Universität Bremen

Faculty 3: Mathematics and Computer Science

Master's Thesis

Reference Security Guide for App-Controlled Smart Home Systems

Referenz-Sicherheitsleitfaden für App-gesteuerte Smart-Home-Systeme

Daniel Müller

Matriculation No. 267 785 1

22. August 2017

Examiner: Dr. Karsten Sohr
Supervisor: Prof. Dr. Ute Bormann
Advisor: Dr. Karsten Sohr

Daniel Müller

Reference Security Guide for App-Controlled Smart Home Systems

Referenz-Sicherheitsleitfaden für App-gesteuerte Smart-Home-Systeme

Master's Thesis, Faculty 3: Mathematics and Computer Science

University of Bremen, August 2017

Declarations

I declare that the sources of all information in this thesis (whether data or comment, paraphrased or directly quoted, published or not) are explicitly stated and appropriately referenced. Any material which is reproduced verbatim from any source is shown as directly quoted.

Bremen, 22. August 2017

_____

Daniel Müller

Acknowledgements

I would like to sincerely thank my advisor Dr. Karsten Sohr for many helpful suggestions, as well as the other team members of the *SecureSmartHomeApp* research project for providing me with material necessary to accomplish this thesis. I also thank Prof. Dr. Ute Bormann for being the supervisor of this document. Lastly, I would like to mention my parents for selflessly supporting me throughout my studies.

Abstract

The market of smart home systems is fast-moving and highly evolving. Many manufacturers are subject to the pressure of quickly developing and publishing their own variant, to gain and retain market share. Furthermore, owing to the novelty of the market, few guidelines concerning the security and best-practices of smart home systems exist. This leads to smart home programs containing fundamental security flaws, rendering them basically inapplicable for secure home automation and surveillance.

On the basis of multiple security analyses of existing systems, this thesis will develop a general best-practice guide and provide a reference list for common security flaws, in addition to recommended safeguarding measures. Manufacturers and developers can refer to the outcomes of this thesis to protect their own system, ideally fostering a more secure smart home generation.

Contents

Chapter 1

Introduction

An increasing amount of homes is equipped with automation devices allowing for an interaction between the virtual and the physical worlds: Actuators and sensors enable house owners to control door locks, radiators and other important domestic parts over the internet using software. These devices, often equipped with slow, but energy-efficient processors and limited memory resources, are especially relevant for research in the context of the Internet of Things (IoT).

However, being able to check and control devices like lights, appliances and windows from any place in the world cannot only be considered as a simplification of daily life. The so-called *smart homes* may also become a large security hazard when attackers are able to gain unauthorized access to these systems. Intruders could be able to open doors or windows without the consent of house owners. Once an attacker has compromised the system, manipulated log files can erase traces, rendering the smart home system insecure for reliable and forensically traceable security surveillance purposes. Sensor data transmitted insecurely by an alarm system could be intercepted, manipulated or forged. Hence, the three well-known security goals confidentiality, integrity and authenticity, as explained in Chapter 2.1, are endangered.

Recent examples of security breaches also show that software updates fixing security vulnerabilities are no matter of course. In a case of 2016, the company *Nest*, belonging to *Alphabet Inc.*, has been informed by security reasearcher Jason Doyle about ways to attack its smart security surveillance cameras. By exploiting specific bugs in the Bluetooth Low Energy protocol, Doyle was able to disable the cameras for several minutes. During that time, potential burglars could secretly enter and leave surveillanced properties. Still, no updates were provided for months. It was only when Doyle made the issue public that Alphabet Inc. reacted and announced an

imminent security patch.[1]

As of today, there are approximately three million smart homes in Germany. In four years' time, the market will have expanded to 13,67 million houses, hence roughly every sixth home is equipped with smart home devices. [sta16] Since the market is rather new, extensively growing and few proved and tested security guidelines exist, programming these systems involves a lot of trial-and-error. The development of any smart home's architecture should be based on an elaborate and clear security catalog instead. The aforementioned guideline will help to prevent common program errors and incorrect security implementations occurring due to market pressure and, as a result, short software release cycles. Lastly, by providing precise techniques, it can help to improve, structure and speed-up the entire development process.

1.1 Main Goals

The main goal of this thesis is to establish a detailed guideline for smart home systems predominantly controlled by mobile apps. This is accomplished by developing and explaining best-practice security techniques for different architectural levels of a generalized smart home model, derived from insights of literature and an interview. Moreover, a threat model (cf. Chapter 2.4) is generated to characterize relevant attackers.

Ordinary security mechanisms are often not applicable in the context of the IoT, for the reasons explained in the following chapters. By summarizing results from analyses of commonly used smart home systems, the following chapters also emphasize that software companies regularly try to adapt these well-known security mechanisms in the context of the IoT, even when it is not appropriate. Alternatively, some companies completely abstain from implementing security for certain issues. This leads to interesting constructs prone to attacks. In order to examine potential security risks, some of these constructs will be analyzed in the course of this document. The analyses will be based on multiple bachelor's and master's theses, predominantly written in the context of the *SecureSmartHomeApp* joint research project of the University of Bremen and neusta mobile solutions GmbH (called *neusta* in the following). The project aims to learn from various (insecure) smart home products on the market, to derive new insights for the development of Coqon, neusta's own smart home solution. The project's main focus is set on

[1] More information on this case can be found under [Gol17].

systems controllable by Android apps. Accordingly, whenever apps are concerned, this thesis will focus on Android versions. The availability of elaborate software for Android app security analysis was also a decisive factor for this choice. Where applicable, this thesis occasionally refers to the Apple iOS app version. Generally, network communication between the various smart home components is assumed to be identical or similar for any platform variant because of standardized protocols and uniform Application Programming Interfaces (APIs), as explained in Chapter 2.3.

The validity of many of the insights derived from other documents could not be verified, due to the absence of specific smart home hardware. However, the findings and security problems described in these resources are still very valuable, as the main goal for both, the research project and this thesis, is not finding out whether specific third-party systems are definitely prone to a certain vulnerability, but rather the derivation of general security protection mechanisms for plausible scenarios.

The outcomes of these analyses will be used to construct the best-practice catalog developers can use for reference for their own projects, adhering to the *Security by Design* principle. A development process explicitly structured according to the aforementioned principle requires the design of security goals and quality assurance measures. An appropriate *Maturity Model*, as several security standards call a method for measuring the elaborateness of development processes[2], then allows for an evaluation of the fulfillment of each security goal.

It is noted that the results of this thesis do not claim to be exhaustive. Security best-practices generally need to be reconsidered regularly and can be defined in such detail that the extent would exceed the scope of a master's thesis as well as hinder the list's generality. However, as neusta's developer Kevin Löhmann stated during an interview in Chapter 3.3, currently no reliable standard or best-practice catalog exists in the context of smart home security. Therefore, the many findings presented in the following chapters can be considered to be a solid and reliable basis and a first step in securing the world of smart home, being extensible and verifiable by even more elaborate research.

[2]For example, the standards *C2M2* [ENE] and *SSE-CMM* [bit] can be considered interesting in the context of Security Maturity Models. Maturity Models and according frameworks will not be considered further in this document.

1.2 Overview

Following the introduction, the second chapter provides background information on the functionality and the general architecture of smart home systems, Android and the fundamentals of network communication. Components incorporated in the process of managing and controlling smart home devices will be examined in detail. In Chapter 3, commonly used smart home systems and apps are presented and analyzed, to point out relevant security flaws. The fourth chapter then revisits the encountered security problems and rates them in matters of an appropriate attacker model. This so-called threat model is eventually used to create preventive countermeasures in the form of a reference guide in the fifth chapter. Finally, the results are evaluated and an outlook is given in Chapter 6.

Chapter 2

Background

This chapter introduces basic concepts and a general architecture for smart home systems in order to create a foundation for the following chapters. The exemplary architecture is referenced at several points in this thesis, to highlight characteristics of the apps and systems presented in Chapter 3.

2.1 Digital Signatures and X.509 Certificates

In a communication scenario between the fictive parties Alice and Bob, it is necessary to ensure that the message received by Bob was really created by Alice, i.e. to provide *authenticity*. At the same time, it is also indispensable to protect the message from being altered on the way, accomplished by assuring message *integrity*. At least, any tampered message needs to be clearly identified by Bob upon receipt. When encryption is involved, *confidentiality* can be assured, resulting in an adversary unable to decipher the contents of the transferred data.

The first two security goals are usually implemented by digital signatures and X.509 certificates. A digital signature is comparable to a written signature on paper and can only be created by the original sender. This is achieved by using an asymmetric encryption method such as RSA, which involves the creation of a key pair. One component of the pair, the *private key*, is kept secret and is only available to Alice. She uses the key in combination with a cryptographic hash function to calculate and encrypt a message digest value.

The digital signature, i.e. the encrypted digest value, can then be verified by anyone possessing

the accompanying digital certificate, binding the public key needed to decrypt the digest value to Alice. The resulting unencrypted hash value originally created by Alice can then be compared to the hash value Bob receives by applying the same cryptographic hash function to the message. If the values do not match, the message was changed since the digital signature was applied.

A digital certificate, on the other hand, is usually also digitally signed, specifically by a trusted third party, called the Certification Authority (CA). This technique diminishes the risk of an attacker creating and signing a message in the name of Alice using his own private key, while distributing a forged public key. The signed digital certificate would eventually prevent the attacker from distributing his public key needed for decryption, as no trusted party would confirm the relation between his public key and Alice. When a certificate is broken, i.e. the corresponding private key is exposed to a third party or the signature algorithm becomes insecure, the certificate's serial number is usually entered into a public Certificate Revocation List (CRL), an enumeration containing IDs of rejected certificates. Software needs to actively request and check the CRL of the corresponding CA.

Digital signatures are not only used in the context of signing plain messages - they are also an essential part of securing World Wide Web Hypertext Transfer Protocol (HTTP) transfers and are preventing attackers from eavesdropping sensitive data, as shown in Chapter 2.3.

Alternatively, in a scenario where asymmetric encryption schemes are not feasible due to limited resources, a value called Message Authentication Code (MAC) can be computed to assure the messages' integrity and authenticity. A MAC differs from digital signatures inasmuch as that the same key is used for encryption and decryption of the hash, requiring a secure key exchange algorithm or a common secret instead.

2.2 Android Apps

The smart home systems analyzed in the course of this thesis mainly rely on mobile apps for controlling. Owing to the better tools for app analysis and the *SecureSmartHomeApp* project's dedication to the Android system, this thesis will focus on the Android version of each app. Thus, this chapter will provide an overview about Android's most important features.[1]

[1] This section is based on the insights and the information described in the preceding bachelor's report [Mül15].

Android is an operating system intended for mobile devices, based on the Linux kernel. It is mainly maintained by *Google*. *Apps* represent programs destined to extend the system; they are usually developed using the programming language *Java*. Every app is eventually distributed in a single, digitally signed file, called APK. Being a simple archive of the ZIP format, the APK contains the compiled source code and various resources like images, layouts and color definitions. Every app is executed in a separate Virtual Machine (VM), which yields independence of the hardware the code is executed on. As a result, whenever an app crashes the VM, the operating system remains stable. Furthermore, apps are only allowed to use specific APIs for operating system and inter-process communication.

The following list further explains the concepts and components typically involved in the development of Android software:

1. **Manifest**

   Every APK contains a file named `AndroidManifest.xml`, used for the specification of the app's essential properties. For example, the manifest contains the name of the app and information about the developer. It also comprises declarations of additional components, like activities or services (see below).

2. **Activities**

   An activity is a single screen layout of an app, typically displayed fullscreen. Basically, this component consists of a class extending `Activity` and an accompanying file in the XML format, whereas the latter describes the screen layout. Every activity is eventually declared in the manifest.

3. **Dialogs**

   A dialog is used when executing the program depends on user input. It will block activity execution and provide Graphical User Interface (GUI) elements like text boxes and textual descriptions of the demanded input.

4. **Permissions**

   In order for apps to leave the VM in a controlled manner, Android uses the concept of permissions. An app needs to actively demand World Wide Web or GPS access, for example. Upon app installation, users have to confirm the requested permissions. Developers can easily inquire permissions for their apps by adding lines to the `AndroidManifest.xml` file.

5. **Intents**

   Intents allow for inter-process communication. They are used whenever an app tries to execute a system-relevant action - for instance, start another activity or call a specific phone number.

6. **Broadcasts**

   Intents can also be utilized to send messages (so-called broadcasts) to other apps. The operating system searches for the destination and is therewith responsible for broadcast delivery. Broadcast-receivers handle message receipts. They are declared in the manifest file.

7. **Services**

   A service is a class extending `Service` (or an according subclass). Basically, it is a component that is executed in the background, for example to perform long-lasting, expensive calculations. A service is not necessarily started by the declaring app - a specific parameter in the `AndroidManifest.xml` permits execution by third-party apps, if desired.

8. **KeyStore**

   Android provides a secure storage for cryptographic keys. The KeyStore actively prevents extraction of the key material by application processes and from the device as a whole. Furthermore, apps can easily specify usage restrictions for the key material. For example, prior to accessing the key material, developers can require the user to authenticate using his PIN, fingerprint, etc. [Dev]

Generally, the programs are distributed through a centralized platform, namely the *Google Play Store*. In doing so, Google can check every submitted app for malicious functionality and regulate them according to proprietary policies.

2.3 Computer Networks

Networks and (wireless) data transmission are an essential part in every smart home scenario. This chapter will introduce the fundamentals of network protocols and transmission standards.

2.3.1 Network Architectures and OSI-Model

Every instance of digital data exchange needs to be strictly structured by communication protocols. The protocols ensure that the transferred data are correctly received and interpreted by each host.

In order to assure that every transferred packet (also called *datagram*) does eventually reach the destination, many protocols specify strict message orders and provide so-called *handshake algorithms*, used for mutual acknowledgments of receipt. As digital information is eventually represented by a binary sequence, communication protocols also need to specify fundamentals like the byte order the information is represented in.

Various protocols have been proposed for different scenarios of communication. To maintain clarity and to prevent repetitions concerning fundamentals like byte order specifications in the definitions of different protocols, computer networks operate on a *stack* of protocols rather than a single protocol. The most common model for telecommunication is defined by the Open Systems Interconnection Model (OSI) concept, hierarchically dividing a communication system into seven protocol layers. The bottom layer allows for protocols managing the raw electrical communication, whereas the top layer is reserved for application protocols like HTTP (cf. Table 2.1).

Many modern computer networks use *packet switching* architectures, requiring transferred information to be divided into structured data packets of equal length, fostering network quality and efficiency. Thus, protocols of the fourth layer (*Transport Layer*) usually provide packeting.

Depending on the protocol variants used in the stack, higher layers may assume a certain level of quality in matters of the underlying connection. For example, if Transmission Control Protocol (TCP) is used as a transport protocol in layer four, the subsequent layers can presume data packets to arrive at the destination due to several handshake mechanisms implemented by TCP. Furthermore, it provides in-order delivery, meaning that the data will always arrive in the order in which it was sent [Gou+02]. On the other hand, if User Datagram Protocol (UDP) is used, data packets are not guaranteed to arrive, in favor of lower latency. In practice, most apps incorporate TCP in their protocol stacks because of the higher reliability; for use cases like video chats, where low latency is important but single video frames are dispensable, UDP would be applicable.

Layer	Name	Exemplary Protocols
7	Application Layer	FTP, DNS, HTTP, NTP
6	Presentation Layer	MIME, TLS, SSL
5	Session Layer	NetBIOS, Pipes, SCP
4	Transport Layer	TCP, UDP
3	Network Layer	IPv4/IPv6, ICMP
2	Data Link Layer	PPP, MAC (not to be confused with Message Authentication Code)
1	Physical Layer	Ethernet (IEEE 802.3), Wireless LAN (IEEE 802.11)

Table 2.1 The OSI-Model, cf. [Wil11]. Many of the protocols mentioned are used as typical examples and are not considered further.

The seventh layer, called *Application Layer*, is used for abstract high-level protocols like File Transfer Protocol (FTP) or DNS. In the context of the internet, HTTP is one of the most common protocols for layer seven. HTTP is an ASCII-based protocol mainly used for the receipt of websites. Owing to its flexibility and ease-of-use, HTTP is also often employed when software communicates with applications hosted on a web server. This allows for the creation of a single API for web browsers and stand-alone software. The general structure of HTTP data is listed in Table 2.2.

It should be noted that, in web communication scenarios, the OSI-Model is often simplified by the TCP/IP-Model, which provides fewer layers and a slightly different naming convention. As this thesis focuses on general communication scenarios, the TCP/IP-Model is not considered further.

2.3.2 Hypertext Transfer Protocol and Hypertext Transfer Protocol Secure

As TCP already provides ordered and error-free data transportation, HTTP can solely implement information requests and responses for web applications. HTTP packets are structured according to Table 2.2. When a client requests a specific resource from a server, the former initially defines a command line, specifying the desired action (e.g. receive a resource, delete a resource, etc.) and the path to the resource. Separated by a new line, the client also submits headers. Headers can be used to supply the request with additional information, for instance to specify the language

the client would like the resource to be in. In case the client needs to transfer data to the server, a message body containing data of arbitrary type follows. An exemplary request is shown in Listing 2.1.

Element	Content
Command (request only)	HTTP defines various verbs for requests, to specify the desired action. Common action verbs are POST, PUT, GET and DELETE, used to create new content, update data, receive information and to delete a resource, respectively.
Status (response only)	Responses begin with a three-digit response code, explaining whether the original request was fulfilled successfully.
Header fields	Headers are optional and can be used to specify the requests' or responses' properties, like content language, content length or data format.
Body	The body is optional and can contain data of any type, binary values are also possible. It can be used to transfer the requested resource's contents or the data to send to the web server.

Table 2.2 General structure of HTTP packets, cf. [Gou+02].

Upon receipt, the server generates a response consisting of a three-digit status response code, various headers (e.g. defining the length of the response or the date the resource was last modified) and the optional response body, containing the content of the requested data as shown in Listing 2.2.

```
1  GET test.html HTTP/1.0
2  ======================
3  Accept-Language: en
4  Accept: text/*
5  Host: www.example.com
6  (...)
```

Listing 2.1 HTTP request example. The equation symbols were inserted for illustration purposes, to separate action and headers.

```
1  200 OK
2  ======================
3  Last-modified: Wed, 29 Mar 2017 00:07:00 GMT
4  Content-Length: 19
5  (...)
6  ======================
7  <HTML>
8  <HEAD>
9  (...)
```

Listing 2.2 HTTP response example. The equation symbols were inserted for illustration purposes, to separate status code, headers and body.

HTTP by itself does not implement security features. The ASCII encoding facilitates traffic interception attacks and software like *Fiddler*[2] for the Windows operating system or *Wireshark*[3] (cf. Chapter 2.6) are effortlessly able to record every HTTP packet. Thus, when the internet became increasingly common in the year 1994, a secure variant of HTTP was developed: Hypertext Transfer Protocol Secure (HTTPS). It is implemented in any major browser and represents the most popular version of secure HTTP [Gou+02]. Basically, HTTPS packets are created in the same way as for HTTP, the difference being that secure protocols, namely SSL and TLS, are used for layer 6 (*Presentation Layer*, cf. Table 2.1). This way, HTTPS encrypts and authenticates HTTP packets, both with cryptography and digital certificate algorithms, before relying on the TCP protocol for stable transport. TLS is the modern successor to the obsolete SSL standard. As of May 2017, the most current variant is called TLS version 1.2. The Bundesamt für Sicherheit in der Informationstechnik (BSI) provides up-to-date security checklists[4], explicitly recommending TLS 1.2 and discouraging the use of versions 1.1 and 1.0 or even deprecated SSL. However, standardization of TLS version 1.3 is currently in progress and has almost finished. The latest Internet Engineering Task Force (IETF) draft already provides insights into the functionality of the protocol and reveals that TLS 1.3, as the increased revision number suggests, is an update with several small security and usability fixes. Still, it does explicitly emphasize its "Work in Progress" state in the introduction, additionally it is said that the standard was not subject to elaborate security analysis yet.[5] Some programs and servers already implement the new TLS standard, in order to encourage developers to incorporate and test it in their projects [Sch17]. The level of security does, however, greatly depend on the selected algorithms: TLS offers flexibility by providing multiple encryption algorithms of variable block length and different key exchange techniques, similar to many other encryption standards. A set of algorithms is referred to as a *cipher suite*.

Even though HTTPS is still optional, various browsers increasingly encourage web developers to provide secure connections to their servers. For example, the browser *Google Chrome* started to inform users about insecure websites containing password fields, when HTTPS encryption is not used. Additionally, Google returns a higher rank for websites providing HTTPS. [Bro16]

[2] *http://www.telerik.com/fiddler*
[3] *http://www.wireshark.org*
[4] Cf. the variant [Sic17, p. 6], published in March 2017.
[5] The most current revision of the draft, as of the beginning of May 2017, can be found at [Res17].

2.3.3 Properties of Hypertext Transfer Protocol Secure

To prevent the server from identifying HTTPS traffic as erroneous HTTP data, connections using the HTTPS protocol can quickly be identified. The corresponding URL starts with *https://* and the connection request is established to server port 443, whereas regular HTTP connections use port 80 and the URL prefix *http://* instead. [Gou+02]

When HTTPS is enabled, before a regular HTTP request can be sent to the server, a handshake mechanism is performed in order for both communication partners to agree on mutual encryption and certification parameters. Upon the completion of the handshake, the Presentation Layer can be employed to encrypt plain HTTP messages. Subsequently, the encrypted messages are packaged using the TCP protocol on Transport Layer.

An essential part of TLS is the authentication mechanism. A web browser is thereby able to verify that a website, specifically the traffic received from the corresponding web server, truly belongs to the expected party. In accordance with Chapter 2.1, digital certificates provide the necessary identity information for servers. Though it is possible for clients to provide digital signatures for identification as well (*mutual authentication*, or *two-way authentication*), it is rarely used due to its complexity [Gou+02]. The certificates are, in addition to other information, comprised of the name of the corresponding party and, in most cases, the DNS name of the server (cf. Figure 2.1). The latter may contain a wildcard symbol (*) to match all subdomains of the same level of a given DNS name (e.g. **.example.com* will match *test1.example.com* and *test4.example.com*, but not *test9.test1.example.com*). The X.509 standard theoretically also allows for IP addresses instead of DNS names [al08, 35 f.].

DNS names are an integral issue when digital signatures are used for servers located in the LAN or for servers without DNS names. Well-known and accepted CAs only generate certificates for servers that are publicly reachable via unique DNS entries. This prevents potential attacks. For instance, if the CA did not verify the target host, an attacker could request the CA to issue an additional, separate valid certificate for a non-unique local server name, e.g. `server1.local`, without difficulty. In a LAN attack scenario, he could then assume the identity of the aforementioned server, facilitating Man-In-The-Middle attacks. [GoD]

Public reachability is rarely given for internal servers with DNS names, as designated DNS servers are usually only integrated into the specific LAN. These servers do not provide public data for internet name resolution, impeding the mandatory CA reachability. This DNS-based

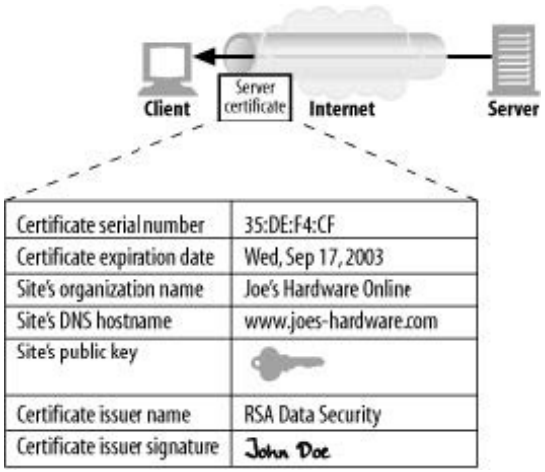

Figure 2.1 An illustration of an X.509 certificate for a domain called *www.joes-hardware.com*. Source: [Gou+02, p. 327]

certification scheme also implies that trusted certificates cannot be officially issued for static nor dynamic IP addresses in general, rendering TLS impractical for many scenarios.

It is always possible to generate self-signed certificates. However, this may hinder a flawless user experience (see below) and, among other disadvantages, may require a large effort in refreshing certificates regularly when dynamic IP addresses are concerned. For products like smart home controllers (cf. Chapter 2.5), a static, fixed IP address cannot be universally guaranteed for every customer's LAN, as other devices may have been bound to the intended address previously. Yet, in specific scenarios, when trusted manually or when used in a proprietary protocol, self-signed certificates can provide a simple and reliable method for traffic encryption.

Whenever a connection to a secure server is established, the browser performs tests to ensure the validity of the certificate. Besides checking the start and end dates, the browser analyzes the CA that vouched for the server. If it is trustworthy and if it signed the server's certificate correctly, the site's host name is compared to the information stored in the certificate. In case any of the information mentioned above is not valid, especially if the certificate was self-signed, the browser will notify the user, potentially even blocking access to the site completely. [Gou+02] Figure 2.2 shows a typical error message Google Chrome displays when a self-signed certificate is used to authenticate a LAN-internal IP address.

To further improve the security of digital certificates and to avoid them being replaced by fraudulent certificates during a Man-In-The-Middle attack, a technique called *certificate pinning* is

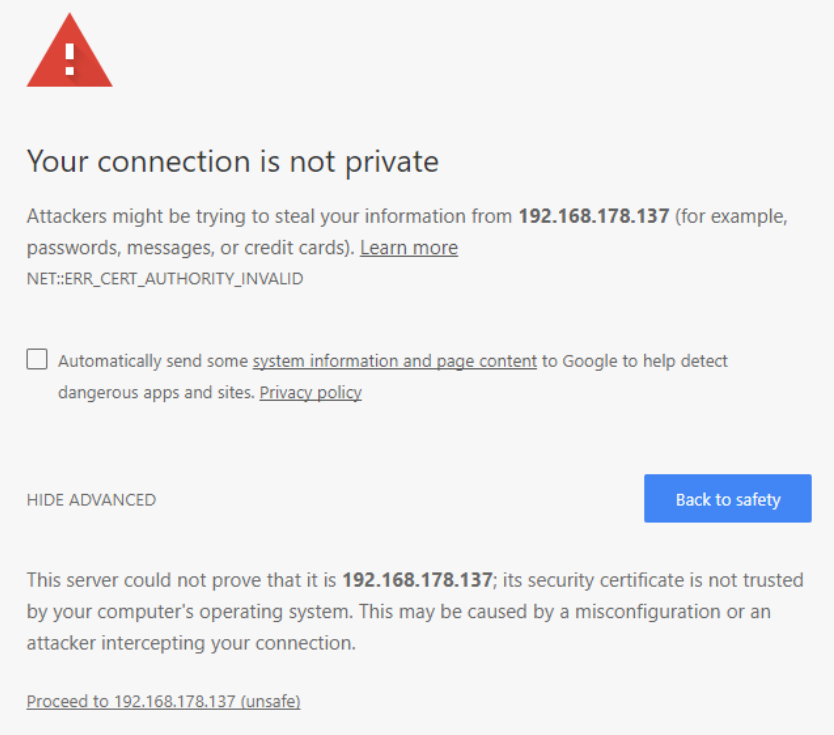

Figure 2.2 An `ERR_CERT_AUTHORITY_INVALID` warning that is displayed when encountering the exemplary self-signed HTTPS certificate for the LAN-internal IP `192.168.178.137` in Google Chrome. Users are free to trust the issuing CA or the certificate manually to prevent this warning. In any case, upon dismissal ("Proceed to 192.168.178.137 (unsafe)"), at least the HTTPS encryption functions correctly.

supported by modern browsers and APIs. Basically, certificate pinning associates an entity with a hash value of the expected digital certificate (or of specific partial information, like the public key). The hash can be transferred in advance by adding a specific HTTPS header. It is then saved upon the first and initial establishment of a connection and remains associated for a certain time span. The latter may lead to problems once the certificate is broken and needs to be replaced by a new one. Web browsers will then compare the new certificate's hash against the associated hash and notice the inconsistency, blocking access to the site until the pinning time span has expired.[6] Another approach is called HTTP Strict Transport Security (HSTS). It defines a special HTTP header, instructing web browsers to request all of the server's contents solely via HTTPS for a specified period of time. This technique prevents access to the site if the certificate cannot be verified, providing no option to ignore warnings, unlike the variant depicted in Figure 2.2.

