Sicherheitsrisiken und Schutzmechanismen in IoT-Anwendungen

Florian Hansen

Hochschule Flensburg
florian.hansen@stud.hs-flensburg.de

Michael Frank

Hochschule Flensburg
michael.frank@stud.hs-flensburg.de

I. Einleitung

Internet of Things (IoT) bezeichnet die Vernetzung von Geräten über das Internet. Diese werden oftmals, aufgrund der Verbindung zum Internet, auch als *intelligente Geräte (engl. smart devices)* bezeichnet. Die Endgeräte können dabei sehr unterschiedlich sein, so reichen diese von einfachen Messgeräten bis hin zu komplexen medizinischen Implantaten [9].

Aufgrund der Anbindung der Geräte an das Internet, ist die Kommunikationen mit diesen einfacher und Daten können leichter extrahiert und übermittelt werden. Diese Prozesse können zudem automatisiert werden, wodurch keine weitere menschliche Interaktion benötigt wird und Zeit bzw. Kosten eingespart werden können. Dadurch ist nicht verwunderlich, dass die Anzahl an IoT-Geräten in den letzten Jahren stark zugenommen hat. So sollen bis Ende 2020 über 25 Milliarden Geräte über das Internet vernetzt sein [9].

Die beeindruckende Anzahl an IoT-Geräten zeigt, dass die Technologie sich heutzutage etabliert hat, doch sollten hierbei nicht die Sicherheitsrisiken bei der Verwendung von IoT-Geräten außer Acht gelassen werden. Abhängig vom Anwendungsbereich der Geräte können diese für essentielle Funtkionalitäten eines System zuständig sein oder mit sehr sensiblen Daten arbeiten. Geräte, welche nicht ausreichend geschützt sind, können durch Angriffe bspw. heruntergefahren werden, was in Abhängigkeit zum Anwendungsbereich zu erheblichen Schäden führen kann [9].

Im Rahmen diese Ausarbeitung werden die Anwendungsbereiche, Sicherheitsrisiken und Schutzmechanismen von IoT-Geräten vorgestellt und anhand eines praktischen Beispiels demonstriert.

II. Anwendungsbereiche

Aufgrund der allgemeinen Digitalisierung werden heutzutage IoT-Geräte in vielen Branchen und Abläufen umfangreich eingesetzt. In dem folgenden Kapitel sollen einige Anwendungsbereiche der IoT vorgestellt werden.

a) Umweltbeobachtung.: Bei der Umweltbeobachtung werden bestimmte Bereiche der Umwelt auf bestimmte ökologische Parameter überwacht. So werden bspw. die Luft-, Wasser- oder Bodenqualität von bestimmten Gebieten oder Standorten untersucht, um potentielle Umweltprobleme oder -gefahren frühzeitig feststellen zu können. Ein detailliertes Beispiel wäre die Erdbebenüberwachung, bei der IoT-Geräte mit seismischen Sensoren ausgestattet werden, um frühzeitig Erdbeben zu erkennen. Im Falle eines Erdbebens können die Geräte weitere System benachrichtigen, welche daraufhin Versorgungseinrichtungen oder Datenzentren abschalten, sowie Menschen in kritischen Wohngebieten benachrichtigen können [9].

b) Gesundheitswesen.: Ärzte benötigen oftmals stetige Information zum körperlichen Zustand eines Patienten, um herauszufinden, ob eine bestimmte Behandlung zum gewünschten Erfolg führt. IoT-Geräte, im speziellen kleine medizinische Implantate, können Ärzte dabei unterstützen bestimmte Gesundheitswerte auszulesen bzw. selbständig auszuwerten. Ärzte können dadurch einfacher Untersuchungen und Diagnosen durchführen [9].

c) Industrie.: In der Produktion von Gütern finden IoT-Geräte sehr unterschiedliche Verwendungszwecke. Mithilfe der IoT können bspw. gesamte Lieferketten optimiert werden, indem IoT-Geräte bestimmte Teilprozesse (Produktion, Transport, etc.) überwachen und in bestimmten Situation zielgerichtete Maßnahmen einleiten. Außerdem werden mithilfe von IoT-Geräten eine Vielzahl von Daten ausgehoben, welche in wachsenden Anwendungsbereichen wie Big Data Analysen oder Künstlicher Intelligenz benötigt werden [9].

d) Wearables.: Bei Wearables handelt es sich um technische Geräte, welche direkt am Körper getragen werden und mit verschiedenen Sensoren ausgestattet sind. Die bekanntesten Beispiele für Wearabeles sind Smart-Watches und Fitnessarmbänder und werden für unterschiedliche Anwendungsbereiche wie bspw. Gesundheitswesen, Sport oder Entertainment eingesetzt [9].

e) Spielzeuge.: IoT-Geräte in Form von Spielzeugen sind oftmals mit Lautsprecher, Mikrofonen und weiteren technischen Komponenten ausgestattet. Funktionalitäten, welche klassische Spielzeuge nicht haben, wären bspw. eine integrierte Spracherkennung, bei dir die Aussagen der Kinder aufgenommen, über ein weiteres System ausgewertet werden und das Spielzeug dementsprechend reagiert. Aufgrund der Verbindung zum Internet stehen den Spielzeugen dadurch eine Vielzahl von neuen Möglichkeiten zur Verfügung [9].

