OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

INFORMATIK UND MATHEMATIK

PENTASYS
Unser Maßstab ist der Mensch

Bachelorarbeit

Konzeption einer sicheren Over-The-Air Update Architektur für vernetzte Fahrzeuge

Eingereicht von: Felix Almesberger
Matrikelnummer: 3061514
Studiengang: Wirtschaftsinformatik
OTH Regensburg

Betreut durch: Prof. Dr. Rudolf Hackenberg
OTH Regensburg
Steffen Renz
PENTASYS AG

Inhaltsverzeichnis

Inhaltsverzeichnis 2
Einleitung 4
Die Evolution des Internets 4
Das vernetzte Fahrzeug 4
Bedeutung von Updates 6
Motivation 6
Aufbau der Arbeit 7
Einführung 8
Grundlagen der Kryptographie 8
Transport Layer Security 11
IT Sicherheit 13
Weitere Begriffserklärungen 16
Analyse & Anforderungen 16
Marktanalyse 16
Formulierung eigener Anforderungen 20
Architektur 25
Überblick 26
Komponenten 26
Geräteverwaltung 27
Datenstruktur 27
Representational State Transfer 29
Bereitstellungsserver 35
Kommunikatonsdienst 36
Kommunikation 37
Überblick 37
Zertifikatsverwaltung 37
Publish / Subscriber Prinzip 38
Kommandos 41
Benachrichtigungen 43
Kompression von Updates 44
Übertragung von Updates 46
Fahrzeug 49
Aufbau eines vernetzten Autos 49
Virtualisierung 51
Updatevorgang 52
Architektur 56
Blue Green Deployment 58
Services / Feature Updates 58

Betriebssystem Updates ... 59
Zertifikatsspeicher ... 60
Prototyp ... 61
Umfang der Implementierung ... 61
Überblick ... 62
Hardwareaufbau ... 62
Sicherheit in der Softwareentwicklung ... 63
Wahl der Entwicklungstechnologie ... 65
Geräteverwaltung ... 66
REST API ... 69
Bereitstellung ... 72
MQTT Kommunikation ... 73
Update Pakete ... 73
Notification Service & Status Receiver Service ... 74
Zertifikate ... 75
Car Service ... 76
Oberfläche ... 77
Command Line Interface ... 77
Beispielhafter Updatevorgang ... 78
Softwareprojekt ... **Fehler! Textmarke nicht definiert.**
Fazit ... 80
Die Verwendung von Go ... 80
Die Implementierung ... 81
Danksagung ... 82
Ehrenwörtliche Erklärung ... 83
Literatur & Abbildungen ... 84
Literaturverzeichnis ... 84
Abbildungsverzeichnis ... 88
Quellen der Abbildungen ... 89

Einleitung

Die Evolution des Internets

Die Ursprünge des Internets reichen in die 1960er Jahre zurück, als das US-Amerikanische Verteidigungsministerium nach Möglichkeiten suchte militärische Einrichtungen zuverlässig zu verbinden. Für Endbenutzer blieb diese Entwicklung lange Zeit uninteressant bis Tim Berners Lee 1993 am CERN das *WorldWideWeb* entwickelte. Diesen ermöglichten Benutzern eine graphische Benutzeroberfläche für die Benutzung von Diensten im Internet. Als bald darauf begonnen wurde Privathaushalte großflächig an das Netz anzuschließen setzte die sogenannte *Internet Revolution* ein. Vollkommen neue Geschäftsmodelle wurden auf Grundlage des Internets geschaffen. (Dr. Markus Siepermann) Diese Entwicklung führte dazu, dass sechs der zehn wertvollsten Unternehmen (nach Börsenwert gemessen) ihren Umsatz hauptsächlich durch das Internet erwirtschaften. (Hannah Steinharter)

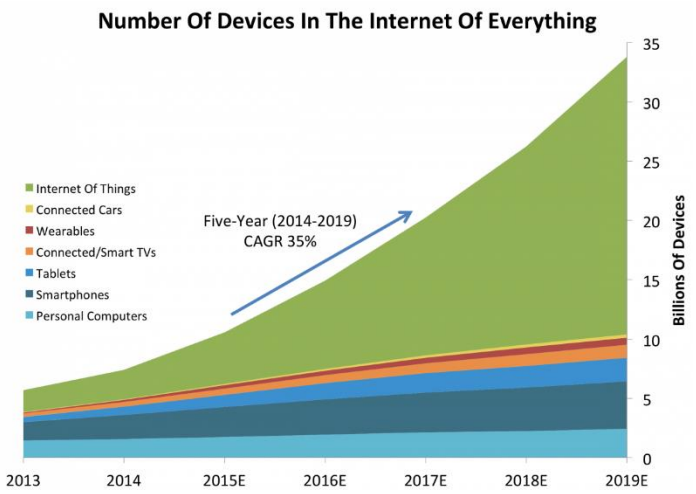

Abbildung 1 Geräte des Internets

Doch diese Technische Entwicklung ist noch nicht abgeschlossen: Durch den flächendeckend ausgebauten Mobilfunk, dem niedrigen Preis für Kommunikationstechnik und dessen Miniaturisierung, breitet sich das Internet immer weiter aus. Es werden immer mehr Gegenstände und Maschinen miteinander vernetzt und bilden das sogenannte *Internet of Things* (Internet der Dinge).

Dadurch können diese Gegenstände selbstständig verschiedene Aufgaben für den Benutzer erledigen. Der Anwendungsbereich erstreckt sich dabei von einer selbstständigen Informationsversorgung über automatische Bestellungen bis hin zu warn und

Notfallfunktionen. Durch die Verbindung mit dem Internet entstehen aber auch Sicherheitsrisiken, die beachtet werden müssen. (Dr. Markus Siepermann) Es kann damit gerechnet werden, dass das Internet der Dinge wird Alltagsleben in vielen Bereichen vereinfachen wird.

Das vernetzte Fahrzeug

Bei einem vernetzten Fahrzeug werden die Konzepte des Internets der Dinge für Fahrzeuge adaptiert. Dafür wird eine Kommunikationseinheit installiert, mit der das Auto (1) mit dem Internet und (2) mit anderen Fahrzeugen kommunizieren kann. Über das Internet können in Echtzeit Daten mit einer Cloud-Plattform ausgetauscht werden. Dadurch werden Benutzerfreundliche Services wie Echtzeitortung oder das Entsperren des Fahrzeugs über das Mobiltelefon möglich.

Für ein komplett autonomes Fahrzeug ist ein permanenter Austausch von Sensordaten zwischen vernetzten Verkehrsteilnehmer in der Umgebung notwendig. Auch der Benutzerservice kann verbessert werden. Tritt ein Fehler am Fahrzeug auf, werden die Diagnosedatendirekt über das Internet an den Hersteller gesendet. Auf einen ärgerlichen und zeitintensiven Besuch in der Servicewerkstatt kann dann mitunter verzichtet werden. Auch Infotainment Systeme gewinnen stetig an Features und Bedeutung. Durch eine dauerhafte Verbindung mit dem Internet lassen sich aktuell Daten, zum Beispiel Karten, oder Stauinformationen anzeigen. Wie bei einem Mobiltelefon wäre es dann auch denkbar Funktionen nachzurüsten.

Die Voraussetzungen dafür sind Großteils bereits vorhanden. Ab dem 01. April 2019 muss jedes in der Europäischen Union verkauftes Auto über eine eSIM Karte verfügen. Darüber sollen Rettungskräfte im Falle eines Unfalls gerufen werden. Über diese SIM Karte lässt sich auch eine Internetverbindung aufbauen.

Durch die Anbindung entstehen aber auch bisher unbekannte Risiken: Autos waren bis jetzt in sich geschlossene Systeme. Ein Angriff auf dessen Infrastruktur erforderte eine physische Anwesenheit. Durch die Vernetzung sind diese prinzipiell auch aus dem Internet angreifbar.

Bedeutung von Updates

Fehlerfreie Software zu entwickeln, die eine gewisse Komplexität überschreitet ist weder theoretisch noch praktisch möglich. Das sorgfältige Testen durch Modul- und Integrationstests mag die Softwarequalität erhöhen, ein Zitat des bekannten niederländische Informatiker *Edsger Djikstra* bringt das Problem jedoch auf den Punkt:

> » Durch Testen kann man stets nur die Anwesenheit, nie aber die Abwesenheit von Fehlern beweisen. « - (Edsger W. Dijkstra)

Studien zeigen, dass pro 1000 Lines of Code (*LoC*) im Industriestandard etwa 15-50 Fehler auftreten. Selbst bei auf dem Markt befindlicher Software von Microsoft, schätzt man die Fehlerrate immer noch auf 0,5 Fehler pro 1000 *LoC*. Fehler in Softwareprojekten sind keine Ausnahme, sondern die Regel. (McConnell 2009)

Entdeckte Fehler werden durch Aufspielen einer neuen Softwareversion behoben. Dies ist insbesondere bei kritischen Komponenten, zu denen Fahrzeuge gehören, sehr wichtig. Fehler in der Software können zu einer direkten Gefahr für den Straßenverkehr führen. Verstärkt wird dieser Effekt durch die steigende Menge an Software in vernetzten Fahrzeugen und deren Durchdringung in immer sensiblere Bereiche des Fahrzeugs (Beispiel: *Autonomes Fahren*). Auch die Verbindung mit dem Internet stellt eine Gefahrenquelle dar. Immer wieder werden auch in erprobten Komponenten Schwachstellen gefunden, die mit bösartiger Absicht missbraucht werden können. Die Updatemöglichkeit bekommt eine besondere Dringlichkeit, da langlebige Produkte, zu denen Autos definiert gehören, nicht ohne Aufwand umgetauscht oder ersetzt werden können.

Motivation

In den USA, wo Rückrufaktionen sehr gut dokumentiert sind zeigt sich folgender Trend. Die Zahl Rückrufaktionen, steigt seit 2010 deutlich an (von ca. 10 auf über 40 Fälle im Jahr 2017). Experten gehen davon aus, dass dieser Wert weiter steigen wird. Rückrufaktionen sind dabei für die Hersteller äußerst kostspielig, Imageschädigend und beim Verbraucher unbeliebt. 25 % aller Probleme, die zwischen 2006 und 2010 durch Rückrufaktionen behoben werden sollten, zeigt eine Umfrage, wurden bis 2012 immer noch nicht behoben. (ADAC Deutschland)

Einen Ausweg bieten sogenannte Over-The-Air-Updates (*OTA*). Dabei werden die Updates nicht mehr manuell eingespielt, das Auto lädt diese selbstständig über eine Internetverbindung

herunter und aktualisiert sich selbst. Die wenigen Hersteller, die bereits OTA unterstützen nutzen es lediglich Aktualisieren von unkritischen Komponenten (Infotainment, Navigation). Eine Flächendeckende Updatemöglichkeit des gesamten Fahrzeugs inklusive Steuergeräte bietet derzeit nur der Kalifornische Autobauer Tesla an. (Stout Automotive 2018)

Durch Smartphones und anderen mobilen Geräten sind die Benutzer es gewöhnt, Features jederzeit nachrüsten zu können. Auch Automatische Updates werden durch diese Geräte seit Anfang an unterstützt. Das derzeitige Fahrzeug hinkt in dieser Entwicklung hinterher.

Over-The-Air Updates bieten sowohl für Fahrzeughersteller- als auch Besitzer Vorteile. Durch weniger Besuche in der Reparaturwerkstatt steigt der Komfort für die Nutzer. Gleichzeitig sind OTA für die Hersteller eine günstige und zuverlässige Methodik, da Updates nun garantiert und kontrolliert ausgerollt werden können. Die meisten großen Automobilhersteller haben diese Vorteile erkannt und planen derartige Funktionen bald auf den Markt zu bringen.

Aufbau der Arbeit

Im Kern soll die Arbeit eine Einführung in die Architektur eines sicheren Over-The-Air Update Systems bieten und Leser an dieses Thema heranführen. Die Arbeit beginnt mit einer Einführung in die Grundlagen der Verwendeten kryptologischen Verfahren und Begriffserklärungen. Anschließend werden Risiken und Angriffsvektoren gezeigt, die bei Betrieb von IT-Infrastruktur auftreten. Durch eine Marktanalyse, werden bereits erhältliche Over-The-Air Update Lösungen analysiert. Darauf aufbauend werden Anforderungen für ein eigenes Konzept formuliert. Dieses Konzept wird detailliert beschrieben. Für jede Komponente des Systems wird analysiert welche Risiken und Angriffsvektoren des Katalogs auftreten können und wie diese effektiv vermieden werden können. Soweit möglich werden hier, bereits industrieerprobte erprobte Konzepte und Technologien verwendet. Ein besonderes Augenmerk liegt auch bei der Ausfallsicherheit eines solchen Systems. Um das erarbeitete Konzept zu überprüfen erfolgte ein schematischer Hardwareaufbau und Implementierung. Die betreuende Firma PENTASYS AG, war besonders daran interessiert ob sich die Programmiersprache Go der Firma Google für derartige Projekte einsetzen lässt. Daher wurde das Konzept mittels Go analysiert. Abschließend wird ein Fazit über das Konzept und die Implementierung sowie ein Ausblick gegeben.

Einführung

Grundlagen der Kryptographie

Symmetrische Verschlüsselung

Es gibt einen Schlüssel, mit dem Nachrichten sowohl verschlüsselt als auch wieder entschlüsselt werden können. Beide Kommunikationspartner benötigen den gleichen Schlüssel. Problematisch hierbei ist das übertragen des Schlüssels selbst. Dieser kann noch nicht über die verschlüsselte Verbindung übertragen und daher im Zweifelsfall abgegriffen werden. Eine analoge persönliche sichere Übertragung ist für die automatisierte verschlüsselte Kommunikation zwischen Maschinen nicht möglich. (BHATTACHARJEE 2018 // 2017, Symmetrical Encryption)

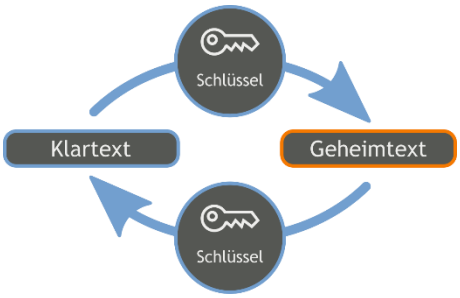

Abbildung 2 Symmetrische Verschlüsselung

Asymmetrische Verschlüsselung

Anstatt eines Schlüssels werden nun zwei mathematisch miteinander verbundener Schlüssel verwendet. Einen öffentlichen und einen privaten Schlüssel. Daten, die mit dem öffentlichen Schlüssel verschlüsselt wurden, können nur mit dem Privaten Schlüssel entschlüsselt werden. Der Vorteil dabei: Ein geheimer Schlüssel muss nicht übertragen werden. Jeder kann mit einem öffentlichen Schlüssel Daten verschlüsseln. Dieser öffentliche Schlüssel darf auch unverschlüsselt übertragen werden. Aber nur der Besitzer des privaten Schlüssels kann diese wieder entschlüsseln. (BHATTACHARJEE 2018 // 2017, Asymmetrical Encryption)

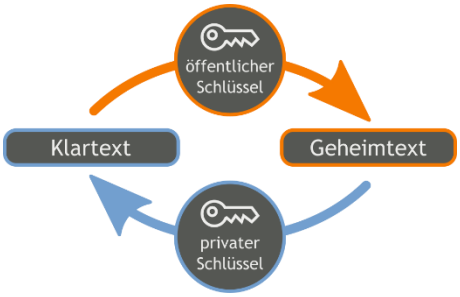

Abbildung 3 Asymmetrische Verschlüsselung

Kerckhoffs' Prinzip

Der niederländische Kryptologe Auguste Kerck formulierte bereits 1883 folgenden Grundsatz der Verschlüsselungen:

> Ein System muss immer noch sicher sein, wenn jeder den Aufbau und die Funktionsweise des Systems kennt, jedoch nicht das Passwort.

Der Verschlüsselungsalgorithmus muss aufgrund seines Aufbaus sicher sein und nicht weil seine Funktionsweise unbekannt ist. Alle modernen als sicher eingestuften Algorithmen erfüllen dieses Prinzip. Das asymmetrische Verfahren RSA, auf dem alle hier genannten Kryptologischen Verfahren Aufbauen ebenfalls. (Computer and information security handbook 2017, S. 43)

Digitale Signatur

Die digitale Signatur ist ein Anwendungsgebiet der Asymmetrischen Verschlüsselung. Mithilfe einer Hashfunktion wird ein bestimmter zu übertragender Inhalt beliebiger Länge in einen eindeutigen Fingerabdruck von fester Länge überführt (»gehasht«), uns anschließend mit dem privaten Schlüssel des Senders verschlüsselt. Dieser Wert nennt sich *Signatur* und wird an den eigentlichen Inhalt angehängt. Der Empfänger erstellt nun selbst den Hashwert des Inhalts. Die Signatur kann mithilfe des ebenfalls bekannten öffentlichen Schlüssels wieder in den Hashwert überführt werden, und auf Gleichheit mit dem erstellten Hash überprüft werden. Damit wird sichergestellt, dass nur der Besitzer des geheimen privaten Schlüssels eine gültige Signatur erstellen kann. Ist die Signatur ungültig, so trat entweder ein Übertragungsfehler auf, oder der Sender ist nicht Eigentümer des passenden privaten Schlüssels. (DocuSign)

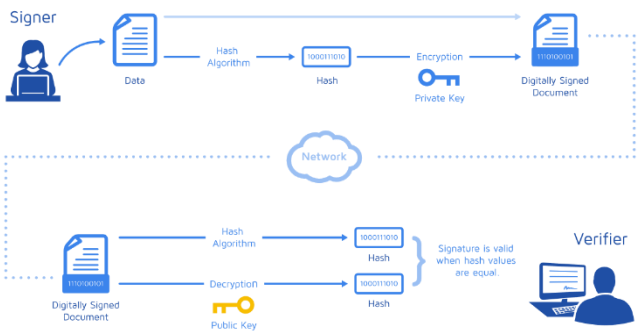

Abbildung 4 Funktionsweise einer digitalen Signatur

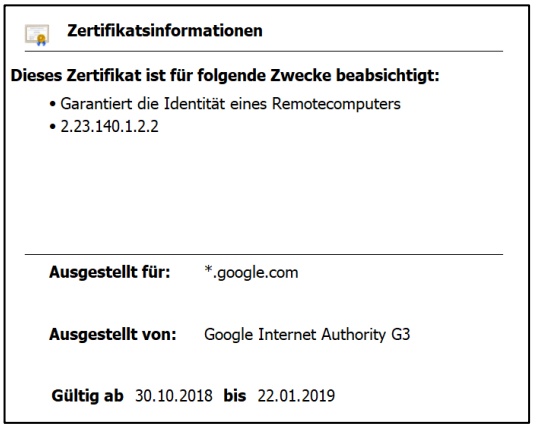

Abbildung 5 Beispiel eines Zertifikats

Über die digitale Signatur kann lediglich geprüft werden, ob der Sender im Besitz eines Privaten Schlüssels ist, der auf einen dem Empfänger bekannten öffentlichen Schlüssel passt. Dieser ist in der Regel wie die Identität des Senders weiterhin unbekannt. Um authentifizierte Kommunikationen zu ermöglichen werden vertrauenswürdige dritte Instanzen eingeführt. Diese werden Zertifizierungsstellen (*Certificate Authorities, CA*) genannt und stellen eine Art Notar innerhalb einer Infrastruktur (Public Key Infrastruktur) dar. Bei diesen CAs ist es jedem Teilnehmer der Public Key Infrastruktur möglich, ein Zertifikat zu beantragen, dass ihn als rechtmäßigen Besitzer einer Ressource ausweist. Es ist Aufgabe der CA, die Gültigkeit gewissenhaft zu überprüfen. Diese Zertifikate werden mit dem privaten Schlüssel einer Zertifizierungsstelle signiert. Die Teilnehmer des Systems müssen den öffentlichen Schlüssel der CA besitzen. Sie können ihn beispielsweise »ab Werk« enthalten. Dadurch lassen sich Zertifikate verifizieren. Durch die Einführung der CA müssen nicht mehr die öffentlichen Schlüssel aller Kommunikationspartner übertragen werden, sondern lediglich der Schlüssel der CA.

Das World-Wide-Web setzt für verschlüsselte Verbindungen ebenfalls auf Public Key Infrastrukturen. Jedes Betriebssystem enthält beispielsweise eine Reihe öffentlichen Schlüssel von Zertifizierungsstellen (CA). Auch mit der Installation von Webbrowsern werden diese Zertifikate installiert.

Die Zertifizierungsstellen sind das Nadelöhr des gesamten Systems und stellen eine potenzielle Verwundbarkeit dar. Wird ein privater Schlüssel einer CA bekannt, so können beliebige gültige Zertifikate ohne entsprechende Kontrolle erstellt werden. Ein effektives ist daher Schlüsselmanagement unerlässlich. Auf Seiten der CA können hochsensible Schlüssel beispielsweise durch Schwellenwert-Krypto Verfahren (Leonid Bender 2018) effektiv gesichert werden. (Ron Ih)

Transport Layer Security

Das Transport Layer Security (TLS) bildet eine Zwischenschicht zwischen der Anwendungsschicht und der Transportschicht. Ziel ist auf der einen Seite eine sichere und effiziente Verschlüsselung des Datenstroms der Anwendungsschicht zu erreichen. Und auf der anderen Seite nur authentifizierte Verbindungen zuzulassen. Der Kommunikationspartner muss immer eindeutig identifizierbar sein. Dadurch werden Man-In-The-Middle-Attacken **(AV 2)** erschwert. Erreicht wird dies durch Verwendung symmetrischer und asymmetrischer Verschlüsselungsverfahren sowie digitaler Zertifikate im Verbund mit einer Public-Key Infrastruktur. Ursprünglich wurde TLS von Netscape 1994 entwickelt, um eine Verschlüsselung für das im Internet gebräuchliche HTTPS Protokoll zu erreichen. Inzwischen wird TLS aber auch für eine Reihe weiterer etablierter Protokolle (FTP*S*, MQTT*S*, SMTP*S*) verwendet. Es stellt für verschlüsselte Verbindung im Internet eine Standardtechnologie dar.