[6]More information on certificate pinning and some of it's security caveats can be found under [OWAb].

2.3.4 Common Standards for Low Energy Data Transmission

IoT nodes only have very restricted capabilities in terms of processing speed and memory. At the same time, ideally, they should be available upon request, i.e. they should not undergo manual on and off cycles, contrary to regular PCs. Still, for power saving reasons, IoT nodes have so-called sleep cycles. A sleep cycle automatically puts a device in an inactive state where data transfer is not possible for a predefined period of time.

Generally speaking, wireless data transmission consumes a comparatively high amount of power. The energy consumption is especially high when fast data rates or cellular networks are involved. This renders regular network standards impracticable for a high number of continuously active nodes, due to high energy cost. Additionally, many connection-oriented protocols cannot cope with sleeping nodes. As a matter of fact, various standards have been proposed since wireless sensor networks achieved worldwide attention, for example *IEEE 802.15.4*, *IETF 6LoWPAN* and *ZigBee*. [KKK14]

For wireless data transmission, most IoT standards rely on frequencies in the range of 434 to 868 MHz or 2400 MHz. The latter is used by WiFi and Bluetooth Low Energy, for example. These frequencies are *unlicensed*, i.e. (German) federal agencies do not impose a license fee. As a result, there are many devices sharing the frequency bands, leading to potential interference issues and strict transmission regulations. [Ohl13] Typically, Bluetooth Low Energy is the method of choice for short-ranged communication, yielding an approximate range of ten meters. Walls or other obstacles may diminish this value even further.

IEEE 802.15.4 focuses on an architecture that provides low deployment cost, low complexity and low power consumption. Accordingly, the standard defines reasonable Physical and Data Link Layer protocols (cf. OSI-Model, Table 2.1). IEEE 802.15.4 is used for Wireless Personal Area Networks (WPANs), i.e. short-ranged wireless networks intended for mutual communication of devices without the need of Wide Area Network (WAN) access.

The basic IEEE 802.15.4 architecture is extended by ZigBee, a standard defining important communication protocols reaching from the Network Layer to the Application Layer of the OSI-Model (Table 2.1). [KKK14, p. 7754] The ZigBee standard was developed with regard to the behaviour of bees - hence its name. In nature, bee tribes are divided in three different groups: The queen, drones and worker bees. The protocol aims at emulating bee tribes, where simple organisms like worker bees cooperate to tackle complex tasks. [KKK14] A single ZigBee network

supports approximately 65,000 devices. AES encryption is supported, in addition to various amenities, e.g. broadcasts. To facilitate cross-brand integration of devices, multiple profiles like *Home Lighting* were predefined by the *ZigBee Alliance*. These profiles provide adapted protocols for communication, categorized by product type. [KKK14]

ZigBee defines two different access methods, called either *Beacon Enabled* or *Non Beacon Enabled*. The former requires nodes to send and receive data in individual time slots. Every device learns its predefined time slot from constant beacon packets sent by a designated WPAN coordinator. On the contrary, a *Non Beacon Enabled* network allows arbitrary devices to send information whenever the communication medium is available. Optionally, the receipt of network data can be confirmed by returning acknowledgment packages. [KKK14, p. 7753]

Owing to the optimization for low energy applications, the data rate of ZigBee is approximately 250 kBit/s. While this is sufficient for scenarios like controlling home automation devices or status monitoring, ZigBee itself cannot be used for more sophisticated methods like video surveillance. However, the small energy consumption allows for a battery lifespan of multiple years. [KKK14, p. 7752]

DECT ULE is a different standard, based on the cordless phone protocol DECT, adapted to suit low energy needs. It is used for many products manufactured by *Panasonic* and implemented in various WiFi routers like *AVM*'s *FritzBox*. DECT ULE can be integrated into any regular DECT phone system and may be used to control smart home devices. Owing to DECT ULE's licensed transmission frequency band of 1900 MHz, interference with other wireless devices like WiFi access points is greatly decreased while maintaining a potential battery lifespan of ten years. Being optimized for quality telephony applications, this standard provides a high data rate of 1 MBit/s. Even though DECT ULE holds a large approximate range of 300 meters, low interference and high data rates, manufacturers rarely use the standard for new products. [Sec15]

The Z-Wave standard does also use a licensed frequency band. Specifically, Z-Wave transmits data at 900 MHz, resulting in few interference issues. Z-Wave devices may traverse distances of up to 65 meters, while transmitting data at a low rate of 40 kBit/s. [Sec15]

HomeMatic and its successor HomeMatic IP are further, proprietary techniques for low energy data transmission, developed by *eQ-3*.

Technology	Frequency	Data rate	Max. Range	Application
Bluetooth Low Energy	2.4 GHz	1 MBit/s	10m	Cellphone-related accessories
DECT ULE	1.9 GHz (licensed)	1 MBit/s	300m	Cordless communication, home automation
HomeMatic	868 MHz	10 kBit/s	400m	Home automation
HomeMatic IP	868 MHz	10 kBit/s	400m	Home automation
ZigBee *(IEEE 802.15.4)*	2.4 GHz	250 kBit/s	75m	Low power IoT networks in general
Z-Wave *(ITU-T G.9959)*	900 MHz (licensed)	40 kBit/s	65m	Home automation

Table 2.3 An alphabetical comparison of different low energy protocols for smart home devices, based on the information provided in [Sec15], [Ohl13], [AGc, p. 11] and additional e-mail communication with eQ-3 support (cf. Appendix A.2.2), concerning statistics about HomeMatic and HomeMatic IP. Transmission frequencies may vary in different countries. Corresponding standards have been added to the technology name where applicable.

The protocol HomeMatic IP, as opposed to its predecessor, focuses on higher security, an easier installation and the integration of the TCP/IP protocol stack. [AGb]

Table 2.3 compares different standards by highlighting their most important properties.

2.3.5 Remote Prodecure Call

Remote Procedure Calls (RPCs), as the name indicates, are used to remotely call a method, for instance when a client tries to invoke a method found in a program installed on a server. JSON-RPC is one implementation of RPCs. It uses the serialization method JSON and therefore each message is human readable.

An RPC call is structured into requests and responses and can therefore easily be incorporated into HTTP. Requests specify various details regarding the method call. Every request starts with a parameter called `jsonrpc`, necessary to determine the version of JSON-RPC (fixedly valued `2.0` in the current standard). Additionally, the name of the method to be invoked and a structured value containing the parameters to be used for the invocation are included [JSO]. Furthermore, RPC requests contain a field named `id`. The client is thereby able to denominate a request with

Field name	Meaning
code	An integral value indicating the type of error, i.e. `-32601` when a called method does not exist.
message	A textual description of the error that occurred.
data	Optional value that may contain additional information about the error.

Table 2.4 The structure of the error field in case an RPC request was not successfully executed. Cf. [JSO].

a unique string, number or the `NULL` value, though the latter is generally not recommended and numbers should be integral only [JSO].

When the `id` parameter is omitted, requests represent so-called *notifications*. In this case, the client does not demand a response, which also implies that the client is not notified in case of execution errors.

On the contrary, when the request contains a value for `id`, the client requires the server to generate a response JSON object. The response contains a field of arbitrary content named `result`, in case the operation was successful. When a failure occurred instead, a specifically structured field called `error` is transmitted (cf. Table 2.4). To allow for a mapping of request and corresponding response, the parameter `id` of the original message is included as well.

Listing 2.3 shows an exemplary RPC request. To facilitate multiple operations at the same time, it is possible to use an array for the request. Accordingly, the server will respond with an array containing a response for every request element, excluding notifications.

```
1  {
2   "jsonrpc": "2.0",
3   "method": "updateWindow",
4   "id": 4,
5   "params":{
6               "windowID": "1",
7               "open": "true",
8               "locale": "de"
9           }
10 }
```

Listing 2.3 A Remote Procedure Call can be used to manipulate an actuator (for instance, to open a window as shown in this example). Owing to its ASCII encoding, this message can easily be sent via HTTP.

RPCs are used in many smart home systems, mainly for the communication between cloud or remote control and home base (cf. Chapter 2.5), for example in the Qivicon system (cf. Chapter 3.1).

2.4 Threat Modeling

A threat model attempts to enumerate, prioritize and document plausible threats and attack scenarios in a structured manner. There exist many different standards to create an extensive and thorough threat model, for example the *STRIDE*[7] methodology introduced by Microsoft in 1999. Each standard focuses on a different aspect of security. This thesis will refer to the simplified methodology proposed by [Sho14], due to its ease-of-use and applicability in the given context. Basically, a simplified threat modeling process consists of the following four steps (cf. [Sho14, 4 ff.]):

1. **System modeling**
   Gather, structure and organize the individual components of the system, e.g. through architectural diagrams as described for the Bosch Smart Home system in Chapter 3.2.1 or depicted in Figure 2.3.
2. **Threat identification**
   Analyze the system to find weak spots and areas of adversary interest.
3. **Identify security measures to protect system from threats**
   Develop techniques to prevent attackers from exploiting vulnerabilities.
4. **Verification of threat model**
   Check the results and verify that no potential threat was left unnoticed.

More extensive methods or multiple iterations of the above methodology may find additional risks and threats.

The generalized *Internet Threat Model* usually assumes the end-systems of a communication to be secure. In contrast, the communication channel is considered to be compromised, meaning that an adversary can easily intercept and read every transferred packet. Additionally, the attacker is assumed to be able to modify and inject network traffic, including sending network packets that appear to be sent by a different machine. [BSG14b, p. 15]

Attacks can be classified into *active* and *passive* attacks, where the former requires the attacker to actively manipulate and inject traffic into the network and the latter restricts the adversary to read access. Passive attackers can only be excluded by observing quantum states [BSG14b, p. 16] (*quantum cryptography*), which is not feasible with current network technologies at the

[7] Refer to *https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx* for more information on STRIDE.

time of writing (May 2017).

The two attack types are often combined to exploit a security vulnerability. For example, this can be achieved by passively sniffing user credentials and, subsequently, actively generating a network packet incorporating the collected credentials for a malicious login attempt.

A threat model is often accompanied by an attacker definition [Sho14, 34 ff.]. According to [BSG14a, p. 22], attackers can be divided into the following categories:

1. **Insider**
   For instance, employees of the smart home system manufacturer or related developers. Users with access to a Home Automation Network (HAN) (cf. Chapter 2.5), actively (or unintentionally passively, through social engineering, for example) attacking the system may also belong to this category.
2. **Hacker** (also: **cracker**)
   Parties attacking the system out of interest, for fun or for a craving for recognition. Could also refer to burglars or tenants intending to tease or annoy neighbors.
3. **Professional attackers**
   Refers to attacking parties like security agencies or spies.
4. **Organized crime**
   Blackmail or economic and industrial espionage scenarios.

By determining *assets* important to protect from attackers, i.e. valuable components like credential databases, a threat model is subsequently able to precisely construct attacker models and therewith to weigh the most likely attack vectors. Other approaches focus on developing attacker models beforehand. However, they are generally considered to be less effective [Sho14, p. 40].

2.5 Basic Architecture of Smart Home Systems

A smart home system usually consists of a regular computer network and several smart home devices connected to it. According to [Kya17, p. 27], a smart home network can be separated into six building blocks:

- **Controller** (also: **home base, (smart) hub**)

  The controller is the core component of every smart home network. It represents the central unit for all other smart home devices to connect to. Furthermore, it is the main interface for users to communicate with the system. The controller aggregates and processes the information received by the connected devices. Elaborate controllers can react automatically, based on a set of pre- and user-defined rules (e.g. *Turn on the foyer lights when the door is unlocked in the dark*), which is generally referred to as *home automation* [ZWab]. Especially in high security environments, it is desirable for the controller to be connected to an Uninterruptible Power Supply (UPS). The controller does not have to be a proprietary device intended solely for the purpose of creating a smart home; a regular computer running Microsoft Windows or a Unix-like operating system will also suffice.
- **Remote control**

  A remote control allows for an interaction with the controller. Today, as smartphones are very common and broadly available, apps replace the need of separate remote control equipment. Some controllers incorporate an HTTP server for interaction and therewith a web interface for general internet browsers.
- **Devices Under Control (DUCs)**

  A Device Under Control (DUC) represents a smart component that is connected to the controller. For example, equipment like appliances, lights or other electronic devices can be considered as DUCs. Many devices provide interfaces like Bluetooth Low Energy or WiFi to facilitate the connection between DUC and controller (cf. Chapter 2.3.4). Further protocols and wireless standards can often be added by using separate devices named *bridges*, acting as intermediary between DUC and controller.
- **Sensors and actuators**

  Sensors yield important data controllers and DUCs can use as a basis for their decisions. They are available in a wide range of types. For example, audio sensors can be used to detect noise. Additionally, temperature or humidity sensors can be employed for detecting whether a thermostat needs adjustment. Consequently, sensors are used for the *perception* of the physical world. More elaborate devices are usually required to be bought separately, as most introductory home automation packages only contain a limited amount of a few basic sensors.

Actuators, on the contrary, are devices used for an *interaction* with the physical world. They are not limited to simple motors, however, as pumps or electric switches provide actuation as well.

- **Home Automation Network (HAN)**
  The connection between DUCs, sensors/actuators, controller and remote control is established by the Home Automation Network. Typical HANs rely on either wireless (e.g. WiFi) or wired (e.g. Power Line or Ethernet) connections.
  Architecturally, a remote control is not necessarily included in the HAN building block. For instance, in a case where a remote control is not in the immediate range of the network, another interface or special technology (e.g. the cloud or VPN) will be used for the remote to connect to the HAN (cf. [Kya17, p. 37]). The HAN definition will get more complicated when communication between remote control and controller is obligatorily performed via external cloud servers. Therefore, to simplify the architecture, Figure 2.3 and its related adaptions only depict the core HAN, not including the remote control.
- **Cloud**
  Most smart home systems are connected to proprietary cloud servers. The cloud often copes with user authentication and command redirection for users out of range of the HAN. Some systems (like Coqon, cf. Chapter 3.3) are dependent on an available cloud connection to operate correctly. Other systems, like Bosch Smart Home (cf. Chapter 3.2), can be operated autarkically.

When not explicitly stated otherwise, this thesis will use the term DUC summarizing both, the above specification of DUCs and the definition of sensors and actuators, to simplify terminology.

The controller is usually connected to the network router via WiFi or Ethernet and needs LAN access. Power Line is a cheap alternative to laying Ethernet cables throughout the house, as it uses residential power lines to transmit network signals.

If remote access is desired, WAN access is obligatory. Opposed to when the controller is directly connected to the WAN (cf. Coqon, Chapter 3.3), a router's NAT functionality will prevent users from directly connecting to the controller from outside the LAN. Therefore, most smart home systems require the user to register for the cloud. The most common mechanism for cloud communication is called *cloud polling*. The device regularly requests the cloud servers for new commands and will also upload its current status in a timely manner. The controller may also continuously poll for new firmware versions. [BW15, p. 12]

There exist multiple other methods to circumvent some of the restrictions imposed by a NAT to allow incoming connections, for example *TCP Hole Punching*. As none of the systems analyzed in Chapter 3 employs these techniques, smart home architectures where external remote controls communicate directly with the home network through the broadband router will not be considered further.

The controller does at the same time function as an authorizing device and a link between HAN and home LAN. If implemented correctly, it will only accept commands from legitimate users. From the privacy and security perspective, the necessity of connecting to a cloud is considered to be an important issue. Besides its NAT circumvention purposes, large companies also officially use the cloud to collect usage statistics. As for the smart thermostat *NEST*, in addition to the common practice of selling sensitive user data, burglars could use leaked information to draw conclusions on whether house owners are at home. [Kya17, p. 70]

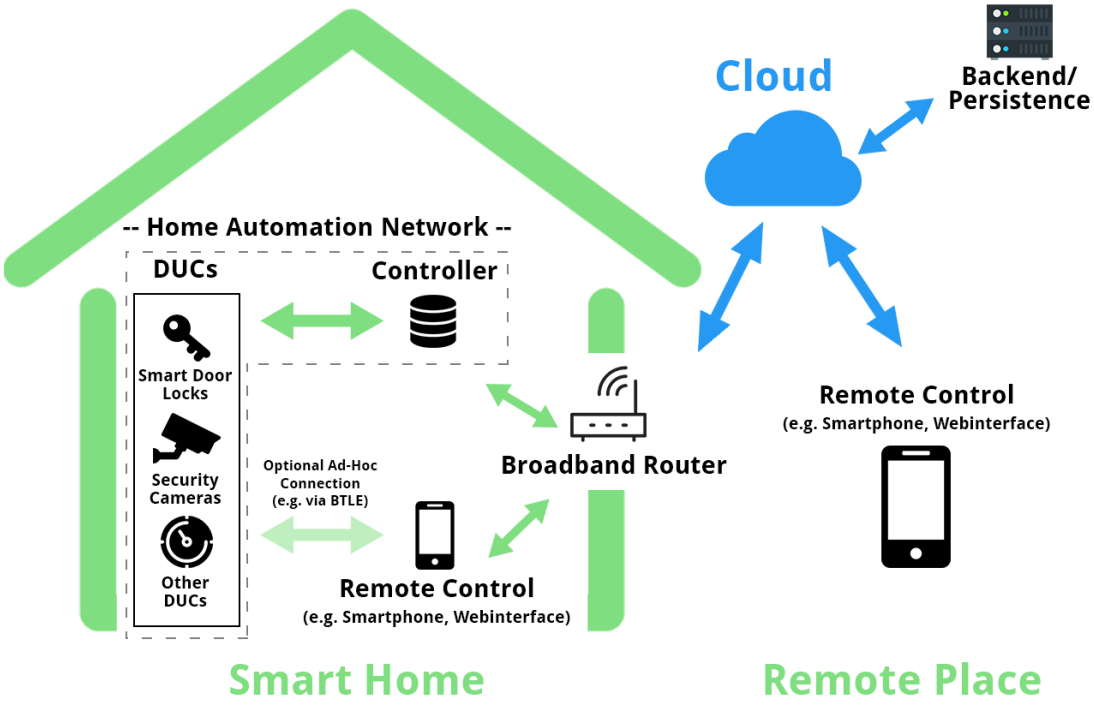

Figure 2.3 An overview of an exemplary smart home architecture. The remote control is not included in the HAN for the reasons described in the according definition on page 23. Graphic adapted on the basis of [Lin14]. Icon source: [FMM].

Every smart home system and transmission standard has its own pairing mechanism to couple DUCs and controllers. The most common ones involve entering the DUC's serial number printed

on the device's label into a controller's interface (i.e. the smartphone app) or pressing designated pairing buttons on both devices. Some smart home devices also employ IP address probing, i.e. sequentially trying to connect to every possible IP address and a specific port, or use a special protocol called Service Discovery Protocol/Universal Plug and Play (SSDP/UPNP) in order to find smart home components. [BW15, p. 12] When third-party DUCs are to be paired with controllers of a different brand, the devices may be intended for different protocols or connection standards. In this case, the controller can be extended by bridges. They aim to translate between the different protocols and connection standards. Bridges also allow for the integration of sensors and actuators that need separate hardware due to computational or architectural limitations.

Figure 2.3 depicts the various components of a typical smart home architecture in relation to each other. The graphic is separated into two use cases: *Smart Home* and *Remote Place*. In addition to the HAN, a direct (ad-hoc) communication between the smartphone and DUCs is shown in the domestic use case. In a situation where the HAN is unavailable due to technical failure, it should still be possible for the remote control to operate the door lock. This can be achieved when the DUC provides an additional technology like RFID to accompany the regular HAN connection mechanisms, for example. However, analyses in Chapter 3 indicated that this method is rarely used.

Multiple large companies have reacted to the trend of smart home systems: Firms like Apple, Google and Samsung have all either created their own proprietary smart home system or have acquired corresponding third parties. Current proprietary systems are usually adaptions from the general architecture mentioned above. Many implementations rely on the open-source home automation framework *openHAB*[8].

[8] *http://www.openhab.org*

2.6 Software Analysis in General

Most software programs can be examined using either *static* or *dynamic* program analysis. The former performs various techniques on the code in its raw, unexecuted state, whereas the latter bases its examination on the program while being executed. For instance, decompiling executables will yield code that can be verified statically, either by using specialized software tools or by investigating manually. In doing so, the internal functionality of a program can be understood. Various techniques that aim at hindering these *reverse engineering* methods, like code obfuscation, necessitate deeper analysis. Thus, when a program is analyzed at runtime, further observations able to verify and extend theses established by static analysis can be made.

To study a program dynamically, techniques like traffic or memory analysis are involved. Special tools like Wireshark intercept network traffic and present it in human-readable form. For some connection scenarios and analysis requirements, in order to being able to intercept the aforementioned traffic, it may be necessary to manipulate the firmware of a network switch or router. This modification will then render the router a recording Man-In-The-Middle, intercepting every received datagram. The many features of Wireshark also allow for a repeat of network packets, i.e. the performance of replay attacks.

Dynamic analysis may also involve the manual or automatic input of (arbitrary) data into GUI interfaces or APIs at runtime, with the intention of finding bugs, exploiting security vulnerabilities or provoking a system or software crash (commonly called *fuzzing*).

The effort of finding vulnerabilities with the above techniques in favor of a more secure software system is called *penetration testing*. Any elaborate software development process should include the latter. Tools like *Burp*[9] automatize some of the most common tests for web applications and quickly present and evaluate results.

[9] *https://portswigger.net/burp*

Chapter 3

Analysis of Commonly Used Smart Home Systems

This chapter presents the results of the analyses of three different smart home systems, namely Qivicon, Bosch Smart Home and Coqon.

First of all, the subsections illustrate the individual system architectures and thereby implicitly support the general architectural model presented in Figure 2.3, also providing an interesting glimpse on the variety of implementation details found in smart home systems currently on the market. Afterwards, the most interesting security issues of each implementation are exposed.

In an effort to present a wide variety of security concerns, the subsections focus on different architectural aspects. For instance, the analysis of Qivicon in Chapter 3.1 mainly presents Android-app-related hazards, which is especially relevant for the *SecureSmartHomeApp* project (cf. Chapter 1.1). Chapter 3.2 explains architectural problems, in addition to app-related issues, by referring to Bosch Smart Home. Finally, Chapter 3.3 summarizes further concerns related to a smart home architecture in general and a typical development process, on the basis of the Coqon system.

The first two chapters are predominantly based on the results of bachelor's and master's theses either associated with the *SecureSmartHomeApp* project or written in advance. As described in Chapter 1.1, these insights could not be verified, as no access to the corresponding smart home systems was available. In accordance with the examiner, the most important outcomes of these documents were summarized and presented in an adequate structure to suit this thesis, without further verification. For more detailed information about specific findings and outcomes of the

individual analyses, reading the corresponding theses is recommended.

Chapter 3.3 is based on an interview performed during the preparation of this thesis. In accordance with the examiner, the correctness of the insights presented in the aforementioned chapter was confirmed by the interview partner, rendering a detailed transcript unnecessary. The confirmation is included in this document (cf. Appendix A.1).

3.1 Qivicon (Deutsche Telekom AG)

At this point, it is noted that the following chapter is primarily based on the findings portrayed in the bachelor's report [Sky17] by Kevin Skyba. As the report was written concurrently to this thesis, in accordance with the examiner, some results are based on personal communication with Skyba and are planned to be included in written form in the final version of his document.

When the first smart home systems evolved, they were prone to incompatibilities and it was necessary to use differently branded controller units in order to supervise every component of the HAN. Different standards and connection technologies competed with each other and several research projects aimed at unifying the communication.

Qivicon, as a result, is a smart home platform created by the initiative of the Deutsche Telekom AG in a joint venture with companies like Miele, Samsung, Philips and others. Since 2013, Qivicon has strived to be a unifying alternative, promoting the development of smart home systems and at the same time providing elaborated home automation technology for a broader audience. The software is based on the open-source automation framework openHAB [Art17]. To establish the HAN, the base supports various connection protocols and home automation systems, like HomeMatic, HomeMatic IP, ZigBee and DECT ULE. Support for additional protocols can be added via USB modules. Therefore, third-party devices can be seamlessly integrated into the system. [Sky17, 5 f.] Qivicon can be adapted and labeled by any license holder because it was designed as a white label product.

Owing to the underlying bachelor's report, the Qivicon variant from Deutsche Telekom AG, also called *Magenta SmartHome*, will be analyzed in the following. Originally, this variant was arbitrarily selected in [Sky17] because it was physically available at the time the corresponding research was planned.

