Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Ein Sicherheitskonzept für IoT-Plattformen

Marvin Wohlfarth

Studiengang:	Softwaretechnik
Prüfer/in:	PD. Dr. rer. nat. Holger Schwarz
Betreuer/in:	Dr. rer. nat. Pascal Hirmer
Beginn am:	13. Juni 2019
Beendet am:	13. Dezember 2019

Kurzfassung

IoT-Plattformen stellen eine Schnittstelle zwischen IoT-Geräten und Anwendungen dar, die mit IoT-Geräten interagieren. Gleichzeitig bieten IoT-Plattformen grafische Benutzeroberflächen für eine intuitive Bedienbarkeit. Daraus resultieren mehrere Angriffsflächen, die eine solche Plattform offenbart. In dieser Masterarbeit wird in einem ersten Schritt eine abstrakte Architektur für IoT-Plattformen definiert. Anhand dieser Architektur werden mögliche Angriffsflächen sowie darauf abzielende, konkrete Attacken herausgearbeitet. Ziel dieser Arbeit ist es, ein Framework zu schaffen, welcher als Leitfaden für eine sichere Konfiguration der unterschiedlichen Ebenen einer IoT-Plattform dient und dabei unterschiedliche Einschränkungen des IoT berücksichtigt, wie zum Beispiel die beschränkte Hardware von IoT-Geräten. Der Leitfaden umfasst dabei die sichere Implementierung einer Webanwendung sowie die Absicherung von APIs. Dafür werden Einstellungen für die sichere Verwaltung von Benutzerdaten und eine sichere Authentifizierung von Benutzern vorgestellt. Außerdem werden Konzepte für eine sichere Konfiguration von MQTT-Brokern dargestellt. Für die Authentifizierung sowie Autorisierung von IoT-Geräten werden die Konzepte OAuth2 und X.509-Zertifikate vorgestellt. Weiter werden sichere Implementierungen von Transportprotokollen vorgestellt, welche an die Beschränkungen von IoT-Geräten angepasst sind. Diese umfassen TLS mit TCP, CoAP mit UDP, DTLS als Erweiterung für UDP, um eine Transportverschlüsselung zu ermöglichen und AugMQTT als zusätzliches Sicherheitsprotokoll für MQTT. Zuletzt werden Empfehlungen für die Konfiguration von IoT-Geräten genannt. Eine konkrete Anwendung wird anhand der Multipurpose-Binding-and-Provisioning-Platform (MBP) für IoT-Geräte vorgestellt. Für die MBP wird das OAuth2-Framework angepasst, um eine Authentifizierung und Autorisierung von IoT-Geräten zu ermöglichen. Außerdem wird eine Komponente zur Konfiguration der Anonymität von Gerätedaten vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

Das Internet of Things (IoT) ist eine aufstrebende Technologie zur Ermöglichung Intelligenter (SMART) Umgebungen, wie z.B. SMART Homes, SMART Factorys, SMART Cities etc. Dabei wird unter IoT im Allgemeinen die Vernetzung von heterogenen, (geografisch) verteilten Geräten verstanden, die mit standardisierten Protokollen und Technologien kommunizieren, um gemeinsame Ziele zu erreichen. Diese Geräte sind typischerweise ausgestattet mit Sensoren und Aktuatoren, mittels denen physische Zustände der Umgebung erfasst werden können bzw. diese Zustände beeinflusst werden können. Beispielsweise könnte die Erfassung einer erhöhten Temperatur in einem Raum (Sensor) dazu führen, dass die Klimaanlage eingeschaltet wird (Aktuator). Diese Geräte haben darüber hinaus unterschiedliche Eigenschaften und Einsatzmöglichkeiten. Um Applikationen im IoT zu erstellen, muss die IoT-Umgebung in einem ersten Schritt aufgebaut werden. Dabei müssen einerseits die Geräte mit den Sensoren und Aktuatoren in der Umgebung platziert werden und diese müssen darüber hinaus konfiguriert und vernetzt werden. Diese Schritte manuell durchzuführen ist jedoch sehr kostenintensiv, vor allem wenn die Anzahl der Geräte steigt. Aus diesem Grund wurde an der Universität Stuttgart die Multi-Purpose Binding and Provisioning Platform (MBP) entwickelt, mit der IoT-Geräte automatisch angebunden werden können, um an Sensordaten zu gelangen oder Aktuatoren zu aktivieren.

Motivation und Ziele

Sicherheit und Privatheit sind wichtige Aspekte im Internet der Dinge, die oftmals jedoch wenig Beachtung finden. Um eine sinnvolle Nutzung von IoT-Applikationen in realen Szenarien zu ermöglichen sind derartige Maßnahmen jedoch essentiell. Häufig werden IoT-Geräte beispielsweise mit Standardpasswörtern ausgeliefert und betrieben. Für IoT-Plattformen gibt es unterschiedliche Anwendungsgebiete mit unterschiedlichen Herausforderungen. So verlangt die Benutzung einer IoT-Plattform in der Industrie ein hohes Maß an Sicherheit, während die private Nutzung einer IoT-Plattform auch den Aspekt der Privatheit von Daten in den Vordergrund stellt. Um die Sicherheit in IoT-Netzwerken zu erhöhen, ist das Ziel dieser Arbeit die Ausarbeitung eines Sicherheitskonzepts. Dieses Konzept wird bekannte Protokolle und Mechanismen des IoT untersuchen, vergleichen und beschreiben, wie sicherheitsrelevante Einstellungen und Erweiterungen vorgenommen werden können. Um ein Sicherheitskonzept zu erstellen, müssen aber zunächst die Anforderungen an dieses erfasst werden. Dazu muss ein Bedrohungsmodell erstellt und ausgewertet werden, um mögliche Angriffsflächen und damit verbundene Risiken identifizieren zu

können. Gegenmaßnahmen und vorbeugende Maßnahmen, um die im Bedrohungsmodell identifizierten Angriffsmöglichkeiten zu verhindern, werden dann im Sicherheitskonzept vorgestellt. Es sollen dabei auch mehrere Möglichkeiten für eine Gegenmaßnahme gezeigt werden, da im IoT die Wahl einer Technologie oder eines Protokolls häufig vom Kontext der IoT-Umgebung abhängig ist (vorhandene Hardwareressourcen, Netzwerk, etc.). In einem weiteren Schritt soll dann die Umsetzung von ausgewählten Sicherheitskonzepten an der MBP erfolgen.

Struktur

Die Arbeit ist wie folgt aufgebaut:

Kapitel 2 – Grundlagen & Definitionen
Grundlegende Begriffe und Definitionen für diese Arbeit werden erklärt.

Kapitel 3 – Verwandte Arbeiten
In diesem Kapitel werden verwandte Arbeiten zum Thema Sicherheit von IoT-Plattformen vorgestellt. Dafür werden Ansätze für Sicherheitsfunktionen auf unterschiedlichen Ebenen vorgestellt, unter anderem auf Protokoll- sowie Benutzerebene. Auch bestehende Sicherheitsframeworks werden vorgestellt.

Kapitel 4 – Architektur von IoT-Plattformen
Die grundlegende Architektur, auf welcher das Sicherheitskonzept aufbaut, wird herausgearbeitet.

Kapitel 5 – Bedrohungsmodell
Das Bedrohungsmodell zeigt die Angriffspunkte einer IoT-Plattform auf sowie die möglichen Arten von Attacken.

Kapitel 6 – Sicherheitskonzept
Das sechste Kapitel stellt das Hauptkonzept dieser Arbeit vor, ein Sicherheitskonzept mit Empfehlungen und Vergleichen von Technologien und Konzepten auf den unterschiedlichen Ebenen der Architektur einer IoT-Plattform.

Kapitel 7 – Anwendung des Sicherheitskonzeptes auf die MBP
Zwei Konzepte wurden als “Proof-of-Concept” in der MBP umgesetzt und implementiert. Dieses Kapitel stellt die Implementierung und das zugrundeliegende Konzept vor.

Kapitel 8 – Zusammenfassung & Ausblick
Als letztes Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und ein kurzer Ausblick auf zukünftige Entwicklungen im Hinblick auf die Sicherheit in IoT-Netzwerken gegeben.

2 Grundlagen & Definitionen

Als Grundlagen für diese Arbeit werden die Begriffe Internet of Things, Sicherheit und Privatsphäre definiert. Weiter wird der Begriff IoT-Plattform vorgestellt und eine Referenzimplementierung genannt.

2.1 Internet of Things

Die zentrale Idee des Internet of Things (IoT) ist die Verbindung heterogener physischen Geräte über das Internet. Dabei soll sowohl die Kommunikation der Geräte mit den Benutzern als auch die Kommunikation der Geräte untereinander ermöglicht werden. Wenn die unterschiedlichsten technischen Geräte mit dem Internet verbunden werden, resultieren daraus auch viele Herausforderungen auf den verschiedenen Ebenen des OSI-Modells [Zim80]. Unter anderem sind dabei die unterschiedlichen Kommunikationsprotokollen auf der Transportebene oder die verschiedenen Technologien auf der Netzwerkebene (z.B. Bluetooth, Wifi, ZigBee, etc.) zu nennen. Die Verwendung von vielen Technologien und Protokollen macht ein IoT-System damit auch automatisch an mehreren Stellen angreifbar. Sicherheit im IoT ist damit ein sehr wichtiges Thema, welches allerdings noch wenig allgemeine Standards, vor allem im Bezug auf Managementplattformen hervorgebracht hat.

2.2 Sicherheit

In der IT definiert der Begriff Sicherheit vor allem die Informationssicherheit in IT-Systemen. Das Hauptaugenmerk liegt auf der Wahrung der Verfügbarkeit, Vertraulichkeit sowie der Integrität von Daten. Diese drei Eigenschaften bilden gemeinsam das CIA-Modell (Confidentiality, Integrity & Availability), welches ein wesentliches Ziel der Informationssicherheit darstellt [MYAZ15]. Die Vertraulichkeit von Daten soll sicherstellen, dass nur autorisierte Benutzer Zugriff auf diese haben. Ein Benutzer kann in diesem Kontext sowohl ein Mensch als auch ein Gerät sein. Integrität beschreibt die Richtigkeit sowie Unveränderlichkeit von Daten während der Übertragung zwischen zwei Stationen. Die Verfügbarkeit bezieht sich auf das Vorhandensein von Daten, die zu jedem Zeitpunkt, wenn ein Benutzer diese einsehen will, verfügbar sein sollen. Auf Basis des CIA-Modells werden in dieser Arbeit Bedrohungen sowie geeignete Gegenmaßnahmen beschrieben, um die Informationssicherheit zu wahren.

2.3 Privatsphäre

Mit der zentralen Idee des IoT, heterogene Geräte über das Internet zu verbinden und eine umfassende Kommunikation zu ermöglichen, wird auch das Speichern von Daten ein wichtiger Punkt. Da viele Geräte nur über beschränkte Hardwareressourcen verfügen, ist eine sichere Übertragung und Speicherung von Daten nicht immer einfach und effizient umzusetzen. Um die Privatsphäre von Benutzern solcher Geräte zu wahren, muss der Zugang zu diesen Daten vom Benutzer konfigurierbar und einschränkbar sein. Sensible Daten, wie z.B. Bilder eine Überwachungskamera, müssen der Kontrolle des Benutzers unterliegen.

2.4 IoT-Plattformen

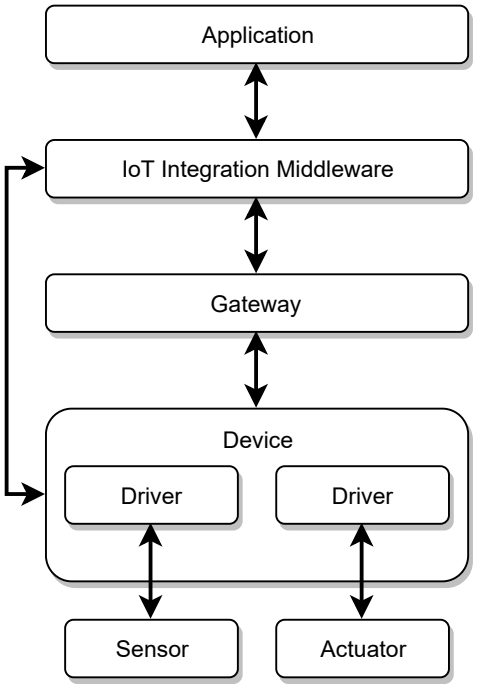


Abbildung 2.1: IoT Referenz Architektur nach [GBF+16].

Mit der Anzahl an verbundenen Geräten im IoT steigt auch die Anzahl an sogenannten IoT-Plattformen. Über IoT-Plattformen können Geräte in einem Netzwerk verwaltet und Daten ausgelesen werden. Dabei gibt es unterschiedliche Ansätze, um Geräte zu integrieren, wobei die grundsätzlichen Funktionen der Plattformen ähnlich sind [GBF+16]. Die Implementierung und verwendeten Technologien unterscheiden sich meist aber stark. Guth et al. stellen eine Referenzarchitektur für IoT-Plattformen vor. Diese wurde aus verschiedenen State-of-the-Art-Plattformen abstrahiert und soll als Basis für die Vergleichbarkeit der Plattformen dienen [GBF+16]. Die in Abbildung 2.1 dargestellte Architektur besteht aus

drei Hauptkomponenten. Die erste Komponente bilden die Geräte (Sensoren, Aktuatoren), welche mittels Treibern integriert werden können. Die Zweite besteht aus der Middleware, die aus der IoT Integration Middleware sowie einem Gateway besteht. Über das Gateway soll der Anschluss von Geräten erleichtert werden, wenn diese bestimmte Protokolle nicht unterstützen, um direkt mit der Middleware kommunizieren zu können. Die IoT Integration Middleware ist verantwortlich für den Empfang sowie die Verarbeitung von Daten und die Weiterleitung dieser an die Anwendung. Weiter werden auch Instruktionen von der Anwendung an die Aktuatoren weitergegeben. Die Anwendungskomponente selbst repräsentiert die Software, welche die von der Middleware bereitgestellten Daten nutzen und weiterverarbeiten kann. Über die Anwendung kann ein menschlicher Benutzer die angeschlossenen Geräte verwalten und Daten auslesen. Mittels dieser Software kann das Netz aus Sensoren und Aktuatoren in größere Systeme integriert werden.

2.4.1 Multi-purpose Binding and Provisioning Platform (MBP) für IoT-Geräte

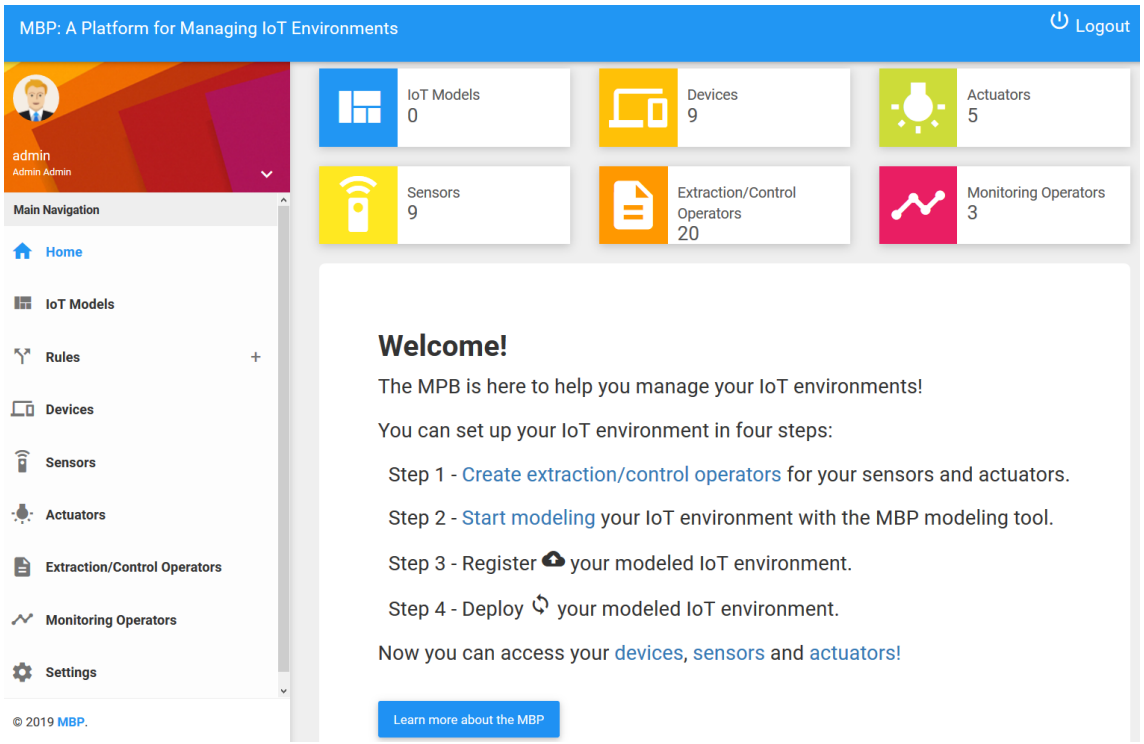


Abbildung 2.2: Benutzeroberfläche der MBP (basierend auf Bootstrap-Templates).

Die MBP für IoT-Geräte ist ein an der Universität Stuttgart entwickelter Forschungsprototyp einer Managementplattform für IoT-Geräte. Diese ermöglicht es, IoT-Geräte zentralisiert über eine Weboberfläche verwalten zu können sowie auf deren Sensoren und Aktuatoren zugreifen zu können [HBS+16]. Der Zugriff auf die Sensoren bzw. Aktuatoren geschieht

über Adapterskripte, die auf den IoT-Geräten deployed werden. Mittels dieser Skripte werden die Daten der Sensoren an einen MQTT Broker geschickt [SHPM18]. Die MBP kann diese Daten abrufen und in einer Monitoring-Ansicht anzeigen [HWBM16a]. Dafür werden sowohl Echtzeitdaten als auch historische Daten in Diagrammen visualisiert. Die Adapterskripte werden können generisch angelegt und so von mehreren Geräten verwendet werden. Das ermöglicht es, große Netzwerke bestehend aus vielen IoT-Geräten, einfach zu erstellen und zu verwalten. Außerdem bietet die MBP ein Modellierungstool für IoT-Umgebungen [HWBM16b]. Mit diesem können IoT-Geräte auf dem Grundriss eines Gebäudes platziert werden, um damit die räumliche Verteilung der Geräte anzeigen zu können. Wenn eine komplette IoT-Umgebung modelliert ist, kann diese direkt aus diesem Tool deployed werden. In Abbildung 2.2 ist die Begrüßungsseite nach dem Benutzerlogin zu sehen. Die MBP läuft auf einem Tomcat-Server und verwendet einen Mosquitto-MQTT-Broker, eine MongoDB als Datenbank für Benutzer- und Gerätedaten sowie eine InfluxDB als Datenbank für Zeitreihendaten (Sensordaten).

3 Verwandte Arbeiten

In diesem Kapitel werden bereits existierende Arbeiten und Ansätze vorgestellt, die sich mit Sicherheit und Privatsphäre im IoT auseinandersetzen.

Um ein Sicherheits- und Privatsphäreframework für IoT-Plattformen entwickeln zu können, muss eine abstrahierte Architektur von verschiedenen IoT-Plattformen als Grundlage dienen. Das IoT World Forum hat das IoT Reference Model [BBD+13] entwickelt, um die Architektur von IoT-Anwendungen modellieren zu können. Das Modell basiert auf dem Informationsfluss von einem physischen Endgerät bis hin zur tatsächlichen Nutzung der Daten durch einen Anwender. Dabei hat das Modell genau wie das OSI-Modell [Zim80] sieben Schichten.

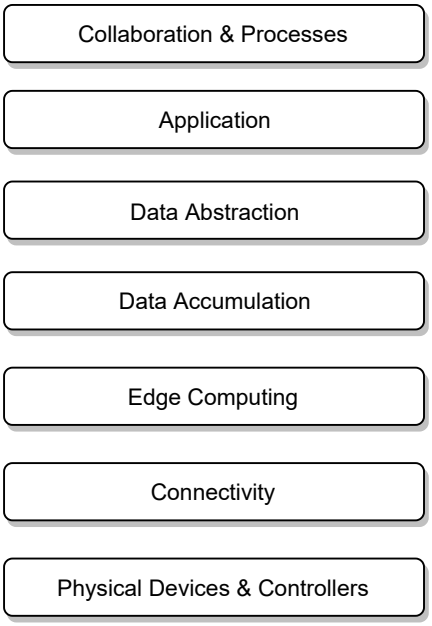


Abbildung 3.1: IoT World Forum Referenzarchitektur nach Jim Green [Gre14].

Nawir et al. stellen in [NAYL16] unterschiedliche Attacken auf ein Netzwerk vor. Dabei werden die Attacken unterschiedlich kategorisiert, beispielsweise nach Host, Kommunikationsprotokoll oder basierend auf dem Zugriffslevel, der dafür nötig ist. Insgesamt unterscheiden die Autoren zwischen den nachfolgenden Kategorien.

Spoofed, Alter, Replay Routing Information Ziel dieser Attacken ist es, an Routing-Informationen zu gelangen, um diese später zu verwenden, um beispielsweise eine Routing-Loop zu kreieren. Eine Routing-Loop entsteht aufgrund gefälschter Nachrichten eines Angreifers, die in das Netzwerk geschickt werden. Bei einer Routing-Schleife ist der Pfad zu einem bestimmten Ziel eine Schleife, die Nachricht kann also nie zugestellt werden. Durch das Versenden von vielen Nachrichten, die eine Routing-Loop auslösen, kann ein Netzwerk überlastet werden. Um an die Routing-Informationen zu kommen wird der Datenaustausch zwischen den Geräten passiv mitgelesen. Die Schwachstelle ist dabei die Erkennbarkeit von IoT-Geräten in einem Netzwerk. Dadurch kann ein Angreifer erkennen, welche Daten von einem IoT-Gerät kommen.

Sybil Attacke Bei dieser Attacke besitzt ein Gerät mehrere "Persönlichkeiten". Dabei täuscht ein kompromittiertes Gerät eine andere (mehrere) Identitäten vor, um beispielsweise falsche Nachrichten in das System einzuschleusen oder höhere Zugriffsrechte zu erlangen.

Denial of Service (DoS) Diese Attacke zielt auf das Netzwerk ab, um die Netzwerkkapazität zu reduzieren oder komplett zu deaktivieren und damit sämtliche IoT-Geräte zu trennen oder deren Kommunikation erheblich zu stören. Dabei wird eine hohe Anzahl von Anfragen an das Netzwerk gesendet. Eine Erweiterung ist die Distributed Denial of Service (DDoS)-Attacke, welche eine hohe Anzahl von Anfragen über viele Geräte schickt. Dadurch ist eine eindeutige Identifizierung des Angreifers oft nicht möglich, auch das Blockieren von IP-Adressen als Gegenmaßnahme wird aufgrund der Vielzahl von Geräten schwieriger.

Attacken auf Gerätelevel Für Angriffe auf Geräte in einem Netzwerk wird zwischen zwei Arten von Geräten unterschieden. Die sogenannten "Low-end device class"-Geräte verfügen als solche über eingeschränkte Hardwareressourcen und haben keine höheren Zugriffsrechte auf das restliche System. In diese Kategorie fallen unter anderem Sensoren. Die zweite Art wird "High-end device class"-Geräte genannt. Diese können, wenn sie von einem Angreifer kompromittiert wurden, viel Schaden in einem Netzwerk anrichten, da sie sowohl über mehr Hardware verfügen und auch höhere Zugriffsrechte besitzen. Ein solches Gerät ist beispielsweise ein Laptop.