III. Angriffsmöglichkeiten

Wie in fast jedem anderen System müssen vor allem IoT-Anwendungen besonders bezüglich ihrer Sicherheit betrachtet werden. Die Anwendungsentwicklung innerhalb des IoT ist

geprägt von vielseitig komplexen Problemen beim Etablieren der IT-Sicherheit. Grundsätzlich können Sicherheitsrisiken in drei Bereichen auftreten, die hier kurz erläutert werden sollen [9].

A. Herausforderungen bei mobilen Geräten

a) Heterogenität: Dieser Bereich behandelt Probleme, die durch die Vielzahl an Anwendungsmöglichkeiten in dem IoT entstehen. Dazu zählen unter anderem die verbauten Hardware-Komponenten, Protokolle für die Kommunikation unter Geräten und Variationen zwischen Geräten und ihrer Rechenleistung [9].

b) Ressourcenbeschränkung: Nicht jedes Gerät beinhaltet einen leistungsstarken Rechner. Oft stößt man bei der Entwicklung von IoT-Geräten auf das Problem, dass nicht genug Speicher oder Rechenleistung vorhanden ist. Nicht nur bei der Implementation der eigentlichen Anwendung macht dies Probleme. Bereits etablierte Sicherheitsalgorithmen können aufgrund fehlender Leistung nicht verwendet werden, da sie sich als unpraktikabel in dem IoT herausstellen. Sicherheitsziele können also aufgrund nicht ausreichender Hardware nicht erreicht werden [9]. Ein Beispiel hierfür ist das Arbeiten mit einer Cloud. Sicherheitsziele sind hier unter anderem die Privatsphäre des Benutzers. Die üblichen kryptographischen Algorithmen, z.B. die attribut-basierte Verschlüsselungen und Pairings, stellen sich als sehr unpraktikabel heraus, da diese zu viel Rechenleistung für die Berechnungen von bilinearen Abbildungen benötigen. Eine vielversprechende Lösung ist das Auslagern rechenintensiver Schritte an einen leistungsstarken Server [4].

c) Dynamische Netzwerktopologie: Gerade bei mobilen Geräten innerhalb des IoT haben wir mit “losen” Verbindungen zu tun. Zum Beispiel ist es möglich, dass ein Smartphone mit diversen WLAN-Hotspots interagiert und damit keine feste Position innerhalb eines Netzwerkes besitzt [9].

B. Definition von Sicherheit

Damit die Sicherheit trotz der oben erwähnten Herausforderungen gewährleistet werden kann, genügt es nicht, dass allseits bekannte CIA-Dreieck zu betrachten. Laut [9] sind Vertraulichkeit, Integrität und Verfügbarkeit nicht die einzigen Voraussetzungen, um Sicherheit für IoT-Geräte zu definieren. Stattdessen müssen individuelle Anforderungen für jeden Anwendungsbereich definiert werden.

Komplexe Anlagen besitzen zum Beispiel verschiedenste Sensoren und Aktoren, um beispielsweise Verfahrenstechniken anzuwenden und diese zu überwachen. Diese Sensoren werden häufig beim Implementieren von IT-Sicherheit vernachlässigt, nachdem sie verbaut wurden. Mögliche Sicherheitsanforderungen für diesen Bereich sind Vertraulichkeit, Authentifizierung, Nachrichtenauthentizität, Integrität, Verfügbarkeit, Zuverlässigkeit, Frische der Daten und Fälschungsdetektierung [9].

Andere Bereiche des IoT benötigen jedoch andere Anforderungen an die IT-Sicherheit. Im Gesundheitswesen wird die Privatsphäre des Patienten eine größere Rolle spielen, als bei technischen Anlagen, sodass die Anforderungen abweichen. Diese sind hier unter anderem Verfügbarkeit, Zuverlässigkeit, Authentifizierung, Integrität, Vertraulichkeit, Privatsphäre, Nachrichtenauthentizität, Fälschungsdetektierung, Verantwortlichkeit und Nichabstreitbarkeit [9].

C. Sicherheitsrisiken pro Domäne

Im vorherigen Abschnitt wurde auf verschiedene IoT-Domänen und ihren Sicherheitsanforderungen eingegangen. Kurz zusammengefasst ist jede Domäne unterschiedlich zu behandeln, wenn es um Sicherheit geht. Angreifer, die Schwachstellen in solchen Systemen finden, haben unter Umständen Zugriff auf eine große Menge von sensiblen Daten. Mögliche Angriffe können auf Grundlage der Sicherheitsanforderungen aus Abschnitt III-B abgeleitet werden [9] und werden in Tabelle V-A1 dargestellt.

