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html (zugegriffen 13. März 2023).
[86] 大A, „FridaAntiRootDetection“. 21. April 2023. Zugegriffen: 22. April 2023. [Online]. Verfügbar unter: https://github.com/AshenOneYe/FridaAntiRootDetection/blob/23d888ea7af3bccdd3d7830ca0f200d976b747a0/antiroot.js
[87] „Security with network protocols“, *Android Developers*. https://developer.android.com/training/articles/security-ssl (zugegriffen 13. März 2023).
[88] „Packet Capture – Apps bei Google Play“. https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture&hl=de_AT&gl=US (zugegriffen 20. Februar 2023).
[89] „Info about fajita | LineageOS Wiki“. https://wiki.lineageos.org/devices/fajita/ (zugegriffen 20. Februar 2023).
[90] „HTTP Toolkit“. https://httptoolkit.com/ (zugegriffen 20. Februar 2023).
[91] Nikolay Elenkov, *Android Security Internals: An In-Depth Guide to Android's Security Architecture*. No Starch Press, 2014. [Online]. Verfügbar unter: https://books.google.at/books?id=y11NBQAAQBAJ&printsec=copyright&redir_esc=y#v=onepage&q&f=false
[92] N. Elenkov, „Android backup extractor“. 21. April 2023. Zugegriffen: 22. April 2023. [Online]. Verfügbar unter: https://github.com/nelenkov/android-backup-extractor
[93] R. Heaton, „How to make a TCP proxy“. 14. Februar 2023. Zugegriffen: 20. Februar 2023. [Online]. Verfügbar unter: https://github.com/robert/how-to-build-a-tcp-proxy/blob/af6042ae1bed665b18ece6cd9f362386b16a3850/fake_dns_server.py
[94] „Invisible proxying“. https://portswigger.net/burp/documentation/desktop/tools/proxy/invisible (zugegriffen 20. Februar 2023).
[95] „Proxy settings“. https://portswigger.net/burp/documentation/desktop/settings/tools/proxy (zugegriffen 20. Februar 2023).
[95] M. Bogdanoski, T. Shuminoski, und A. Risteski, „Analysis of the SYN Flood DoS Attack“, *IJCNIS*, Bd. 5, Nr. 8, S. 15–11, Juni 2013, doi: 10.5815/ijcnis.2013.08.01. http://www.mecs-press.org/ijcnis/ijcnis-v5-n8/v5n8-1.html
[97] throbscottle, „How to Reverse Engineer a Schematic From a Circuit Board“, *Instructables*. https://www.instructables.com/How-to-reverse-engineer-a-schematic-from-a-circuit/ (zugegriffen 28. Februar 2023).
[98] S. Patel, „Reverse Engineering a Printed Circuit Board“, *Electronic Design*, August 2022. https://www.electronicdesign.com/markets/automation/article/21249478/mermar-electronics-reverse-engineering-a-printed-circuit-board (zugegriffen 28. Februar 2023).
[99] „KL26 Sub-Family - Reference Manual“, 2013, [Online]. Verfügbar unter: https://www.pjrc.com/teensy/KL26P121M48SF4RM.pdf
[100] M. Hunter, „Using the Kinetis Security and Flash Protection Features“, [Online]. Verfügbar unter: https://www.nxp.com/docs/en/application-note/AN4507.pdf

[101] „datasheet A4950“. https://www.digchip.com/datasheets/parts/datasheet/029/A4950-pdf.php (zugegriffen 19. März 2023).
[102] „DMC3025LDV datasheet“. https://datasheetspdf.com/pdf-file/1072122/Diodes/DMC3025LDV/1 (zugegriffen 19. März 2023).
[103] „74HC14D - Hex inverting Schmitt trigger“, *Nexperia*. https://www.nexperia.com/product/74HC14D (zugegriffen 28. Februar 2023).