Funktionsweise

Die zu übertragende Nachricht kann bei Asymmetrische Verschlüsselungen immer maximal so lang sein, wie der Schlüssel selbst. Dies verringert im Vergleich zu symmetrischen Verfahren, die diese Einschränkungen nicht haben, die Effektivität. TLS umgeht dieses Problem mit einer initialen Phase, in der mittels asymmetrischer Verschlüsselung ein gemeinsamer Symmetrischer Schlüssel vereinbart wird. Dieser wird für die restliche Kommunikation verwendet. Nach Aufbau der verschlüsselten Verbindung fordert der Client, (bei aktivierter *Client-Authentifizierung* auch der Server) das digitale Zertifikat der Gegenseite an. Anhand verschiedener Kriterien kann entschieden werden ob der Kommunikationspartner vertrauenswürdig ist. Üblicherweise wird beim Übertragen von Webseiten von einem Browser überprüft ob (1) die im Zertifikat vermerkte URL mit der angeforderten URL übereinstimmt, und (2) das Zertifikat von einer vertrauenswürdigen Certificate Authority ausgegeben wurde und (3) ob das Zertifikat noch gültig ist. (Easttom 2016, Kapitel 13)

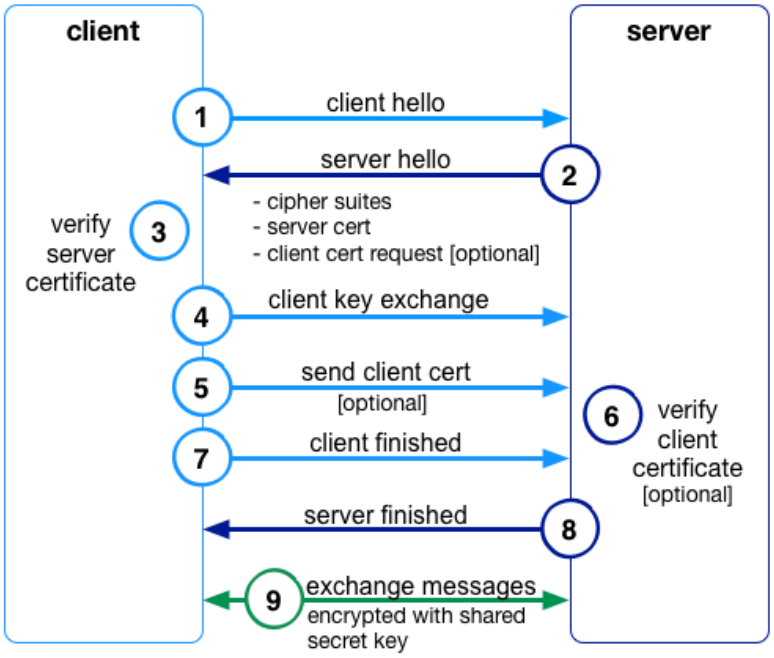

Abbildung 6 TLS Handshake

Best Practices

Es existieren eine Reihe von Schwachstellen, die das System angreifbar machen. Dabei handelt es sich meist um Angriffe gegen ältere Versionen von TLS. Das Bundesamt für Sicherheit in der Informationstechnik geben Best Practices heraus wie TLS dennoch sicher betrieben werden kann. (Bundesamt für Sicherheit in der Informationstechnik)

1. Verwenden von 2048 Bit Private Keys
2. Private Keys geheim halten
3. Auswählen einer sicheren Version (TLS 1.1, TLS 1.2)
4. Sichere Verschlüsselung auswählen
5. Forward Secrecy Protocol
6. Client-Initiated Renegotiation
7. Nach neuen Schwachstellen Ausschau halten

IT Sicherheit

Bedrohungen

Eine Bedrohung des Systems zielt darauf ab, eine oder mehrere Schwachstellen oder Verwundbarkeiten auszunutzen, um einen Verlust der Datenintegrität, der Informationsvertraulichkeit oder Verfügbarkeit zu erreichen oder um die Authentizität von Subjekten zu gefährden. Schwachstellen eines Systems ergeben sich aus Konfigurationsfehler und Implementierungsfehlern. Auch Personen, die selbst einen legitimen Zugriff auf ein System haben stellen eine Gefahrenquelle dar. Bei den Angreifern kann es sich um eine Reihe verschiedener Gruppierungen handeln: Hacker, Politische Gruppierungen, Konkurrenz, Mitarbeiter und Nachrichtendienste, die aus unterschiedlichsten Gründen agieren. Neben aktiven Angreifern von außen gibt es auch noch die unabsichtlich herbeigeführten Bedrohungen durch Höhere Gewalt (Stromausfall) sowie technisches, menschliches oder organisatorisches Versagen. Bedrohungen können in Kategorien unterteilt werden: 1. Unbefugter Informationsgewinn, 2. Unbefugte Modifikation, 3. Unbefugte Erzeugung, 4. Unbefugte Unterbrechung. (Rene Pfeiffer und Michael Kafka)

Mögliche Sicherheitsrisiken

Die an den häufigsten auftretenden Sicherheitsrisiken wurden in diesem Kapitel gesammelt und ausgeführt. Ein guter Startpunkt dafür ist die Open Web Application Security Project. »[Die] OWASP ist eine offene Community, die es Organisationen ermöglicht, vertrauenswürdige Anwendungen zu konzipieren, zu entwickeln, zu erwerben, zu betreiben und zu verwalten.« (The Open Web Application Security Project)

R1: Injection
Beispielsweise SQL-Injections treten auf, wenn nicht vertrauenswürdige Daten von einem Interpreter als Teil eines Kommandos verarbeitet werden. Ein Angreifer kann damit durch öffentliche zugängliche Eingabedaten (Formular) eigene Kommandos ausführen.

R2: Fehler in der Authentifizierung
Anwendungsfunktionen, die im Zusammenhang mit Authentifizierung und Session-Management stehen, werden häufig falsch implementiert. Das erlaubt Angreifern, Passwörter oder Session-Token zu kompromittieren und dadurch die Identität anderer Benutzer vortäuscht.

R3: Verlust der Vertraulichkeit sensibler Daten

Anwendungen schützen sensible Daten oftmals unzureichend. Dadurch können Angreifer dieses Abgreifen oder modifizieren. Diese für bösartige Zwecke missbraucht werden.

R4: Fehler in der Zugriffskontrolle

Häufig werden die Zugriffsrechte für authentifizierte Nutzer nicht korrekt um- bzw. durchgesetzt. Angreifer können entsprechende Schwachstellen ausnutzen, um auf Funktionen oder Daten zuzugreifen, für die sie keine Zugriffsberechtigung haben. Dies kann Zugriffe auf Accounts anderer Nutzer sowie auf vertrauliche Daten oder aber die Manipulation von Nutzerdaten, Zugriffsrechten etc. zur Folge haben.

R5: Sicherheitsrelevante Fehlkonfiguration

Fehlkonfigurationen von Sicherheitseinstellungen sind das am häufigsten auftretenden Problem. Ursachen sind unsichere Standardkonfigurationen, unvollständige oder ad-hoc durchgeführte Konfigurationen, ungeschützte Cloud-Speicher, fehlkonfigurierte HTTP-Header und Fehlerausgaben, die vertrauliche Daten enthalten. Betriebssysteme, Frameworks, Bibliotheken und Anwendungen müssen sicher konfiguriert werden und zeitnah Patches und Updates erhalten.

R6: Cross-Site-Scripting

XSS tritt auf, wenn Anwendungen nicht vertrauenswürdige Daten entgegennehmen und ohne Validierung oder Umkodierung an einen Webbrowser senden. XSS tritt auch auf, wenn eine Anwendung HTML- oder JavaScript-Code auf Basis von Nutzereingaben erzeugt. XSS erlaubt es einem Angreifer, Scriptcode im Browser eines Opfers auszuführen und so Benutzersitzungen zu übernehmen, Seiteninhalte verändert anzuzeigen oder den Benutzer auf bösartige Seiten umzuleiten

R7: Unsichere Deserialisierung

Unsichere, weil unzureichend geprüfte Deserialisierungen können zu Remote-Code-Execution. Schwachstellen führen. Aber auch wenn das nicht der Fall ist, können Deserialisierungsfehler Angriffsmuster wie Replay-Angriffe, Injections und Erschleichung erweiterter Zugriffsrechte ermöglichen.

R8: Nutzung von Komponenten mit bekannten Schwachstellen

Komponenten wie Bibliotheken, Frameworks etc. werden mit den Berechtigungen der zugehörigen Anwendung ausgeführt. Wird eine verwundbare Komponente ausgenutzt, kann ein solcher Angriff von Datenverlusten bis hin zu einer Übernahme des Systems führen.

Applikationen und APIs, die Komponenten mit bekannten Schwachstellen einsetzen, können Schutzmaßnahmen unterlaufen und so Angriffe mit schwerwiegenden Auswirkungen verursachen.

R9: Cross-Site-Request-Forgery

Anfragen auf angemeldete Seiten werden nicht im Sinne des Benutzers gemacht, sondern durch Angreifer.

R10: Verhinderung von Diensten (DDoS) (Überlastung)

Absichtlich herbeigeführte Überlastung eines Systems, führt dazu, dass dieser keine Anfragen mehr entgegennehmen kann.

R11: Man in the Middle / Unsichere Kommunikation

Ungesicherte bzw. schlecht gesicherte Kommunikation kann abgehört werden.

R12: Physischer Zugriff auf Ressourcen

Direkter Zugriff auf Hardware / Festplatten und Geräte.

R13: Unsichere Passwörter / Secret Management

Keine Passwortrichtlinien. Zulassen von einfachsten Passwörtern, keine Begrenzung der Anmeldeversuche. Passwörter können erraten werden.

R14: Unzureichendes Zertifikatsmanagement

Benutzte Zertifikate laufen nicht ab, Zertifikate werden nicht auf Gültigkeit überprüft, Zertifikate können nicht als ungültig markiert werden. Dadurch können selbst kompromittierte Zertifikate weiterhin benutzt werden.

R15: Social Engineering

Angreifer erschleichen sich das Vertrauen von autorisierten Benutzern des Systems. Dieses wird ausgenutzt, um Passwörter oder ähnlichen Zugang zum System zu erschleichen.

R16: Buffer Overflow

Durch Fehler in der Programmierung ist es möglich auf geschützte Speicherbereiche zuzugreifen unautorisiert Daten fremder Prozesse auszulesen oder im schlimmsten Fall eigenen Code ausführen zu können. (Syracuse University)

Das konzedierte System muss gegen diese Risiken gesichert sein. Daher wird im Laufe der Arbeit immer wieder Bezug auf diese Risiken genommen um (Design-) Entscheidungen zu begründen.

Weitere Begriffserklärungen

Authentisierung, Authentifizierung & Autorisierung

»Der erste Schritt zur Authentifizierung stellt die Authentisierung dar. Ein Benutzer legt mit der Authentisierung einen Nachweis einer bestimmten Identität vor, die vom System zu verifizieren und zu bestätigen ist. […]

Auf die Authentisierung folgt die Authentifizierung. Sie stellt die eigentliche Prüfung der vom Nutzer behaupteten Identität dar. […] Er ist in Besitz geeigneter Informationen, um die Identität des Users anhand der mitgeteilten Merkmale zweifelsfrei zu verifizieren.

Ist die Authentifizierung abgeschlossen und die Identität vom System bestätigt, übernimmt die Autorisierung die eigentliche Zuteilung der Zugriffsberechtigungen des Users. Hierfür sind im System für jede Identität Informationen oder Regeln für die gewünschten Zugriffsberechtigungen hinterlegt. Nach der Autorisierung kann der Benutzer bestimmte Leistungen oder Services nutzen. Lässt sich die Identität nicht bestätigen oder es sind keine Zugriffsrechte für eine Identität im System hinterlegt, wird die Nutzung untersagt.« (https://www.security-insider.de/was-ist-authentifizierung-a-617991/)

KAPITEL 3

Analyse & Anforderungen

Marktanalyse

Es gibt bereits Hersteller, die OTA-Updates in unterschiedlicher Tiefe implementiert haben. Tesla ist das prominenteste Beispiel. Auch viele bekannte Zuliefererbetriebe arbeiten an solchen Technologien. Um einen Überblick über mögliche Funktionsweisen und Anforderungen an ein solches System zu bekommen wurden öffentliche Unterlagen verschiedener Unternehmen analysiert.

Tesla

Der kalifornische Autobauer ist schon seit geraumer Zeit als Wegbereiter für neue Technologien bekannt. Als erster bekannter Autohersteller führte Tesla Over-The-Air Updates mit seinem Model Tesla S im Jahr 2012 ein. (Sam Byford) Diese beschränken sich nicht auf Sicherheitsunkritische Systeme wie das Infotainment. Es werden regelmäßig Fahrassistenzsysteme, ja sogar Bremssteuerungen aktualisiert. Der genaue Aufbau des Systems ist nicht bekannt. Um dennoch eine Vorstellung über die Möglichkeiten zu bekommen, werden einige bedeutenden Updates dargestellt.

- Ein Fehler, im Ladesystem konnte zu einem Brand des Fahrzeugs führen. Die ca. 30.000 betroffenen Fahrzeuge wurden durch ein Software-Update »repariert«. (Alex Brisbourne)
- Die US-Amerikanische Verbraucherschutzorganisation Consumer Reports berichtete im Mai 2018 über lange Bremswege. Tesla verkürzte diese durch ein Software-Update um bis zu sechs Meter. (Andreas Donath)
- Das Model S wird mit unterschiedlichen Reichweiten angeboten. Diese wird im günstigeren Modell durch Software heruntergeregelt. Im Zuge des Hurrikan Irma, der im September 2017 Florida bedrohte, mussten die Bewohner evakuiert werden. Tesla schaltete daraufhin per Update die vollständige Reichweite für alle Kunden frei. (Brian Fung 2017)
- Der bekannte Auto-Pilot wird ebenfalls regelmäßig durch Software-Updates um Features erweitert oder es werden Schwachstellen behoben. (Tesla)

Bosch

Die Robert Bosch GmbH ist eine der größten Automobilzulieferer weltweit. Im Jahr 2016 wurde auf der Messe »Bosch Connected World« ein System zur Over-The-Air Updatemöglichkeiten von Fahrzeugen demonstriert. Das System besteht wie in Abbildung 7 zu sehen, aus drei Komponenten, der *CCU* (Connectivity Control Unit, Kommunikationseinheit), dem *Authoring-System* und der *Bosch IoT Cloud*. (Bosch Software Innovations)

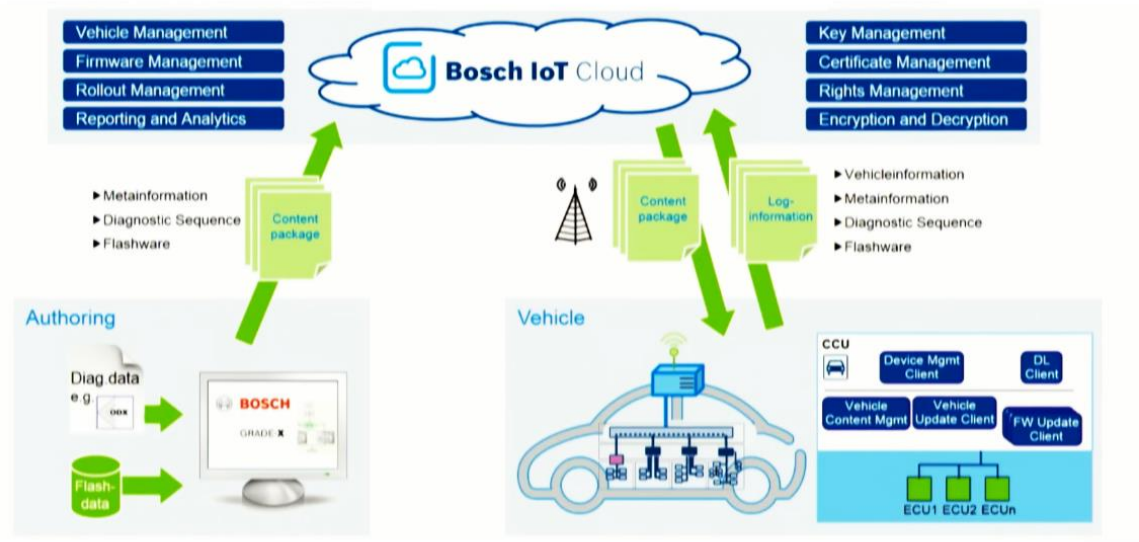

Abbildung 7 Aufbau der Bosch IoT Cloud

Die Bosch IoT Cloud stellt eine Geräteverwaltung dar. Hier werden alle Fahrzeuge und Updates registriert. Anschließend ist ein digitaler Zwilling des Fahrzeugs vorhanden. Das heißt alle Daten des Fahrzeuges werden in Echtzeit mit der Cloud synchronisiert. So lässt sich dessen aktueller Status (Telemetriedaten, Softwareversion, Standort) von überall nachverfolgen. Updates können einzelnen Autos oder ganzen Modellgruppen zugewiesen werden. Geräte werden stufenweise aktualisiert. Dabei werden nicht alle Fahrzeuge gleichzeitig aktualisiert, diese werden in kleinere Gruppen unterteilt, die sequentiell abgearbeitet werden. Falls ein Fehler im Updatevorgang erkannt wird, wird der Updatevorgang der restlichen Gruppen abgebrochen. Der Fehler bleibt gering. (Bosch Software Innovations)

Die Kommunikationseinheit beinhaltet Technologien um mit der Außenwelt (Bosch IoT Cloud), über WLAN, Mobilfunk, etc. zu kommunizieren. Intern erfolgt über das Bus System des Autos (CAN, Automotive Ethernet) eine Anbindung an die weiteren Steuergeräten, Sensoren, etc. eines Fahrzeugs Ein Auto enthält eine Vielzahl an Steuergeräte, eines für den Motor, ein ABS Steuergerät, das Infotainment-System und weiteres (Wolfgang Pester 2015). Da Vorkommen kann, dass bei einem Updatevorgang mehrere Steuergeräte aktualisiert werden, muss dieser Vorgang von einer zentralen Instanz gesteuert werden. Bei Bosch ist das die Central Connectivity Unit (CCU).

Wichtig zu erwähnen ist, dass diese Plattform die eigentliche Logik des Updates nicht enthält. Es wird lediglich dafür gesorgt, dass nicht näher definierte Updatepakete sicher und

unbeschädigt in ein Fahrzeug übermittelt werden. Ein sogenanntes Authoring System sorgt dafür, beliebige Dateiformatem, die für Updates benötigt werden, in ein für die Übertragung geeignetes Format zu übertragen.

Harman

Harman ist ebenfalls ein Zulieferer der Automobilindustrie, sein Fokus liegt auf der Entwicklung von Infotainmentsystemen. Die beworbene OTA Technologie verspricht seinen Kunden, das gesamte Auto updaten zu können. Von der Head-Unit angefangen, bis zum kleinsten Steuergerät. Dabei sollen sich die Autos bequem administrieren lassen. Es wird explizit damit geworben Angriffe wie:

1. Netzwerkprobleme
2. Cyber-Security Attacken
3. Dateimanipulation

zuverlässig zu verhindern. Um Updates datensparsam zu übertragen wird auf eine Technologie unter der Bezeichnung »Smart Delta« zurückgegriffen. (Harman)

Formulierung eigener Anforderungen

User Stories?

Aus den bisherigen Beispielen sollen nun Anforderungen kondensiert werden. Dabei hat sich herausgestellt, dass es zwei Akteure gibt, die Anforderungen an das System stellen. Der *Fahrzeughersteller*, der das System betreut, Updates erstellt und aufspielt. Und den Fahrzeughalter, der diese konsumiert. Die Größte Schwierigkeit beim Formulieren von Software Anforderungen ist die Kommunikation zwischen denjenigen, die das Produkt benutzen und denjenigen die es entwickeln. Eine in der agilen Softwareentwicklung häufig eingesetzte Art der Formulierung sind User Stories. Der Fokus liegt dabei auf **Was** passieren soll, und nicht **Wie** dieses Verhalten technisch realisiert werden kann.

Die Anforderungen werden in Form von User Stories dargelegt. Sie haben immer die gleiche Form:

> » Als (Anwendertyp) möchte ich (folgende Aktion durchführen), [damit (dieses Ziel erreicht wird)]. «

User Stories werden in der Regel vom Kunden des Projekts geschrieben, und beinhalten keine Technischen Spezifikationen, sondern beschreiben wie sich ein System verhalten soll. (Cohn 2013, Kapitel 1)

Beispiel einer guten User Story:

- Als Benutzer möchte ich nach neuen Stellenangeboten suchen können.
- Als Administrator möchte ich andere Nutzer sperren können

Schlechte Beispiele:

- Das Produkt soll in C++ Programmiert werden
- Das Programm verbindet sich durch einen Connection-Pool mit der Datenbank

Daraus folgen die Anforderungen der Akteure an das System. Anforderungen, die angedacht waren, aber aufgrund des Umfangs nicht behandelt wurden sind in runde Klammern gefasst.

Als Fahrzeughersteller möchte ich…

US A1: …Autos in einem zentralen Verzeichnis anlegen und löschen können.

US A2: …Den aktuellen Status eines Autos erfragen können.

US A3: …Updates verschiedener Art in einem zentralen Verzeichnis anlegen können.

US A4: (…Updates individuell oder in Gruppen zuweisen können.)

US A5: …atomare Updatevorgänge haben. Es muss sichergestellt sein, dass Updates entweder vollständig oder gar nicht ausgeführt werden.

US A6: …Benutzer für dieses zentrale Verzeichnis anlegen und löschen können.

US A7: (…verschiedenen Benutzern verschiedene Berechtigungen erteilen.)

US A8: …sicherstellen, dass nur befugte Benutzer auf das System zugreifen können.

US A9: …die Möglichkeit haben, beliebige Steuergeräte des Autos aktualisieren können.

US A10: …den Regelmäßigen Austausch von Zertifikaten unterstützen.

US A11: …für die Verwaltung eine WebApp verwenden.

Als Fahrzeughalter möchte ich…

US B1: …einen Updatevorgang, der sich nicht negativ auf die Performance meines Autos auswirkt.

US B2: …stets eine aktuelle Softwareversion für mein Auto beziehen, um immer die neuesten Features nutzen zu können.

US B4: …stets eine aktuelle Softwareversion für mein Auto beziehen, um immer eine maximal große Sicherheit zu genießen.

US B5: …Schnelle Updatevorgänge

US B6: …Keine extra Kosten durch diese Technologie.

US B7: …Funktionsfähigen Updatevorgang unter jede Bedingung. (Verlust der Energieversorgung, Verlust von Mobilfunk, Bedienungsfehler)

US B8: …Updates ortsunabhängig ausführen können

Anforderungen

Diese User Stories können jetzt in Anforderungen übertragen werden. Dabei wird unterschieden in: *Funktionale Anforderungen*, die beschreiben was das Produkt tun soll, und

Nichtfunktionalen Anforderungen. Diese sind Randbedingungen des Systems und beschreiben wie leistungsfähig, schnell das System seine Anforderungen erledigt. Anforderungen, die angedacht waren, aber aufgrund des Umfangs nicht behandelt wurden sind in runde Klammern gefasst.

Funktionale Anforderungen

A1: Es muss eine Zentrale Verwaltungsinstanz geben.

A2: Es müssen Fahrzeuge angelegt, angesehen, bearbeitet und gelöscht werden können.

A3: Es müssen Updates angelegt, angesehen, bearbeitet und gelöscht werden können.

A4: Es müssen Benutzer angelegt, angesehen, bearbeitet und gelöscht werden können.

A5: Fahrzeugdaten müssen, Model, Besitzer, ID und Telemetriedaten enthalten.

A6: Es muss eine genormte Schnittstelle existieren, um verschiedene Arten von Updates zu übertragen.

A7: (Benutzer müssen Berechtigungen des Systems beinhalten.)

A8: Es muss eine Infrastruktur vorhanden sein, um Updates auf einem Auto aufzuspielen.

A9: (Updates müssen eine Gruppenzugehörigkeit besitzen.)

A10: Updates müssen auch über eine schlechte Verbindung übertragen werden können.

A11: Update Vorgänge müssen so weit wie möglich ausgeführt werden, ohne dass das Fahrzeug stehen oder Neugestartet werden muss.

A12: Updates dürfen die Nutzererlebnis des Fahrzeugs nicht verschlechtern.

A13: Updates müssen in jedem Zustand abgebrochen werden können, ohne einen funktionsunfähigen Zustand zu hinterlassen.

A15: (Die Geräteverwaltung kann über eine WebApp angesprochen werden.)

A16: Nur berechtigte Benutzer dürfen auf die Geräteverwaltung zugreifen.

Nicht Funktionale Anforderungen

A21: Authentifizierung und Autorisierung erfolgt Systemweit Zertifikatsbasierend

A22: Aufbau eines Resilienten Systems. (robust, lauffähig unter allen Bedingungen)

A23: Ein Updatevorgang kann aus mehreren kleineren Updatevorgängen zusammengesetzt sein.

A24: (Eine Testabdeckung des Codes in Höhe von 90% garantieren.)

A12: Ansteuerung der Geräteverwaltung über verschiedene Arten (Oberfläche, Maschinell), Implementierung einer API.

A13: Zertifikate müssen jederzeit durch die Geräteverwaltung ausgetauscht werden können.