IV. Verteidigung gegen Angriffe

Um Angriffen, die in Tabelle V-A1 genannt wurden, entgegenzuwirken, sollen nun einige Maßnahmen für die Abwehr besprochen werden.

a) Gegenmaßnahmen gegen unsichere Web-Interfaces und Netzwerkdienste.: Damit ein Fluten von Authentifizierungsanfragen verhindert werden kann, können Standardports geblockt werden. Für SSH ist dies beispielsweise der Port 22 und für Telnet der Port 23 bzw. 2323. Grundsätzlich sollten nur die Ports geöffnet werden, die wirklich benötigt werden. Auch das *Universial Plug and Play Protocol* (UPnP) sollte deaktiviert werden. Außerdem werden häufig Standardeinstellungen wie Benutzernamen und Passwörter unverändert übernommen. Dies ist offensichtlich ein großes Problem, denn ein Angreifer wird diese Information ohne großen Aufwand herausfinden können. Man sollte also darauf achten, individuelle und starke Passwörter zu verwenden. Auch die Software selbst sollte regelmäßig Updates eingespielt bekommen, denn häufig werden Schwachstellen erst nach einem Release festgestellt und Patches veröffentlicht [9].

b) Gegenmaßnahmen gegen Routing-Protokolle.: Da IoT-Geräte sehr eingeschränkter Natur sind, ist das Schützen von IoT-Netzwerken eine echte Herausforderung [7]. Sinkhole-Angriffe können laut [9] verhindert werden, indem robuste Authentifikation-Schemata, geografisches Routing und systematisches Rerouting verwendet werden. Geografisches Routing bedeutet, dass Datenpakete abhängig der Position von Knotenpunkten und der Zieladresse innerhalb des Netzwerks umgeleitet werden. Die Chancen, dass Forwarding- und Black-Hole-Angriffe durchführbar sind, können mithilfe von Source-Routing reduziert werden. Verwendet man Multipath-Routing, dann können Selective-Forwarding-Angriffe abgewehrt werden. Hierbei werden verschiedene Pfade innerhalb des Netzwerks verwendet. Hello-Flood-Angriffe können durch bidirektionale Authentifikation verhindert werden. Wormhole-Angriffe können ebenfalls durch geografisches Routing erschwert durchführbar gemacht werden. Aber auch durch physische Überwachung oder Source-Routing können Angriffe dieser Art verhindert

Anwendungsdomäne	Mögliche Angriffe
Umgebungsüberwachung	DoS/DDoS, MitM, Node capture, Sinkhole, Hello flood, Traffic analysis, Hacking, Side channel, Sybil, Selective forwarding, Back hole, Tampering, Wormhole, Masquerading
Gesundheitswesen	DoS/DDoS, MitM, Malicious code, Spoofing, Hacking, Tampering, Eavesdropping, Hijacking, Replay, Backdoof, Tag tracking, Tag cloning, Identity theft, Masquerading, Node capture, Side channel
Feuerwehr	DoS/DDoS, MitM, Sybil, Hello flood, Node capture, Sinkhole, Black hole, Selective forwarding, Hacking, Tampering, Malicious code, Hijacking, Side channel, Traffic analysis, GPS jamming
Herstellung	DoS/DDoS, MitM, Masquerading, Backdoor, Identity theft, Replay, Hijacking, Hacking, Eavesdropping, Selective forwarding, Back hole, Sinkhole, Node capture, Sybil, Spoofing, Traffic analysis, Side channel, Tampering, Wormhole, Malicious code, Wormhole, Economic espionage
Wearables	DoS/DDoS, Eavesdropping, MitM, Malicious code, Identity theft, Hacking, Backdoor, Hijacking, inappropriate network configuration
Spielzeuge	DoS/DDoS, Eavesdropping, MitM, Identity theft, Hijacking, Hacking, Backdoor, Masquerading, Spoofing, Malicious code, inappropriate network configuration

Tabelle I
SICHERHEITSRISIKEN PRO DOMÄNE [9]

werden. Gegenmaßnamen gegen Sybil-Angriffe ist die Evaluierung der Einzigartigkeit von Geräten im Netzwerk. Dies bedeutet lediglich, dass jedes Gerät eine eindeutige Identifikationsnummer besitzen muss. Auch mithilfe von sogenannten Random-Key-Pre-Distribution-Schemata [6] können Sybil-Angriffe verhindert werden.

c) Gegenmaßnahmen gegen unerlaubten Informationstransport.: Bevor ein Smart-Device einen Datenaustausch beginnt, sollte sich dieses Gerät vorerst authentifizieren. Ebenfalls sollte eine beidseitige Authentifizierung implementiert werden, damit beide Parteien verifizieren können, ob es sich um den richtigen Endpunkt handelt und sie mit dem gewünschten Ziel kommunizieren. Die Idee hierbei ist, dass zum Beispiel digitale Signaturen (SHA, ECDSA) bzw. symmetrische äquivalente (HMAC) ausgetauscht werden. Dies dient vor allem für die Einhaltung der Integrität der Daten. Ebenfalls können Signaturen verwendet werden, um einen Secure-Boot zu implementieren. Dies hilft dabei, dass das IoT-Gerät nur Codes ausführt, die vertrauenswürdig, also signiert sind. Vor allem wird dadurch verhindert, dass zum Beispiel die Firmware des Geräts überspielt wird. Um die Angriffe abzuwehren, die die Vertraulichkeit betreffen, werden Informationen über einen sicheren, verschlüsselten Kanal übertragen. Dies betrifft auch die Kommunikation mit externen Diensten, wie einer Cloud [9].