[103] B. Richter und A. Moradi, „Template attacks on nano-scale CMOS devices“, *J Cryptogr Eng*, Bd. 10, Nr. 3, S. 275–285, Sep. 2020, doi: 10.1007/s13389-020-00225-8. https://link.springer.com/content/pdf/10.1007/s13389-020-00225-8.pdf
[104] H. Yamada, S. Okura, M. Shirahata, und T. Fujino, „Modeling attacks against device authentication using CMOS image sensor PUF“, *IEICE Electron. Express*, Bd. 18, Nr. 7, S. 20210058–20210058, Apr. 2021, doi: 10.1587/elex.18.20210058. https://www.jstage.jst.go.jp/article/elex/18/7/18_18.20210058/_article
[106] „slus900d.pdf“. Zugegriffen: 19. März 2023. [Online]. Verfügbar unter: https://www.ti.com/lit/ds/slus900d/slus900d.pdf?ts=1679158586042&ref_url=https%253A%252F%252Fwww.google.com%252F
[107] „0a-esp8266ex_datasheet_en.pdf“. Zugegriffen: 19. März 2023. [Online]. Verfügbar unter: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
[108] „gd25q32.pdf“. Zugegriffen: 19. März 2023. [Online]. Verfügbar unter: https://www.elm-tech.com/en/products/spi-flash-memory/gd25q32/gd25q32.pdf
[109] „There’s A Hole In Your SoC: Glitching The MediaTek BootROM“, *NCC Group Research Blog*, 15. Oktober 2020. https://research.nccgroup.com/2020/10/15/theres-a-hole-in-your-soc-glitching-the-mediatek-bootrom/ (zugegriffen 22. April 2023).
[110] „eMMCvsNAND.pdf“. Zugegriffen: 19. Februar 2023. [Online]. Verfügbar unter: https://rusolut.com/wp-content/uploads/2018/10/eMMCvsNAND.pdf
[111] A. Alanda, D. Satria, H. A. Mooduto, und B. Kurniawan, „Mobile Application Security Penetration Testing Based on OWASP“, *IOP Conf. Ser.: Mater. Sci. Eng.*, Bd. 846, Nr. 1, S. 012036, Mai 2020, doi: 10.1088/1757-899X/846/1/012036.

Appendix

Frida Script

```
/*
 * Dieses Script wurde aus mehreren Scripts zusammengestellt. Die wichtigsten Teile wurden aus
folgenden Quellen entnommen:
 *
 * - https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/
 * - https://codeshare.frida.re/@avltree9798/universal-android-ssl-pinning-bypass/
 * - https://pastebin.com/TVJD63uM
 */

setTimeout(function () {
    Java.perform(function () {
        console.log("---");
        console.log("Unpinning Android app...");

        /// -- Generic hook to protect against SSLPeerUnverifiedException -- ///

        // In some cases, with unusual cert pinning approaches, or heavy obfuscation, we can't
        // match the real method & package names. This is a problem! Fortunately, we can still
        // always match built-in types, so here we spot all failures that use the built-in cert
        // error type (notably this includes OkHttp), and after the first failure, we dynamically
        // generate & inject a patch to completely disable the method that threw the error.