Externe Anforderungen

Durch die Umwelt ergeben sich weitere Anforderungen. Alle betriebenen Systeme unterliegen gesetzlichen Bestimmung. In diesem Falle, IT-Sicherheits-, und Datenschutzgesetze. Die folgende von Gesetzen Analyse ist auf die Gesetze der Bunderepublik Deutschland beschränkt

IT-Sicherheitsgesetz

Betreiber von kritischen Infrastrukturen (KRITIS) werden verpflichtet eine Reihe von Sicherheitsrelevanter Maßnahmen umzusetzen. »Zu den kritischen Infrastrukturen zählt eine ganze Reihe verschiedener Sektoren. Durch die besonderen Anforderungen an die Betreiber dieser Infrastrukturen verfolgt das IT-Sicherheitsgesetz die Ziele, Schaden vom Einzelnen abzuwenden, das Gemeinwesen zu schützen und negative Auswirkungen für alle zu minimieren. Als kritische Infrastrukturen sind unter anderem die Sektoren Energie, Ernährung und Wasser, Informationstechnik und Telekommunikation, Finanzen, Gesundheit sowie Transport und Verkehr definiert« (Definition IT-Sicherheitsgesetz) Diese werden verpflichtet einen bestimmten Mindestgrundschutz, der sich an branchenspezifischen, im Gesetz vermerkten, Sicherheitsstandards orientiert, einzuhalten. Interessanterweise wird in den Bestimmungen des IT-Sicherheitsgesetzes, empfohlen immer aktuelle Updates einzuspielen. (Bundesamt für Sicherheit in der Informationstechnik)

Datenschutzgesetz

Das System sammelt Telemetriedaten des Fahrzeugs, dazu können der Aufenthaltsort des Autos und weitere private Daten des Fahrzeughalters gehören. Diese sensiblen Daten unterliegen Datenschutzbestimmungen, wie der Datenschutzgrundverordnung (DSGVO). Die Einhaltung aller Datenschutzbestimmung erfordert hohen Aufwand und ist nicht notwendig,

um eine funktionierende Architektur zu betrachten. Daher wird dieses Thema in dieser Arbeit ausgeklammert.

Architektur

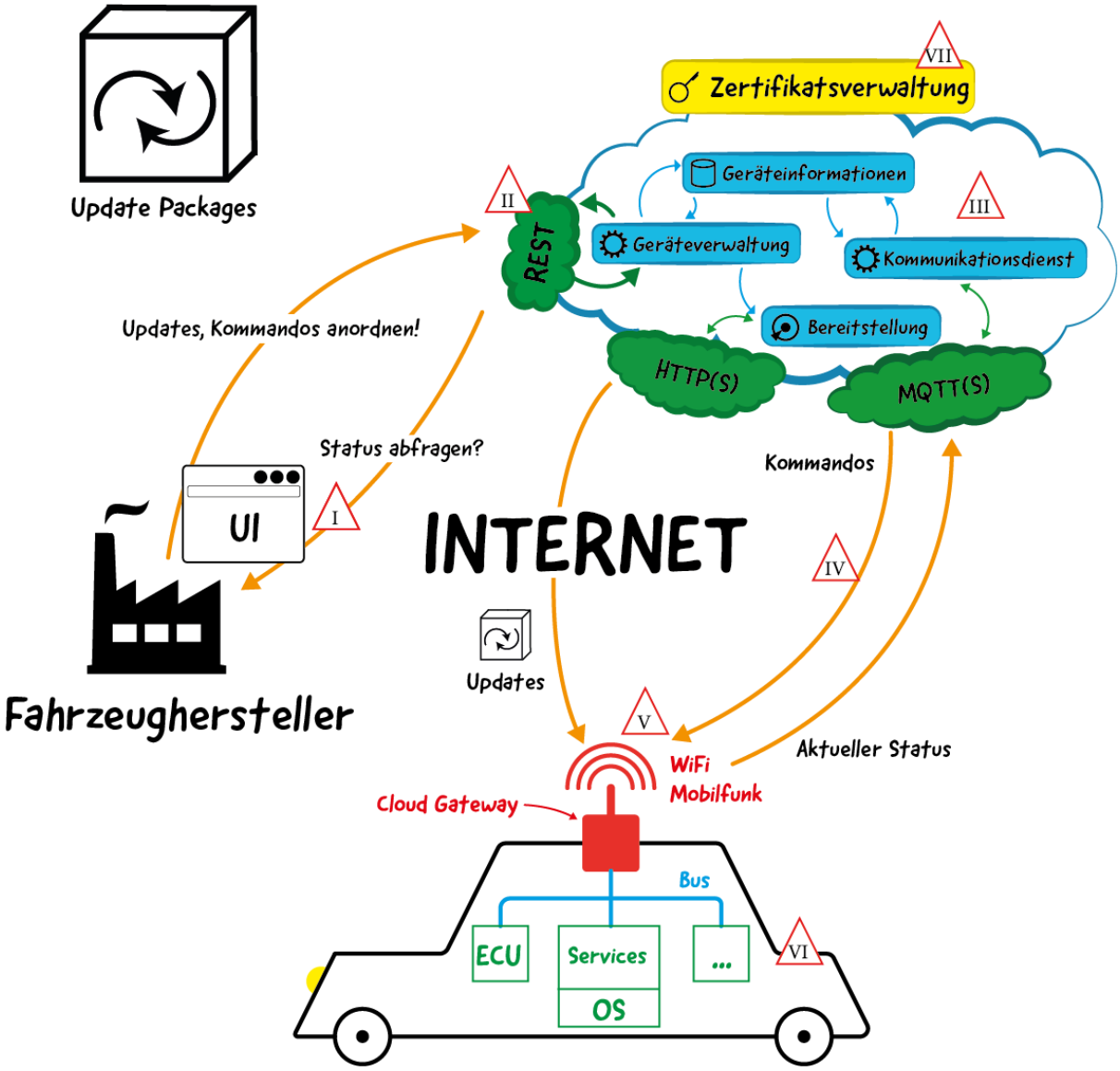

Abbildung 8 Schematische Darstellung des Konzepts

Überblick

Aus der Anforderung **A1** ergibt sich die Notwendigkeit einer Zentralen Instanz (ab hier Geräteverwaltung genannt). Auf dieser werden Autos und Updates und Benutzer verwaltet. Diese soll über eine API Verfügen, die von einer beliebigen Oberfläche angesteuert werden kann. Die Fahrzeughersteller sollen über eine Web-App mit dieser kommunizieren können. (**A15**). Dabei sollen nur autorisierte Benutzer Zugang haben (**A16**). Es ist geplant, die Infrastruktur, die benötigt wird, um die Komponenten zu betreiben nicht selbst aufzubauen, sondern auf Webanbieter wie Amazon AWS, Microsoft Azure oder Google Cloud zurückzugreifen. Dadurch kann das Behandeln von bestimmten Risiken ausgelagert werden. Beispielsweise die physische Zugriffskontrolle (**R12**) und die Sicherheitsrelevante Fehlkonfiguration (**R5**). Zu den Komponenten gehört ein Kommunikationsdienst, der Fahrzeug über vorhandene Updates informiert und Statusaktualisierungen (Telemetriedaten wie Standort, Benachrichtigungen über ausgeführte Udpates, etc.) von diesen entgegennehmen kann. Der Kommunikationsdienst und die Geräteverwaltung teilen sich dabei die gleiche Datenpersistenz. Um Updates auch an Autos ausliefern zu können existiert ein Bereitstellungsserver. Auf der Seite des Fahrzeugs steht ein Gerät, das mit der Cloud kommuniziert und Updates ausführt. Beispielsweise die Head-Unit eines Fahrzeugs, oder ein eigenes Steuergerät. Dieses Gerät muss über Schnittstellen verfügen, um sowohl mobil mit dem Internet verbunden zu sein als auch mit der anderen Peripherie des Fahrzeugs kommunizieren können. Durch ein Authoring System können beliebige Update Pakete in ein für die Bereistellung und Übertragung geeignetes Dateiformat konvertiert werden.

Komponenten

Auflistung der einzelnen Komponenten der Gesamtarchitektur.

I: Oberfläche der Geräteverwaltung

II: Api der Geräteverwaltung

III: Backend

IV: Kommunikationskanal zwischen Auto und Backend

V: Kommunikationsschnittstelle des Fahrzeugs

VI: Cloud Gateway

VII: Zertifikatsverwaltung (PKI)

VIII: Update Package

Geräteverwaltung

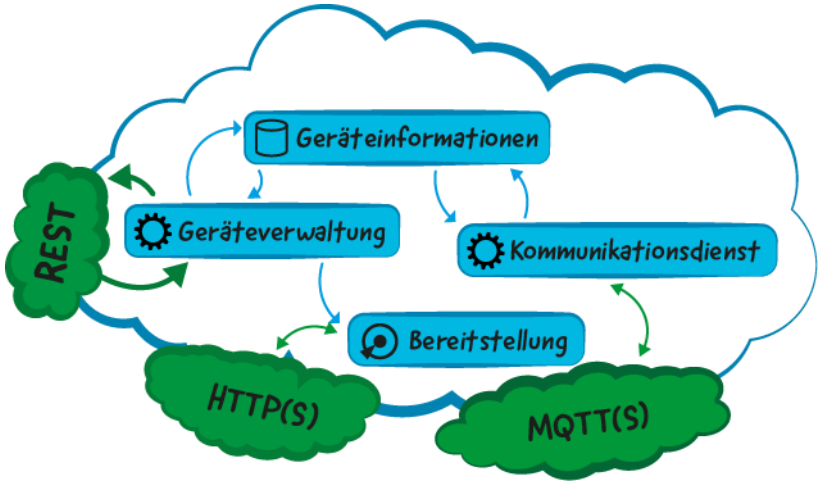

Abbildung 9 Überblick über die Geräteverwaltung

Die Geräteverwaltung besteht aus einer Reihe von Services und Kommunikationsschnittstellen, um mit der Fahrzeugflotte zu kommunizieren, deren Zustand zu persistieren und Kommandos wie Updates an diese zu versenden. Über eine weitere Schnittstelle kann sie von autorisierten Benutzern angesteuert werden.

Datenstruktur

Die Datenstruktur ergibt sich direkt aus den Anforderungen (**A1, A2, A3, A4, A5, A7, A9**). Im Mittelpunkt stehen sowohl die einzelnen Geräte (Device) als auch Updates (Update). Die Geräte (Device) Entität enthält alle aktuellen Informationen eines Geräts. Etwa Besitzer, Aktueller Standort, Modelltyp, Fahrzeugnummer. Die Update Entität stellt ein Update dar. Es enthält einen Verweis auf ein Updatepaket, welches auf dem Bereitstellungsserver verfügbar ist. Auch enthält es eine Reihe von Voraussetzungen, die erfüllt sein müssen, damit das Update überhaupt heruntergeladen wird. Dazu gehören am wichtigsten ob der Fahrzeugtyp und dessen aktuelle Softwareversion dem Update entsprechen. Anforderungen ob das Auto für das Update in Bewegung sein darf, oder den minimalen Ladestatus der Batterie. Die Entität UpdateTask stellt nun einen konkreten Auftrag für einen Updatevorgang dar. Dabei wird ein Gerät mit einem Update und dem Gerät auch der Aktuelle Status gesichert. Der Status eines Updates kann drei Werte annehmen.

1. Update noch nicht ausgeliefert
2. Update ausgeliefert
3. Update ausgeführt

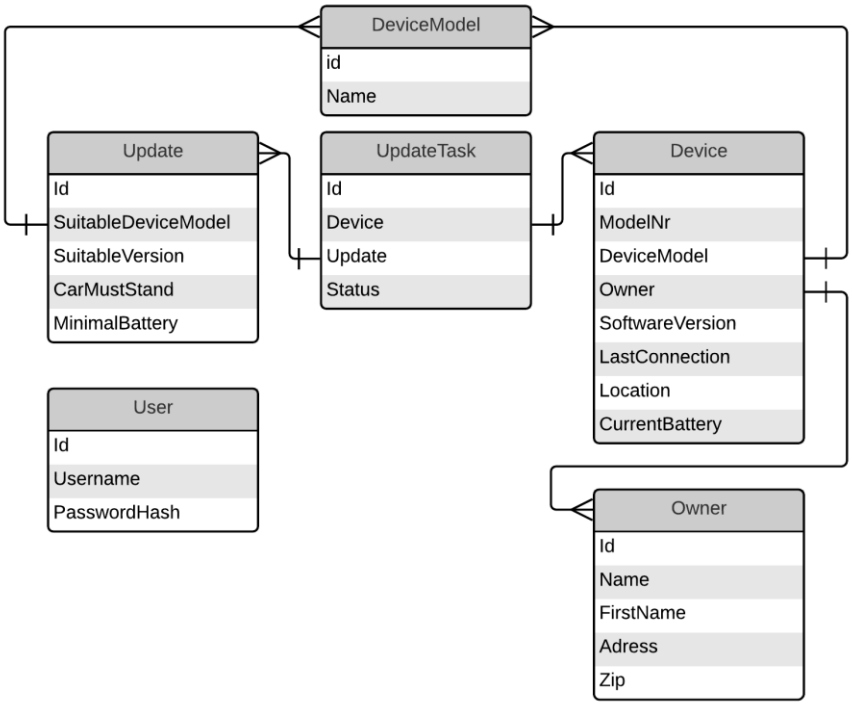

Abbildung 10 Datenmodell der Geräteverwaltung

Um eine Benutzerverwaltung durch das System zu ermöglichen existiert die Entität User. Diese enthält einen Benutzernamen und ein Passwort, dass für den Login verwendet wird. Aus Sicherheitsgründen wird dieses nicht im Klartext gesichert, sondern mit einem sicheren Verfahren »gehasht« (siehe **R2**, **R3**). Für ein sicheres Authentisierungsverfahren ist der Einsatz von Passwortrichtlinien unerlässlich. (**R13**) Lässt sich ein Passwort leicht durch ausprobieren erraten, nützt auch die sicherste Speicherung des Passwortes nichts. Die Sicherheit eines Systems ist nur so stark, wie die des schwächsten Glieds.

Representational State Transfer

Die Schnittstelle auf die Geräteverwaltung wird als Rest-API ausgeführt werden. Rest steht dabei für Represential State Transfer und ist ein Konzept aus Verteilte Systeme, etwa Client-Server Anwendungen, für Remote Procedure Calls. Dabei werden viele Paradigmen aus dem World-Wide-Web wiederverwendet. So wird in der Regel HTTP(S) als Übertragungsprotokoll verwendet. Welche Ressourcen abgefragt werden, bzw. welche Aktionen ausgeführt werden sollen wird über die verwendete URL spezifiziert. Für eine Verschlüsselung der Verbindung wird HTTPS also http über TLS eingesetzt. Die aus HTTP bekannten Methoden GET, POST, PUT, DELETE werden in dementsprechenden Aktionen für Ressourcen übersetzt. Dabei gilt: *GET* stellt einen Ressourcenzugriff dar (*Read*), POST legt eine Ressource an (*Create*), *PUT* aktualisiert eine Ressource (*Update*) und *DELETE* löscht diese (*Delete*). Die Übertragung von Daten erfolgt durch serialisierte Objekte, die in der Regel entweder als JSON oder XML dargestellt werden. Im Konzept wird das leichtgewichtige JSON Format verwendet. Das HTTP Protokoll an sich ist Statuslos. Jede Anfrage an den Webserver ist unabhängig voneinander. Daraus folgt, dass der Webserver erstmals nicht weiß was der aktuelle Benutzer vorhergehend getan hat, beispielsweise ob dieser schon eingeloggt ist. Um einen Status wiederherzustellen gibt es zwei Möglichkeiten. Die klassische Variante ist das Einführen von Session-Cookies. Dabei legt der Server alle Daten zum aktuellen Status eines Benutzers in einer Datenbank ab und verknüpft diese mit einer eindeutigen ID, diese muss vom Client bei jeder Anfrage mitgesendet werden. Eine neuere Variante sind sogenannte WebTokens (JWT). Der Zustand wird nicht vom Server gesichert, sondern vom Client selbst. Dabei werden die Daten allerdings vom Server selbst signiert, damit diese nicht manipulierbar sind.

Datenobjekte der API

Für die Übertragung von Daten über die API werden eigene Datenobjekte, sogenannte Models verwendet. Diese sind oft ähnlich zu den Datenobjekten (Entitäten) der Datenbank, enthalten aber nur die notwendigen Informationen und sind für die serialisierte Übertragung optimiert. Abbildung 11 zeigt ein Klassendiagram für jedes verwendete Objekt dargestellt. In der Schnittstellendokumentation wird dann auf diese Objekte verwiesen.

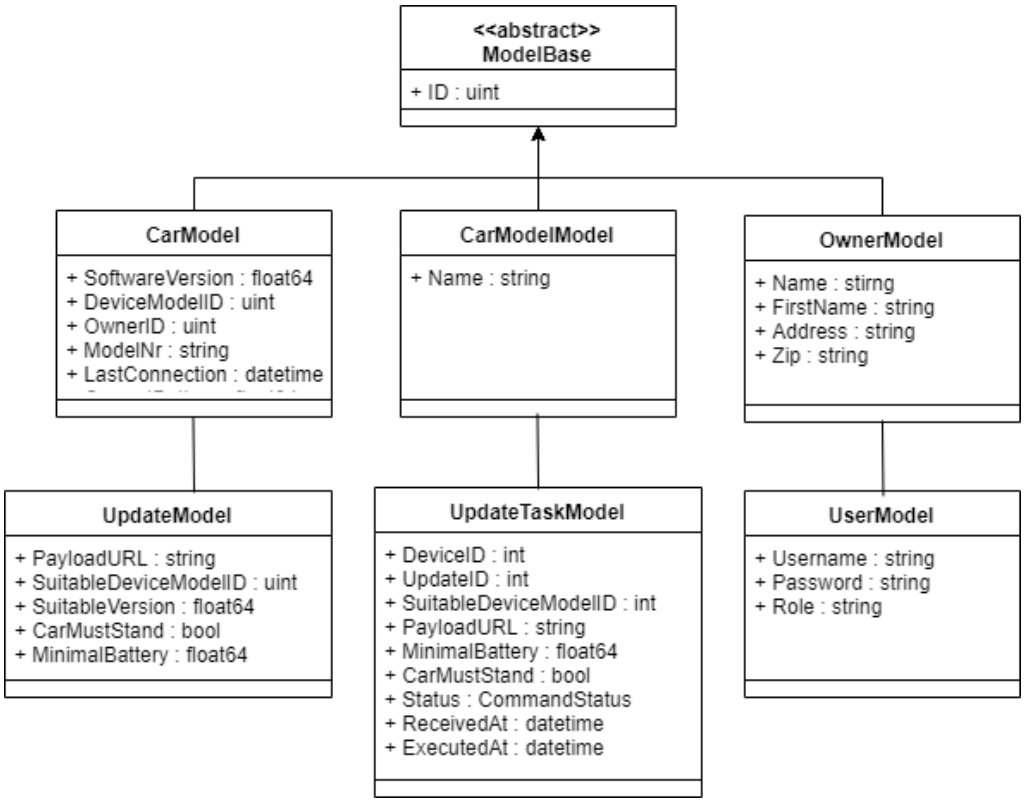

Abbildung 11 Datenmodelle der Rest Api

Schnittstellendokumentation

Alle Methoden der API Schnittstelle werden aufgezeigt. Dabei wird die verwendete URL, die HTTP Methode und ebenso versendeten Datenstrukturen gezeigt. Der Body Parameter, stellt Daten dar, die vom Benutzer an den Server versendet werden. Andersherum ist der Response Parameter eine Antwort des Servers. Falls eine Aktion nicht erfolgreich abläuft wird diese mit einem HTTP Fehlercode quittiert. Der Wert `nil` steht hierbei für einen leeren Wert.

Titel:	Alle Geräte anzeigen
URL:	`/devices/`
Methode:	GET
Request:	`nil`
Response:	`Array<DeviceModel>`

Titel:	Gerät hinzufügen

URL:	`/devices/`
Methode:	POST
Request:	`DeviceModel`
Response:	`Success`

Titel:	Gerät aktualisieren
URL:	`/devices/`
Methode:	PUT
Request:	`DeviceModel`
Response:	`Success`

Titel:	Gerät löschen
URL:	`/devices/{id}`
Methode:	DELETE
Response:	`Success`

Titel:	Alle Gerätemodelle anzeigen
URL:	`/devices/models/`
Methode:	GET
Request:	`nil`
Response:	`Array<CarModelModel>`

Titel:	Gerätemodell hinzufügen
URL:	`/devices/models/`
Methode:	POST
Request:	`CarModelModel`
Response:	`Success`

Titel:	Gerätemodell aktualisieren
URL:	`/devices/models`

Methode:	PUT
Request:	`CarModelModel`
Response:	`Success`

Titel:	Gerät löschen
URL:	`/devices/models/{id}`
Methode:	DELETE
Response:	`Success`

Titel:	Alle Besitzer anzeigen
URL:	`/owners/`
Methode:	GET
Request:	`nil`
Response:	`Array<OwnerModel>`

Titel:	Besitzer hinzufügen
URL:	`/owners/`
Methode:	POST
Request:	`OwnerModel`
Response:	`Success`

Titel:	Besitzer aktualisieren
URL:	`/owners/`
Methode:	PUT
Request:	`OwnerModel`
Response:	`Success`

Titel:	Besitzer löschen
URL:	`/owners/{id}`
Methode:	DELETE
Response:	`Success`

Titel:	Alle Update Tasks anzeigen
URL:	`/updatetasks/`
Methode:	GET
Request:	`nil`
Response:	`Array<UpdateTaskModel>`

Titel:	UpdateTask hinzufügen
URL:	`/updatetasks/`
Methode:	POST
Request:	`UpdateTaskModel`
Response:	`Success`

Titel:	UpdateTask aktualisieren
URL:	`/owners/`
Methode:	PUT
Request:	`UpdateTaskModel`
Response:	`Success`

Titel:	UpdateTask löschen
URL:	`/updatetasks/{id}`
Methode:	DELETE
Response:	`Success`

Titel:	Anmelden
URL:	`/login/`
Methode:	GET
Request:	`LoginModel`
Response:	`Success`

Risikoanalyse

Die Api stellt einen großen Angriffspunkt auf das Gesamtsystem dar. Wer in dieses System eindringt, kann die gleichen Aktionen ausführen wie ein autorisierter Benutzer. Daher ist es unerlässlich dieses System abzusichern. Es folgt eine Aufstellung der größten Risiken bei der Entwicklung von Apis und Gegenmaßnahmen.

Gefahr:	Unsichere Serialisierung (**R7**)
Erklärung:	Das REST Paradigma setzt sehr stark auf die Übertragung von serialisierten Objekten. Manipulierte Objekte können zu nicht autorisierten Maßnahmen führen.
Maßnahmen:	Verschlüsselte Verbindung verwenden. Zum Beispiel TLS.

Gefahr:	Unsichere Verbindung (R11)
Erklärung:	Die Verbindung zwischen Oberfläche und API-Endpunkt erfolgt unverschlüsselt. Eine Man In The Middle Attacke ist möglich
Maßnahmen:	Jedes Objekt nach Deserialisierung auf Gültigkeit überprüfen.

Gefahr:	Injection (**R1**)
Erklärung:	Eingabeparameter werden ungeprüft in Abfragen (Beispielsweise SQL) inkludiert. Dadurch lassen sich diese Abfragen verändern und unautorisierte Kommandos ausführen.
Maßnahmen:	Verwenden von bekannten Softwarekomponenten, die Eingaben auf Korrektheit überprüfen. Insbesondere verwenden von erprobten SQL Frameworks.

Gefahr:	Fehler in der Authentifizierung (**R2**)
Erklärung:	1. Benutzernamen / Passwörter können leicht erraten werden 2. Unverschlüsseltes Übertragen und Sichern. 3. Unendliches probieren von Benutzernamen / Passwort Kombinationen möglich.
Maßnahmen:	1. Verwenden von Passwortrichtlinien 2. Verschlüsselte Verbindung 3. Anzahl an Anmeldeversuche limitieren 4. Verwenden von 2-Faktor-Authentifizierung

---

Gefahr:	Fehler in Zugriffskontrolle (**R4**)
Erklärung:	Eingabeparameter werden ungeprüft in Abfragen (Beispielsweise SQL) inkludiert. Dadurch lassen sich diese Abfragen verändern und unautorisierte Kommandos ausführen.
Maßnahmen:	Verwenden von bekannten Softwarekomponenten, die Eingaben auf Korrektheit überprüfen. Insbesondere verwenden von erprobten SQL Frameworks.