d) Gegenmaßnahmen gegen physische Angriffe.: Um zum Beispiel Node-Capturing zu verhindern, müssen Geräte physisch versteckt werden, um den Zugang zu erschweren. Außerdem können die Geräte so gebaut werden, sodass diese nur sehr schwer auseinanderbaubar sind. USB-Ports sollten zudem auch geschützt werden, damit ein Angreifer keine Schadsoftware über diese einspielen kann. Schutzmaßnahmen gegen das Manipulieren von Daten können durch regelmäßige Änderung der Schlüssel eingeführt werden. Side-Channel-Angriffe können durch spezielle, resistente Chipsätze verhindert werden. Das Messen von elektromagnetische Strahlung muss beim Entwickeln von IoT-Geräten ebenfalls berücksichtigt werden. Das Verschleiern von Informationen auf diesem Wege muss also ebenfalls implementiert werden, damit Angreifer nicht mithilfe von elektrischen Signalen Zugriff bekommen [9].

e) Gegenmaßnahmen gegen GPS-Jamming.: Um GPS-Jamming-Angriffe abzuwehren, können Kerbfilter eingesetzt werden [5], [9].

f) Gegenmaßnahmen gegen Tag-Tracking und -Cloning.: Eine Schutzmaßnahme gegen das Tag-Tracking ist, dass die Tags gegenüber einem Angreifer zufällig aussehen und eine gleiche Verteilung besitzen, damit dieser keine Schlüsse aus dem Aufbau der Tags ziehen kann. Um Tag-Cloning zu verhindern, muss sichergestellt werden, dass die eigentlichen Informationen unter dem Tag nicht zugänglich sind, um einen neuen validen Tag zu erzeugen [9].

g) Gegenmaßnahmen gegen falsche Netwerkkonfiguration.: Die Schutzmaßnahme ist hier sozialer Natur. Ein Schulen der Benutzer über die Wichtigkeit der IT-Sicherheit ist unabdingbar. Es müssen starke Regeln für Passwörter verwendet und sicherheitsrelevantes Logging aktiviert werden [9].

V. EXPERIMENT

In diesem Kapitel werden die Sicherheitsgefahren und -risiken von IoT-Geräten demonstriert. Dafür wurde eine Raspberry Pi-Sicherheitskamera mithilfe der Open-Source-Bibliothek *pi-aREST* [10], [11] entwickelt. Die Sicherheitskamera sendet mithilfe von *pi-aREST* in einem festgelegten Zeitintervall ein Bild, des überwachten Bereichs, an ein externes System. Die Entscheidung *pi-aREST* als Bibliothek für die Entwicklung des IoT-Gerätes zu verwenden, hatte mehrere Gründe. Ein Grund war, dass die Bibliothek im speziellen für die Arbeit mit dem Raspberry Pi entwickelt wurde und eine direkte Implementierung für das Erstellen von Bildern mithilfe einer angeschlossenen Kamera enthält. Außerdem ist der Quellcode Open-Source, wodurch die Bibliothek einfacher untersucht und auf potentielle Sicherheitsrisiken überprüft werden kann [10]. Der Entwickler der Bibliothek bietet zudem einen eigenen Überwachungs- und Kontroll-Service, für die mit der Bibliothek entwickelten IoT-Geräte, an. Wodurch die Untersuchung der möglichen Sicherheitsprobleme anhand von realistischen Szenarien demonstriert werden kann [1].

A. pi-aREST

pi-aREST ist eine Bibliothek für die JavaScript-Laufzeitumgebung *NodeJS* [2], welche mithilfe des Package-Managers *npm* [11] in eine Node-Anwendung

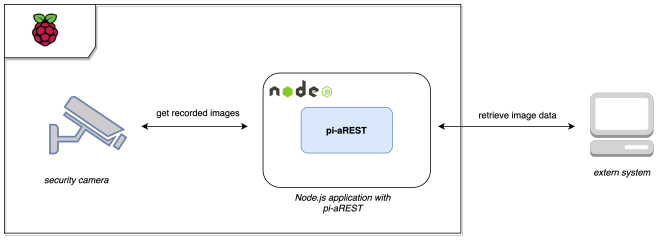

Abbildung 1. Sicherheitskamera-Architektur

importiert werden kann. Durch das Einbinden von *pi-aREST* in eine eigene Node-Anwendung werden Schnittstellen zur Kommunikation mit anderen Systemen und Geräten geöffnet. Der Datenaustausch erfolgt dabei mithilfe der Protokolle *HTTP* und *MQTT* und kann sowohl in lokalen Netzwerken, als auch im Internet stattfinden [10].

1) HTTP-Server: Nachdem Ausführen der Node-Anwendung mit der importierten Bibliothek, startet *pi-aREST* einen Express-HTTP-Server auf dem IoT-Gerät zum Bereitstellen einer REST-API. Die API besitzt verschiedene Endpunkte, welche entweder innerhalb des Netzwerkes oder aus dem Internet aufgerufen werden können. [10]

2) MQTT-Client: Message Queuing Telemetry Transport (MQTT) ist ein Open-Source Kommunikationsprotokoll auf der Anwendungsebene. Das Protokoll zeichnet sich durch ressourcensparende Datenpakete mit einem geringen Overhead aus, wodurch es bspw. oftmals in der Kommunikation zwischen IoT-Geräten (M2M) eingesetzt wird. Das Protokoll basiert auf der Publish/Subscribe-Architektur. So nehmen Clients die Rollen eines *Publishers (Sender)* oder *Subscribers (Empfänger)* ein und kommunizieren über den sogenannten *Broker* miteinander. [8], [12]

a) Publisher: Dieser Client sendet Nachrichten an einen *Broker*, wobei die übersendeten Nachrichten ein spezielles *Topic (Thema)* enthalten. Das übergebene *Topic* ist wichtig für das Weiterleiten der Nachrichten an andere Clients [8], [12].