        try {
            const UnverifiedCertError = Java.use('javax.net.ssl.SSLPeerUnverifiedException');
            UnverifiedCertError.$init.implementation = function (str) {
                console.log('  --> Unexpected SSL verification failure, adding dynamic patch...');

                try {
                    const stackTrace = Java.use('java.lang.Thread').currentThread().getStackTrace();
                    const exceptionStackIndex = stackTrace.findIndex(stack =>
                        stack.getClassName() === "javax.net.ssl.SSLPeerUnverifiedException"
                    );
                    const callingFunctionStack = stackTrace[exceptionStackIndex + 1];

                    const className = callingFunctionStack.getClassName();
                    const methodName = callingFunctionStack.getMethodName();

                    console.log(`      Thrown by ${className}->${methodName}`);

                    const callingClass = Java.use(className);
```

```
                    const callingMethod = callingClass[methodName];

                    if (callingMethod.implementation) return; // Already patched by Frida - skip it

                    console.log('      Attempting to patch automatically...');
                    const returnTypeName = callingMethod.returnType.type;

                    callingMethod.implementation = function () {
                        console.log(`  --> Bypassing ${className}->${methodName} (automatic exception
patch)`);

                        // This is not a perfect fix! Most unknown cases like this are really just
                        // checkCert(cert) methods though, so doing nothing is perfect, and if we
                        // do need an actual return value then this is probably the best we can do,
                        // and at least we're logging the method name so you can patch it manually:

                        if (returnTypeName === 'void') {
                            return;
                        } else {
                            return null;
                        }
                    };

                    console.log(`      [+] ${className}->${methodName} (automatic exception patch)`);
                } catch (e) {
                    console.log('      [ ] Failed to automatically patch failure');
                }

                return this.$init(str);
            };
            console.log('[+] SSLPeerUnverifiedException auto-patcher');
        } catch (err) {
            console.log('[ ] SSLPeerUnverifiedException auto-patcher');
        }

        /// -- Specific targeted hooks: -- ///

        // HttpsURLConnection
        try {
            const HttpsURLConnection = Java.use("javax.net.ssl.HttpsURLConnection");
            HttpsURLConnection.setDefaultHostnameVerifier.implementation = function (hostnameVerifier)
{
                console.log('  --> Bypassing HttpsURLConnection (setDefaultHostnameVerifier)');
```

```
            return; // Do nothing, i.e. don't change the hostname verifier
        };
        console.log('[+] HttpsURLConnection (setDefaultHostnameVerifier)');
    } catch (err) {
        console.log('[ ] HttpsURLConnection (setDefaultHostnameVerifier)');
    }
    try {
        const HttpsURLConnection = Java.use("javax.net.ssl.HttpsURLConnection");
        HttpsURLConnection.setSSLSocketFactory.implementation = function (SSLSocketFactory) {
            console.log('  --> Bypassing HttpsURLConnection (setSSLSocketFactory)');
            return; // Do nothing, i.e. don't change the SSL socket factory
        };
        console.log('[+] HttpsURLConnection (setSSLSocketFactory)');
    } catch (err) {
        console.log('[ ] HttpsURLConnection (setSSLSocketFactory)');
    }
    try {
        const HttpsURLConnection = Java.use("javax.net.ssl.HttpsURLConnection");
        HttpsURLConnection.setHostnameVerifier.implementation = function (hostnameVerifier) {
            console.log('  --> Bypassing HttpsURLConnection (setHostnameVerifier)');
            return; // Do nothing, i.e. don't change the hostname verifier
        };
        console.log('[+] HttpsURLConnection (setHostnameVerifier)');
    } catch (err) {
        console.log('[ ] HttpsURLConnection (setHostnameVerifier)');
    }

    // SSLContext
    try {
        const X509TrustManager = Java.use('javax.net.ssl.X509TrustManager');