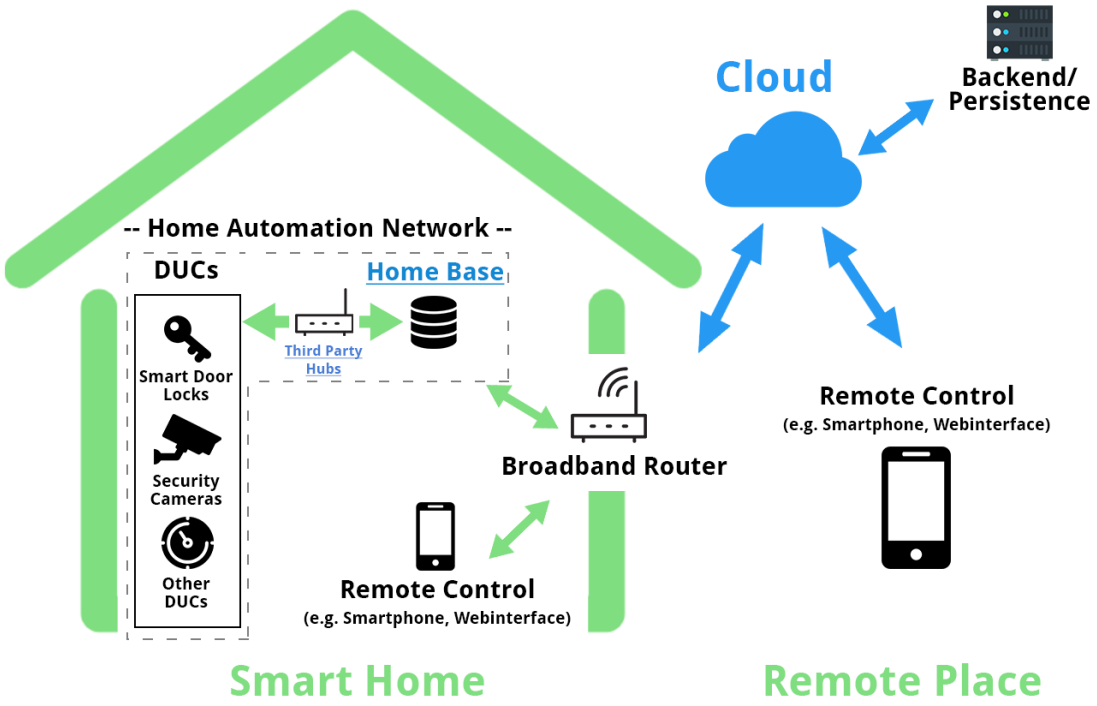

Figure 3.1 An illustration of the Qivicon system architecture, based on [Sky17, p. 6]. Direct connections between remote and DUC, as shown in Figure 2.3, are not intended by Qivicon. Further parts adapted or renamed in the Qivicon context have been underlined in the graphic.

3.1.1 Architecture

Figure 3.1 depicts the general smart home model presented in Chapter 2.5, adapted for Qivicon.

In the Qivicon context, the so-called *Home Base* functions as the smart home controller. It is connected to DUCs, either directly, or indirectly via bridges (called *Hubs*). The home base integrates into the home network by Ethernet cable or WiFi. Upon installation, the user is required to register for and to connect to the Qivicon cloud. Therefore, every Qivicon adaption encryptedly communicates with servers administered by the T-Systems GmbH. [Qiv] According to the Qivicon manual [AGe], the base provides a LAN-internal web interface for local management. After the initial set-up, internet access is not required [Qiv].

DUCs can be programmed to react based on either events or scenes; the former defines reactions to signals received from sensors like motion detectors, whereas the latter refers to more complicated situations in a case where one DUC should react to a state of a different one. Scenes can

also be described as to-do lists for the connected electronics [ZWaa]. An exemplary scene could be a radiator that should be regulated automatically when a nearby window is opened [Sky17, p. 5]. Qivicon refers to the general concept of scenes using the term *situations*. Scenes are saved on the smart home controller and their processing does not need cloud access [Skyba, personal communication].

Though [Qiv] states that Qivicon partners may integrate their own cloud servers, it is not clear whether these servers can only be used for authentication or may cope with command processing as well. When analyzing the APK, Skyba found according leads indicating the former [Skyba, personal communication], restricting third parties in providing own command processing servers. We contacted Deutsche Telekom AG multiple times, in an effort to get a precise statement on what possibilities third-party companies have when providing their own servers. Unfortunately, three e-mails addressed to Telekom's support team remained unanswered and one e-mail directed at Qivicon support received a response stating that the Qivicon team was unable to answer the request (cf. Appendix A.2.1). Hotline support employees were also unable to answer the inquiry. In an effort to gain a specialist telephone number for Qivicon, we visited two different Telekom stores in Delmenhorst and Bremen. However, employees explained the absence of a specialist phone number for in-depth technical details about Qivicon and neither could contact to the development team be provided. The manufacturer's behavior reminds of the *Security-by-Obscurity* principle, where it is often mistakenly believed that hiding details about code and architectural properties will enhance security.

For remotely invoking procedures on the home base, the protocol JSON-RPC is used and data are usually transferred via the Qivicon cloud. Without additional encryption, the JSON-RPC protocol's datagrams would be human-readable and therefore easy to manipulate, as the protocol does not implement security by itself. Responses to commands are also transmitted through the cloud by using the same protocol. If the remote control app is not configured to enable local connections, i.e. limit the communication to the current LAN, cloud servers are employed for the transmission even when the remote control is in the same network as the controller. [Skyba, personal communication] Besides privacy issues, a cloud is generally prone to various security hazards. If the connection is not encrypted properly, attackers could be able to intercept transferred information. Furthermore, as the base inevitably provides an interface for commands received from the WAN, attackers do not need to be connected to the corresponding LAN to send forged commands to the controller. Thus, once cloud servers are compromised, attackers

could gain control over every connected HAN. In addition to security concerns, the cloud needs to be available at any time for external access, which binds the public availability of the system to the goodwill of the provider.

3.1.2 Security Analysis of the App

Skyba analyzed the app in version 4.5.1 by combining static and dynamic program analysis, whereas most of the insights from the dynamic examination were not yet included in the preliminary version [Sky17].

The static analysis was initially concerned with the examination of the app's architecture, revealing the strong dependency on third-party libraries. For example, the app employs a third-party analytics library of an unknown version, created by a company called `AT Internet`. Such frameworks are usually employed to measure an app's or website's performance, for instance by automatically collecting crash report data or by recording the interaction behavior of users.

Further analyses performed in the course of this thesis revealed that the APK contains an outdated version of the `jQuery` JavaScript library. It is saved under `assets\features\src\libs\jquery-1.11.0.js`. The old version `jQuery 1` has received various security and feature improvements, resulting in revision `1.12.4`. In general, the most current version of `jQuery`, as of 23.07.2017, is `3.2.1`. [jQu] Another interesting outcome from our additional analyses is that some resources were created on the basis of *html5-boilerplate*[1], a framework that is usually employed for quickly creating HTML 5 web interfaces. This assumption is made from various HTML comments in `assets\registration\info_de.html`, like the one quoted in Listing 3.1.

```
6     <!-- Always force latest IE rendering engine (even in intranet) & ↩
          Chrome Frame.
7         Remove this if you use the .htaccess -->
```

Listing 3.1 Various hints contained in the Qivicon APK suggest the use of third-party frameworks. This comment was found in `assets\registration\info_de.html`.

A Google search confirmed that this text, among others, is contained in the framework's source code. However, the HTML code snippets found were not considered security relevant. Still, it is interesting to notice that, even though the HTML file's structure is very simple and far from complicated, developers made the effort to refer to a third-party framework. The reason for

[1] *https://github.com/h5bp/html5-boilerplate*

this decision is unclear. Perhaps developers used the framework in other, more extensive parts of the system as well, for example the controller's LAN-internal web interface. Owing to the absence of a controller for testing, this assumption could not be verified. Surprisingly, according to the framework's changelog, the last update was in the beginning of 2016, more than one and a half years ago as of the time of writing. This leads to the conclusion that the framework is not maintained in a timely manner.

When the code was analyzed by Skyba, he also found out that the framework *Otto*[2] was used, a library simplifying the binding of methods to events [Sky17, p. 42]. Even though the usage of third-party libraries is common, external frameworks are often prone to security flaws and generally impose further risks.[3]

The app contains Android's *WebView* element, which is basically an integrated web browser that enables apps to display HTML content. These GUI controls can prove problematic, when external resources are requested dynamically or JavaScript code execution is enabled. An analysis of the APK revealed a folder structure named `assets\features`, containing various HTML and JavaScript files, intended for advertising new app features [Sky17, p. 38]. The plain HTML files do not request additional resources from remote servers and are considered secure. The absence of external requests prevents security risks like Man-In-The-Middle attacks. Another collection of HTML and CSS files can be found under `assets\registration`. The HTML files provide various forms used in connection with the user's registration process and do also not load resources dynamically over the World Wide Web. In order for the user to proceed with the registration process, the file `assets\registration\index-de.html` contains an external link to `https://www.qivicon.com/register/de/PADE2110412057825/`. The HTTPS protocol and the according authentication process in connection with digital certificates do actively prevent attack scenarios where a different server could identify itself as `www.qivicon.com` and intercept data using phishing methods. Also, as the forms are included in the APK and are not loaded from a web resource, they cannot be easily replaced by phishing variants. This is considered best-practice.

```
1    endpoint.oauth=https://global.telekom.com/gcp-web-api/oauth
2    endpoint.oauth.token=https://global.telekom.com/gcp-web-api/oauth
3    endpoint.remote=https://acs.qivicon.com/mprm/remote/json-rpc
4    endpoint.backend=https://acs.qivicon.com/mprm/backend/remote/json
5    -rpc
6    endpoint.websockets=wss://acs.qivicon.com:44304/remote/events/
```

[2] *http://square.github.io/otto/*
[3] The security issues and statistics described on [Wis16] can be considered interesting in this context.

```
7    app.url=http://localhost/app01
```

Listing 3.2 On page 38, Skyba shows an extract of the data found in `qivicon.connector.properties`.

Analyses of configuration files contained in the APK archive revealed additional URLs pointing to Telekom and Qivicon servers. The file `qivicon.connector.properties`, shown as an extract in Listing 3.2, stores the URLs `https://global.telekom.com/gcp-web-api/oauth` and `https://acs.qivicon.com/mprm/backend/remote/json-rpc` in fields called `endpoint.oauth` and `endpoint.backend`, respectively. The URLs indicate that the OAuth protocol is used for authentication and JSON-RPC for remote procedure calls. The security of both does greatly depend on a correct implementation. Generally, both calls are secured via the HTTPS protocol, which can be considered *state-of-the-art*, if implemented correctly. However, static code analysis of the APK revealed an insecure handling of OAuth tokens. Class `com.qivicon.client.android.storage.SecurePersistentTokenStorage` contains an implementation for the encrypted storage of the aforementioned token. Its constructor provides an argument called `passPhrase`, being the key used for encryption. In the app's network implementation in `de.telekom.networking.QiviconClient`, a `SecurePersistentTokenStorage` object is instantiated using a short static (i.e. hard-coded) passphrase, namely `sPjmCqLtWbLXiuoR`. The encryption method is called every time a new sensitive OAuth token is generated. Precisely, the encryption mechanism used is `PBEWithMD5AndDES`. The latter is considered insecure: The Data Encryption Standard (DES) algorithm uses a small key size and is well-known for security vulnerabilities. It is unknown why Qivicon's developers did not refer to the more secure Triple DES (3DES) or Advanced Encryption Standard (AES) variants, as each one could be implemented by substituting a few lines of code. [Skyba, personal communication]

Contrary to the iOS version of the Qivicon app, the Android version uses certificate pinning (cf. Chapter 2.3.3) to prevent Man-In-The-Middle attacks. It is unclear why the iOS version does not implement aforementioned technique, as it is regarded as an effective method heavily increasing the necessary effort for according attacks. [Skyba, personal communication]

3.2 Bosch Smart Home (Robert Bosch Smart Home GmbH)

This chapter is based on the findings described in the master's thesis [Kol17] by Philipp Kolloge and the bachelor's report [Bar16] by Jan Bartkowski. As [Kol17] was written concurrently to this thesis, the main focus will remain on [Bar16]. Contrary to Kolloge, Bartkowski did focus his research on the system's LAN-internal communication, with disabled cloud (i.e. remote access) functionality.

Both documents base their research results on the *Bosch Smart Home Raumklima Starter-Paket*, which contains three DUCs, in addition to one controller: Two *Bosch Smart Home Radiator Thermostat* units and a single *Bosch Smart Home Door/Window Contact* element. Each DUC receives firmware updates automatically once it is connected to the controller [Homb].

3.2.1 Architecture

Robert Bosch Smart Home GmbH's proprietary smart home system, namely *Bosch Smart Home*, is not based on a white label product like Qivicon. The system can receive commands through the cloud, though cloud access and cloud registration are not obligatory. This is a key difference to Qivicon and Coqon (cf. Chapters 3.1 and 3.3). Personal data are only stored on the controller unit, except for cases where the user does explicitly enable cloud access. In this case, any data are transferred encryptedly. [Homa] As for Qivicon, DUCs can be combined for scenes, i.e. complex behavior for home automation.

In addition to the complete system (*Bosch Smart System Solutions*), consisting of DUCs connected to a controller unit, Bosch also offers so-called *Smart Single Solutions*. The latter is recommended when a more simple solution without a smart home controller is desired for specific use cases. Currently, Bosch offers three different Single Solutions, whereas each requires a separate controlling app: Two different security cameras and a smoke detector, called *Twinguard*. In future versions of Bosch Smart Home, Single Solution devices will be integrable into the complete system. [Homa] To integrate into the home LAN and in order to being accessible by a remote control, Twinguard requires a bridge, called *gateway*. Bosch's security cameras, on the contrary, only require a WiFi signal. Apps can then communicate with these cameras directly, without the need for a separate controller or gateway. [Homa] This is another key difference to

Name	Value	Data Format
Mac	The Ethernet MAC address of the controller.	A unique 48-bit value, different for every unit.
Key	A string that is used as a second factor for pairing, to prevent attackers from generating arbitrary MAC addresses.	A 16-character string, consisting of arbitrarily cased letters and numbers.
S/N	A unique serial number.	A 23-digit number.

Table 3.1 Information used to identify the home base unit, according to [Bar16, 26 f.,35] and [Kolloge, personal communication]. The QR-code encodes `Mac` and `Key`. Hence, bold values are required for pairing.

the other smart home systems examined in this thesis. Single Solutions were neither tested by Bartkowski nor Kolloge and are therefore not considered further, though a security analysis of the direct communication could yield additional interesting insights.

Controller and bridge are integrated into the home network via LAN cable. No WiFi interface is provided. Initially, to improve usability, the app uses an integrated QR-code scanner to pair controller and app. The QR-code is printed on the back of the controller and contains two fields, organised in the JSON format: `id` and `ss`. The former represents the MAC address of the controller, whereas the latter is a 16-character code equal to the value printed next to `Key` on the label on the controller's back side. [Bar16, p. 27] It is used as a second-factor mechanism to ensure that the user does really possess the device in question and helps to prevent brute-force attacks. Alternatively, the user may enter the corresponding identification numbers manually. The pairing mechanism can only be completed successfully when a designated button on the controller is pressed for three seconds; a third factor, used to authenticate the pairing request. [Bar16]

The process to add single DUCs is similar. When a DUC is started for the first time, the pairing process is initiated automatically. Every device is also labeled with identification numbers and an accompanying QR-code. However, the code's content is different for DUCs. It resolves to a 65-digit alphanumeric string, not encoded in JSON form. It does contain a 24-character `SGTIN` value from the DUC's label, however the meaning of the remaining characters is not clear. Bartkowski, p. 27 assumes that the QR-code also encodes the DUC label's `Key` value in encrypted or hashed form. He was not able to verify his assumption and it is also unclear why the QR-code's basic format is not identical for both, DUCs and controllers.

Name	Value	Data Format
SGTIN	A unique value referencing manufacturer and item.	A 24-character value.
Key	A string that is used as a second factor for pairing, to prevent attackers from generating arbitrary `SGTIN`s.	A 25-character string.

Table 3.2 Information used to identify DUCs, according to [Bar16, 26 f.] and [GS111, p. 30]. Both values need to be entered manually in case the user does decide not to scan the QR-code.

The controller's label contains three fields, precisely `Mac`, `Key` and the serial number, labeled `S/N`. DUC labels, on the other hand, contain only two values, `SGTIN` and `Key`. The different fields are explained in Tables 3.1 and 3.2. Both tables illustrate the large potential trial-and-error domain for brute-force attacks, occurring due to the second-factor mechanism.

To facilitate hazard analysis, Bartkowski created the system overview depicted in Figure 3.2. It illustrates the various components involved in the communication process. Dashed lines indicate trust zones, i.e. system components belonging together due to bundled distribution. Elements of the same zone can mutually assume each other to be trustworthy in terms of security, because in general use cases, their internal components cannot be manipulated by attackers due to various protection mechanisms imposed by operating systems.[4] The development of such diagrams is an excellent starting point for the first step of threat modeling, namely *System Modeling* (cf. Chapter 2.4). Therefore, the following paragraphs are dedicated to describe the figure in detail.

Figure 3.2 can be separated into three main parts. Its top section refers to Bosch servers, also referred to as the *Bosch Smart Home Cloud*. When remote access is disabled, these servers are almost exclusively contacted for clock synchronization via the NTP protocol; one server is used for HTTPS connections, though Bartkowski was not able to examine the specific communication due to a strong TLS cipher suite [Bar16, p. 52].

The bottom part of Figure 3.2 depicts the two components *Controller* and *App* in detail. The controller provides ports 8443 and 5353 for network internal communication, specifically used for TCP and MDNS protocols, respectively. The protocol MDNS generally supports device discovery, whereas TCP is used for command transfer. As TCP, in its basic form, is not encrypted, the

[4] According protection techniques of operating systems are outside the scope of this thesis and will therefore not be considered further.

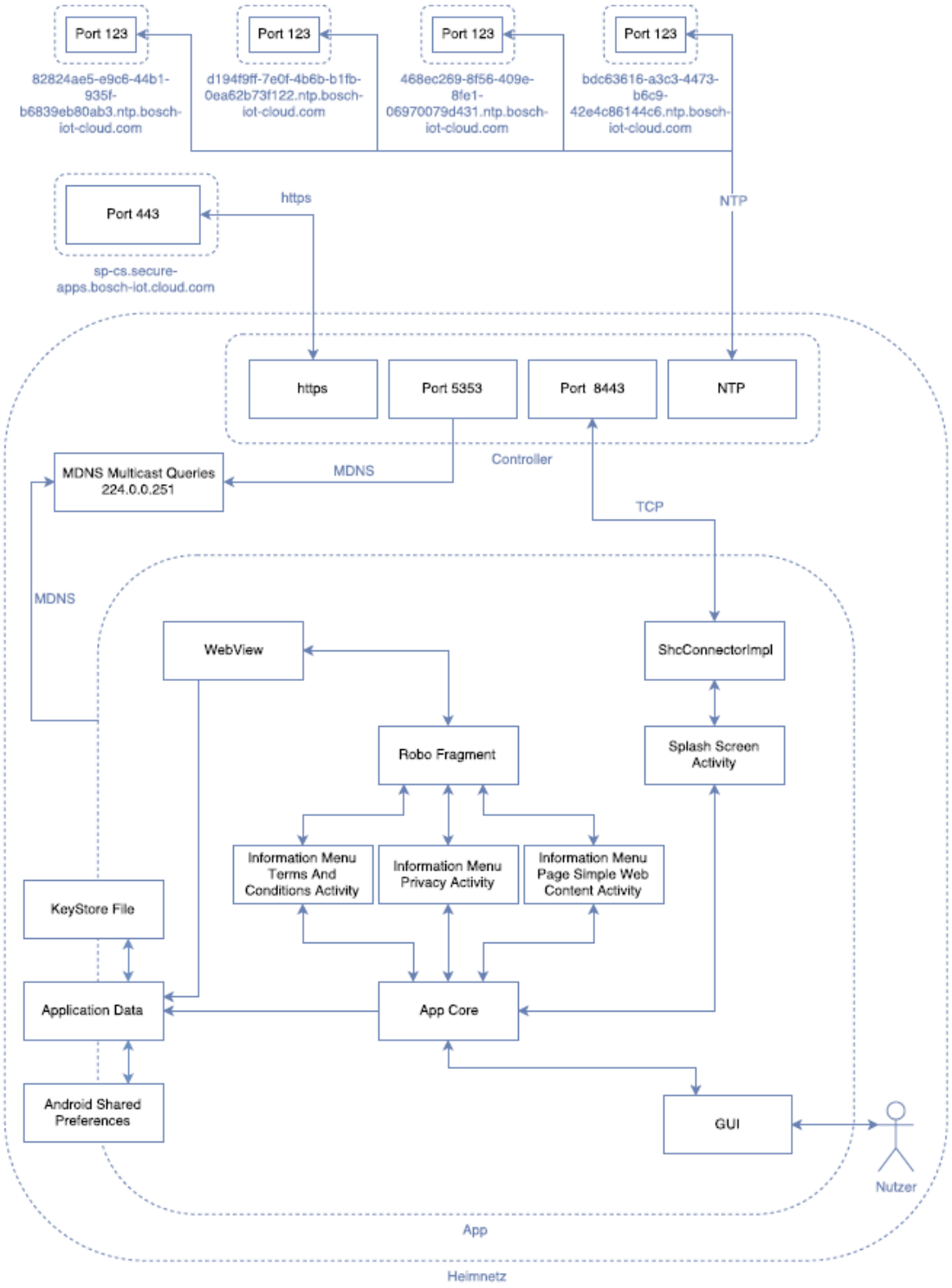

Figure 3.2 A more detailed view on the architecture of the Bosch Smart Home system that was obtained by combining static and dynamic analyses. Source: [Bar16, p. 23]

two components have to implement their own encryption algorithm.

To retain clarity, Figure 3.2 groups most GUI classes and code unimportant to the scope of this analysis together into the abstract objects labeled *GUI* and *App Core*, respectively. A user will interact with the GUI object and is a potential attacker. He may perform malicious actions (fuzzing) and therewith endanger security, hence he is located outside of the app's trust zone. Images or other resources of arbitrary format are represented by an abstract object called *Application Data*. Any app can easily save key-value pairs, using the so-called *Android Shared Preferences* API. Additionally, the app saves encryption parameters in a secure KeyStore file. An attacker may access this information if he is able to gain root privileges. In real-world scenarios, this is difficult to achieve. As a result, the data storage was positioned on the trust zone line.

The class `ShcConnectorImpl` provides methods for communicating with the controller. When remote access is disabled, Bartkowski was not able to log any WAN connections made by the app. This is in accordance with Bosch's claim on [Homa], stating that the Bosch Smart Home System can be installed autarkically from the cloud by communicating predominantly LAN-internally. Bosch notes, however, that at least the controller will use the World Wide Web for time synchronization and update requests, even when remote access is disabled [Homa]. Therefore, internet access is still mandatory.

Figure 3.3 shows the adapted version of the general architecture originally presented in Chapter 2.5.

3.2.2 Methodology for Traffic and Port Analysis

As the Bosch Smart Home controller solely communicates via Ethernet, simple WiFi sniffing methods were not applicable to analyze the network traffic. Thus, in order to capture the network traffic and derive the insights presented in the following subsections, Bartkowski installed the custom firmware *OpenWRT*[5] on his *TP-Link WR841N* router and configured it to act as a network switch. The Linux variant OpenWRT was then extended by a package called *tcpdump-mini* and configured to record the network traffic to a *pcap* file. The latter could then be received via SCP and examined manually in a network analysis program like Wireshark.

[5] *http://www.openwrt.org*

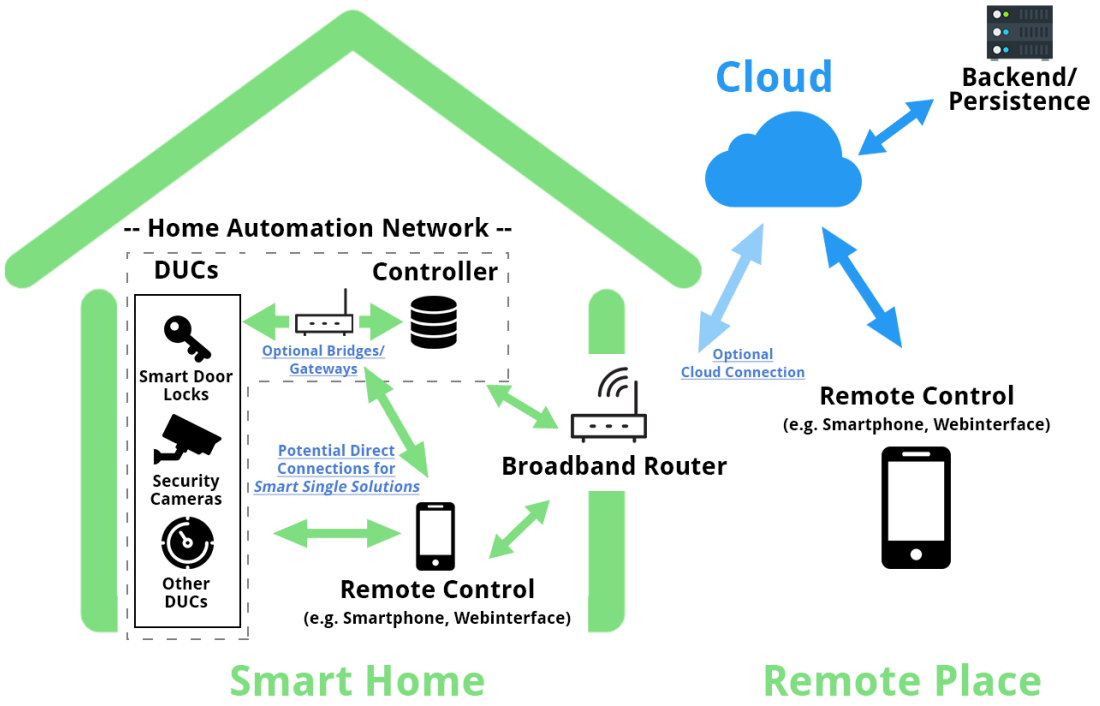

Figure 3.3 An illustration of the Bosch Smart Home system architecture. Parts adapted or renamed in relation to Figure 2.3 have been underlined in the graphic. Even though the cloud connection is optional, internet access is mandatory.

An `nmap` network port scan on the controller revealed a single open port, namely port number 8443, which is offering an HTTPS service. It is used for the remote control to connect to the home base. This is a positive result, as no unnecessary applications are listening for incoming connections, keeping the attack surface reasonably small.

3.2.3 Security Analysis of the App

In the iOS version of the app, specifically app version 5.0.3, Bartkowski, p. 29 found a security vulnerability. He was able to view the password used for the smart home controller in plaintext form. To exploit this vulnerability, it is necessary to stop the app. Then, the phone has to be disconnected from any data network (WiFi and cellular), for example by switching it into airplane mode. Starting the Bosch app will then lead to an error message, stating that no internet connection is available. Subsequently, dismissing the message reveals the login screen.

Owing to a checkbox, labeled *Passwort anzeigen* (display password), he was able to view his original password in the corresponding text box. Exploiting the described security problem involves physical access to the affected device and requires the user to abstain from enabling passcode or TouchID functionality. The password being displayed in plaintext form indicates that the app is internally saving the login credentials without encryption or hashing algorithms. This problem gets even more critical considering that many internet users will utilize the same password for multiple services and websites, for example their e-mail accounts.

Even though the iOS operating system generally prevents file and preferences access across apps, some phone users perform a modification called *jailbreak*, where unsigned third-party apps may illegitimately gain access to these data.[6] In this case, physical access to the device is no longer necessary. Instead, a malicious app stealing the unencrypted credentials needs to be actively installed by the user. Potential security vulnerabilities imposed by the jailbreak may render the iOS device insecure and therefore also allow remote attacks over the internet.