b) Subscriber: Im Gegensatz zum *Publisher* sendet der *Subscriber* keine Nachrichten an *Broker*, sondern abonniert *(subscribe)* beim *Broker* ein bestimmtes *Topic*. Wenn ein *Publisher* eine Nachricht mit dem abonnierten *Topic* an den *Broker* sendet, erhalten alle *Subscriber* dieses *Topics* die Nachricht [8], [12].

c) Broker: Die Funktionalität des *Brokers* wurde bereits teilweise bei den anderen Protokoll-Teilnehmern beschrieben. Seine Hauptaufgabe ist das Steuern des Datenverkehrs zwischen den verschiedenen Clients. So nimmt dieser die Nachrichten der *Publisher* an und sendet diese nur an die Subscriber, die das jeweilige Topic abonniert haben [8], [12].

Beim Ausführen der Node-Anwendung mit *pi-aREST* verbindet sich das IoT-Gerät als Publisher mit einem MQTT-Broker über den Port 1883. Die Broker-Adresse wird von der Node-Anwendung vorgegeben. *pi-aREST* sendet daraufhin die Verbindungsanfrage an den Broker. Das IoT-Gerät sendet nach dem erfolgreichen Verbindungsaufbau in einem regelmäßigen

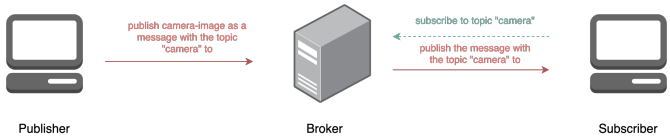

Abbildung 2. MQTT-Architektur für die Node-Anwendung [8]

Zeitintervall die Daten mit einem speziellen Topic an den Broker. In diesem Experiment sind dies die Überwachungsbilder.

VI. Testaufbau

Bevor wir mit der Durchführung des Experiments beginnen, stellen wir im Folgenden den Aufbau unserer Testumgebung vor. Da es sich bei [9] um ein Framework handelt, welches Entwickler dabei unterstützen soll, neue IoT-Anwendungen zu entwerfen, müssen wir uns Gedanken über ein mögliches Szenario machen. Wir haben uns dazu entschieden, eine einfache Sicherheitskamera zu entwickeln. Hierfür wird im ersten Schritt nach bekannten Frameworks geschaut, deren Anwendungszweck im IoT-Bereich zu finden ist. Anschließend sollen diese Frameworks verwendet werden, um die Beispielanwendung umzusetzen. Dann wird mithilfe des Leitfadens von [9] untersucht, ob das entstandene Produkt bezüglich der IT-Sicherheit bestanden hat.

A. Wahl der Komponenten

Wir haben uns für *aREST* als Framework entschieden, da der Autor dieses selbst für seinen Online-Dienst verwendet und das Repository auf GitHub unter allen Projekten, die MQTT als Protokoll verwenden, einen überdurchschnittlichen Bekanntheitsgrad besitzt. Mithilfe von *aREST* und NodeJS wird eine Sicherheitskamera entwickelt, die in Echtzeit Bilder übertragen soll. Als Plattform wird ein Raspberry PI 3B+ mit Raspbian Lite als Betriebssystem und ein entsprechendes Kameramodul verwendet.

Um zusätzliche Knotenpunkte im Netzwerk zu erzeugen, werden insgesamt zwei weitere, virtuelle Maschinen (VM) aufgesetzt und mit dem Netzwerk verbunden. Die erste VM fungiert als MQTT-Broker und versorgt alle Abonnenten mit entsprechenden Informationen. Geräte, die bestimmte Topics abonnieren, erhalten dementsprechend eine Nachricht, falls sich der abonnierte Wert verändert. So kann das Netzwerk theoretisch um weitere Knotenpunkte bzw. Endgeräte erweitert werden und wir erhalten damit eine realistische Abbildung eines solchen Szenarios. Die zweite VM stellt den Eindringling bzw. Angreifer des Systems dar und hat später die Aufgabe, sensible Daten auszuspähen und einzelne Knotenpunkte zu attackieren. Der gesamte Aufbau wird in Abbildung 3 dargestellt.

B. Festlegung sensibler Daten

Damit aussagekräftige Ergebnisse erzielt werden können, muss in erster Linie festgestellt werden, welche Daten überhaupt sensibel sind. Die Unterscheidung ist maßgebend für die Analyse der IT-Sicherheit des Systems und hängt immer vom Kontext der Anwendung ab [9]. Zum Beispiel

Endpunkt	Aufgabe
/	Liest die allgemeinen Raspberry Pi-Informationen (ID, Name) und weitere Metadaten, die in einem JavaScript Objekt festgehalten sind, aus und gibt diese zurück
/:variable	Anhand des übergebenen Parameters *variable* wird in einem JavaScript-Objekt, dass Metadaten zum Raspberry Pi enthält, überprüft, ob dieses ein Attribut mit diesem Namen enthält. Wenn dies der Fall ist, wird es an den Anfragenden zurückgegebenen. Unter JavaScript kann das Attribut auch eine Funktion sein, falls dies der Fall ist, wird diese ausgeführt und das Ergebnis zurückgegeben.
/camera/snapshot	Erstellt ein Bild durch die am Raspberry PI angeschlossene Kamera und hinterlegt dieses im Dateisystem.
/:command/:pin	Liest den Status des über den Parameter *pin* angegebenen digitalen Pins aus und sendet diesen Status zurück.
/digital/:pin/:state	Ermöglicht das Setzten eines Status zu einem Pin, indem über den Parameter *pin* der zu aktualisierende Pin ausgewählt wird und der Parameter *state* den neuen Status festlegt.