        const SSLContext = Java.use('javax.net.ssl.SSLContext');

        const TrustManager = Java.registerClass({
            // Implement a custom TrustManager
            name: 'dev.asd.test.TrustManager',
            implements: [X509TrustManager],
            methods: {
                checkClientTrusted: function (chain, authType) { },
                checkServerTrusted: function (chain, authType) { },
                getAcceptedIssuers: function () { return []; }
            }
        });
```

```
            // Prepare the TrustManager array to pass to SSLContext.init()
            const TrustManagers = [TrustManager.$new()];

            // Get a handle on the init() on the SSLContext class
            const SSLContext_init = SSLContext.init.overload(
                '[Ljavax.net.ssl.KeyManager;', '[Ljavax.net.ssl.TrustManager;',
'java.security.SecureRandom'
            );

            // Override the init method, specifying the custom TrustManager
            SSLContext_init.implementation = function (keyManager, trustManager, secureRandom) {
                console.log('  --> Bypassing Trustmanager (Android < 7) request');
                SSLContext_init.call(this, keyManager, TrustManagers, secureRandom);
            };
            console.log('[+] SSLContext');
        } catch (err) {
            console.log('[ ] SSLContext');
        }

        // TrustManagerImpl (Android > 7)
        try {
            const array_list = Java.use("java.util.ArrayList");
            const TrustManagerImpl = Java.use('com.android.org.conscrypt.TrustManagerImpl');

            // This step is notably what defeats the most common case: network security config
            TrustManagerImpl.checkTrustedRecursive.implementation = function(a1, a2, a3, a4, a5, a6) {
                console.log('  --> Bypassing TrustManagerImpl checkTrusted ');
                return array_list.$new();
            }

            TrustManagerImpl.verifyChain.implementation = function (untrustedChain, trustAnchorChain,
host, clientAuth, ocspData, tlsSctData) {
                console.log('  --> Bypassing TrustManagerImpl verifyChain: ' + host);
                return untrustedChain;
            };
            console.log('[+] TrustManagerImpl');
        } catch (err) {
            console.log('[ ] TrustManagerImpl');
        }

        // OkHTTPv3 (quadruple bypass)
        try {
            // Bypass OkHTTPv3 {1}
```

```
            const okhttp3_Activity_1 = Java.use('okhttp3.CertificatePinner');
            okhttp3_Activity_1.check.overload('java.lang.String', 'java.util.List').implementation =
function (a, b) {
                console.log('  --> Bypassing OkHTTPv3 (list): ' + a);
                return;
            };
            console.log('[+] OkHTTPv3 (list)');
        } catch (err) {
            console.log('[ ] OkHTTPv3 (list)');
        }
        try {
            // Bypass OkHTTPv3 {2}
            // This method of CertificatePinner.check could be found in some old Android app
            const okhttp3_Activity_2 = Java.use('okhttp3.CertificatePinner');
            okhttp3_Activity_2.check.overload('java.lang.String',
'java.security.cert.Certificate').implementation = function (a, b) {
                console.log('  --> Bypassing OkHTTPv3 (cert): ' + a);
                return;
            };
            console.log('[+] OkHTTPv3 (cert)');
        } catch (err) {
            console.log('[ ] OkHTTPv3 (cert)');
        }
        try {
            // Bypass OkHTTPv3 {3}
            const okhttp3_Activity_3 = Java.use('okhttp3.CertificatePinner');
            okhttp3_Activity_3.check.overload('java.lang.String',
'[Ljava.security.cert.Certificate;').implementation = function (a, b) {
                console.log('  --> Bypassing OkHTTPv3 (cert array): ' + a);
                return;
            };
            console.log('[+] OkHTTPv3 (cert array)');
        } catch (err) {
            console.log('[ ] OkHTTPv3 (cert array)');
        }
        try {
            // Bypass OkHTTPv3 {4}
            const okhttp3_Activity_4 = Java.use('okhttp3.CertificatePinner');
            okhttp3_Activity_4['check$okhttp'].implementation = function (a, b) {
                console.log('  --> Bypassing OkHTTPv3 ($okhttp): ' + a);
                return;
            };