Gefahr:	Social Engineering (**R15**)
Erklärung:	Erschleichen eines Systemzugangs von unberechtigten, durch Vertrauens Gewinnung einer berechtigten Person.
Maßnahmen:	Nicht jeder Benutzer darf auf jede Ressource zugreifen. Einschränken der Rechte. (Die Ausarbeitung eines Berechtigungskonzepts wurde in dieser Arbeit ausgespart)

Bereitstellungsserver

Ein Webserver, der die eigentlichen Daten der Updates bereithält. Zu beachten ist, dass auch hier die Kommunikation nur mit durch Zertifikate legitimierten Klienten kommuniziert werden darf und die Verbindung verschlüsselt ablaufen muss. Bei einem flächendeckenden Ausrollen von Updates an viele Geräte gleichzeitig kommt es zu erhebliche Datenraten. Diese können von einem einzelnen Server nicht mehr verarbeitet werden. (**R10**, Verhinderung von Diensten) Es empfiehlt sich der Einsatz von mehreren Instanzen eines Webservers in Kombination mit Lastverteilern. Dieser verteilt die Last der Anfragen gleichmäßig auf die vorhandenen Instanzen. Dadurch lässt sich ein skalierbares System bauen. Wie bereits genannt sollte diese Infrastruktur nicht selbst betrieben werden. Anbieter von Cloud Produkten, wie etwa

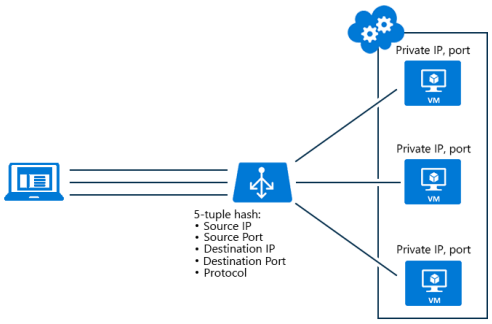

Abbildung 12 Funktionsweise eines Load Balanders (hier von Microsoft Azure)

Microsoft Azure, Google Cloud oder Amazon AWS haben große Erfahrung im Aufbau skalierbaren Systemen, die einer hohen Bandbreite standhalten.

Kommunikatonsdienst

Von der Geräteverwaltung aus können Updates angestoßen werden. Diese werden in Form von UpdateTasks in der Datenbank persistiert. Die Aufgabe des Kommunikationsdienstes ist es nun, diese Tasks an entsprechende Fahrzeuge zu übermittle. Dabei wird in einem festgelegten Intervall in der Datenpersistenz nachgesehen welche Benachrichtigungen noch ausstehend sind. Anschließend werden diese über einen Kommunikationskanal, der im folgenden Kapitel noch genauer beschrieben wird, versendet. Es ist auch zu überlegen, Updates nur zu statistisch günstigen Zeiten zu versenden. Ein Beispiel wäre, Updates die ein stillstehendes Fahrzeug voraussetzen, nur am Abend zu versenden, wo diese statistisch gesehen oft in ihrer Garage stehen.

Kommunikation

Überblick

Dieses Kapitel umfasst alle Kanäle, der Kommunikation zwischen Fahrzeug und der Geräteverwaltung. Besonderes Augenmerk liegt auf der Absicherung dieser Verbindungen. Allgemein gesagt werden kann, dass wieder alle Verbindungen und eingesetzte Protokolle mit dem TLS Protokoll kompatibel sind. Dieses stellt hier sowohl Verschlüsselung als auch Autorisierung bereit. Die Autorisierung der Geräteverwaltung als auch der einzelnen Geräte erfolgt Zertifikatsbasierend (**A21**). Die Geräteverwaltung agiert dabei als Certificate Authority einer eigenen Public Key Infrastruktur. Der Kommunikationsumfang umfasst das bidirektionale Austauschen von Nachrichten und das Herunterladen von Updates oder ähnlichen Datenpaketen. Bei allen TLS Verbindungen ist die Option der Clientseitigen Autorisierung aktiviert. Damit werden nur Geräteverwaltung bekannten Geräte für die Kommunikation zugelassen.

Zertifikatsverwaltung

Zertifikate unterliegen einer strengen Verwaltung, ein Diebstahl dieser Schlüssel hat weitreichende Folgen:

1. **Bekanntwerden des Privaten Schlüssels der CA**

➔ Es können beliebig viele gültige aber unautorisierte Gerätezertifikate erstellt werden.

➔ Es kann eine gültige Geräteverwaltung gemimt werden.

2. **Bekanntwerden von Geräteschlüsseln**

➔ Es besteht die Möglichkeit sich als fremdes Gerät auszugeben.

Der öffentliche Schlüssel der Geräteverwaltung ist ebenfalls »ab Werk« in jedem Auto installiert und dient der Überprüfung der Geräteverwaltung. Alle Zertifikate verfügen, ähnlich offiziellen Dokumenten wie Reisepässen, über ein Ablaufdatum. Sie müssen inregelmäßigen Abständen erneut zertifiziert werden. Damit lässt sich der Missbrauch von gestohlenen Zertifikaten eindämmen. (Vincent Lynch) Prinzipiell gilt: Je öfter ein Zertifikat ausgewechselt

wird, desto geringer fällt die Wahrscheinlichkeit aus, dieses zu stehlen und schädliche Aktionen damit auszuführen. Jedoch ist das Wechseln und die dadurch notwendige Übertragung eines neuen Zertifikats, ein Akt, der mit bestimmten Kosten verbunden ist. (Übertragung kostet Geld) Die Fahrzeuge sind für Besitzer oder Angreifern potentiell physisch verfügbar, es ist also durchaus im Bereich des Machbaren Zertifikate aus dessen Speicher zu extrahieren. Im Gegensatz dazu steht die Geräteverwaltung in einem gesicherten Umfeld und kann physisch nicht Angegriffen werden. Es sollte für beide Systeme ein vernünftiger Kompromiss zwischen Kosten und Sicherheit getroffen werden. Das Konzept orientiert sich an üblichen Laufzeiten im Bereich des Internets der Dinge. Ein Gerätezertifikat hat eine Laufzeit von *einem Monat* bei den Gerätezertifikaten und *einem Jahr* bei der Geräteverwaltung.

Publish / Subscriber Prinzip

Das Publisher-Subscriber (Pub-Sub) Pattern entkoppelt den Sender einer Nachricht vom jeweiligen Empfänger. Dazu wird ein sogenannter **Broker** (in Abbildung 13 als Warteschlangenmanager bezeichnet) als Mittelsmann eingeführt. Um eine Nachricht zu senden, wird eine sogenannte **Topic** benötigt. Dabei handelt es sich um eine eindeutige Bezeichnung, ein Postfach, an das die Nachricht gesendet wird. Beliebig viele Sender können Nachrichten an den Broker an ein bestimmtes Topic schicken (Veröffentlichung, engl.: Subscribe). Empfänger können beim Broker ihr Interesse an bestimmten Topics anmelden (Subskription). Wird nun eine Nachricht veröffentlicht, informiert der Broker, alle an diesem Topic interessierten Empfängern. Durch die Entkopplung ist es möglich asynchrone Nachrichten zu verschicken. Durch diese Entkoppelung können Nachrichten auch dann versendet werden, wenn der Empfänger, durch Unterbrechung der Internetverbindung nicht direkt erreichbar ist. Nachdem eine Verbindung wieder ausgebaut wird, liefert der Broker alle noch ausstehenden Nachrichten nach, (IBM)

Abbildung 13 Pub/Sub Aufbau

Das MQTT Protokoll

IBM implementierte dieses Pub-Sub Pattern Ende der 90er Jahre unter dem Namen *Message Queuing Telemetry Protocol* (MQTT). Es wurde für die Anbindung von Pipeline-Sensoren über eine unzuverlässige Satellitenleitung entwickelt. Dabei wurden soweit möglich bereits (Dhanjani 2015) standardisierte Technologien wie das TCP/IP Stack verwendet. Seit 2014 ist MQTT ein offener Standard. Da dieses Protokoll ein sehr schlankes Protokoll ist und außerdem sehr robust, eignet es sich optimal für viele Anwendungsfälle des Internets der Dinge und wird in diesem Umfeld auch gerne verwendet. Das MQTT Protokoll lässt sich verschlüsselt über das TLS (Transport Layer Security) Übertragen.

Topics

Die bereits erwähnten Topics haben in MQTT etwa folgenden Aufbau:

```
/cars/12345678/location
```

Sie folgen damit der aus dem Dateisystem eines Computers bekannten Hierarchie.
Ferner können Platzhalter (Wildcards) eingesetzt werden, um Interesse an einer Gruppe von Topics anzumelden

- # Multilevel Wildcard: `devices/#` → Alles nach dem # wird abboniert.
- + Singlelevel Wildcard devices/12345678/+ → Alles auf diesem Level wird abboniert.

Authentifizierung

Authentifizierung kann sowohl über ein Benutzername & Passwort oder über Zertifikate erfolgen.

Autorisierung

Ein weiterer Punkt der Absicherung von MQTT Verbindungen ist ein strenges Rechtekonzept. Es muss klar definiert werden, wer welches Topic abonnieren darf und wer an welches Topic senden darf. Dieser Mechanismus ist in nicht Teil des Protokolls selbst, sondern ist von der Implementierung des Brokers abhängig. Ein in der Industrie weit verbreiteter Broker HiveMQTT stellt ein SDK für Java Bereit mit dem eine eigene Autorisierungslogik implementiert werden können. (HiveMQ Developer Team)

```
public List<MqttTopicPermission> getPermissionsForClient(ClientData clientData) {
    List<MqttTopicPermission> mqttTopicPermissions = new ArrayList<MqttTopicPermission>();
```

```
    mqttTopicPermissions.add(
             new MqttTopicPermission(
                    clientData.getClientId() + “/#”,            // Topic
                    MqttTopicPermission.ALLOWED_QOS.ALL,        // QoS
                    MqttTopicPermission.ALLOWED_ACTIVITY.ALL)); // Publish, Subscribe, All
    return mqttTopicPermissions;
}
```

Kommandos

Aus der Anforderung **A13** ergibt sich die Notwendigkeit neben Updates auch einen Zertifikatswechsel auf dem Fahrzeug anstoßen zu können. Es ist denkbar, dass weitere Aktionen nachgerüstet werden müssen. Um nicht für jede nachgerüstete Aktion die RPC-Kommunikationsschnittstelle anpassen zu müssen ist es notwendig unterschiedliche Arten von parametrisierten Befehlen über die gleiche Schnittstelle zu übertragen. In diesem Fall hilft das sogenannte Kommando-Prinzip.

Wichtige Anmerkung:

Da ein Update ein spezifisches Kommando darstellt wird in den folgenden Kapiteln die Übertragung und Ausführung von Kommandos und nicht Updates beschrieben.

Definition:

» Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Schlange zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.« (Gamma 1998, S. 273)

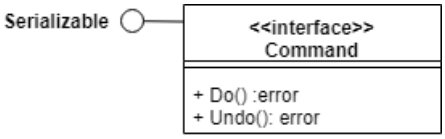

Abbildung 14 Kommandoklasse

Anstatt eine Methode eines Objekts aufzurufen. Wird dieser Aufruf in einem Objekt gekapselt, das alle zur Ausführung benötigter Parameter enthält und über eine Einheitliche Schnittstelle verfügt. Diese enthält üblicherweise Methoden zum Ausführen und Rückgängigmachung des Befehls. Wenn man darüber hinaus eine (De-)Serialisierung dieser Objekte erlaubt, kann man Kommandos erzeugen, diese über einen beliebige Kommunikationskanal versenden und an einem anderen Standort ausführen. Dies eignet sich hervorragend für unsere Zwecke. Das Backend erzeugt bei gegebenem Anlass ein Kommando zum Updaten oder ein Kommando zum Zertifikatswechsel, oder ein anderes Kommando und versendet eine serialisierte Version davon. Das Fahrzeug empfängt dies, und führt es nach einer Deserialisierung aus. An späterer

De/Serialisierung:
Serialisierung ist der Prozess der Konvertierung eines Objekts in einen Bytestream, um das Objekt zu speichern [...] Hauptzweck ist es, den Zustand eines Objekts zu speichern, um es bei Bedarf neu erstellen zu können. Der umgekehrte Vorgang wird als Deserialisierung bezeichnet. (Microsoft)

Stelle wird sich zeigen, warum das Kommandoprinzip weitere entscheidende Vorteile mit sich bringt.

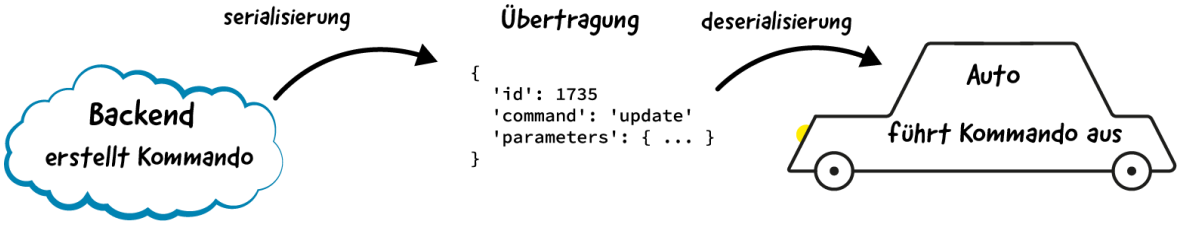

Abbildung 15 Übertragung von Kommandos

Ereignisgesteuerte Architektur

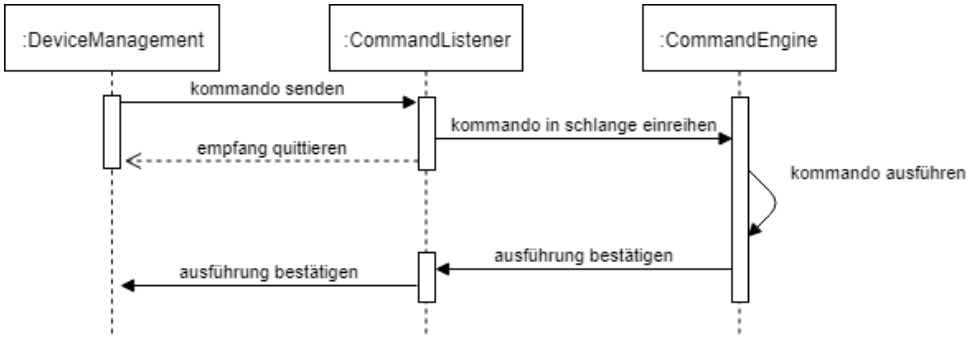

Abbildung 16 Sequenzdiagramm einer Kommandoausführung- Übertragung

Die Geräteverwaltung muss im Bilde darüber sein, ob die versendeten Kommandos erfolgreich übertragen und ausgeführt wurden. Dafür eignet sich wie in Abbildung 16 zu sehen eine ereignisgesteuerte Architektur. Kommandos werden nach dem Prinzip *Fire & Forget* versendet. Im schlimmsten Fall werden diese mehrfach versendet. Das Auto versendet dann selbstständig Benachrichtigungen über empfangene oder ausgeführte Kommandos

Der Lebenszyklus eines Kommandos hat dabei drei Schritte.

1. Nicht Übertragen

2. Übertragen

3. Ausgeführt.

Benachrichtigungen

Benachrichtigungen sind bidirektional, sie können vom Backend an das Fahrzeug gesendet werden und andersherum. Das Konzept sieht drei verschiedene Arten von Benachrichtigungen vor.

B1: Neues Kommando: Der Server schickt ein Kommando an ein Fahrzeug

B2: Änderung des Kommandostatus. Das Auto meldet zurück, dass ein Kommando entweder erfolgreich empfangen oder ausgeführt wurde.

B3: Änderung von Telemetriedaten: Das Auto meldet eine Änderung seiner Telemetriedaten

Für die Kommunikation wird das zuvor schon näher beschriebene MQTT Protokoll verwendet. Jede Benachrichtigung enthält einen eigenen Kommunikationskanal

B1: `/device/{id}/commands/statuschanged`

B2: `/device/{id}/commands/notify`

B3: `/device/{id}/telemetry/changed`

`{id}` ≙ `Fahrzeugnummer`

Autorisierung

Die Autorisierung erfolgt über Zertifikatsbasierend.

Authentisierung

Es müssen Rechte definiert werden, welche Partei welche Topics abonnieren und abhören darf (**R4**). Ist dies Logik nicht oder falsch implementiert wäre es möglich mithilfe eines gekaperten Autos andere Autos mit falschen Kommandos und daraus folgend auch Updates zu versorgen.

	Geräteverwaltung		Fahrzeug	
	veröffentlichen	abbonieren	veröffentlichen	abbonieren
B1		X	X	
B2	X			X
B3		X	X	

Kompression von Updates

Abbildung 17 JPEG Kompression (von links nach rechts nimmt die Qualität ab)

Neben der Kommunikationssicherheit zwischen Auto und dem Cloud-Backend ist die Größe der zu übertragenden Daten von Bedeutung. Da die Kommunikation zu einem großen Teil über das Mobilfunknetz abläuft, besteht ein direkter Zusammenhang zwischen der Anzahl an Versendeter / Empfangene Daten und den Kosten der Übertragung. Der Übertragungsweg ist außerdem das Nadelöhr der des Systems. Prinzipiell muss von einer langsamen und unzuverlässigen Verbindung ausgegangen werden. Ein langer Downloadvorgang verschlechtert die User-Experience für den Besitzer (**A10, A12**). Die Updates werden daher mithilfe von Kompressionsalgorithmen soweit es möglich ist in ihrer Datengröße reduziert. Prinzipiell werden diese Algorithmen in *Verlustbehaftete* und *Verlustfreie Algorithmen* unterteilt. Bei der Verlustbehafteten Kompression wird in Kauf genommen, dass sich die komprimierte Version nicht mehr Bit genau in das Original übertragen lässt. Als Beispiel hierfür kann das MP3-Format für Audiodateien genannt werden. Dieses verringert die Qualität und damit auch Größe des Originals deutlich (Bei MP3 kann die Größe bei annehmbarer Qualität auf ein Zehntel reduziert werden. Durch geschickt angewandte Algorithmen fällt dies jedoch nicht oder kaum auf. Ein anderes Beispiel ist das JPEG Bildformat. In Abbildung 17 ist eine Blume von links nach rechts mit größerer Kompression abgebildet. Je weiter nach rechts man geht, desto kleiner wird die Größe, die Qualität nimmt ebenfalls ab.

Updates jedoch bestehen Programmlogik. Jedes Bit der ausführbaren Datei ist Teil dieser Logik und darf nicht verloren gehen. Die komprimierte Datei muss wieder exakt in das Original überführt werden können. Es werden daher verlustfreie Kompressionsalgorithmus verwendet. Bekannte Vertreter dieser Art sind das ZIP oder RAR Archiv.

Wörterbuchverfahren

Ein Verfahren ist die Verwendung von Wörterbüchern. Dabei werden die Daten in öfter Vorkommende Teile, hier »Wörter« genannt, zerteilt. Diese werden in einem separaten Wörterbuch abgespeichert. Anstelle der Wörter wird nun der Index des Wortes im Wörterbuch abgespeichert. Kommen viele Daten oft hintereinander vor, so werden diese nicht hintereinandergeschrieben, sondern deren Anzahl vermerkt.

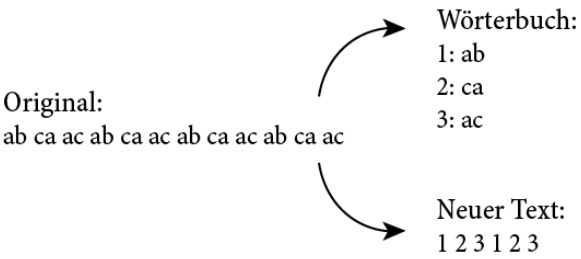

Abbildung 18 Funktionsweise Wörterbuchkompression

Deltakompression

Bei der Übertragung von Aktualisierungen erscheint es überflüssig, alle Daten erneut zu übertragen. Es müssen nur die Änderungen zwischen dem aktuellen vorhandenen und der neuen Version übertragen werden. Dieses System nennt man *Deltakompression*. In der Theorie funktioniert das bei reinen Textdateien sehr gut. Es werden Anweisungen versendet, an welcher Stelle des aktuellen Stands neue Segmente eingefügt werden müssen, beziehungsweise wo Segmente gelöscht werden müssen, um den neuen Stand zu erreichen

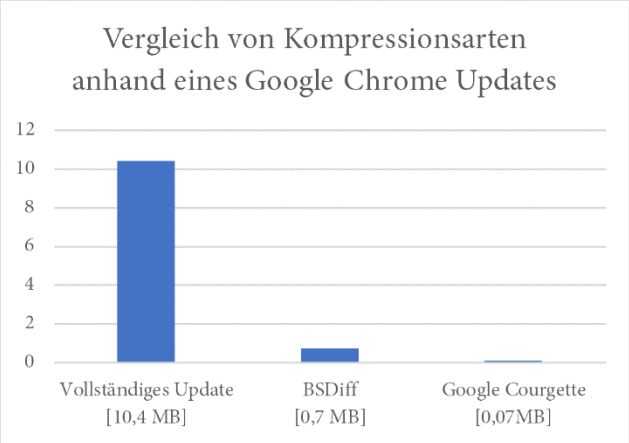

Abbildung 19 Vergleich von Kompressionsalgorithmen

Bei Softwareupdates handelt es sich aber in erster Linie um Kompilate. Das heißt Programme, die von einer Programmiersprache in ein für Maschinen lesbares Format gebracht wird. Kleine Änderungen des Quellcodes können sehr große Änderungen im Maschinencode bewirken. Zwei Doktoranden der University of Oxford nahmen sich dieses Problems (bekannt unter dem Namen Pointerproblematik) an und entwickelten BSDiff. Diese Technologie wurde auch für das ausrollen von Updates für den Chrome Browser verwendet. Das Entwicklungsteam von Google Chrome sah jedoch weiteres Optimierungspotential, dieses System noch verbessern zu können. Dabei entstand ein erweitertes Verfahren, das Courgette heißt. Ein Vergleich in

Abbildung 19 zeigt den Unterschied zwischen einem exemplarischen vollen Update, einer Deltakompression mit BSDiff und mittels Courgette. Dabei hatte BSDiff Komprimierung nur noch 6,7 % des Originals. Mittels Courgette konnte dies nochmal auf ein weiteres zehntel reduziert werden. (Stephen Adams)

Übertragung von Updates

Die Anforderung **A6** gibt vor verschiedene Arten von Updates übertragen zu können. Denkbar ist die Notwendigkeit verschiedene Geräte von verschiedenen Herstellern zu aktualisieren. Es muss eine Möglichkeit geschaffen werden beliebige Dateien und Dateiformate sicher, robust und schnell über eine einheitliche Schnittstelle übertragen zu können. Sicher bedeutet in diesem Zusammenhang vor allem, dass das Update Paket auf dem Weg zum Fahrzeug nicht verändert wurde. Anforderung **A10** besagt, dass Updates auch unter schlechten Umständen installiert werden sollen. Aus Anforderung **A8** ergibt sich die Notwendigkeit eines Ortsunabhängigen Updatevorgangs. Updates müssen auch in Gebieten mit unzureichender Internetverbindung übertragen werden können

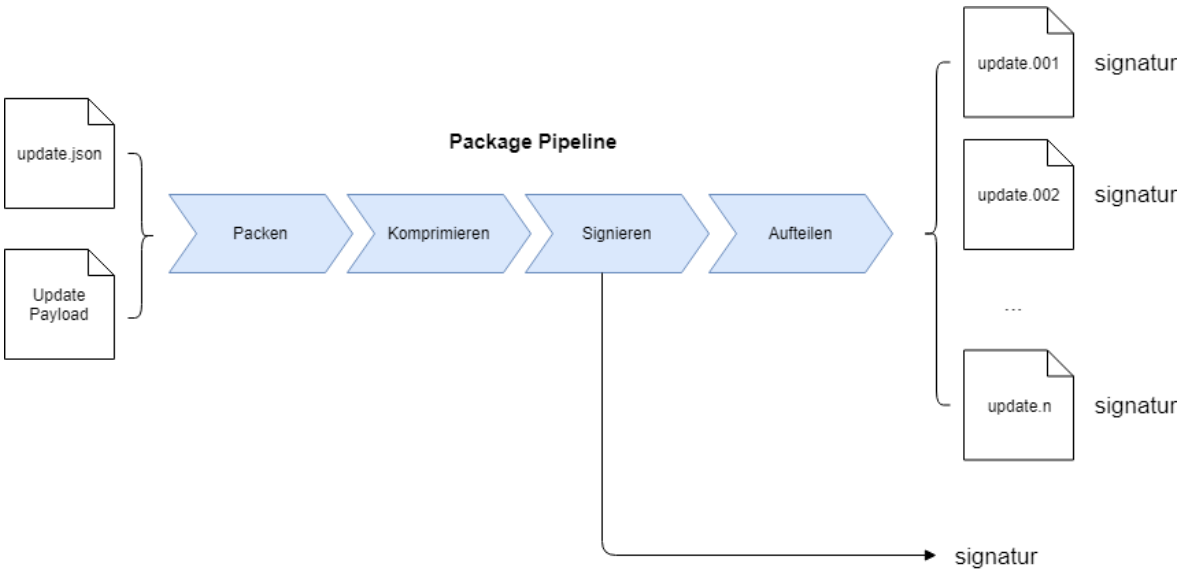

Abbildung 20 Aufbau der Package Pipeline

Um aus einem beliebigen Update ein verteilbares Update Paket zu erzeugen sind in der Entwickelten Pipeline verschiedene Schritte notwendig, die hier beleuchtet werden sollen.