Tabelle II
ENDPUNKTE EINES *pi-aREST*-HTTP-SERVERS [10]

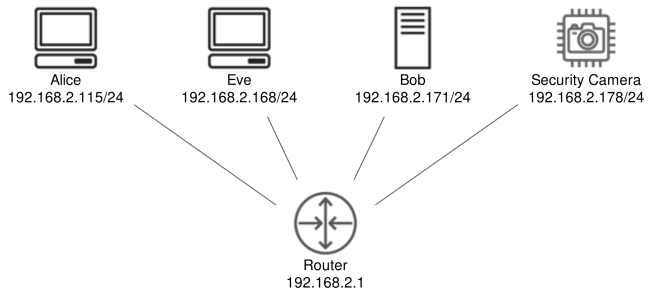

Abbildung 3. Darstellung des Testaufbaus und dessen Komponenten

ist es möglicherweise erwünscht, dass Wetterdaten öffentlich einsehbar sind. Auch wenn diese von einem Sensor ausgelesen werden, der sich nicht im Besitz des Anfragenden befindet. Bei Smart-Home-Systemen ist es jedoch eher selten erwünscht, dass sich unauthentifizierte Personen Zugang zum Eigenheim beschaffen können. Die Klassifizierung, ob Daten sensibel sind oder nicht, hängt also vom jeweiligen Kontext ab. Ein weiteres Beispiel hierfür ist das autonome Fahren. Nicht jeder sollte sich mit dem eigenen KFZ verbinden und Daten auslesen bzw. Aktoren bedienen können.

Auch die Sicherheitskamera, die in diesem Experiment entwickelt werden soll, besitzt aufgrund ihres Einsatzes sensible Daten. Wie üblich sollte nur authentifiziertes Personal Zugriff zu Aufnahmen der Kamera erhalten. Auch sollte es nicht einfach möglich sein, die Kamera außer Betrieb zu nehmen. Im Falle eines Ausfalls könnte der Angreifer bzw. Einbrecher auf das Gelände oder in das Gebäude eindringen, ohne gesehen zu werden. Zudem sollte es nicht möglich sein, dass die Aufnahmen der Sicherheitskamera manipuliert bzw. ausgetauscht werden können. Mögliche Angriffe sind die Manipulation von Daten, die Einschränkung der Verfügbarkeit, Blackhole-Angriffe bzw. Selective-Forwarding und das Abhören von Daten (eavesdropping).

VII. DURCHFÜHRUNG DES EXPERIMENTS

Nachdem der Testaufbau besprochen wurde, gilt es nun in die Rolle des Angreifers zu schlüpfen und mögliche Angriffe zu diskutieren und auszuführen. Außerdem soll besprochen werden, wie die Angriffe hätten verhindert werden können, sodass die Arbeit von [9] Anwendung findet. Wie in Abbildung 3 dargestellt, werden die Komponenten zunächst aufgebaut. Folgende Netzwerkteilnehmer werden konfiguriert:

- **Alice** erhält die IP-Adresse 192.168.2.115 und stellt einen MQTT-Client dar. Es werden bestimmte Topics abonniert, sodass dieses Gerät benachrichtigt wird, falls sich bestimmte Werte im System ändern.
- **Bob** erhält die IP-Adresse 192.168.2.171 und stellt den MQTT-Broker dar. Dieser empfängt Daten von Sensoren und speichert diese unter einem bestimmten Topic ab. Abonnenten des Topics werden dann von ihm benachrichtigt. Als Anwendung wird *Aedes* verwendet.
- Der **Sicherheitskamera** (Security Camera) wird die IP-Adresse 192.168.2.178 zugewiesen und dient in dem Aufbau als Sensor. Es werden periodisch Bildaufnahmen erzeugt und an Bob, dem Broker, gesendet.
- **Eve** stellt den Angreifer dar und erhält die IP-Adresse 192.168.2.168.
- Der **Router** stellt mit der IP-Adresse 192.168.2.1 das Standard-Gateway des Netzwerks dar.