The Android version (specifically 5.0.0-prod) did not show this vulnerability, as the error message was displayed on a screen-filling activity which cannot be closed.

Bosch reacted to an e-mail about this issue by Bartkowski, responding that the password is saved using Apple's secure Keychain mechanism. Furthermore, according to Bosch, jailbreak users were reducing the device's security at their own risk. They also confirmed a security fix for the checkbox problem for the subsequent app version. Unfortunately, without physical access to the system, it was not possible for us to verify whether the security vulnerability was fixed since Bosch's response, as the app does not allow for the input of user credentials until the pairing mechanism (cf. Chapter 3.2.1) is completed successfully. Philipp Kolloge focused his research on the Android version and was also not able to verify the claim. However, generally speaking, contrary to the behavior of the Deutsche Telekom AG described in Chapter 3.1.1, Bosch's response is assessed to be more appropriate and considerate when dealing with security.

Various tests revealed that the Bosch Smart Home app is not vulnerable to XSS or SQL injection attacks. Potential malicious strings were saved and displayed correctly throughout the app. Generally, as login credentials are necessary to access input elements potentially prone to XSS or SQL injection attacks, the risk of attackers exploiting corresponding vulnerabilities is low.

As soon as an `SGTIN` number is entered manually to pair a DUC, the app checks if the corre-

[6] More information on iOS jailbreaks and their security problems can be found on [Inc].

sponding device is already known to the system and potentially displays a message, stating that the device is already paired [Bar16, p. 27]. This happens without the input of the accompanying `Key` value. Theoretically, according to Bartkowski, p. 27, this behavior may be exploited to determine `SGTIN`s for devices that are integrated into the system on a trial-and-error basis. Depending on the specification of the `SGTIN` format, generating and testing every possible value on a trial-and-error basis may need the large amount of more than nine decillion tries (cf. Formula 3.1).

$$(26 \text{ char. in alphabet})^{24 \text{ char. SGTIN length}} \approx 9.11 * 10^{33} \text{ tries} \tag{3.1}$$

However, as the `SGTIN` identifies the *manufacturer* in addition to the product itself (cf. Table 3.2), the value is expected to contain fixed characters. Depending on the specific format, this may reduce the brute-forcing effort to a more reasonable level.

Another interesting issue is the use of digital certificates in the Bosch app. As Bartkowski, p. 37 found out, the app generates a certificate that is probably used for the remote access functionality, to provide a way of identifying the device. He was not able to encounter it being transferred in the course of his analysis, probably due to disabled remote access. Nevertheless, code analysis revealed that the certificate is created with an expiration period of 100 years, indicating that a renewal of a generated certificate is not intended. The certificate is saved to the secure *KeyStore File* element of Figure 3.2, which is considered best-practice. It is possible (and recommended) that developers use the self-generated certificate for TLS mutual authentication.

Subsequently, Bartkowski analyzed whether it was possible for remote controls to connect to the controller from outside the home LAN, even though the remote access functionality was disabled in-app. Therefore, he examined how the system components identify the home network. Analysis of the *Android Shared Preferences* (cf. Figure 3.2) indicated that the home network's SSID is stored in a file named `modellayer.persistence.preferences.xml` [Bar16, p. 35]. The designated `isConnectedToHomeWifi()` method, found in class `ShcConnectorImpl`, checks if the app user is connected to a WiFi network with the SSID saved in the preferences. Further analyses confirmed that the home network recognition is based solely on the comparison of two SSIDs, which cannot be considered secure, as the SSID is an arbitrary string and not a unique identifier for wireless networks. Additionally, it was discovered that any SSID is accepted if the `HomeWifiSSID` preference entry is set to `null`, which happens when the app is installed for the first time, for example. [Bar16, p. 40]

An attempt to connect to the controller via the cellular network failed because the `isConnectedToHomeWifi()` function calls the utility method `SystemServiceUtils.isWifiConnectedToSSID()`, which compares the two SSIDs. Prior to the comparison, a simple `isWifiConnected()` check is performed. Owing to the connection to the cellular network, the latter will return `false`, which will in turn return a negative result for `isConnectedToHomeWifi()`.[7]

In another scenario, Bartkowski set up a separate WiFi network with a different SSID and altered the `HomeWifiSSID` value stored in the app's preferences. Subsequently, the remote control was connected to the new network. Upon start, as the current WiFi SSID and `HomeWifiSSID` matched, the app did indeed try to connect to the controller [Bar16, p. 43]. However the connection failed because the controller was still connected to the original WiFi network and the app uses private IP addresses for local communication. Nevertheless, the user is able to enter the controller's IP address manually. NAT technology generally prevents direct LAN host access from the WAN. However, in case of misconfigured or compromised network routers and when more elaborate techniques like port forwarding are involved, attackers may use the app to communicate with a controller directly, across different networks.

Additionally, once `HomeWifiSSID` is set manually in the app's preferences and remote control and controller are linked to the same network of an SSID different to the one stored in the preferences, the remote control will still connect to the controller locally. This leads to the conclusion that a connection attempt to the controller's saved local IP address is made before the home network checks are performed. [Bar16, p. 44] As a result, when remote control and controller are in different networks, a malicious host may exploit this behavior, assuming the controller's identity by acting as a Man-In-The-Middle. Owing to limited experimental conditions, Bartkowski, p. 45 was not able to check whether the controller verifies the origin of received requests, though this behavior is greatly encouraged. Also, the system should rely on the WiFi Access Point (AP)'s unique BSSID value instead of the arbitrary SSID. However, different APs for the same network will have different BSSIDs, raising the need for a method to detect the BSSID of every associated AP.

[7] The corresponding source code extracted from the app can be found in [Bar16, p. 41].

3.2.4 Security Analysis of the Controller

Analyzing the controller is more difficult because of its blackbox property [Bar16, p. 47]. Neither compiled nor decompiled firmware code was available for examination, which rendered well-known reverse engineering techniques inapplicable. Accordingly, the controller's network interfaces and connections, being the only digitally accessible channel and material, were examined.

Digital certificates need reliable time information for clients to assure their validity. Hardware clocks constantly need electric power; additionally, it can never be guaranteed that users set the clock correctly. Consequently, as the Bosch Smart Home controller is intended for continuous use, it does not contain a hardware battery and relies on the Network Time Protocol (NTP) for clock synchronization. The controller synchronizes the internal clock automatically during every boot process [Bar16, p. 47], accessing four different servers four times. Though the reason for the high amount of requests is unclear [Bar16, p. 49], this algorithm renders internet access mandatory for operation. Bartkowski did not state whether the NTP implementation uses encryption or authentication to prevent tampering, as the clock server's IP addresses are retrieved using the susceptible DNS protocol.

After the initial time synchronization, the controller connects to Bosch's servers in constant time intervals of 300 seconds. The HTTPS connection is encrypted using TLS 1.2, which is the recommended version for encrypted TCP connections and, when set up correctly, is currently considered secure (cf. Chapter 2.3.2). A secure setup depends on the chosen cipher suite. For Bosch Smart Home, server and controller agree on `TLS_DHE_RSA_WITH_AES_256_GCM_SHA384` [Bar16, p. 51], a cipher suite that is recommended and contained in the security checklist of the BSI (cf. [Sic17, p. 10]). However, as TLS 1.3 standardization is in progress, future updates should focus on implementing TLS 1.3. Even though the controller requests the IP address of the CA's CRL server via DNS upon the establishment of the timed HTTPS connections, a transfer of the corresponding list could not be observed [Bar16, p. 51]. Therefore, at least for this connection, it is assumed that the app does not check the list, which is a security hazard in case the manufacturers certificate is broken. The malicious party could then spuriously identify its server as Bosch's, which could in turn lead to an interception or manipulation of sensitive data sent by the controller, depending on the type of information that is transferred every 300 seconds.[8] Owing to the disabled remote access functionality, Bartkowski did not test whether

[8] Bartkowski was not able to further examine the specific data that was transferred, due to its secure TLS encryption.

the CRL is retrieved when the controller receives commands over the internet. If the CRL check is omitted there as well, an attacker in possession of the private key corresponding to Bosch's certificate could potentially control any of their smart home controllers.

Whenever the app initializes a login process, Bartkowski found out that, even though the controller provides a LAN-local HTTPS service on port 8443, a seemingly proprietary protocol is used, probably due to the difficulties of HTTPS for LAN-internal communication. The protocol's contents were not dechipherable because of their unknown structure. Nevertheless, the word `user` was legible in some packets, leading to the conclusion that the login credentials might be encoded, facilitating replay attacks. Attackers with further knowledge of the protocol structure could easily incorporate the data in new packets, in case no signed nonce values are used as countermeasures. As data packets of multiple login attempts contained the same payload, it can be assumed that the system repeatedly uses the same static encryption key for its proprietary protocol - or no encryption at all [Bar16, p. 56].

3.3 Coqon (neusta next GmbH & Co. KG)

This chapter is based on an interview with Kevin Löhmann, project manager of neusta mobile solutions GmbH, actively involved in the development of Coqon. The session took place on the 23.06.2017 at the University of Bremen. It aimed to discover basic properties of Coqon and to assess the system's security situation. The latter involved a combination of both, questions about security problems solved in the past and a questionnaire about Coqon's current communication architecture.[9]

3.3.1 Architecture

Coqon consists of the *Q-Box* (i.e. the smart home controller) and DUCs named *QLight, QHeat, QWeather, etc.*, depending on the use case. The controller can be integrated into the home network either by WiFi or by Ethernet. Being developed since the end of 2014 and released to

[9]The interview guideline can be found in the Appendix, A.1. Instead of transcribing the whole interview, in accordance with the examiner, Kevin Löhmann confirmed the validity of the statements found in this chapter in an e-mail, also found in Appendix A.1.

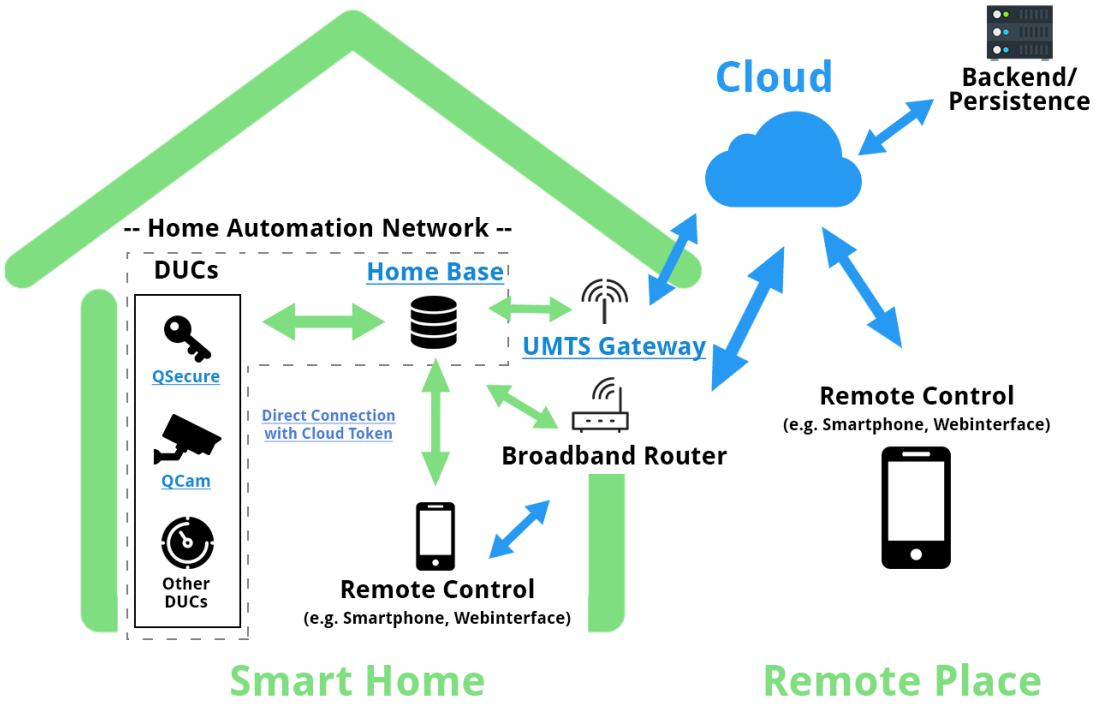

Figure 3.4 An illustration of the Coqon architecture. Cloud access is obligatory, hence the blue color of the remote control's domestic connection. It is used to initiate the LAN-internal communication between controller and remote control. Parts adapted or renamed in relation to Figure 2.3 have been underlined in the graphic.

the market in 2015, the system is relatively new. As for Qivicon, it is developed on the basis of the openHAB framework. The latter was extended by various plugins. For example, the original openHAB Z-Wave plugin did not pass the official Z-Wave certification process, impeding acquisition of official Z-Wave hardware.

Similar to Qivicon and Bosch Smart Home, Coqon embeds its program logic into the controller. This facilitates network-local connections and provides a certain degree of independence from cloud servers. The system does also allow for the specification of complex home automation scenes. Scene rules are defined based on an `If Cond_1 [and Cond_n]* then Action` grammar format.

Contrary to the other smart home systems that were analyzed in the course of this thesis, Coqon offers a trusted UMTS channel connected to the cloud. UMTS network coverage is therefore obligatory for Coqon and has two main purposes: Most of the time, it is used in cooperation with the regular WAN access through the user's broadband internet gateway to improve security.

Every UMTS connection is established through an encrypted VPN tunnel. An initial registration and pairing process through a web interface located at *https://cloud.coqon.de/* facilitates the mapping of user accounts to the corresponding smart home controller hardware. During the registration process, the controller's serial number is used to identify and activate the specific device. After the initial registration, this trusted channel is used to securely transfer tokens and other sensitive data. However, as the use of data networks intended for mobile communication is generally associated with operating costs, usage fees are imposed in two-year intervals, whereas the first interval is included with the acquisition of Coqon.

A second property of the UMTS channel is the ability of being used as a backup medium in case the regular World Wide Web connection fails. Even though functionality over UMTS is reduced, users can still operate the system with a temporary broadband outage. Absent features for UMTS-only connections include video streaming from surveillance cameras and firmware updates, due to high bandwidth requirements and data size. On the contrary, when the UMTS connection is lost or fails, the broadband connection is used alternatively. In case both connections are unavailable, messages destined for the cloud are cached and delivered as soon as the connection is reestablished.

The most important wireless DUC transmission standards for Coqon are Z-Wave and a proprietary alternative, namely *Q-Wave*. Being a company secret, further information about Q-Wave was not provided. The DUCs offered in the company's online store include various devices supporting both Z-Wave and Q-Wave, depending on the usage scenario. For example, smart wall-mounted buttons usually base their communication on the Q-Wave protocol, whereas other devices like smart thermostats employ Z-Wave. The protocol of choice depends on the manufacturer of the specific DUC and does not follow specific guidelines. Upon the specification of Coqon, ZigBee was rejected due to a smaller ecosystem, contrary to the more common alternative Z-Wave. As opposed to Qivicon, for example, Coqon cannot be extended by additional connection protocols using simple USB ports. Specifically, the controller does provide an internal expansion slot, where a new logic board can be plugged in, ideally by a qualified technician. Currently, no extension board is available, though offering additional boards is considered possible for a future release.

Generally speaking, Coqon focuses on implementing a secure alternative to existing smart home solutions. This is also demonstrated on the corresponding website[10], where the security of the

[10] *http://www.coqon.de*

system is emphasized.

As a result, communication with the controller does always require cloud authorization. Direct connections between remote control and DUCs are not intended. LAN-internal communication between remote control and controller is initialized by the cloud as well. The cloud servers return a token which can be used by the remote control to authenticate network packets. Concurrently, the controller receives this token via the secure UMTS channel. The strong cloud dependency prevents access to all digital Q-Box user interfaces once the manufacturer's servers or the internet connections become unavailable.

To improve usability, the controller does not provide a LAN-internal HTTP platform; every communication with the controller is either performed via the Android or iOS apps or the public web interface *https://cloud.coqon.de/*. The fixed URL of the latter, as opposed to LAN-internal IP addresses or LAN-internal URLs, simplifies remembrance. Users do not have to memorize multiple network-dependent addresses and have a single, static location for their web control interface.

3.3.2 Security-Relevant Insights from the Interview

Software security is a sensitive topic, accordingly not every question of the list in Appendix A.1 could be answered in reasonable detail. Therefore, this chapter cannot examine every issue with full accuracy. However, the interview's questions brought up multiple topics that were considered interesting by Kevin Löhmann; the questionnaire revealed system issues still in need of security optimization. He confirmed that, even though Coqon aims to be one of the most secure smart home systems, some security topics are still up for discussion, as few best-practices or standards exist coping with the special requirements of IoT networks and LAN-internal communication. Owing to the novelty of the smart home market, many of the techniques are in a work-in-progress state and poorly reviewed, making them inapplicable or hazardous for security-relevant scenarios.

First, it was verified whether the usage of UMTS jamming devices could lead to a service interruption of Coqon. This is especially relevant for cases where burglars jam UMTS to prevent important signals from motion detectors or other security-relevant devices from reaching the cloud and eventually the end user. However, as Löhmann stated, jamming the UMTS connection will lead to Coqon using broadband as a backup medium. As explained in the previous

subsection, in a case where both connections are unavailable, the Q-Box will cache every packet and deliver them as soon as the connection is reestablished. This behavior is assessed to be the most appropriate for the concerned architecture.

Owing to the sensitivity of the topic, it could not be clarified whether the LAN-internal communication of Coqon is encrypted by reasonable algorithms and key bit lengths. The same applies to whether and how replay attacks are prevented. Accordingly, these issues must be verified by additional dynamic and static program analyses of the Coqon controller firmware and app.

Löhmann mentioned the known problems of SSL for LAN communication (cf. Chapter 2.3.2) and therefore proposed message payload encryption to be the most efficient solution. However, a partially encrypted network request or packet may still reveal information like headers or other sensitive meta data. Manually encrypting message payloads leads to a common problem for information security: The concept of *Build your own Security*, rarely leading to safe systems. Implementing own security algorithms or security-relevant protocols is prone to implementation errors and requires a lot of security-theoretic knowledge. As a result, he generally discouraged this concept and recommended to adhere to well-known libraries, for instance for adapted TLS implementations.

The message authentication process revealed potential security hazards. The secure UMTS channel, as described before, is used to generate communication tokens. These tokens are then used to authenticate and establish communication between app and controller. Tokens are reused and not regenerated for every message, due to the otherwise high latency. Theoretically, even though tokens have a limited lifespan and are refreshed after a certain, yet undisclosed period of time, they might be extracted and reused by attackers. Dynamic software analysis has to verify if this attack vector is feasible in real-world scenarios or prevented by additional protection mechanisms like packet encryption. Generally, the user is advised to employ the system only in association with effective network encryption, i.e. WPA2.

External cloud connections are secured using TLS and certificates, whereas the latter are signed by an official and well-known CA. Details about the certificates or TLS version were not disclosed and are to be analyzed separately. For TLS, Coqon relies on Java-internal and Microsoft libraries. The specific libraries in use were also left undisclosed. Löhmann did not know whether the system actively requests the CRL.

Similar to Bosch Smart Home, the system contacts a (proprietary) time server for NTP clock

synchronization. This information is transferred through the encrypted UMTS tunnel. The controller requests the time using a built-in Linux NTP command, where two alternative servers are provided in addition to Coqon's proprietary time server. However, in most scenarios, only one time server is contacted for synchronization, contrary to the Bosch system (cf. Chapter 3.2.1). Once the server is compromised, controllers could work on fraudulent time information, representing a potential security risk (cf. the recommendations on page 83).

In an effort to summarize the most interesting security problems that occurred during Coqon's development, Löhmann mentioned the unidirectionality of the UMTS connection. Contrary to hosts connected to the internet through regular network routers, devices attached directly to a cellular network are generally not protected by NAT. This allows for direct connections to the UMTS-enabled Coqon controllers. As a result, developers implemented a one-way behavior of the cellular link. Coqon controllers generally reject incoming WAN connections, except for responses to previous requests. If an official cloud server is compromised, the adversary will be unable to arbitrarily send commands to every Q-Box.

Controllers may upload diagnostic data to Coqon's cloud servers, to help technical-support employees in identifying and fixing system malfunctions. The upload is performed using the encrypted variant of FTP, namely Secure File Transfer Protocol (SFTP). Whenever a controller uploads diagnostic data, the file's access rights are manipulated in an effort to prevent access, modification and deletion of the same user. Hence, once an attacker retrieves the SFTP credentials integrated into the controller's firmware, he may be able to upload data to the server, but he will be unable to retrieve or modify the actual asset of adversary interest, i.e. the stored sensitive information. The modification of access rights is considered an important step in securing the system.

To achieve a basic level of security, the system implements best-practices included in the BSI *IT-Grundschutz-Kataloge* (cf. [Sic]). For example, the system does not display which of the two credentials, user name or password, is incorrect when a login attempt fails. To implement the well-known *Separation of Privilege* and *Least Privilege* (cf. [SS00]) security principles, several user roles with minimal access rights are implemented, precisely *Support*, *Craftsman*, *Owner*, *User*. Each role has a number of unique and shared access rights associated. For example, only Support users can access diagnostic information and only Craftsmen can pair more elaborate device types like flush-mounted buttons.

The app relies on the trustworthy integrated KeyStore to save sensitive information for Android OS versions of 4.3 and above. As the app requires Android 4.0.3 as a minimum, the information is saved differently on legacy versions. The particular method for saving sensitive information on legacy versions was not disclosed and requires static app analysis.

Whenever login credentials are entered incorrectly five times, the corresponding account is locked by the server for a certain amount of time, rejecting further login trials. Besides, the corresponding user is informed via e-mail. This technique impedes brute-force attacks. Cloud servers rely on firewalls to cope with DDoS attacks; however, it was noticed that the system may not be able to handle unexpectedly large DDoS attacks. A large attack may render every Coqon smart home system useless for a certain amount of time, due to the dependence on the Coqon cloud.

For neusta, security is recognized as being an important part of the development process. Yet, the questionnaire revealed a large remaining potential for optimization. Regular penetration tests, focused security reviews and other means were considered useful during the interview. The same applies to third-party libraries; even though Coqon comprises various external frameworks, it is not assured that up-to-date versions and security fixes are integrated regularly. Version checks are done sporadically, in a manual, individual and uncontrolled manner. The development team did not create a certain attacker model or perform hazard analysis beforehand.

As firmware updates are able to substitute the complete operating system of the controller, the risk of security vulnerabilities not being fixable in the future is considered to be low. As a conclusion from the interview's insights, Coqon provides many recommendable approaches for security. Still, the development process offers various issues that can be easily optimized, to increase the overall level of security.

3.4 Conclusion

Referring to the architectures and systems analyzed in this chapter, it becomes obvious that every variant offers a mixture of positive and negative aspects. Coqon's UMTS channel, for example, is an effective method to provide a backup connection in case the regular broadband link is not available. However, a strong cloud dependency implies various disadvantages that were explained in the preceding chapters. Bosch Smart Home, on the other hand, made a positive impression

because of a rather cloud-independent architecture. Still, internet access is mandatory for this system. Lastly, Qivicon enables LAN-local home automation without internet access, once the system is configured correctly. Generally, an even more independent solution without the need for the World Wide Web, e.g. for high-security environments, is desirable. As for security in general, Coqon's behavior of providing an alternative secure VPN channel is considered to be the best architectural technique against Man-In-The-Middle and other network-based attacks. As multiple security-relevant questions remained open during the according interview, an in-depth security analysis will have to verify whether Coqon's security is indeed on the expected level across the whole system.

Chapter 4 will revisit some of the most important security vulnerabilities of this section in addition to other issues, while Chapter 5 eventually presents best-practice solutions to many of the encountered problems.

Chapter 4

Threat Model

A smart home system offers various connection interfaces. Many devices from different manufacturers are incorporated and operated simultaneously in every HAN. As a result, there are multiple potentially vulnerable areas to protect and a common ”level of security” across all devices cannot be universally assumed, as it is not evident whether all components implement each protocol correctly. Furthermore, some systems may adhere to outdated standards proven to be insecure, in order to support a large variety of third-party DUCs and to therewith foster a higher market share.

Figure 4.1 emphasizes the most important areas of potential weaknesses attackers could exploit to compromise the smart home. The areas are predominantly derived from the app analyses of Chapter 3, therefore they may vary for other smart home systems, for example for architectures where external remote controls are able to communicate directly with the home network via the domestic broadband router (cf. Chapter 2.5). Referring to Figure 4.1 when appropriate, this section will give an in-depth analysis of plausible attack scenarios, in addition to a characterization of the typical attacker.

Owing to of the complexity and the variety of potential smart home architectures, the following list of scenarios and attackers obtained by applying the methodology introduced in Chapter 2.4 does not claim to be exhaustive. Moreover, security researcher Ross J. Anderson describes in one of his publications that a threat analysis performed by a single investigator will rarely cover every issue and *even the best will miss things* [And08, p. 396]. Generally, he derived from an experiment that employing multiple experts to perform threat analysis at the same time is regarded to be more successful and will reveal most issues, due to the different perspectives and approaches each person will consider appropriate:

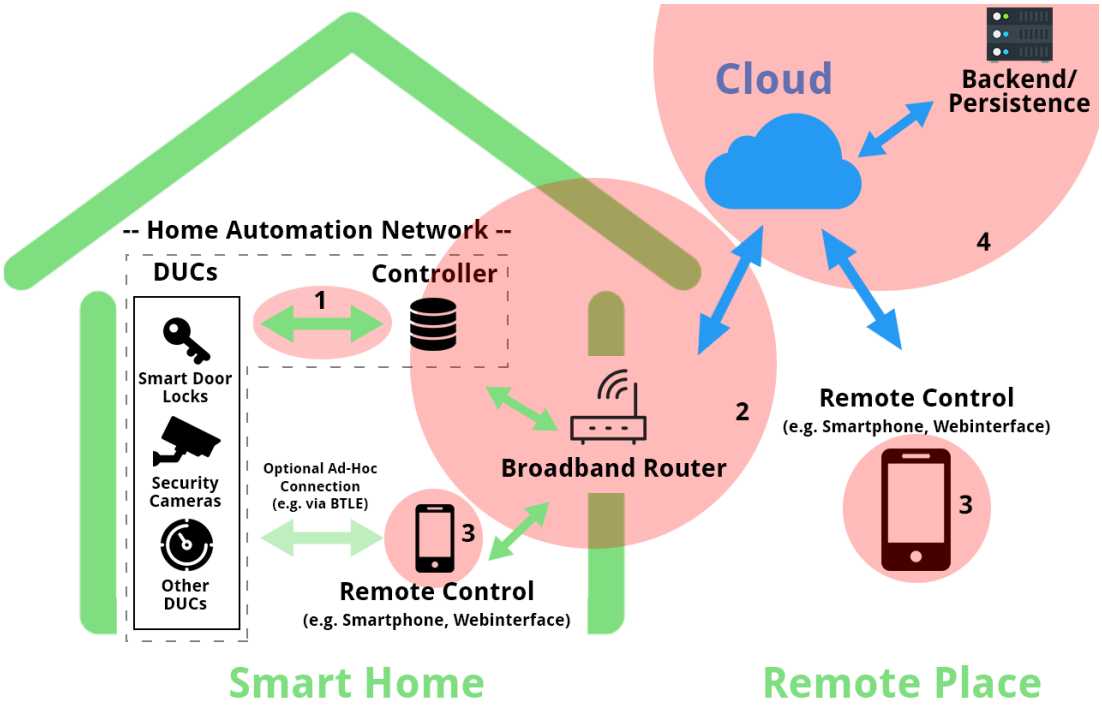

Figure 4.1 Areas in smart home systems especially prone to potential attacks have been highlighted with red circles and are numbered for reference. Each area's boundaries may blur and vary, depending on definitions and architectures. The graphic is based on Figure 2.3.