Angriffe auf Zugriffslevel Angreifer können auf zwei Arten Zugriff auf ein IoT-Ökosystem und dessen Geräte bekommen: aktiv oder passiv. Passive Attacken beinhalten das Monitoring sowie Abhören des Netzwerkverkehrs, um Informationen über das Netzwerk sowie Zugangsdaten zu erlangen. Aktive Attacken hingegen versuchen die Sicherheitsmechanismen eines Systems zu brechen, um an Daten zu gelangen oder Geräte zu kompromittieren.

Standortbasierte Attacken Diese Art von Angriff berücksichtigt den Ort, von dem die Attacke auf ein IoT-System ausgeht. Unterschieden wird dabei zwischen internen und externen Attacken. Interne Attacken werden von einer Komponente innerhalb des Systems gestartet ("Insider"). Dabei hat der Angreifer bereits Zugriff auf ein Netzwerk, beispielsweise der Mitarbeiter einer Firma. Externe Attacken dagegen werden per Remoteverbindung von außerhalb auf das zu attackierende System durchgeführt, dabei hat der Angreifer kein Wissen über die Architektur etc.

Angriffsstrategien Es wird zwischen physischen und logischen Angriffen unterschieden. Physische Angriffe beschreiben die Kompromittierung eines Gerätes vor Ort, in dem man z.B. die Stromversorgung unterbricht, um ein Gateway auszuschalten. Auf der anderen Seite zielen logische Angriffe auf die Kommunikationskanäle ab. Die physischen Geräte werden dabei nicht beachtet.

Angriffe auf Informationslevel Diese Kategorie von Angriffen konzentriert sich auf den Schaden, den ein Angreifer auf Informationslevel anrichtet. Als "Interruption" wird die Nichtverfügbarkeit von Informationen beschrieben. Dabei ist das einzige Ziel des Angreifers, dass keine Daten gesendet werden. Beim sogenannten "Eavesdropping" hört der Angreifer einen Kommunikationskanal ab und hindert bei einer Übertragung von Daten den Empfänger daran, das Datenpaket zu empfangen. Als "Alteration" wird das Verändern der Informationen durch einen Angreifer auf dem IoT-Gerät genannt, bevor diese letztendlich weitergeschickt werden. Als "Fabrication" wird die Attacke genannt, wenn ein Angreifer eine Nachricht imitiert. Dafür wird diese nachgebaut und anschließend versucht, diese in das IoT-Netzwerk einzuschleusen. Beim "Message Replay" fängt der Angreifer eine Nachricht ab, verändert sie, und schickt die kompromittierte Nachricht an den ursprünglichen Empfänger weiter. Als letzter Angriff in dieser Kategorie wird "Man-in-the-Middle" genannt. Dabei läuft die Kommunikation zweier Geräte immer über den Angreifer, ohne das dieser erkannt wird. Der Angreifer kann Daten verändern und weiterschicken.

Host-basierte Angriffe IoT-Geräte können auf verschiedene Arten angegriffen werden. In dieser Kategorie wird zwischen den verschiedenen Hosts unterschieden. Auf Benutzerebene heißt das, dass ein Benutzer einer IoT-Plattform geheime Daten, wie z.B. Passwörter preisgibt. Auf Softwareebene sind das Sicherheitslücken, die ein Angreifer erkennt und ausnutzt, um an Daten zu gelangen. Die dritte Ebene ist die Hardwareebene. Dabei hat ein Angreifer Zugriff auf die Hardware eines IoT-Gerätes und kann dieses kompromittieren.

Protokoll-basierte Angriffe Protokoll-basierte Attacken werden in zwei Möglichkeiten aufgeteilt. Bei der ersten Möglichkeit nutzen Angreifer die Abweichung von einem Protokoll, welches in einem IoT-Netzwerk für die Kommunikation verwendet wird. Eine Abweichung definiert dabei beispielsweise das Aufweichen von Sicherheitsmechanismen, um das

Protokoll auch auf beschränkten Geräten benutzen zu können. Diese Abweichung wird von den Betreibern des IoT-Netzwerks vorgenommen. Die zweite Möglichkeit eine protokollbasierten Attacke ist die Störung eines verwendeten Protokolls. Wenn ein potenzieller Angreifer das verwendete Protokoll kennt, kann er dieses stören (z.B. durch gefälschte Kontrollnachrichten) und dadurch die Verfügbarkeit von IoT-Geräten beeinflussen.

Mahmoud et al. geben in ihrem Paper einen Überblick über den aktuellen Stand der Sicherheitsfeatures im IoT sowie die damit verbundenen Herausforderungen [MYAZ15]. Dabei verwenden die Autoren ein 3-Schichten-Modell, welches aus der Sensorebene, der Netzwerkebene sowie der Anwendungsebene besteht. Die Sensorebene umfasst die Sensoren, welche Daten sammeln und zur Verfügung stellen. Die Netzwerkebene beinhaltet sämtliche Funktionalitäten zur Übertragung von Daten in einem IoT-Netzwerk sowie der Kommunikation von Geräten. Beispielhaft werden hier Wifi, Bluetooth oder LTE genannt. Auf der Anwendungsebene können die gesammelten und übertragenen Daten in Anwendungen genutzt werden, beispielsweise in einem Smart Home.

Sicherheitsfeatures Weiter wird zwischen zwei verschiedenen Arten der Herausforderungen an die Sicherheit unterschieden. Zum einen die technologisch-bedingten Herausforderungen, da im IoT ein breites Feld an unterschiedlichen Geräten miteinander kommuniziert und damit Sicherheit in unterschiedlichen Technologien erforderlich ist. Auf der anderen Seite gibt es die Anforderung, gängige Sicherheitsprinzipien korrekt umzusetzen, um ein sicheres Netzwerk zu schaffen. Folgende Sicherheitsprinzipien werden dabei benannt:

- Vertraulichkeit
- Integrität
- Verfügbarkeit
- Authentifizierung
- Leichtgewichtige Lösungen
- Heterogenität
- Policys
- Schlüsselverwaltungssystem

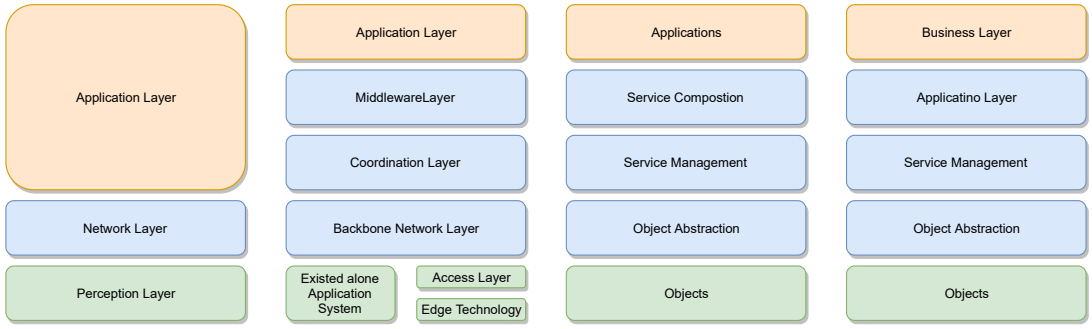
Herausforderungen Auf jeder der vorgestellten Ebenen im IoT sind Angriffe auf die Sicherheit möglich. In den Herausforderungen wird zwischen aktiven und passiven Attacken unterschieden. Aktive Attacken blockieren das Netzwerk, während passive Attacken das Netzwerk abhören und Daten auslesen. Außerdem können Angriffe von externen Quellen kommen oder bereits innerhalb eines Netzwerks ausgeführt werden. In diesem Abschnitt werden die Anforderungen an die Sicherheit auf jeder Ebene vorgestellt.

Die Probleme auf der Perception-Layer oder auch Sensorebene sind unter anderem, dass die Signalstärke in drahtlosen Netzwerken gestört werden kann. Dadurch können Sensoren keine Daten mehr liefern und sind nicht mehr erreichbar. Außerdem befinden sich IoT-Geräte häufig in externen Umgebungen und haben zusätzlich eine große räumliche Distanz. Dadurch sind sie anfällig für physische Angriffe. Ein weiteres Problem ist, dass Sensoren sehr limitierte Geräte und damit anfällig für verschiedenste Attacken sind. Limitiert bedeutet in diesem Kontext, das sie nur über sehr begrenzte Hardwareresourcen verfügen. Diese Attacken sind unter anderem “Replay Attack”, “Timing Attack”, “Node Capture Attack” oder “Malicious Data Attack”.

Auf der Netzwerkebene sind vor allem drei Angriffsmöglichkeiten zu nennen. Zunächst sind das DoS-Attacken. Als eine weitere, passive Angriffsmöglichkeit gilt die Analyse sowie Monitoring des Netzwerkverkehrs. Sehr anfällig ist das IoT auf Netzwerkebene für eine Man-in-the-Middle-Attacke. Im IoT ist Machine-to-Machine-Kommunikation eine zentrale Funktion. Diese bringt aber das Problem der Kompatibilität mit sich [MYAZ15]. Das bedeutet, dass es fast unmöglich ist, herkömmliche Netzwerkprotokolle mit ihren Schutzmechanismen unverändert zu benutzen. Dies eröffnet Angreifern eine weitere Angriffsfläche, wenn diese Protokolle angepasst werden müssen.

Es existieren noch keine globalen Richtlinien für die Interaktion und die Entwicklung von IoT-Anwendungen. Verschiedene Anwendungen verwenden verschiedene Authentifizierungsmechanismen [MYAZ15], um einen Benutzer zu authentifizieren. Auf der Anwendungsebene muss außerdem beachtet werden, dass unterschiedliche Benutzer auch unterschiedlich mit den Daten umgehen. Anwendungen, die Daten von Sensoren bereitstellen, müssen Werkzeuge zur Verfügung stellen, mit deren Hilfe der Benutzer die Sichtbarkeit seiner Daten festlegen kann.

Al-Fuqaha et al. stellen wichtige Technologien und Protokolle für das IoT vor und beschreiben einige Anwendungsgebiete. Dabei ist interessant, dass mehrere Architekturmodelle [AGM+15] für das IoT vorgestellt werden. Diese haben in einem 3-Schichten Modell mit einer Anwendungsschicht, einer Netzwerkschicht und einer Wahrnehmungsschicht ihre Grundlage. Neue Modelle wurden entwickelt, um eine Verfeinerung der einzelnen Ebenen zu ermöglichen. So werden die physischen Geräte auf der Wahrnehmungsebene auch als Objekte klassifiziert. Die Daten dieser Objekte werden über die nächste Ebene, die Object-Abstraction-Ebene, in die Service-Management-Ebene transferiert. In Abbildung 3.2b werden diese Aufgaben in den Komponenten in der Koordinationsebene ausgeführt. Die Daten der IoT-Geräte werden über eine Backbone-Netzwerk-Ebene an die Middleware angeschlossen. Die Service-Management-Ebene in den Abbildungen 3.2c und 3.2d stellt die Middleware da, welche in Abbildung 3.2b äquivalent als Middleware dargestellt wird. Die Anwendungsebene bietet Services und Schnittstellen für Benutzer und andere Systeme an, welche Zugriff auf die IoT-Geräte und deren Daten benötigten. Die Businessebene in Abbildung 3.2d ergänzt die Architektur um eine Ebene, die für das Design von Businessmodellen, Graphen, Prozessabläufen etc. verantwortlich ist.



(a) Drei Ebenen. **(b)** Middleware basiert. **(c)** SOA basiert. **(d)** Fünf Ebenen.

Abbildung 3.2: Überblick über verschiedene Ansätze für IoT-Referenzarchitekturen.

Lösungen für kontrollierte Zugriffe auf Daten von IoT-Geräten, die im Bereich der Web- und Cloud-Anwendungen gängig sind, können nicht direkt im IoT-Umfeld eingesetzt werden [SPC+17]. Sciancalepore et al. stellen mit OAuth-IoT ein Framework vor, welches den OAuth2-Authorization-Flow, verschiedene Token-Formate sowie den Protokoll-Stack der Internet Engineering Task Force (IETF) verwendet, um eine Zugriffskontrolle auf die Daten von Geräten im IoT zu ermöglichen. Das Framework besteht aus vier Hauptkomponenten, welche auch grafisch in Abbildung 3.3 dargestellt ist. Die erste Komponente ist das IoT-Netzwerk, bestehend aus unterschiedlichen IoT-Geräten, die an eine Sink-Node (Basisstation) angeschlossen sind. Dieser Sink-Node ist mit einem Gateway verbunden, welches als Resource-Server fungiert. Einzelne IoT-Geräte (Sensoren, Aktuatoren) werden als Leaf-Nodes bezeichnet. Das Gateway ist die Hauptkomponente in diesem Framework und ist als OAuth2-Resource-Server realisiert. Weiter stellt das Gateway eine Schnittstelle zwischen OAuth2 und dem IETF Protocol Stack zur Verfügung und eine Liste der verfügbaren Ressourcen. Auch ein Caching von Daten der IoT-Geräte ist möglich. Neben dem Authorization Server, der die Authentifizierung mit Tokens umsetzt, ist der Client eine weitere Komponente, welche außerhalb des Netzwerkes agiert (z.B. eine Anwendung) und auf Ressourcen zugreifen möchte.

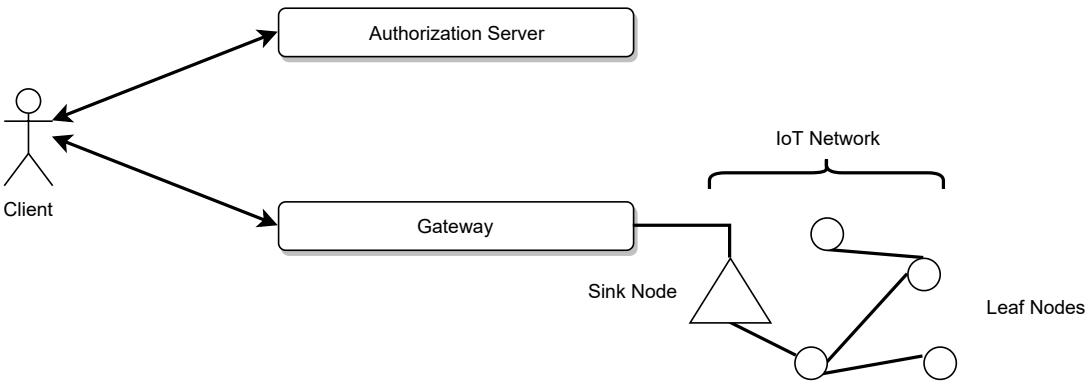


Abbildung 3.3: Referenzarchitektur für das OAuth-IoT-Framework.

Um Zugriff auf Sensorwerte aus dem IoT-Netzwerk zu bekommen, muss der Client zunächst die Authentifizierung am Authorization-Server starten. Ist diese erfolgreich, muss der Client das erhaltene Token mit dem Gateway austauschen. Das Gateway überprüft das Token und führt dann die Anfrage an die IoT-Geräte durch. Dabei hat der Client niemals direkten Zugriff auf das IoT-Netzwerk, sondern kommuniziert nur mit dem Gateway. Dieses verarbeitet die Anfragen und sendet entsprechende Antworten. Das Gateway besitzt zudem eine Routingtabelle, in der alle IDs der angeschlossenen und aktiven IoT-Geräte beinhaltet. In einer zusätzlichen Datentabelle (welche als Cache fungiert) werden alle Informationen zu einem IoT-Gerät inklusive der eigentlichen Daten dieses Gerätes gespeichert. Die Authentifizierung und Autorisierung gilt in diesem Framework lediglich für Clients, die außerhalb eines IoT-Netzwerks aktiv sind, nicht aber für die IoT-Geräte selbst.

Das IoT-OAS-Framework von Cirani et al. stellt eine Architektur vor, die in RESTful-Umgebungen mit HTTP/CoAP-Services zum Einsatz kommt [CPG+15]. Dabei kommt ein externer, OAuth-basierter Authentifizierungsservice (OAS) zum Einsatz, um die Rechenlast der Authentifizierungsfunktionalität von den IoT-Geräten auszulagern. Dadurch reduziert sich der Konfigurationsaufwand auf einem IoT-Gerät erheblich, außerdem müssen IoT-Geräte so weniger Requests/Responses verarbeiten. Ein IoT-Gerät muss keine Autorisierungslogik implementieren, sondern lediglich einen Autorisierungsservice über HTTP/CoAP aufrufen. Der Ablauf zur Autorisierung unterscheidet sich dabei in einigen wesentlichen Punkten. Ein "Service Provider" (SP) bietet Ressourcen eines Benutzers oder Gerätes an, ein "Service Consumer" (SC) möchte Zugriff auf die Ressourcen eines SP erhalten. Im standardmäßigen OAuth-Workflow verifiziert der SP bei einer Anfrage des SC dessen Identität und sendet ein "Request Token" (RT) zurück. Mit diesem RT kann der SC dann ein "Access Token" beim Autorisierungsserver des SP erhalten. Das OAS-Framework verändert den Workflow dahingehend, dass der "Service Provider" keinerlei Authentifizierungslogik besitzt. Die Authentifizierung des SC wird an eine externe Komponente, den IoT-OAS, ausgelagert. Dieser verifiziert den SC und händigt ein entsprechendes Token aus. Auch im zweiten Schritt, wenn der SC beim Autorisierungsserver des SP ein "Access Token" anfordert, benachrichtigt dieser den SP über die erfolgreiche Authentifizierung im vorherigen Schritt und händigt das RT aus. Nun erhält der SP ein "Access Token" beim IoT-OAS und gibt dieses an den SC zurück. Auf der Seite des "Service Provider" bleibt lediglich die Funktion des Authentifizierungsservers zu implementieren, allerdings kann auch diese an eine externe Komponente ausgelagert werden. Allerdings verdoppelt sich die Anzahl der übertragenen Nachrichtenpakete, da die ursprünglichen Nachrichten zur Autorisierung und Authentifizierung an die externen Services weitergegeben werden müssen.

Ein weiteres Framework zur Authentifizierung und Autorisierung im IoT mit beschränkten Geräten stellen Seitz et al. mit dem ACE-OAuth-Framework vor [SSW+19]. Mit "Authentication and Authorization for Constrained Environments" (ACE) unter der Verwendung des OAuth2-Frameworks soll ein Standard für Authentifizierung und Autorisierung im IoT geschaffen werden, wenn das CoAP-Protokoll als Alternative zu HTTP verwendet wird. Das Framework verwendet bekannte Standards und erweiterte diese um Funktionalitäten, wenn diese keine Lösungen für die Herausforderungen im IoT bieten. Zu diesen Erweiterungen

gehören eine Bereitstellung von Zugangsdaten, ein “Proof-of-Possession”-Verfahren für Zugangstokens und sogeannte “ACE Profiles”. Eine Erweiterung des Supports anderer Protokolle ist für die Zukunft geplant. Als die vier Hauptbestandteile des Frameworks werden OAuth2, das CoAP-Protokoll, “Concise Binary Object Representation (CBOR) [BH13]” als kompakte Kodierung und das CBOR-basierte, sichere Nachrichtenformat “CBOR Object Signing and Encryption (COSE)” [Sch17] genannt. Das OAuth2-Framework wird zusammen mit “Proof-of-Possesion (PoP)”-Tokens verwendet. Das PoP-Sicherheitskonzept nimmt den Autorisierungsserver als zuverlässige Dritten an und bindet Zugangstoken an kryptografische Schlüssel. Diese Schlüssel müssen dann von den Clients verwendet werden, um gegenüber dem Ressourcenserver zu beweisen, das sie der rechtmäßige Besitzer des Zugangstokens sind. Der kryptografische Schlüssel kann entweder symmetrisch oder asymmetrisch sein, je nach Anwendungsfall. Um die Sicherheit während der Übertragung mit CoAP zu sichern, wird COSE auf der Anwendungsebene eingesetzt, welches CBOR als Datenformat für die Nachrichten verwendet. CBOR ist für kleine Nachrichtengrößen entwickelt worden und erweitert JSON [Bra14] um die Funktion, Binärdaten direkt kodieren zu können, ohne sie zuvor in Base64-kodierte Strings umwandeln zu müssen. COSE ermöglicht das Signieren und Verschlüsseln von CBOR-Nachrichten, um diese sicher per CoAP übertragen zu können. Die genannten ACE-Profile werden vom Autorisierungsserver verwaltet, dieser achtet auch darauf, dass die Profile von Clients und Ressourcenserver zueinander kompatibel sind. In den ACE-Profilen wird das Kommunikations- und Sicherheitsprotokoll festgelegt, welches zur sicheren Übertragung von Nachrichten verwendet wird. Wird zum Beispiel HTTP als Protokoll festgelegt, ist JSON als Datenstruktur der Nachricht zu verwenden. Wird allerdings CoAP spezifiziert, muss CBOR anstelle von JSON verwendet werden.

Das Web-of-Things (WoT) [Wor19] ist ein Ansatz des World-Wide-Web-Consortiums (W3C), um die Kompatibilität von IoT-Plattformen sowie deren Benutzung durch Anwendungen auszubauen. Dabei ist das primäre Ziel die Beschreibung und Ergänzung von existierenden IoT-Standards und IoT-Lösungen. Durch den Einfluss der Web-of-Things-Interest-Group, welche eine Gruppe von Stakeholdern des WoT ist, wurden die folgenden Bausteine des WoT festgelegt. Diese sind die WoT-Things-Description, die WoT-Architektur, die WoT-Binding-Templates, die WoT-Scripting-API sowie die WoT-Security-and-Privacy-Guidelines. Die WoT-Things-Description bietet ein Datenformat für die Metadaten und die Netzwerkschnittstellen von sogenannten “Things”. Die WoT-Architektur definiert eine abstrahierte Architektur für das WoT, welche auf einem Set von Anforderungen basiert. Dieses Set wurde aus verschiedenen Anwendungsfällen in unterschiedlichen Einsatzgebieten generiert. Die WoT-Architektur soll ein abstraktes Konzept definieren, welches auf unterschiedliche Anwendungsszenarien angepasst werden kann. Die WoT-Binding-Templates bieten einen informativen Leitfaden, wie die Netzwerkschnittstellen der “Things” für spezifische Protokoll und IoT-Ökosysteme definiert werden sollten. Außerdem bietet die WoT-Scripting-API als ein weiterer Baustein die Möglichkeit, Anwendungslogik mittels einer Javascript-API auf einem “Thing” implementieren zu können. Als letzter Baustein

definieren die WoT-Security-and-Privacy-Guidelines Richtlinien für die sichere Implementierung sowie Konfiguration von “Things”, aber es werden auch sicherheitsrelevante Themen vorgestellt, die es beim Aufbau eines IoT-Systems zu berücksichtigen gilt.

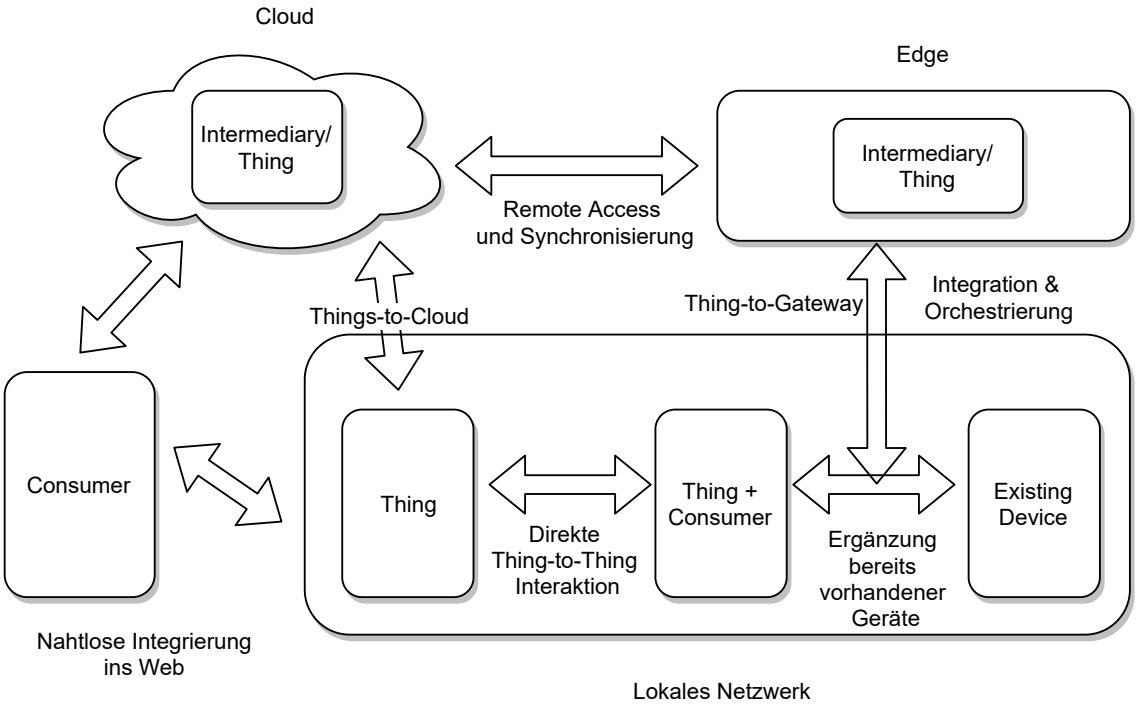


Abbildung 3.4: Referenzarchitektur des Web of Things (WoT).