A. Eavesdropping

Ein entscheidender Grund, warum HTTPS als Ergänzung zu HTTP entwickelt wurde, ist das Senden und Empfangen von sensiblen Informationen wie Passwörter und anderen benutzerspezifischen Eigenschaften. Dies ist auch bei MQTT bzw. MQTTS der Fall. Deshalb wird in diesem Abschnitt untersucht, wie sicher das Framework *pi-aREST* diesbezüglich ist. Als erster Punkt kann genannt werden, dass es nicht möglich ist, den Kommunikationsport einzustellen. Dieser ist fest in das Framework eingearbeitet und lautet 1883, welcher im Allgemeinen als unsicher gilt. Dieser Port wird meist für das Protokoll MQTT und nicht für das sichere Äquivalent MQTTS verwendet. Da die Ports jedoch vom Serveradministrator individuell verwendet werden können, muss nun untersucht werden, ob die Kommunikation tatsächlich unverschlüsselt verläuft. Hierfür wird auf *Eve* das Netzwerkanalyse-Tool *Wireshark* installiert und ausgeführt. Zudem werden die Dienste von der Sicherheitskamera und Bob gestartet, sodass diese miteinander kommunizieren und Daten kontinuierlich ausgetauscht werden. Werden Pakete von einem Netzwerkteilnehmer zu einem anderen gesendet, sind diese an jedem Knotenpunkt im Netzwerk abgreifbar. Jeder Knotenpunkt entscheidet dann anhand der Paket-Header, ob das jeweilige Paket für ihn bestimmt ist. Dadurch, dass in der Tat keine Verschlüsselung der Daten erfolgt, können mithilfe von *Wireshark*

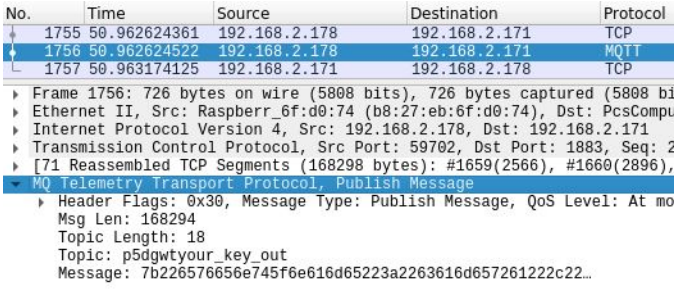

Abbildung 4. Ausgespähte Pakete des unsicheren MQTT-Protokolls mit Wireshark

Bildaufnahmen ausgespäht werden, wie Abbildung 4 zeigt. Gerade für Kameras, dessen Aufnahmen im Grunde nicht für jedermann zugänglich sein sollen, ist dies eine kritische Sicherheitslücke. Diese könnte beispielsweise durch ein geeignetes Verschlüsselungsverfahren geschlossen werden, sodass nur Parteien mit den jeweiligen geheimen Schlüsseln Zugriff zu den Daten erlangen können. Die Verwendung des Protokolls MQTTS könnte zum Beispiel die Sicherheitslücke schließen. Dieses Experiment zeigt, dass durch simples Zuhören sensible Daten ausgespäht werden können, ohne in den Informationsfluss einzugreifen oder zu beeinflussen. Voraussetzung ist natürlich, dass sich der Angreifer im selben Netzwerk befindet.

B. Man-in-the-Middle-Angriff

Bei einem Blackhole-Angriff handelt es sich um einen Denial-of-Service-Angriff (DoS), bei dem die Verfügbarkeit der Daten angegriffen wird. Dabei wird der Informationsfluss zwischen zwei Knotenpunkten über eine dritte Instanz umgeleitet. Anschließend werden bestimmte Informationspakete verworfen, sodass diese dem eigentlichen Ziel nicht mehr zur Verfügung stehen [3]. Als würden die Informationen in ein schwarzes Loch fallen.

Auch dies soll in diesem Experiment exemplarisch durchgeführt werden. Hierfür werden die Dienste von Bob und der Sicherheitskamera gestartet, sodass ein Informationsfluss stattfindet. Es werden also wieder kontinuierlich Bilder von der Kamera aufgenommen und an den Broker gesendet. Das Ziel in diesem Versuch ist es, die Verbindung zu unterbrechen bzw. Pakete verschwinden zu lassen, ohne einen kompletten Knotenpunkt des Netzwerks auszuschalten, wie es bei einem DDoS-Angriff der Fall ist. Die erste Hürde ist also, sich als Angreifer zwischen die beiden Kommunikationspartner zu schalten, ohne dass diese es bemerken und anschließend keine Pakete weiterzuleiten. Ein solcher Angreifer wird traditionell als *Man in the Middle* (MitM) bezeichnet. Es stellt sich nun also die Frage, wie man den Informationsfluss über eine dritte Instanz leiten kann. Die einfachste Methode ist das ARP-Spoofing. Hierbei wird den beiden Teilnehmern vorgetäuscht, man selbst sei der richtige Gesprächspartner.

Das *Address Resolution Protocol* (ARP) ist ein häufig eingesetztes Protokoll, welches Adressen des *Internet Protocols* (IP) auf Adressen des *Link-Layers* (MAC-Adressen) abbildet. Teilnehmer des Netzwerks, welches ARP verwendet, nehmen alle ARP-Pakete an, egal wer diese sendet. Dies hat vor allem Performancegründe, bildet jedoch eine Schwachstelle für einen MitM-Angriff im lokalen Netzwerk.