> The lesson to be learned from this case study is that requirements engineering, like software testing, is susceptible to a useful degree of parallelization. So if your target system is something novel, then instead of paying a single consultant to think about it for twenty days, consider getting fifteen people with diverse backgrounds to think about it for a day each. [And08, 845 f.]

As a result, the following outcomes should function as an elaborate basis and a checklist for potential security flaws when developing smart home software. This chapter will occasionally refer to further material when an issue exceeds the boundaries of this thesis.

4.1 Threat Classification

In an effort to reasonably sort the findings of the threat analysis in relation to the criticality and likelihood of each item, the two metrics introduced in Tables 4.1 and 4.2 will be employed. This two-category hazard assessment system, separated into *Severity* and *Probability*, is based on the *Risk Mishap Index* for military systems engineering[1]. The standard originally ranks hazards with regard to monetary loss and injury. Accordingly, the categories in Tables 4.1 and 4.2 have been adjusted to fit the context of this thesis.

It is noted and highlighted that the individual assessments are based on subjective estimations, due to the lack of quantitative statistics about attack type probabilities and the absence of a reasonable and standardized metric for the severity of particular attacks. However, the values should not be treated as being useless or fully arbitrary. Besides the Risk Mishap Index, similar other approaches to prioritize threat analysis outcomes are commonly used, for example under the name of *Annual Loss Expectancy (ALE)*, standardized by the National Institute of Standards and Technology (NIST). Ross J. Anderson also stresses a caveat when the prioritization of hazards is concerned:

> But in real life, the process of producing such a table is all too often just iterative guesswork. (...) The point is, ALEs may be of some value, but they should not be elevated into a religion. [And08, 846 f.]

Following the individual nature of any software system and the inconsistent susceptibility to attacks, the categorization should be regarded as a general approximation that may deviate for a particular real-world entity. As described for general threat analyses in the beginning of Chapter 4, organizing a team of multiple security experts for the assessment of each hazard could lead to more precise results. However, in the course of the *SecureSmartHomeApp* project, this approach was not feasible due to limited resources.

As each attack is eventually ranked by two indicators, Table 4.3 can be used to identify the combined risk level, mapping a specific attack to a fixed value. The latter may also help to rank scheduled security-relevant implementations in terms of importance.

[1] As described in `MIL-STD-882`. Refer to [Def00, p. 11] for the corresponding section of the standard.

Severity	Level	Explanation
High	1	Vulnerabilities provide the attacker with elevated access rights and may allow him to take complete control over the system or network.
Medium	2	An attack may reveal sensitive data and the attacker might be able to execute predefined and static commands, for example via replay attacks.
Negligible	3	This attack may reveal unimportant, encrypted or insensitive data. The attacker will not gain privileged access to the system.

Table 4.1 A simplified, three-level severity model, based on [Def00, p. 11]. It will be used in the following subsections to assess attack impacts.

Probability	Level	Explanation
Likely	A	Amateur attackers may test and perform these approaches manually, or even automatically by using penetration frameworks or very simple processes. Special resources and extensive training are not required.
Possible	B	Skilled attackers may aim to exploit this vulnerability.
Remote	C	It is unlikely for most adversaries to exploit a given vulnerability, for instance due to extensive resource requirements or obligatory physical access to a secure storage.
Eliminated	D	Can only be obtained if an attack is prevented by contrary measures. For issues of this category, the *Severity* value will be omitted, as the attack is not further feasible with reasonable resources.

Table 4.2 Each attack vector will be rated according to a four-level probability model, based on [Def00, p. 11].

Probability \ Severity	High (1)	Medium (2)	Negligible (3)
Likely (A)	High	High	Medium
Possible (B)	High	Serious	Medium
Remote (C)	Serious	Medium	Low
Eliminated (D)	Eliminated		

Table 4.3 Based on [Def00, p. 12], this table is used to derive the risk level, a value defined by *Severity* and *Probability*.

Usually, the classification meters the severity and probability of *attacks*. However, in the context of *assets* (cf. Chapter 4.2), the classification is used in an adapted way and can be interpreted as follows:

- **Severity**
  Table 4.1 originally referred to *Severity* as being a measurement for the privileges an adversary gains by the performance of a specific *attack*. In this context, however, *Severity* assesses the privileges gained by compromising or taking control over a specific *asset*.
- **Probability**
  This value measures the likelihood of an attacker trying to get access to or to compromise a specific asset, in contrast to the likelihood of the performance of a specific attack.
- **Risk**
  With increasing risk, the importance of implementing reliable security measures for a specific asset increases as well.

4.2 Attacker and Attack Characterization

Multiple assets of adversarial interest were identified during the threat modeling process. They were individually assessed and set in relation to each other, adhering to the classification of Chapter 4.1.

1. **User credentials**
   Presuming the user identification process does not employ two-factor authentication, the complete (domestic) system is compromised once the plain user credentials leak. The adversary will then have full control over the system and he may also exclude the regular user by changing the password. Credentials can be revealed either by social engineering or by passive and active network attacks, for example.

   Severity (cf. Table 4.1): Level 1
   The (domestic) system is completely breached upon credential leakage, as described above. This incursion does not affect the manufacturer's cloud servers.
   Probability (cf. Table 4.2): Level A
   User credentials represent one of the most appealing target to attackers and even amateurs

may try to intercept them by recording network traffic or by starting a brute-force attack.

Risk (derived accordingly from Table 4.3) ⇒ High

2. **Controller**

   The smart home controller represents the system's essential node for interaction. Once compromised, the attacker will be able to take extensive control over the HAN, possibly rendering the security features of other areas (cf. Figure 4.1) useless. Potential vulnerabilities, among others, involve insecure web or app interfaces and deficient command authentication.

   Severity: Level 1

   The adversary gains full access to the domestic system once this asset is compromised.

   Probability: Level A

   A smart home controller offers many connection interfaces and commonly relies on various, potentially outdated frameworks, leading to vulnerabilities that even amateurs may exploit.

   Risk ⇒ High

3. **Persistence and cloud servers**

   Being able to fetch arbitrary rows from the backend's SQL database may provide the attacker with sensitive user data. For some smart home architectures, having general control over cloud servers can yield full access to every active instance of the system.

   Severity: Level 1

   The severity level depends on the data the attacker gains access to. If sensitive user data like passwords or credit card information are concerned, the severity is considered to be *High*. Adversaries may use this information to gain access to the system in question, in addition to separate systems of other manufacturers.

   Probability: Level A

   Depending on the database's and server's interfaces, tools like Burp (cf. Chapter 2.6) provide amateur attackers with means to automatically test an API for a variety of vulnerabilities.

   Risk ⇒ High

4. **DUCs**

   Obtaining control over the DUCs in a smart home system can render the need for user credentials unnecessary. As described in the following chapters, depending on the attacker's main intentions, it may be sufficient to control a specific actor (e.g. a door lock) or to install

a custom firmware on a DUC. There are multiple potential ways to compromise DUCs, for example by exploiting security weaknesses in the ZigBee protocol (cf. Chapter 4.2.1).

Severity: Level 1

This value depends on the particular DUC types. If door locks are concerned, burglars may enter the smart home without notice. Thus, for a worst-case scenario, the severity level is considered to be *High*.

Probability: Level B

In order to exploit DUC and wireless protocol vulnerabilities, special transmitting devices are required. Therefore, a relevant adversary is presumed to act on a higher level of proficiency.

Risk ⇒ High

5. **Home network**

   Generally, gaining access to an encrypted wireless network is difficult to accomplish, especially when the home network is encrypted using a strong encryption algorithm like WPA2. By compromising an associated HAN, the attacker often implicitly gains control over devices eventually connected to the home LAN. Hence, he could use these devices to perform network-internal attacks like traffic interception. Additionally, he could exploit vulnerabilities of computers connected to the same LAN and eventually take control over these hosts.

   Severity: Level 1 - Level 2

   Once the attacker has surpassed the home network's encryption, he may intercept sensitive user credentials. Depending on whether the attacker is able to perform passive or active network attacks, the severity level is considered to be *High* to *Medium*.

   Probability: Level C

   Owing to the large number of different steps involved, this scenario is considered to be rather complicated for real-world situations. It does not represent the most intuitive technique for intruding into (W)LANs.

   Risk ⇒ Serious - Medium

The severity of compromising each asset and the corresponding attack probability greatly depend on the specific attack methods and the extent of the revealed data. Consequently, Chapters 4.2.1 and following analyze specific attack variants in detail.

As described before, one of the most appealing targets to certain attackers is being able to

control DUCs. This especially applies to door locks or security surveillance cameras to facilitate physical, criminal acts like burglary. Ideally, this would happen without the system noticing a state change, as described in Chapter 4.2.1.

Furthermore, depending on the attacker's intentions, IoT devices could be used for cyber-criminal offenses, for example DDoS attacks. By bundling insecure DUC units, like digitalized toasters and coffee machines, into a large botnet, this scenario was recently encountered in reality. In 2016, the botnet *Mirai*[2] appeared, negatively influencing the availability of various web services. In a different case of May 2017, just after the registration of a particular domain name decreased the amount of *WannaCry* ransomware infections, insecure IoT devices were collectively used in an attempt to hinder the registered domain's availability, to reinitiate *WannaCry* [Gie17].

As recent news articles consistently reveal new attacks almost on a daily basis, and as they are usually performed by independent attackers and so-called *hacker collectives*, attacker category two (Hackers/Crackers, cf. Chapter 2.4) is regarded to be the most relevant at the time of writing. This attacker type usually only possesses information publicly available on the internet. However, by employing common techniques for system analysis like port scanning, the adversary may be able to redeem additional information useful for more advanced attacks. The *Insider* attacker category is assumed to have acquired additional, unpublished information about security vulnerabilities of protocols or software frameworks. However, based on our research, this type is not assessed to be the most relevant.

By observing recent news articles, assets one, two and four (user credentials, controller and DUCs, respectively) of the above list are considered to be the most critical. The typical attacker defined above predominantly aims to exploit the (wireless) network communication, as it is the only component available to him without physical access to any of the affected smart home devices. Plus, many different network attacks are described on the World Wide Web and easy to perform, especially for infrastructures using WiFi.

The following sections determine threats for the system depicted in Figure 4.1, considering the reasonable computing resources typically available to the attackers characterized above. The attacks are sorted in the descending order of risk, probability and finally severity. The according ratings are supplied with causal remarks where deemed necessary. Eventually, Chapter 5 derives protection mechanisms and security recommendations for each issue.

[2] *https://www.golem.de/specials/mirai/*

4.2.1 Area 1: DUC Connections

It has already been demonstrated in literature that wireless IoT protocols like ZigBee and HomeMatic contain intriguing security flaws. For example, Daniel AJ Sokolov explains how attackers can easily intercept the key used for encrypting ZigBee traffic when new devices are paired. Subsequently, the key can be used to open door locks and control other devices without the user noticing, as the system will continue to display a door lock as *closed*. Alternatively, attackers may spoof sensor data and output values that do not correspond to a real situation. [Sok15]

A major disadvantage of the alternative HomeMatic, on the contrary, is that AES-capable devices ship with a predefined encryption key identical for every DUC.[3] Additionally, message authentication is optional and usually disabled for many devices (excluding smart door and window locks). [Ren14]; [Plö15]

Plus, it is always possible to temporarily interrupt wireless connections using jamming devices. In case of emergency, when interfering transmitters are employed, an intrusion detector's signal may not be transferred to the controller.

As a result, the security evaluation for this area is simplified and regarded to be identical with the previous categorization referring to the DUC asset (cf. Chapter 4.2).

Severity: Level 1
Probability: Level B
Risk ⇒ High. For security recommendations refer to page 73.

4.2.2 Area 2: LAN and WAN Connections

The smart home controller, being the device that functions as an information hub receiving, processing and forwarding commands to DUCs, needs to be regarded as one of the most important assets of the system (cf. Chapter 4.2).

In order to further explain the different ways of attacking Area 2, Figure 4.2 depicts a typical communication scenario in detail. It is shown that a controller is basically connected to four

[3]The key is available on the internet, precisely at *https://pastebin.com/eiDnuS8N* (retrieved on 06.06.2017). As a result, the constructed attacker, being restricted to information publicly available, is able to exploit this vulnerability.

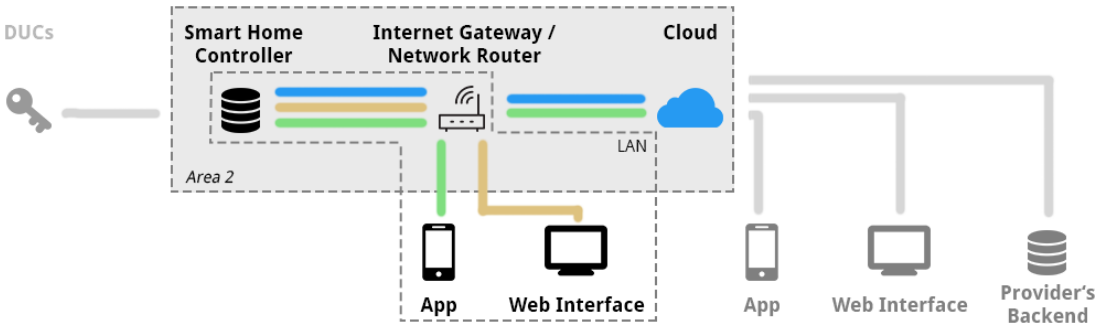

Figure 4.2 A general overview of the HAN's communication in Area 2, cf. Figure 4.1. Icon source: [FMM]

different channels of communication. First, there is the connection to DUCs, which references Area 1 (Chapter 4.2.1). Second, three different lines symbolize in- and outgoing connections from and to the app (green), a potential additional LAN web server provided by the controller (brown), and finally the cloud (blue), each representing connections where sensitive data may be transferred. As demonstrated in Chapter 3, many remote control apps rely on a cloud connection. As a result, in Figure 4.2, the app's green connection line is drawn between controller and router, as well as between router and cloud.

Based on this communication diagram, a malicious party could now perform the following passive and active attacks.

Passive Attacks

1. **Traffic interception**

   The most typical and probably one of the most likely attacks is traffic interception, especially relevant when the adversary is located in the same LAN. In this scenario, an attacker tries to use a packet sniffing program like Wireshark to record data packets sent on any of the depicted connections from Figure 4.2. He may then analyze the traffic in order to examine and understand the employed protocols, specify message formats and decipher the information sent. Depending on the particular protocols, sensitive information like user credentials may be disclosed. The attacker will probably use this method as a basis for further attacks.

   Severity: Level 2

   Traffic interception may reveal sensitive data. However, an attacker has to wait for specific packets and timing is important. Furthermore, a *passive* attack itself does not always

immediately yield elevated access rights; if the attack provided the malicious party with login credentials, for example, other mechanisms like second-factor authentication may still hinder it from compromising the system. Hence, this attack's severity is set to Level 2.

Probability: Level A

Risk ⇒ High. For security recommendations refer to page 76.

2. **Port scans and reconnaissance tools**

   A device of this area may be offering open ports for different services of arbitrary protocols. An automated port scan performed by a tool like `nmap` can yield a list of reachable ports which, in turn, may provide access to potentially vulnerable software. A reconnaissance tool is generally employed to identify the used operating system and the installed software versions.

   Severity: Level 3

   A port scan does only reveal initial information about the installed software versions and the system architecture and does not provide the attacker with elevated access rights per se, hence the severity is set to Level 3.

   Probability: Level A

   Risk ⇒ Medium. For security recommendations refer to page 77.

Active Attacks

1. **Packet spoofing**

   As soon as the attacker is able to understand the data format, he could create own datagrams destined for a specific communication partner. One of the most interesting destinations is the controller. When command authentication is badly implemented, being able to affect the behavior of the smart home base may eventually provide the attacker with extensive control over the complete HAN. Chapter 3.2 demonstrated how an adversary could communicate with the home base from another network, even though remote access was disabled. Therefore, spoofed packets may also be received from a different network.

   Severity: Level 1

   Probability: Level A

   Risk ⇒ High. For security recommendations refer to page 77.

2. **Replay attack**

   The system needs to take precautionary measures for replay attacks, where the attacker resends recorded packets to evoke expected or unexpected behavior. From an attacker's perspective, the system will ideally redo the same actions that were performed when the system initially received the packet. Generally, by using replay attacks, the adversary is limited to a specific set of actions. He has to actively capture a packet that disarms the alarm system, for example.

   Severity: Level 2

   Probability: Level A

   Risk ⇒ **High**. For security recommendations refer to page 78.

3. **Man-In-The-Middle attack**

   LAN-internally, the attacker may use techniques like ARP poisoning to impersonate as another communication party. For instance, against remote control devices (app or web interface), the attacker could identify himself as the controller. At the same time, for the controller, he could impersonate as a remote control unit. Thus, by intercepting, manipulating or deleting datagrams, he may alter or even prevent the user's commands. The latter is especially critical in the context of alarm systems, for example. The Man-In-The-Middle could reply with a datagram stating that a burglar alarm was successfully armed, even if the corresponding instruction packet was purposefully discarded. Many other possibilities render this attack one of the most dangerous scenarios for Area 2.

   Severity: Level 1

   Probability: Level B

   Risk ⇒ **High**. For security recommendations refer to page 78.

In order to perform any of the above in Area 2 on the basis of the exemplary architecture of Figure 2.3 and its restrictions, the malicious party needs direct access to the victim's network or the HAN. This can be achieved in many ways, for instance by computer viruses, social engineering techniques, cracking insecure WEP encryption keys and gets even easier if the AP is not encrypted. This is the case for many public hotspots. For instance, a plausible scenario could be a user connected to a hotspot in a restaurant, using his phone to connect to the smart home cloud, in order to control home appliances. An attacker could then intercept network traffic and perform some of the attacks stated above.

To facilitate attack preparation on this layer without the risk of mistakenly alerting the house owner, an attacker would need access to his own, separate HAN. This may be a cost factor discouraging less proficient attackers.

4.2.3 Area 3: Remote Controls

Remote controls, especially apps, are an appealing target to attackers, because vulnerabilities can be tested and examined in an emulation system. Static software analysis often eliminates the need for a real HAN for testing. Potential security flaws can be found and analyzed without any time pressure. By using static analysis, it is only secondary to intercept specific network packets. When used in combination with dynamic program analysis, the system's reaction to an attack can often be directly observed.

Attackers in possession of the app could try a broad range of potential attacks. Owing to the relation to the *SecureSmartHomeApp* project, this section will predominantly focus on Android issues.

1. **Exploiting outdated third-party libraries**
   Many apps ship with outdated libraries, as shown in Chapter 3.1. Badly maintained software is a critical hazard, as security exploits are often freely available on the internet. As a result, this information is also available to the attacker constructed in Chapter 4.2. Potentially dangerous libraries include JavaScript frameworks like jQuery, in case the app relies on WebViews to display information or user input.

   Severity: Level 1
   This value does greatly depend on the type of library and the severeness of the vulnerability. However, outdated third-party programs like *Adobe Flash* provide attackers with the ability to execute arbitrary code. Accordingly, in an assumption of a worst-case scenario, the severity is set to be *High*.
   Probability: Level A
   Attackers can easily and quickly identify apps that use outdated libraries, for example by decompiling APKs.
   Risk ⇒ High. For security recommendations refer to page 79.

2. **SQL injection**

   When user input is not verified correctly, SQL injection could enable attackers to modify commands sent by the remote control. Depending on the architecture, the adversary may change the command's target smart home controller ID or manipulate the provider's database.

   Severity: Level 1

   Probability: Level B

   To exploit an SQL injection vulnerability in-app, elaborate app analysis and network traffic surveillance is necessary. This requires advanced knowledge and is probably not performed by amateur attackers.

   Risk ⇒ High. For security recommendations refer to page 80.

3. **User authentication**

   If the app does not verify the user's identity, an attacker in possession of the device may manipulate settings, passwords or perform other malicious actions. He is able to obtain full control over the system.

   Severity: Level 1

   Probability: Level B

   This vulnerability is only relevant for attackers of opportunity and adversaries that specifically target a certain household. Getting physical device access is not feasible for every scenario, hence the probability is set to be *Possible*.

   Risk ⇒ High. For security recommendations refer to page 81.

4. **Exploit dynamically included resources**

   Whenever apps dynamically load resources from external servers, extract URL parameters or include files from a memory card, attackers could be able to replace legitimate data by malicious code. Once attackers generate phishing input forms and trick a WebView into loading the malicious resource (cf. Chapter 3.1, p. 32), they can intercept sensitive user data. Alternatively, in a case where a web form is providing a selection of actions like *Unlock Door* or *Open Windows*, a dynamically loaded malicious user interface could confirm a locked door by displaying *Command executed*, even though the web interface does not transfer the command to the controller.

   Severity: Level 2

This level depends on the type of resource that is loaded dynamically. It is assumed that the extensive control interfaces described above are rarely included dynamically. Still, as it was highlighted in Chapter 3.1, this scenario could be plausible for a user registration form.

Probability: Level B

See above: It is assumed that important control interfaces are rarely included dynamically.

Risk ⇒ Serious. For security recommendations refer to page 82.

5. **GUI attacks**

   Security vulnerabilities found in the Android system allow for an attack named *Cloak & Dagger* [Tho] where invisible overlays can be used to intercept inputs made on the virtual keyboard across different apps. This allows for an interception of user credentials.

   Severity: Level 1

   An attacker may use the intercepted credentials to log into the smart home base and issue arbitrary commands.

   Probability: Level C

   Google stated that apps including the described behaviour are excluded from the app store and that the vulnerabilities will be fixed with Android version O and above [Tho].

   Risk ⇒ Serious. For security recommendations refer to page 82.

6. **Digital certificates**

   As shown in Chapter 3.2, the system should not rely on the internal clock when certificate validity timestamps are checked. Otherwise, outdated and broken certificates might be erroneously held valid when an attacker manipulates the device's time settings. The same problem occurs when the app does not check the CRL (cf. Chapter 3.2, p. 43).

   Severity: Level 1

   An attacker may use broken certificates to maliciously sign self-generated commands or perform Man-In-The-Middle attacks.

   Probability: Level C

   Digital certificates of large companies are rarely revoked due to high security policies and strong signature algorithms. Moreover, other attack vectors are assumed to be more likely.

   Risk ⇒ Serious. For security recommendations refer to page 83.

7. **Insecure KeyStore**

   Rooted devices or vulnerable proprietary KeyStore implementations may reveal sensitive

data like certificates, passwords or keys. Owing to the simple decompilation process of software, hard-coded storage of token values, passwords and other sensitive information is an additional important security risk.

Severity: Level 1

Once the KeyStore is compromised, the attacker will have access to every sensitive file of the app. He is then able to generate and sign own commands. Furthermore, program code can easily be decompiled, yielding hard-coded keys in cleartext form.

Probability: Level C

Both, Bosch Smart Home (Chapter 3.2) and Coqon (Chapter 3.3) rely on the Android KeyStore to save credentials and certificates. However, Qivicon (Chapter 3.1) proved that some apps use deprecated, vulnerable encryption algorithms and persist data using extractable, static keys. Still, when compared to the other attacks, it is assumed to be comparatively unlikely for the constructed attacker to mainly focus on a single, proprietary credential storage implementation concerned with transient OAuth keys. Furthermore, relatively few Android users own a rooted device, which would otherwise greatly increase the risk of using the integrated KeyStore.

Risk ⇒ Serious. For security recommendations refer to page 84.

8. **Insecure broadcasts or services**

   As shown in the bachelor's report [Mül15], a critical security aspect of Android apps is the insecure usage of broadcasts and exported services. Even though no analysis of Chapter 3 referred to security issues concerned with this topic, whenever broadcasts or services are used, sensitive data may leak and attackers could instruct the remote control app to perform malicious actions.

   Severity: Level 3

   Preceding analyses showed that it is possible for broadcasts to contain sensitive user data, though the probability is generally assumed to be relatively low [Mül15]. Android services exploitable for arbitrary command execution are considered to be a comparatively rare implementation, too. However, attackers (and burglars) may intercept broadcasts about actions like unlocked doors or changed brightness of smart light bulbs and draw certain conclusions.

   Probability: Level B

   Risk ⇒ Medium. For security recommendations refer to page 84.

Besides, for general app development, there are resources summarizing the most common vulnerabilities for the Android system, e.g. lists created by the Open Web Application Security Project (OWASP) [OWAd], along with recommendations for prevention.

4.2.4 Area 4: Cloud and Provider Servers

As described previously, every tested smart home system of Chapter 3 incorporates functionality relying on a communication with manufacturer's servers, commonly referred to as the *cloud*. Potential cloud-related attacks include, but are not limited to:

1. **Brute forcing**

   If the system does not limit the amount of login queries a single client can perform, it is likely that a malicious party will setup brute-force software to automatically try different username and password combinations.

   Severity: Level 1
   Probability: Level A
   Risk ⇒ **High**. For security recommendations refer to page 85.

2. **SQL injection**

   Parts of the manufacturer's backend probably communicate with a database. If the attacker is able to influence or completely define the corresponding query parameters, he may be able to shape particular values. This provokes unwanted behavior, like the potential return of a list of registered users or the deletion of specific database entries. This vulnerability is especially likely when URL parameters are parsed by a PHP script and not filtered for special characters, e.g. inverted commas.

   Severity: Level 1
   Probability: Level A
   Risk ⇒ **High**. For security recommendations refer to page 86.

3. **DNS spoofing**

   If an attacker is able to introduce false data into domain name servers, smart home system requests may be redirected to malicious servers.

Severity: Level 1

Probability: Level B

Risk ⇒ High. For security recommendations refer to page 86.

4. **Identity forgery**

   An attacker could try to connect to the manufacturer's interface assuming the identity of a particular remote control and invoke arbitrary functionality on a corresponding smart home controller.