Die grundlegende Architektur des WoT wird in Abbildung 3.4 dargestellt. Ein “Thing” ist die Abstraktion eines physischen oder virtuellen Objekts, welches von standardisierten Metadaten beschrieben wird [KML+19]. Diese Beschreibung mit Metadaten wird WoT-Thing-Description (TD) genannt und ist in Abbildung 3.5 veranschaulicht. Um mit dem “Thing” zu interagieren, muss ein Konsument in der Lage sein, die TD analysieren und verarbeiten zu können. Die Metadaten ermöglichen es dem Konsumenten, Protokolle und Datenstrukturen an das “Thing” anzupassen, um dessen Ressourcen nutzen zu können. Mittels der TD können auch herkömmliche, nicht vernetzte Geräte integriert werden. Wenn lokale Netzwerke bestehend aus “Things” nicht an das Internet angeschlossen werden können, werden sogenannte “Intermediaries” eingesetzt. Diese besitzen ebenfalls eine TD und agieren als eine Art Proxy zwischen Konsumenten und den eigentlichen “Things”. Mit diesem Konzept der Schnittstellendefinition ist das WoT auf allen relevanten Ebenen einer IoT-Anwendung einsetzbar. Zu diesen Ebenen gehört die Geräteebene, die Edgeebene und die Cloudebene.

Die Sicherheits- und Privatsphärerichtlinien [RM19] geben ein Bedrohungsmodell für das WoT an und beschreiben, wie vorhandene Sicherheitsmodelle sowie Mechanismen unterstützt und umgesetzt werden können. Das Bedrohungsmodell definiert zunächst die WoT-Stakeholder und anschließend die zu schützenden Güter des WoT. Bei diesen

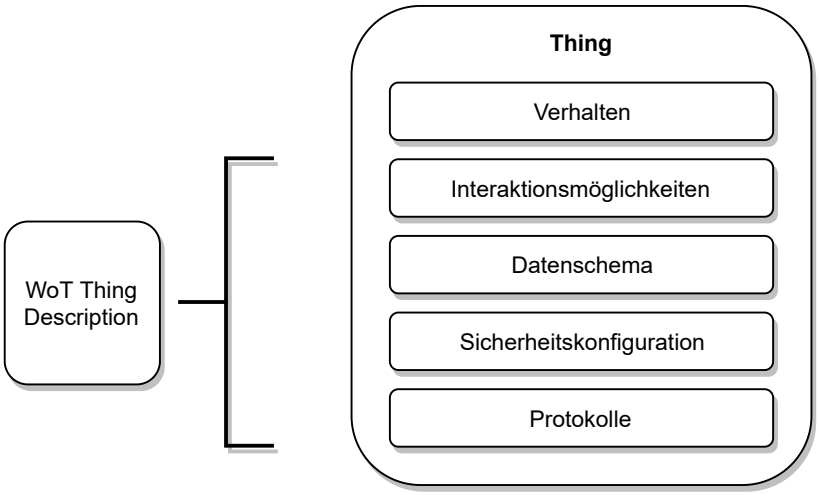


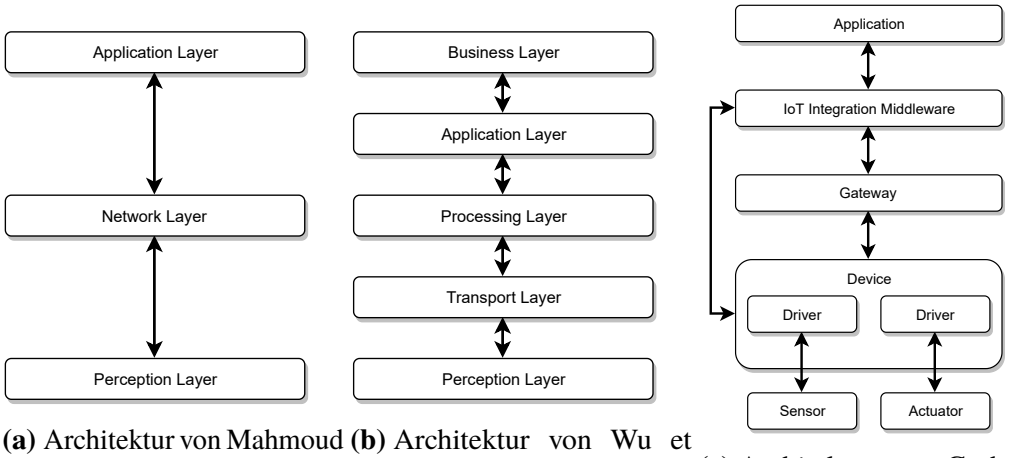
Abbildung 3.5: Metadaten eines “Things”, die in einer TD festgelegt sind.

Gütern handelt es sich um die “Things Descriptions”, Benutzerdaten, Systemdaten, WoT-Thing-Ressourcen, Geräte mit Aktuatoren, Verhaltensmetriken und Sicherheitsmetadaten (z.B. Zertifikate, geheime Schlüssel). Um die Angriffsflächen korrekt identifizieren zu können, muss zunächst ein Vertrauensmodell bestimmt werden. Dies geschieht mittels Grenzen, die verschiedenene Ausführungskontexte voneinander abgrenzen. So trennt die Script-Execution-Boundary die Ausführungskontexte von einzelnen Skripten auf mehreren “Things”, die Runtime-Instance-Execution-Boundary trennt dann den Kontext zwischen verschiedenen Instanzen der WoT-Runtime. In einer Instanz können aber mehrere Skripte ausgeführt werden. Auf Basis der Grenzen sind die Angriffsflächen festgelegt. Die eigentlichen Bedrohungen werden unterteilt in Bedrohung, den ausführenden Gegner, die bedrohten Güter sowie die Angriffsmethoden und mögliche Vorbedingungen. Nicht berücksichtigt werden in diesem Bedrohungsmodell Angriffe auf Nicht-WoT-Endpunkte und native Attacken auf Geräte (lokale Kompromittierung).

Der AWS-IoT-Device-Defender [Ama19] ist ein weiterer Ansatz für ein Sicherheitskonzept im IoT und wird von Amazon entwickelt. Dieser AWS-Service basiert auf dem AWS-IoT-Core und übernimmt das Sicherheitsmanagement für IoT-Geräte. Dafür überprüft dieser Service die vordefinierten Konfigurationen aller verwalteten IoT-Geräte kontinuierlich. Eine sichere Konfiguration eines IoT-Gerätes bedeutet etwa die korrekte Verschlüsselung von Gerätedaten und eine sichere Authentifizierung sowie Autorisierung des Gerätes. Der IoT-Device-Defender sendet eine Alarmmeldung, wenn es eine fehlerhafte Konfiguration gibt. Weiter kann der IoT-Device-Defender auch vordefinierte Sicherheitsmetriken der Geräte überwachen. Diese Metriken definieren das erwartete Verhalten eines Gerätes, bei Abweichungen wird wiederum eine Warnung gesendet. Über diesen Service können auch Sicherheits-Updates auf IoT-Geräte ausgerollt werden, um mögliche Sicherheitslücken zu schließen.

4 Architektur von IoT-Plattformen

Plattformen im IoT werden auf unterschiedlichste Arten und auf verschiedenen Ebenen attackiert. Die Plattformen bestehen aus verschiedenen Komponenten mit unterschiedlichen Funktionen, welche in Ebenen eingeordnet werden. In diesem Kapitel wird die Architektur vorgestellt, welche die unterschiedlichen Ebenen einer IoT-Plattform definiert, die jeweils eigene Sicherheitskonzepte benötigen. Die in Abbildung 4.2 dargestellte Architektur entstand



(a) Architektur von Mahmoud et al. [MYAZ15]

(b) Architektur von Wu et al. [WLL+10]

(c) Architektur von Guth et al. [GBF+16]

Abbildung 4.1: Vergleich von unterschiedlichen IoT Architekturen.

auf Grundlage der vorgestellten Modelle aus [AGM+15], [MYAZ15] sowie [GBF+16]. Diese Modelle sind in Abbildung 4.1 abgebildet. Die unterste Ebene bildet die Geräteebene. Diese umfasst sämtliche elektronischen Geräte, die im IoT mit dem Internet verbunden werden. Mahmoud et al. stellen in [MYAZ15] ein 3-Schichten-Modell als elementare IoT-Architektur vor. Dieses besteht aus einer Anwendungsschicht, einer Netzwerkschicht sowie einer Geräteebene. Bei diesem Modell übernimmt die Anwendungsschicht die Aufgabe zur Wahrung der Vertraulichkeit, Integrität und Verfügbarkeit der von der Geräteebene gelieferten Daten. Allerdings fehlt hier die Middlewareebene, die im IoT mittlerweile eine wichtige Funktion erfüllt und beispielsweise Machine-to-Machine-Kommunikation ermöglicht. Eine Middleware ermöglicht die Vernetzung einer Vielzahl von Geräten und liefert mehr als eine reine Netzwerkebene mit Funktechnologien. Al-Fuqaha et al. stellen in einer Übersicht verschiedene Ansätze zur Modellierung von IoT-Architekturen dar [AGM+15]. Eines dieser Modelle enthält als zusätzliche Schicht zum vorangegangen Modell eine Middleware. Weiter wird auch ein 5-Schichtenmodell vorgestellt, welches von Wu et al. erstellt

wurde [WLL+10]. Als fünfte Ebene ist in dieser Architektur eine Businessebene über der Anwendungsebene zu finden. Diese abstrahiert die Verwaltung der kompletten Plattform bis zu den Geräten sowie die Anwendung von Businessmodellen. Als letzte Grundlage dient die IoT-Referenzarchitektur nach Guth et al. [GBF+16]. Diese Referenzarchitektur ist das Ergebnis eines Vergleichs von mehreren IoT-Plattformen. Bei diesem Vergleich wurden die Architekturen dieser IoT-Plattformen extrahiert und abstrahiert zusammengefasst. In Kapitel 2 ist diese Architektur unter 2.4 genauer erklärt. Ausgehend von diesen Architekturen wurde als unterste Schicht die Geräteebene definiert, da die Geräte ein Hauptbestandteil des IoT und somit in jeder Architektur vorhanden sind. Die nächste Schicht ist die Transportebene, welche die Funktionalitäten, die zur Kommunikation der Geräte mit der IoT-Plattform bzw. der Middleware benötigt werden, bereitstellt. Die dritte Ebene bildet die Middleware. In dieser Schicht werden alle Komponenten einer IoT-Plattform zusammengefasst, die nicht direkt die Funktionalität der Anwendungsebene bereitstellen. Dazu zählen unter anderem MQTT-Broker und auch Datenbanken, die zum Speichern der Daten aus der Geräteebene dienen. Als oberste Schicht ist die Anwendungsebene gegeben. Auf dieser Ebene findet sich die IoT-Cloudplattform mit einer Benutzeroberfläche wieder. Aber auch andere Anwendungen, die zur Verwaltung der IoT-Plattform benötigt werden, befinden sich auf dieser Ebene. Die Kommunikation findet dabei immer nur zwischen der Middleware und der Anwendungsebene, bzw. der Geräteebene statt.

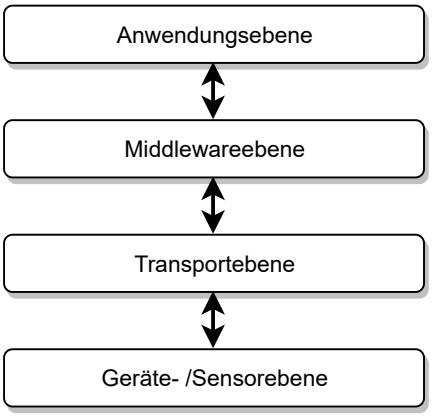


Abbildung 4.2: Architektur der Ebenen einer Cloudplattform im IoT

5 Bedrohungsmodell

Die im Kapitel 2 vorgestellten IoT-Plattformen bieten eine Fläche für verschiedenste Angriffsmöglichkeiten. In diesem Abschnitt werden die Angriffsflächen von IoT-Plattformen vorgestellt. Zu jedem Angriff wird außerdem ein reales Beispiel genannt und beschrieben. Bei der Auswahl der Bedrohungen werden nur Attacken auf User- und Softwareebene berücksichtigt, d.h. entweder die fehlerhafte bzw. nachlässige Konfiguration von Komponenten oder tatsächliche Sicherheitslücken in der verwendeten Software. Nicht beachtet werden Angriffe auf gängige Kommunikationsprotokolle oder Verschlüsselungsverfahren. Die Bedrohungen werden aus Kapitel 3, dem OWASP-IoT-Projekt sowie dem STRIDE-Bedrohungsmodell [Mic09] zusammengetragen.

Um ein Bedrohungsmodell für IoT-Plattformen zu erstellen, müssen verschiedene Sicherheitsrisiken identifiziert werden. Zunächst muss analysiert werden, welche Güter Gegenstand von möglichen Attacken sind. Darunter fallen beispielsweise Benutzerdaten, Kreditkartennummern, private Schlüssel, Zugangsdaten, etc. Für jedes dieser Güter muss aufgelistet werden, wie sie gespeichert werden und wer Zugriff darauf hat. Im nächsten Schritt werden die Bedrohungen aufgezählt. Dabei sollte man einfallsreich vorgehen und viele Möglichkeiten in Betracht ziehen. Bedrohungen sind zum Beispiel Hacker, geschäftliche Konkurrenten, ehemalige Mitarbeiter oder auch Geheimdienste. Im dritten Schritt werden dann die potenziellen Schwachstellen im eigenen System identifiziert. Als mögliche Schwachstellen zählen zum Beispiel menschliche Fehler, unsichere Kommunikationskanäle und schlecht gesicherte Netzwerke. Um diese Schwachstellen festzustellen, ist es hilfreich, ein Datenflussdiagramm zu erstellen. In Abbildung 5.1 ist das Datenflussdiagramm für die verschiedenen Ebenen einer IoT-Plattform dargestellt. Dieses Diagramm basiert auf der vorgestellten Architektur für das Sicherheitsframework in Kapitel 4 und wurde auf Grundlage der Richtlinien zum Modellieren von Bedrohungen nach Swiderski et al. [SS04] erstellt. In dieser Darstellung repräsentiert ein Rechteck ein externes Objekt, welches einen Dateninput liefert, einen Datenoutput erwartet oder beides. Kreise repräsentieren Komponenten des zu untersuchenden Systems sowie dessen Datenfluss. Ein Pfeil mit einer Richtung beschreibt den Datenfluss. Die gestrichelten, grünen Linien unterteilen das Diagramm in einen vertrauenswürdigen und einen nicht vertrauenswürdigen Teil. In diesem Fall ist die IoT-Plattform als vertrauenswürdig einzustufen, dagegen sind die externen Objekte Benutzer und IoT-Geräte rot eingefärbt und gelten damit als Bedrohung. Für jedes Objekt in diesem Diagramm, ausgenommen die externen Komponenten, werden die Bedrohungen ermittelt. Dabei sollte man jedes Objekt analysieren, oder jedes Objekt gegen Bedrohungen aus dem STRIDE-Modell testen. Als Ausgangspunkt wird dafür immer ein externes Objekt gewählt.

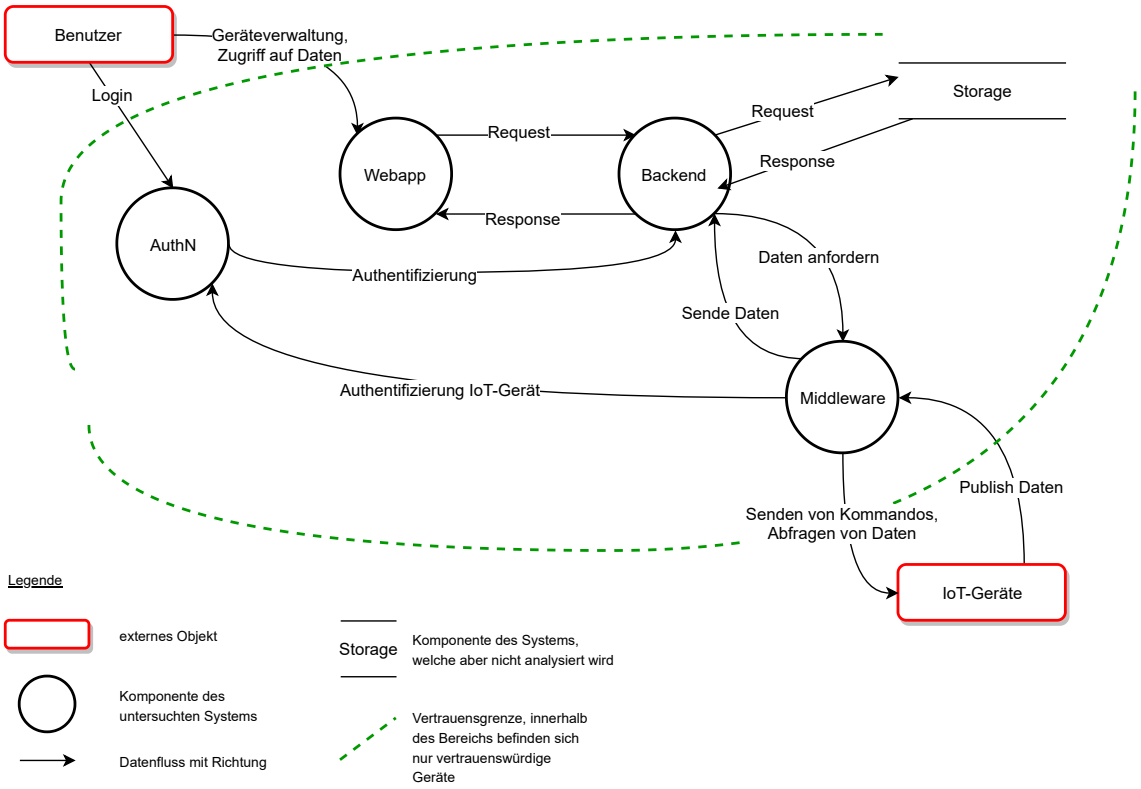


Abbildung 5.1: Datenflussmodell einer generischen IoT-Plattform.

5.1 STRIDE

Das STRIDE-Bedrohungsmodell [Mic09] ermöglicht eine Kategorisierung von Bedrohungen. Diese Kategorien bilden das Akronym STRIDE. Die erste Kategorie wird “Spoofing Identity” genannt und umfasst Attacken auf die Identitäten von Benutzern. Dabei werden Informationen eines Benutzers, die zur Authentifizierung verwendet werden (z.B. eine Benutzername/Passwort Kombination) gestohlen. Diese werden dann von Dritten verwendet, um sich als dieser Benutzer auszugeben. Die zweite Kategorie ist “Tampering with data”, welche alle Bedrohungen umfasst, die auf die unautorisierte und böswillige Veränderung von persistenten Daten abzielen. Unter “Repudiation” werden Bedrohungen zusammengefasst, die eine fehlerhafte Protokollierung von Benutzeraktionen ausnutzen. Das bedeutet, ein Angreifer kann die Logs in einem System beeinflussen und verändern. Das System hat keinen Nachweis, dass diese Aktionen eines Angreifers nicht autorisiert waren. Eine weitere Bedrohung ist die unautorisierte Veröffentlichung von geschützten Daten. Dabei haben Benutzer Zugriff auf Daten, für die sie eigentlich keine Autorisierung und Rechte haben. Denial-of-Service (DoS) beschreibt die sechste Kategorie von Bedrohungen. Solche Attacken zielen darauf ab, den normalen Betrieb eines Webservices zu stören, damit dieser für valide Benutzer nicht verfügbar ist. In der letzten Kategorie geht es um Bedrohungen, die eine nicht genehmigte Erhöhung von Benutzerrechten zur Folge haben. Infolgedessen kann ein Benutzer Rechte in einem System erlangen, für die er keine Genehmigung hat, und

damit Zugriff auf vertrauliche Daten bekommen. Außerdem kann ein Benutzer dadurch den Betrieb eines Systems erheblich beeinflussen. In solch einem Fall hat der Angreifer bereits alle Sicherheitsmechanismen überwunden und kann nur schwer als Angreifer identifiziert werden.

5.2 OWASP IoT Project

Das Open Web Application Security Project IoT [OWA18] wird entwickelt, um Sicherheitsprobleme im Kontext des IoT zu beschreiben und eine Übersicht über Best-Practices zu geben. Damit soll Entwicklern, Unternehmern und Kunden geholfen werden, bessere Entscheidung bei der Entwicklung von IoT-Systemen zu treffen. Im Jahr 2018 wurde eine Liste mit den Top 10 Sicherheitsproblemen veröffentlicht:

1. Schwache, hart kodierte Passwörter
2. Unsichere Netzwerkservices
3. Unsichere Schnittstellen
4. Keine Update-Mechanismen für Betriebssysteme und Software
5. Benutzung von unsicheren, veralteten Komponenten
6. Ungenügende Wahrung der Privatsphäre
7. Unsicherer Datentransfer sowie Datenspeicherung
8. Fehlendes Gerätemanagement
9. Unsichere Standardeinstellungen auf Geräten und in Anwendungen
10. Fehlende Absicherung gegen physische Angriffe

5.3 Bedrohungsmodell für IoT-Plattformen

In diesem Abschnitt werden zunächst die zu schützenden Güter vorgestellt. Anschließend werden die potenziellen Schwachstellen von IoT-Plattformen sowie die möglichen Attacken, die eine oder mehrere Schwachstellen ausnutzen, vorgestellt. Zu jeder Attacke wird auch ein Beispiel aus der realen Welt genannt. Gegenmaßnahmen werden dann im Kapitel 6 behandelt.

5.3.1 Wertgegenstände

Im ersten Schritt muss definiert werden, welche Güter in der Umgebung einer IoT-Plattform schützenswert sind. Das sind auf der Anwendungsebene Benutzerdaten aus Benutzerprofilen sowie Zugangsdaten für die Benutzeroberfläche einer IoT-Plattform und geheime Schlüssel (z.B. SSH-Keys) für den Zugriff auf IoT-Geräte. Auf der nächsten Ebene, der Middlewareebene, gilt es, vor allem die übertragenen Daten zwischen IoT-Geräten und der IoT-Plattform vor unbefugtem Auslesen zu schützen. Auch soll ein unautorisiertes Verändern von Daten in der Middleware und das bösartigte Einspeisen falscher Daten verhindert werden. Hier sind vor allem Daten der IoT-Geräte die zu schützenden Güter. Gleiches gilt für die Transportebene. Auf der letzten Ebene, der Geräteebene, sind die IoT-Geräte selbst die wichtigen Objekte. Der Zugriff auf diese muss geschützt werden, da die IoT-Geräte den elementaren Bestandteil für eine IoT-Plattform bilden.