            console.log('[+] OkHTTPv3 ($okhttp)');
```

```
        } catch (err) {
            console.log('[ ] OkHTTPv3 ($okhttp)');
        }

        // Trustkit (triple bypass)
        try {
            // Bypass Trustkit {1}
            const trustkit_Activity_1 =
Java.use('com.datatheorem.android.trustkit.pinning.OkHostnameVerifier');
            trustkit_Activity_1.verify.overload('java.lang.String',
'javax.net.ssl.SSLSession').implementation = function (a, b) {
                console.log('  --> Bypassing Trustkit OkHostnameVerifier(SSLSession): ' + a);
                return true;
            };
            console.log('[+] Trustkit OkHostnameVerifier(SSLSession)');
        } catch (err) {
            console.log('[ ] Trustkit OkHostnameVerifier(SSLSession)');
        }
        try {
            // Bypass Trustkit {2}
            const trustkit Activity 2 =
Java.use('com.datatheorem.android.trustkit.pinning.OkHostnameVerifier');
            trustkit_Activity_2.verify.overload('java.lang.String',
'java.security.cert.X509Certificate').implementation = function (a, b) {
                console.log('  --> Bypassing Trustkit OkHostnameVerifier(cert): ' + a);
                return true;
            };
            console.log('[+] Trustkit OkHostnameVerifier(cert)');
        } catch (err) {
            console.log('[ ] Trustkit OkHostnameVerifier(cert)');
        }
        try {
            // Bypass Trustkit {3}
            const trustkit_PinningTrustManager =
Java.use('com.datatheorem.android.trustkit.pinning.PinningTrustManager');
            trustkit_PinningTrustManager.checkServerTrusted.implementation = function () {
                console.log('  --> Bypassing Trustkit PinningTrustManager');
            };
            console.log('[+] Trustkit PinningTrustManager');
        } catch (err) {
            console.log('[ ] Trustkit PinningTrustManager');
        }
```

```
        // Appcelerator Titanium
        try {
            const appcelerator_PinningTrustManager =
Java.use('appcelerator.https.PinningTrustManager');
            appcelerator_PinningTrustManager.checkServerTrusted.implementation = function () {
                console.log('  --> Bypassing Appcelerator PinningTrustManager');
            };
            console.log('[+] Appcelerator PinningTrustManager');
        } catch (err) {
            console.log('[ ] Appcelerator PinningTrustManager');
        }

        // OpenSSLSocketImpl Conscrypt
        try {
            const OpenSSLSocketImpl = Java.use('com.android.org.conscrypt.OpenSSLSocketImpl');
            OpenSSLSocketImpl.verifyCertificateChain.implementation = function (certRefs, JavaObject,
authMethod) {
                console.log('  --> Bypassing OpenSSLSocketImpl Conscrypt');
            };
            console.log('[+] OpenSSLSocketImpl Conscrypt');
        } catch (err) {
            console.log('[ ] OpenSSLSocketImpl Conscrypt');
        }

        // OpenSSLEngineSocketImpl Conscrypt
        try {
            const OpenSSLEngineSocketImpl_Activity =
Java.use('com.android.org.conscrypt.OpenSSLEngineSocketImpl');
            OpenSSLEngineSocketImpl_Activity.verifyCertificateChain.overload('[Ljava.lang.Long;',
'java.lang.String').implementation = function (a, b) {
                console.log('  --> Bypassing OpenSSLEngineSocketImpl Conscrypt: ' + b);
            };
            console.log('[+] OpenSSLEngineSocketImpl Conscrypt');
        } catch (err) {
            console.log('[ ] OpenSSLEngineSocketImpl Conscrypt');
        }

        // OpenSSLSocketImpl Apache Harmony
        try {
            const OpenSSLSocketImpl_Harmony =
Java.use('org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl');
            OpenSSLSocketImpl_Harmony.verifyCertificateChain.implementation = function
(asn1DerEncodedCertificateChain, authMethod) {
```

```
                console.log('  --> Bypassing OpenSSLSocketImpl Apache Harmony');
            };
            console.log('[+] OpenSSLSocketImpl Apache Harmony');
        } catch (err) {
            console.log('[ ] OpenSSLSocketImpl Apache Harmony');
        }

        // PhoneGap sslCertificateChecker (https://github.com/EddyVerbruggen/SSLCertificateChecker-
PhoneGap-Plugin)
        try {
            const phonegap_Activity = Java.use('nl.xservices.plugins.sslCertificateChecker');