   Severity: Level 1

   Probability: Level B

   The probability assessment depends on the complexity of the authentication process. If no (effective) authentication is performed, Level A may be assumed.

   Risk ⇒ High. For security recommendations refer to page 86.

5. **DDoS**

   Using a botnet, a large amount of clients could maliciously connect to the provider's servers at the same time, provoking the system to collapse. Generally speaking, this is an attack scenario potentially concerning many system components, for example to interfere with the availability of alarm systems. To prevent duplicity, this attack vector is not mentioned for the other areas.

   Severity: Level 2

   Probability: Level B

   Risk ⇒ Serious. For security recommendations refer to page 87.

6. **Port scans and reconnaissance tools**

   Cf. Area 2: *Port scans and reconnaissance tools*, page 63.

   Severity: Level 3

   Probability: Level A

   Risk ⇒ Medium. For security recommendations refer to page 85.

Probability \ Severity	High [13]	Medium [4]	Negligible [3]
Likely [8]	High [4]	High [2]	Medium [2]
Possible [9]	High [6]	Serious [2]	Medium [1]
Remote [3]	Serious [3]	Medium [0]	Low [0]
Eliminated [0]	Eliminated [0]		

Table 4.4 An overview of the number of attack vectors identified in each category, based on Table 4.3. The more general risk assessment for the assets defined in Chapter 4.2 is not included.

4.2.5 Conclusion

To summarize the factors listed above, Table 4.4 provides a comparison of the number of attack vectors for each risk level. It becomes obvious that a large amount of issues falls into the *High* category, leaving few issues in the lower-right half of the table. On the one hand, this highlights the immediate need for security in the smart home sector. On the other hand, many unlikely and negligible issues were purposefully excluded from the analyses for the sake of clarity. An exemplary scenario deemed too unlikely, specific and complex for a general and reasonable threat analysis in the context of this thesis would be a case where a user buys a second-hand smart home controller, purposefully manipulated with a malicious firmware, intended for the particular user.

After the best-practice fixing proposals were given in Chapter 5, Table 5.2 can be used for comparison, to highlight the improvement that can be obtained by applying certain safeguarding techniques.

Chapter 5

Reference Guide

This section provides a reference for developers. The insights can be used when constructing new or revising existing smart home software systems. Multiple security mechanisms are evaluated and the most appropriate strategies are highlighted.

Generally, a reference guide can be organized in multiple ways. For example, the guide could rely solely on class diagrams and other Unified Modeling Language (UML) graphics. However, in an effort to keep every recommendation of this chapter as comprehensible and coherent as possible, the list structure introduced in the previous chapter is maintained, resulting in best-practices for four different security areas. The hazards identified in Chapter 4 are used as a basis therefor. Subsection 5.5 lists additional actions that can easily be integrated into any software development process, to foster the creation of secure software.

Chapter 5.6 combines the results and presents them in the form of an optimized version of the general smart home architecture originally depicted in Figure 2.3.

5.1 Area 1: DUC Connections

In the course of the smart home software analysis (Chapter 3), the wireless communication protocols ZigBee, Z-Wave, HomeMatic, HomeMatic IP, DECT ULE and other, less popular variants were encountered. Subsequently, Chapter 4.2.1 proved various protocols insecure, often due to irreparable design flaws.

Following these insights, it is obvious that ZigBee should not be used. However, as described previously, the evolving smart home market requires manufacturers to develop systems compatible to many different DUCs. Therefore, an abstinence of the aforementioned protocol may not be economically acceptable. In this case, it is recommended to actively inform users about potential security risks when ZigBee devices are paired, especially when security-relevant DUCs like door locks are involved. Following the way Google Chrome notifies users about insecure websites (cf. Chapter 2.3.2), smart home software should explain risks in a comprehensible manner. A corresponding checkbox element could provide an active opt-in mechanism, requiring users to confirm the comprehension of the issue. Newly developed DUCs should always rely on a different communication protocol.

When HomeMatic is concerned, the system should always enforce the integrated message authentication features. Additionally, HomeMatic-capable devices need to actively require a reasonable change of the default encryption key, which, in turn, should consist of at least 32 random hexadecimal characters[1] [Plö15].

For HomeMatic IP, no security vulnerabilities could be found on the internet and in literature. Instead, the IT security entities *AV-Test* and *ESCRYPT* awarded HomeMatic IP to be *Secure* in 2016 [eQ316]. Therefore, assuming the constructed attacker to not own any insider knowledge about vulnerabilities, the use of the protocol is encouraged.

The low energy transmission protocol DECT ULE employs a 128 bit AES encryption scheme. As for HomeMatic IP, no security vulnerabilities could be found on the internet and in literature, therefore, in the context of the attacker model, the protocol is assumed to be secure at the time of writing.

To detect interference from jamming devices as described in Chapter 4.2.1, a protocol scheduling regular beacon packets to ensure the presence of every DUC could be considered useful; however, owing to the high energy and processing power requirements, this is not feasible in most IoT scenarios. On the contrary, the controller could use own interference detection techniques, which, in turn, may lead to many false alarms due to the large number of wireless devices in operation today[2]. As a result, ideally, security relevant DUCs should be connected to the smart home

[1] This value may be lower in case special characters are supported.
[2] Many examples of users complaining about false jamming alarms are available on the internet, for example: *https://suretydiy.com/forums/topic/securityrf-receiver-jammed/* or *http://forums.xfinity.com/t5/Devices-and-Equipment/Getting-the-message-quot-Zigbee-Transceiver-Jammed-quot/td-p/2836934*, retrieved on 06.06.2017.

Technology	Usage Recommendation
Wire (e.g. Ethernet)	Optimum
DECT ULE	Encouraged
HomeMatic IP	Encouraged
Z-Wave	Encouraged
HomeMatic	Only encouraged if certain security prerequisites are met.
ZigBee	Discouraged

Table 5.1 The results of the analysis of different transmission standards.

network using a wired connection, for example with the help of appropriate bridges.

The DUC connection implementation requires special attention and deliberate security reviews. Research indicates that not every IoT vulnerability is based on a broken-by-design-flaw, i.e. a mistake in a protocol's specification. The early paper [FG13] describes a vulnerability in a Z-Wave-enabled door lock that occurred due to an incorrect implementation of a corresponding key exchange mechanism. Additionally, Z-Wave, in its early versions, did provide, but not require AES-encryption [NET16]. Recently, the *Z-Wave Alliance* provided an extensive framework for simplifying encrypted Z-Wave communication, namely *Security 2 (S2) Framework*, which is now mandatory for all newly certified Z-Wave DUCs [All17]. Hence, the Z-Wave protocol is considered to be secure in its most current version.

Generally speaking, as these protocols are relatively new and updated revisions fix fundamental security problems on a regular basis, it is recommended to adhere to the newest protocol versions whenever possible. Owing to the complex potential system configurations, a lack of compatibility and the fact that Security-by-Obscurity does rarely hold, it is discouraged to develop a proprietary communication protocol.

When device pairing is concerned, the Bosch Smart Home system presented ideal behavior by implementing second- or even triple-factor-pairing, as demonstrated in Chapter 3.2.1.

Table 5.1 summarizes the information stated above and yields the recommendations for Area 1.

Recommendation: Use a secure protocol from Table 5.1.
Severity after implementation: Level 2
The severity is set to Level 2, because in most real-world scenarios, not using any dangerous

protocol is rarely feasible due to market pressure and compatibility reasons.

Probability after implementation: Level C

The same justification as for the severity value applies.

Risk ⇒ Medium

5.2 Area 2: LAN and WAN Connections

In an effort to eliminate or minimize the impact described in Chapter 4.2.2, the following techniques were identified.

Passive Attacks

1. **Traffic interception** (cf. page 62)

   To diminish the consequences of this attack, the same technique that hindered Bartkowski from analyzing network-internal traffic in Chapter 3.2.4 is recommended: Traffic should be encrypted in a way that no information about the transferred data is revealed. This includes the encryption of sensitive keywords like `user` (contrary to the findings in Chapter 3.2.4, page 44) using a strong encryption algorithm, AES-128 at minimum. As for TLS-encrypted connections, a minimum version of TLS 1.2 in connection with the ciphersuites recommended in the BSI checklists [Sic17] should be employed. As described in Chapter 3.2.4, TLS 1.3 is about to be released. Accordingly, the system architecture should be kept maintainable and secure software update mechanisms should be provided. Where applicable, the mutual authentication, certificate pinning and HSTS (cf. Chapter 3.1) features of TLS and HTTPS can be taken into consideration. These three factors are commonly disregarded and little-known to developers. Chapter 3.2.3 highlighted the generation of a self-signed certificate for Bosch Smart Home, potentially useful for mutual authentication implementations.

   Recommendation: Implement encryption and advanced techniques like certificate pinning and HSTS.

   Severity after implementation: Level 3

   Probability after implementation: Level C

The attacker could try to break the encryption scheme. If strong encryption algorithms are used, the cost of brute-force attacks is very high, which leads to a probability value of *Remote*.

Risk ⇒ Low

2. **Port scans and reconnaissance tools** (cf. page 63)

   It should be ensured that no smart home device provides ports not actively needed for the functionality of the system. This implies the deactivation of unnecessary system services, for instance services included in the basic operating system of the smart home controller. This was examined in Chapter 3.2.2, where the Bosch Smart Home system provided only a single open port. For high-security environments, specialized firewalls support the detection of port scans and block according traffic. However, attackers may analyze a particular controller in a different network, detect open ports and connect to them in the target network without performing additional scans, rendering a firewall's protection insufficient for this attack scenario.

   Recommendation: Disable unused services, perform manual port scans to reveal ports open by mistake. Keep network-enabled services and software up-to-date.

   Severity after implementation: Level 3

   Probability after implementation: Level A

   The severity and probability values remain unchanged, as a port scan cannot be prevented by simple and reasonable techniques; the above recommendations are solely meant to further reduce the potential impact of the insights of a port analysis. Firewalls may be able to detect port scans. However, elaborate scanning techniques are able to hide port scans efficiently, by scattering them across a large time span.

   Risk ⇒ Medium

Active Attacks

1. **Packet spoofing** (cf. page 63)

   To provide the controller with a method of verifying that a specific packet was really sent by the intended communication partner, the datagrams need to be digitally signed. Therefore, manufacturers should equip smart home hardware with certificates. Smartphone apps should generate certificates unique to the corresponding device and save them securely in a KeyStore. The uniqueness is essential, otherwise the attacker might use his own smartphone

to function as a legitimate remote control. Subsequently, a reasonable pairing mechanism is needed to make sure the smart home controller knows about each remote control's certificate, to facilitate signature verification. Coqon demonstrated an advanced technique for message authentication over a separate, trusted channel (cf. Chapter 3.3).
When message tokens are employed for security (as for Coqon, cf. Chapter 3.3), they should be refreshed regularly, ideally for every datagram.

Recommendation: Implement message authentication and signing. Refresh tokens regularly.
Probability after implementation: Level D
Risk ⇒ Eliminated

2. **Replay attack** (cf. page 64)
   To prevent replay attacks, the smart home system's network protocol should include unique numbers or strings (so-called *nonce values*) for every relevant packet. Upon receipt, the system must actively reject packages containing a nonce value that was used before. This is only useful in combination with encrypted or digitally signed datagrams, to prevent attackers from simply changing the nonce value contained in the packet to another arbitrary value.

   Recommendation: Use an authenticated nonce value algorithm for datagrams.
   Probability after implementation: Level D
   Risk ⇒ Eliminated

3. **Man-In-The-Middle attack** (cf. page 64)
   To prevent most Man-In-The-Middle attacks, packet authentication with digital certificates, as described above under *Packet spoofing*, is recommended. Further, authentication is also essential for all web servers, including public cloud-based controlling websites and network-internal web interfaces hosted on the smart home controller. Usually, this is done via TLS, but as described in Chapter 2.3.3, self-signed certificates for network-internal IP addresses may lead to security prompts imposed by the web browser and other issues. These dialogs require the users to manually check and potentially trust the controllers certificate upon first access. This may unsettle inexperienced users and should be clarified in the owner's manual.
   The system's communication protocols should incorporate signed (i.e. authenticated) confirmation messages with nonce values (for example by including sequential package numbers),

e.g. via handshake algorithms, to prevent adversaries from maliciously dropping important data packets.

A different concern is a reliable home network detection, contrary to the one implemented by the Bosch system on page 42. Tips can be found in the according Chapter 3.2.3.

Recommendation: Implement message authentication and signing, implement handshake protocol and use TLS wherever feasible. For home network detection, rely on unique values (e.g. the BSSID).

Severity after implementation: Level 3

Probability after implementation: Level C

Adversaries could still interfere with the network by completely dropping every single packet, i.e. also the handshake protocol's packets. However, in this case, due to the recommended and reliable connection-oriented TCP protocol, devices can detect when the whole connection cannot be established and when the initialization of the handshake protocol fails.

When no handshake protocol is used, the attacker may simply assume the controller's identity and drop datagrams. The regular, unauthenticated TCP protocol would state successful packet delivery, even though the original controller has never received the datagram.

Risk ⇒ Low

5.3 Area 3: Remote Controls

This section is concerned with security recommendations related to web interfaces and Android apps, originally presented in Chapter 4.2.3.

1. **Exploiting outdated third-party libraries** (cf. page 65)

   Generally, it is discouraged to rely on extensive third-party libraries for solutions of small problems. Otherwise, the necessary effort to ensure every component of the system is updated to the most current revision increases. Additionally, large libraries may contain various bugs that are not always obvious due to the specific framework's complexity. Results from the HPE Security Fortify Open Review (FOR) analysis performed in 2016 showed that only 49 percent of several scanned programs included the latest version of some library [Wis16]. As [Wis16] continues, *developers can no longer afford to use third-party libraries without also keeping track of the libraries' updates and security profiles.* Chapter 3.1.2 demonstrated

how developers used a badly maintained framework, called *html5-boilerplate*, for the implementation of small and very simple HTML structures, whereas the latter could have been created manually in minutes.

To simplify the maintenance of third-party software, the software development process can be adjusted accordingly, as described in Chapter 5.5. Generally, the firmware of every smart home device (including DUCs) should be updatable, as implemented by Bosch Smart Home (Chapter 3.2). When implementing firmware update functionality, it has to be ensured that only official, digitally signed and authenticated firmware files can be installed. The system should provide special backup functionality in case an update fails.

Recommendation: Update third-party frameworks regularly, provide firmware update mechanisms and only rely on external libraries when reasonable.

Severity after implementation: Level 2

Probability after implementation: Level C

As shown in the hazard analysis in Chapter 4.2.3, third-party libraries always represent a potential security hazard. When libraries are used in their up-to-date versions, the security risk greatly decreases. Still, there may be some libraries used in a project that are not maintained anymore. Furthermore, proficient adversaries may invest a large amount of time in the analysis of a specific library, to find security bugs unknown to the developers. Thus, the risk is not assessed to be eliminated, because theoretically, it might still be possible for attackers to exploit external frameworks in the special cases described above. However, it is assumed that framework updates fix the most critical security flaws. The *Severity* value was adjusted accordingly.

Risk ⇒ Medium

2. **SQL injection** (cf. page 66)

   Input fields eventually connected to a database or a different kind of data persistence should always encode and filter special characters like ', ", <, > and others in a safe way, to prevent the system from mistakenly interpreting them as code or SQL commands. If data size is not a concern, this can be easily incorporated by using `Base64` for input encoding. Generally, a better method would be the usage of special *escaping functions*: Several programming languages provide functions to replace special characters by a safe representation. In PHP, for instance, the function `mysqli_real_escape_string()` is often used. It is important to note that some other functions, like PHP's `htmlspecialchars()`, will not escape characters

like ' or \, as these characters might be safe to use for HTML, but are potentially dangerous in the SQL syntax. `mysqli_real_escape_string()` does not escape the characters % and _, which have special meanings in some SQL statements as well.
Furthermore, the validity of the data should be checked. For instance, when a specific user is logged in, he should only be able to control his own associated smart home controller. Access to a controller of a different user should be denied. The same applies to values typically located in a specific interval. When the system asks for a time value, for example to schedule specific scenarios, the number should lie in a specific range, i.e. reach from 0 to 12, or 0 to 23, depending on the time format. When a value like `test` or 1234 is received instead, the system can assume, log and detect tampering.

Recommendation: Perform input filtering and escaping on every user input, also in URL parameters. Consider that every user input might contain potentially dangerous code. Perform validity checks on the received data.
Probability after implementation: Level D
Risk ⇒ Eliminated

3. **User authentication** (cf. page 66)
Ideally, users should authenticate upon every use of the app, to prevent unwarranted access. A reasonable implementation would also help to prevent the security problem described for Bosch Smart Home on page 39, concerned with login credential disclosure. This applies especially if the device is not protected by a passcode. For modern devices, this process can be simplified by using fingerprint sensors, e.g. TouchID for iOS. Security relevant actions, like the change of a password, must require an input of the (previous) password prior to execution. Too many wrong authentication trials should block access for a specific amount of time, to prevent brute-force attacks (cf. *Brute forcing*, page 85).

Recommendation: Implement easy-to-use ways for user authentication, for example by using fingerprint sensors. Provide alternatives for devices without fingerprint hardware and people with privacy concerns. Encourage end-users to employ a passcode for the device and actively check if the target device has a passcode enabled, to make further decisions on whether additional authentication measures are necessary.
Probability after implementation: Level D
Risk ⇒ Eliminated

4. **Exploit dynamically included resources** (cf. page 66)

   To prevent attackers from exploiting dynamically loaded resources, it is discouraged to request essential components from the WAN or LAN without authentication. For instance, as explained in the previous chapters, DNS entries can easily be manipulated and used to replace the expected content with fraudulent data. When dynamic loading cannot be omitted, correctly implemented TLS connections and digital signatures are indispensable to verify the provider's and file's identity. To prevent XSS attacks, similar to SQL injection prevention, checking dynamically loaded user input like URL parameters for special characters is required.

   Recommendation: Implement reliable authentication, only request resources from trusted sources and perform checks to verify the resource's content.

   Probability after implementation: Level D

   Risk ⇒ Eliminated

5. **GUI attacks** (cf. page 67)

   In order to be able to assume a certain basic level of security, the app should require a recent version of Android as an installation prerequisite. Android improves security with every revision, and the related attack vector described in the hazard analysis chapter on page 67 will be fixed for Android versions of O and above. At the time of writing, version O is still in development and can therefore not be set as a requirement. In the meantime, developers can only rely on Google preventing app store access for malicious apps. However, in a few years' time, as soon as usage statistics rate the version with a reasonable market share, the minimum operating system version requirement of the app should be adjusted.
   Generally, it is noted that old devices often do not offer support for recent Android versions. As a result, developers should actively monitor the market share of mobile operating systems and regularly choose a reasonable decision on the minimum requirements that does not exclude a high amount of potential users. When dangerous vulnerability exploits become known, the app may also check the installed operating system's version and potentially inform the user about an outdated version, recommending an update and stating easily understandable background information about the particular threat situation.
   The underlying operating system of every component of the smart home should be updated in regular intervals as well. This is not only to prevent potential GUI attacks, but, as stated above, to provide a certain level of minimum security for the complete ecosystem.

Recommendation: Regularly adjust the minimum operating system requirements of the app(s), keep operating systems up-to-date, inform users about outdated versions when appropriate.

Severity after implementation: Level 2

Probability after implementation: Level C

The most severe security problems will probably be fixed by updates. However, as updates are generally not provided on a daily basis, there is still a possibility for newly identified issues remaining attackable for a few weeks.

Risk ⇒ Medium

6. **Digital certificates** (cf. page 67)

   To ensure a correct implementation of digital signatures and digital certificates, the system should perform checks of the CRL. Then, broken certificates can be easily detected. Also, as described in the hazard analysis section, the system must not rely on the internal clock. A separate, legitimate and authenticated time server must be used in connection with a suitable protocol like NTP. As a single time server can easily fail or get compromised, the system should rely on three different time servers and combine their values in regular intervals. This is in accordance with well-known design principles for fail-safe systems: If one out of two time servers provided an incorrect value, the system would be unable to determine the correct value. As a result, three servers are necessary for reliable decisions.

   As described in the preceding recommendations, the system should provide precautionary update measures to ensure a broken certificate can easily be replaced by a firmware update. This is to prevent DUCs and other devices from remaining attackable forever. At the same time, making the certificate replaceable creates an additional asset worth of high protection. Past analyses of a well-known Android app, performed by the university's department this thesis is written in, revealed erroneous code instructing a security relevant component to proceed on any verification error of the certificate chain (for example by overriding the `onReceivedSslError(view, handler, error)` method of a WebView with `handler.proceed()`). As a result, a secure system should correctly verify the certificate chain to detect any inconsistencies and under no circumstances ignore verification errors. Where possible, certificate pinning as described in Chapter 3.1 is recommended.

   Recommendation: Pay special attention to the correct implementation of certificate algorithms, CRLs, firmware updatability and time synchronization. Consider certificate pinning.

Probability after implementation: Level D

Risk ⇒ Eliminated

7. **Insecure KeyStore** (cf. page 67)

   It is encouraged to use Android's integrated KeyStore, due to additional support and security measures imposed by the operating system. Even though users and apps with root access (*jailbreak*) are still able to avoid the protection mechanisms of the KeyStore, the latter is considered best-practice, as users are often informed about the security risks associated with jailbreaking a device. Sensitive data must never be saved unencryptedly. Always assume that hard-coded keys integrated into the program will be disclosed eventually, most probably by decompiling; Security-by-Obscurity is rarely applicable.

   Recommendation: Implement data persistence based on Android's KeyStore or iOS' Keychain. Always encrypt sensitive data and never store key material directly in code. If the latter cannot be avoided, refer to Coqon's behavior of securely adjusting SFTP usage rights in Chapter 3.3.2.

   Severity after implementation: Level 3

   Probability after implementation: Level C

   Risk ⇒ Low

8. **Insecure broadcasts or services** (cf. page 68)

   Android broadcasts and services can easily be programmed securely. Refer to the preceding bachelor's report [Mül15] for precise instructions and detailed recommendations.

   Recommendation: Refer to [Mül15].

   Probability after implementation: Level D

   Risk ⇒ Eliminated

A secure smart home app should actively support a correct security configuration. Therefore, the system should provide tips for essential configuration settings that cannot be changed by the system itself. For example, the app could provide a *Security Status Monitor* function, asserting the current encryption method of the WiFi network and, if not already set, recommend the WPA2 standard. Furthermore, adhering to the well-known security principle of *Least Privilege* (cf. [OWAc]), the Security Monitor might insist on disabling unused features, for instance the remote access functionality. As a last example, the app may advise users to rely on an Uninterruptible Power Supply for powering the controller and wired connections instead of wireless techniques

when possible.

5.4 Area 4: Cloud and Provider Servers

The following recommendations are based on the issues originally described in Chapter 4.2.4.

1. **Brute forcing** (cf. page 69)

   Servers must block clients performing too many failing login attempts after a specific number of trials (e.g. block a host for one hour if five incorrect logins were encountered). It is also possible to notify the user via e-mail, as implemented by Coqon (cf. Chapter 3.3). To impede brute-force attacks, information about the wrong credential type (e.g. username or password) must be left undisclosed. The length of the blocking period must be kept reasonable to assure legitimate users are not mistakenly excluded from the system. Additionally, user passwords should provide a certain minimum security level (e.g. consist of at least 8 characters, etc.). Generally, whenever default passwords are set by the system, a change must be required.

   Recommendation: Implement blocking mechanisms for too many incorrect login attempts, require a minimum security level for passwords and prevent continued usage of default passwords.

   Severity after implementation: Level 3

   Probability after implementation: Level C

   Brute forcing will take much longer when access is blocked after a few trials. Depending on how long access is impossible, attackers may focus on other attack vectors. However, brute forcing is not considered to be completely *Eliminated* by blocking periods. Still, this is assessed to be a best-practice, equally weighing up costs and benefits.

   Risk ⇒ Low

2. **Port scans and reconnaissance tools** (cf. page 70)

   Cf. Area 2: *Port scans and reconnaissance tools*, page 77.

   Recommendation: Disable unused services, perform manual port scans to reveal ports open by mistake. Keep network-enabled services and software up-to-date.

   Severity after implementation: Level 3

Probability after implementation: Level A

The severity and probability values remain unchanged as a port scan cannot be prevented by simple and reasonable techniques; the above recommendations are solely meant to further reduce the potential impact of the insights of a port analysis.

Risk ⇒ Medium

3. **SQL injection** (cf. page 69)

   Cf. Area 3: *SQL injection*, page 80.

   Recommendation: Perform input filtering and escaping on every user input, also in URL parameters. Consider that every user input might contain potentially dangerous code. Perform validity checks on the received data.

   Probability after implementation: Level D

   Risk ⇒ Eliminated

4. **DNS spoofing** (cf. page 69)

   For HTTP connections, including external HTTP links (cf. Chapter 3.1.2), the usage of the secure variant HTTPS is essential. The latter provides data security as well as verified identities while relying on DNS. Security-relevant systems without the need for domain names, e.g. because of specialized protocols not relying on HTTPS, can be addressed by static IP addresses, to reduce dependence on the susceptible DNS system.

   Recommendation: For regular HTTP connections, use TLS in its newest version. Use static IP addresses for other protocols when possible.

   Probability after implementation: Level D

   Risk ⇒ Eliminated

5. **Identity forgery** (cf. page 70)

   Cloud servers must at least assure that commands were really issued by the intended end-user. A server must verify the remote control's identity before executing commands, for example with the support of device pairing and certificates, as explained under *Packet spoofing* on page 77. A challenge-response protocol can also be employed: The server sends a unique value to the remote control, for example a timestamp. The app then encrypts the timestamp using a Unique Device ID (UDID). The latter is easily retrievable by code for Android and iOS devices, and is saved on the server during an initial pairing process (cf. *Packet spoofing*, Chapter 5.2). When the encrypted value is transferred back to the

server, the latter can compare it to his own encrypted value. If the values match, the device's identity is confirmed without having to directly transfer identity information over the line. The app might utilize an adapted version of the UDID and save it in the KeyStore upon the initial pairing process, to hinder other programs installed on the same device from maliciously performing the challenge-response protocol themselves. The server's identity should also be verified, for example by using similar means.

Recommendation: Implement device authentication, for example with challenge-response protocols.
Probability after implementation: Level D
Risk ⇒ Eliminated

6. **DDoS** (cf. page 70)
   DDoS attacks are generally hard to prevent. Firewalls can provide basic measures to avoid small attacks. For bigger attacks, companies like *Cloudflare*[3] provide networks with appropriate resources to cope with extremely large amounts of incoming connections. If a system outage cannot be assured to be impossible, the smart home system should be operable on its own, without cloud access, possibly with reduced functionality.