5.3.2 Schwachstellen

Ökosystem Als Ökosystem wird das gesamte Zusammenspiel aus IoT-Plattform, Geräten wie beispielsweise Sensoren sowie der Kommunikation über ein Netzwerk verstanden. Eine große Schwachstelle in diesem Kontext ist die Kompatibilität der unterschiedlichen Standards zueinander, die in einem IoT-Netzwerk aufgrund der Vielfalt an Geräten benutzt werden. Dies betrifft vor allem die Kommunikationsprotokolle, da um eine Kommunikation unter den einzelnen Komponenten ermöglichen zu können, die Sicherheitsfeatures dieser Protokolle aufgeweicht und angepasst werden müssen. Das offenbart einem Angreifer neue Angriffsflächen.

Physische Schnittstellen Diese Angriffsfläche beschreibt Bedrohungen, wenn ein Angreifer direkten, physischen Zugriff auf ein Gerät in einem IoT-Netzwerk hat. Dadurch kann ein Gerät auf einen unsicheren Status vor einem Update zurückgesetzt werden oder das lokale Speichermedium kann entfernt werden. Schwachstellen sind hier unter anderem externe USB-Ports. Eine weitere Gefahr bieten Command-Line-Interfaces (CLI), die ein Gerät zur lokalen Verwaltung anbietet. Diese können von einem Angreifer missbraucht werden, um einem Gerät höhere Rechte zu verschaffen. Des Weiteren werden bei einem physischen Zugriff sensible Daten wie eine Geräte-ID enthüllt.

Web Interface Eine der größten Angriffsflächen stellen Web-Oberflächen dar. Hier sind vor allem die Zugangsdaten von Benutzern als Schwachstelle zu nennen. Darunter fallen berechenbare Benutzernamen, schwache Passwörter, Standardpasswörter und unsichere Methoden zur Passwortwiederherstellung. Eine weitere, potenzielle Schwachstelle stellt

das Session-Management für einen Benutzer dar. Ist ein authentifizierter Benutzer eingeloggt, so darf er nur Aktionen ausführen und Daten einsehen, für die er autorisiert ist. Dafür sind geeignete Mechanismen zur Webauthentifizierung, Session-Management und Zugriffskontrolle nötig.

Netzwerkdienste Ein Netzwerkdienst stellt in diesem Zusammenhang eine Komponente in einem IoT-Netzwerk dar, die zum Beispiel für die Speicherung von Daten oder die Kommunikation zwischen Geräten zuständig ist. Ein Ausfall dieser Dienste bedeutet gleichzeitig einen Ausfall des gesamten Ökosystems, da die verteilten Geräte und Systeme des IoT-Netzwerks nicht mehr miteinander kommunizieren können. Eine Schwachstelle ist daher die Überlastung des Netzwerks und ein damit verbundener Zusammenbruch der Kommunikation. Eine weitere Schwachstelle stellt die unverschlüsselte Übertragung und damit das mögliche Abfangen und Verändern der gesendeten Daten dar. Allerdings kann sich auch eine vorhandene Verschlüsselung als Schwachstelle herausstellen, wenn diese beispielsweise schlecht implementiert ist oder ein bereits geknackter Algorithmus verwendet wird. Ist eine verschlüsselte Übertragung eingerichtet, gibt es trotzdem noch die Möglichkeit, dass ungesicherte Standardports weiterhin erreichbar sind. Konfiguriert man MQTT zum Beispiel für die Verwendung von SSL, so muss die exklusive Verwendung auch erzwungen werden. Ansonsten ist der MQTT-Broker sowohl per HTTP als auch per HTTPS erreichbar.

Lokale Datenspeicherung Ein weitere Angriffsfläche bietet die lokale Speicherung von Daten in Geräten, vor allem bei Netzwerkdiensten. Ein Hautproblem bildet hierbei das unverschlüsselte Speichern von Daten. Dies ermöglicht ein nicht-autorisiertes Auslesen von Daten aus der Middleware, beispielsweise einem MQTT-Broker oder einer Datenbank. Weitere Bedrohungen bestehen in der mehrfachen Benutzung von symmetrischen Schlüsseln. Dadurch wird einem Angreifer das Entschlüsseln selbst bei angewendeter Datenverschlüsselung vereinfacht.

Third-party-Backend-APIs Mit der Verwendung von Software außerhalb eines IoT-Netzwerkes mittels Drittanbieter-Schnittstellen vergrößert sich auch die Angriffsfläche und es entstehen weitere mögliche Schwachstellen. Diese externen Zugangspunkte müssen geeignet abgesichert werden, da ansonsten sensible Daten aus dem IoT-Netzwerk für Dritte einsehbar sind.

Update-Management Damit die Sicherheit in einer IoT-Umgebung gewährleistet wird, müssen entdeckte Schwachstellen in der Implementierung und benutzter Software behoben werden. Dazu muss eine geeignete Update-Politik vorhanden sein. Ein System, welches nicht regelmäßig aktualisiert wird und kein Update-Management besitzt, enthält Schwachstellen und möglicherweise auch offene Sicherheitslücken in Betriebssystemen sowie in sonstiger installierter Software. Diese können von einem Angreifer aktiv ausgenutzt werden, um die

Kontrolle über das System zu erlangen und Daten zu extrahieren. Weitere Schwachstellen sind nicht signierte Updates und daraus resultierende Malicious-Updates, die von einem potenziellen Angreifer eingeschleust werden.

Authentifizierung & Autorisierung Wenn eine IoT-Umgebung eine Authentifizierung sowie eine Autorisierung innerhalb des Netzwerks zwischen den einzelnen Komponenten und Geräten besitzt, ist diese ein Hauptziel für Angriffe. Schwachstellen sind hier vor allem Passwörter, clientseitige Session-Ids, OAuth-Tokens, Cookies, etc. Durch Wiederverwenden von Session-Keys wird das Risiko erhöht, dass diese abgefangen und von Angreifern verwendet werden. Implementiert das Netzwerk keine Authentifizierung oder Autorisierung, ist es einem Angreifer schutzlos ausgeliefert. Eine weitere Schwachstelle bietet auch ein mögliches Benutzerrollenkonzept.

Hardware (Sensors) Die Schwachstellen bei der Hardware in einem IoT-Netzwerk sind vor allem die schlechte Absicherung gegen physischen Zugriff. Dies ist meist bedingt durch die geringen, vorhandenen Ressourcen. Auch die Rechte eines Gerätes können eine Schwachstelle sein, so sollte ein IoT-Gerät keine administrativen Rechte besitzen. Eine weitere Gefahr droht bei Firmware- und Softwareupdates. Diese sollten vor dem Aufspielen überprüft werden, da ansonsten ein Angreifer eine modifizierte Version aufspielen kann.

5.3.3 Angriffe

Physische Schnittstellen Über den direkt Zugriff auf IoT-Geräte kann Malware der Ransomware in ein IoT-Netzwerk eingeschleust werden.

Web Interface Brute-Force-Angriffe sind weit verbreitete und effektive Angriffe auf die Logindaten von Benutzern einer Webseite. Bei einem Brute-Force-Angriff werden viele Möglichkeiten ausprobiert, um ein schwaches Passwort zu erraten. Wörterbuchangriffe sind eine Erweiterung des Brute-Force-Angriffs. Dabei wird eine Passwortliste zu Hilfe genommen, um das Passwort des Benutzers zu erraten. Meist reichen diese einfachen Angriffe aus, um Eindringlingen Zugang zu einer IoT-Plattform zu ermöglichen. Credential-Stuffing stellt eine weitere Möglichkeit dar. Dabei werden Benutzernamen und Passwortkombinationen aus bekannten Datenlecks (auch von anderen Webseiten) ausprobiert, um Zugriff zu erlangen. Mit einem Long-Password-Denial-Of-Service kann durch das Absenden eines sehr langen Passworts (z.B. 1 Millionen Zeichen) die CPU und der Arbeitsspeicher eines Servers überlastet werden. Dies führt zu einer Nichtverfügbarkeit der Website. Grund dafür ist der serverseitig eingesetzte Hashing-Algorithmus, der dieses Passwort verarbeitet. Weitere Angriffsmöglichkeiten stellen klassische Angriffe auf Weboberflächen dar, wie zum Beispiel Cross-Site-Scripting (XSS) oder SQL-Injections.
Beim XSS wird in eine vertrauenswürdige Webseite ein bösartiges Skript eingespeist. Wenn die Webanwendung die eingegebenen Daten nicht überprüft, werden diese anschließend an

einen Browser eines anderen Benutzer weitergesendet. Dadurch kann der Angreifer indirekt an den Browser eines Opfers bösartigen Code senden, um beispielsweise ein Passwort abzufangen.
Bei einer SQL-Injection handelt es sich um das nicht autorisierter Ausführen eines SQL-Statements, welches von einem Angreifer in einem Eingabefeld der Webanwendung eingegeben wird. Dadurch kann dieser Daten auslesen, löschen oder verändern. Cross-Site-Request-Forgery (CSRF) ist eine weitere Attacke, bei der ein Angreifer einen bereits angemeldeten Benutzer dazu bringt, eine Transaktion in der Applikation auszuführen. Diese Transaktion löst dann eine zustandsverändernde Anfrage an den Server aus, wie zum Beispiel das Ändern der Mailadresse. Ein Angreifer kann beispielsweise einen Link per Mail an ein potenzielles Opfer senden. Ist der Benutzer bereits angemeldet und öffnet den Link, wird automatisch seine Mailadresse geändert.

Ein Beispiel für die Attacken auf Passwörter ist der Angriff auf die Anwendung Instagram und deren Recovery-Code-Mechanismus. Bei dieser Sicherheitslücke handelt es sich um eine Schwachstelle während dem Prozess zum Zurücksetzen des Loginpassworts bei Instagram. Wenn ein Benutzer sein Passwort vergessen hat, so kann er dieses zurücksetzen lassen und bekommt dann einen sechsstelligen Code an seine hinterlegte Mailadresse oder Telefonnummer zugeschickt [Kum19b]. Diesen Code muss der Benutzer innerhalb von 10 Minuten eingeben, um damit seine Identität zu beweisen. Laxman Muthiyah hat herausgefunden, dass dieser Code per Brute-Force Attacke erraten werden kann. Dazu benutzte er mehrere IP-Adressen um innerhalb der 10 Minuten 200.000 verschiedene Codes mittels nebenläufigen Anfragen auszuprobieren. Dies machte 20% aller Möglichkeiten aus und war ausreichend, um einen Account zu übernehmen. Mit 5000 IP-Adressen könnte man auch die 1 Millionen Möglichkeiten ausprobieren und damit die komplette Attacke durchführen.

Netzwerkdienste Gefährdet sind diese Services vor allem durch Denial-of-Service (DoS) Attacken. Eine DoS-Attacke führt eine Vielzahl von gleichzeitigen Anfragen an ein Netzwerk aus. Dies führt dazu, dass ein Service oder Gerät nicht mehr erreichbar ist, da das Netzwerk überlastet wird. Die Verwendung eines kaputten Verschlüsselungsalgorithmus, wie zum Beispiel der Data-Encryption-Standard [NIS99], bietet eine weitere Angriffsfläche. Als weitere Bedrohung ist hier die Replay-Attacke zu nennen. Bei dieser Attacke nimmt ein Angreifer die Identität eines bereits bekannten Gerätes an, täuscht damit das Netzwerk und kann falsche Daten in Umlauf bringen.

Brian Krebs ist ein Security-Journalist und sein Blog wurde als Ergebnis seiner Veröffentlichungen zum Booter-Dienst vDoS-Opfer des größten jemals festgestellten DDoS-Angriff [Sch16]. Die Angriffe auf die Server betrugen bis zu 620 Gigabit pro Sekunde. Dabei wurde der Angriff von einem riesigen Botnetz unterstützt, welches aus vielen ungesicherten IoT-Geräten bestand, z.B. Router oder IP-Kameras.

Lokale Datenspeicherung Attacken auf unverschlüsselte Daten, die beispielsweise in einem Broker oder einer Datenbank liegen, werden häufig mit Standardbenutzernamen und Passwörtern ausgeführt. Dabei bedarf es keines speziellen Angriffs, um Zugang zu erlangen. Oft werden Daten auch mit schwachen oder bereits unsicheren Verschlüsselungsalgorithmen geschützt, welche von Angreifern mit geringem Aufwand entschlüsselt werden können.

Der Sicherheitsforscher John Wethington fand die Datenbank einer Smart-City in China öffentlich zugänglich über einen Webbrowser [Whi19]. In dieser Datenbank waren unter anderem Scans der Gesichtserkennung von hunderten Personen gespeichert. Die Daten dieser Scans stammen von mehreren Kameras in Peking.

Backend-APIs Gängige Angriffe auf ungesicherte APIs sind Injection- und Extraction-Attacken. Bei einer Injection werden Daten von nicht autorisierten Dritten in das System eingeschleust. Eine Extraction bedeutet, dass nicht autorisierte Dritte Zugriff auf private Daten haben. Eine weitere Attacke ist die XML-External-Entity-Attacke. Dabei wird von einem schlecht konfigurierten XML-Parser die Referenz in einer XML-Datenstruktur zu einer externen Entität gelesen und verarbeitet. Darüber kann ein Angreifer eine DoS-Attacke ausführen und Zugriff auf Daten bekommen.

Im April 2019 wurde bekannt, dass JustDial, Indiens größte Suchmaschine, persönlich identifizierbare Daten seiner Kunden über einen ungesicherten API-Endpunkt zugänglich gemacht hatte [Kum19a]. Diese Daten enthalten Benutzernamen, Mailadressen, Telefonnummern, Adresse, Geschlecht, Geburtsdaten, Fotos und weitere sensible Daten. Der Endpunkt wurde von JustDial als veraltet identifiziert und auf dem Server vergessen.

Update-Management Werden die Signaturen von Update-Dateien vor der Installation nicht überprüft, kann ein Angreifer kompromittierte Software über den Updatevorgang auf einem Gerät einschleusen und dadurch Hintertüren einrichten. Eine weitere Angriffsmöglichkeit sind bestehende Attacken auf bekannte Sicherheitslücken, sofern das IoT-Netzwerk nicht über ein geeignetes Update-Management verfügt, um Sicherheitsupdates schnellstmöglich auf alle Geräte verteilen zu können.

Ein aktuelles Beispiel liefern die Angriffe auf die Suchmaschine Elasticsearch [Sch19]. Diese besitzt in früheren Versionen eine Sicherheitslücke, für die es aber bereits Sicherheitsupdates gibt. Allerdings wurde dieses noch nicht auf allen Servern, die Elasticsearch betreiben installiert. Nutzt ein Angreifer die Lücke aus, kann er die Kontrolle über den kompletten Server übernehmen und den Server beispielsweise für (D)DoS-Attacken missbrauchen.

Authentifizierung & Autorisierung Eine bekannte Attacke ist der Man-in-the-Middle (MITM). Dabei hat der Angreifer Zugriff auf die gesamte Kommunikation zwischen zwei Endpunkten und kann damit sämtliche Daten mitlesen. Dem Zugriff auf die Kommunikation gehen allerdings bereits Schwachstellen in der Kommunikation selbst voraus. Der Angreifer

teilt dabei die originale TCP-Verbindung in zwei neue Verbindungen auf, eine zwischen Client und Angreifer sowie zwischen Server und Angreifer. Der Angreifer kann dadurch sämtliche Daten mitlesen, modifizieren und neue Daten einbauen. Dies gilt auch für HTTPS-Verbindungen mit TLS. Eine Attacke auf die Session eines Benutzers wird Session-Hijacking-Attack genannt. Das Ziel des Angreifers ist dabei das Abgreifen der Session-ID, die ein Server nach der erfolgreichen Authentifizierung des Benutzers an diesen zurückschickt. Diese Session-ID kann beispielsweise vom MITM abgefangen werden. Eine Modifizierung dieser Attacke wird Session-Prediction-Angriff genannt. Der Angreifer versucht dabei aus gesammelten validen Session-IDs eine neue valide Session-ID zu erstellen, mit der er Zugriff auf eine Anwendung bekommen kann. Dafür müssen zunächst echte Session-IDs gesammelt und analysiert werden. Die Struktur sowie die Verschlüsselung oder der Hashing-Algorithmus müssen nachvollzogen werden. Eine weitere Variation von Angriffen auf die Session-ID eines Benutzers ist die Session-Fixation-Attacke. Dabei wir die Session-ID des Benutzers bereits vor dem Login von einem Angreifer festgelegt, zum Beispiel über einen modifizierten Link des Angreifers mit einem URL-Parameter.

In einem Smart-Refrigerator von Samsung wurde eine Lücke entdeckt, die einen MITM-Angriff ermöglichen könnte und damit die Google-Benutzerdaten abgreifen könnte. Der Kühlschrank kann den Google-Kalender des Benutzers integrieren, dazu hat Samsung SSL implementiert, um die Integration abzusichern. Aber wie Hacker herausfanden validiert der Kühlschrank das SSL-Zertifikat nicht. Dadurch kann ein Angreifer, der sich im selben Netzwerk befindet, den Kalenderclient im Kühlschrank täuschen und die Google-Benutzerdaten abgreifen.

Hardware (Sensors) Ein Angreifer kann die Daten verfälschen, die ein Sensor liefert oder den Sensor physisch beschädigen. Außerdem kann ein IoT-Gerät auch von einem Angreifer komplett übernommen werden. Dies kann physisch vor Ort geschehen, durch das Ausnutzen bekannter Sicherheitslücken oder das Einspielen kompromittierte Updates.

Ein Beispiel für gekaperte IoT-Geräte ist das Mirai Botnet. Dies ist ein Botnetz, welches aus unsicher konfigurierten IoT-Geräten, wie z.B. Kaffeemaschinen oder IP-Kameras besteht, die von Angreifern übernommen und kontrolliert wurden. Damit wurde unter anderem der Blogger Brian Krebs angegriffen.

Timing-Attack Der Angreifer versucht die Zeit zu analysieren, die ein IoT-Gerät für einen kryptografischen Algorithmus benötigt, um mit diesem Wissen den privaten Schlüssel berechnen zu können. Dafür ist eine präzise Messung der Zeit, die ein Gerät für eine logische Operation benötigt, notwendig. Da sich die Zeiten für unterschiedlichen Input unterscheiden, kann der Angreifer anhand des Outputs und der benötigten Zeit den Input zurückberechnen.

Trusted-Platform-Module-Chips können genutzt werden, um kryptografische Schlüssel zu erzeugen, die sich nicht einfach aus der Hardware extrahieren lassen. Aus einem TPM-Chip von Intel haben es Kryptographen geschafft, mittels einer Timing-Attacke den Signaturschlüssel zu berechnen. Grund dafür waren kleine, aber messbare Zeitunterschiede bei den Signaturoperationen.

6 Sicherheitskonzept

In diesem Kapitel wird das Sicherheitskonzept vorgestellt, welches eine Grundlage für die Architektur der Sicherheit auf den verschiedenen Ebenen einer IoT-Cloudplattform bietet. Das Sicherkeitskonzept ist dabei in die Ebenen unterteilt, die in Kapitel 4 in einem Architekturmodell definiert sind. In Abschnitt 6.5 werden die vorgestellten Technologien und Konzepte noch einmal zusammengefasst.

6.1 Anwendungsebene

Diese Ebene in der Architektur einer IoT-Anwendung beschreibt die Anwendungsschicht, die im allgemeinen Fall auch eine grafische Benutzeroberfläche beinhaltet. Über diese Benutzeroberfläche wird die IoT-Plattform vom Benutzer verwendet und verwaltet. Die Bedrohungen sind hier vor allem API-Missbrauch, bzw. ein unzureichend gesichertes API sowie Brute-Force- und Wörterbuchangriffe auf die Logindaten der Benutzer. Als Gegenmaßnahmen sind eine ausreichende Authentifizierung sowie Autorisierung des Benutzers notwendig.

Die Authentifizierung eines Benutzers kann anhand einiger Richtlinien korrekt und sicher umgesetzt werden. Zunächst sollten die verwendeten Benutzernamen nicht zwischen Groß- und Kleinschreibung unterscheiden. Noch besser ist die Verwendung einer Mailadresse als Benutzername, da diese eine externe (schwache) Validierung des Benutzers über den Mailprovider erfordert. Für einen effizienten Schutz gegen bekannte und automatisierte Attacken auf Passwörter, bietet eine Passwortkontrolle eine erste Hürde. Diese verhindert beim Anlegen eines Benutzeraccounts, das schwache Passwörter gewählt werden. Nach [GFN+17] sind Passwörter mit weniger als acht Zeichen als schwach einzustufen. Daher sollte eine minimale Länge festgelegt werden. Aber auch eine Obergrenze sollte festgelegt werden, um eine Long-Password-DoS-Attacke zu verhindern.

Zudem muss ein sicherer Mechanismus zur Passwortwiederherstellung entworfen werden, um im Falle des Verlustes eines Passwortes den Zugang zu einem Benutzerkonto wiederherstellen zu können. Für die Umsetzung gibt es noch keinen gängigen Standard, zwei Konzepte sind allerdings sehr verbreitet. Als erste Möglichkeit gibt es das Konzept der Sicherheitsfragen. Diese Sicherheitsfragen werden beim Einrichten eines Benutzerkontos festgelegt und vom Benutzer beantwortet. Für die Auswahl der Sicherheitsfragen müssen die folgenden Eigenschaften berücksichtigt werden. Die Antworten müssen unvergesslich, gleichbleibend, möglichst einzigartig und sicher sein. Sicher bedeutet in diesem Kontext,

dass die Antwort auf eine Frage nicht einfach erraten oder leicht recherchiert werden kann. Auch sollte eine Mindestlänge der Antworten festgelegt werden, um ein leichtes Erraten zu verhindern. Hat der Benutzer seine Sicherheitsfragen richtig beantwortet, wird im nächsten Schritt ein neues Passwort gesetzt.

Eine weitere Möglichkeit zum Zurücksetzen eines Passworts bietet das Ausnutzen eines weiteren Authentifizierungsfaktors. Dafür wird beispielsweise ein generierter Link an die vom Benutzer hinterlegte Mailadresse gesendet. Über diesen Link wird dann ein neues Passwort gesetzt. Ein Nachteil ist hier, dass die Annahme getroffen werden muss, dass das Mailkonto des Benutzers nicht kompromittiert ist. Als Erweiterung kann auch auf einen Link verzichtet werden und dafür ein Code an die Mailadresse oder das Smartphone des Benutzers gesendet werden. Dieser Code muss dann zusammen mit dem Benutzernamen eingegeben werden, um ein neues Passwort zu setzen. Bei der Umsetzung dieses Konzepts wird die Authentifizierung des Benutzers an externe Faktoren aus, z.B. den Mailprovider, ausgelagert.