Eve hat nun also die Aufgabe, sich bei Bob als die Sicherheitskamera und bei der Sicherheitskamera sich als Bob auszugeben. Hierbei wird eine Schwachstelle des ARP-Protokolls ausgenutzt. ARP implementiert nämlich keine Authentifizierung der Knotenpunkte, die das Protokoll verwenden. Als Angreifer sendet Eve also ARP-Pakete, mit denen sie Bob weiß macht, sie sei die Sicherheitskamera. Bob merkt sich dies und speichert sich den Eintrag in seine persönliche ARP-Tabelle. Anschließend gibt sich Eve als Bob bei der Sicherheitskamera aus und sendet entsprechende ARP-Pakete zu ihr. Auch diese merkt sich dies in ihrer eigenen Tabelle. Hiermit verläuft der Informationsfluss zwischen der Sicherheitskamera und Bob über Eve. Damit Pakete nun tatsächlich weitergeleitet werden, müsste dies erst konfiguriert werden. Wird dies nicht getan, werden alle Pakete einfach zu Eve gesendet und dann verworfen, also genau das, was das Ziel dieses Versuchs ist. Die Pakete werden somit verschluckt und tauchen nicht bei dem eigentlichen Zielen auf. Auf das Experiment bezogen bedeutet dies, dass die aufgenommenen Bilder von der Sicherheitskamera nicht zu dem Broker gelangen und eine Aktualisierung des entsprechenden Topics ausfällt. Die Verfügbarkeit wurde damit als grundlegende Sicherheitsanforderung verletzt.

Des Weiteren können nun mächtigere Angriffe durchgeführt werden, wie beispielsweise das Manipulieren der Bilddaten. Bei einem Einbruch könnte der Angreifer die Bilder z.B. so verändern, dass dieser nicht auf den Aufnahmen zu sehen ist. Da das Framework, mit dem die Anwendung entwickelt wurde, lediglich das MQTT- und nicht das sicherere MQTTS-Protokoll verwendet, kann die Integrität der Daten nicht gewährleistet werden. Die Verwendung von MQTTS bzw. das Überprüfen der Integrität könnte dieses Problem also lösen.

VIII. Fazit und Zukünftige Arbeit

Wir haben besprochen, welche Protokolle in der IoT verwendet werden, um Daten miteinander kontinuierlich auszutauschen und dass Sicherheitsanforderungen abhängig von ihrer Domäne sind. So sind Anforderungen an die Automobil-Industrie andere als die der Spielzeug-Domäne. Eine detaillierte Betrachtung des Kontexts einer IoT-Anwendung ist demnach essentiell für das Etablieren von IT-Sicherheit. Mit diesem Wissen haben wir uns ein Projekt bzw. Framework ausgesucht, welches bekannte Protokolle wie MQTT implementiert und einen Datenaustausch zwischen IoT-Geräten ermöglicht. Mithilfe dieses Frameworks wurde eine Sicherheitskamera entwickelt und anschließend auf IT-Sicherheit überprüft. Dabei wurden ebenfalls die verwendeten Angriffstechniken besprochen. Als Ergebnis kam vor allem heraus, dass das Framework ausschließlich die Kommunikation über unverschlüsselte Kanäle anbietet. Dies ist jedoch in sehr vielen IoT-Anwendungen eine Verletzung der IT-Sicherheit. Besonders bei unserem Beispiel mit der Sicherheitskamera konnten wir zeigen, dass aufgrund der Tatsache, dass keine

geeignete Verschlüsselung verwendet wird und Bildaufnahmen der Kamera mit einfachsten Mitteln ausgespäht werden können. Auch schützt das Framework nicht vor Fälschung der Daten, sodass die Bildaufnahmen manipuliert zum Broker gelangen können. Das Framework bietet derzeitig viele Verbesserungsmöglichkeiten, wovon einige bereits in dieser Arbeit genannt wurden. In zukünftiger Arbeit können die von uns genannten Vorschläge umgesetzt und das Framework ein weiteres Mal untersucht werden, um mögliche neue oder bereits bestehende Probleme und Schwachstellen zu finden, welche in dieser Arbeit noch nicht thematisiert wurden.

LITERATUR

[1] Easily Control Your Arduino, Raspberry Pi & ESP8266/32 Projects With a RESTful Framework. https://arest.io/.

[2] Node.js. https://nodejs.org/.

[3] I. Aad, J.-P. Hubaux, and E. W. Knightly. Denial of Service Resilience in Ad Hoc Networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, page 202–215, New York, NY, USA, 2004. Association for Computing Machinery.

[4] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, and R. Attia. PHOABE: Securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted IoT. *Computer Networks*, (133):141–156, 2018.

[5] D. Borio, C. O'Driscoll, and J. Fortuny. GNSS Jammers: Effects and countermeasures. In *2012 6th ESA Workshop on Satellite Navigation Technologies (Navitec 2012) European Workshop on GNSS Signals and Signal Processing*, pages 1–7, 2012.

[6] Haowen Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *2003 Symposium on Security and Privacy, 2003.*, pages 197–213, 2003.

[7] B. D. Patel and A. D. Patel. A Trust Based Solution for Detection of Network Layer Attacks in Sensor Networks. In *2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, pages 121–126, 2016.

[8] M. Sallat. Das Anwendungsprotokoll MQTT im Internet of Things . 2018.

[9] M. G. Samaila, J. a. B. F. Sequeiros, M. M. Freire, and P. R. M. Inácio. Security Threats and Possible Countermeasures in IoT Applications Covering Different Industry Domains. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA, 2018. Association for Computing Machinery.

[10] M. Schwartz. pi-aREST. https://github.com/marcoschwartz/pi-aREST.

[11] M. Schwartz. pi-aREST. https://www.npmjs.com/package/pi-arest.

[12] H. Wong and T. Luo. Man-in-the-Middle Attacks on MQTT-based IoT Using BERT Based Adversarial Message Generation. MO 65401, USA. Missouri University of Science and Technology, Department of Computer Science.