   Recommendation: Consider referring to external providers for reliable infrastructure, make the system as independent of the cloud as possible for emergency situations.
   Severity after implementation: Level 3
   Probability after implementation: Level C
   Risk ⇒ Low

5.5 Adjusting the Software Development Process

The following advisories represent general, regular actions to perform in order to ensure the system's security qualities. They can easily be integrated into many different development process models. Owing to the large variety of development management approaches, the advisories will not be integrated into any model. In fact, they will be categorized in actions to perform during development and after the software is published. Developers and companies can then adjust their

[3] *https://www.cloudflare.com/*

own development processes accordingly. The following actions will not be assessed concerning severity, probability nor risk, as they are considered means to further improve a system's security after the risk-minimizing measures of the preceding chapters have already been implemented.

Actions to perform during development

1. **Setup an artifact server**

   To simplify the use of up-to-date third-party libraries across software components, i.e. the app, the controller's firmware, etc., it is recommended to configure a project-related artifact repository[4] in combination with a build management tool like Maven[5] or Gradle[6]. Once a dependency receives an update and is refreshed on the artifact server, every recompiled software component will automatically include the up-to-date library. As a result, the development process must include testing and other steps to ensure the software is compatible with the updated library. It is generally advised to only rely on well-known and well-maintained libraries, to minimize the risks of undiscovered security issues.

2. **Perform hazard analysis, provide backup functionality**

   To support a reliable security architecture in a *Security by Design* manner, it is essential to consider potential security hazards from the beginning of development, by performing threat modeling according to Chapter 2.4. For large commercial systems, instead of adhering to the simplified thread modeling approach introduced in this document, following Microsoft's well-known STRIDE methodology is recommended. Many exemplary hazards were already exposed in the previous chapters. However, this list is not exhaustive and will vary for every usage scenario.

   In case a single security mechanism or connection fails, the architecture should include backup functionality. Coqon (Chapter 3.3), as an example, provides a UMTS channel when the WiFi signal is lost. However, as soon as both connections are interrupted, the complete system becomes unusable, leaving potential for optimization. As a result, the system should not depend on a single component (i.e. the cloud) and provide means to resume operation with limited functionality upon failure, for example.

3. **Rely on standards when possible**

   The OWASP 2017 top ten list of application security risks, which is currently in a work-in-progress-state, lists *Broken Authentication and Session Management* at the second position.

[4] For example, the open-source variant Artifactory, *https://www.jfrog.com/artifactory/*
[5] *https://maven.apache.org/*
[6] *https://gradle.org/*

According to the list, *developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws.* [OWAe] The statement was implicitly confirmed by Coqon's project manager Kevin Löhmann in Chapter 3.3: The principle of *Build your own Security* rarely leads to reliable results. Therefore, it is recommended to refer to well-known and broadly accepted standards and frameworks for encryption and other security-relevant techniques whenever possible.

4. **Adhere to the principles of security**

   Various checklists and resources about the general development of secure systems exist. The development process should adhere to the most common resources and might also have to be adapted for security. For example, Figure 5.1 depicts an adapted version of the classic waterfall development process model, where security measures have been integrated at many points. Additionally, developers should be aware of basic resources like the OWASP list of software security principles (cf. [OWAa]), the 2016 top ten list of mobile security issues (cf. [OWAd]) and BSI's extensive *Grundschutzkataloge* (cf. [Sic]). Moreover, the FOR analysis mentioned in Chapter 5.3 lists many common vulnerabilities as well (cf. [Ent]).

   For instance, a security checklist imposed by security analyst Garry McGraw listed the *Keep trust to yourself* principle, stating *(...) don't give out more information than you need to* [Fes01]. With regard to Bosch Smart Home, it becomes obvious that flaws like the display of paired `SGTINs` (cf. Chapter 3.2.3, specifically page 40) would have been avoidable by obeying aforementioned principle. The same checklist also mentions *Follow the principle of least privilege - That means don't give out more privilege than you have to. For example, don't provide access to your entire file system* [Fes01]. Coqon's user role separation, limited SFTP user rights and its one-way-behaviour for cellular connections are adequate examples in favor of this principle. Lastly, the *Practice defense in depth* [Fes01] proposal encourages developers to implement techniques like second-factor pairing (cf. the pairing mechanism of Bosch Smart Home, Chapter 3.2.1) or second-factor authentication.

5. **Employ security specialists**

   Note: This recommendation is also applicable for the Actions to perform upon and after release *phase.*

   Do not let moderate security become a by-product of development. Creating a solid security architecture requires time, extensive training and knowledge. As a result, it is recommended to allow for separate teams or external companies solely for security architecture develop-

ment, testing and validation. As an example, there exist many companies specialized in penetration testing for complex software systems, possessing insider information about particular security vulnerabilities and elaborate safeguarding techniques. As described in the introduction of Chapter 4, a single security expert does rarely suffice. Different perspectives are essential for hazard analysis.

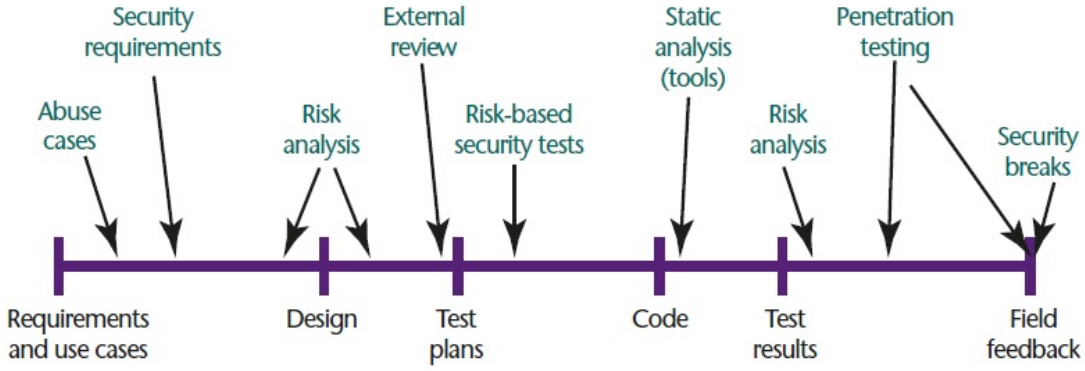

Figure 5.1 An adapted version of the waterfall development process model, where hazard analyses, security reviews, penetration tests and other means have been incorporated at various steps of the model. Source: [McG, p. 34]

Actions to perform upon and after release

1. **Regularly check libraries for updates**

   As mentioned in the previous subsections, it is recommended to regularly check third-party components for potential security updates after software release. The same applies to a revision of operating system requirements, as described for *GUI attacks* on page 82.

2. **Regularly ensure validity of certificates**

   Check every certificate's start and end dates for validity. Add broken certificates to official CRLs, provide according updates.

3. **Monitor log files for anomalies**

   The system's and server's log files should be checked regularly to detect abnormal use of the system, i.e. DDoS or brute-force attacks, or connection attempts to URLs and files that shouldn't be accessed by regular users. Firewalls and other technical aids can detect these problems automatically. This procedure can help to trace attack targets and to identify (additional) security assets (cf. Chapter 4.2)

4. **Regularly check journals for security vulnerabilities and new standards**

   At the time of writing, new security vulnerabilities related to smart home are presented

in the news almost on a daily basis. It is essential for developers and/or project managers to pursue various sources for the most recent findings about security vulnerabilities, especially in the context of wireless protocols like ZigBee (cf. Chapters 4.2.1, 5.1) or the security of cipher suites. The same applies to new protocols, as TLS 1.3 is currently under development (cf. Chapter 2.3.2) and an implementation of newer standards is generally recommended; presuming a verification and validation of a standard's security has been conducted in advance. New attacker models and threats require a regular update of the initial threat analysis.

5.6 Conclusion

As a comparison to the original risk assessment of the attack vectors in Table 4.4, Table 5.2 now provides a summary of the risk levels defined after the application of the precautionary measures. The graph in Figure 5.2 compares the values to the original risk distribution. It becomes obvious that most security hazards can be easily prevented and reduced to the *Eliminated* or *Low* category, leaving no issue in the *High* category. However, considering a general smart home architecture, not every issue is easily fixable with current technology. It is up to future research to develop and propose more effective alternatives to the according hazards.

Probability \ Severity	High [0]	Medium [3]	Negligible [7]
Likely [2]	High [0]	High [0]	Medium [2]
Possible [0]	High [0]	Serious [0]	Medium [0]
Remote [8]	Serious [0]	Medium [3]	Low [5]
Eliminated [10]	Eliminated [10]		

Table 5.2 An overview of the number of attack vectors identified in each category after the implementation of security fixes, cf. Table 4.4. It is noted that the column sums do not include the *Eliminated* category, as the according eliminated threats were not assessed with regard to the *Severity* indicator.

Figure 5.3 illustrates the optimized system architecture. Basically, the model is based on the original architecture from Figure 2.3, which was explained in the according Chapter 2.5, and differs in small details. For example, derived from Coqon (Chapter 3.3), a secure channel is

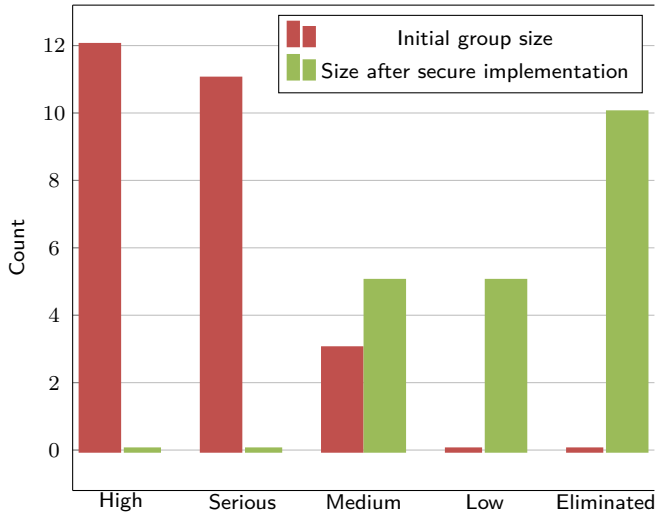

Figure 5.2 The current attack vector risk distribution of Table 5.2 compared to the original values of Table 4.4.

recommended, ideally separate from the broadband connection. Every connection should be encrypted, where possible via HTTPS (especially for cloud connections) or by implementing a separate mechanism (e.g. provided by a security framework) for custom LAN protocols. Generally, each suitable HTTPS implementation should be extended by the additional security techniques certificate pinning and HSTS introduced in Chapter 2.3.3.

Owing to the simple implementation and general acceptance, the cloud communication should adhere to the commonly used JSON-RPC protocol described in Chapter 2.3.5. Furthermore, the system is recommended to provide a cloud-independent LAN-local remote control interface, to enable control when a cloud connection is not available or undesired. A direct link between DUCs and remote control, as depicted in Figure 2.3, is not considered to be a recommended part of the reference architecture: Owing to the wide variety of protocols and DUCs typically involved in a smart home, it cannot be guaranteed that every DUC provides aforementioned direct interface. Additionally, in order for the controller to maintain an up-to-date overview of the DUCs' states, it is important for it to remain the central controlling interface. Direct connections may otherwise invalidate the controller's internal state representation. To simplify a central user credential management for authentication, the well-known and frequently-used OAuth protocol is proposed and was already encountered in the Qivicon analysis in Chapter 3.1. However, it is essential to handle access tokens correctly, contrary to the findings in Chapter 3.1.2. To support the traceability of bugs and attacks, the complete system must log security-

relevant actions, both, locally on the controller and on cloud servers, depending on a particular action's scope. When logging is concerned, legislative privacy restrictions are to be considered.

Lastly, by incorporating a UPS, the controller's availability can be greatly improved.

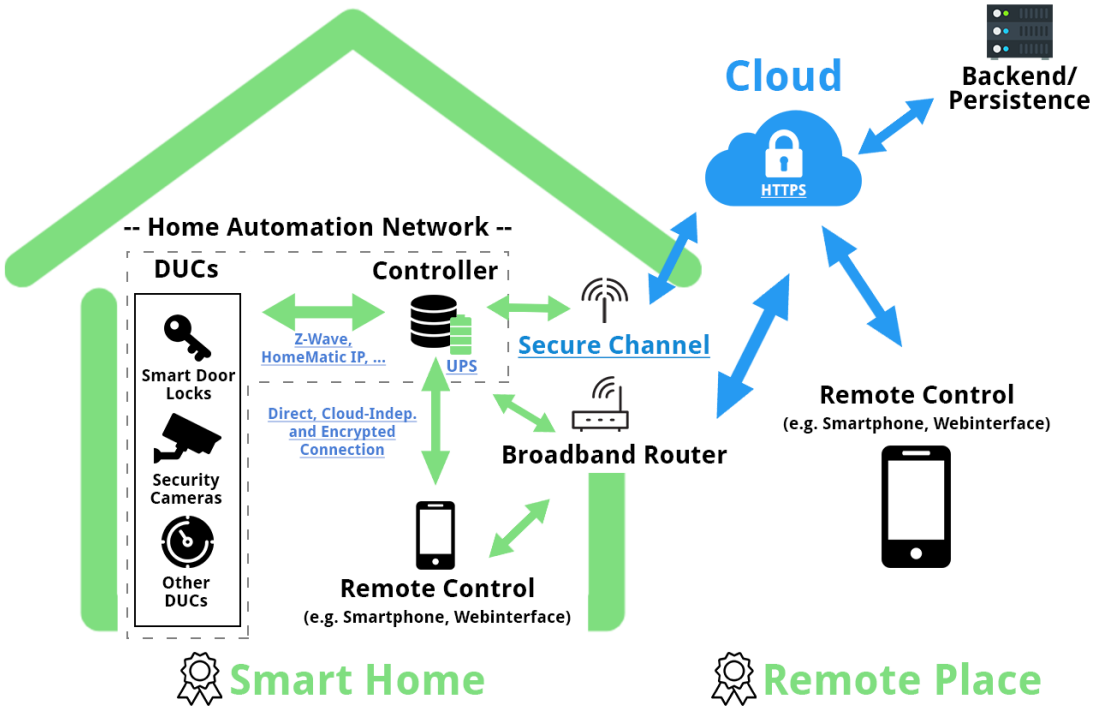

Figure 5.3 An illustration of the recommended system architecture, based on Figure 2.3. Parts adapted or renamed in relation to Figure 2.3 have been underlined in the graphic. The certificate icons in the bottom part of the graphic illustrate that every system entity (e.g. controller, remote control, etc.) should provide reliable identity information through certificates, as explained in the previous sections. Icon source: [FMM]

Chapter 6

Conclusion and Outlook

The preceding chapters revealed many potential security vulnerabilities in addition to intriguing problems in real-world systems. It was demonstrated that smart home applications have their own specific requirements and hazards, due to the novelty of the market and the rarity and obscurity of well-founded, field-tested measure catalogs in the context of the IoT's security. The recommendations that were derived from architectural examinations and security analyses can be seen as a solid basis to obliterate the most common and profound mistakes, related to both, insecure code and development processes. On the other hand, the results can only be regarded as the start of the process of securing the world of smart home systems, as multiple questions remained unanswered and many manufacturers still rely on the principle of Security-by-Obscurity, treating security like a nuisance.

Further analyses, especially in the context of the *SecureSmartHomeApp* project, should continue with the examination of the Coqon (Chapter 3.3) system. Any well-known and widely used smart home system should be checked by applying the static and dynamic program analysis techniques introduced in the beginning of this thesis (Chapter 2.6) or by performing penetration tests, to reveal additional hazards. Manufacturers have to understand that marketing claims concerning the security of their proprietary system do not suffice: The systems are often used as intermediary between public and private space, for example when smart door locks are concerned. Thus, the systems protect values beyond material means, rendering them a very attractive target to any kind of attacker, requiring thorough security. Insecure systems are ubiquitous in the news, large security breaches greatly diminish the reputation of the affected manufacturers.

An additional topic described in the original research application and definition of the *SecureSmartHomeApp* project is concerned with the practical implementation of a security framework or,

alternatively, the definition of development patterns for smart home software. This thesis provides a solid basis therefor and highlights the areas the most relevant for a potential framework in Chapter 4. Consequently, many of the recommendations given in Chapter 5 can be implemented directly. The security framework would highly simplify and accelerate the improvement of smart home systems.

As many of the vulnerabilities presented in Chapter 3 could not be verified, future analyses should also check whether the mentioned flaws still exist in the most current software and app versions. The same applies to the best-practices given in Chapter 5: Developers will have to verify and potentially adapt the recommendations to assure they are applicable for their individual system configurations. Furthermore, as iOS apps are often excluded from the analyses, a separate examination concerning iOS security in the context of smart home systems will be helpful and relevant as well. iOS security is especially interesting, as some insights surprisingly revealed that app security varies for Android and iOS versions (cf. the certificate pinning issue, Chapter 3.1.2). Future research has to develop new techniques, ideally to reduce every remaining issue in Chapter 5.6 to the category *Eliminated*.

To conclude, it is noted that the results and recommendations were given on the basis of a fast moving ecosystem. New technologies evolve regularly and security vulnerabilities are found almost on a daily basis. Researchers may find security vulnerabilities in protocols like Z-Wave, considered safe at the time of writing. Therefore, in the future, the recommended technologies, algorithms and best-practices will probably change.

Appendix A

Appendix

A.1 Interview Guideline

The interview guideline of Listing A.1 was developed and used to determine the statements of Chapter 3.3, where the smart home system Coqon was analyzed. This questionnaire can easily be transferred and adapted for other smart home systems, allowing developers and researchers to expand Chapter 3.

```
Main facts about Coqon
======================
1.  What are the key differences of Coqon when compared to other smart↩
     home systems?
2.  What are the main goals of Coqon? Is security an important factor ↩
    and which standards are used to ensure the latter?
3.  Do neusta mobile solutions GmbH or neusta next GmbH & Co. KG ↩
    develop own DUCs?
4.  Does Coqon support complex scenarios (i.e. disable the radiator ↩
    when a window is opened)?
5.  Which wireless protocols are supported? Z-Wave, ZigBee, HomeMatic,↩
     etc.?
6.  How can the controller be integrated into the home network? LAN, ↩
    WLAN, etc.?
7.  What names does Coqon use for DUCs, controllers and other ↩
    important devices?
8.  Can end-users expand the controller's capabilities, for example by↩
     using USB bridges or other devices providing additional wireless ↩
    communication protocols?
9.  What are the mobile systems apps are provided for? Android, iOS, ↩
    Windows Phone, etc.? Generally, what operating systems are ↩
    supported?
10. Is the firmware updatable? What is the structure of the update ↩
    process?
11. When did the development team of Coqon start working on the ↩
    project? When was it first released to the market?

System architecture
===================
12. Does the system rely on a framework like openHAB? Which operating ↩
    system is used for the controller?
13. Are direct connections between DUCs and remote control intended (e↩
    .g. via BTLE)?
14. Is WAN access required, i.e. does the system need cloud or ↩
    internet access? If yes, what happens when the World Wide Web ↩
```

```
    connection is not available or Coqon's servers are down?
15. Are local connections intended, i.e. can remote control and home ↩
    base communicate directly (and solely) via LAN?
16. Does the controller provide a LAN-internal HTTP web interface? How↩
     does the user access the interface?
17. Does the manufacturer provide a public web (cloud) interface? How ↩
    does the user access the interface?
18. How are commands transferred from remote control to home base (and↩
     to DUCs)? Which protocols are used?

Security
========
19. (How) is LAN network traffic encrypted? How does the system deal ↩
    with the absence of (user-friendly) HTTPS/TLS?
20. Do WAN cloud connections rely on TLS?
21. Does the system check CRLs?
22. Does the system use self-signed certificates or cooperate with a ↩
    well-known CA?
23. Does the system use NTP servers for reliable time information?
24. Name at least three interesting security problems that occurred ↩
    during the system's development.
25. How did developers fix these security problems? Did they derive ↩
    best-practices for specific problems or did they rely on work-↩
    arounds?
26. How does the system cope with potential replay attacks?
27. How does the system ensure command authentication, i.e. is the ↩
    system protected against adversaries maliciously sending their own↩
     commands to the controller?
28. Is the system protected against brute-force attacks?
29. What means does the system use to defend itself against DDoS ↩
    attacks?
30. Does the development process incorporate special means to ↩
    regularly ensure a certain level of security, e.g. are there ↩
    regular security reviews or penetration tests?
31. What is considered to be the most important security asset (cf. ↩
    Chapter 4.1)?
32. How does the development process ensure that third-party libraries↩
     are constantly updated and used in their most current versions?
33. Does the development process incorporate the construction of ↩
    attacker models or other specific security quality assurance ↩
    measures?
34. What standards do developers adhere to in the context of security?
35. Can a firmware update replace broken certificates?

36. In your opinion, is there anything else relevant in the context of↩
     security that was not covered by the preceding questions?
```

Listing A.1 Questions used for the interview with Kevin Löhmann, translated from German into English

The validity of the interview's outcomes stated in Chapter 3.3 was confirmed by Kevin Löhmann on 21.08.2017 through the e-mail in Listing A.2.

```
Hello Mr. Müller,

I hereby confirm that the presentation of the coqon system in your ↩
    master thesis is in line with the contents of our interview.

Thank you for the review and the helpful conclusions in your thesis.

Best regards,
```

```
Kevin Löhmann.

---------------------------------------
neusta mobile solutions GmbH
Ein team neusta Unternehmen

Kevin Löhmann
Senior Consultant / Project Manager

Konsul-Smidt-Str. 24
28217 Bremen
Germany

Fon   +49 421 79 27 75 34
Fax   +49 421 79 27 75 59
Mobil +49 173 249 46 33

k.loehmann@neusta.de

www.neusta-ms.de | www.team-neusta.de
---------------------------------------

Handelsregister Bremen 22019
Geschäftsführung Holger Bothmer
```

Listing A.2 The confirmation e-mail of Kevin Löhmann, rendering a transcript of the interview unnecessary, in accordance with the examiner. It was received on 21.08.2017, 13:45.

A.2 Mail Communication

During the writing process of this thesis, in an effort to acquire additional information about smart home software and security implementations, we contacted several developers and companies. Excerpts of the most important mails are listed below for reference.

A.2.1 Communication with Deutsche Telekom AG and Qivicon Support

In an effort to receive answers to technical questions about Qivicon, the mail communication in Listings A.3 and A.4 was recorded and printed with permission.

```
Sehr geehrter Herr Müller,

vielen Dank für Ihre E-Mail. Leider ist es unseren Teams derzeit nicht↩
     möglich Ihre Anfrage zu unterstützen.

Wir wünschen Ihnen viel Erfolg bei Ihrem Projekt,

Ihr QIVICON Team
```

Listing A.3 A response by Qivicon support, referring to technical questions. The e-mail was referenced in Chapter 3.1.1.

```
Hallo Herr *****,

vielen Dank für das Telefonat und Ihre Hilfsbereitschaft!

Hier meine Fragen zum Qivicon-System:
- Im Qivicon-FAQ (https://www.qivicon.com/de/support/fragen-und-↩
    antworten/sicherheit/), Frage "Wie werden Qualität und Sicherheit ↩
    garantiert?", heißt es, dass die Qivicon-Server von der Telekom ↩
    betrieben werden. Darunter steht nun, dass Partner eigene Server ↩
    betreiben können. Jedoch haben wir nach einer Untersuchung des ↩
    Systems die Vermutung, dass Qivicon-Adaptionen nicht komplett ↩
    unabhängig vom Telekom-Backend betreibbar sind. Deshalb haben wir ↩
    uns gefragt, welche Kommunikation definitiv über die Telekom-↩
    Server läuft (unsere Vermutung: das Weiterleiten der RPC-Aufrufe ↩
    an die Qivicon Home Base), und welchen Teil Anbieter auf die ↩
    eigenen Server auslagern können (die Authentifizierung mittels ↩
    oAuth).

- Lässt sich ein Qivicon-System auch noch weiter betreiben, wenn die ↩
    Telekom-Server nicht erreichbar sind, zum Beispiel über eine ↩
    lokale Direktverbindung zwischen Smartphone und Home Base? Gibt es↩
     in diesem Fall irgendwelche Einschränkungen?

- Ist jetzt oder in Zukunft eine direkte Kommunikation (ohne Base) ↩
    zwischen den Aktoren und dem Smartphone, z.B. via BTLE, vorgesehen↩
    ?


Nochmals vielen Dank für Ihre Hilfe. (...)
```

Listing A.4 A different e-mail written in the course of this thesis with several questions about the Qivicon system architecture. It remained repeatedly unanswered by a Telekom technical support employee.

A.2.2 Communication with eQ-3 Support

Detailed information about the HomeMatic standard is sparse on the internet. In reaction to questions concerning the speed and range of HomeMatic and HomeMatic IP, eQ-3 support sent the messages printed with permission in Listings A.5 and A.6.

```
Sehr geehrter Herr Müller,

hiermit möchten wir uns für das von Ihnen entgegengebrachte Interesse ↩
    an unseren Produkten bedanken.

Ihre Fragen beantworten wir wie folgt:

(...)

Die Homematic Zentrale CCU2 und der Homematic IP Access Point habe ↩
    eine Funkreichweite von bis zu 400m (Freifeld).

(...)

Mit freundlichen Grüßen aus Leer

Ihr eQ-3 Support-Team

(...)
```

Listing A.5 E-mail received on 13.06.2017, 12:35.

```
Sehr geehrter Herr Müller,

Nach Rücksprache können wir Ihnen mitteilen, dass die Rate im ↩
    Regelbetrieb bei 10 kBit/s liegt. Dies gilt bei beiden Systemen.

Mit freundlichen Grüßen aus Leer

Ihr eQ-3 Support-Team

(...)
```

Listing A.6 E-mail received on 14.06.2017, 13:31.