Im nächsten Schritt muss auch ein sicheres Speichern des Passworts in einer Datenbank gewährleistet werden. Idealerweise kennt nur der Benutzer das Passwort, die IoT-Plattform selbst arbeitet nur mit einer gehashten Version und kennt das eigentliche Passwort nicht. Um auch große Passwörter möglichst effizient hashen zu können, kann ein zweistufiges Hashing zum Einsatz kommen. Im ersten Schritt wird das Passwort mit einem schnellen Verfahren wie SHA-512 verarbeitet, um dann in einem zweiten Schritt einen langsameren, aber sichereren Hashing-Algorithmus zu verwenden (z.B. bcrypt). Zusätzlich zum Passwort wird ein Salt-Wert verwendet. Dieser wird gemeinsam mit dem Passwort verschlüsselt und in der Datenbank gespeichert. Ein Salt-Wert ist ein kryptografisch starke, randomisiert gewählte Zeichenkette und pro Passwort einzigartig. Bei der Auswahl geeigneter Algorithmen sollte auf die richtige Balance zwischen Sicherheit und Benutzbarkeit für den Anwender geachtet werden. Die Verifizierung der Zugangsdaten sollte eine akzeptable Dauer beanspruchen, aber trotzdem sicher gegenüber Wörterbuchangriffen und Brute-Force-Attacken sein. Weiter wird der Einsatz einer Multi-Faktor-Authentifizierung (MFA) empfohlen, um Angriffe auf die Passwörter der Benutzer nutzlos zu machen. Die MFA wird auch Zwei-Faktor-Authentifizierung (2FA) genannt und verlangt von einem Benutzer einen weiteren Identitätsnachweis, abgesehen von einem Passwort. Es gibt vier unterschiedliche Typen von Faktoren. Das klassische Passwort oder ein PIN sind Faktoren, die der Benutzer weiß. In diese Kategorie fallen auch die Sicherheitsfragen, die im vorherigen Absatz vorgestellt wurden. Allerdings sind die Antworten auf diese Fragen in den meisten Fällen einfach zu erraten oder über eine Recherche nachzuvollziehen. Hardware-Token, Zertifikate, SMS oder Ähnliches sind Faktoren, die ein Benutzer besitzt. Als dritte Art eines Faktors gelten Dinge, die ein Benutzer eindeutig besitzt, beispielsweise ein Fingerabdruck oder eine Gesichtserkennung. Der letzte Faktor ist die Abhängigkeit des Standorts. Eine Anwendung von MFA benutzt mindestens zwei unterschiedliche Typen von Faktoren aus dieser Liste.

Als eine weitere Sicherheitsschicht ist die eine Re-Authentifizierung sinnvoll. Diese wird erforderlich, wenn der Benutzer sensible Daten, wie z.B. sein Passwort ändern möchte. Um sich zu authentifizieren, muss der Benutzer zuerst sein altes Passwort eingeben. Für eine

Autorisierung mit unterschiedlichen Benutzerrechten, muss ein geeignetes Rollenkonzept mit entsprechender Rechtevergabe entwickelt werden. Eine weitere Angriffsmöglichkeit bieten Drittanbieter-Schnittstellen, die von der IoT-Plattform verwendet werden. Die Wartung dieser Schnittstellen sowie die Behebung von Sicherheitslücken mittels Updates liegt nicht im Verantwortungsbereich der Entwickler der IoT-Plattform und sind damit nicht Teil des Sicherheitskonzepts.

Für einen effektiven Schutz vor Cross-Site-Scripting-Attacken müssen einige Regeln beachtet werden. Als grundlegende Regel gilt, nicht vertrauenswürdige Daten niemals in HTML platzieren, außer in davor definierten Stellen. Allerdings sollte nicht eine Blacklist für nicht vertrauenswürdige Daten, sondern eine Whitelist für vertrauenswürdige Eingaben angelegt werden. Dadurch werden unbekannte und potenziell gefährliche Eingaben verhindert. Weiter sollten nicht vertrauenswürdige Daten mit "\" markiert werden, bevor sie in HTML-Elemente eingefügt werden. Dieser Vorgang wird HTML-Entity-Encoding genannt und verhindert, das ein fremdes Skript oder Änhliches ausgeführt wird. Je nach gewünschter Sicherheitsstufe werden diese Regeln auf HTML-Attribute, JavaScript, CSS-Tags und URL-Parameter angewendet. Für eine korrekte Codierung und Markierung von nicht vertrauenswürdigen Daten sollte ein Security-Encoding-Library verwendet werden, ein Beispiel dafür ist das OWASP-Java-Encoder-Project [Ich18].
Ergänzend dazu sollte eine Content-Security-Policy (CSP) eingeführt werden. Diese wird als Header von einer Webanwendung gesetzt und dient zum clientseitigen Schutz. CSP erzwingt HTTPS und das Laden von Ressourcen (Skripte, Bilder, etc.) von Services, denen der Server der Webanwendung vertraut. CSP kann unter anderem als Feld im HTTP-Response-Header ausgeliefert werden. Dieses Feld wird mit "Content-Security-Policy" deklariert. Um die einzelnen Policies zu definieren, werden verschiedene Typen von Direktiven gesetzt. Fetch-Direktiven legen beispielsweise fest, von wo der Browser des Clients Ressourcen laden darf. Diese Direktive wird dann mit einem Feld "img-src" festgelegt, in der die URLs zum Laden von Bildern spezifiziert werden. Dadurch bietet CSP eine effektive zweite Schutzschicht als Ergänzung zu den vorherigen Maßnahmen.
Auch für den Zugriff auf API-Endpunkte muss der Benutzer authentifiziert werden und seine Identität bestätigen. Danach wird er autorisiert, d.h. er muss auch die entsprechenden Rechte besitzen, um bestimmte API-Endpunkte benutzen zu dürfen. Um eine API abzusichern, gibt es unterschiedliche Möglichkeiten. Grundlegend ist die Verwendung von HTTPS, um die Übertragung von Authentifizierungsdaten zu verschlüsseln. Gleichzeitig können Clients dadurch einen Service authentifizieren und die Integrität der übertragenen Nachrichten ist garantiert. In modernen Webanwendung werden in der Regel REST-APIs implementiert. Representational-State-Transfer (REST) ist ein Architekturstil, der von Roy Fielding vorgestellt wurde [Fie00]. Dabei werden Informationen immer als eine Ressource angesehen, die über einen Uniform-Resource-Identifier von einem Client konsumiert werden kann. REST-APIs sind zustandslos und verwenden HTTP als Kommunikationsprotokoll. Da REST-APIs den Zustand nicht speichern, muss jeder API-Endpunkt bei jedem Zugriff durch einen Client diesen neu authentifizieren. Um dies zu vermeiden, sollte eine zentrale Benutzerauthentifizierung an einem Identiy-Provider (IdP) implementiert werden. Ist die

Authentifizierung erfolgreich, wird dem Client ein Access-Token ausgehändigt. Eine Möglichkeit zur Authentifizierung ist die Verwendung von JSON-Web-Tokens (JWT) [JBS15b], die genaue Anwendung wird im weiteren Verlauf erläutert.
Für eine einfache Gestaltung der Zugriffskontrolle einer API gibt es die Möglichkeit, API-Keys zu verwenden. Ein API-Key wird für einen Client generiert und auf dem Server hinterlegt. Bei jeder Anfrage an die REST-API muss der Client diesen API-Key mitsenden. Allerdings sind diese Keys leicht zu kompromittieren, da jeder, der im Besitz des Keys ist, auch Zugriff auf die geschützten API-Endpunkte hat. Die API selbst sollte die erlaubten HTTP-Methoden (GET, PUT, POST, etc.) auf API-Endpunkte einschränken, sodass nur die erlaubten Methoden für Anfragen verwendet werden können und alle anderen Anfragen abgelehnt werden. Wenn ein Endpunkt Input von einem Client empfängt, muss eine Validierung dieses Inputs vorgenommen werden. Diese Validierung betrifft die Länge, das Format, den Typ des Inhalts sowie den Umfang der Eingabe. Zusätzlich sollten Anfragen auf eine bestimmte Größe limitiert werden. Für die Validierung können Frameworks benutzt werden, wie zum Beispiel “Bean Validation” von RedHat [Red19]. Weiter sollten sichere Parser für eingehende Nachrichten benutzt werden (z.B. XML-Parser). Auch die korrekte Verwendung von HTTP-Status-Codes sollte umgesetzt werden und spezifisch für jeden Status auch den richtigen Code zurückgeben.
OAuth 2.0 [Har12] ist ein Industriestandard für Autorisierungsprotokolle, baut auf HTTP auf und führte eine zusätzliche Autorisierungsebene ein. In herkömmlichen Client-Server-Architekturen muss ein Client, der auf eine Ressource zugreifen möchte, sich mit den Zugangsdaten des Ressourcenbesitzer autorisieren. Dies beinhaltet, dass der Client diese Zugangsdaten für einen erneuten Zugriff bei sich speichern muss (häufig ein Passwort). Wenn der Ressourcenbesitzer einem Client den Zugriff entziehen möchte, so muss er dies für alle Clients machen, da er das Passwort widerrufen muss. Das OAuth2-Framework definiert vier Rollen. Den Ressourcenbesitzer, den Ressourcenserver, einen Client und den Autorisierungsserver. OAuth2 ermöglicht es, dass ein Client bei einer dritten Partei, dem Autorisierungsserver, ein Access-Token erhält. Der Protokollablauf ist in Abbildung 6.1 abgebildet. Dieses Access-Token definiert, im Gegensatz zu einem Passwort, einen Geltungsbereich, eine Lebensdauer und weitere Zugriffsattribute. Diese Access-Tokens werden aber erst dann an einen Client ausgehändigt, wenn der Ressourcenbesitzer dies genehmigt. Der Client benutzt das Access-Token dann, um auf geschützte Ressourcen des Ressourcenbesitzers zuzugreifen.

Das OAuth2-Framework definiert verschiedene “Grant Types”. Diese geben unterschiedliche Möglichkeiten an, wie ein Benutzer authentifiziert werden kann. Es gibt aktuell vier Haupttypen, es können im Rahmen des Frameworks aber Weitere entwickelt werden.
Der erste Typ ist der Autorisierungscode. Nach der erfolgreichen Authentifizierung eines Benutzers durch dessen Benutzernamen und Passwort, wird ein Code an den Client zurückgegeben. Der Client ist ein Service oder eine Anwendung, die vom Benutzer verwendet wird. Mit dem Code kann der Client dann ein Access-Token beim Autorisierungsserver anfordern. Als Erweiterung wird der Einsatz eines Proof-Key-for-Code-Exchange (PKCE) [SBA15] empfohlen. Diese Erweiterung verhindert ist eine Gegenmaßnahme, falls ein Autorisierungscode abgefangen wird. Ein Angreifer kann diesen dann nicht verwenden,

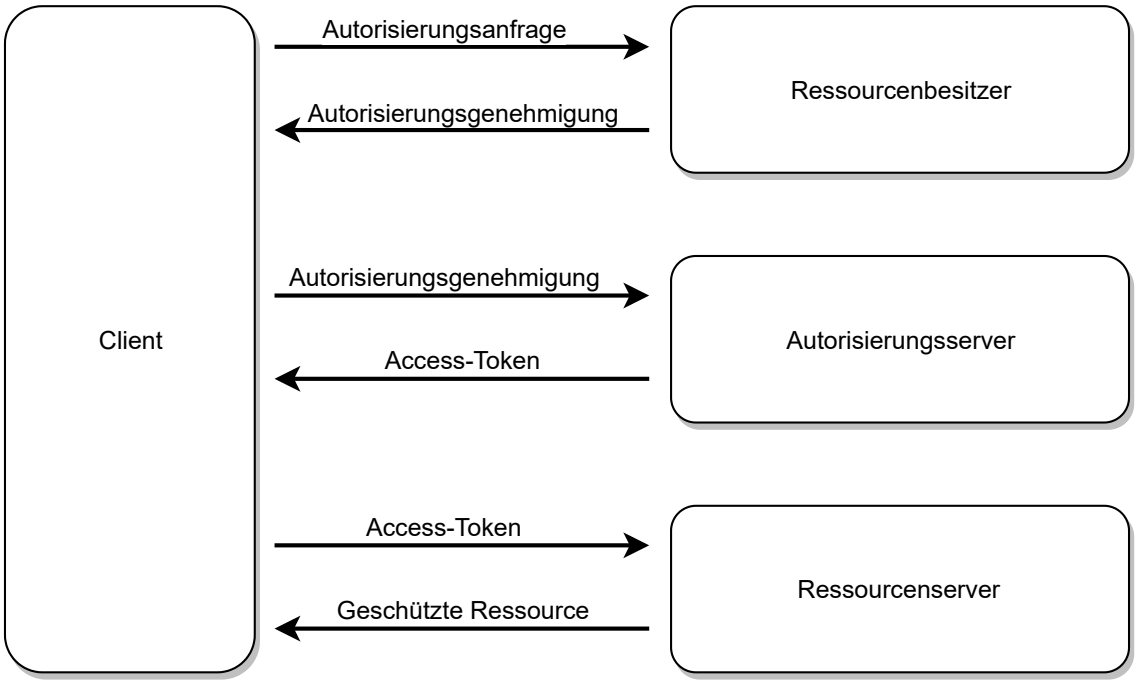


Abbildung 6.1: Abstrakte Darstellung des OAuth2-Protokolls.

um ein Access-Token zu empfangen. Der Verifizierungscode wird von einem Client zu Beginn generiert und besteht aus Zahlen und Buchstaben mit einer minimalen Länge von 43 Zeichen (Maximum sind 128 Zeichen). Anschließend erstellt der Client eine Code-Challenge. Wenn der Client es unterstützt, wird S256 als Algorithmus zur Umformung des Codes verwendet, ansonsten ist der Verifizierungscode selbst die Code-Challenge. S256 verwendet den SHA-256-Hashing-Algorithmus und codiert das Ergebnis mit Base64. Mit der Anfrage für einen Autorisierungscode sendet der Client dann auch diese Code-Challenge an den Autorisierungsserver. Dafür werden in der Anfrage mit “code_challenge” und “code_challenge_method” zwei zusätzliche Parameter übergeben. Der Autorisierungsserver speichert den Autorisierungscode und die Code-Challenge entweder auf dem Server oder verschlüsselt im Code selbst. Im nächsten Schritt sendet der Client dann eine Anfrage für ein Access-Token an den Ressourcenserver. Dieser erkennt in der Anfrage den zusätzlichen Parameter “code_verifier”, berechnet die Code-Challenge und vergleicht das Ergebnis mit dem Code, der mit dem Autorisierungscode mitgesendet wurde.
Als zweite Variante gibt es die Zugangsdaten eines Clients. Dieses Verfahren wird dann verwendet, wenn kein Benutzer involviert ist. Der Client authentifiziert sich mit einer Client-ID und einem Secret (Passwort) beim Autorisierungsserver. Dieser vergleicht die Daten mit den hinterlegten Zugangsdaten für Clients und antwortet im Erfolgsfall mit einem Access-Token.
Die dritte Möglichkeit ist ein sogenannter Geräte-Code und wird auf Geräten ohne Browser oder mit beschränkten Eingabemöglichkeiten (z.B. ein Fernseher) verwendet. Zuerst sendet ein Client (z.B. eine App auf dem Fernseher) eine Anfrage an den Autorisierungsserver mit einer Client-ID und dem Grant-Type Geräte-Code. Der Autorisierungsserver antwortet dann

mit einem spezifischen Geräte-Code, einem Benutzer-Code sowie einer Verifizierungs-URL. Der Benutzer muss dann auf einem anderen Gerät diese URL aufrufen und den angezeigten Benutzer-Code eingeben. In der Zwischenzeit sollte der Client wiederholt beim Autorisierungsserver ein Access-Token mit seiner Client-ID sowie dem Geräte-Code anfordern. Diese Anfragen sollten sich wiederholen, bis entweder der Benutzer den Zugriff genehmigt, ablehnt oder der Benutzer-Code seine Gültigkeit verliert.
Die letzte Möglichkeit, um ein Access-Token anzufordern, ist das Refresh-Token. Dieses wird mit einem Access-Token zusammen an einen Client gesendet. Ist das Access-Token abgelaufen, kann ein Client dieses Refresh-Token an den Autorisierungsserver senden und erhält im Gegenzug ein neues Access- sowie Refresh-Token.
Ein Access-Token wird überwiegend als Bearer-Token [JH12] ausgegeben, welches ein einfacher, zusammenhängender String aus hexadezimalen Zeichen ist, ohne weitere Bedeutung für Clients. In Listing 6.1 ist beispielhaft die Antwort eines Autorisierungsservers nach einer erfolgreichen Authentifizierung zu sehen. Enthalten sind das Access-Token, der Token-Typ, die Gültigkeitsdauer, das Refresh-Token sowie der "Scope". Die Gültigkeit der Tokens wird vom Autorisierungsserver festgelegt. Ist das Access-Token abgelaufen, wird mit dem Refresh-Token ein neues angefordert. Der Parameter "Scope" definiert die Rechte, die der Client mit der Verwendung des Tokens hat. Hat er beispielsweise nur den "Scope" lesen, kann er nur GET-Anfragen an die API senden, aber keine Daten anlegen oder verändern.

```
1  {
2  "access_token":"MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3",
3  "token_type":"bearer",
4  "expires_in":3600,
5  "refresh_token":"IwOGYzYTlmM2YxOTQ5MGE3YmNmMDFkNTVk",
6  "scope":"create"
7  }
```

Listing 6.1: Token-Antwort eines Autorisierungsservers.

Eine weitere Möglichkeit zur Codierung von Tokens sind JSON-Web-Tokens (JWT) [JBS15b]. In JWT werden alle relevanten Informationen (Gültigkeitsdauer, Benutzername, Scopes, etc.) zusammen als JSON-Objekt kodiert. Dadurch erübrigt sich das Speichern der Tokens in einer Datenbank, da alle nötigen Informationen im JWT codiert sind und in jeder Anfrage mitgesendet werden. Ein Beispiel dafür ist in Listing 6.2 und Listing 6.3 dargestellt. Ein JWT kann entweder zusätzlich signiert oder verschlüsselt werden. Die JSON-Web-Signature-Spezifikation (JWS) [JBS15a] definiert die Signatur eines JWT, die JSON-Web-Encryption-Spezifikation (JWE) [JH15] dagegen die Verschlüsselung. Der JSON-Object-Signing-and-Encryption-Header (JOSE) beinhaltet typischerweise zwei Parameter, die angeben, welcher kryptografischen Algorithmus verwendet wird, um die Payload zu schützen. Ausgehend davon wird auch festgelegt, von welchem Typ das kodierte JWT-Objekt ist (JWS oder JWE).

```
{
"typ": "JWT",
"alg": "HS256"
}
```

Listing 6.2: Header eines Access-Token, kodiert als JWT.

In Listing 6.2 ist als “typ” ein JWT und als Parameter “alg” ein Signaturalgorithmus angegeben, welcher das JWT gleichzeitig als JWS-Objekt identifiziert. Die Payload eines JWS-Objekts ist ein einfacher, Base64-kodierter Text, welcher allerdings eine Signatur zur Authentifizierung erhält. Mögliche Signaturalgorithmen sind in den JSON-Web-Algorithmen (JWA) [Jon15] spezifiziert. Im dargestellten Beispiel ist dies der häufig genutzte Algorithmus “HS256”. Dieser Algorithmus verwendet Hash-based-Message-Authentication-Codes (HMAC) zur Signierung einer Nachricht mittels eines gemeinsam genutzten Schlüssels. Die dabei verwendete Hash-Funktion ist SHA-256. Dies ermöglicht eine einfache Möglichkeit, um JWTs zu erstellen und zu validieren. Jede am Protokoll teilnehmende Partei, die den Schlüssel kennt, kann JWTs erstellen. Dadurch gibt es für einen Empfänger keine Gewissheit über die Identität des Senders. Ist dies erforderlich, muss ein asymmetrischer Algorithmus verwendet werden. Wird das JWT als JWE-Objekt identifiziert, so wird der Payload komplett verschlüsselt.

```
{
"exp": 1575039816,
"user_name": "admin",
"authorities": [
        "ROLE_CLIENT"
],
"jti": "282bf2c3-389d-418f-8a80-2b65446060e0",
"client_id": "test-client",
"scope": [
        "write"
]
}
```

Listing 6.3: Payload eines Access-Token, kodiert als JWT.

Für eine sichere Anwendung des OAuth2-Frameworks beschreiben Lodderstedt et al. einige Best-Current-Practices. Grundsätzlich sollte jegliche Kommunikation des OAuth2-Workflows mit TLS Ende-zu-Ende verschlüsselt werden. Zusätzlich dazu kann einer Token-Replay-Attacke vorgebeugt werden, indem Access-Tokens auf den jeweiligen Client beschränkt werden. Dies wird ermöglicht durch Token-Binding [JCBD18] als TLS-Erweiterung. Dafür wird ein Access-Token an ein Token-Binding-Schlüsselpaar gebunden. Bevor ein Client nun eine Anfrage mit einem Access-Token durchführen kann, wird im Zuge des TLS-Handshakes der Besitz des Tokens mittels des Schlüsselpaares überprüft. Der Vorgang geschieht bevor ein Access-Token übertragen wird. Dadurch kann auch eine MITM-Attacke verhindert werden, da dieser nicht im Besitz des kryptografischen Schlüssels des Clients ist. Das gleiche Verfahren kann auch für Refresh-Tokens angewendet werden. Eine weitere Möglichkeit ist die Verwendung von Zertifikaten im Rahmen von Mutual-TLS mit OAuth2 [CBSL19]. Dies ermöglicht eine gegenseitige Authentifizierung des Clients

und des Autorisierungsservers sowie Zertifikat-gebundene Access-Tokens. Dadurch können nur Clients, die im Besitz der privaten Schlüssels (passend zum Zertifikat) sind, das dazugehörige Access-Token verwenden, um auf eine Ressource zugreifen zu können. Für die Anwendung von Mutual-TLS mit X.509-Client-Zertifikaten bietet die Erweiterung zwei Möglichkeiten, die jeweils während dem TLS-Handshake angewendet werden. Um Mutual-TLS-Authentifizierung zu aktivieren, muss ein Autorisierungsserver entsprechend konfiguriert werden und die Authentifizierung vom Client verlangen. Für jede Anfrage muss der Client seine Client-ID als Parameter mit angeben. Diese wird vom Autorisierungsserver verwendet, um die entsprechende Konfiguration auf dem Server zu finden. Anschließend kann das im Zuge des TLS-Handshakes präsentierte Zertifikat mit den erwarteten Daten verglichen werden. Als erste Möglichkeit, um die Verknüpfung zwischen dem Client und dem entsprechenden Zertifikat zu bilden, gibt es die Public-Key-Infrastructure-Mutual-TLS-Methode. Hierbei wird eine vorhandene Public-Key-Infrastructure (PKI) verwendet, um valide X.509-Zertifikate zu erstellen und zur Authentifizierung zu benutzen. Der TLS-Handshake wird dazu benutzt, um zu überprüfen, ob der Client im Besitz des privaten Schlüssels des Zertifikats ist. Zusätzlich wird in den Zertifikaten ein eindeutiger Single-Subject-Alternative-Name (SAN) gesetzt. Den zu erwartenden SAN übermittelt der Client bereits als zusätzlichen Parameter während der Registrierung am Autorisierungsserver. Der Client ist genau dann authentifiziert, wenn die SAN aus dem Zertifikat mit der am Autorisierungsserver registrierten SAN des Clients übereinstimmt. Die zweite Möglichkeit sind selbstsiginierte Zertifikate. Der Client erstellt ein eigenes Zertifikat und registriert dieses beim Autorisierungsserver. Bei jeder Anfrage wird überprüft, ob der Client im Besitz des privaten Schlüssels zum dazugehörigen Zertifikat ist. Ein Client ist authentifiziert, wenn das präsentierte Zertifikat mit einem der hinterlegten Zertifikate für diesen Client übereinstimmt. Allerdings ist bei einem selbstsignierten Zertifikat keine Gewissheit gegeben, dass der Zertifikatsinhaber auch derjenige ist, für den er sich ausgibt. Eine vollständige Authentifizierung auf Basis einer vertrauenswürdigen Zertifizierungsstelle ist nicht möglich. Werden Client-Zertifikate im Rahmen von Mutual-TLS zur Authentifizierung des Clients verwendet, können auch Access-Token und Refresh-Token an diese Zertifikate gebunden werden. Ein Zertifikathash kann direkt in den Token selbst hinterlegt werden, wenn diese als JWTs formatiert sind. Das Zertifikat wird dafür mit SHA-256 gehasht, anschließend Base64-kodiert und als Parameter “x5t#S256” im JWT unter dem Confirmation-Claim hinterlegt. Confirmation-Claims [JBT16] sind mittels “cnf” im JWT hinterlegt und werden benutzt, um den Proof-of-Possession-Schlüssel (PoP) zu identifizieren. Als PoP-Schlüssel dient hierbei der Zertifikathash. Listing 6.4 zeigt ein Beispiel für einen Confirmation-Claim.

```
1        {
2        "cnf":{
3                "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
4        }
5        }
```

Listing 6.4: Confirmation-Claim für ein (X.509-Zertifikat).