Packen

Ein Update Paket kann aus mehreren Komponenten bestehen. Beispielsweise kann ein Update selbst aus mehreren Teilupdates bestehen. Diese müssen in einem Container zusammengepackt werden. Es ist auch denkbar, dass verwendete Kompressionsalgorithmen, diese Dateien auch automatisch packen und dieser Schritt entfällt.

Kompression

Als Kompression sollte ein Deltakompressionsverfahren angewendet werden. Das konkrete Testen des favorisierten Algorithmus konnte aus Zeitgründen nicht mehr im Rahmen dieser Arbeit realisiert werden.

Signieren

Um die Integrität des Pakets als zu gewährleisten, wird dieses von der Geräteverwaltung (mithilfe des privaten Schlüssels) signiert. Jedes Gerät kann die Signatur mit dem öffentlichen Schlüssel der Geräteverwaltung, der bekannt ist, überprüfen.

Aufteilen

Es muss angenommen werden, dass die primäre Verbindung zum Internet eine Mobilfunkverbindung ist. Um ein robustes System für die Übertragung von Updates zu entwickeln ist es wichtig zu wissen, wie schnell und zuverlässig diese Verbindung ist. In Deutschland werben einige Mobilfunkanbieter mit einer Netzabdeckung von 99%. Die Bevölkerung ist in Deutschland jedoch sehr ungleichmäßig verteilt. So leben 80% der Bevölkerung in nur 13% der Fläche. Die Aussage bezieht sich auf die Bevölkerungsgebiete und nicht auf die Gesamtfläche und ist daher mit Vorsicht zu genießen. Wenn man bedenkt, dass sich Automobile auf Straßen bewegen und diese (vor allem Fernstraßen) auch außerhalb von Ballungsräumen befinden ergibt sich eine andere Situation. Vor allem der Ausbau von neueren Mobilfunktechnologien wie LTE läuft im ländlichen Gebiet nur schleppend voran.

Das Magazin Connect testet jedes Jahr die Qualität der Mobilfunknetze Deutschlands, Österreichs und der Schweiz. Die Analyse soll sich hier auf Deutschland beschränken. Der Test untersucht unter anderem die Verbindungsqualität auf Fernstraßen. Die Statistik zeigt, dass in 90% der Testfahrt der Datendurchsatz schneller als 7 MBit waren. Damit ließe sich eine 100 MB Datei in 114 Sekunden herunterladen. Eine andere Untersuchung des Artikels zeigt jedoch, dass die Wahrscheinlichkeit eine Datei vollständig herunterzuladen mit steigender Größe sinkt.

(Hannes Rügheimer) Durch die relativ hohe Geschwindigkeit mit der sich Fahrzeuge bewegen wird ein häufiger Wechsel von Funkzellen notwendig. Die dadurch oftmals auch nur kurzen Verbindungsabbrüche können Downloads scheitern lassen. Durch die Tatsache, dass die Übertragung von kleineren Dateien erfolgreicher ist, macht es Sinn große Dateien zu zerteilen und diese Sequentiell zu übertragen. Diese werden in einem Cache zwischengespeichert. Bei einem ungewollten Verbindungsabbruch muss nicht wieder das gesamte Paket heruntergeladen werden. Es kann beim gescheiterten Paket begonnen werden.

Fahrzeug

Aufbau eines vernetzten Autos

Die Dichte an elektronischen Komponenten in einem Automobil hat sich in den letzten zehn Jahren nahezu verdoppelt. Dies betrifft sowohl die Anzahl als auch die Komplexität. Die nicht enden wollende Menge von Sensoren und Steuergeräten müssen miteinander kommunizieren. Dazu kommen immer datenintensivere Komponenten wie Kameras und Radarsysteme zum Einsatz. Diese setzen hohe Anforderungen an das verwendete Bus System. Heute verwendeten Standards, wie der 1991 von Bosch eingeführte CAN Bus geraten immer mehr an ihre Belastungsgrenze. [BELEG] Bei der Suche nach einem neuen Standard war es naheliegend sich bei bereits bestehenden Technologien umzusehen. Das Ethernet-Protokoll ist in der Industrie als Übertragungsstandard erprobt und weit verbreitetet. Um Anforderungen eines Fahrzeugs, wie hohe Temperaturen im Motorraum standzuhalten wurde es zu Automotive Ethernet weiterentwickelt. In Zusammenhang mit Ethernet erfolgt die Kommunikation über das IP-Protokoll. Dadurch ist die Technik nahezu identisch zu einer Netzstruktur eines Rechenzentrums. Viele Technologien, die bereits aus der Softwareentwicklung für Server im Internet bekannt sind, können nun auch im Automobil verwendet werden, ohne groß angepasst werden zu müssen.

Service Orientierte Architektur (SOA)

Dazu gehört vor allem der Einsatz einer Service Orientierten Architektur. Eine Vielzahl an Software Systemen sind heutzutage als sogenannter Monolith ausgeführt. Das gesamte System in eine einzelne stark verwebte Anwendung gegossen ist. Bei immer größer werdenden Systemen nimmt die Wartbarkeit solcher Systeme stetig ab [BELEG].

Ein heute vor allem bei großen vernetzten Architekturen ist die Service-Orientierte-Architektur (SOA).

> »Service-Orientierte Architektur (SOA) ist ein Architekturstil für Geschäftsanwendungen. In SOA wird versucht die Software direkt an den

Geschäftsprozessen einer Firma auszurichten. Dazu wird das System in so genannte Dienste (Services) unterteilt. Dienste sind kleine, lose-gekoppelte und eigenständige Softwarekomponenten. Durch das Kombinieren dieser Dienste entsteht ein Anwendungssystem, welches leicht anpassbar und änderbar bleiben soll. […].[BELEG]«

Die Aufteilung und Entkopplung bieten entscheidende Vorteile. So ist es einfacher kleine Komponenten zu Programmieren und diese zu warten. Dadurch, dass die Komponenten nicht mehr im gleichen Prozess laufen, können diese auch in Unterschiedlichen, zum jeweiligen Aufgabengebiet passenden Technologien, implementiert werden. Für einen flüssigen und Störungsfreien Updatevorgang ist diese granulare Teilung unerlässlich. Die Anforderung **A11** besagt, Updates sollen möglichst ohne Neustart oder Anhalten des Geräts erfolgen. Ein monolithisches System kann nur in seiner Gesamtheit aktualisiert werden, selbst wenn nur kleine für das System unbedeutende Teile davon betroffen sind. Kleine lose Komponenten können bei laufendem Betrieb ausgewechselt, ohne die Stabilität des Gesamtsystems zu schwächen. Ein für Updates unerheblicher aber interessanter Vorteil ist die Schaffung von Redundanz. Es können immer mehrere Instanzen eines Service auf verschiedenen Geräten gestartet sein. Im Falle eines Ausfalls, kann der Ersatzservice direkt einspringen. Dadurch kann eine höhere Ausfallsicherheit erreicht werden. In Abbildung 21 und 22 wird schemenhaft gezeigt, welche Services in einem Auto existieren könnten und wie ein System als Monolith bzw. SOA aussehen könnte.

Abbildung 22 Aufbau Softwaremonolith

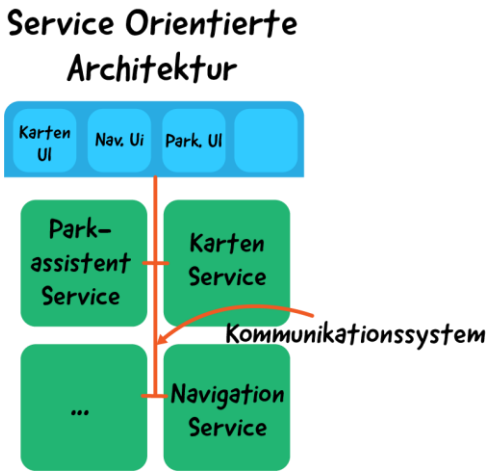

Abbildung 21 Aufbau Service Orientierte Architektur

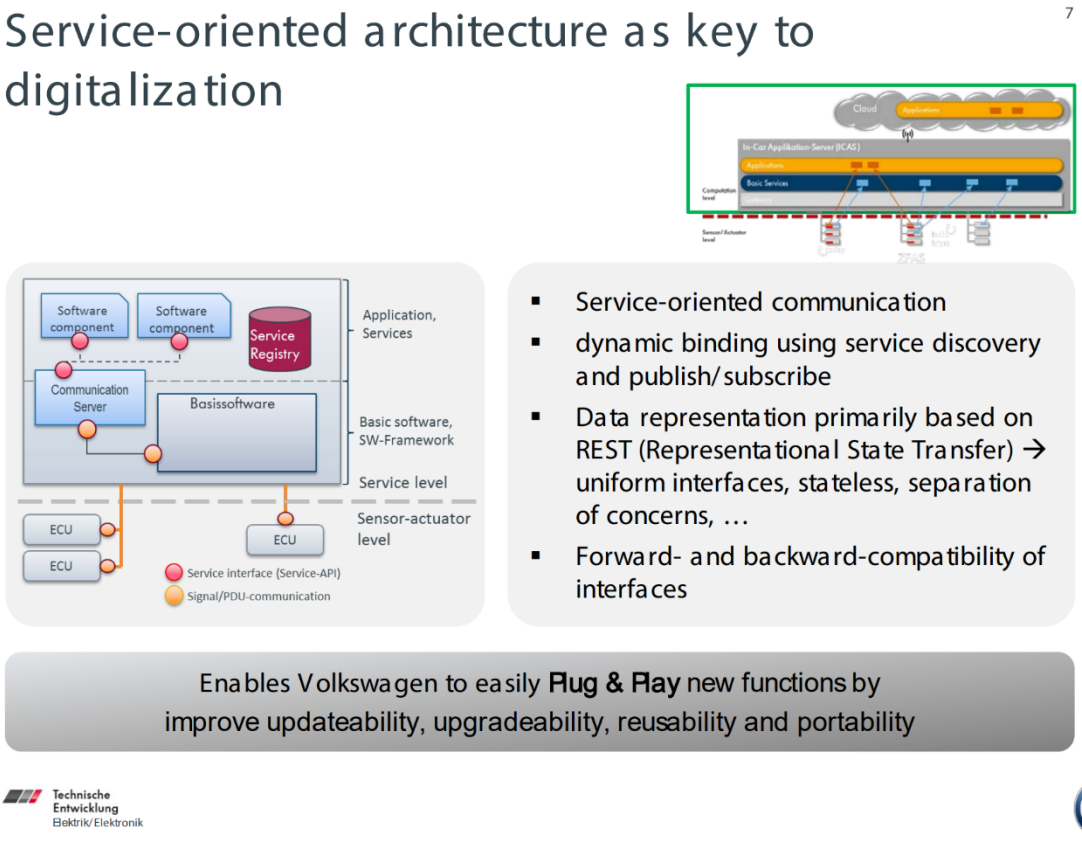

Abbildung 23 Einsatz von SOA bei Volkswagen

Abbildung 23 zeigt eine Folie der Technischen Entwicklung von Volkswagen, die SOA als Schlüssel zu einer erfolgreichen Digitalisierung von Fahrzeugen sehen.

Virtualisierung

Bei vernetzten Systemen ist es ebenfalls üblich, diese Services nicht im Userspace des Betriebssystems zu betreiben, sondern voneinander abgeschottet durch sog. Containern oder Virtuellen Maschinen (VM). Diese schaffen standardisierte stabile immer gleiche Laufzeitumgebungen für Anwendungen egal wie das eigentliche System aufgebaut ist. Damit kann dessen Funktionsfähigkeit garantiert werden. Ein weiterer Vorteil ist die dadurch erreichte effektive Rechte- / Zugriffsverwaltung. Jeder Container / VM ist eine abgekapselte Sandbox. Ähnlich wie Schotten eines Schiffes den Einbruch von Wasser auf einen

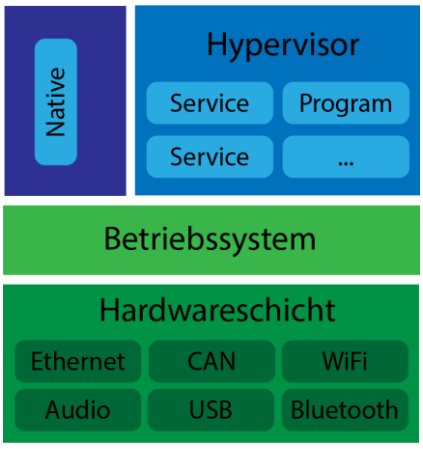

Abbildung 24 Schematischer Hardware / Softwareaufbau einer Komponente in einem Auto

lokalen Bereich begrenzen sollen, kann eine kontaminierte Komponente nicht aus der eigenen Sandbox ausbrechen. Für dieses Abschotten gibt es zwei verschiedene Realisierungsoptionen.

1. **Virtuelle Maschinen**
   Dabei wird eine komplette Hardware inklusive Betriebssystem simuliert. Vorteile sind die absolute Trennung von Ressourcen. Nachteile sind der höhere Overhead durch das simulierte Betriebssystem

2. **Container**
   Alle »Container« werden auf dem gleichen System ausgeführt, durch Methoden, die des Kernels des Linux Betriebssystems bereitstellt, können diese effektiv voneinander getrennt werden. Vorteile sind der geringere Overhead aufgrund der gemeinsam genutzten Ressourcen. Dafür muss eine nicht so strikte Trennung der Ressourcen in Kauf genommen werden.

Abbildung 24 zeigt den Schematischen Aufbau einer Head Unit eines Automobils, der so oder ähnlich bereits eingesetzt wird.

Technologien von externen Firmen wie etwa Apples CarPlay oder Android Auto können in einer virtuellen Maschine ausgeführt werden. Wird nun eine Schwachstelle in den externen Systemen entdeckt, so kann maximal Kontrolle über die Sandbox nicht jedoch über das gesamte System erlangt werden. (Green Hills Software) Für einen weiteren Überblick über Service Orientierte Architektur im Automotive Umfeld ist die Schrift Opening up New Fail-safe Layers in Distributed Vehicle Computer Systems (Johannes Büttner, Markus Kucera, Thomas Waas) zu empfehlen.

Updatevorgang

Updates sind Kommandos

Das Konzept besagt, dass es sich bei Updates lediglich um spezialisierte Kommandos handelt. Serialisierte Kommandoobjekte zu übertragen zeigt einen weiteren sehr wichtigen Vorteil für das Aufbauen eines robusten Systems. Kommandos müssen nicht sofort ausgeführt werden, sondern können zwischenzeitlich persistiert werden und zu einem späteren Zeitpunkt wieder hergestellt werden. Dies spielt vor allem dann eine Rolle, wenn das System unerwartet beendet wird. Für das Erreichen eines Resilienten (Belastbaren) Systems (**A13**) ist das unerlässlich.

Updates sind Kompositionen

Als Komposition versteht man eine baumähnliche Datenstruktur. Dabei können Elemente selbst aus beliebig vielen Kind Objekten zusammengesetzt sein. Das Konzept sieht vor, dass ein Updatevorgang aus mehreren kleineren Updatevorgängen bestehen kann. (**A23**).

Eine Aktualisierung auf eine neue Hauptversion beispielsweise wird sicherlich Updates für mehrere Peripheriegeräte eines Fahrzeugs besitzen. Hier trifft das Konzept der Komposition zu. Ein einzelner Updatevorgang selbst eine ist eine Komposition von Kommandos: Update herunterladen → Update entpacken → Update überprüfen → Update installieren. Die Abbildung 25 zeigt schematisch eine derartige Objektstruktur.

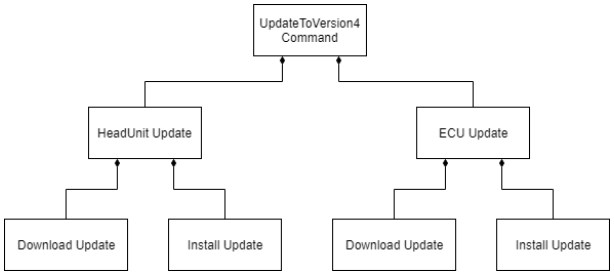

Abbildung 25 Struktur einer Komposition

Updates sind deklarativ

Updates enthalten eine Reihe von deklarativen Aussagen, welcher Zielzustand auf dem Auto erreicht werden soll, und welche Voraussetzung für die Ausführung vorhanden sein müssen.

Ausführung der Kommandos

Alle Kommandos werden vom Fahrzeug empfangen, persistiert und in einer FIFO-Schlange eingeordnet. Die zuerst Empfangenen sollen damit auch zuerst abgearbeitet werden. Eine Endlosschleife (Resilienzschleife genannt) arbeitet dies Schlange ab- Resilient bedeutet in diesem Fall Widerstandsfähig. Durch diese Endlosschleife wird sichergestellt, dass Kommandos in jedem Fall, auch im Falle einer Systemstörung irgendwann ausgeführt werden und

ein definierter Zielzustand erreicht wird.

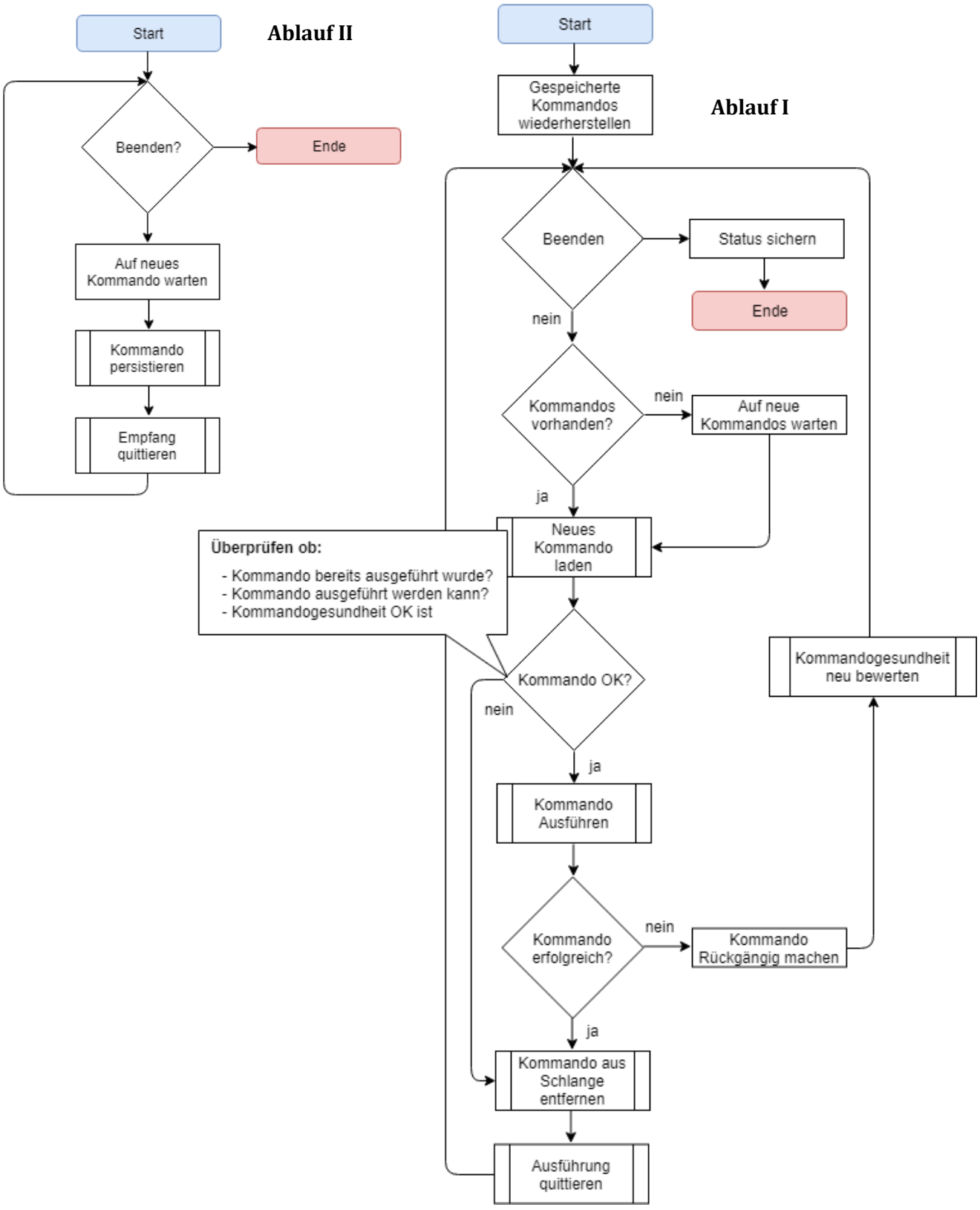

Abbildung 26 Ablaufplan der Resilienzschleife

In Abbildung 26 sind nun schematisch die zwei Abläufe des Systems abgebildet. **Ablauf II** wartet permanent auf das Eintreffen neuer Kommandos und sichert diese sofort. Dadurch gehen diese bei Fehlern des Systems nicht verloren. Der Empfang wird durch eine Rückmeldung an die Geräteverwaltung quittiert. **Ablauf** I zeigt wie diese Kommandos abgearbeitet werden. Nach dem Hochfahren des Systems werden alle bereits persistierten Kommandos wiederhergestellt. Anschließend wird die Hauptschleife gestartet. Diese wird bis zum herunterfahren des Systems nicht beendet. Anschließend wird sequenziell überprüft ob Kommandos zum Ausführen bereitstehen. Falls nicht wird gewartet. Steht ein Kommando zur Ausführung bereit wird geprüft, ob dieses *(1)* bereits ausgeführt wurde und *(2)* ob es ausgeführt werden kann und *(3)* ob dessen »Kommandogesundheit« in Ordnung ist.

Die Fehlerfreie Übertragung von Paketen sagt noch nichts über die Korrektheit der Logik des Kommandos aus. Für den Fall, ein nicht ausführbares Kommando wird im einfachsten Fall definiert, wie oft ein Update fehlerhaft ausgeführt werden darf, bevor es aus der Warteschlange entfernt wird.

Sind alle Tests erfolgreich wird das Kommando ausgeführt. Treten dabei Fehler auf, ist es wichtig, die vom Kommando ausgeführten Änderungen wieder vollständig rückgängig zu machen. (**A 13**) Erfolgreiche Updates werden aus der Warteschlange entfernt.

Architektur

In diesem Kapitel werden die für die Kommandoausführung benötigten Komponenten beschrieben.

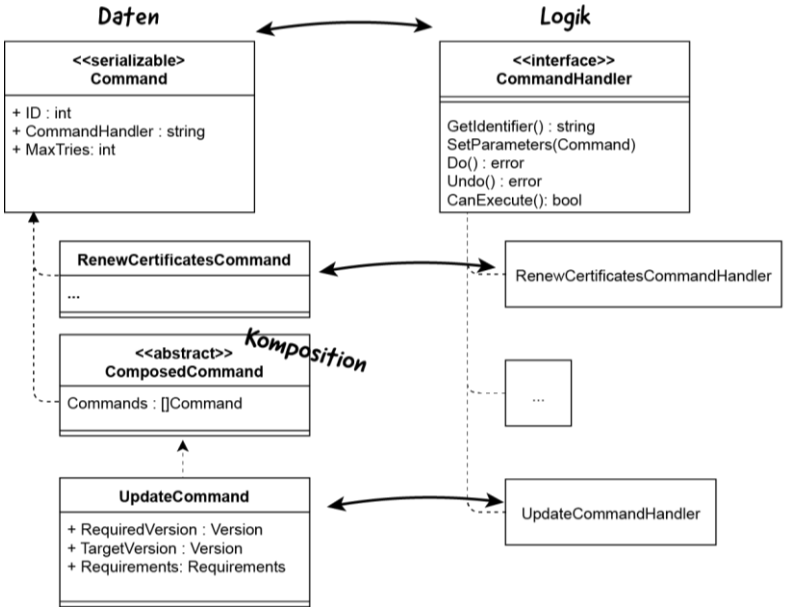

Abbildung 27 Für Update benötigte Klassen

Ein Kommando wird in zwei Komponenten aufgeteilt. Einmal die Command Klasse, diese ist serialisierbar und wird von der Geräteverwaltung an die Fahrzeuge Übertragung. Für jede Art von Command wird ein eigener CommandHandler benötigt, dieser besitzt die Logik, um eben jene Kommandos auszuführen. Um ein Kommando einem dementsprechenden Handler zuordnen zu können teilen beide einen gleichen eindeutigen Identifier. Im Konzept handelt es sich hierbei um einen String der durch das Feld *CommandHandler* der Command Klasse und durch die *GetIdentifier()* Methode der CommandHandler Klassen abgefragt werden kann.

Die Command Klasse ist eine Abstrakte Basisiklasse von der alle eigentlichen Kommandos erben müssen. Sie stellt die benötigte Basisfunktionalität bereit. Dazu gehört die Serialisierbarkeit, die Bestimmung der Maximalen Anzahl an Versuchen, mit der dieses Kommando ausgeführt werden soll, eine eindeutige ID und den CommandHandler. Durch das ComposedCommand wir die im Konzept beschriebene Komposition von Updates verwirklicht. Auf der anderen Seite gibt es die CommandHandler Schnittstelle. Damit werden Standardisierte Methoden zur Ausführung, und Rückabwicklung bereitgestellt. Durch diesen Aufbau lässt sich das System leicht um neue Kommandos und Updates erweitern. Um

Funktionalität hinzuzufügen müssen lediglich eine neue Command und eine neue CommandHandler Klasse definiert werden. In der Abbildung sind bereits *UpdateCommand* und *RenewCertificateCommand* zu sehen.

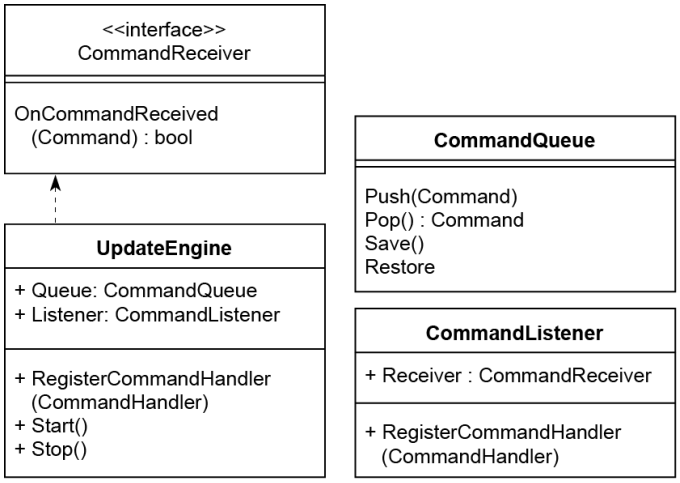

Abbildung 28 Komponenten der Resilienzschleife

Die für die Ausführung der Resilienzschleife (Abb. 26) benötigten Komponenten werden in Abb. 28 beschrieben. Die UpdateEngine ist dabei das Herzstück der Schleife, die durch dessen Start und Stop Methoden gesteuert werden können. Durch die Methode RegisterCommandHandler können neue Arten von Kommandos registriert werden. Der CommandListener registriert sich am MQTT Broker und meldet Interesse an Kommandos für das aktuelle Fahrzeug an. Empfangene Kommandos werden an die UpdateEngine weitergeleitet. Die CommandQueue verwaltet und persistiert alle empfangenen Kommandos.

Blue Green Deployment

Aus Anforderung **A14** geht hervor, dass Updatevorgänge Atomar sein müssen. Also entweder erfolgreich ausgeführt oder wieder sauber entfernt werden. Es muss zu jederzeit ein lauffähiges System bestehen. Wenn ein Updatevorgang abgebrochen wird, kann dies einen unsauberen Zustand zur Folge haben. Dieses Problem ist bei der Entwicklung von Services im Cloudumfeld bereits seit längerem bekannt. Eine Lösung nennt sich *Blue Green* Deployment. Dabei gibt es immer mindestens zwei Umgebungen. Eine Testumgebung und die Produktivumgebung. Das Update wird immer auf die gerade nicht genutzte Umgebung installiert, dann wird geprüft ob das Update funktionsfähig ist. Wenn der Test positiv verläuft wird die Umgebung »getauscht«, ansonsten wird die Testumgebung aufgeräumt und alles bleibt im Originalzustand. Ist das Update erfolgreich wird die ehemalige Testumgebung die aktive Produktivumgebung und umgekehrt. Die Schwierigkeit besteht in genau diesem »austauschen« der Umgebung. Für verschiedene Systeme existieren hier verschiedene Lösungswege.

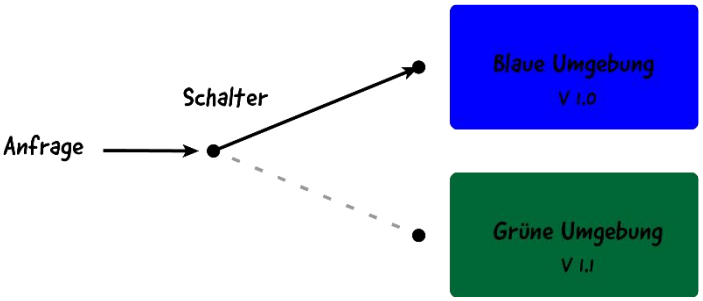

Abbildung 30 Benötigte Infrastruktur für ein Blue / Green Deployment

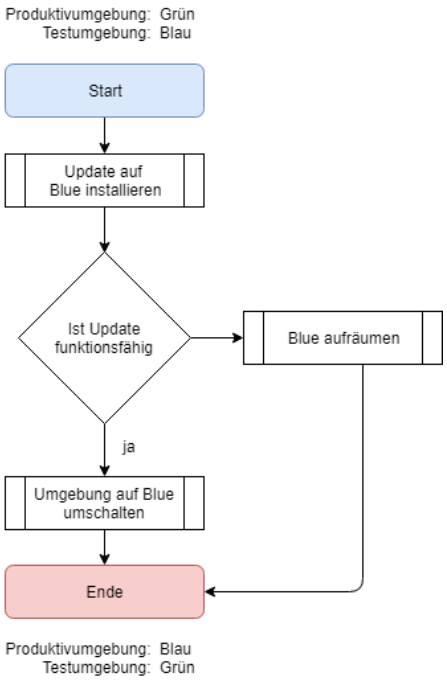

Abbildung 29 Ablaufplan eines Blue / Green Deployments

Services / Feature Updates

Im Rahmen der Service Orientierten Architektur ist die Logik des Autos in Services aufgeteilt, diese laufen virtualisiert in einem Hypervisor. Oft existieren bereits mehrere Instanzen eines Service, redundant um eine höhere Ausfallsicherheit zu garantieren. Von einem Router wird entschieden welche Instanz Anfragen entgegennehmen soll. Das macht den Updatevorgang jetzt sehr einfach. Nach einer erfolgreichen Aktualisierung eines Service werden alle Anfragen auf die neue Version umgeleitet. Der alte Stand kann anschließend entfernt werden.

Betriebssystem Updates

Dieses Kapitel bezieht sich auf das Aktualisieren von Systemen, die auf Linux artigen Betriebssystemen basieren. Head Unit Module tun das bereits meistens. Das Updaten von Motorsteuerungen oder weiteren Geräten geschieht meist durch proprietäre Techniken, die hier nicht beschrieben werden. Das Betriebssystem zu aktualisieren basiert ebenfalls auf der Blue Green Deployment Technologie. Es gibt verschiedene Möglichkeiten wie dies Realisiert werden kann:

Verwenden von zwei Partitionen

Das System verfügt über zwei Partitionen, Die neue Version des Betriebssystems wird auf die nicht genutzte Partition installiert. Falls dieser Vorgang erfolgreich abgeschlossen wurde, schaltet der Bootloader auf eben diese Partition um. Im Falle eines Fehlers bleibt die aktuelle Partition aktiv.

OS Tree

Das Verwenden von zwei Partition erzeugt einen deutlichen Overhead. Das System existiert in der Regel doppelt. Dabei wird sehr viel Speicherplatz verbraucht. Die Bibliothek OS Tree verfolgt einen anderen Ansatz. Um zu verstehen wie diese Technologie funktioniert muss man wissen was Symbolische Verknüpfungen sind. Dabei handelt es sich um einen Link, der von Programmen wie eine Datei behandelt wird, aber nur einen Verweis auf eine Datei darstellt. OS Tree ersetzt nun alle wichtigen Dateien eines Betriebssystems durch solche Links. Damit können parallel zwei Versionen eines Betriebssystems installiert werden. Es müssen lediglich die Links, die auf eine bestimmte Version zeigen verändert werden. OS Tree ist inzwischen eine etablierte Bibliothek die von vielen bekannten Softwareherstellern wie zum Beispiel QT verwendet wird.

Zertifikatsspeicher

Jedes Gerät verfügt über mindestens zwei Zertifikate, einmal den öffentlichen Schlüssel der Certificate Authority und einmal ein eigenes Zertifikat, mit dem das eigene Gerät autorisiert wird. Das unautorisierte Abgreifen oder Austauschen dieser Geräteszertifikate kann schwerwiegende Folgen haben.

1. **Auslesen des Gerätezertifikats**

   Der Angreifer hat nun selbst Zugriff auf die Infrastruktur, kann für das Auto bestimmte Informationen auslesen und im Namen des Geräts Informationen versenden. Der Angreifer kann sich als das gekaperte Auto ausgeben.

2. **Austausch des CA Schlüssels**

   Die Authentifizierung der CA erfolgt über den öffentlichen Schlüssel der CA. Wird dieser ausgetauscht. Kann sich ein Angreifer als legitimer Kommunikationspartner des Geräts ausgeben. Dadurch können illegitime Updates ausgeführt werden.

Die benötigten Zertifikate sollten daher nicht im gleichen, sondern seperaten Speicherbereich wie die anderen Programme gesichert werden. Dieser kann als Read-Only Speicher ausgeführt sein. Eine andere Möglichkeit ist der Einsatz eines Trused Platform Modules.

Trusted Platform Module

Ein Trusted Platform Modul stellt einen sicheren Hafen für kryptologischen Methoden eines Systems dar. Üblicherweise handelt es sich um ein spezielles Hardwaremodul, das Zertifikate speichert und Ent-/Verschlüsselungen von Inhalten durchführt. Durch das Auslagern aller sicherheitskritischen Prozesse auf eine extra dafür konzipierte Hardware kann die Sicherheit des Gesamtsystems erheblich gesteigert werden.

Prototyp

Umfang der Implementierung

Ziel der Implementierung ist es, die Funktionsweise des Konzepts exemplarisch darzustellen. Aufgrund des großen Umfangs wurden viele Faktoren, die in einer real existierenden Umgebung eine Rolle spielen würden, vernachlässigt.

- **Bandbreite:**
  Die Systeme sind nicht darauf ausgelegt mehrere tausend Geräte gleichzeitig und performant zu beantworten. Sie sind daher auch nicht skalierbar.
- **Sicherheit:**

  Das Zertifikatsmanagement ist exemplarisch und nicht vollständig implementiert. Das System erhebt nicht den Anspruch auch tatsächlich absolut sicher zu sein.
- **Endgerät:**

  Das Endgerät ist kein tatsächliches Auto, sondern eine Attrappe. Auf dieser läuft eine Beispielanwendung, die eine Oberfläche einer Head-Unit simuliert.
- **Zertifikate**:

  Verschlüsselte Übertragungen und Zertifikatsbasierte Authentifizierung findet nicht auf allen Kanälen statt.

Überblick

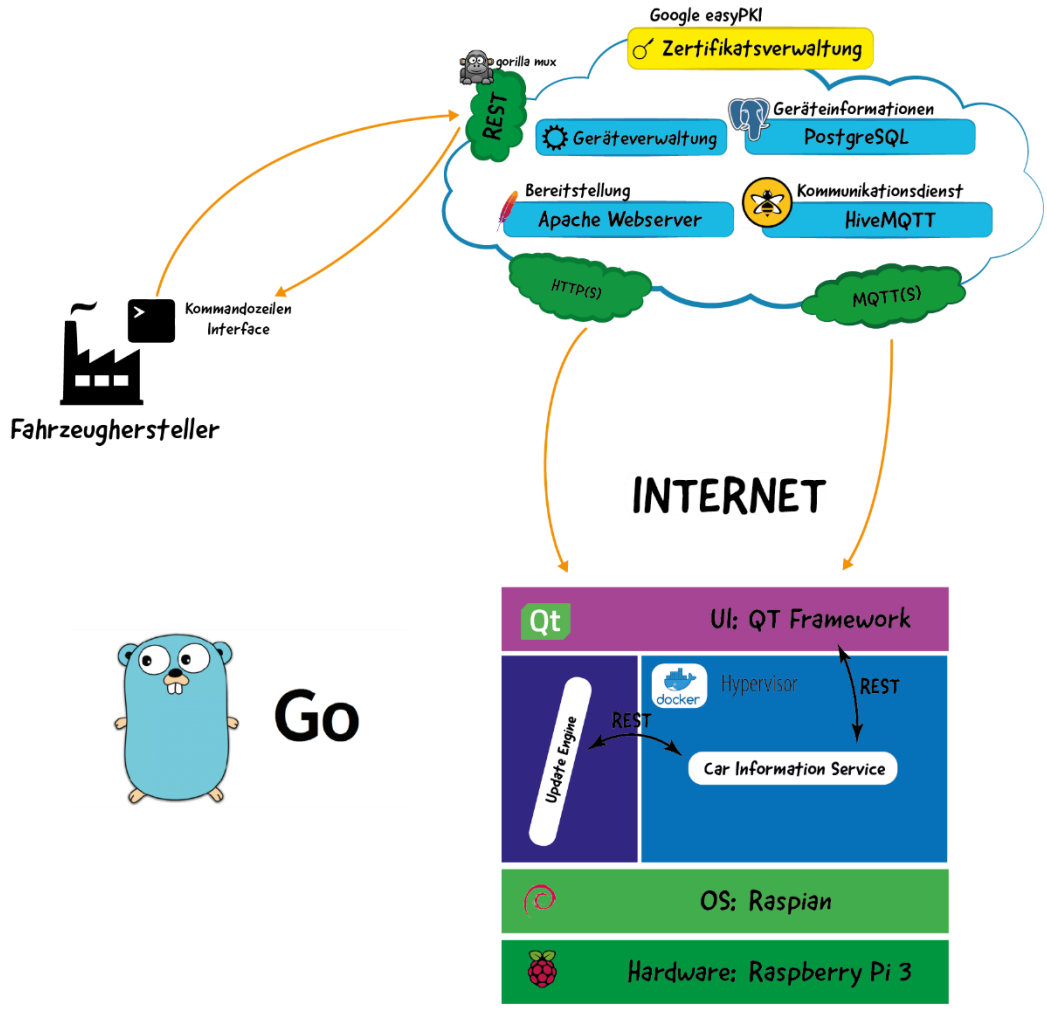

Abbildung 31 Überblick über die in der Implementierung verwendeten Komponenten

Hardwareaufbau

Das Endgerät selbst, die Head Unit eines Automobils wird durch einen Raspberry Pi 3 B+ in Verbindung mit einem 7“ Touchscreen simuliert. Der Raspberry Pi verfügt über einen WLAN Adapter um mit dem Internet verbunden werden zu können. Die Geräteverwaltung inklusiver Benutzeroberfläche wird auf einem Windows Computer installiert. Als „Hypervisor“ wird Docker verwendet.

Netzwerktopologie

Das »Fahrzeug« und die Geräteverwaltung befinden sich im gleichen Netzwerk. Eine Verbindung zum Internet ist daher nicht nötig. Sowohl das »Auto« als auch die Geräteverwaltung erhalten eine feste IP-Addresse:

```
Geräteverwaltung:  192.168.178.1
Auto:              192.168.178.2
```

Auf einen DNS Server, der die Adresse der Geräteverwaltung dynamisch auflöst wird verzichtet. Das »Auto« kommuniziert immer mit dieser festen IP Adresse. Auf eine detaillierte Einrichtung von Firewalls mit Portfreigaben, usw. wurde aus Zeitgründen verzichtet.

Sicherheit in der Softwareentwicklung

In diesem Kapitel soll kurz angerissen werden, welche Möglichkeiten es gibt Risiken bei der Entwicklung von Software zu vermeiden und handzuhaben. Weiter werden Best Practices im Umgang mit Kryptographischen Komponenten gezeigt.

Softwaretests

Der Reibungslose Ablauf von Software muss sichergestellt werden, dabei gibt es verschiedene Wege, die sich im Laufe der Zeit entwickelten. Der einfachste Weg ist das entwickelte System einfach über die Oberfläche auszuprobieren. Dabei wird jedoch oft nicht der gesamte Umfang des Programms getestet. Unit Tests (dt. Modultests) greifen tiefer. Für jedes Softwaremodul werden Tests implementiert, diese sind selbst Code, der anderen Code auf bestimmte funktionsweisen überprüft. Das Modul stellt eine Blackbox dar. Aus den Anforderungen ist bekannt, wie sich dieses System verhalten soll. Also welches Ergebnis erwartet wird, wenn bestimmte Methoden mit bestimmten Parametern aufgerufen werden. Unit Tests klopfen so die gesamte Funktionalität des Systems ab. Unit Tests überprüfen nur einzelne Softwarekomponenten eines Systems. Vorteil ist auch die Testbarkeit von Code bereits während der Entwicklung, bevor das Gesamtsystem überhaupt ausführbar ist. In viele Projekten ist eine bestimmte Testabdeckung in Prozent (wie viel Code wird durch Tests abgedeckt) verpflichtend.

Modularer Code

Die Implementierung wurde modular aufgebaut. Der Softwarecode ist in logische Teile untergliedert. Ein Modul ist beispielsweise die Datenpersistenz. Durch das abkapseln dieser Module durch Schnittstellen, können diese auf leichte Art und Weise ausgetauscht werden. Einzelne Module können sowohl dann ebenfalls zwischen Serversystem oder den einzelnen Fahrzeugen getauscht werden.

Wahl der Komponenten

Oftmals stellt sich die Frage, ob eine Komponente selbst implementiert werden muss oder auf bereits am Markt existierende Komponenten zurückgegriffen werden kann. Dieses ist oft kostengünstiger, wenn nicht sogar kostenlos. Meistens ist es besser sich auf eine bereits etablierte und getestete Komponente zurückzugreifen, oft ist auch nicht genügend Manpower für derartige Unterfangen verfügbar.

Man muss sich allerdings bewusst darüber sein, dass diese externen Komponenten, gewollt oder ungewollt, Schwachstellen des eigenen Systems darstellen können. Ein Beispiel ist der Heartbleed Bug der OpenSSL Bibliothek. Viele Anwendungen setzten OpenSSL zur Abwicklung von TLS Verbindungen ein. Der Bug ermöglichte es Angreifern unerlaubt Speicherinhalte des Servers auszulesen. Darunter befanden sich sensible Daten, wie Passwörter oder auch Zertifikate. Die Wahl von Open Source Projekten bringt den Vorteil, die korrekte Funktionsweise im Zweifelsfall selbst überprüfen zu können.

Wahl der Entwicklungstechnologie

Es gibt eine große Anzahl an Programmiersprachen, die alle für unterschiedliche Einsatzgebiete optimiert wurden. Die Firma PENTASYS AG, die meine Bachelorarbeit betreute, hatte ein Interesse daran die Programmiersprache Go der Firma Google auf Zweckmäßigkeit für diese Art von Anwendung zu überprüfen. [GO verfügt bereits über die richtigen Tools]

Geschichte

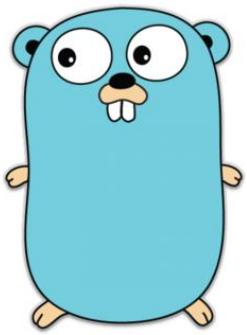

Abbildung 33 Gopher, das Maskottchen der Programmiersprache Go

Google war Ende der 2000er Jahre unzufrieden mit den am Markt existierenden Programmiersprachen. Man erwartete von einer Programmiersprache, dass diese einfach zu erlernen ist und, über eine konsistente leichte Syntax verfügt, wie etwa die schwach typisierte Programmiersprache Python. Schnell zu kompilieren ist und ressourcenschonend arbeitet wie die Programmiersprache C. Und über eine sichere etablierte Laufzeitumgebung wie Java verfügt. Ein Nachteil von älteren Sprachen, ist deren Schwäche bei Nebenläufigkeit, die bei Erscheinen dieser oft noch keine Rolle spielte. Alle Features zur Nebenläufigkeit wurden erst nachträglich eingefügt worden. Bei GO wurde dies direkt bei Entwicklung der Sprache bedacht. GO soll vor allem für die Entwicklung von skalierbaren Netzwerkdiensten fungieren. Mit Rob Pike und Ken Thompson, die bereits am Betriebssystem Unix mitgeschrieben haben sind zwei sehr erfahrene Informatiker an der Entwicklung der Sprache dabei.

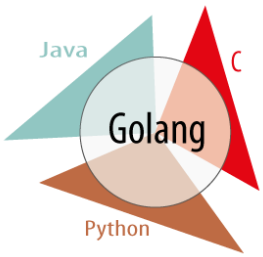

Abbildung 32 Eigenschaften im Vergleich zu anderen Programmiersprachen

Sprachfeatures

GO ist eine stark typisierte Programmiersprache. Da bei der manuellen Speicherverwaltung wie es bei C / C++ der Fall ist häufig Buffer Overflow Fehler auftreten können, verfügt GO über einen *Garbage Collector*. Um das Arbeiten in großen Teams zu erleichtern minimiert GO die Keywords der Sprache auf ein Minimum. Dadurch gibt es nur einen Weg wie ein Problem gelöst werden kann, Code unterschiedlicher Entwickler sieht damit meistens gleich aus.

Umfangreiche Standardbibliothek

Mindestens genauso wichtig wie die Programmiersprache ist die enthaltene Standardbibliothek. Darunter versteht man den Funktionsumfang, der ohne installation von Software dritter zur Verfügung steht. Hier kann GO enorm davon profitieren, dass Google hinter der Entwicklung steckt. Die Standardbibliothek ist umfangreich und gut getestet. Kryptologische Methoden die für einen Aufbau von TLS Verbindungen oder für den Umgang von Zertifikaten notwendig sind, existieren bereits. Die gesamte Bibliothek ist Open Source. Die Programmiersprache wird neben Google auch von weiteren namhaften großen Firmen (Microsoft, etc.) intensiv genutzt. Diese tragen auch zur Standardbibliothek bei (Alessandro Segala).

Portabilität

Code, der mit GO geschrieben wurde besitzt eine hohe Portabilität. Dieser kann sowohl für x86, x64, ARM und viele weitere verbreitete Befehlssätze kompiliert werden. Daraus ergeben sich eine Reihe von Vorteilen. Der erste ist die Wiederverwendbarkeit von Code. Datenstrukturen und Logik können sowohl auf Server als auch auf Clientseite eingesetzt werden. Dadurch kann Code zwischen Geräteverwaltung und Fahrzeug geteilt werden. Dies hat den Vorteil, dass (1) doppelter Code vermieden werden kann und dadurch Fehler vermieden werden können. Und (2) die Enwicklungsteams die gleichen Technologien verwenden und dadurch besser zusammen arbeiten können.

Ausführungsumgebung

Viele etablierte Programmiersprachen benötigen eine sehr große Laufzeitumgebung. Die in der Industrie weit verbreitete Programmiersprache Java benötigt zur Ausführung die »Java Runtime Environment« mit einer Größe von über 200 MB (eigene Recherche). Für Umgebungen wie Server oder Heimrechnern, bei denen die Ressourcen großzügig bemessen sind ist das keine Einschränkungen. Vor allem im Embedded-Bereich sind die Ressourcen jedoch knapp bemessen. Go verfügt über das Konzept der *Single Binary Executable*, dabei werden alle benötigten Komponenten in eine Datei kompiliert. Eine weitere keine weiteren Abhängigkeiten oder eine Laufzeitumgebung notwendig.

Geräteverwaltung

Datenpersistenz

Um die Daten der Geräteverwaltung zu persistieren wird eine Datenbank benötigt. Als erstes muss die Prinzipielle Art der Datenbank festgelegt werden. Man unterscheidet prinzipiell zwischen NoSQL und SQL Datenbanken. Der große Vorteil von NoSQL Datenbanken ist die Skalierungsmöglichkeit. Ab einer bestimmten Menge an Nutzern und Daten lassen sich SQL Datenbanken nicht mehr sinnvoll verwenden. Nachteil ist die schwierigere Handhabung und Abfragemöglichkeiten. Da es sich bei der Prototyp Implementierung um ein Proof-Of-Konzept handelt, besteht keine hohe Durchsatzrate und es wurde auf eine SQL Datenbank gesetzt. Das im Konzept gezeigte Datenmodell lässt sich jedoch auch bei NoSQL Datenbanken umsetzen. Voraussetzungen waren, dass die gewählte Datenbank von GO direkt angesprochen werden kann und diese kostenlos Verfügbar ist. Als SQL Datenbank wurde PostgreSQL gewählt, diese ist kostenlos verfügbar. Go verfügt über eine native Unterstützung und PostgreSQL wird auch in größeren Industrieprojekten eingesetzt. Um mit der Datenbank zu kommunizieren wurde ein Object-Relation-Mapper eingesetzt. [Gorm] Dieser erspart das Schreiben von fehleranfälligen SQL-Abfragen, sondern ermöglicht einen Objektorientierten Datenzugriff. Da die SQL Queries nicht mehr selbst geschrieben werden, können auch SQL-Injections (R1) effektiv vermieden werden. Im Zuge des Modularen Aufbaus des Softwarecodes wird von anderen Modulen nicht direkt auf die Datenbank zugegriffen. Vielmehr werden alle Methoden, um die Daten zu verwalten in einem Modul ausgelagert. Dabei handelt es sich um ein Repository. (Edward Hieatt and Rob Mee) Durch diese Kapselung lässt sich die konkrete Verwendete Datenbank zu einem späteren Zeitpunkt noch leicht geändert werden. Für Modultests kann eine andere Datenbank verwendet werden, um Testdaten in einer Produktionsdatenbank verhindern zu können. Die Schnittstelle, mit der auf Daten zugegriffen werden kann sieht folgendermaßen aus:

```
type Repository interface {
  GetAll(out interface{})
  Get(id int, out interface{})
  AddOrUpdate(model entities.Entity) error
  Add(model interface{}) error
  Update(model entities.Entity) error
  First(out interface{}, predicate interface{})
  GetWhere(out interface{}, predicate interface{})
  Delete(model entities.Entity)
  Close()
  CreateSchema()
  IsSchemaCreated() bool
}
```

Das Repository stellt dabei die typischen CRUD (*Create* - Erzeugen, *Read* - Lesen, *Update* - Aktualisieren, *Delete* - Löschen) Methoden bereit die für Datenbanken typisch sind. Daneben eine Reihe von Helfermethoden, wie *AddOrUpdate*, mit der eine Entität erzeugt, aktualisiert wird, falls sie bereits existiert.

Alle Entitäten (so werden die Objekte der Datenbank genannt) die in einer SQL Datenbank persistiert werden sollen implementieren die gleiche Schnittstelle

```
type Entity interface {
  Validate() error
  GetID() uint
}
```

Durch die *Validate* Methode können Daten auf ihre Gültigkeit überprüft werden und nur bei dann gesichert werden. Dadurch werden ungültige Daten in der Datenbank vermieden. Eine SQL Datenbank benötigt einen eindeutigen Primärschlüssel. Dieser ist in der Implementierung eine inkrementierende positive Ganzzahl (*unsigned int*). Über *GetID()* kann diese Abgefragt werden.

REST API

In diesem Kapitel werden übliche Techniken, die bei der Implementierung von REST APIs zum Einsatz kommen besprochen, und welche Komponenten in Go dafür eingesetzt werden können.

Model View Controller

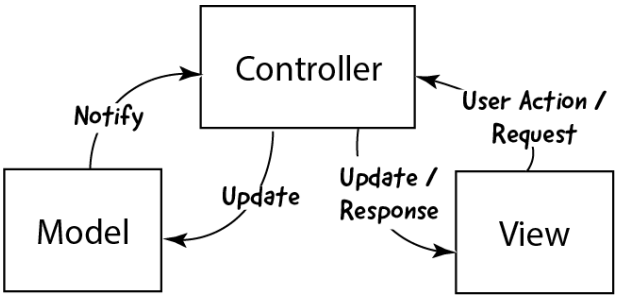

Abbildung 34 Model View Controller Struktur

Üblicherweise wird das Architekturprinzip Model-View-Controller (MVC) für derartige Anwendungen verwendet. Dahinter verbirgt sich die Idee, ein großes Programm in kleinere Komponenten aufzuteilen, um dieses besser handhaben zu können (*Seperation of Concerns*).

Bestandteil	Bedeutung	Bedeutung bei REST APIs
Model:	Die eigentliche Logik und Daten etc.	Logik / Daten aus Datenbank, etc
View:	Oberfläche	Oberfläche „entfällt“, es handelt sich hierbei um serialisierte Daten im JSON Format.
Controller:	Steuert die Oberfläche. Setzt Aktionen der Oberfläche in Logik um und leitet deren Ergebnisse an Oberfläche weiter.	Behandelt HTTP Requests und wandelt diese in entsprechende Logik um.

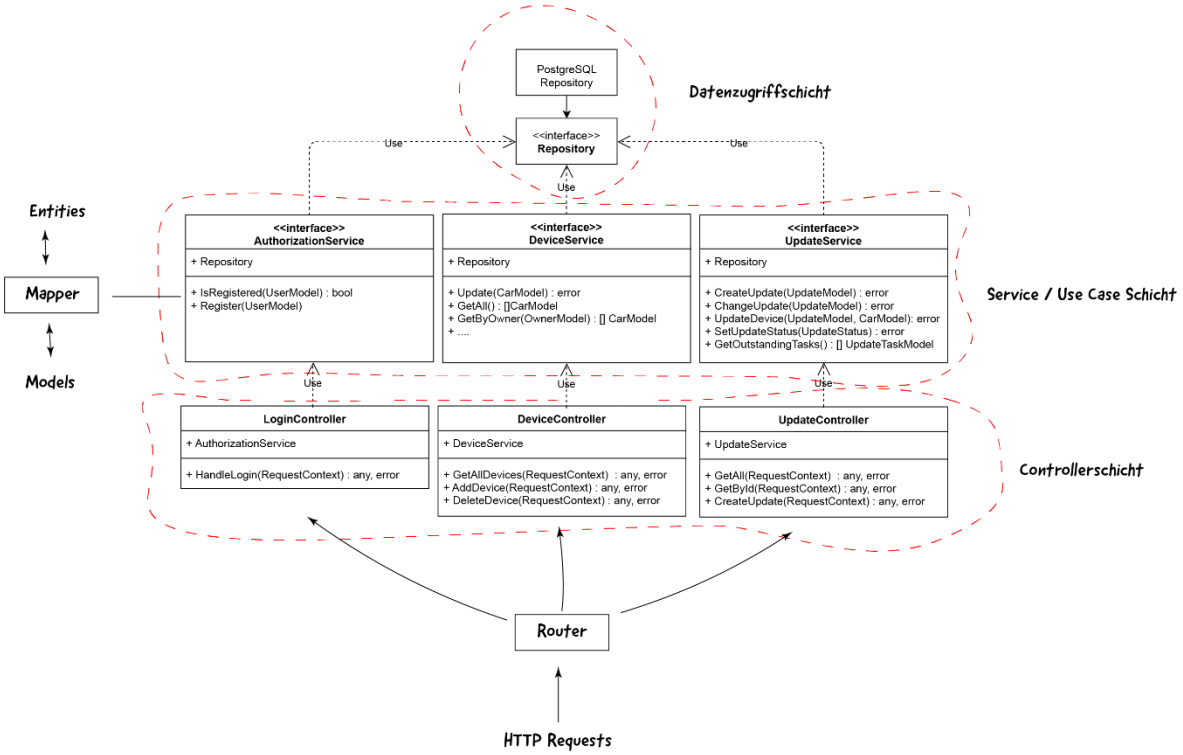

Abbildung 35 Aufbau der Rest Api Implementierung

Architektur

In Abbildung 35 ist die Kernarchitektur der API zu sehen. An der Spitze steht die Datenzugriffsschicht. Es handelt sich dabei um das bereits beschriebene Repository. Die gesamte Logik, die im MVC Muster dem Model entspricht, wurden in Services gekapselt. Dementsprechend existiert ein Service für die Benutzerverwaltung, einen für die Geräteverwaltung und einen für die Authentifizierung / Autorisierung. Weitere Services wurden aus Platzgründen nicht dargestellt. Die Aufgabe der Controllerschicht ist es, die ankommenden HTTP Anfragen so zu übersetzen, dass Sie von der Logikschicht auch verstanden werden. Für jede dokumentierte REST Methode (URL) existiert eine korrelierende Methode in einem entsprechenden Controller. Konkret bedeutet dies, dass die in der Anfrage enthaltenen Daten in eine für die Logikschicht verstandenes Datenobjekt übertragen wird. Nach der Ausführung der Logik wird die Antwort wiederum in eine Antwort im HTTP Response Format übersetzt. Die Controller sind damit die Schnittstelle zwischen der internen Logik der Anwendung (den Services) und dem HTTP Protokoll. Der Router steht noch vor den Controllern. Dieser entscheidet anhand der URL welcher Controller mit welcher Methode die angegebene URL verarbeiten soll. Die Authentifizierung wird bereits im Router erledigt. Um eine hohe Testbarkeit **A24** zu erreichen, ist die Datenzugriffsschicht und die Serviceschicht als

Schnittstelle definiert. Die eigentlich verwendete Implementierung wird erst zur Laufzeit festgelegt. Dadurch lässt sich die echte Datenbankimplementierung durch einen Mock (einen Dummy) austauschen und die Services können getestet werden, ohne die echte Datenbank zu berühren. Auch kann überprüft werden ob die Controllerschicht die Anfragen richtig behandelt.

Die Serviceschicht bildet auch die Grenze mit welchen Datenobjekten gearbeitet wird. Es gibt den Grundsatz in der Softwareentwicklung niemals Datenobjekte der Datenbank (Entitäten) direkt an den Benutzer weiterzugeben oder andersherum. Zu groß ist die Gefahr, dass ein absichtliches manipulieren dieser Objekte unbemerkt bleibt. (**R7**). Daher wird nach außen mit Models, die im Konzept bereits beschrieben wurden. In der Serviceschicht konvertiert eine als Mapper bezeichnete Komponente Models in Entitäten und zurück.

Gorilla Mux

Komponenten wie der Router, der Anfragen anhand der URL an entsprechende Controller weiterleitet sind nicht Teil der Standardbibliothek und mussten daher durch eine externe Softwarekomponente nachgerüstet werden. Das Gorilla Web Toolkit ist ein weitverbreitetes Leichtgewichtes Framework für GO für Webtechnologien. Es ist Open Source und wird aktiv weiterentwickelt. Für das Konfigurieren einer Routings genügen exemplarisch diese Zeilen:

```
func main() {
    r := mux.NewRouter()
    r.HandleFunc("/", HomeHandler)
    r.HandleFunc("/products", ProductsHandler)
    r.HandleFunc("/articles", ArticlesHandler)
    http.Handle("/", r)
}
```

Sessions

Da HTTP und damit auch die REST API Zustandslos sind werden wie Konzept gezeigt Sessions verwendet. Diese eindeutige ID wird vom Server erzeugt und vom Client bei jeder Anfrage mitgesendet. Das Gorilla Mux Framework bietet ein Session Management an, das hier verwendet wurde. Dadurch kann ein Objekt mit dem aktuellen Zustand des Nutzers gespeichert werden. Folgende Daten werden in der Implementierung genutzt.

```
type UserToken struct {
        Username        string
        IPAddress       string
        Authenticated  bool
}
```

Dabei wird neben dem Benutzernamen und dessen Status auch vermerkt mit welcher IP-Adresse dieser angemeldet wurde. Falls sich im Laufe einer Session die IP Adresse des Aufrufers ändert kann dies ein Zeichen dafür sein, dass eine Session gekapert wurde. (**R4**).

TLS

Das http Modul der GO Standardbibliothek, auf der die Rest Api Implementierung aufbaut, unterstützt TLS standardmäßig. Folgender Aufruf genügt um einen Webserver mit aktiviertem TLS zu starten.

```
http.ListenAndServeTLS(endpoint, pemCertificate, pemKey, router)
```

Der Aufruf benötigt die Zertifikate und einen Router, der die Logik der Webschnittstelle bereitstellt. In dieser Implementierung wird wie bereits erwähnt Gorilla Mux dafür verwendet.

Bereitstellung

Für die Beispielimplementierung soll ein im Konzept empfohlener Load Balancer vernachlässigt werden. Vielmehr genügt ein einfacher Webserver, der TLS unterstützt. Die Wahl fiel auf den Apache Web Server, dieser läuft auf allen gängigen Betriebssystemen darunter Linux Distributionen und Windows. TLS Verbindungen inklusive Client Authentifizierung werden unterstützt. Apache Web Server wird auf 50% a (Apache HTTP Server Market Share)ller weltweiten Webseiten verwendet, ebenso von großen Cloudanbietern unterstützt und stellt daher auch im Realbetrieb ein verlässliches System zur Verfügung. Der Testaufbau verwendet einen Docker Container von Apachen. Über ein mit dem Hostsystem geteilter Ordner können Updatepakete freigegeben werden.

MQTT Kommunikation

Der im Konzept bereits beschriebene HiveMQ Broker wurde für die MQTT Kommunikation verwendet. Die im Konzept genannten Zertifikate zur Authentifizierung wurden nicht verwendet. Auch auf die Implementierung von Autorisierungsregelungen musste aus Zeitgründen verzichtet werden.

Update Pakete

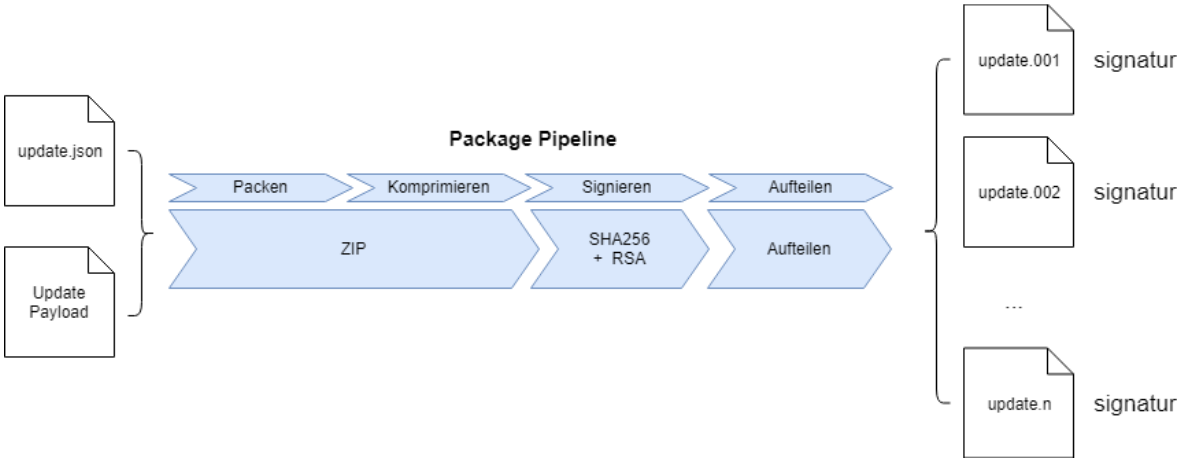

Abbildung 36 Konkrete Package Pipeline Implementierung

In der Implementierung wurden verschiedene Verfahren ausgewählt, um die Funktionsweise schematisch darzustellen.

1. **Packen & Komprimieren**

   Im Konzept wurde zur Verwendung eines Deltakompressionsalgorithmus wie Google Courgette geraten. Diese Implementierung war jedoch im Rahmen dieser Arbeit zu umfangreich. Daher wurde das bekannte ZIP Format verwendet. Dieses wird durch die Standardbibliothek von Go direkte unterstützt. Da das ZIP Format einen Container darstellt wird der Arbeitsschritt des Packens auch von diesem Verfahren übernommen.

2. **Signieren**

   Zum Signieren wird das SHA256 Hash auf die Dateien angewendet und mittels RSA und dem privaten Schlüssel der Geräteverwaltung signiert.

3. **Aufteilen**

   Die signierte Datei wird in Stücke der Größe 1 MB geteilt. Alle Teilstücke werden ebenfalls signiert.

Da Go eine hohe Portabilität besitzt kann dieser Softwarekomponente sowohl auf dem Gerät als auch auf dem Backend verwendet werden. Wie in Abbildung 37 zu erkennen ist, sind die einzelnen Bestandteile der Package Pipeline wieder als Service implementiert und durch Interfaces getrennt, um diese leicht austauschen zu können. Der Packer und der Unpacker teilen sich eine Basisklasse, der den geteilten Code zwischen Geräteverwaltung und Fahrzeug bereitstellt.

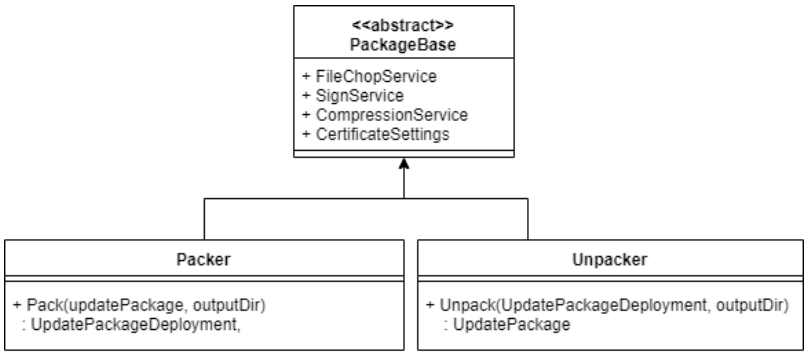

Abbildung 37 Klassenstruktur des (Ent-)packers

Notification Service & Status Receiver Service

Diese beiden Services wurden im Konzept [Kapitel] bereits beschrieben und wurden dementsprechend als Service in Go implementiert. Sie greifen zusammen mit der Geräteverwaltungs API auf eine geteilte Datenpersistenzschicht zu. Auch diese Services werden als Docker Container betrieben.

Zertifikate

Generierung der notwendigen Zertifikate

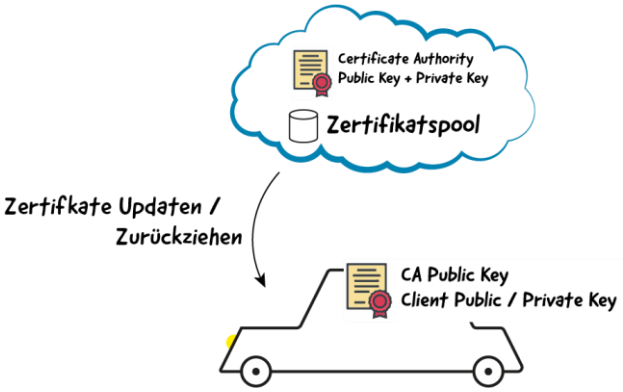

Abbildung 38 Zertifikate des Systems

Die Certificate Authority, also die Geräteverwaltung, benötigt einen Satz von Private und Public Keys. Dieser wird für eine verschlüsselte TLS Verbindung benutzt. Außerdem um die Zertifikate der Fahrzeuge zu signieren. Jedes Fahrzeug erhält ein solches Zertifikat. Dieses dient ebenfalls zur verschlüsselten Verbindung aber vor allem zur Authentifizierung im System selbst. Es wurde darauf verzichtet ein Zertifikatsmanagement Tool wie etwa HashiCorp Vault oder ähnlich zu verwenden. Um das System exemplarisch darzustellen ist der Zertifikatspool ein Ordner im Dateisystem der Geräteverwaltung, der alle Gerätezertifikate enthält. Um die benötigten Schlüssel zu erzeugen wird das OpenSource Tool OpenSSL verwendet. Dieses lässt sich als Komponente in Go verwenden oder aber auch über die Kommandozeile wie hier gezeigt ansteuern. Es werden Kommandozeilenbefehle gezeigt, um die erforderlichen Dateien zu erzeugen.

Erzeugung eines Privaten Schlüssels der Certificate Authority

```
>>>openssl genrsa -aes256 -out ca-key.pem 2048
Generating RSA private key, 2048 bit long modulus
Enter pass phrase for ca-key.pem:NOVA
```

Erzeugung des Öffentlichens Zertifikats der Certificate Authority

Dieser wird später von allen Geräten importiert

```
>>>openssl req -x509 -new -nodes -extensions v3_ca -key ca-key.pem -days 365 -out ca-root.pem -sha512
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:BY
Locality Name (eg, city) []:München
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NOVA OTA
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:nova.de
```

Erzeugen eines Privaten Schlüssels pro Gerät

```
>>>openssl genrsa -out zertifikat-key.pem 4096
```

```
Generating RSA private key, 4096 bit long modulus
```

Erzeugung eines Öffentlichen Zertifikats für jedes Gerät

```
>>>openssl req -new -key zertifikat-key.pem -out zertifikat.csr -sha512
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:BY
Locality Name (eg, city) []:München
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NOVA
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:DEVICE-NAME
```

Signierung des Gertäezertifikats durch die Certificate Authority

```
>>>openssl x509 -req -in zertifikat.csr -CA ca-root.pem -CAkey ca-key.pem -CAcreateserial -out zertifikat-pub.pem -days 30 -sha512
Signature ok
subject=/C=DE/ST=BY/L=M\xC2\x81nchen/O=NOVA/CN=DEVICE
Getting CA Private Key
Enter pass phrase for ca-key.pem:NOVA
```

Car Service

Um die Prototypen exemplarisch Logik zu geben wurde ein CarService implementiert. Dieser stellt die Standardfunktionalität eines Fahrzeugs bereit. Fahren / Anhalten, das Aufladen der Batterie. Daneben enthält es die aktuellen Eigenschaften darunter die Fahrzeugnummer, den aktuellen Batterie und Mobilfunkstatus, die simuliert werden. Dabei sinkt der Batteriestatus beständig. Der Mobilfunk schwankt ständig. Der Car Service exponiert seine Funktionsweise wieder über eine REST Schnittstelle. Der Service ist wieder als Docker Container ausgeführt.

Als Konkreter Updatevorgang soll nun in der Ursprünglichen Auslieferung des Fahrzeugs ein Fehler in eben diesem Service vorhanden sein. Das Fahrzeug lässt sich nicht starten. Durch ein Update soll nun eben dieser Fehler behoben werden können.

Oberfläche

Das Ziel der Oberflächenimplementierung war es, eine Technologie zu verwenden, die durchaus auch von großen Automobilherstellern genutzt werden. Oft anzutreffen ist hier das QT Framework. Dieses C++ Framework bietet eine Oberflächenbibliothek, die sich beliebig an das gewünschte Design der Hersteller anpassen lässt und viele unterschiedliche Plattformen unterstützt. Eine andere Möglichkeit ist das Verwenden von Webtechnologien. Die angezeigte Oberfläche ist quasi eine Webseite, die in einem Browser dargestellt wird. Webbrowser exisitieren für alle gängigen Platformen. Der Autor hatte bereits Vorkenntnisse in QT und daher wurde dieses Framework für eine Exemplarische Oberfläche ausgewählt. Eine sehr gute Gegenüberstellung dieser beiden Technologien kann in folgendem Video betrachtet werden: HTML5 vs. Qt (https://www.youtube.com/watch?v=Z4CwVN9RRbE).

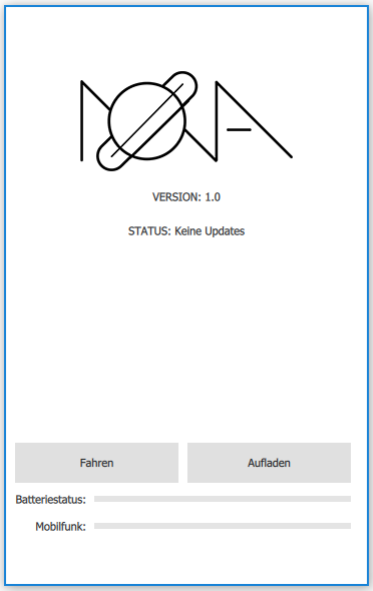

Abbildung 39 Testoberfläche

Die Oberfläche steuert die REST API des Car Service fern.. (High performance HTML5 engines enabling a standards based solution for delivery of HMI, services and applications.)

Command Line Interface

Für die Ansteuerung der Geräteverwaltung wurde auf die Implementierung einer Web App verzichtet und stattdessen ein Konsolenprogramm entwickelt. Mit diesem kann in vollem Umfang auf die Geräteverwaltung zugegriffen werden. In Abbildung 40 sind die implementierten Kommandos sichtbar.

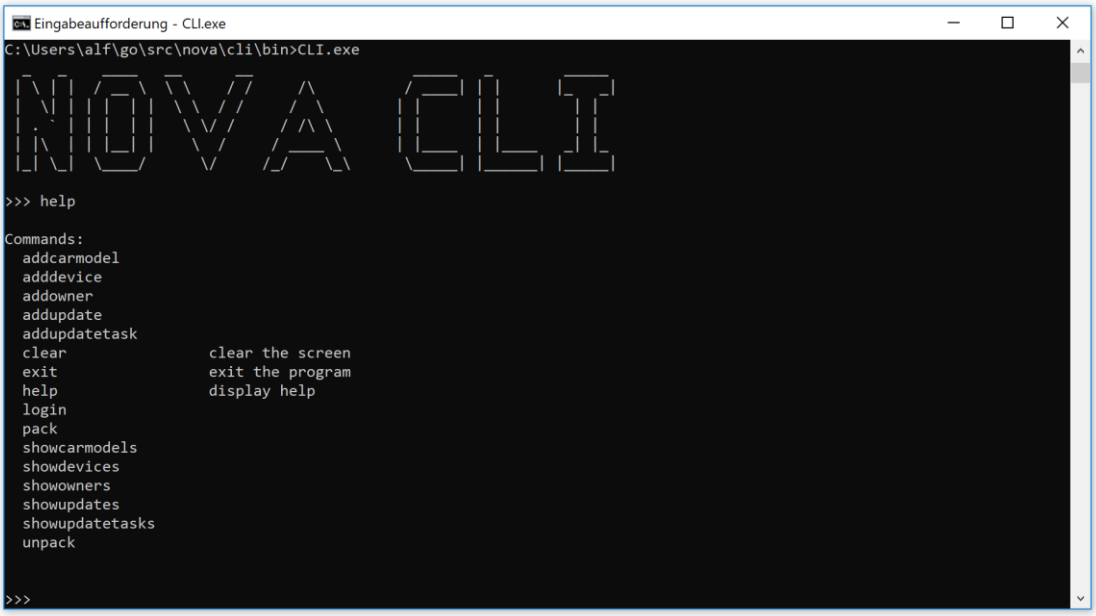

Abbildung 40 Command Line Interface der Implementierung

Beispielhafter Updatevorgang

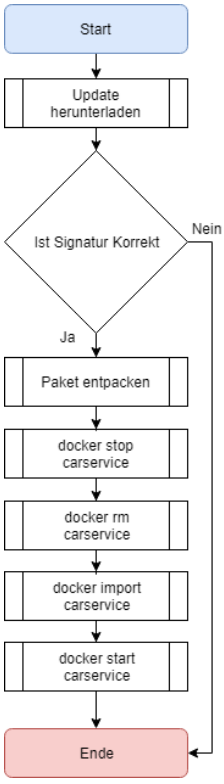

Abbildung 41 Ablauf des Updatevorgangs

Es soll nun der defekte Car Service ausgetauscht werden. Da es sich dabei um einen Docker Container handelt. Muss ein DockerUpdateCommand und ein dazugehöriger DockerUpdateCommandHandler implementiert werden. Anschließend muss der Handler bei der UpdateEngine registiert werden. Abbildung 41 zeigt nun wie dieser Update Vorgang abläuft. Da es sich um einen Test handelt wurde ebenfalls auf das Blue Green Deployment verzichtet. Das Update Paket wird heruntergeladen und auf Gültigkeit überprüft. Anschließend wird der aktuell laufende Docker Service gestoppt und durch die neue Version ersetzt, die dann gestartet wird. In der Abbildung sind die Befehle der Docker Command Line Interface ausgeführt die verwendet werden.

Demonstration

Das Softwareprojekt und ein Video, das den schematischen Updatevorgang zeigt werden auf einem USB-Stick beigelegt.

Fazit

Die aktuelle Methode der meisten Hersteller, die Software ihrer Fahrzeuge gegebenenfalls bei Kundendienst oder geplanten Werkstattbesuchen zu aktualisieren ist bereits jetzt unzulänglich. Die Kunden sind nicht bereit diesen Mehraufwand aufzubringen. In der Zukunft ist mit einer stetig wachsenden Menge an Software in vernetzten Fahrzeugen und damit mehr Updates zu rechnen. Das aktualisieren von Software über das Internet ist dabei eine Technologie, die bereits in vielen Bereichen eingesetzt wird. Darunter gehört PC-Software für Endanwender aber auch Betriebssysteme für Mobiltelefone. Auch der Fahrzeughersteller Tesla, zeigt wie dieses Konzept für Fahrzeuge umgesetzt werden kann. Über Kurz oder Lang werden alle Autos über dieses Technik verfügen. Autohersteller werden durch geringere Kosten pro Update profitieren und Fahrzeughalter durch sicherere Fahrzeuge, die stetig mit neuen Funktionen ausgerüstet werden.

Für das Konzept konnte auf viele bereits existierende Ideen und Techniken zurückgegriffen werden. Diese stammten meistens aus dem Bereich des Internets der Dinge, zu der vernetzte Fahrzeuge zweifelsohne gehören. Verschlüsselte und Authentifizierte Verbindungen konnten durchgehend mit TLS umgesetzt werden. Robuste Kommunikation ist mittels MQTT möglich. Für das entwickeln der benötigten APIs konnte ebenfalls auf den defacto Standard REST gesetzt werden, der wiederum auf HTTP aufsetzt.

Zu erwähnen sei auch, wie bisher fachfremden Technologien, wie *Ethernet*, *Internet Protocol* und der *Service-Orientierten Architektur*, in das Automobil Einzug halten, und wie dadurch die Komfort des und Robustheit des Updatevorgangs gesteigert werden kann

Die Verwendung von Go

Das Hauptziel für die Firma PENTASYS AG war es, die Eignung der Programmiersprache Go für das Entwickeln eines Over-The-Air Updatesystems zu überprüfen. Die Implementierung verlief weitgehend problemlos, Alle Komponenten mit Ausnahme der Benutzeroberfläche konnten in Go realisiert werden. Viele Eigenschaften der Go Laufzeitumgebung erwiesen sich als äußerst hilfreich für den Aufbau eines verteilten Systems. So konnten Komponenten bei Fahrzeug und

Geräteverwaltung wiederverwendet werden. Dazu gehören sämtliche Datenklassen der REST APIs. Die Komponenten der Package Pipeline werden ebenfalls gemeinsam verwende. Durch die *Single Binary Executable* Philosophie ist es sehr einfach verschiedene Services in Container zu kapseln und beispielsweise in Docker auszuführen. Go ist eine noch sehr junge Sprache, aber lebendige Sprache, die aktiv weiterentwickelt wird. Auch die Entwicklungsumgebungen und Tools sind in großen Umfang vorhanden.

Im Gegensatz zu anderen Programmiersprachen muss bei Go jedoch auf einige komfortable Features verzichtet werden. So ist eine objektorientierte Programmierung nicht direkt möglich. Das Exception-Handling wird nicht wie weit verbreitet über *try-catch*-Blöcke abgewickelt, sondern über einen error-Rückgabewert. Dieses führt oft zu sehr viel Standardcode ohne viel Logik (*Boilerplate*). Viele verbreitete Programmiersprachen bieten Funktionen an, um Daten effektiv zu Selektieren und Transformieren zu können. (Beispiel Java: *Streams*, C#: *LINQ*), diese sind bei Go nicht vorhanden. Zusammenfassend kann gesagt werden, dass Go für die Verwirklichung Großer Projekte auf verschiedenen Plattformen eingesetzt werden kann.

Die Implementierung

Wegen des großen Umfangs eines solchen Projekts mussten sehr viele Abstriche in Kauf genommen werden, so mussten große Teile von Autorisierung und Authentifizierung ausgespart werden. Im vorherigen Kapitel wurde dokumentiert, welche Teile nicht umgesetzt werden konnten.. Die Funktionsfähigkeit des Konzepts konnte dennoch gezeigt werden.

Das *CarSec Labor* der OTH Regensburg entwickelt derzeit Evaluation Boards zum Thema IT Security im Automobilen Umfeld. Die Firmware dieser Boards muss in regelmäßigen Abständen aktualisiert werden. Es ist denkbar diese Implementierung weiterzuentwickeln und dafür zu verwenden.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Zuerst gebührt der Dank der Firma PENTASYS AG in München und besonders Steffen Renz, der meine Arbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Ein besonderer Dank gilt der Ostbayerischen Technischen Hochschule Regensburg und dem CarSec Labor unter Leitung von Herrn Professor Dr. Rudolf Hackenberg, der diese ebenfalls Arbeit betreute. Dieser hatte stets ein offenes Ohr für mein Anliegen und zeigte mir stets praktische Beispiele für dieses Anwendungsgebiet.

Bei den Zahlreichen Leser, die diese Arbeit korrigierten, möchte ich abschließend bedanken.

Felix Almesberger

15. Februar 2019

Ehrenwörtliche Erklärung

Mir ist bekannt, dass dieses Exemplar der Bachelor- bzw. der Masterarbeit als Prüfungsleistung in das Eigentum des Freistaates Bayern übergeht.

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und außer den angeführten keine weiteren Hilfsmittel benützt habe. Soweit aus den im Literaturverzeichnis angegebenen Werken und Internetquellen einzelne Stellen dem Wortlaut oder dem Sinn nach entnommen sind, sind sie in jedem Fall unter der Angabe der Entlehnung kenntlich gemacht.

Die Versicherung der selbständigen Arbeit bezieht sich auch auf die in der Arbeit enthaltenen Zeichen-, Kartenskizzen und bildlichen Darstellungen.

Ich versichere, dass meine Bachelor- bzw. Masterarbeit bis jetzt bei keiner anderen Stelle veröffentlicht wurde. Zudem ist mit bewusst, dass eine Veröffentlichung vor der abgeschlossenen Bewertung nicht erfolgen darf.

Ich bin mir darüber im Klaren, dass ein Verstoß hiergegen zum Ausschluss von der Prüfung führt oder die Prüfung ungültig macht.

Regensburg den 15. Februar 2019

_____

Felix Almesberger

Literatur & Abbildungen

Literaturverzeichnis

Literaturverzeichnis

ADAC Deutschland (Hg.): Rückrufe und Softwareupdates. Online verfügbar unter https://www.adac.de/infotestrat/reparatur-pflege-und-wartung/rueckrufe/softwareupdates.aspx.

Alessandro Segala: Ready, Set, Go lang: Brian Ketelsen explains Go's fast growing popularity. Online verfügbar unter https://cloudblogs.microsoft.com/opensource/2018/02/21/go-lang-brian-ketelsen-explains-fast-growth/.

Alex Brisbourne: Tesla's Over-the-Air Fix: Best Example Yet of the Internet of Things? Hg. v. Wired, zuletzt geprüft am 07.01.2019.

Andreas Donath: Tesla macht Bremsen des Model 3 durch Software besser. ABS-Verbesserung. Online verfügbar unter https://www.golem.de/news/abs-verbesserung-tesla-macht-bremsen-vom-model-3-durch-software-besser-1805-134611.html, zuletzt geprüft am 13.01.2019.

Apache HTTP Server Market Share. Online verfügbar unter https://www.datanyze.com/market-share/web-and-application-servers/apache-http-server-market-share.

BHATTACHARJEE, SRAVANI (2018 // 2017): PRACTICAL INDUSTRIAL INTERNET OF THINGS SECURITY // Internet of things. A practitioner's guide for securing connected… machines // Evolutions and innovations. [S.l.]: PACKT PUBLISHING LIMITED; John Wiley & Sons; ISTE Ltd (Computer engineering series, volume 4).

Bosch Software Innovations: Firmware update over the air – insights and live demo. Online verfügbar unter https://www.youtube.com/watch?v=4Ixv6-njtw8.

Bosch Software Innovations: How to manage and control software updates for IoT devices? Online verfügbar unter https://www.youtube.com/watch?v=pRFovbnIxxw.

Brian Fung (2017): As Hurricane Irma bore down, Tesla gave some Florida drivers more battery juice. Here's why that's a big deal. In: *The Washington Post*, 11.09.2017.

Bundesamt für Sicherheit in der Informationstechnik: Fragen und Antworten zum Inkrafttreten des IT-Sicherheitsgesetzes. Online verfügbar unter https://www.bsi.bund.de/DE/Themen/Industrie_KRITIS/KRITIS/IT-SiG/FAQ/FAQ_IT_SiG/faq_it_sig_node.html.

Bundesamt für Sicherheit in der Informationstechnik: TLS / SSL Best Practice. Empfehlung: IT im Unternehmen. Hg. v. Bundesamt für Sicherheit in der Informationstechnik. Online verfügbar unter https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS_012.pdf%3Bjsessionid%3D8F96D96295E8D67EF48F0074A01E8E51.1_cid369%3F__blob%3DpublicationFile%26v%3D4.

Cohn, Mike (2013): User stories applied. For agile software development. 18. print. Boston, Mass.: Addison-Wesley (Addison-Wesley signature series).

Computer and information security handbook (2017). Third edition. Cambridge, MA: Morgan Kaufmann.

Definition IT-Sicherheitsgesetz. Was ist das IT-Sicherheitsgesetz? Online verfügbar unter https://www.security-insider.de/was-ist-das-it-sicherheitsgesetz-a-644438/.

Dhanjani, Nitesh (2015): Abusing the Internet of things. Blackouts, freakouts, and stakeouts. First edition. Sebastopol, CA: O'Reilly Media.

DocuSign (Hg.): Wie digitale Signaturen funktionieren. Online verfügbar unter https://www.docusign.de/wie-es-funktioniert/elektronische-signatur/digitale-signatur/digitale-signatur-faq.

Dr. Markus Siepermann: Definition: Internet. Online verfügbar unter https://wirtschaftslexikon.gabler.de/definition/internet-37192.

Dr. Markus Siepermann: Definition: Internet der Dinge. Online verfügbar unter https://wirtschaftslexikon.gabler.de/definition/internet-der-dinge-53187/version-276282.

Easttom, Chuck (2016): Modern cryptography. Applied mathematics for encryption and information security. New York: McGraw-Hill Education.

Edsger W. Dijkstra: The Humble Programmer.

Edward Hieatt and Rob Mee: Repository. Patterns of Enterprise Application Architecture. Online verfügbar unter https://martinfowler.com/eaaCatalog/repository.html.

Gamma, Erich (1998): Design patterns CD. Elements of reusable object-oriented software. Reading, Mass.: Addison-Wesley (Addison-Wesley professional computing series).

Green Hills Software (Hg.): Green Hills Software Enables Safe, Secure CarPlay and Android Auto in the Connected Car. INTEGRITY Multivisor Securely Consolidates Safety- and Security-Critical Functions with AllGo's RACE Smartphone Connectivity. Online verfügbar unter https://www.ghs.com/news/20160105_CES_CTS_CarPlay-Android-Auto-AllGo.html.

Hannah Steinharter: Das sind die zehn wertvollsten Unternehmen der Welt. Hg. v. Handelsblatt. Online verfügbar unter https://www.handelsblatt.com/finanzen/anlagestrategie/trends/apple-google-amazon-das-sind-die-zehn-wertvollsten-unternehmen-der-welt/22856326.html?ticket=ST-1363196-FsjVf5Du2NiumBt6egMa-ap1.

Hannes Rügheimer: Der große Mobilfunknetztest 2018. In: *Connect* (01/18).

Harman (Hg.): Remote Vehicle Updating Services (OTA). Online verfügbar unter https://services.harman.com/solutions/ota-software-management/harman-remote-vehicle-updating-service-ota.

High performance HTML5 engines enabling a standards based solution for delivery of HMI, services and applications. Online verfügbar unter https://www.access-company.com/en/products/browser/netfront-html5-platforms/.

HiveMQ Developer Team: HiveMQ Authorization Example. Online verfügbar unter https://github.com/hivemq/hivemq-authorization-example/blob/master/src/main/java/com/hivemq/authorization/example/callbacks/AuthorizationCallback.java.

https://www.security-insider.de/was-ist-authentifizierung-a-617991/: Was ist Authentifizierung? Definiton Authentisierung, Authentifizierung und Autorisierung. Online verfügbar unter https://www.security-insider.de/was-ist-authentifizierung-a-617991/, zuletzt geprüft am 21.01.2019.

IBM (Hg.): Publish/subscribe messaging. Online verfügbar unter https://www.ibm.com/support/knowledgecenter/mk/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q004870_.htm.

Johannes Büttner, Markus Kucera, Thomas Waas: Opening up New Fail-safe Layers in Distributed Vehicle Computer Systems.

Leonid Bender (2018): Threshold Cryptography and distributed key generation, 20.07.2018. Online verfügbar unter https://orbs.com/wp-content/uploads/2018/07/Threshold-Cryptography-and-Digital-Key-Distribution.pdf.

McConnell, Steve (2009): Code Complete. 2nd ed. Sebastopol: O'Reilly Media Inc.

Rene Pfeiffer; Michael Kafka: Ausgewählte Angriffsvektoren. DeepSec In-Depth Security Conference.

Ron Ih: Public Key Infrastructure Explained. Online verfügbar unter https://www.kyrio.com/blog/internet-of-things-security/public-key-infrastructure-explained/.

Sam Byford: Tesla Model S getting first ever over-the-air car firmware upgrade next week. Online verfügbar unter https://www.theverge.com/2012/9/24/3385506/tesla-model-s-over-the-air-car-firmware-update.

Stephen Adams: Software Updates: Courgette. Hg. v. The Chromium Projects. Online verfügbar unter https://www.chromium.org/developers/design-documents/software-updates-courgette.

Stout Automotive (Hg.) (2018): 2018 Automotive Warranty & Recall Report.

Syracuse University: Buffer-Overflow Vulnerabilities and Attacks. Online verfügbar unter http://www.cis.syr.edu/~wedu/Teaching/IntrCompSec/LectureNotes_New/Buffer_Overflow.pdf.

Tesla: Introducing Software Version 9.0. Online verfügbar unter https://www.tesla.com/de_DE/blog/introducing-software-version-9.

The Open Web Application Security Project: OWASP Top 10. Online verfügbar unter https://www.owasp.org/images/9/90/OWASP_Top_10-2017_de_V1.0.pdf.

Vincent Lynch: Why Do SSL Certificates Expire? Expiration Helps SSL Stay Secure, and No, It's Not a CA Conspiracy. Online verfügbar unter https://www.thesslstore.com/blog/ssl-certificates-expire/.

Wolfgang Pester (2015): Das unterschätzte Übergewicht der Premiumautos. Online verfügbar unter https://www.welt.de/motor/article138223665/Das-unterschaetzte-Uebergewicht-der-Premiumautos.html, zuletzt aktualisiert am 09.03.2015.

Abbildungsverzeichnis

Abbildung 1 Die Entwicklung des Internets 4
Abbildung 2 Symmetrische Verschlüsselung 8
Abbildung 3 Asymmetrische Verschlüsselung 8
Abbildung 4 Funktionsweise einer digitalen Signatur 9
Abbildung 5 Beispiel eines Zertifikats 10
Abbildung 6 TLS Handshake 12
Abbildung 7 Aufbau der Bosch IoT Cloud 18
Abbildung 8 Schematische Darstellung des Konzepts 25
Abbildung 9 Überblick über die Geräteverwaltung 27
Abbildung 10 Datenmodell der Geräteverwaltung 28
Abbildung 11 Datenmodelle der Rest Api 30
Abbildung 12 Funktionsweise eines Load Balanders (hier von Microsoft Azure) 35
Abbildung 13 Pub/Sub Aufbau 38

Abbildung 14 Kommandoklasse 41
Abbildung 15 Übertragung von Kommandos 42
Abbildung 16 Sequenzdiagramm einer Kommandoausführung- Übertragung 42
Abbildung 17 JPEG Kompression (von links nach rechts nimmt die Qualität ab) 44
Abbildung 18 Funktionsweise Wörterbuchkompression 45
Abbildung 19 Vergleich von Kompressionsalgorithmen 45
Abbildung 20 Aufbau der Package Pipeline 46
Abbildung 21 Aufbau Service Orientierte Architektur 50
Abbildung 22 Aufbau Softwaremonolith 50
Abbildung 23 Einsatz von SOA bei Volkswagen 51
Abbildung 24 Schematischer Hardware / Softwareaufbau einer Komponente in einem Auto 51

Abbildung 25 Struktur einer Komposition 53
Abbildung 26 Ablaufplan der Resilienzschleife 54
Abbildung 27 Für Update benötigte Klassen 56
Abbildung 28 Komponenten der Resilienzschleife 57
Abbildung 29 Ablaufplan eines Blue / Green Deployments 58
Abbildung 30 Benötigte Infrastruktur für ein Blue / Green Deployments 58
Abbildung 31 Überblick über die in der Implementierung verwendeten Komponenten 62
Abbildung 32 Go vereint Vorteile verschiedener Programmiersprachen 65
Abbildung 33 Gopher, das Maskottchen der Programmiersprache Go 65
Abbildung 34 Model View Controller Struktur 69
Abbildung 35 Aufbau der Rest Api Implementierung 70
Abbildung 36 Konkrete Package Pipeline Implementierung 73
Abbildung 37 Klassenstruktur des (Ent-)packers 74
Abbildung 38 Zertifikate des Systems 75
Abbildung 39 Testoberfläche 77
Abbildung 40 Command Line Interface der Implementierung 78
Abbildung 41 Ablauf des Updatevorgangs 78

Quellen der Abbildungen

Eigene Abbildungen: 5, 8, 9, 10, 11, 14, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 36, 37, 38, 39, 40

1: https://uk.businessinsider.com/internet-of-everything-2015-bi-2014-12?IR=T
2: https://de.wikipedia.org/wiki/Symmetrisches_Kryptosystem
3: https://de.wikipedia.org/wiki/Asymmetrisches_Kryptosystem
4: https://medium.com/@meruja/digital-signature-generation-75cc63b7e1b4
6: https://www.researchgate.net/figure/Overview-of-the-SSL-TLS-handshake-18_fig5_322936890
7: https://www.youtube.com/watch?v=4Ixv6-njtw8
12: https://geekflare.com/cloud-load-balancer/
13 https://www.ibm.com/support/knowledgecenter/en/SSMKHH_10.0.0/com.ibm.etools.mft. doc/aq01120_.htm
17: https://de.wikipedia.org/wiki/JPEG
23: https://assets.vector.com/cms/content/events/2017/vAES17/vAES17_01_Ethernet-Adaptive-AUTOSAR-at-VW_Krieger_Wille.pdf
32: https://entwickler.de/online/development/golang-das-neue-java-579800853.html
33: https://github.com/golang-samples/gopher-vector