            phonegap_Activity.execute.overload('java.lang.String', 'org.json.JSONArray',
'org.apache.cordova.CallbackContext').implementation = function (a, b, c) {
                console.log('  --> Bypassing PhoneGap sslCertificateChecker: ' + a);
                return true;
            };
            console.log('[+] PhoneGap sslCertificateChecker');
        } catch (err) {
            console.log('[ ] PhoneGap sslCertificateChecker');
        }

        // IBM MobileFirst pinTrustedCertificatePublicKey (double bypass)
        try {
            // Bypass IBM MobileFirst {1}
            const WLClient_Activity_1 = Java.use('com.worklight.wlclient.api.WLClient');

WLClient_Activity_1.getInstance().pinTrustedCertificatePublicKey.overload('java.lang.String').implemen
tation = function (cert) {
                console.log('  --> Bypassing IBM MobileFirst pinTrustedCertificatePublicKey (string):
' + cert);
                return;
            };
            console.log('[+] IBM MobileFirst pinTrustedCertificatePublicKey (string)');
        } catch (err) {
            console.log('[ ] IBM MobileFirst pinTrustedCertificatePublicKey (string)');
        }
        try {
            // Bypass IBM MobileFirst {2}
            const WLClient_Activity_2 = Java.use('com.worklight.wlclient.api.WLClient');

WLClient_Activity_2.getInstance().pinTrustedCertificatePublicKey.overload('[Ljava.lang.String;').imple
mentation = function (cert) {
```

```
                console.log('  --> Bypassing IBM MobileFirst pinTrustedCertificatePublicKey (string
array): ' + cert);
                return;
            };
            console.log('[+] IBM MobileFirst pinTrustedCertificatePublicKey (string array)');
        } catch (err) {
            console.log('[ ] IBM MobileFirst pinTrustedCertificatePublicKey (string array)');
        }

        // IBM WorkLight (ancestor of MobileFirst) HostNameVerifierWithCertificatePinning (quadruple
bypass)
        try {
            // Bypass IBM WorkLight {1}
            const worklight_Activity_1 =
Java.use('com.worklight.wlclient.certificatepinning.HostNameVerifierWithCertificatePinning');
            worklight Activity 1.verify.overload('java.lang.String',
'javax.net.ssl.SSLSocket').implementation = function (a, b) {
                console.log('  --> Bypassing IBM WorkLight HostNameVerifierWithCertificatePinning
(SSLSocket): ' + a);
                return;
            };
            console.log('[+] IBM WorkLight HostNameVerifierWithCertificatePinning (SSLSocket)');
        } catch (err) {
            console.log('[ ] IBM WorkLight HostNameVerifierWithCertificatePinning (SSLSocket)');
        }
        try {
            // Bypass IBM WorkLight {2}
            const worklight_Activity_2 =
Java.use('com.worklight.wlclient.certificatepinning.HostNameVerifierWithCertificatePinning');
            worklight_Activity_2.verify.overload('java.lang.String',
'java.security.cert.X509Certificate').implementation = function (a, b) {
                console.log('  --> Bypassing IBM WorkLight HostNameVerifierWithCertificatePinning
(cert): ' + a);
                return;
            };
            console.log('[+] IBM WorkLight HostNameVerifierWithCertificatePinning (cert)');
        } catch (err) {
            console.log('[ ] IBM WorkLight HostNameVerifierWithCertificatePinning (cert)');
        }
        try {
            // Bypass IBM WorkLight {3}
            const worklight_Activity_3 =
Java.use('com.worklight.wlclient.certificatepinning.HostNameVerifierWithCertificatePinning');
```

```
            worklight_Activity_3.verify.overload('java.lang.String', '[Ljava.lang.String;',
'[Ljava.lang.String;').implementation = function (a, b) {
                console.log('  --> Bypassing IBM WorkLight HostNameVerifierWithCertificatePinning
(string string): ' + a);
                return;
            };
            console.log('[+] IBM WorkLight HostNameVerifierWithCertificatePinning (string string)');
        } catch (err) {
            console.log('[ ] IBM WorkLight HostNameVerifierWithCertificatePinning (string string)');
        }
        try {
            // Bypass IBM WorkLight {4}
            const worklight_Activity_4 =
Java.use('com.worklight.wlclient.certificatepinning.HostNameVerifierWithCertificatePinning');
            worklight_Activity_4.verify.overload('java.lang.String',
'javax.net.ssl.SSLSession').implementation = function (a, b) {
                console.log('  --> Bypassing IBM WorkLight HostNameVerifierWithCertificatePinning
(SSLSession): ' + a);
                return true;
            };
            console.log('[+] IBM WorkLight HostNameVerifierWithCertificatePinning (SSLSession)');
        } catch (err) {
            console.log('[ ] IBM WorkLight HostNameVerifierWithCertificatePinning (SSLSession)');
        }

        // Conscrypt CertPinManager
        try {
            const conscrypt_CertPinManager_Activity =
Java.use('com.android.org.conscrypt.CertPinManager');
            conscrypt_CertPinManager_Activity.isChainValid.overload('java.lang.String',
'java.util.List').implementation = function (a, b) {
                console.log('  --> Bypassing Conscrypt CertPinManager: ' + a);
                return true;
            };
            console.log('[+] Conscrypt CertPinManager');
        } catch (err) {
            console.log('[ ] Conscrypt CertPinManager');
        }

        // CWAC-Netsecurity (unofficial back-port pinner for Android<4.2) CertPinManager
        try {
            const cwac_CertPinManager_Activity =
Java.use('com.commonsware.cwac.netsecurity.conscrypt.CertPinManager');
```

```
            cwac_CertPinManager_Activity.isChainValid.overload('java.lang.String',
'java.util.List').implementation = function (a, b) {
                console.log('  --> Bypassing CWAC-Netsecurity CertPinManager: ' + a);
                return true;
            };
            console.log('[+] CWAC-Netsecurity CertPinManager');
        } catch (err) {
            console.log('[ ] CWAC-Netsecurity CertPinManager');
        }

        // Worklight Androidgap WLCertificatePinningPlugin
        try {
            const androidgap_WLCertificatePinningPlugin_Activity =
Java.use('com.worklight.androidgap.plugin.WLCertificatePinningPlugin');
            androidgap_WLCertificatePinningPlugin_Activity.execute.overload('java.lang.String',
'org.json.JSONArray', 'org.apache.cordova.CallbackContext').implementation = function (a, b, c) {
                console.log('  --> Bypassing Worklight Androidgap WLCertificatePinningPlugin: ' + a);
                return true;
            };
            console.log('[+] Worklight Androidgap WLCertificatePinningPlugin');
        } catch (err) {
            console.log('[ ] Worklight Androidgap WLCertificatePinningPlugin');
        }

        // Netty FingerprintTrustManagerFactory
        try {
            const netty_FingerprintTrustManagerFactory =
Java.use('io.netty.handler.ssl.util.FingerprintTrustManagerFactory');
            netty_FingerprintTrustManagerFactory.checkTrusted.implementation = function (type, chain)
{
                console.log('  --> Bypassing Netty FingerprintTrustManagerFactory');
            };
            console.log('[+] Netty FingerprintTrustManagerFactory');
        } catch (err) {
            console.log('[ ] Netty FingerprintTrustManagerFactory');
        }

        // Squareup CertificatePinner [OkHTTP<v3] (double bypass)
        try {
            // Bypass Squareup CertificatePinner {1}
            const Squareup_CertificatePinner_Activity_1 =
Java.use('com.squareup.okhttp.CertificatePinner');
```

```
            Squareup_CertificatePinner_Activity_1.check.overload('java.lang.String',
'java.security.cert.Certificate').implementation = function (a, b) {
                console.log('  --> Bypassing Squareup CertificatePinner (cert): ' + a);
                return;
            };
            console.log('[+] Squareup CertificatePinner (cert)');
        } catch (err) {
            console.log('[ ] Squareup CertificatePinner (cert)');
        }
        try {
            // Bypass Squareup CertificatePinner {2}
            const Squareup_CertificatePinner_Activity_2 =
Java.use('com.squareup.okhttp.CertificatePinner');
            Squareup_CertificatePinner_Activity_2.check.overload('java.lang.String',
'java.util.List').implementation = function (a, b) {
                console.log('  --> Bypassing Squareup CertificatePinner (list): ' + a);
                return;
            };
            console.log('[+] Squareup CertificatePinner (list)');
        } catch (err) {
            console.log('[ ] Squareup CertificatePinner (list)');
        }

        // Squareup OkHostnameVerifier [OkHTTP v3] (double bypass)
        try {
            // Bypass Squareup OkHostnameVerifier {1}
            const Squareup_OkHostnameVerifier_Activity_1 =
Java.use('com.squareup.okhttp.internal.tls.OkHostnameVerifier');
            Squareup_OkHostnameVerifier_Activity_1.verify.overload('java.lang.String',
'java.security.cert.X509Certificate').implementation = function (a, b) {
                console.log('  --> Bypassing Squareup OkHostnameVerifier (cert): ' + a);
                return true;
            };
            console.log('[+] Squareup OkHostnameVerifier (cert)');
        } catch (err) {
            console.log('[ ] Squareup OkHostnameVerifier (cert)');
        }
        try {
            // Bypass Squareup OkHostnameVerifier {2}
            const Squareup_OkHostnameVerifier_Activity_2 =
Java.use('com.squareup.okhttp.internal.tls.OkHostnameVerifier');
            Squareup_OkHostnameVerifier_Activity_2.verify.overload('java.lang.String',
'javax.net.ssl.SSLSession').implementation = function (a, b) {
```

```
                console.log('  --> Bypassing Squareup OkHostnameVerifier (SSLSession): ' + a);
                return true;
            };
            console.log('[+] Squareup OkHostnameVerifier (SSLSession)');
        } catch (err) {
            console.log('[ ] Squareup OkHostnameVerifier (SSLSession)');
        }

        // Android WebViewClient (double bypass)
        try {
            // Bypass WebViewClient {1} (deprecated from Android 6)
            const AndroidWebViewClient_Activity_1 = Java.use('android.webkit.WebViewClient');
            AndroidWebViewClient_Activity_1.onReceivedSslError.overload('android.webkit.WebView',
'android.webkit.SslErrorHandler', 'android.net.http.SslError').implementation = function (obj1, obj2,
obj3) {
                console.log('  --> Bypassing Android WebViewClient (SslErrorHandler)');
            };
            console.log('[+] Android WebViewClient (SslErrorHandler)');
        } catch (err) {
            console.log('[ ] Android WebViewClient (SslErrorHandler)');
        }
        try {
            // Bypass WebViewClient {2}
            const AndroidWebViewClient_Activity_2 = Java.use('android.webkit.WebViewClient');
            AndroidWebViewClient_Activity_2.onReceivedSslError.overload('android.webkit.WebView',
'android.webkit.WebResourceRequest', 'android.webkit.WebResourceError').implementation = function
(obj1, obj2, obj3) {
                console.log('  --> Bypassing Android WebViewClient (WebResourceError)');
            };
            console.log('[+] Android WebViewClient (WebResourceError)');
        } catch (err) {
            console.log('[ ] Android WebViewClient (WebResourceError)');
        }

        // Apache Cordova WebViewClient
        try {
            const CordovaWebViewClient_Activity = Java.use('org.apache.cordova.CordovaWebViewClient');
            CordovaWebViewClient_Activity.onReceivedSslError.overload('android.webkit.WebView',
'android.webkit.SslErrorHandler', 'android.net.http.SslError').implementation = function (obj1, obj2,
obj3) {
                console.log('  --> Bypassing Apache Cordova WebViewClient');
                obj3.proceed();
            };
```

```
        } catch (err) {
            console.log('[ ] Apache Cordova WebViewClient');
        }

        // Boye AbstractVerifier
        try {
            const boye_AbstractVerifier =
Java.use('ch.boye.httpclientandroidlib.conn.ssl.AbstractVerifier');
            boye_AbstractVerifier.verify.implementation = function (host, ssl) {
                console.log('  --> Bypassing Boye AbstractVerifier: ' + host);
            };
        } catch (err) {
            console.log('[ ] Boye AbstractVerifier');
        }

               // Appmattus
               try {
            const appmatus Activity =
Java.use('com.appmattus.certificatetransparency.internal.verifier.CertificateTransparencyInterceptor')
;
            appmatus Activity['intercept'].implementation = function (a) {
                console.log('  --> Bypassing Appmattus (Transparency)');
                return a.proceed(a.request());
            };
            console.log('[+] Appmattus (Transparency)');
        } catch (err) {
            console.log('[ ] Appmattus (Transparency)');
        }

        console.log("Unpinning setup completed");
        console.log("---");
    });

}, 0);
```