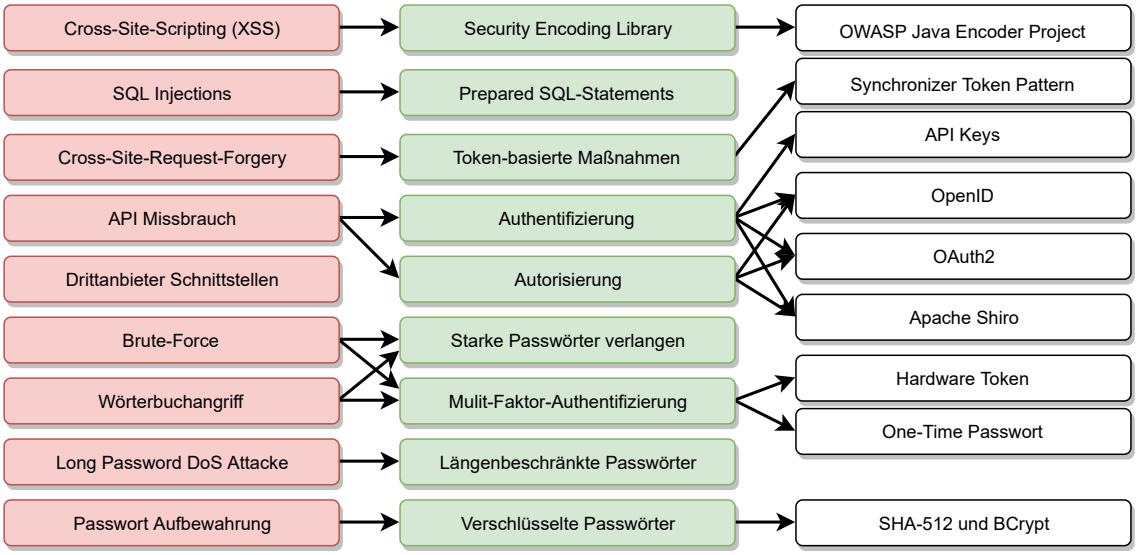


Abbildung 6.2: Angriffe auf Anwendungsebene mit Schutzmaßnahmen sowie beispielhaften Technologien.

6.2 Middlewareebene

Die zweite Ebene bildet die Middleware. Diese besteht in diesem Konzept aus allen Komponenten, die für die geforderten Funktionen der Plattform benötigt werden. Dazu gehören auch Komponenten zur Datenhaltung, z.B. Datenbanken oder Messaging-Komponenten, wie ein MQTT Broker. Auch auf dieser Ebene liegt der Hauptfokus auf Konzepten zur Authentifizierung und Autorisierung, damit nur authentifizierte Geräte Daten an die Middleware schicken sowie nur autorisierte Anwendungen und Geräte Daten aus dieser auslesen dürfen.

Als eines der am häufigsten benutzten Protokolle hat sich das Message Queueing Telemetry Transport (MQTT) Protokoll als Quasi-Standard für die Kommunikation zwischen Anwendungen und IoT-Geräten etabliert. Um Sicherheit mit der Benutzung des MQTT-Protokolls zu gewährleisten, müssen Konzepte auf Transport- sowie Anwendungsebene implementiert werden. Um Daten vor dem unbefugten Zugriff Dritter zu schützen, wird eine lokale Verschlüsselung empfohlen. Der geeignetste und am meisten verwendete Verschlüsselungsalgorithmus dafür ist der Advanced-Encryption-Standard (AES) [DBN+11]. Im Bereich der beschränkten Geräte müssen allerdings Alternativen verwendet werden, da der AES viele Hardwareressourcen benötigt. Empfehlungen für leichtgewichtige Verschlüsselungsalgorithmen sind im ISO-Standard 29192 [ISO12] zu finden. Auf Transportebene wird TLS [RD08] als Verschlüsselungsverfahren eingesetzt, um ein Mitlesen der Daten während der Übertragung zu verhindern.

Auf Anwendungsebene wird definiert, wie sich ein Gerät authentifizieren kann und welche Rechte es bekommt, es wird zwischen Authentifizierung und Autorisierung unterschieden. Das Basiskonzept ist eine einfache Authentifizierung mittels Benutzername und Passwort, welche als Felder in der initialen CONNECT-Nachricht vom Client mitgeschickt werden. Der Header in dieser Nachricht besteht aus einem fixen und einem variablen Teil. Der fixe Teil hat eine Länge von zwei Bytes und definiert nur den Pakettyp (in diesem Fall ein CONNECT-Paket) sowie die restliche Länge aus variablem Header und Payload. Der variable Header ist 10 Bytes groß und besteht in einem CONNECT-Paket aus den folgenden vier Feldern. Das erste Feld ist der Protokollname mit dem Wert MQTT. Im Zweiten wird der Protokolllevel festgelegt. Das Level entspricht der Protokollversion, z.B. für MQTT in der Version 3.1.1 ist das Level vier. Das dritte Feld legt unterschiedliche Verbindungsparameter fest. Diese sind “Clean Session”, “Will Flag”, “Will QoS”, “Will Retain”, “Password Flag” und “Username Flag”. Das letzte Feld im variablen Header ist der “Keep Alive”-Parameter. Dieser legt ein Zeitintervall fest, welches als Maximum zwischen dem Empfang von zwei Kontrollpaketen auf Serverseite gilt. Wird dieses Zeitintervall überschritten, muss der Server die Verbindung zum Client schließen. Sind die Parameter “Password Flag” und “Username Flag” auf 1 gesetzt, müssen in der Payload ein Benutzername sowie ein Passwort vorhanden sein. Die Payload muss Werte entsprechend der Parameter im Header enthalten. Das Passwort darf dabei bis zu 65535 Bytes groß sein. Dieses Nachricht wird an den Broker übermittelt. Ist die Verbindungsanfrage valide und sind die Zugangsdaten korrekt, bestätigt der Server die Verbindung mit einer CONNACK-Nachricht, welche den Statuscode 0 beinhaltet. Um eine erweiterte Authentifizierung zu ermöglichen, können

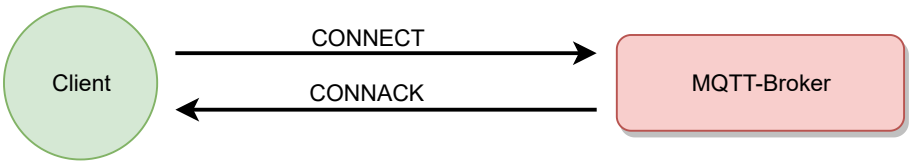


Abbildung 6.3: Verbindungsaufbau zwischen Client und MQTT-Broker.

weitere Konzepte umgesetzt werden. Zum einen der Client-Identifier als Ergänzung zur Kombination aus Benutzername und Passwort. Jeder MQTT-Client muss einen eindeutigen Client-Identifier besitzen, welcher zusammen mit der Kombination aus Benutzername und Passwort validiert wird. Dabei wird überprüft, ob zum einen der Benutzername sowie das zugehörige Passwort korrekt sind, und zum anderen ob die Client-ID zu genau dieser Kombination gehört. Die Client-ID ist ein UTF-8-kodierter [Yer03] Text, welcher standardmäßig eine Länge von bis zu 23 Zeichen besitzt. Manche MQTT-Broker bieten allerdings auch die Möglichkeit, längere IDs zu verwenden. Wird eine leere Client-ID im Header gesetzt, ist die Verbindung zwischen Client und Broker zustandslos und der Broker speichert keinerlei Daten des Clients. Ist eine Verbindung dagegen persistent, so speichert der Broker sämtliche Abonnements sowie alle verpassten Nachrichten eines Clients. Ist die Client-ID nicht gesetzt, muss im Header der Schalter für eine “Clean-Session” gesetzt werden.
Zum anderen gibt es eine zertifikatbasierte Authentifizierung, bei der X.509 Zertifikate, die

als Standard für Public-Key-Zertifikate gelten [BSP+08], verwendet werden. Das Zertifikat wird im Zuge des TLS-Handshakes dem Broker übergeben, nachdem das Serverzertifikat vom Client validiert wurde. Der Broker verwendet dann die Informationen aus dem Zertifikat, um den Client zu authentifizieren. Dies bietet die Möglichkeit der Authentifizierung von Clients auf Transportebene. Ungültige Clients können gesperrt werden noch bevor CONNECT-Nachrichten gesendet werden. Als große Schwierigkeit bei der Verwendung von Clientzertifikaten wird die Bereitstellung auf dem Gerät angesehen. So muss im Vorfeld über eine sichere Verbindung das Zertifikat für jeden neuen Client erstellt und auf dem Gerät hinterlegt werden. Dafür sollte eine Public-Key-Infrastruktur (PKI) [BSP+08] aufgesetzt und verwendet werden.
Eine PKI besteht aus einer Zertifizierungsstelle, einer Registrierungsstelle, einer Zertifikatssperrliste sowie einem Verzeichnisdienst. Die Zertifizierungsstelle stellt Zertifikate an Clients aus und unterstützt andere Clients bei der Verifizierung eines Zertifikats. Im ersten Schritt muss der Client ein Schlüsselpaar erstellen (bestehend aus einem privaten und einem öffentlichen Schlüssel) und den öffentlichen Schlüssel der Zertifizierungsstelle speichern. Im nächsten Schritt stellt die Zertifizierungsstelle ein Zertifikat für den öffentlichen Schlüssel des Clients aus und hinterlegt dieses in einem Verzeichnisdienst, nachdem der Client sich authentifiziert hat. Dazu wird die Registrierungsstelle benutzt, um sicherzustellen, dass die Identität eines Clients mit den angegeben Daten übereinstimmt. Wenn ein Zertifikat von der Zertifizierungsstelle widerrufen wird, beispielsweise weil der private Schlüssel des Clients kompromittiert wurde, wird das Zertifikat in der Zertifikatssperrliste eingetragen. Ein Client kann sich nun gegenüber einem anderen Client mit diesem Zertifikat authentifizieren. Die Echtheit dieses Zertifikats bestätigt dabei die Signatur der Zertifizierungsstelle. Diese kann der andere Client mittels des öffentlichen Schlüssels der Zertifizierungsstelle überprüfen. Da die Gültigkeit eines Zertifikats für einen Client in der Regel langfristig ausgelegt ist, muss eine Möglichkeit bestehen, um Zertifikate zu widerrufen und Clients mit ungültigen Zertifikaten zu erkennen, z.B. im Fall der Kompromittierung eines Gerätes durch einen Angreifer. Als erste Möglichkeit gibt es die bereits vorgestellten Zertifikatssperrlisten, um auf Brokerseite ungültige Zertifikate zu erkennen. Zertifikatssperrlisten sind einfache Listen, welche die ungültigen Zertifikate beinhalten. Für kleinere IoT-Systeme mit wenigen Clients und somit weniger (möglichen) ungültigen Zertifikaten sind Widerruflisten eine gute Lösung. Wird die Liste aber entsprechend groß (z.B. Millionen von Zertifikaten), wird der Authentifizierungsmechanismus auf Brokerseite entsprechend aufwendiger. Außerdem ist bei der Verwendung von mehreren Brokern die Synchronisation der Liste ein Problem.
Eine zweite Möglichkeit bietet das sogenannte Online-Certificate-Status-Protocol (OCSP) [SMA+13]. Das OCSP ermöglicht es, den aktuellen Status eines Zertifikates zu überprüfen. Der Status eines Zertifikats kann entweder gültig, widerrufen oder unbekannt sein. Dabei agiert der MQTT-Broker als OCSP-Client und sendet eine Anfrage auf die Gültigkeit eines Client Zertifikats an den sogenannten OCSP-Responder. Der OCSP-Responder ist ein Validierungsservice, welcher in der Regel vom Herausgeber des Zertifikats betrieben wird. Dieser gibt eine sekundengenaue Auskunft darüber, ob ein angefragtes Zertifikat ungültig ist oder nicht. Die OCSP-Response wird vom OCSP-Responder digital signiert. Allerdings bietet der OCSP-Responder keine Validierung des Zertifikats an, die Korrektheit des angeforderten Zertifikats wird nicht überprüft.

Eine weitere Möglichkeit bietet das OAuth2-Framework [Har12]. Der Client meldet sich mit Benutzername und Passwort beim Autorisierungsserver an. Ist die Anmeldung erfolgreich, bekommt der Client ein Token zugeschickt, mit dem er sich beim Broker authentifizieren kann.

Zur Umsetzung von Autorisierungsmechanismen bietet der MQTT-Broker Zugriffskontrolllisten, eine rollen-basierte Zugriffskontrolle oder die Verwendung des OAuth2-Frameworks an. In Zugriffskontrolllisten wird festgelegt, welcher Client auf welche Ressourcen und mit welchen Rechten (schreiben, lesen) zugreifen darf. Eine rollen-basierte Zugriffskontrolle bietet eine weitere Ebene zwischen den Ressourcen und Benutzern (Clients). Rollen sind in diesem Kontext abstrahierte Ressourcen, was eine Verwaltung der einzelnen Rechte eines Clients sehr vereinfacht, da nicht für jede Ressource die Berechtigten sowie die Rechte definiert werden müssen. Als dritte Alternative gibt es das OAuth2-Framework. Der Workflow ist hierbei derselbe wie bei der Authentifizierung, das Token definiert auch die Rechte, die der Client besitzt. In der umgekehrten Richtung bietet MQTT allerdings keine Möglichkeit für den Client den MQTT-Server zu authentifizieren. Lediglich bei der Nutzung von TLS kann der Client über das SSL-Zertifikat den Server authentifizieren. Eine grundlegende Sicherheitsmaßnahme, um zu verhindern dass Clients auf fremde Topics publishen können ist, dass generell nur der Ersteller eines Topics auch Nachrichten auf dieses senden kann. Zwar können andere Clients auf das Topic subscriben, allerdings nicht die Daten des Topics selbst verfälschen, indem sie nicht-autorisiert Nachrichten publishen.

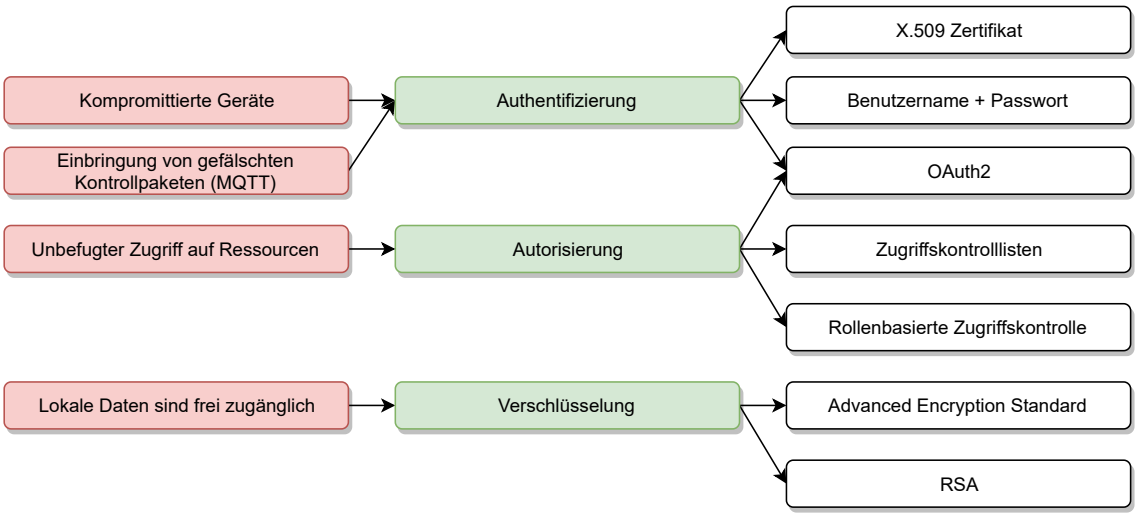


Abbildung 6.4: Angriffe auf Middlewareebene mit Schutzmaßnahmen sowie beispielhaften Technologien.

6.3 Transportebene

Die Transportebene umfasst die Kommunikation der Ebenen untereinander. Dazu zählen die Netzwerktechnologien und Kommunikationsprotokolle. Essentiell als Schutzmaßnahme ist eine Transportverschlüsselung. Transport-Layer-Security (TLS) ist ein Verschlüsselungsprotokoll, welches auf dem verbindungsorientierten Transportprotokoll TCP [Pos81] aufgesetzt wird. Mit der Zunahme von beschränkten Geräten in IoT-Netzwerken findet auch die Anwendung des Constrained Application Protokolls (CoAP) [SHB14] immer mehr Verwendung. Auf Geräten mit beschränkter Hardware ist ein verbindungsorientiertes Transportprotokoll sehr schwergewichtig und daher oft nicht realisierbar. Ist es in einer Client-Server-Anwendungen trotzdem erforderlich das Request-Response-Pattern umzusetzen, kann CoAP zum Einsatz kommen. HTTP ist ressourcenintensiv in der Implementierung sowie in der Netzwerkauslastung. Damit ist eine Implementierung und Nutzung mit Class-1-Geräten (vgl. Tabelle 6.1) nicht möglich. Außerdem basiert HTTP auf TCP, im Gegensatz dazu verwendet CoAP UDP als Transportprotokoll, ist damit verbindungslos und ressourcenschonender. Mögliche Paketverluste mit UDP werden im IoT typischerweise gering gewichtet, wohingegen die Effizienz im Vordergrund steht. CoAP ermöglicht RESTful-Services [BCS12] auf Basis von HTTP und definiert die vier bekannten Requestmethoden: GET, PUT, POST und DELETE. CoAP definiert weiter einen komprimierten, lediglich vier Byte großen Nachrichtenheader. Mit optionalen Headern kann der Header für eine Nachricht maximal 10 bis 20 Bytes groß werden. CoAP kann außerdem mittels einer Erweiterung auch das Publish-Subscribe-Pattern unterstützen [BCS12] und ist vollständig kompatibel mit HTTP. Da in einem verbindungslosen Dienst der Sender nicht auf die Bestätigung des Empfängers über den Empfang einer Nachricht warten muss, gibt es für CoAP Konzepte zur Reduzierung der erneuten Übertragung von Nachrichten. Dadurch wird die Netzwerkauslastung reduziert. CoCoA+ [AB17] ist ein solches Konzept für eine erweiterte Überlastkontrolle.

Um mit dem verbindungslosen Transportprotokoll UDP [Pos80] und der Verwendung von CoAP trotzdem eine Verschlüsselung des Transportes zu ermöglichen, wurde DTLS [RM12] als eine Variante des TLS entwickelt. Der Vorteil ist dabei, die Sicherheitsfunktionen von TLS auf einem leichtgewichtigen Transportprotokoll verwenden zu können. Dazu soll TLS möglichst komplett über UDP genutzt und nur die nötigsten Änderungen vorgenommen werden. Das größte Unterschied von UDP zu TCP liegt dabei in der Zuverlässigkeit beim Senden von Nachrichten. Während dem TLS-Handshake zum Verbindungsaufbau gibt es eine definierte Ordnung, in welcher die Nachrichten gesendet und empfangen werden müssen. Diese strikte Ordnung ist inkompatibel zu den Eigenschaften von UDP wie der Neuanordnung oder dem Verlust von Nachrichtenpaketen. Als eine weitere Hürde sind TLS-Handshake-Nachrichten meist größer als jedes Datagram. Um das Problem des Paketverlustes während des Handshakes zu lösen, gibt es einen simplen “Retransmission Timer” auf beiden Seiten, beim Sender und beim Empfänger. Läuft dieser Timer ab, wird eine erneute Nachricht mit dem Zusatz “retransmit” gesendet, sodass der Empfänger weiß, dass eine Nachricht verloren ging. Um eine falsche Reihenfolge der Nachrichten zu verhindern, werden diese mit spezifischen Sequenznummern versehen. Dadurch kann der Empfänger

immer bestimmen, ob eine Nachricht auch diejenige ist, die als nächste erwartet wird. Ist das nicht der Fall, wird diese Nachricht in eine Warteschlange verschoben. Da sowohl TLS- als auch DTLS-Handshake-Nachrichten bis zu $2^{24} - 1$ Bytes groß werden können, UPD Datagramme aber auf weniger als 1500 Bytes limitiert sind, ist die Nachrichtengröße ein Problem. Daher werden DTLS-Handshake-Nachrichten fragmentiert in mehrere Datagramme und jeweils in einem eigenen IP-Datagramm versendet. Der Empfänger kann dann die Nachricht aus den empfangenen Fragmenten wieder zusammensetzen.

Auf der Transportebene muss auch die Entscheidung für ein geeignetes Kommunikationsprotokoll getroffen werden. Wie bereits in Kapitel 6.2 vorgestellt, wird häufig MQTT im IoT verwendet. Eine Erweiterung des MQTT-Protokolls stellt MQTT-SN [ST13] dar. Dabei steht die Abkürzung SN für Sensor-Networks. Der Zugriff auf die Sensornetzwerke ist dabei nicht direkt, sondern geschieht über Gateways. Sensornetzwerke sind große Netzwerke, die viele Sensoren und Aktuatoren enthalten. Um die Daten, welche diese Geräte sammeln, für IoT-Anwendungen verfügbar zu machen, müssen die Sensoren an IoT-Systeme angeschlossen werden. Da alle Geräte in einem Sensornetzwerk nur über sehr beschränkte Hardware verfügen, wird mit MQTT-SN ein leichtgewichtiges Protokoll vorgestellt. MQTT-SN unterstützt alle Eigenschaften des herkömmlichen MQTT-Protokolls, ist aber unabhängig vom verwendeten Transportprotokoll und benötigt keine TCP/IP-Protokollstack. Eine Verwendung von UDP als Transportprotokoll ist allerdings möglich, eine sichere Konfiguration wird im weiteren Verlauf vorgestellt. Ein Sensor kann unter Verwendung eines geeigneten Transportprotokolls Daten an ein Gateway publishen. Das Gateway übersetzt dann die Nachricht von MQTT-SN in MQTT und leitet die Nachricht an einen herkömmlichen MQTT-Broker weiter. Ab diesem Punkt sind die Sensordaten für Anwendungen verfügbar.
Für eine sichere Verwendung des MQTT-Protokolls wird die Einrichtung von TLS empfohlen. Ein großer Nachteil von TLS ist allerdings der signifikant höhere Kommunikations- und Berechnungsaufwand für die Validierung von Zertifikaten. Außerdem müssen Listen mit widerrufenen Zertifikaten angelegt und gepflegt werden, um die Verwendung von ungültigen Zertifikaten verhindern zu können. Dieses Problem geht das einfache Sicherheitskonzept AugMQTT [SKCH16] an, das auf dem AugPAKE-Protokoll basiert. In einem Password-only-Authenticated-Key-Exchange-Protokoll (PAKE) authentifizieren sich zwei Parteien mittels generierter Schlüssel. Diese Schlüssel werden mit einem Passwort generiert, welches nur die beteiligten Parteien kennen. Dadurch ist eine gegenseitige Authentifizierung möglich, sowie der Aufbau von sicheren Kommunikationskanälen. Dieses Protokoll ist allerdings anfällig für Wörterbuchattacken auf die Passwörter, welche den zentralen Schutz darstellen. Wird ein Passwort kompromittiert, so ist es für einen Angreifer möglich, sich unbemerkt in die Kommunikation einzuschleichen. Ein PAKE-Protokoll gilt als sicher, wenn keine Wörterbuchangriffe direkt auf die Passwörter selbst ausgeführt werden können, sondern die Validierung eines geratenen Passworts nur im direkten Austausch mit einer ehrlichen Partei möglich ist. Das Augmented-PAKE-Protokoll (AugPAKE) [SK18] führt eine weitere Schutzschicht ein, die als Schutz vor der Kompromittierung von Servern fungiert. Dabei speichert der Server keine passwortähnlichen Daten, sondern nur einen Passwortvalidierer und ist damit nicht weiter anfällig für Angriffe auf das Passwort selbst. Der

Angreifer kann sich damit nicht als ein authentifizierter Benutzer ausgeben, da das eigentliche Passwort nicht aus dem Validierer berechnet werden kann. Das AugPAKE-Protokoll hat zum Schlüsselaustausch einen ähnlichen Berechnungsaufwand wie das Diffie-Hellman-Protokoll [Res99], dieses bietet aber von sich aus keine Authentifizierung. Das Protokoll besteht aus zwei Phasen, der Initialisierung und der eigentlichen Ausführung des Protokolls. Während der Initialisierung berechnet der Benutzer einen Passwortvalidierer $W = g^{w'}$, welcher auf dem Server gespeichert wird. Dabei ist w' eine gehashte Version des eigentlichen Passwortes des Benutzers. Anschließend muss der Benutzer zwei und der Server 2.17 modulare Potenzierungen ausführen, um so gegenseitig zu beweisen, dass sie das Passwort kennen. Dabei tauschen der Benutzer und der Server insgesamt vier Nachrichten aus, jeweils zwei werden gesendet. Als Ergebnis besitzen beide Parteien einen authentifizierten Session Key. Das AugPAKE-Protokoll ist ein Zero-Knowledge-Proof-Protokoll und sicher gegen passive, aktive und offline Wörterbuchattacken. AugMQTT überträgt das AugPAKE-Protokoll auf das MQTT-Protokoll. Zwischen einem Publisher/Subscriber und dem Broker wird initial wie beschrieben das AugPAKE-Protokoll ausgeführt. Anschließend kann der Publisher eine mit einem sicheren, symmetrischen Schlüssel verschlüsselte Nachricht an den Broker senden, welche den authentifizierten Session Key (SK) und den eigentlichen Nachrichteninhalt enthält. Nachdem der Broker die Nachricht entschlüsselt hat, wird der Inhalt in einem entsprechenden Topic gespeichert. Wenn der Broker die verschlüsselte Nachricht an einen Subscriber schickt, existiert auch für den Subscriber und den Broker ein SK, der Subscriber kann dementsprechend die Nachricht entschlüsseln. Damit bietet AugMQTT nicht nur Authentifizierung für Publisher, Subscriber und Broker, sondern auch Vertraulichkeit und Integrität der MQTT-Nachrichten. Im Vergleich mit TLS benötigt AugMQTT keine Validierung für Zertifikate und muss auch keine Gültigkeitschecks für Zertifikate ausführen. Dadurch wird auch keine komplexe PKI im Hintergrund benötigt.