A.3 List of Figures

A.4 List of Tables

A.5 List of Abbreviations

3DES Triple DES, S. 33

AES Advanced Encryption Standard, S. 33
ALE Annual Loss Expectancy, S. 55
AP Access Point, S. 42, 64
API Application Programming Interface, S. 3, 7, 10, 15, 26, 38, 58
ARP Address Resolution Protocol, S. 107

BSI Bundesamt für Sicherheit in der Informationstechnik, S. 12, 43, 49, 76, 89

CA Certification Authority, S. 6, 13–15, 43, 48, 104, *Glossary:* Certification Authority
CRL Certificate Revocation List, S. 6, 43, 44, 48, 67, 83, 90

DES Data Encryption Standard, S. 33
DUC Device Under Control, S. 22–25, 29, 34–36, 40, 44, 46, 47, 53, 58–62, 74, 75, 80, 83, 92, 104, 105, 107, 108

FOR HPE Security Fortify Open Review, S. 79, 89
FTP File Transfer Protocol, S. 10, 49

GUI Graphical User Interface, S. 7, 26, 32, 38, 67, 82, 90

HAN Home Automation Network, S. 21, 23–25, 28, 31, 53, 58, 59, 62–65, 104, 107
HSTS HTTP Strict Transport Security, S. 15, 76, 92
HTTP Hypertext Transfer Protocol, S. 6, 9–13, 15, 18, 19, 22, 47, 86, 105, 106, 110, 111
HTTPS Hypertext Transfer Protocol Secure, S. 12, 13, 15, 32, 33, 36, 39, 43, 44, 76, 86, 92, 104, 107

IETF Internet Engineering Task Force, S. 12
IoT Internet of Things, S. 1, 2, 16, 18, 47, 60, 61, 74, 75, 95, 108, 109, 112

LAN Local Area Network, S. 10, 13–15, 23, 24, 29, 30, 32, 34, 35, 38, 41, 42, 44, 45, 47, 48, 51, 59, 62, 64, 82, 92, 104, 107

MAC Message Authentication Code, S. 6, 10, 35, *Glossary:* Message Authentication Code

NIST National Institute of Standards and Technology, S. 55

OSI Open Systems Interconnection Model, S. 9, 10, 16, 105, 109–111
OWASP Open Web Application Security Project, S. 69, 88, 89

SFTP Secure File Transfer Protocol, S. 49, 84, 89
SSDP/UPNP Service Discovery Protocol/Universal Plug and Play, S. 25

UDID Unique Device ID, S. 86, 87
UDP User Datagram Protocol, S. 9, 10
UML Unified Modeling Language, S. 73
UPS Uninterruptible Power Supply, S. 22, 84, 93

VM Virtual Machine, S. 7

WAN Wide Area Network, S. 16, 23, 30, 38, 42, 45, 49, 82, 109

A.6 Glossary

APK (Android Package)
A file format used for distributing Android apps in compiled form.
S. 7, 30–33, 65

ARP Poisoning/ARP Spoofing
An attack where an adversary sends malicious Address Resolution Protocol (ARP) packets to bind a MAC address (e.g. his own) to a certain IP. It is therewith possible to intercept LAN packets destined for another host.
S. 64

ASCII
A well-known seven-bit code used to map characters to bit strings.
S. 10, 12, 19, 109

Authenticity
One of three classic security goals, referencing the importance of being able to prove that a message was really created by the expected party. Cf. *Digital signature*.
S. 1, 5, 6, 8, 12–14, 23, 30, 32, 33, 35, 41, 43, 48, 57, 58, 61, 63, 66, 70, 74, 76, 78–83, 87–89, 92, 108, 109, 111

Bluetooth Low Energy (BTLE)
A low-energy wireless transmission standard, based on Bluetooth, featuring IPv6 in its newest version.
S. 1, 16, 18, 22

Bridge (Smart Home)
An adaptor allowing for the integration of new protocols into the HAN. Bridges may also perform external calculations or provide means of communication for lightweight DUCs with very low CPU power, space or power consumption requirements.
S. 22, 25, 29, 34, 35, 75

Broadcast (Android)
Refers to system-internal messages sent by Android apps, for the reason of communicating with other apps or the operating system. They often contain sensitive data or inform apps about changes in the system's environment, for example when the device is physically rotated.
S. 8, 68, 84

BSSID (Basic Service Set Identifier)
A unique string identifying a specific WiFi access point.
S. 42, 79

Certificate Pinning
A technique used to increase the security of HTTPS and digital certificates, which associates hosts with a fingerprint of the expected certificate.
S. 14, 15, 33, 76, 83, 92, 96

Certification Authority
A trustworthy party vouching for relations between digital certificates and their owners.
S. 6, 106, 108

Cipher Suite
A selection of algorithms, block lengths and key exchange techniques, used for instance in the TLS standard for secure communication.
S. 12, 36, 43, 91

Confidentiality
One of three classic security goals, referencing the importance of preventing any third party from intercepting sensitive data.
S. 1, 5, 111

Cryptographic Hash Function
An algorithm generating a digest value of fixed length for messages of arbitrary length. Ideally, the value should be unique and equally distributed for any input message so that it is hard to find two messages with the same hash value.
S. 5, 6, 109

DDoS (Distributed Denial of Service)
An attack aiming at flooding web servers with data from a large amount of different hosts at the same time, to influence a service's availability.
S. 50, 60, 70, 87, 90

DECT ULE (Digital Enhanced Cordless Telecommunications Ultra-Low Energy)
A cost-efficient, low energy version of the communication standard DECT. DECT is commonly used for wireless telephones and voice transmission, ULE extends the standard by additional data transmission features useful for home automation, security and climate control. ULE devices can easily communicate the network status and alerting conditions via voice messages and SMS. [All]
S. 17, 18, 28, 73–75

Digital Certificate
A signed file issued by a trusted third party (cf. *Certification Authority*). Binds a public key to persons, organizations and other entities.
S. 6, 12–15, 32, 41, 43, 44, 48, 67, 68, 77, 78, 83, 86, 90, 104, 107, 109

Digital Signature
A method providing authenticity and integrity for digital data.
S. 5–7, 13, 44, 67, 77, 78, 80, 82, 83, 107–109

DNS (Domain Name System)
A protocol mapping readable host names like *www.example.com* to a server's IP address.
S. 10, 13, 43, 69, 82, 86, 109

HomeMatic
A common system and protocol for home automation with a large variety of DUCs, cf. [Homc]; [AGd].
S. 17, 18, 28, 61, 73–75, 103, 105, 108

HomeMatic IP
The successor to HomeMatic, providing IPv6 and therewith a more efficient solution for the IoT, cf. [AGa].
S. 17, 18, 28, 73–75, 103, 105

Integrity
One of three classic security goals, referencing the importance of being able to prove that a message was not changed by a third party on the way from sender to receiver. Cf. *Digital signature*.
S. 1, 5, 6, 108, 109

JSON (JavaScript Object Notation)
A method for concisely and legibly serializing objects for network transfer or persistence, using JavaScript syntax.
S. 18, 19, 35, 109

JSON-RPC
A technique used to remotely call software procedures via computer networks. The according parameters are encoded in the human readable JSON ASCII format.
S. 18, 30, 33, 92, 110

Keychain
A system developed by Apple, securely saving login credentials, certificates and other sensitive data.
S. 40, 84, 109

KeyStore
The Android KeyStore system lets developers store cryptographic keys and certificates in a container to make it more difficult to extract them from the device [Dev]. For iOS devices, a complementary technique named *Keychain* exists.
S. 8, 38, 41, 50, 67, 68, 77, 84, 87

MDNS (Multicast DNS)
A technique to establish DNS functionality without a central management server, developed in the context of the IoT [CK13].
S. 36

Message Authentication Code
Ensures integrity and authenticity of a message by encrypting a hash value using a symmetric encryption scheme. Cf. *Digital signature*.
S. 6, 10, 106

MITM (Man-In-The-Middle)
An attack method where, in a communication scenario between A and B, an attacker C identifies himself as B against A and as A against B. Thereby he is able to intercept or manipulate network traffic, as any communication is made (and forwarded) via C.
S. 13, 14, 26, 32, 33, 42, 51, 64, 67, 78

NAT (Network Address Translation)
A NAT maps a single public IP address, belonging to a WAN access point, to network internal (private) IP addresses of hosts.
S. 23, 24, 42, 49

Nonce Value
A unique, arbitrary value that is incorporated in data packets to ensure they are not interpreted twice, for example due to replay attacks.
S. 44, 78

NTP (Network Time Protocol)
A protocol for synchronizing clocks in computer networks, found on the Application Layer of the OSI-Model.
S. 10, 36, 43, 48, 49, 83

OAuth
An open authorization protocol. Users can permit software to access data provided by another service without disclosing any login credentials.
S. 33, 68, 92

Obfuscation
A technique that aims to hinder reverse engineering, mainly by renaming and reordering methods during the compilation process.
S. 26, 110

Phishing
An attack scenario where the user is redirected to a fraudulent website looking similar or identical to a trustworthy one. The malicious site usually provides input fields for sensitive data like passwords or credit card numbers. The data will then be sent to the attacker.
S. 32

QR-Code
A graphical representation of digital information, similar to a bar code.
S. 35, 36, 105

Replay Attack
An attack scenario where a malicious party intercepts network packets and resends them to their original destination, in order to invoke and/or repeat a specific system behavior, without the consent of the user.
S. 26, 44, 56, 64, 78, 109

RFID
A wireless, low-energy connection standard. Its range is often set to be approximately one meter. However, it can achieve 100 meters or more, depending on the implementation.
S. 25

RPC (Remote Procedure Call)
Cf. JSON-RPC.
S. 18, 19, 105

SCP (Secure Copy Protocol)
A protocol allowing for encrypted file transfer over computer networks, located on the Session Layer [Had96] of the OSI-Model (cf. Chapter 2.3.1).
S. 38

Security-by-Obscurity
A principle assuming that attacks are less likely to occur and succeed if attackers do not have precise knowledge of a system's security implementation. For instance, this is the case when code is obfuscated. In real-world scenarios, this principle usually does not hold.
S. 30, 75, 84, 95

Service (Android)
A component of an Android app that is executed in the background for processing operations of long duration. Alternatively, a service may sleep until specific commands are received.
S. 8, 68, 84

SQL (Structured Query Language)
A commonly used system for relational database schemes.
S. 58, 80, 81, 110

SQL Injection Attack
A scenario where an attacker tries to manipulate or read the database of a system by injecting SQL commands into URLs or input fields.
S. 40, 66, 69, 80, 82, 86

SSID (Service Set Identifier)
An arbitrary string labeling a wireless network, imposed by the network administrator.
S. 41, 42

SSL (Secure Sockets Layer)
The obsolete predecessor of TLS for secure HTTP connections.
S. 10, 12, 48

TCP (Transmission Control Protocol)
A reliable network protocol located on the Transport Layer of the OSI-Model, cf. Chapter 2.3.1.
S. 9, 10, 12, 13, 24, 36, 43, 79

TCP/IP-Model
A more general and simplified version of the OSI-Model, specifically for internet usage.
S. 10, 18

TouchID
An authentication method used by iOS devices, involving fingerprint scanning.
S. 40, 81

Traffic Interception Attack
An attack where an adversary intercepts data packets between sender and receiver, endangering the security goal *confidentiality*.
S. 12

Transport Level Security (TLS)
The modern and recommended security protocol for secure HTTP connections.
S. 10, 12–14, 36, 41, 43, 48, 76, 78, 79, 82, 86, 91, 107, 110

UMTS (Universal Mobile Telecommunications System)
A standard for cellular network communication. UMTS is the successor of the slower EDGE technology.
S. 45–50, 88

URL (Uniform Resource Locator)
A string referencing a resource, e.g. *http://www.example.com/index.html*.
S. 13, 33, 47, 66, 69, 81, 82, 86, 90, 110

VPN (Virtual Private Network)
A technique and protocol used to securely connect to a different computer network, independently of the communication medium.
S. 23, 46, 51

WEP (Wired Equivalent Privacy)
A deprecated, insecure standard for encrypting data sent through a WiFi network.
S. 64, 111

White Label Product
A product that comes unbranded and can be labeled with the logo of a third-party company.
S. 28, 34

WPA2 (Wi-Fi Protected Access 2)
A secure standard for encrypting data sent through a WiFi network. WPA2 is the successor of WEP.
S. 48, 59, 84

WPAN (Wireless Personal Area Network)
A wireless network that is only used for small distances and devices around a single person's workspace, contrary to WiFi [KKK14, p. 7752].
S. 16, 17

XSS Attack (Cross-Site Scripting)
A technique where an attacker tries to inject code into web applications, to force client applications like web browsers to execute malicious code.
S. 40, 82

Z-Wave
A common standard for wireless data transfer in the context of the IoT.
S. 17, 18, 45, 46, 73, 75, 96

ZigBee
A common standard for wireless data transfer in the context of the IoT.
S. 16–18, 28, 46, 59, 61, 73–75, 91

A.7 Bibliography

Literature

[And08] Ross J. Anderson. *Security Engineering. A Guide to Building Dependable Distributed Systems*. 2nd ed. Wiley, Apr. 2008. ISBN: 9780470068526.

[BSG14a] Carsten Bormann, Karsten Sohr, and Stefanie Gerdes. *isec: Informationssicherheit. Introductory presentation.* Slides accompanying the lecture 'Informationssicherheit' of the University of Bremen. Oct. 2014.

[BSG14b] Carsten Bormann, Karsten Sohr, and Stefanie Gerdes. *Netzsicherheit. Presentation no. 7.* Slides accompanying the lecture 'Informationssicherheit' of the University of Bremen. Nov. 2014.

[Gou+02] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy. *HTTP: The Definitive Guide*. O'Reilly Media, Inc., Sept. 2002. ISBN: 9781449379582.

[KKK14] Kanchan Kaushal, Taranvir Kaur, and Jaspinder Kaur. "ZigBee based Wireless Sensor Networks". In: *International Journal of Computer Science and Information Technologies* 5 (2014). ISSN: 0975-9646.

[Kya17] Othmar Kyas. *How To Smart Home: A Step by Step Guide for Smart Homes & Building Automation*. 5th ed. Key Concept Press, May 2017. ISBN: 9783944980126.

[Sch17] Jürgen Schmidt. "Weniger ist mehr. Was die anstehende Version TLS 1.3 bringt". In: *c't* (Feb. 2017), p. 172.

[Sho14] Adam Shostack. *Threat Modeling. Designing for Security*. John Wiley & Sons, 2014. ISBN: 9781118809990.

Online sources

[AGa] eQ-3 AG. *Die neue Smart Home Generation*. [Retrieved on 26.04.2017]. URL: *http://www.eq-3.de/produkte/homematic-ip.html*.

[AGb] eQ-3 AG. *FAQ - häufige Fragen*. [Retrieved on 13.06.2017]. URL: *http://www.eq-3.de/service/faq.html*.

[AGc] eQ-3 AG. *FAQ - häufige Fragen*. [Retrieved on 13.06.2017]. URL: *http://www.eq-3.de/Downloads/eq3/download%20bereich/handbuecher/Homematic_IP-Anwenderhandbuch.pdf*.

[AGd] eQ-3 AG. *Homematic - Ein ganzes Haus voller guter Ideen*. [Retrieved on 26.04.2017]. URL: *http://www.eq-3.de/produkte/homematic.html*.

[AGe] Deutsche Telekom AG. *QIVICON Home Base*. [Retrieved on 11.07.2017]. URL: *https://www.qivicon.com/assets/Products/Home-Base-USER-GUIDE-3101-German-1.pdf*.

[al08] D. Cooper et. al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. [Retrieved on 10.08.2017]. May 2008. URL: *https://tools.ietf.org/html/rfc5280*.

[All] ULE Alliance. *Why ULE?* [Retrieved on 26.04.2017]. URL: *https://www.ulealliance.org/why-ule*.

[All17] Z-Wave Alliance. *Mandatory Security Implementation for All Z-Wave Certified IoT Devices Takes Effect Today*. [Retrieved on 06.06.2017]. Apr. 2017. URL: *http://z-wavealliance.org/mandatory-security-implementation-z-wave-certified-iot-devices-takes-effect-today/*.

[Art17] Michael Artmann. *openHAB 2 - Neues Level für die Smart-Home-Steuerung*. [Retrieved on 11.05.2017]. Feb. 2017. URL: *https://www.homeandsmart.de/openhab-2-smart-home-software-open-source*.

[bit] bitkom. *ISO/IEC 21827 (SSE-CMM)*. [Retrieved on 20.04.2017]. URL: *http://www.kompass-sicherheitsstandards.de/43822.aspx*.

[Bro16] Chris Brook. *Chrome to Label Some HTTP Sites 'Not Secure' in 2017*. [Retrieved on 30.03.2017]. Sept. 2016. URL: *https://threatpost.com/chrome-to-label-some-http-sites-not-secure-in-2017/120452/*.

[BW15] Mario Ballano Barcena and Candid Wueest. *Insecurity of the Internet of Things*. [Retrieved on 17.05.2017]. Mar. 2015. URL: *https://www.symantec.com/content/en/us/enterprise/fact_sheets/b-insecurity-in-the-internet-of-things-ds.pdf*.

[CK13] S. Cheshire and M. Krochmal. *Multicast DNS*. [Retrieved on 26.04.2017]. Feb. 2013. URL: *https://tools.ietf.org/html/rfc6762*.

[Def00] Department of Defence. *System Safety: MIL-STD-882E*. [Retrieved on 30.05.2017]. Feb. 2000. URL: *http://www.system-safety.org/Documents/MIL-STD-882E.pdf*.

[Dev] Android Developers. *Android Keystore System*. [Retrieved on 26.04.2017]. URL: *https://developer.android.com/training/articles/keystore.html*.

[ENE] ENERGY.GOV. *CYBERSECURITY CAPABILITY MATURITY MODEL (C2M2)*. [Retrieved on 20.04.2017]. URL: *https://energy.gov/oe/services/cybersecurity/cybersecurity-capability-maturity-model-c2m2-program/cybersecurity*.

[Ent] Hewlett Packard Enterprise. *HPE Security Research. Cyber Risk Report 2016*. [Retrieved on 07.08.2017]. URL: *https://www.thehaguesecuritydelta.com/media/com_hsd/report/57/document/4aa6-3786enw.pdf*.

[eQ316] eQ-3. *AV-TEST zeichnet Homematic IP mit dem Siegel „Geprüftes Smart Home Produkt" aus*. [Retrieved on 06.06.2017]. Jan. 2016. URL: *http://www.eq-3.de/aktuel*

les/newsreader/av-test-zeichnet-homematic-ip-mit-dem-siegel-gepruefte-smart-home-produkt-aus.html.

[Fes01] Paul Festa. *Gary McGraw: 10 steps to secure software*. [Retrieved on 02.07.2017]. Dec. 2001. URL: *http://www.zdnet.com/article/gary-mcgraw-10-steps-to-secure-software/*.

[FG13] Behrang Fouladi and Sahand Ghanoun. *Security Evaluation of the Z-Wave Wireless Protocol*. [Retrieved on 06.06.2017]. 2013. URL: *https://sensepost.com/cms/resources/conferences/2013/bh_zwave/Security%20Evaluation%20of%20Z-Wave_WP.pdf*.

[Gie17] Hauke Gierow. *Wenn Mirai und Wanna Cry sich zusammentun*. [Retrieved on 25.05.2017]. May 2017. URL: *https://www.golem.de/news/ransomware-wenn-mirai-und-wanna-cry-sich-zusammentun-1705-127962.html*.

[GoD] GoDaddy.com. *Can I request a certificate for an intranet name or IP address?* [Retrieved on 14.06.2017]. URL: *https://my.godaddy.com/help/can-i-request-a-certificate-for-an-intranet-name-or-ip-address-6935*.

[Gol17] Golem. *Nest-Kamera kann per Bluetooth deaktiviert werden*. [Retrieved on 22.03.2017]. Mar. 2017. URL: *https://www.golem.de/news/ueberwachungskamera-nest-kamera-kann-per-bluetooth-deaktiviert-werden-1703-126858.html*.

[GS111] GS1. *GS1 EPC Tag Data Standard 1.6*. [Retrieved on 27.04.2017]. Sept. 2011. URL: *http://www.gs1.org/sites/default/files/docs/epc/tds_1_6-RatifiedStd-20110922.pdf*.

[Had96] Rhys Haden. *The OSI Model*. [Retrieved on 26.04.2017]. 1996. URL: *http://www.rhyshaden.com/osi.htm*.

[Homa] Bosch Smart Home. *Bosch Smart Home FAQ*. [Retrieved on 21.04.2017]. URL: *https://www.bosch-smarthome.com/de/de/service/faq*.

[Homb] Bosch Smart Home. *Raumklima Starter-Paket*. [Retrieved on 24.07.2017]. URL: *https://www.bosch-smarthome.com/de/de/produkte/smart-system-solutions/raumklima-starter-paket*.

[Homc] HomeMatic. *HomeMatic–wir machen Home Control einfach*. [Retrieved on 26.04.2017]. URL: *https://www.homematic.com*.

[Inc] Apple Inc. *Unauthorized modification of iOS can cause security vulnerabilities, instability, shortened battery life, and other issues*. [Retrieved on 27.04.2017]. URL: *https://support.apple.com/en-us/HT201954*.

[jQu] jQuery.com. *jQuery Core – All Versions*. [Retrieved on 11.05.2017]. URL: *https://code.jquery.com/jquery/*.

[JSO] JSON-RPC. *JSON-RPC 2.0 Specification*. [Retrieved on 18.04.2017]. URL: *http://www.jsonrpc.org/specification*.

[McG] Gary McGraw. *Software Security*. [Retrieved on 10.08.2017]. URL: *https://www.cigital.com/papers/download/software-security-gem.pdf*.

[NET16] NETWORKWORLD. *EZ-Wave: A Z-Wave hacking tool capable of breaking bulbs, abusing Z-Wave devices*. [Retrieved on 06.06.2017]. Jan. 2016. URL: *http://www.networkworld.com/article/3024217/security/ez-wave-z-wave-hacking-tool-capable-of-breaking-bulbs-and-abusing-z-wave-devices.html*.

[Ohl13] Günther Ohland. *Funkprotokolle: Z-Wave, HomeMatic und RWE im Vergleich*. [Retrieved on 06.06.2017]. Aug. 2013. URL: *http://www.pc-magazin.de/ratgeber/funkprotokolle-ueberblick-rwe-zwave-homematic-1530051.html*.

[OWAa] Open Web Application Security Project (OWASP). *Category:Principle*. [Retrieved on 03.07.2017]. URL: *https://www.owasp.org/index.php/Category:Principle*.

[OWAb] Open Web Application Security Project (OWASP). *Certificate and Public Key Pinning*. [Retrieved on 10.07.2017]. URL: *https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#What_Is_Pinning.3F*.

[OWAc] Open Web Application Security Project (OWASP). *Least privilege*. [Retrieved on 29.06.2017]. URL: *https://www.owasp.org/index.php/Least_privilege*.

[OWAd] Open Web Application Security Project (OWASP). *Mobile Top 10 2016-Top 10*. [Retrieved on 06.08.2017]. URL: *https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10*.

[OWAe] Open Web Application Security Project (OWASP). *OWASP Top 10 Application Security Risks - 2017*. [Retrieved on 10.08.2017]. URL: *https://www.owasp.org/index.php/Top_10_2017-Top_10*.

[Plö15] Henryk Plölz. *On the Security of AES in HomeMatic*. [Retrieved on 11.05.2017]. Sept. 2015. URL: *https://blog.ploetzli.ch/2015/on-the-security-of-aes-in-homematic/*.

[Qiv] Qivicon. *QIVICON Fragen und Antworten*. [Retrieved on 13.04.2017]. URL: *https://www.qivicon.com/de/support/fragen-und-antworten/sicherheit/*.

[Ren14] Tessel Renzenbrink. *30C3: Hacks Demonstrate Insecurity of Home Automation Devices*. [Retrieved on 11.05.2017]. Jan. 2014. URL: *https://www.elektormagazine.com/articles/30c3-hacks-demonstrate-insecurity-of-home-automation-devices*.

[Res17] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. [Retrieved on 04.05.2017]. Apr. 2017. URL: *https://tools.ietf.org/html/draft-ietf-tls-tls13-20*.

[Sec15] Unifore Security. *SMART HOME TECHNOLOGIES ZIGBEE VS Z-WAVE, DECT-ULE*. [Retrieved on 13.06.2017]. Mar. 2015. URL: *https://www.unifore.net/home-alarm-system/smart-home-technologies-zigbee-vs-z-wave-dect-ule.html*.

[Sic] Bundesamt für Sicherheit in der Informationstechnik (BSI). *IT-Grundschutz-Kataloge*. [Retrieved on 29.06.2017]. URL: *https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/inhalt_node.html*.

[Sic17] Bundesamt für Sicherheit in der Informationstechnik (BSI). *TLS nach TR-03116-4 - Checkliste für Diensteanbieter*. [Retrieved on 04.05.2017]. Mar. 2017. URL: *https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/TLS-Checkliste.pdf?___blob=publicationFile&v=3*.

[Sok15] Daniel AJ Sokolov. *Deepsec: ZigBee macht Smart Home zum offenen Haus*. [Retrieved on 11.05.2017]. Nov. 2015. URL: *https://www.heise.de/security/meldung/Deepsec-ZigBee-macht-Smart-Home-zum-offenen-Haus-3010287.html*.

[SS00] Jerome H. Saltzer and Michael D. Schroeder. *The Protection of Information in Computer Systems*. [Retrieved on 03.07.2017]. 2000. URL: *http://www.cs.virginia.edu/~evans/cs551/saltzer/*.

[sta16] statista. *Smart Home*. [Retrieved on 16.03.2017]. Oct. 2016. URL: *https://de.statista.com/outlook/279/137/smart-home/deutschland*.

[Tho] Jörg Thoma. *Rechtemissbrauch ermöglicht unsichtbare Tastaturmitschnitte*. [Retrieved on 07.06.2017]. URL: *https://www.golem.de/news/android-apps-rechtemissbrauch-erlaubt-unsichtbare-tastaturmitschnitte-1705-128055.html*.

[Wil11] Sean Wilkins. *OSI and TCP/IP Model Layers*. [Retrieved on 23.03.2017]. Nov. 2011. URL: *http://www.pearsonitcertification.com/articles/article.aspx?p=1804869*.

[Wis16] Stan Wisseman. *Third-party libraries are one of the most insecure parts of an application*. [Retrieved on 09.05.2017]. 2016. URL: *https://techbeacon.com/third-party-libraries-are-one-most-insecure-parts-application*.

[ZWaa] Z-Wave. *Setting scenes: the magic of smart homes*. [Retrieved on 13.04.2017]. URL: *http://blog.z-wave.com/setting-scenes-the-magic-of-smart-homes*.

[ZWab] Z-Wave. *Smart Home and Home Automation: What's The Difference?* [Retrieved on 06.07.2017]. URL: *http://blog.z-wave.com/smart-home-and-home-automation-whats-the-difference*.

Sources referenced for images only

[FMM] Freepik, Madebyoliver, and Vectors Market. *Flaticon.com*. [Retrieved on 04.04.2017]. URL: *http://www.flaticon.com/*.

[Lin14] Cees Links. *Sentrollers, and the World of Small Data*. [Retrieved on 04.04.2017]. June 2014. URL: *http://www.hometoys.com/article/2014/06/sentrollers-and-the-world-of-small-data/2164/*.

Employed master's and bachelor's theses

[Bar16] Jan Bartkowski. "Sicherheitsanalyse eines App-gesteuerten Smart Home Systems". Sept. 2016.

[Kol17] Philipp Kolloge. "Sicherheitsanalyse eines App-gesteuerten Smart Home Systems II". July 2017.

[Mül15] Daniel Müller. "Untersuchung zur Broadcastsicherheit von Android-Apps. Analysis of Android broadcast security". Sept. 2015.

[Sky17] Kevin Skyba. "Sicherheitsanalyse einer Smarthome Android App. Am Beispiel der Qivicon Smarthome Plattform". Unpublished preliminary revision as of 09.06.2017. June 2017.

A.8 Disc

The following disc contains the digital version of this thesis in the PDF format.