Um Angriffe gegen die Netzwerkschicht grundsätzlich zu verhindern, kann ein IoT-Netzwerk ein eigenes Virtual-Private-Network (VPN) zur Kommunikation verwenden. Dabei wird der gesamte Netzwerkverkehr zwischen zwei Geräten verschlüsselt über das Internet übertragen. Allerdings ist der Einsatz eines VPN mit beschränkten Geräten schwierig, da eine extra Konfiguration verwendet werden muss. Diese ist für diese Art der Geräte zu schwergewichtig.

Herkömmliche Mechanismen zum Schutz vor DDoS-Angriffen benötigen eine hohe Stromversorgung, Rechenleistung und Langzeitmonitoring. Dies ist in einem IoT-Netzwerk mit beschränkten Geräten nicht gegeben. Wenn das IoT-Netzwerk in einem eigenen Netzwerk läuft und die IoT-Geräte sowie die Middleware keine direkte Verbindung zum Internet haben, ist die Konfiguration einer Firewall auf Anwendungsebene ausreichend. Dafür gibt es beispielsweise die Uncomplicated Firewall (UFW) unter Linux. Wenn das IoT-Netzwerk über das Internet kommuniziert, gibt es für den lokalen (D)DoS-Schutz auf den IoT-Geräten unterschiedliche Ansätze. Mit der Verwendung von DTLS ist eine Gegenmaßnahme vor DoS-Angriffen der sogenannte “Return Routability Check” [GKS19]. Bei diesem Check überprüft der Empfänger einer Nachricht mithilfe eines Cookies, ob der Sender unter der genannten Adresse (IP + Port) erreichbar ist. Dieser Cookie ist eine Änderung von DTLS gegenüber TLS und wird während des Handshakes ausgetauscht,

wenn der Empfänger den Sender verifizieren möchte (optional). Dabei wird während dem Handshake der genannte Cookie an den Sender geschickt. Dieser muss diesen Cookie in einer neuen Verbindungsanfrage mitschicken, der Empfänger kann dann die Daten aus dem Cookie validieren. Dieser Check verhindert vor allem Angriffe von gefälschten oder verschleierten IP-Adressen. Ein weiterer Mechanismus stellt ein kryptografisches Puzzle dar, welches von einem Sender zunächst gelöst werden muss [GKS19]. Allerdings ist die Performanz dieses Mechanismus in beschränkten Netzwerken nicht ausreichend, und Angreifer mit geeigneter Hardware können diese Puzzle trotzdem lösen. Sind genügend Hardwareressourcen vorhanden, kann im Falle eines DoS-Angriffes die Schwierigkeit dieser Puzzle sukzessive erhöht werden. Dadurch wird eine Überlastung von IoT-Geräten verhindert.

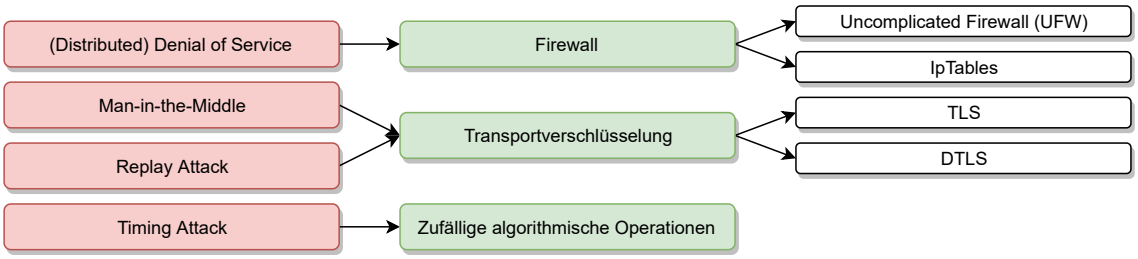


Abbildung 6.5: Angriffe auf Netzwerkebene mit Schutzmaßnahmen.

6.4 Geräteebene

Die unterste Ebene umfasst die IoT-Geräte. Dieser Begriff bezeichnet die IoT-Geräte, die Sensoren und Aktuatoren bereitstellen und Daten liefern, die über die Anwendungsebene eingesehen und verwaltet werden können. Die IoT-Geräte haben häufig nur sehr beschränkte Hardwareressourcen und werden in drei Klassen durch RAM und Flashspeichergröße unterschieden [BEK14]. Aufgrund ihrer beschränkten Hardware ist es schwierig, gängige Sicherheitsmechanismen zu implementieren. Eine Transportverschlüsselung mit TLS beispielsweise benötigt mehr Rechengeschwindigkeit, als die Geräte in diesen drei Klassen bieten.

Name	RAM **Data size**	Flash **Code size**
Class 0	<< 10 KiB	<< 100 KiB
Class 1	10 KiB	100 KiB
Class 2	50 KiB	250 KiB

Tabelle 6.1: Klassifizierung von IoT-Geräten mit eingeschränkter Hardware.

Neben Angriffen auf das Netzwerk, für welche Gegenmaßnahmen im Kapitel 6.3 vorgestellt werden, stellt vor allem die Kompromittierung eines Geräts eine große Bedrohung dar. Um eine möglichst einfache Konfiguration und Bedienung zu ermöglichen, werden IoT-Geräte häufig mit Standardpasswörtern und Standardbenutzernamen ausgeliefert. In sehr häufigen

Fällen werden weder die Passwörter noch die Benutzernamen vom Anwender geändert. Damit ist es für einen Angreifer sehr einfach, sich Zugriff auf diese Geräte zu verschaffen. Als ein Teil dieses Sicherheitskonzeptes ist es also notwendig, die Passwörter auf IoT-Geräten nach der initialen Konfiguration sofort zu ändern. Das neue Passwort sollte, wie schon in Abschnitt 6.1 beschrieben, nicht kürzer als acht Zeichen sein, da es sonst als schwach einzustufen ist. Passwörter sollten weder lexikalische Wörter noch sich wiederholende oder sequentielle Zeichen enthalten (z.B. 'aaaaaa' oder '123456') [GFN+17]. Außerdem sollten auch keine kontextspezifischen Wörter, wie z.B. der Name eines Gerätes, verwendet werden. Weiter sollte dem Benutzer eine Messung und Anzeige der Stärke eines eingegebene Passworts visuell angezeigt werden. Diese Überprüfung erfolgt nach den oben genannten Kriterien. Außerdem sollte ein gewähltes Passwort mit einer Liste an geleakten Passwörtern überprüft werden. Ein Beispiel ist dafür die Seite haveibeenpwned [Hun19], welche Passwörter auflistet, die bereits Teil eines Datenlecks sind. Diese Überprüfung ist sinnvoll, da viele (auch sichere) Passwörter aus Komfortgründen mehrfach verwendet werden. Sollten IoT-Geräte vom Hersteller nicht automatisch mit einmaligen und starken Passwörtern ausgeliefert, so sollte ein Passwortwechsel durch den Anwender beim Einrichten erzwungen werden. In Kombination mit einer Überprüfung des neuen Passwortes können sichere Zugangsdaten für IoT-Geräte realisiert werden. In Kalifornien beispielsweise wird ab dem 01.01.2020 per Gesetz der Einsatz von Standardpasswörtern auf elektronischen Geräten verboten [Köv18]. Ein weltweiter Standard wäre in diesem Fall ein wichtiger und weiterer Schritt. Für das sichere Speichern sowie den Schutz von Zugangsdaten gegenüber einem Angreifer ist der Hersteller des IoT-Gerätes verantwortlich, Maßnahmen dafür sind nicht Teil dieses Konzepts. Wenn es die Hardware eines IoT-Gerätes zulässt, sollte immer eine Public-Key-Authentifizierung mittels SSH [LY06] verwendet werden. Dabei kann sich der Benutzer mithilfe eines Schlüsselpaares, bestehend aus einem privaten und einem öffentlichen Schlüssel, beim Gerät anmelden. Dazu muss der öffentliche Schlüssel auf dem Gerät hinterlegt werden. Der Benutzer erstellt bei der Anmeldung eine Signatur mit seinem privaten Schlüssel. Das Gerät validiert diese Signatur anschließend mithilfe des öffentlichen Schlüssels und gibt den Zugang frei.

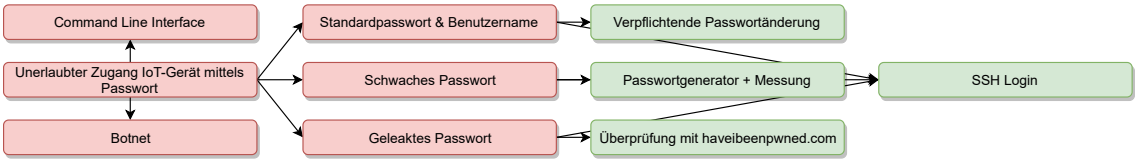


Abbildung 6.6: Angriffe auf Geräteebene mit Schutzmaßnahmen.

6.5 Zusammenfassung

In Abschnitt 6.1 werden Konzepte zur Passwortkontrolle, Wiederherstellung von Passwörtern, sowie deren sichere Verwahrung genannt. Als weitere Sicherheitskonzepte werden eine Multi-Faktor-Authentifizierung und ein Konzept zur Prävention vor XSS-Attacken

beschrieben. Außerdem werden Möglichkeiten zur Absicherung einer API vorgestellt. Als letztes Konzept wird OAuth2 vorgestellt, mit den erweiternden Funktionen JWT, Token-Binding und Mutual-TLS-Authentifizierung.
Anschließend wird in Abschnitt 6.2 das MQTT-Protokoll vorgestellt und Konzepte für den MQTT-Broker vorgestellt, die eine Authentifizierung und Autorisierung von IoT-Geräten ermöglichen. Die Authentifizierung umfasst die Methodiken zur Verwendung einer Client-ID, X.509-Zertifikaten in einer PKI und die Validierung von Zertifikaten mittels Sperrlisten oder OCSP. Für die Autorisierung werden Zugriffskontrolllisten, rollen-basierte Kontrolle und OAuth2 vorgestellt.
Im Abschnitt 6.3 werden die Möglichkeiten für eine sichere Implementierung von Transportprotokollen vorgestellt. Diese umfassen TLS mit TCP, CoAP mit UDP, DTLS als Erweiterung für UDP mit TLS sowie AugMQTT mit dem AugPAKE-Protokoll als Erweiterung für MQTT.
Abschnitt 6.4 erläutert einige Konfigurationsmöglichkeiten für IoT-Geräte. Der Fokus liegt dabei vor allem auf sicheren Zugangsdaten und der Verwendung von Public-Keys.

7 Anwendung des Sicherheitskonzeptes auf die MBP

Aus den im vorangegangen Kapitel vorgestellten Bedrohungen und Angriffsszenarien werden nun konkrete Umsetzungen aus dem Sicherheitsframework für die MBP vorgestellt. Zunächst erfolgt eine IST-Zustandsanalyse der MBP, um den aktuellen Stand der Sicherheitsfeatures festzuhalten. Dieser Stand datiert vom August 2019.

7.1 IST-Zustand MBP

Auf dem aktuellen Entwicklungsstand bietet die MBP auf der Anwendungsebene einen unverschlüsselten Passwortlogin sowie eine "Basic HTTP Authentifizierung" für die API-Endpunkte. Die Authentifizierung bietet allerdings keinerlei Verschlüsselung, sondern nur einen Base64-kodierten Authorization-Header mit einer Benutzernamen und Passwortkombination, die im öffentlichen Repository dokumentiert sind. Weiter werden sämtliche Zugangsdaten (SSH-Keys, Zugangsdaten) unverschlüsselt (teilweise hartkodiert) abgespeichert. Auf der Middlewarebene kann nach aktuellem Stand jeder Client auf alle MQTT-Topics subscriben und auch eigene Daten publishen. Dazu muss nur der Topic-Name bekannt sein. Die Daten von Sensoren werden unverschlüsselt zwischen Broker und IoT-Gerät gesendet und auch unverschlüsselt in der Datenbank abgespeichert. Die Verbindung zur Datenbank ist mit einem Standardbenutzer eingerichtet und unverschlüsselt. Die Verwendung von MQTT und HTTP auf der Transportebene erfolgt ohne den Einsatz von TLS, ist also unverschlüsselt. Beim Einrichten eines neuen Gerätes über die Weboberfläche der MBP muss ein SSH-Key übergeben werden. Damit kann sich das Backend der MBP mit dem IoT-Gerät verbinden und Skripte deployen. Eine Überprüfung der Skripte findet nicht statt. Außerdem ist ein Login mit Benutzername und Passwort auf dem IoT-Gerät weiter möglich. Der MQTT-Broker ist aktuell ungeschützt, es findet keine Authentifizierung oder Autorisierung der IoT-Geräte statt. Der Zugriff auf die Datenbank ist mit einem Passwort gesichert.

7.1.1 Analyse mit OWASP

Für eine automatisierte Analyse der Webapplikation der MBP wird das OWASP-Zed-Attack-Proxy-Tool (ZAP) verwendet. Mit dieser Software können automatisiert Sicherheitslücken in Webanwendungen identifiziert werden. Für den Scan wurde die MBP entsprechend der Installationsanleitung auf Github deployed. Das Ergebnis des Scans zeigt insgesamt vier Warnungen. ZAP unterteilt die Verwundbarkeiten einer Anwendung in vier Risikolevel, von "Hoch" bis "Information". Für die MBP gibt es eine Warnung mit mittlerem und drei Warnungen mit geringem Risikolevel. Die schwache Authentifizierungsmethode für die API mittels "Basic HTTP Authentifizierung" erhält den mittleren Risikolevel. Als geringes Risiko stuft das ZAP-Tool das Fehlen von Anti-CSRF-Tokens, der nicht aktivierten "Web Browser XSS Protection" und das Fehlen des "Content-Type Headers" ein.

Risik Level	**Warning**
Medium	Weak Authentication Method
Low	Absence of Anti-CSRF Tokens
Low	Web Browser XSS Protection Not Enabled
Low	Content-Type Header Missing

Tabelle 7.1: Ergebnis des OWASP-ZAP-Scans.

Als Lösungen für die genannten Schwachstellen nennt das Tool Konzepte, die bereits in Kapitel 5 vorgestellt wurden. Für die Weak-Authentication-Method schläg OWASP den Schutz mit HTTPS oder einer stärkeren Authentifizierungsmethode vor. Für das Fehlen von Anti-CSRF-Token empfiehlt es das Verwenden einer Bibliothek, welche diese Schwachstelle nicht hat. Der fehlende XSS-Schutz soll mit einer Einstellung im Response-Header aktiviert werden. Für die letzte Warnung, den fehlenden Content-Type-Header, empfiehlt das Tool die Überprüfung für jede einzelne Seite, ob der Content-Type für den angeforderten Inhalt auch passend ist.

7.2 Authentifizierung & Autorisierung mit OAuth2

Da die MBP in der vorliegenden Version keine Authentifizierung von IoT-Geräten unterstützt und auch der Mosquitto-Broker lediglich mit einer Benutzername und Passwortkombination geschützt ist, wird der folgende Authentifizierungsprozess für eine sichere Authentifizierung sowie Autorisierung der an die MBP angeschlossenen Geräte implementiert. Die zusätzliche Verwendung von HTTPS ist Grundvoraussetzung für eine verschlüsselte Nachrichtenübertragung. Wird kein HTTPS verwendet, können die Tokens von Dritten mitgelesen und abgefangen werden. Die MBP verwendet Mosquitto als MQTT-Broker. Standardmäßig bietet dieser keine Möglichkeit zu einer anderen Authentifizierung als Benutzernamen und Passwörter. Mit einem Auth-Plugin können aber andere Verfahren nachgerüstet werden. Für die Implementierung von OAuth2 wird das "Mosquitto Go Auth" Plugin [Góm19] verwendet. Dieses muss aber zunächst gemeinsam mit Mosquitto kompiliert werden.

Anschließend wird in der Konfigurationsdatei von Mosquitto das Plugin für die Benutzung des JWT-Backends eingestellt. Für OAuth2 mit JWT müssen die IP-Adresse, der TCP sowie die URI für den Autorisierungsserver angegeben werden. An diese Adresse wird der Broker dann das entsprechende Token senden. Wenn der zurückgegebene Status 200 entspricht, ist der Client autorisiert und darf auf das entsprechende Topic publishen oder subscriben. Außerdem kann in der Konfigurationsdatei die Authentifizierung mit Benutzername und Passwort deaktiviert werden. Das JWT mit dem Access-Token wird dann anstelle des Benutzernamens in der MQTT-Nachricht mitgegeben. Als weitere Sicherheitseinstellung wird Mosquitto so konfiguriert, dass nur der Ersteller eines Topics auf dieses publishen kann. Andere Clients können auf dieses Topic nur subscriben.

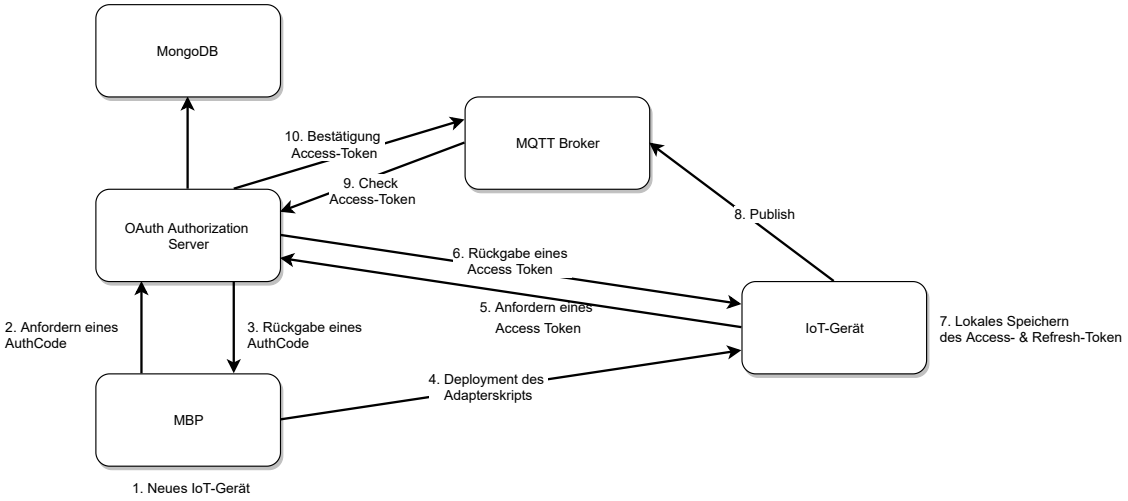


Abbildung 7.1: Architektur der Implementierung von OAuth2 zur Authentifizierung von IoT-Geräten.

Nachdem der Benutzer sich in die MBP eingeloggt hat, kann er ein neues IoT-Gerät anlegen. Dazu muss zunächst ein Adapter erstellt werden, welcher als vorkonfigurierten Parameter einen Autorisierungscode verlangt. Wenn der Benutzer einen neuen Sensor (vgl. Abbildung 7.1 Schritt 1) anlegt, wird das Feld des Autorisierungscodes automatisch mit einem Code ausgefüllt. Im Hintergrund wird eine Anfrage an den OAuth-Authorization-Server geschickt und der Client (in diesem Fall der Benutzer) mit seinem Benutzernamen und Passwort authentifiziert. Als Antwort schickt der Server einen Autorisierungscode zurück, nachdem überprüft wurde, ob der Benutzer die entsprechenden Rechte zum Anlegen neuer Geräte hat. Danach wird das Adapter-Skript auf das Gerät deployed (Schritt 4 in Abbildung 7.1) und der Autorisierungscode als Parameter an das Pythonskript mit übergeben. Wird das Skript gestartet, muss das IoT-Gerät zunächst beim Authorization-Server ein Access-Token beantragen, indem es den Autorisierungscode an den entsprechenden OAuth-Endpunkt schickt. Das IoT-Gerät erhält dann in Schritt 5 ein Access-Token sowie ein Refresh-Token zurück. Da das Access-Token nur eine begrenzte Gültigkeit hat (voreingestellt sind zehn Minuten), muss nach Ablauf dieser Zeit ein neues Token beim Autorisierungsserver angefragt werden. Dabei wird das Refresh-Token als Secret mitgeschickt, um zu beweisen,

dass sich das IoT-Gerät bereits authentifiziert hat. Im Anschluss erhält das IoT-Gerät sowohl ein neues Access-Token als auch ein neues Refresh-Token. Um per MQTT mit dem Broker kommunizieren (Schritt 8) zu können, muss das IoT-Gerät statt dem Benutzernamen das Token mitschicken. Bei der ersten Publish- oder Subscribe-Nachricht checkt der Broker am Autorisierungsserver, ob das Access-Token gültig ist und speichert es in einer lokalen Access-Control-List (ACL). Das IoT-Gerät ist nun vollständig autorisiert und authentifiziert. Bei jeder neuen Nachricht überprüft der Broker das Token mit dem gespeicherten Token in der ACL.

7.3 Datenanonymisierung

Über die MBP kann jeder Benutzer die Daten von angeschlossenen IoT-Geräten und deren Sensoren einsehen oder Kommandos an Aktuatoren senden. Da aber mitunter auch sensible Daten gespeichert werden, wie z.B. Koordinaten von einem GPS-Tracker oder Bilder einer Kamera, soll die MBP die Möglichkeit bieten, bestimmte Daten verschleiern zu können. Das Ziel ist, dass die Daten nicht mehr eindeutig zu bestimmten Personen oder Umgebungen zuordenbar sind.

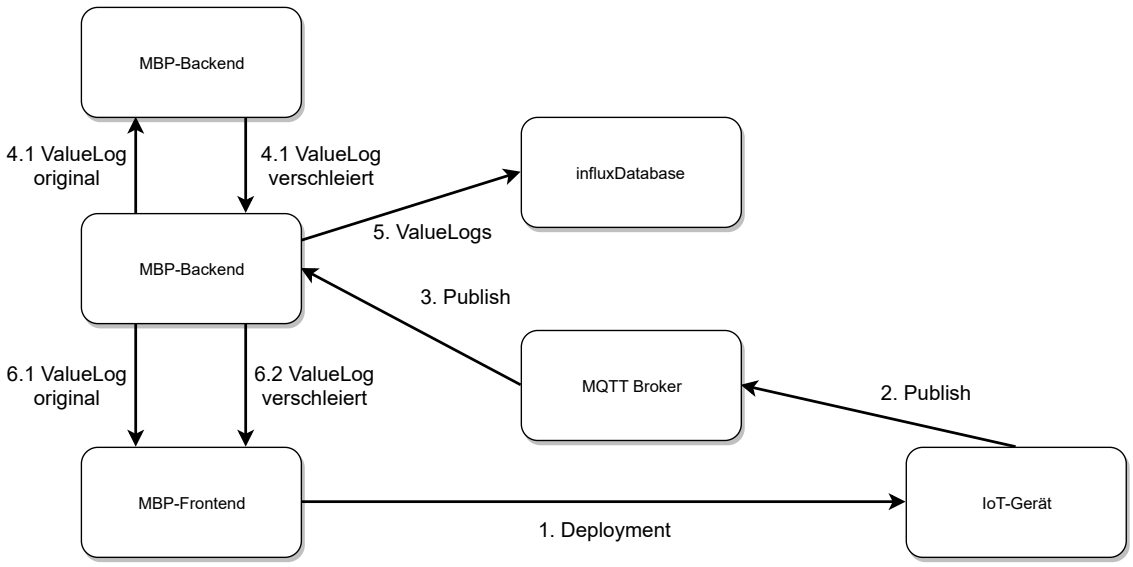


Abbildung 7.2: Architektur der Implementierung zur Anonymisierung von Daten der IoT-Geräte.

Dazu wird wie in Abschnitt 7.2 ein weiterer vorkonfigurierter Parameter für Adapter eingeführt. Dieser besteht aus einem einfachen Schalter mit dem Namen “noisy_data”. Legt ein Benutzer einen neuen Sensor an, kann er vor dem Deployment des Adapterskripts diesen Schalter auswählen. Nach dem Deployment des Skripts beginnt das IoT-Gerät damit Daten auf ein Topic zu publishen, mit einem MQTT-Nachrichtenpayload, der um einen JSON-Eintrag erweitert (”$noisy\_data$” : $true$) wurde. In diesem Topic befinden sich die

originalen Daten. Anschließend werden die Daten vom Broker ins Backend gesendet. Ist der JSON-Eintrag “noisy_data” vorhanden und TRUE, so werden zwei Einträge in der InfluxDB erstellt. Ein Eintrag mit den originalen Daten, der andere Eintrag mit verschleierten Daten. Diese Daten werden in einer extra Komponente verarbeitet. Je nach Datentyp (es werden zunächst nur GPS- und Bilddaten unterstützt) werden die Daten verschleiert. Bei GPS-Daten wird innerhalb eines definierten Radius (beispielsweise 50 Meter) um die originalen Koordinaten ein zufälliger Punkt ausgewählt. Die Koordinaten dieses Punktes werden anstelle der originalen Daten in der influxDB gespeichert. Im Falle einer Bilddatei wird mit dem Algorithmus für das Gaußsche Rauschen das Bild verrauscht und dadurch unkenntlicher. Wenn der Benutzer in der MBP auch der Eigentümer eines Sensors ist, werden in der Ansicht für die historischen Werte die originalen Daten angezeigt. Ist der Benutzer allerdings nicht der Eigentümer, werden ihm die verrauschten Daten angezeigt.

8 Zusammenfassung & Ausblick

Das Ziel dieser Masterarbeit ist es, ein Framework zu schaffen, welches als Leitfaden benutzt werden soll, um die Sicherheit in IoT-Plattformen durch die Implementierung der Konzepte zu erhöhen. Dafür wird zu Beginn dieser Arbeit eine grundlegende Architektur von IoT-Plattformen festgelegt. Im nächsten Schritt werden Schwachstellen in dieser Architektur identifiziert sowie mögliche Angriffe beschrieben. Anschließend werden für die einzelnen Ebenen einer IoT-Plattform Gegenmaßnahmen vorgestellt. Dabei werden unterschiedliche Konzepte und Ansätze vorgestellt und verschiedene Technologien miteinander verglichen. Im letzten Schritt wird dann die Anwendung ausgewählter Konzepte auf die MBP vorgestellt. Dabei geht es um die beispielhafte Implementierung des OAuth2-Frameworks für die IoT-Plattform MBP sowie deren IoT-Geräte und die Umsetzung einer Komponente zur Erhöhung der Privatheit von Daten. Die Umsetzung von Sicherheitsmechanismen für IoT-Plattformen ist für die Hersteller ein kostenintensiver Prozess und wird dadurch immer noch häufig vernachlässigt, denn dabei müssen sowohl klassische Sicherheitseinstellungen für Webanwendungen, als auch für IoT-Geräte und Middlewarekomponenten vorgenommen werden. Das erfordert Wissen auf vielen Ebenen einer Anwendung. Aus Komfortgründen werden die meisten Benutzer keine sicherheitsrelevanten Einstellungen vornehmen, wenn sie nicht dazu aufgefordert werden und diese nicht einfach umzusetzen sind. Ein erster Schritt in die richtige Richtung ist dabei das Verbot von Standardzugangsdaten, wie es zum Beispiel Kalifornien vormacht [Köv18]. Ein Problem sind allerdings bestehende Systeme, da für diese IoT-Umgebungen Sicherheitsmechanismen nicht vorgesehen und nur sehr schwer nachrüstbar sind. Dafür muss es Vorgaben geben, die einen verpflichtenden Einsatz von Technologien aus diesem Konzept, angepasst für den Einsatzbereich, vorgeben, um die Sicherheit von IoT-Plattformen zu erhöhen. Bezüglich der MBP soll in Zukunft ein Rollenkonzept entwickelt werden, um einen geeigneten Authentifizierungs- und Autorisierungsworkflow mit unterschiedlichen Rechten der Benutzer zu ermöglichen. Weiter HTTPS implementiert werden, um Nachrichten verschlüsselt übertragen zu können. Wichtig ist dies auch mit Blick auf die Geräteauthentifizierung mit OAuth2, da die Access-Tokens ansonsten im Klartext lesbar sind für andere Geräte oder Benutzer im Netzwerk. Ein geeignetes Updatemanagement für Geräte, die in der MBP registriert sind, ist für wichtige Sicherheitsupdates essentiell und sollte eingeführt werden. Außerdem muss die Registrierung sowie der Login für Benutzer überarbeitet und abgesichert werden. Dazu gehört die verpflichtende Angabe eine Mailadresse, um einen externen Authentifizierungspunkt für einen Benutzer zu schaffen, und die Umsetzung einer Zwei-Faktor-Authentifizierung. Ein weiterer Punkt ist die sichere Verwahrung von Zugangsdaten und SSH-Keys für IoT-Geräte, diese sollten beispielsweise in einem Spring-Vault gespeichert werden.

Literaturverzeichnis

[AB17] E. Ancillotti, R. Bruno. „Comparison of CoAP and CoCoA+ congestion control mechanisms for different IoT application scenarios“. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. Juli 2017, S. 1186–1192. DOI: 10.1109/ISCC.2017.8024686 (zitiert auf S. 53).

[AGM+15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash. „Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications“. In: *IEEE Communications Surveys Tutorials* 17.4 (Fourthquarter 2015), S. 2347–2376. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2444095 (zitiert auf S. 23, 29).

[Ama19] Amazon Web Services. *IoT Device Defender*. 2019. URL: https://aws.amazon.com/de/iot-device-defender/ (zitiert auf S. 28).

[BBD+13] M. Bauer, N. Bui, J. De Loof, C. Magerkurth, A. Nettsträter, J. Stefa, J. W. Walewski. „IoT Reference Model“. In: *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Hrsg. von A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, S. Meissner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 113–162. ISBN: 978-3-642-40403-0. DOI: 10.1007/978-3-642-40403-0_7. URL: https://doi.org/10.1007/978-3-642-40403-0_7 (zitiert auf S. 19).

[BCS12] C. Bormann, A. P. Castellani, Z. Shelby. „CoAP: An Application Protocol for Billions of Tiny Internet Nodes“. In: *IEEE Internet Computing* 16.2 (März 2012), S. 62–67. ISSN: 1089-7801. DOI: 10.1109/MIC.2012.29 (zitiert auf S. 53).

[BEK14] C. Bormann, M. Ersue, A. Keränen. *Terminology for Constrained-Node Networks*. RFC 7228. Mai 2014. DOI: 10.17487/RFC7228. URL: https://rfc-editor.org/rfc/rfc7228.txt (zitiert auf S. 56).

[BH13] C. Bormann, P. E. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 7049. Okt. 2013. DOI: 10.17487/RFC7049. URL: https://rfc-editor.org/rfc/rfc7049.txt (zitiert auf S. 26).

[Bra14] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. März 2014. DOI: 10.17487/RFC7159. URL: https://rfc-editor.org/rfc/rfc7159.txt (zitiert auf S. 26).

[BSP+08] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, D. Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. Mai 2008. DOI: 10.17487/RFC5280. URL: https://rfc-editor.org/rfc/rfc5280.txt (zitiert auf S. 51).

[CBSL19] B. Campbell, J. Bradley, N. Sakimura, T. Lodderstedt. *OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens*. Internet-Draft draft-ietf-oauth-mtls-17. Work in Progress. Internet Engineering Task Force, Aug. 2019. 32 S. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-mtls-17 (zitiert auf S. 47).

[CPG+15] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, G. Ferrari. „IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios“. In: *IEEE Sensors Journal* 15.2 (Feb. 2015), S. 1224–1234. ISSN: 1530-437X. DOI: 10.1109/JSEN.2014.2361406 (zitiert auf S. 25).

[DBN+11] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, J. F. D. Jr. *Advanced encryption standard (AES)*. 2011. DOI: 10.6028/nist.fips.197 (zitiert auf S. 49).

[Fie00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Diss. Irvine: University of California, 2000 (zitiert auf S. 43).

[GBF+16] J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann, L. Reinfurt. „Comparison of IoT platform architectures: A field study based on a reference architecture“. In: *2016 Cloudification of the Internet of Things (CIoT)*. Nov. 2016, S. 1–6. DOI: 10.1109/CIOT.2016.7872918 (zitiert auf S. 16, 29, 30).

[GFN+17] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, M. F. Theofanos. *Digital identity guidelines: authentication and lifecycle management*. Techn. Ber. Juni 2017. DOI: 10.6028/nist.sp.800-63b (zitiert auf S. 41, 57).

[GKS19] O. Garcia-Morchon, S. Kumar, M. Sethi. *Internet of Things (IoT) Security: State of the Art and Challenges*. RFC 8576. Apr. 2019. DOI: 10.17487/RFC8576. URL: https://rfc-editor.org/rfc/rfc8576.txt (zitiert auf S. 55, 56).

[Góm19] I. Gómez. 2019. URL: https://github.com/iegomez/mosquitto-go-auth (zitiert auf S. 60).

[Gre14] J. Green. *IoT Reference Model*. 2014. URL: http://cdn.iotwf.com/resources/72/IoT_Reference_Model_04_June_2014.pdf (besucht am 29. 09. 2019) (zitiert auf S. 19).

[Har12] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Okt. 2012. DOI: 10.17487/RFC6749. URL: https://rfc-editor.org/rfc/rfc6749.txt (zitiert auf S. 44, 52).

[HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland. „Automating the Provisioning and Configuration of Devices in the Internet of Things“. Englisch. In: *Complex Systems Informatics and Modeling Quarterly* 9 (Dezember 2016), S. 28–43. ISSN: 2255 - 9922. DOI: 10.7250/csimq.2016-9.02. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2016-23&engl=0 (zitiert auf S. 17).

[Hun19] T. Hunt. *Have I Been Pwned (HIBP)*. 2019. URL: https://haveibeenpwned.com (zitiert auf S. 57).

[HWBM16a] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Automated Sensor Registration, Binding and Sensor Data Provisioning“. Englisch. In: *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016)*. Bd. 1612. CEUR Workshop Proceedings. Ljubljana, Slovenia: CEUR-WS.org, Juni 2016, S. 81–88. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2016-22&engl=0 (zitiert auf S. 18).

[HWBM16b] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Dynamic Ontology-based Sensor Binding“. Englisch. In: *Advances in Databases and Information Systems. 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28-31, 2016, Proceedings*. Bd. 9809. Information Systems and Applications, incl. Internet/Web, and HCI. Prague, Czech Republic: Springer International Publishing, Aug. 2016, S. 323–337. ISBN: 978-3-319-44039-2. DOI: 10.1007/978-3-319-44039-2. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2016-25&engl=0 (zitiert auf S. 18).

[Ich18] J. Ichnowski. *OWASP Java Encoder Project*. OWASP, 2018. URL: https://www.owasp.org/index.php/OWASP_Java_Encoder_Project (zitiert auf S. 43).

[ISO12] ISO29192. *Information technology — Security techniques — Lightweight cryptography — Part 2: Block ciphers*. Standard. International Organization for Standardization, 2012 (zitiert auf S. 49).

[JBS15a] M. Jones, J. Bradley, N. Sakimura. *JSON Web Signature (JWS)*. RFC 7515. Mai 2015. DOI: 10.17487/RFC7515. URL: https://rfc-editor.org/rfc/rfc7515.txt (zitiert auf S. 46).

[JBS15b] M. Jones, J. Bradley, N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. Mai 2015. DOI: 10.17487/RFC7519. URL: https://rfc-editor.org/rfc/rfc7519.txt (zitiert auf S. 44, 46).

[JBT16] M. Jones, J. Bradley, H. Tschofenig. *Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)*. RFC 7800. Apr. 2016. DOI: 10.17487/RFC7800. URL: https://rfc-editor.org/rfc/rfc7800.txt (zitiert auf S. 48).

[JCBD18] M. Jones, B. Campbell, J. Bradley, W. Denniss. *OAuth 2.0 Token Binding*. Internet-Draft draft-ietf-oauth-token-binding-08. Work in Progress. Internet Engineering Task Force, Okt. 2018. 30 S. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-token-binding-08 (zitiert auf S. 47).

[JH12] M. Jones, D. Hardt. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC 6750. Okt. 2012. DOI: 10.17487/RFC6750. URL: https://rfc-editor.org/rfc/rfc6750.txt (zitiert auf S. 46).

[JH15] M. Jones, J. Hildebrand. *JSON Web Encryption (JWE)*. RFC 7516. Mai 2015. DOI: 10.17487/RFC7516. URL: https://rfc-editor.org/rfc/rfc7516.txt (zitiert auf S. 46).

[Jon15] M. Jones. *JSON Web Algorithms (JWA)*. RFC 7518. Mai 2015. DOI: 10.17487/RFC7518. URL: https://rfc-editor.org/rfc/rfc7518.txt (zitiert auf S. 47).

[KML+19] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, K. Kajimoto. *Web of Things (WoT) Architecture*. W3C, 2019. URL: https://www.w3.org/TR/2019/CR-wot-architecture-20191106/ (zitiert auf S. 27).

[Köv18] C. Köver. *Internet der Dinge: Kalifornien verbietet Standardpasswörter – ein Modell für Deutschland?* 9. Okt. 2018. URL: https://netzpolitik.org/2018/internet-der-dinge-kalifornien-verbietet-standardpasswoerter-ein-modell-fuer-deutschland/ (zitiert auf S. 57, 65).

[Kum19a] M. Kumar. *Over 100 Million JustDial Users' Personal Data Found Exposed On the Internet*. https://thehackernews.com/2019/04/justdial-hacked-data-breach.html. 17. Apr. 2019. (Besucht am 01. 09. 2019) (zitiert auf S. 38).

[Kum19b] M. Kumar. *This Flaw Could Have Allowed Hackers to Hack Any Instagram Account Within 10 Minutes*. https://thehackernews.com/2019/07/hack-instagram-accounts.html. 15. Juli 2019. (Besucht am 01. 09. 2019) (zitiert auf S. 37).

[LY06] C. M. Lonvick, T. Ylonen. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. Jan. 2006. DOI: 10.17487/RFC4252. URL: https://rfc-editor.org/rfc/rfc4252.txt (zitiert auf S. 57).

[Mic09] Microsoft. 2009. URL: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20) (zitiert auf S. 31, 32).

[MYAZ15] R. Mahmoud, T. Yousuf, F. Aloul, I. Zualkernan. „Internet of things (IoT) security: Current status, challenges and prospective measures“. In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. Dez. 2015, S. 336–341. DOI: 10.1109/ICITST.2015.7412116 (zitiert auf S. 15, 22, 23, 29).

[NAYL16] M. Nawir, A. Amir, N. Yaakob, O. B. Lynn. „Internet of Things (IoT): Taxonomy of security attacks“. In: *2016 3rd International Conference on Electronic Design (ICED)*. Aug. 2016, S. 321–326. DOI: 10.1109/ICED.2016.7804660 (zitiert auf S. 19).

[NIS99] NIST. *DATA ENCRYPTION STANDARD (DES)*. National Institute of Standards und Technology, 1999. URL: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf (zitiert auf S. 37).

[OWA18] OWASP. *OWASP Internet of Things (IoT) Project*. 2018. URL: https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project (besucht am 29. 09. 2019) (zitiert auf S. 33).

[Pos80] J. Postel. *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://rfc-editor.org/rfc/rfc768.txt (zitiert auf S. 53).

[Pos81] J. Postel. *Transmission Control Protocol*. RFC 793. Sep. 1981. DOI: 10.17487/RFC0793. URL: https://rfc-editor.org/rfc/rfc793.txt (zitiert auf S. 53).

[RD08] E. Rescorla, T. Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: 10.17487/RFC5246. URL: https://rfc-editor.org/rfc/rfc5246.txt (zitiert auf S. 49).

[Red19] RedHat. *Bean Validation*. 2019. URL: https://beanvalidation.org/ (zitiert auf S. 44).

[Res99] E. Rescorla. *Diffie-Hellman Key Agreement Method*. RFC 2631. Juni 1999. DOI: 10.17487/RFC2631. URL: https://rfc-editor.org/rfc/rfc2631.txt (zitiert auf S. 55).

[RM12] E. Rescorla, N. Modadugu. *Datagram Transport Layer Security Version 1.2*. RFC 6347. Jan. 2012. DOI: 10.17487/RFC6347. URL: https://rfc-editor.org/rfc/rfc6347.txt (zitiert auf S. 53).

[RM19] E. Reshetova, M. McCool. *Web of Things (WoT) Security and Privacy Guidelines*. 2019. URL: https://www.w3.org/TR/wot-security/ (zitiert auf S. 27).

[SBA15] N. Sakimura, J. Bradley, N. Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636. Sep. 2015. DOI: 10.17487/RFC7636. URL: https://rfc-editor.org/rfc/rfc7636.txt (zitiert auf S. 44).

[Sch16] F. A. Scherschel. *Security-Journalist Brian Krebs war Ziel eines massiven DDoS-Angriffs*. https://www.heise.de/security/meldung/Security-Journalist-Brian-Krebs-war-Ziel-eines-massiven-DDoS-Angriffs-3329988.html, 23. Sep. 2016. (Besucht am 01. 09. 2019) (zitiert auf S. 37).

[Sch17] J. Schaad. *CBOR Object Signing and Encryption (COSE)*. RFC 8152. Juli 2017. DOI: 10.17487/RFC8152. URL: https://rfc-editor.org/rfc/rfc8152.txt (zitiert auf S. 26).

[Sch19] F. Scherschel. *Angriffe auf Elasticsearch: Linux-Server werden zu DDoS-Schleudern*. heise, 24. Juli 2019 (zitiert auf S. 38).

[SHB14] Z. Shelby, K. Hartke, C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. Juni 2014. DOI: 10.17487/RFC7252. URL: https://rfc-editor.org/rfc/rfc7252.txt (zitiert auf S. 53).

[SHPM18] A. C. F. da Silva, P. Hirmer, R. K. Peres, B. Mitschang. „An Approach for CEP Query Shipping to Support Distributed IoT Environments“. Englisch. In: *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, Oktober 2018, S. 247–252. ISBN: 978-1-5386-3227-7. DOI: 10.1109/PERCOMW.2018.8480241. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2018-34&engl= (zitiert auf S. 18).

[SK18] S. Shin, K. Kobara. *Augmented Password-Authenticated Key Exchange (AugPAKE)*. Internet-Draft draft-irtf-cfrg-augpake-09. Work in Progress. Internet Engineering Task Force, Jan. 2018. 20 S. URL: https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-augpake-09 (zitiert auf S. 54).

[SKCH16] S. Shin, K. Kobara, C.-C. Chuang, W. Huang. „A security framework for MQTT“. In: *2016 IEEE Conference on Communications and Network Security (CNS)*. Okt. 2016, S. 432–436. DOI: 10.1109/CNS.2016.7860532 (zitiert auf S. 54).

[SMA+13] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, D. C. Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC 6960. Juni 2013. DOI: 10.17487/RFC6960. URL: https://rfc-editor.org/rfc/rfc6960.txt (zitiert auf S. 51).

[SPC+17] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, G. Bianchi. „OAuth-IoT: An access control framework for the Internet of Things based on open standards“. In: Juli 2017. DOI: 10.1109/ISCC.2017.8024606 (zitiert auf S. 24).

[SS04] F. Swiderski, W. Snyder. *Threat Modeling*. O'Reilly Media, 2004. ISBN: 9780735637696. URL: https://books.google.de/books?id=qWjoUuFSmf8C (zitiert auf S. 31).

[SSW+19] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, H. Tschofenig. *Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)*. Internet-Draft draft-ietf-ace-oauth-authz-24. Work in Progress. Internet Engineering Task Force, März 2019. 83 S. URL: https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-24 (zitiert auf S. 25).

[ST13] A. Stanford-Clark, H. L. Truong. *MQTT For Sensor Networks (MQTT-SN) Protocol Specification*. 14. Nov. 2013. URL: https://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf (zitiert auf S. 54).

[Whi19] Z. Whittaker. *Security lapse exposed a Chinese smart city surveillance system*. https://techcrunch.com/2019/05/03/china-smart-city-exposed/. 3. Mai 2019. (Besucht am 01. 09. 2019) (zitiert auf S. 38).

[WLL+10] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, H. Du. „Research on the architecture of Internet of Things“. In: Bd. 5. Sep. 2010, S. V5–484. DOI: 10.1109/ICACTE.2010.5579493 (zitiert auf S. 29, 30).

[Wor19] WorldWide Web Consortium (W3C). *Web of Things*. 2019. URL: https://www.w3.org/WoT/ (zitiert auf S. 26).

[Yer03] F. Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC 3629. Nov. 2003. DOI: 10.17487/RFC3629. URL: https://rfc-editor.org/rfc/rfc3629.txt (zitiert auf S. 50).

[Zim80] H. Zimmermann. „OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection“. In: *IEEE Transactions on Communications* 28.4 (Apr. 1980), S. 425–432. DOI: 10.1109/TCOM.1980.1094702 (zitiert auf S. 15, 19).

Alle URLs wurden zuletzt am 10. Dezember 2019 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift